UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Gabriel Medeiros das Neves

CheckBERT: Um Sistema de Identificação de Informações Falsas Através da Comparação Semântica de Sentenças com Notícias de Portais Jornalísticos de Língua Inglesa Utilizando Google BERT

Florianópolis
2023

Gabriel Medeiros das Neves

CheckBERT: Um Sistema de Identificação de Informações Falsas Através da Comparação Semântica de Sentenças com Notícias de Portais Jornalísticos de Língua Inglesa Utilizando Google BERT

Trabalho de Conclusão de Curso submetido ao curso de Graduação em Sistemas de Informação da Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.
Orientadora: Prof[a]. Carina Friedrich Dorneles, Dra.

Florianópolis
2023

Gabriel Medeiros das Neves

CheckBERT: Um Sistema de Identificação de Informações Falsas Através da Comparação Semântica de Sentenças com Notícias de Portais Jornalísticos de Língua Inglesa Utilizando Google BERT

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Sistemas de Informaçãoe aprovado em sua forma final pelo curso de Graduação em Sistemas de Informação.

Florianópolis, 11 de Julho de 2023.

Prof. Álvaro Junio Pereira Franco, Dr.
Coordenador do Curso

Banca Examinadora:

Prof[a]. Carina Friedrich Dorneles, Dra.
Orientadora

Prof. Renato Fileto, Dr.
Avaliador

Prof. Mateus Grellert da Silva, Dr.
Avaliador

Este trabalho é dedicado aos meus amados pais, que sempre me incentivaram na busca pelo conhecimento e pela excelência, embora nunca tenham faltado com amor e apoio em todos os momentos da minha vida. Obrigado por estarem sempre presentes.

AGRADECIMENTOS

Gostaria de expressar minha profunda gratidão às pessoas que desempenharam um papel fundamental na realização deste trabalho, seja direta ou indiretamente.

Em primeiro lugar, à minha orientadora Prof[a]. Dra. Carina Friedrich Dorneles. Sua orientação, bom humor e riqueza de conhecimento foram essenciais durante todo o trabalho. Sou muito grato pela oportunidade de trabalhar e aprender com você, foi uma experiência excelente do começo ao fim.

Além disso, também gostaria de agradecer minha familia e amigos. Vocês estiveram comigo durante toda essa jornada, e cada incentivo e apoio dado foi essencial para que eu seguisse em frente. Meu mais sincero agradecimento.

RESUMO

Em uma pesquisa realizada pela Kaspersky em parceria com a CORPA entre dezembro de 2019 e janeiro de 2020, foi estimado que, em média, 62% dos brasileiros não conseguem identificar notícias falsas na internet. Mais recentemente, em julho de 2021, a pesquisa realizada pelos institutos IDEIA e Vero constatou que 79% dos brasileiros entrevistados consideram a disseminação de noticias falsas um problema grave. Diante desse contexto, o presente trabalho propõe a construção de um sistema capaz de identificar informações falsas em sentenças textuais, por meio da comparação das sentenças com o conteúdo de artigos publicados em portais de notícia de língua inglesa. Essa comparação é realizada através da análise de similaridade semântica entre a sentença fornecida e os artigos de notícia previamente coletados pelo sistema, através de um modelo profundo de processamento de linguagem natural chamado Google BERT. Ao final do projeto, espera-se entregar um software capaz de realizar esse tipo de análise de maneira totalmente automatizada, a fim de facilitar o procedimento de identificação de notícias falsas, que ainda requer intervenção manual em todas as suas etapas.

Palavras-chave: Google BERT. Processamento de Linguagem Natural. Web scraping.

ABSTRACT

In a survey conducted between December 2019 and January 2020 by Kaspersky, in partnership with CORPA, it was estimated that, on average, 62% of the Brazilians are not able to identify fake news on the internet. More recently, in July 2021, a survey carried out by IDEIA and Vero institutes had 79% of the interviewed Brazilians affirming that they consider the dissemination of fake news a severe problem. In this regard, the present work proposes the implementation of a system capable of identifying false information in textual sentences by comparing those sentences with the content of published articles on USA news portals. This comparison process is performed through the semantic similarity analysis between the given sentence and previously collected news articles by a deep Natural Language Processing model called Google BERT. At the end of the project, it is expected that the delivered software will be able to execute the described analysis in a fully automated manner, in order to facilitate the procedure for identifying fake news, which still requires manual intervention in all its steps.

Keywords: Google BERT. Natural Language Processing. Web scraping.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
ASGI	*Asynchronous Server Gateway Interface*
BERT	*Bidirectional Encoder Representations from Transformers*
DAG	*Directed Acyclic Graph*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
JSON	*JavaScript Object Notation*
MLOps	*Machine Learning Operations*
NLI	*Natural Language Inference*
NLP	*Natural Language Processing*
POC	*Proof of Concept*
SEO	*Search Engine Optimization*
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

A propagação de informações falsas não é exclusividade do século XXI, ainda que o tema seja extremamente relevante nos tempos atuais. Ao longo da história, diversos episódios de rumores falsos acabaram alterando o rumo das esferas políticas e sociais da época, desde o suicídio do general romano Marco Antônio, ao receber a falsa notícia do falecimento de sua amante Cleópatra, até a divulgação massiva de notícias falsas durante eventos marcantes da última década, como as eleições de 2016 nos Estados Unidos e a saída do Reino Unido da União Europeia (RUBIO HANCOCK, JAIME, 2016). Contudo, conforme o acesso e o compartilhamento de informações foi sendo progressivamente facilitado pelas inovações digitais, a disseminação de notícias falsas também tornou-se mais recorrente, ao ponto que, em 2017, a expressão "fake news", utilizada para referir-se a notícias falsas, foi eleita como expressão do ano pelo dicionário Collins, segundo a matéria publicada pela revista Veja (2017).

Entendendo o crescimento e a notoriedade do tema no meio digital, algumas organizações construíram ferramentas para auxiliar a população a identificar a veracidade dos fatos que lhe são apresentados. Dentre essas ferramentas, podemos citar a FactCheck.org[1], o Washington Post Fact Checker[2] e o Politifact[3]. Além disso, no Brasil, podemos verificar a autenticidade de fatos por portais como Fato ou Fake[4], Comprova[5], Lupa[6] e outros. Entretanto, o funcionamento de todas essas ferramentas depende da investigação por parte de uma equipe de jornalistas, que então publicam seu parecer sobre a exatidão dos fatos e apresentam as fontes e referências utilizadas. Todavia, esse processo frequentemente não consegue acompanhar a velocidade da comunicação na internet e, juntamente com interfaces nem sempre amigáveis ao usuário final e a impraticidade de verificar a precisão dos fatos em diferentes portais, acaba resultando na incerteza, ou ainda na aceitação, de informações incorretas ou falsas.

À vista disso, o presente trabalho visa facilitar o reconhecimento de informações falsas através da comparação com notícias publicadas nos principais portais de notícias de língua inglesa, bem como em algumas das ferramentas identificadas acima. Esse processo de identificação ocorre sem intervenções manuais, fazendo uso de um modelo contextualizado de *Natural Language Processing* (NLP) chamado *Bidirectional Encoder Representations from Transformers* (BERT), que utiliza *Deep Learning* para compreender o significado das palavras em um texto da mesma maneira que humanos o fazem.

Ao final do projeto, espera-se que o software desenvolvido seja capaz de assimilar a sentença fornecida pelo usuário, identificar artigos relacionados e retornar a classificação

[1] https://www.factcheck.org/
[2] https://www.washingtonpost.com/news/fact-checker/
[3] https://www.politifact.com/
[4] https://g1.globo.com/fato-ou-fake/
[5] https://projetocomprova.com.br/
[6] https://lupa.uol.com.br/

obtida pelo modelo BERT, juntamente com o *Uniform Resource Locator* (URL) do artigo utilizado para classificação e outras informações relevantes, como sua data de publicação e título, por exemplo.

1.1 OBJETIVOS GERAIS

No que diz respeito ao objetivo geral do trabalho, é proposta a criação de um software capaz de identificar se uma sentença fornecida como entrada pelo usuário final da ferramenta é falsa. A identificação se dá através da comparação semântica do conteúdo textual de artigos de portais de notícias de língua inglesa. Para isso, é utilizado um modelo de NLP chamado Google BERT, com a finalidade de identificar contexto e significado nos textos jornalísticos, e assim possibilitar a classificação do relacionamento entre a sentença fornecida pelo usuário e o conteúdo dos artigos, o qual pode ser classificado como concordante, discordante ou neutro.

1.2 OBJETIVOS ESPECÍFICOS

Buscando a realização do objetivo geral destacado acima, foram identificados os seguintes objetivos específicos:

- Desenvolvimento de *pipelines* de dados utilizando técnicas de *web scraping* para extração das informações necessárias em portais jornalísticos de língua inglesa;
- Implementação de um módulo que utilize o Google BERT para tarefas de *Natural Language Inference* (NLI), com o intuito de identificar o relacionamento entre a sentença fornecida pelo usuário e artigos relacionados;
- Implementação de um sistema web capaz de fornecer todas as funcionalidades do modelo por meio de uma interface gráfica de simples utilização;

1.3 ORGANIZAÇÃO DO RELATÓRIO

Além do capítulo de Introdução, este trabalho contém quatro outros capítulos: o Capítulo 2, que propõe-se a abordar conceitos essenciais para a compreensão do projeto; o Capítulo 3, que visa apresentar trabalhos relacionados ao tema; o Capítulo 4, que detalha a toda a concepção e implementação do CheckBERT; e o Capítulo 5, que apresenta as considerações finais do projeto e sugestões de trabalhos futuros.

2 CONCEITOS BÁSICOS

Neste capítulo, são tratados os conceitos fundamentais para a compreensão da trabalho.

2.1 FAKE NEWS

Esta seção conceitua *Fake news* e apresenta as principais técnicas para sua identificação.

2.1.1 Conceito

Fake news é um termo em inglês que se refere, de acordo com Allcott e Gentzkow (2017), a artigos jornalísticos que são intencionalmente e comprovadamente falsos, podendo enganar leitores. Esses artigos geralmente possuem alegações completamente irreais ou ligadas a teorias da conspiração, mas também podem ser construídos a base de fatos, contendo apenas pequenas manipulações de narrativa, que afetam o julgamento do leitor sobre a notícia. Dentre as diversas motivações para a produção e propagação de *fake news*, o artigo de Lazer *et al.* (2018) destaca que as notícias produzidas frequentemente tem como objetivo influenciar opiniões políticas, manipular escolhas pessoais, ou simplesmente causar confusão e desordem na sociedade.

2.1.2 Técnicas de identificação

Identificar *fake news* é um desafio crítico na era da informação digital, especialmente devido ao seu alto impacto na sociedade e a capacidade de gerar e compartilhar conteúdo rapidamente na internet, que vem crescendo gradualmente. Tendo isso em vista, muitos esforços foram conduzidos na última decada para facilitar o processo de identificação. Dentre eles, destacam-se os seguintes métodos:

- **Verificação de fatos:** a verificação de fatos é, atualmente, uma das principais técnicas de identificação de *fake news*. Esta técnica consiste na comparação de informações de uma notícia com fontes confiáveis e verificadas, geralmente através de um processo investigativo de caráter jornalístico. Embora esse tipo de identificação seja muito assertivo, ele ainda apresenta muitos processos manuais, o que dificulta o combate da disseminação acelerada na internet.

- **Análise de fontes:** a credibilidade da fonte de uma notícia é um indicador significativo da sua veracidade, uma vez que fontes confiáveis geralmente têm uma reputação estabelecida de precisão e confiabilidade. Dentre as características que são levadas em consideração para avaliar a credibilidade de uma fonte, ressaltam-se a reputação

da fonte, a qualidade de seu site, a presença de viés e a transparência do processo de reportagem.

- **Análise de sentimento:** a análise de sentimento envolve a avaliação do tom emocional do título ou texto de uma notícia. Essa avaliação é realizada pois sabe-se que uma das técnicas comuns na criação de *fake news* é a utilização de escritas emocionalmente carregadas, projetadas para atrair a atenção e manipular as reações dos leitores.

- **Análise de rede:** *fake news* muitas vezes se espalham de maneiras diferentes de notícias verdadeiras, particularmente nas redes sociais. Por conta disso, a análise de como uma notícia se propaga através da rede (usuários envolvidos, velocidade de propagação, impulsionamento por anúncios, etc.) pode ajudar a identificar sua veracidade.

2.2 WEB SCRAPING

Nesta seção são abordados os entendimentos fundamentais sobre *Web Scraping* para a leitura desse trabalho.

2.2.1 Conceito

Web Scraping refere-se ao processo de extração de dados de páginas da web através da análise das respostas de protocolo *Hypertext Transfer Protocol* (HTTP) dos sites, conforme explicado por Mitchell (2018). Esse processo permite a coleta de informações específicas de maneira automatizada, as quais podem ser utilizadas para tarefas como análise de mercado, monitoramento de redes sociais, desenvolvimento de estratégias de *Search Engine Optimization* (SEO), aprimoramento de modelos de aprendizado de máquina, entre outros.

2.2.2 Técnicas de implementação

A seguir estão listadas as principais técnicas de *web scraping*, de acordo com Pickell (2023):

1. **Cópia e colagem manual:** essa técnica envolve a cópia de trechos específicos de páginas da web de maneira manual, seguida pela colagem dos trechos copiados em algum arquivo de texto ou planilha.

2. **Extração automatizada de respostas HTTP:** aqui são utilizadas linguagens de programação, como Python[1] ou Ruby[2], para executar requisições de tipo HTTP e

[1] https://www.python.org/
[2] https://www.ruby-lang.org/en/

extrair as informações desejadas das respostas obtidas.

3. **Extração automatizada incorporando navegadores:** essa técnica utiliza linguagens de programação para executar e utilizar navegadores web de maneira automatizada. É especialmente útil para lidar com páginas da web dinâmicas e interativas, que possuem informações geradas pela execução de *scripts* por parte do navegador.

4. **Reconhecimento de anotação semântica:** essa técnica faz uso das marcações semânticas ou metadados (que estão presentes nas páginas da web) para localizar e extrair seções desejadas de informação.

5. **Análise auxiliada por visão computacional:** através de aprendizado de máquina e visão computacional, essa técnica é capaz de extrair não apenas as informações textuais das páginas desejadas, como também ilustrações visuais diversas.

2.3 GOOGLE BERT

Esta seção tem como objetivo apresentar brevemente o Google BERT e ilustrar suas características primordiais para o entendimento do trabalho.

2.3.1 Conceito

O Google BERT é um modelo contextualizado de NLP constituído de uma rede neural profunda baseada na arquitetura *Transformer*, que, segundo Devlin *et al.* (2018), diferencia-se por relacionar as palavras de uma sentença entre si, não apenas seguindo a ordem linear de leitura. Por conta dessa abordagem, o BERT oferece uma solução mais robusta para o entendimento contextual das palavras em um texto, e supera limitações inerentes aos métodos anteriores de modelagem de linguagem natural, como o Word2Vec[3] e o GloVe[4]. Por consequência, o BERT vem ganhando espaço em diferentes áreas do estudo de NLP e, segundo a Google em seu evento virtual *Search*, já é utilizado em praticamente todas as consultas realizadas em inglês em sua plataforma de pesquisa, conforme publicado por Schwartz (2020).

Além disso, por ser um modelo contextualizado, o BERT já possui diversas representações de palavras e suas relações, possibilitando sua utilização direta em diversas tarefas de diferentes escopos, ainda que, majoritariamente, o modelo apresente resultados melhores após ser ajustado para contextos específicos.

2.3.2 Codificações Bi-Encoder e Cross-Encoder

A partir de Agosto de 2019, com a publicação do artigo *"Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks"* (REIMERS; GUREVYCH, 2019), um novo

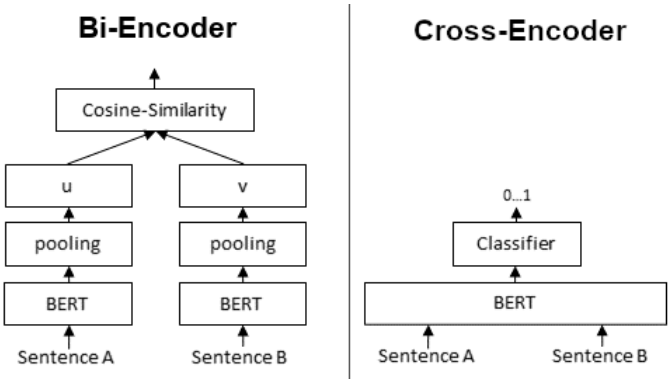
[3] https://code.google.com/archive/p/word2vec/
[4] https://nlp.stanford.edu/projects/glove/

paradigma de codificação foi disponbilizado para tarefas de similaridade semântica entre sentenças: a codificação *Bi-Encoder*. Esse novo tipo de codificação constrói representações vetoriais de sentenças chamadas de *embeddings*, que podem ser reutilizadas em diferentes análises e comparadas através de funções de similaridade, como a função euclidiana ou a função de similaridade por cosseno, por exemplo.

Por meio do uso dessa nova abordagem, foi possível acelerar significativamente tarefas de busca de similaridade semântica. Conforme indicado no artigo de Reimers e Gurevych (2019), a tarefa de buscar as duas sentenças mais semelhantes em um conjunto de 10000 sentenças, que necessitaria de cerca de 65 horas para ser finalizada, agora pode ser finalizada em cinco segundos. Este avanço se deve à velocidade da comparação dos *embeddings* gerados pela codificação *Bi-encoder*, que supera significativamente a velocidade da comparação de sentenças realizada pela codificação *Cross-Encoder*.

Contudo, é importante ressaltar que, apesar de sua eficiência, a codificação *Bi-Encoder* tende a apresentar resultados inferiores em relação à codificação *Cross-Encoder*. A razão por trás disso é a diferença fundamental no processo de codificação entre essas duas abordagens. Como ilustrado na Figura 1, a codificação *Bi-Encoder* constrói *embeddings* de sentenças individuais, enquanto a codificação *Cross-Encoder* compara duas sentenças diferentes ao mesmo tempo. Esta abordagem permite um entendimento maior do contexto e, por conseguinte, melhores resultados nas classificações de similaridade.

Figura 1 – Diferenças *Bi-Encoder* e *Cross-Encoder*



Fonte: SBERT.net, 2022.

No entanto, a codificação *Cross-Encoder* apresenta uma limitação significativa: uma vez que a análise é realizada entre duas sentenças, a transformação efetuada pelo modelo não pode ser reutilizada. Isso significa que, ao contrário da codificação *Bi-Encoder*, a codificação *Cross-Encoder* não gera *embeddings* reutilizáveis. Esse é um fator a ser considerado ao escolher entre essas duas abordagens de codificação, já que a capacidade de reutilizar *embeddings* pode ser vantajosa em muitos cenários de análise de texto.

Em suma, as codificações *Bi-Encoder* e *Cross-Encoder* oferecem abordagens distin-

tas para a comparação semântica de sentenças. A escolha entre elas depende da tarefa em questão, bem como de considerações práticas como eficiência computacional e a necessidade de reutilização de *embeddings*.

2.4 ELASTICSEARCH

Esta seção visa conceituar o ElasticSearch e apresentar suas principais características.

2.4.1 Conceito

O ElasticSearch é uma plataforma de código aberto para pesquisa e análise de dados de maneira eficiente, podendo também ser utilizada para armazenamento (como banco de dados). Sua implementação foi realizada em Java e é baseada na biblioteca Apache Lucene, que fornece diversos recursos e mecanismos robustos para buscas textuais. Além disso, o ElasticSearch possui uma arquitetura distribuída e trabalha com diversos tipos de dados, os quais são organizados em formas de documentos (uma representação de dados complexos em formato JSON), possibilitando seu uso de maneira escalável em diferentes tipos de aplicações.

2.4.2 Principais Características

No que diz respeito às principais características do ElasticSearch, destacam-se as seguintes particularidades:

1. **Alta Disponibilidade:** o ElasticSearch foi construído seguindo a arquitetura de cluster, que consiste em um ou mais nós (servidores) operando simultaneamente. Assim sendo, mesmo que um dos nós falhe, o cluster pode continuar a funcionar, garantindo alta disponibilidade.

2. **Escalabilidade Horizontal:** À medida que o volume de dados aumenta, novos nós podem ser adicionados a arquitetura para garantir eficiência e desempenho.

3. **Buscas Complexas:** Além de suportar uma ampla gama de tipos de consultas (como consultas de texto completo e consultas estruturadas, por exemplo), o ElasticSearch apresenta alta performance na execução de buscas complexas em grandes volumes de dados.

4. **Mapeamento de *Schema*:** É possível definir *schemas* de dados para cada um dos indíces construídos, garantindo a estrutura e tipagem de cada um dos documentos armazenados.

3 TRABALHOS RELACIONADOS

Neste capítulo, são introduzidos trabalhos relacionados à temática central deste projeto: a utilização de modelos BERT em contextos de análise de similaridade semântica. Ao longo do segmento, cada um dos trabalhos é brevemente descrito e, por fim, comparado ao trabalho proposto nesta monografia.

3.1 FAKEBERT

O artigo de Kaliyar, Goswami e Narang (2021) propõe a utilização de uma abordagem de aprendizado profundo baseada em arquiteturas BERT denominada FakeBERT, que visa identificar informações falsas em trechos de publicações em redes sociais. Essa abordagem mescla a utilização de redes neurais convolucionais profundas e a arquitetura bidirecional do BERT, com o intuito de minimizar problemas de ambiguidade que, segundo os autores, são o maior desafio durante o entendimento de modelos de processamento de linguagem natural. Quanto ao treinamento e testagem do modelo, foram realizados experimentos em um conjunto de dados de notícias publicadas durante o período eleitoral de 2016 nos Estados Unidos, contando tanto com notícias verdadeiras quanto falsas. A testagem comparou o FakeBERT com outros modelos que utilizam redes neurais convolucionais e redes neurais de memória de longo e curto prazo, e os resultados apresentaram uma vantagem notória para o FakeBERT, que alcançou uma acurácia de 98%.

3.2 COVID-TWITTER-BERT

Tendo em vista a recente pandemia do coronavírus e seus desdobramentos, bem como a relevância do Twitter na disseminação de notícias e no entendimento da percepção da sociedade sobre determinado tema, o artigo *"COVID-Twitter-BERT: A Natural Language Processing Model to Analyse COVID-19 Content on Twitter"*, de Müller, Salathé e Kummervold (2020), visa o desenvolvimento de um modelo de processamento de linguagem natural a partir do enriquecimento do modelo existente BERT-Large, que utiliza a arquitetura BERT e foi pré-treinado com dados da Wikipédia. Para isso, foram extraídos mais de 160 milhões de tweets referentes ao coronavírus, dos quais, após tratamento e filtragem, foram selecionados cerca de 22,5 milhões para o treinamento e testagem do modelo. Ao final, o modelo desenvolvido alcançou seu objetivo em sua principal atribuição - auxiliar tarefas de classificação semântica dentro do contexto do COVID-19, como a identificação de sentimento em uma sentença ou a classificação do conteúdo textual como pessoal ou jornalístico - e foi capaz de obter resultados superiores quando comparado ou modelo pré-treinado BERT-Large, que foi utilizado como base para o estudo.

3.3 QUADRO COMPARATIVO

Esta seção apresenta apresenta uma análise comparativa entre os sistemas propostos nos trabalhos relacionados e o sistema apresentado por este trabalho, conforme demonstrado no Quadro 1.

A análise demonstra que os trabalhos relacionados, embora utilizem o Google BERT para atacar o mesmo problema, apresentam estratégias diferentes para combatê-lo. Isso porque tanto o FakeBERT quanto o COVID-Twitter-BERT utilizam conjuntos de dados derivados de postagens de redes sociais para a concepção de um modelo de categorização de textos. Por outro lado, o CheckBERT executa a extração de artigos de portais de notícias e os utiliza para o treinamento de um modelo de categorização do relacionamento entre dois textos: a sentença fornecida pelo usuário final e o texto do artigo mais relevante. Adicionalmente, o CheckBERT também oferece os resultados da classificação e informações sobre o artigo em uma interface web.

Dessa maneira, entende-se que ambos os trabalhos relacionados apresentam contribuições consideráveis para o combate à desinformação na internet, evidenciando resultados notáveis em seus respectivos contextos. Analogamente, o CheckBERT fornece uma solução final web destinada a simplificar o processo de verificação de informações inverídicas, contribuindo para o mesmo propósito, mas sob uma perspectiva distinta.

Quadro 1 – Comparação entre os sistemas propostos nos trabalhos relacionados e o CheckBERT.

Trabalho	**Fonte de dados**	**Contexto dos dados**	**Estratégia de extração de dados**	**Taxa de atualização de dados**	**Modelos utilizados**	**Resultado final do trabalho**
FakeBERT	Redes sociais.	Eleição presidencial norte-americana de 2016.	Não possui.	Não atualiza a base de dados inicial.	BERT + CNN.	Modelo otimizado para detecção de notícias falsas através da análise semântica do conteúdo textual de uma publicação.
COVID-Twitter-BERT	Redes sociais.	COVID.	Não possui.	Não atualiza a base de dados inicial.	BERT.	Modelo otimizado para detecção de notícias falsas através da análise semântica do conteúdo textual de um tweet.
CheckBERT	Portais jornalísticos.	Conteúdo jornalístico diverso.	Coleta dados da web utilizando técnicas de *Web Scraping*.	Configurável (a cada 15 minutos por padrão).	BERT.	Software web capaz de receber uma sentença, identificar um artigo relacionado na base de dados, comparar relacionamento semântico entre a sentença e o texto do artigo, e por fim retornar para o usuário dados do artigo e da classificação realizada pelo modelo BERT.

Fonte: Elaboração própria.

4 CHECKBERT

Este capítulo visa descrever a concepção e implementação do sistema proposto neste trabalho. Para isso, o capítulo foi dividido em três seções principais: a Seção 4.1, que apresenta conceitos gerais da ferramenta, como o fluxo de atividades e a arquitetura do sistema; a Seção 4.2, que detalha a implementação de cada um dos componentes do sistema; e a Seção 4.3, que expõe os experimentos realizados com o Google BERT no contexto de alcançar melhores resultados dentro do escopo do projeto.

4.1 VISÃO GERAL

Nesta seção, são introduzidas duas subseções com o intuito de descrever os aspectos centrais que compõem a concepção do CheckBERT: a subseção de Fluxo de atividades, que tem como propósito esclarecer o funcionamento sequencial do CheckBERT; e a subseção de Arquitetura, que objetiva expor, sem maior detalhamento, cada um dos módulos desenvolvidos e as interações existentes entre eles, e assim proporcionar uma visão mais precisa da estrutura do CheckBERT.

4.1.1 Fluxo de atividades

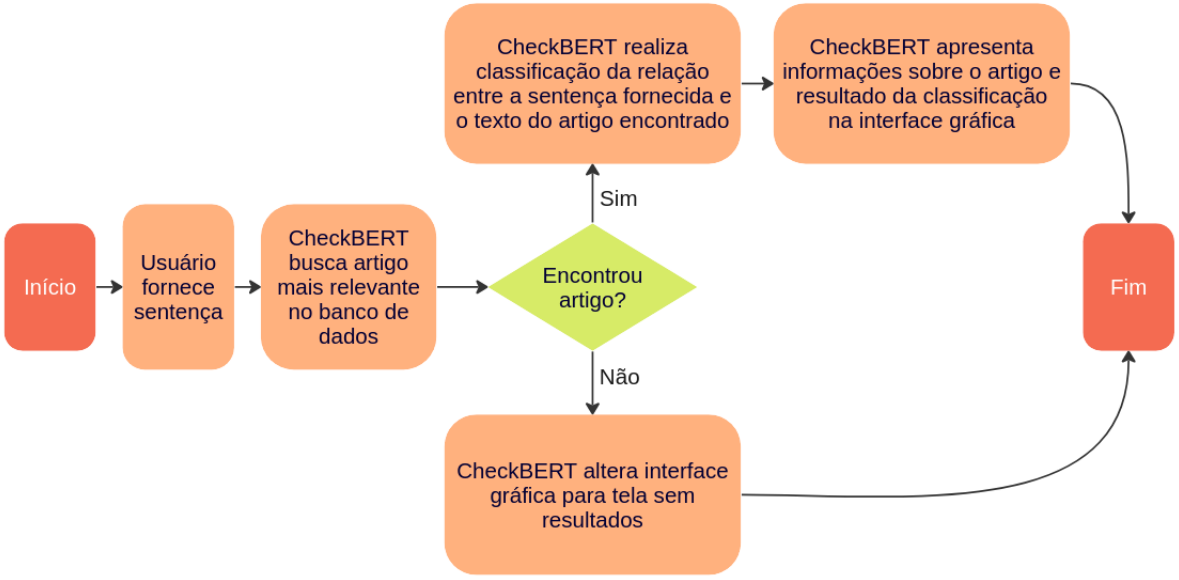
Conforme descrito na Figura 2, o fluxo de atividades durante o uso do CheckBERT é iniciado a partir do fornecimento de uma sentença por parte do usuário, de maneira manual. Essa etapa constitui a única interação exigida do usuário final ao longo de todo o processo de verificação. Logo após, o CheckBERT utiliza a sentença fornecida para efetuar a procura do artigo mais relevante no banco de dados, e então realiza a classificação do relacionamento entre o texto do artigo mais relevante e a sentença fornecida, que pode ser categorizado em um dos seguintes rótulos: concordante, discordante ou neutro. Após realizar a classificação, o CheckBERT atualiza a interface gráfica e apresenta ao usuário os dados do artigo e da classificação.

Os dados apresentados variam de acordo com a proveniência dos artigos, uma vez que o CheckBERT aceita artigos de dois gêneros de portais distintos. O primeiro gênero corresponde aos portais de notícia, que divulgam exclusivamente os fatos relacionados a um evento específico. Já o segundo gênero corresponde aos portais verificadores, que averiguam a veridicidade de uma alegação particular e expõem os fatos que conduziram à classificação efetuada.

Qualquer que seja a proveniência do artigo, é esperado que o portal disponibilize ao menos as seguintes informações:

1. Título do artigo;
2. Data de publicação do artigo;

Figura 2 – Fluxo de atividades durante o uso do CheckBERT.



Fonte: Elaboração própria.

3. URL para leitura gratuita do artigo;

Adicionalmente, artigos de sites verificadores também devem possuir as seguintes informações:

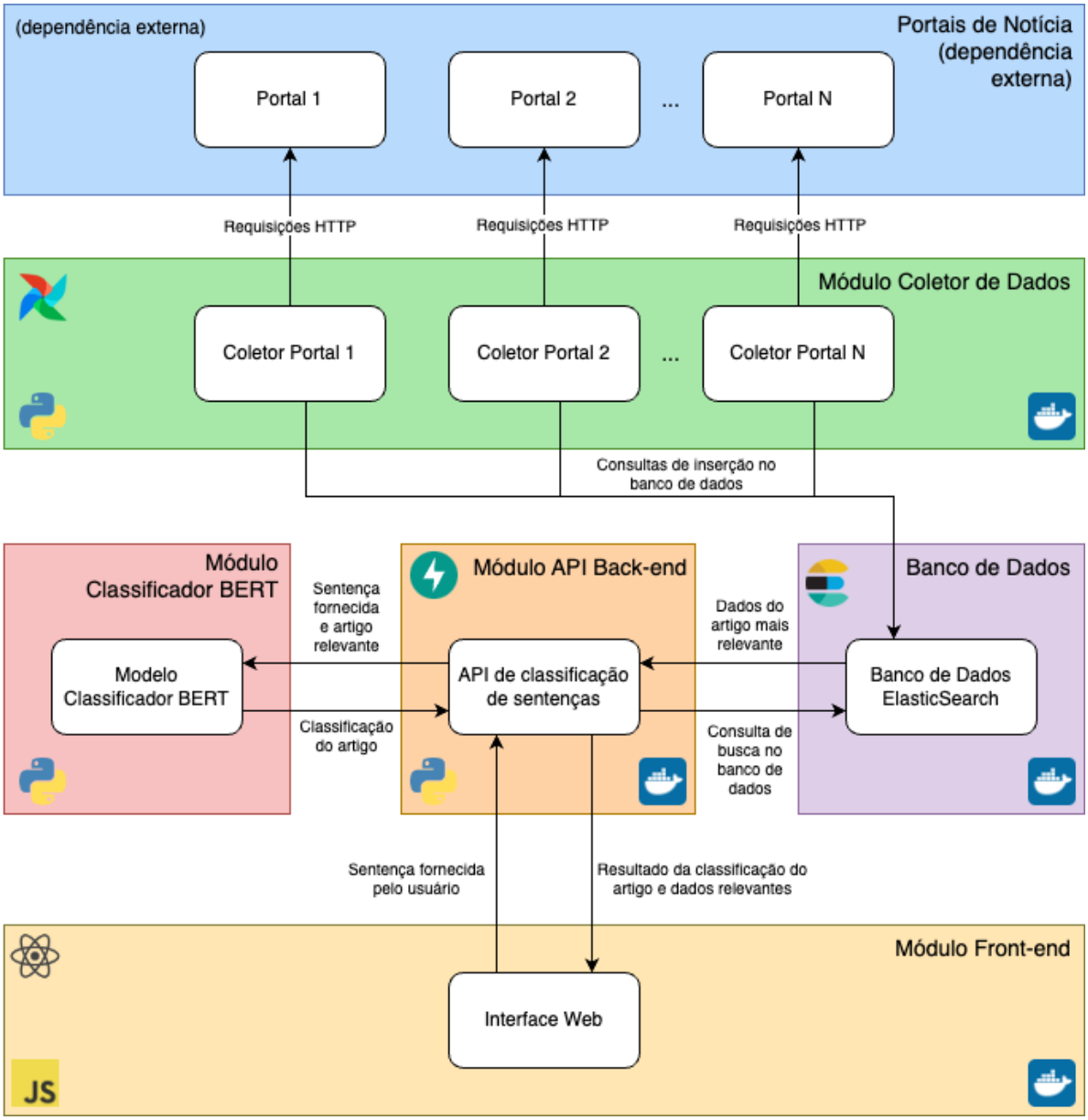
1. Classificação da alegação atribuída pelo portal de notícia (varia de acordo com o portal);
2. Autor da alegação (pode corresponder ao veículo utilizado para publicação, como Facebook ou Twitter, caso o portal decida omitir o autor da publicação);

Por fim, faz-se necessário evidenciar que os artigos de sites verificadores são utilizados para o treinamento do modelo de classificação BERT, mas esse processo não é realizado durante o uso da ferramenta, e está apresentado em mais detalhes nas seções seguintes.

4.1.2 Arquitetura

Como demonstrado na Figura 3, o CheckBERT é dividido em quatro módulos principais e um banco de dados, além dos portais de notícia com os quais o Módulo Coletor de Dados interage, e que representam dependências externas do projeto. Todos os módulos comunicam-se com ao menos um outro módulo do sistema, mas apenas a módulo Interface Gráfica possui interação com o usuário.

Figura 3 – Arquitetura do CheckBERT.



Fonte: Elaboração própria.

Naturalmente, cada módulo detém funções e finalidades únicas, alinhadas às suas respectivas necessidades para a execução do CheckBERT. Em virtude disso, todos os módulos, juntamente com o banco de dados, foram brevemente descritos abaixo. São eles:

- **Módulo Coletor de Dados:** Utiliza técnicas de *web scraping* para extrair o conteúdo textual dos artigos dos portais jornalísticos selecionados, os quais possuem fundamental importância tanto o para o treinamento do Classificador BERT quanto para a experiência do usuário ao navegar pela ferramenta.

- **Banco de Dados:** Responsável por armazenar todos os dados de artigos coletados

pelo Coletor de Dados.

- **Módulo Classificador BERT:** Contém todos os scripts referentes ao treinamento do modelo BERT para tarefas de NLI, que, de acordo com Souza (2020), consistem na validação de que determinado trecho de texto em língua natural, chamado de premissa, acarreta em outro texto, chamado hipótese. No contexto desse trabalho, o modelo é utilizado para validar se algum trecho de texto do artigo mais relevante acarreta (entende-se como concorda, discorda ou não possui relação) na sentença fornecida pelo usuário.

- **Módulo API Back-end:** Oferece ao Front-end uma interface de acesso às funcionalidades do Classificador BERT e do Banco de Dados, intermediando a comunicação e retornando os dados necessários em formato *JavaScript Object Notation* (JSON) durante a utilização da ferramenta.

- **Módulo Interface Gráfica:** Representa o único meio de comunicação entre o usuário e o CheckBERT, sendo tanto o ponto de entrada quanto de saída dos dados durante o fluxo de atividades do CheckBERT.

4.2 IMPLEMENTAÇÃO

Esta seção visa elencar as principais tecnologias utilizadas durante o desenvolvimento do CheckBERT e detalhar a implementação de cada um dos módulos mencionados na seção anterior.

4.2.1 Tecnologias utilizadas

Para o desenvolvimento do trabalho, foram utilizadas diversas tecnologias desenvolvidas por terceiros, dentre elas, as mais relevantes são:

- **Python:** Linguagem de programação de código aberto selecionada para o desenvolvimento de todos os módulos do CheckBERT, com exceção do módulo de Interface Gráfica. Foi utilizada em sua versão 3.10[1] da implementação CPython, sendo a ferramenta Poetry 1.2.2[2] selecionada para a gestão de dependências e as plataformas Jupyter Notebook[3] e PyCharm Community Edition[4] selecionadas como ambientes de desenvolvimento.

- **Apache Airflow:** Ferramenta de código aberto utilizada para programar, orquestrar e monitorar fluxos de trabalho complexos. Permite a definição de pipelines de dados

[1] https://www.python.org/downloads/release/python-3100/
[2] https://python-poetry.org/blog/announcing-poetry-1.2.2/
[3] https://jupyter.org/
[4] https://www.jetbrains.com/pycharm/

por meio de código, facilitando a execução dinâmica e robusta de tarefas voltadas ao processamento de dados. Foi utilizada em sua versão 2.5.1[5] como componente fundamental do módulo Coletor de Dados.

- **FastAPI:** Framework web de código aberto voltado para construção de *Application Programming Interface* (API)s em Python, que, quando comparado a outros frameworks similares da linguagem, destaca-se por seu alto desempenho, fácil utilização e ampla flexibilidade. Por conta disso, foi selecionado como tecnologia principal para a implementação do módulo API Back-end em sua versão 0.95.0[6].

- **Sentence Transformers:** Biblioteca Python de código aberto baseada nos *frameworks Pytorch*[7] e *Transformers*[8] para geração de *embeddings* (representações vetoriais de palavras, textos ou imagens) e/ou integração com modelos de arquitetura Transformer com maior nível de abstração. Foi utilizada no módulo Classificador BERT em sua versão 2.2.2[9] para carregamento, treinamento e utilização dos modelos BERT.

- **ElasticSearch:** Banco de dados orientado a documentos de código aberto, capaz de lidar com diversos tipos de dados, incluindo textuais, numéricos, geoespaciais, estruturados e não estruturados. É conhecido pela sua escalabilidade e capacidade de executar pesquisas complexas em alta velocidade, especialmente para buscas em campos textuais. Foi utilizado em sua versão 7.17.9[10] como componente fundamental do módulo Banco de Dados, sendo utilizado para armazenamento e consulta dos artigos coletados.

- **Javascript:** Linguagem de programação interpretada, orientada a objetos e de primeira classe, utilizada predominantemente em interações do lado do cliente em navegadores web. Foi empregada em conjunto com a extensão de sintaxe JSX, que permite a mescla de HTML com Javascript em projetos React[11], na implementação do módulo Interface de Usuário.

- **React:** Biblioteca JavaScript de código aberto voltada para a construção de interfaces gráficas, especialmente para aplicações de página única. Foi utilizada em sua versão 18.2.0 no desenvolvimento do módulo Interface Gráfica.

[5] https://airflow.apache.org/docs/apache-airflow/2.5.1/
[6] https://pypi.org/project/fastapi/0.95.0/
[7] https://pytorch.org/
[8] https://huggingface.co/docs/transformers/index
[9] https://pypi.org/project/sentence-transformers/2.2.2/
[10] https://www.elastic.co/guide/en/elasticsearch/reference/7.17/release-notes-7.17.9.html
[11] https://react.dev/

4.2.2 Módulo Coletor de Dados

Conforme apresentado na seção anterior, o módulo Coletor de Dados foi concebido com o objetivo de extrair dados textuais de diversas fontes de notícias, utilizando para isso técnicas de *web scraping*. Para isso, o módulo possui como premissas fundamentais a integridade dos dados coletados e a verificação frequente de novos artigos nas fontes de notícias, idealmente múltiplas vezes ao dia.

Devido ao volume substancial de dados a serem coletados e a possibilidade de incorporar múltiplas fontes de notícias, tornou-se evidente que a arquitetura desejada deveria permitir o monitoramento individual de cada notícia extraída, além de prover um registro detalhado e métricas de desempenho do CheckBERT. Com base nesses requisitos, optou-se pela utilização do Apache Airflow como base para a implementação e monitoramento das *pipelines* de extração de dados.

Dadas as restrições de tempo para a conclusão do projeto e o seu escopo, decidiu-se que, inicialmente, apenas uma fonte de dados seria explorada e serviria como *Proof of Concept* (POC), que consiste em uma demonstração da viabilidade do projeto. Entretanto, vale ressaltar que a arquitetura escolhida permite a incorporação de múltiplas *pipelines*, à medida que novas fontes de notícias forem sendo integradas ao CheckBERT. Tendo isso em vista, a fonte de dados selecionada para essa POC foi a Politifact, uma plataforma reconhecida por classificar a veracidade de alegações de natureza política do cenário norte-americano, de maneira imparcial. A escolha se deu primariamente pela adequação dos dados da plataforma às necessidades do projeto, uma vez que a Politifact fornece as alegações proferidas, suas fontes (quem proferiu-as ou onde foram publicadas), suas classificações de veracidade e os fatos avaliados para chegar em tais classificações. Além de possuir uma base de verificações de milhares de artigos ao longo de diversos anos.

A partir da definição, foi realizado um estudo do portal para entender precisamente como um usuário navega o site e como se dá o processamento de informações durante a navegação, uma vez que, em concordância com Nemeslaki e Pocsarovszky (2011), esse conhecimento é essencial antes de qualquer etapa de implementação de *Web Scrapers*.

Logo após o entendimento do site, procedeu-se no Airflow com a implementação de diversos scripts Python ligados sequencialmente na estrutura de um *Directed Acyclic Graph* (DAG), que constitui um grafo de arestas dirigidas no qual não existem laços de repetição (FIORE; DEVESAS CAMPOS, 2013). Este DAG, ilustrado na Figura 4, define o comportamento da extração de dados em 5 tarefas. São elas:

- **get_rulings:** retorna uma lista de *strings* contendo as possíveis classificações de sentenças realizadas pelo Politifact. São elas: *true* (verdade), *mostly-true* (verdade em sua maioria), *half-true* (meia-verdade), *mostly-false* (mentira em sua maioria), *false* (mentira) e *pants-fire* (mentira e possui alegação considerada ridícula pelo portal).

- **get_newer_articles_urls:** para cada uma das possíveis classificações retornadas por *get_rulings*, verifica nas variáveis do Airflow qual foi a última página visitada (valor inteiro) e se as notícias com a classificação recebida já foram completamente extraídas (valor booleano), definindo valores iniciais caso as variáveis não existam. Logo após, envia requisições com protocolo HTTP de método GET para as próximas cinco páginas a partir da última página visitada, com o intuito de extrair os URLs dos artigos presentes no conteúdo *Hypertext Markup Language* (HTML) das respostas. É importante evidenciar, contudo, que caso as notícias com a classificação recebida já tenham sido completamente extraídas, as requisições HTTP são enviadas para as primeiras cinco páginas, de maneira a possivelmente coletar artigos mais recentes. Ao final da execução, o *script* atualiza as variáveis do Airflow com os novos valores.
- **flat_article_urls_list:** recebe os URLs coletados por *get_newer_articles_urls* em um *array* bi-dimensional (uma vez que cada classificação possui sua lista de URLs) e realiza a conversão para um *array* unidimensional.
- **filter_already_extracted_urls:** verifica no banco de dados se algum dos URLs recebidos de *flat_article_urls_list* já foi extraído, e retorna apenas os URLs novos.
- **scrap_article:** envia uma requisição HTTP de método GET para cada uma dos URLs recebidas de *filter_already_extracted_urls* e extrai as informações desejadas do conteúdo HTML das respostas, sendo elas: sentença averiguada, fonte da sentença (autor ou local de publicação), data de publicação/proferimento da sentença, classificação dada pelo Politifact e URL da verificação no site do Politifact.

Figura 4 – DAG PolitiFact.

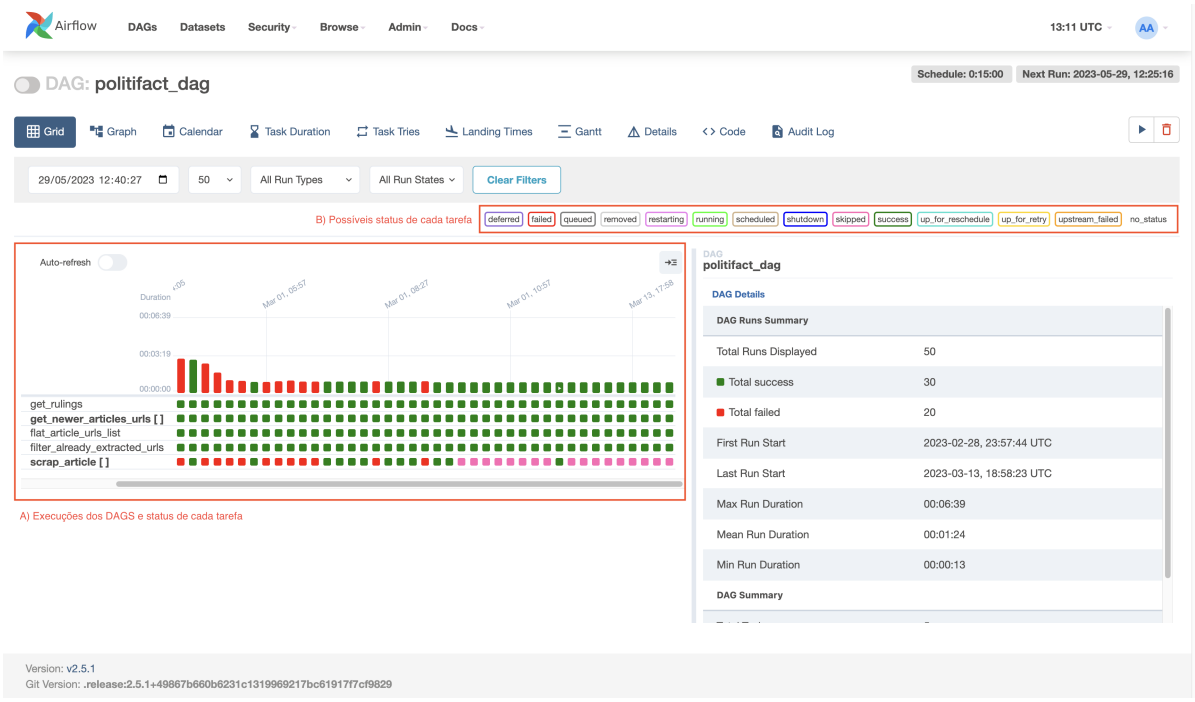
Fonte: Visualização em grafo de uma execução do DAG PolitiFact na interface gráfica do Airflow.

Ademais, no que se refere ao monitoramento da *pipeline*, o Airflow oferece recursos robustos para acompanhar de perto o desempenho das tarefas e o fluxo dos dados. Dentre esses recursos, destaca-se o painel *Grid*, disponível para cada DAG implementado. Este painel, como demonstrado na Figura 5, oferece uma visão clara do status de cada tarefa, permitindo um rastreamento intuitivo do progresso e rápida identificação de eventuais problemas, além de possibilitar o acesso aos logs de cada tarefa, que veio a ser útil para entender processos que falharam.

Com o intuito de facilitar o entendimento do painel, foram destacadas duas seções importantes na Figura 5: a seção A, onde é possível verificar a execução de cada DAG e o

status de suas tarefas (representadas pelos quadrados coloridos); e a seção B, que descreve os possíveis status de cada tarefa, aqui representados por cores diferentes. Nesse sentido, ainda na Figura 5, é possível observar que algumas instâncias da tarefa *scrap_article* não foram bem sucedidas (quadrados de cor vermelha). Estas falhas correspondem a artigos que, embora ainda listados nas páginas de acesso do site Politifact, já não se encontram disponíveis. Todavia, faz-se necessário ressaltar que tais ocorrências não impediram a execução das demais instâncias de *scrap_article*, ou, em outras palavras, a falha na extração de um artigo não impediu a extração dos demais artigos. As tarefas coloridas em rosa, por outro lado, representam execuções que foram intencionalmente puladas, uma vez que não havia artigos novos para serem coletados por *scrap_article*.

Figura 5 – Painel de monitoramento Grid da DAG PolitiFact no Airflow.



Fonte: Visualização em grade das execuções do DAG PolitiFact na interface gráfica do Airflow (com seções destacadas pelo autor).

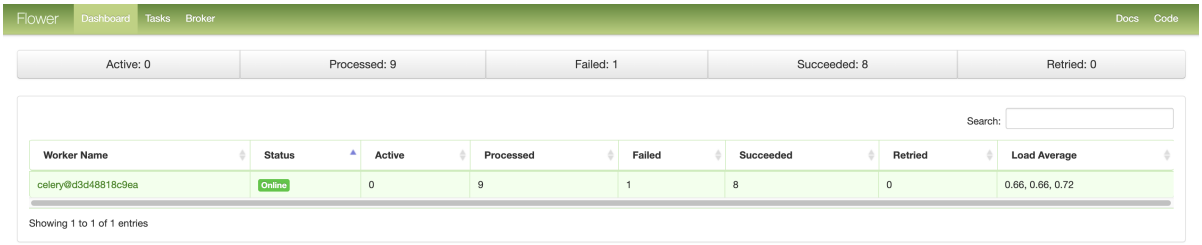
Complementando os recursos internos do Airflow, também foi utilizada a ferramenta Flower[12], que destaca-se na monitoração e administração de clusters Celery[13]. Esse tipo de cluster é utilizado pelo Airflow para garantir o funcionamento correto de cada um dos DAGs definidos, tendo em vista que a arquitetura do Airflow faz uso do Celery em seu gerenciamento de tarefas distribuídas.

Sendo assim, enquanto o painel do Airflow apresenta as informações de cada um dos DAGs e suas execuções, é através do painel Flower que se realiza o monitoramento

[12] https://flower.readthedocs.io/en/latest/
[13] https://docs.celeryq.dev/en/stable/

da infraestrutura assíncrona da ferramenta. A Figura 6, por exemplo, ilustra o dashboard inicial do Flower, no qual é possível verificar o nome de cada *worker* do Celery, o status de suas tarefas e as médias de carga do gerenciamento de filas do sistema. Dessa forma, uma visão integrada do desempenho e da saúde da pipeline de dados foi assegurada, contribuindo para a manutenção de um alto padrão de qualidade na coleta e manipulação dos dados.

Figura 6 – Dashboard Flower.



Fonte: Página inicial da interface gráfica do Flower.

Finalmente, toda a configuração do Airflow, juntamente com os scripts que definem o DAG do Politifact, foram encapsulados em múltiplos contâiners *Docker*[14] através da ferramenta *Docker Compose*[15], de maneira a garantir a consistência do ambiente de execução e facilitar a portabilidade e a replicabilidade do CheckBERT. Dessa forma, conforme outros DAGs de integração de dados forem sendo adicionados ao módulo, nenhuma outra alteração será necessária.

4.2.3 Banco de Dados

A decisão de empregar o ElasticSearch como banco de dados para este projeto baseou-se na sua eficiente indexação de campos textuais e alta performance. Estas características, aliadas a uma arquitetura distribuída, proporcionam a construção de uma pipeline de dados eficiente e garantem uma navegação fluida da interface web. Adicionalmente, optou-se por configurar um *cluster* com múltiplos nodos, adotando-se a mesma abordagem de containerização utilizada no módulo Coletor de Dados. Esta estratégia favorece o escalonamento horizontal do CheckBERT, permitindo que a performance do banco de dados seja mantida mesmo com o crescimento do volume de dados armazenados. Para o armazenamento dos dados dos artigos, criou-se um índice denominado *politifact_raw*, cujo mapeamento de dados está detalhado no Quadro 2.

[14] https://www.docker.com/
[15] https://docs.docker.com/compose/

Quadro 2 – Mapeamento do índice politifact_raw no ElasticSearch.

Propriedade	**Tipo**	**Descrição**
url	keyword	URL do artigo no site do Politifact.
author	text	Fonte da sentença (pode ser uma pessoa ou um local de publicação).
statement	text	Sentença que foi verificada.
statement_date	date	Data de publicação/proferimento de sentença.
label	keyword	Classificação da sentença pelo pelo PolitiFact.
article_text	text	Texto do artigo no PolitiFact.

Fonte: Elaboração própria.

4.2.4 Módulo Classificador BERT

Assim como descrito na Subseção 4.1.2, o módulo Classificador BERT tem como principal finalidade a preparação de um modelo BERT otimizado para tarefas de NLI no contexto proposto pelo trabalho. Para isso, foi realizada uma série de experimentações com modelos pré-treinados BERT (detalhadas na Seção 4.3), abrangendo tanto codificação *Bi-Encoder* quanto codificação *Cross-Encoder*.

Ao final da experimentação, o modelo selecionado foi integrado ao módulo API Back-end, onde realiza a classificação das sentenças. Este modelo possui codificação *Cross-Encoder* e admite um *array* de pares de strings como entrada, compostos pela sentença fornecida pelo usuário e o texto do artigo relacionado. Subsequentemente, cada par é então classificado pelo modelo, o qual retorna uma tríade de valores númericos correspondentes aos *scores* obtidos para cada tipo de classificação (concordância, neutralidade e discordância, respectivamente). Por fim, o modelo retorna o rótulo da classificação de maior valor, uma vez que, quanto maior o *score* atribuído, maior é a confiança do modelo na classificação correspondente.

4.2.5 Módulo API Back-end

Como mencionado na Subseção 4.1.2, o módulo API Back-end propõe-se a intermediar a comunicação entre o front-end e o back-end da ferramenta, através de mensagens JSON. Para tal finalidade, foi construída uma API utilizando a biblioteca Python chamada FastAPI, que fornece roteamento rápido, validação de requisições e uma interface gráfica de documentação no endpoint */docs*. A API foi projetada para rodar utilizando *Uvicorn*[16], um servidor do tipo *Asynchronous Server Gateway Interface* (ASGI) que proporciona um

[16] https://www.uvicorn.org/

desempenho superior devido ao seu suporte nativo para a concorrência.

A API implementada possui apenas dois *endpoints*: um na raiz, utilizado apenas para verificar o funcionamento da API e sem contribuições significativas para o CheckBERT, e outro no caminho */search*, responsável por receber as solicitações de busca e classificação de sentenças. Na implementação do endpoint */search*, a API faz uma consulta no banco de dados utilizando a sentença fornecida pelo usuário, e comparando-a com os títulos e textos dos artigos armazenados. Entretanto, dá-se um maior peso para os títulos, pois entende-se que geralmente as buscas fornecidas pelos usuários estão mais relacionadas com o título do artigo, em especial no contexto de informações falsas. A consulta utilizada para esta finalidade está representada na Figura 7, na qual destacou-se a seção A, onde o maior peso é atribuído aos títulos dos artigos, e a seção B, que verifica o texto dos artigos utilizando o peso padrão. Além disso, os resultados retornados pelo banco de dados são limitados a apenas um documento, tendo em vista que apenas o artigo mais relevante (isto é, aquele com maior similaridade em relação a sentença fornecida pelo usuário, de acordo com o algoritmo padrão do ElasticSearch) é desejado.

Figura 7 – Consulta por artigo mais relevante no banco de dados ElasticSearch.

```
{
  "bool": {
    "should": [
      {                                                    A)
        "match": {
          "statement": {
            "query": "<Busca Realizada Pelo Usuário>",
            "boost": 2
          }
        }
      },
      {                                                    B)
        "match": {
          "article_text": {
            "query": "<Busca Realizada Pelo Usuário>"
          }
        }
      }
    ]
  }
}
```

Fonte: Elaboração própria.

As respostas geradas pelo endpoint */search* são estruturadas de duas maneiras distintas, a depender da existência de um artigo relevante no banco de dados. A primeira estrutura, representada na Figura 8, acontece quando é encontrado um artigo relevante no banco de dados, e possui informações sobre o artigo encontrado e sobre a classificação realizada pelo modelo. Já a segunda estrutura, demonstrada na Figura 9, acontece quando

não há artigos relevantes no banco de dados (novamente, de acordo com o algoritmo padrão de buscas do ElasticSearch), e apenas possui uma mensagem indicando que este é o caso.

Ademais, destaca-se que todo o módulo foi encapsulado utilizando Docker, seguindo a mesma abordagem utilizada no módulo Coletor de Dados.

Figura 8 – Resposta do endpoint /search caso exista um artigo relevante no banco de dados.


```
{
  "result": {
    "query": "Barack Obama was arrested",
    "article": {
      "url": "https://www.politifact.com/factchecks/2021/jan/18/facebook-posts/obama-wasnt-arrested-bidens-inauguration/",
      "author": "Facebook posts",
      "statement": "“Barack Obama was arrested.”",
      "statement_date": "2021-01-17T00:00:00+00:00",
      "label": "pants-fire"
    },
    "prediction": {
      "label": "contradiction"
    }
  }
}
```


Fonte: Elaboração própria.

Figura 9 – Resposta do endpoint /search caso não exista um artigo relevante no banco de dados.


```
{
  "result": "No matches found."
}
```


Fonte: Elaboração própria.

4.2.6 Módulo Interface Gráfica

O módulo de Interface Gráfica é o único módulo que possui interação direta com o usuário, fornecendo uma interface amigável e intuitiva de consultar a classificação de sentenças por meio de uma barra de pesquisa simples. Este módulo foi totalmente desenvolvido utilizando a biblioteca *React*, que facilitou a criação de interfaces dinâmicas e interativas. Adicionalmente, o utilitário *styled-components* foi empregado para a aplicação de estilos CSS nos componentes React, contribuindo para uma experiência visual mais agradável e coerente. Além disso, é importante salientar que todo o módulo foi encapsulado com o uso da tecnologia Docker, seguindo a mesma abordagem utilizada nos módulos Coletor de Dados e API Back-end.

Diante disso, a Figura 10 expõe uma ilustração gráfica constituída por oito subfiguras, na qual cada subfigura representa uma das possíveis telas do CheckBERT. A seguir,

todas as telas apresentadas estão descritas resumidamente, com o intuito de facilitar o entendimento do uso da ferramenta:

- **Tela inicial:** ilustrada na Figura 10a, a tela inicial representa a primeira interface exposta durante a utilização do CheckBERT. É através dela que o usuário fornece sua primeira busca e a envia teclando *Enter*. Além disso, é também utilizada após uma consulta que não possui artigos relacionados no banco de dados (com exceção da barra de pesquisa, que neste cenário continua preenchida com a busca fornecida pelo usuário).
- **Tela de carregamento:** representada pela Figura 10b, esta é a tela apresentada logo após o envio de uma busca, enquanto a requisição ainda está sendo processada pela API. Raramente se mantém por mais de dois segundos.
- **Telas de classificação:** correspondem às diferentes interfaces que o usuário encontra após uma pesquisa, e variam de acordo com a classificação atribuída pelo Politifact à alegação verificada. Visando exemplificar as possíveis variações de interfaces, foram elencadas subfiguras para cada uma das possíveis classificações atribuídas, as quais estão destacadas em vermelho. São elas: verdadeira (Figura 10c), verdadeira em sua maioria (Figura 10d), meia-verdade (Figura 10e), falsa em sua maioria (Figura 10f), falsa (Figura 10g) ou falsa e possuindo uma alegação ridícula (Figura 10h).

4.3 EXPERIMENTOS

Nesta seção, são detalhados os experimentos realizados para testes de modelos a serem usados no módulo Classificador BERT, aspirando avaliar as diferentes alternativas devido ao caráter exploratório do trabalho.

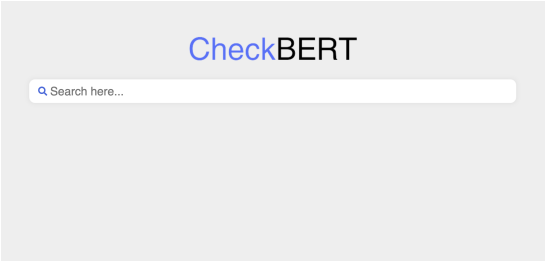
4.3.1 Conjunto de dados

Esta subseção propõe-se a detalhar o conjunto de dados utilizado em todos os experimentos, bem como seu processo de preparação.

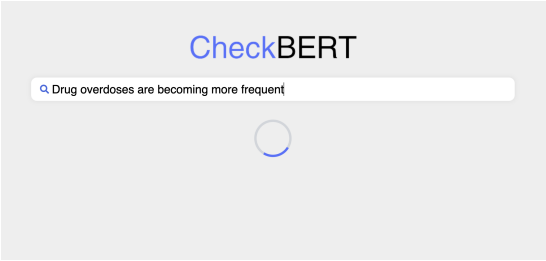
Inicialmente, selecionou-se todos os artigos do Politifact coletados pelo módulo Coletor de Dados até o dia 13 de março de 2023, totalizando 21620 artigos disponíveis no site. Esses artigos foram então convertidos para um objeto *DataFrame* da biblioteca *pandas*, que foi utilizada em todas as operações subsequentes de transformação dos dados.

No que diz respeito às transformações realizadas, primeiramente as classificações originalmente concedidas pelo site foram convertidas de acordo com a necessidade do desenvolvimento de modelos NLI. Mais especificamente, as classificações *true* e *mostly-true* foram mapeadas para *entailment* (concordância), *half-true* foi convertida para *neutral*

Figura 10 – Telas CheckBERT.

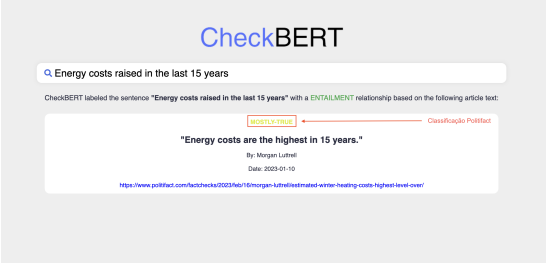


(a) Tela inicial.

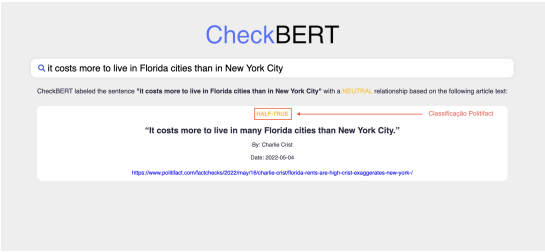


(b) Tela de carregamento.

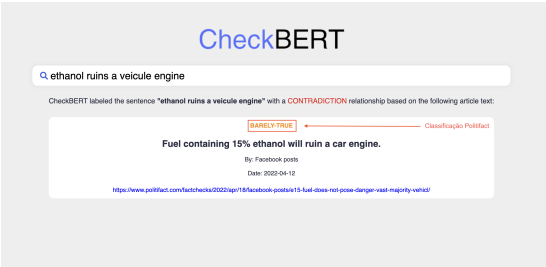
(c) Tela de classificação com alegação classificada como verdadeira pelo Politifact.



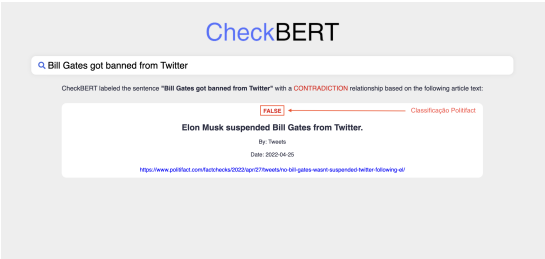
(d) Tela de classificação com alegação classificada como verdadeira em sua maioria pelo Politifact.



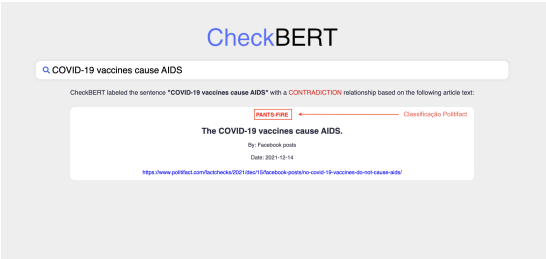
(e) Tela de classificação com alegação classificada como meia-verdade pelo Politifact.



(f) Tela de classificação com alegação classificada como falsa em sua maioria pelo Politifact.



(g) Tela de classificação com alegação classificada como falsa pelo Politifact.



(h) Tela de classificação com alegação classificada como falsa possuindo alegação ridícula pelo Politifact.

Fonte: Elaboração própria.

(neutralidade), e *mostly-false*, *false* e *pants-fire* foram mapeadas para *contradiction* (discordância).

Em seguida, construiu-se uma base de dados balanceada a partir da fusão de três amostras distintas, cada uma contendo 3642 registros de uma das classificações mapeadas no processo anterior (*entailment*, *neutral* e *contradiction*). Esses registros foram selecionados de maneira aleatória - porém reprodutível - utilizando a função *DataFrame.sample()* da biblioteca *pandas*, com o parâmetro *random_state* definido como 1. Esta abordagem assegura que cada experimento pode ser replicado exatamente da mesma maneira, garantindo assim a integridade e comparabilidade dos resultados.

Ao término do processo de transformação e amostragem, obteve-se um conjunto de dados balanceado com 10926 registros. Cada registro é composto por três variáveis: *label*, que representa a classificação mapeada no processo anterior; *statement*, que representa a alegação verificada pelo Politifact; e *article_text*, que representa o texto do artigo do Politifact. Uma descrição mais detalhada destas variáveis é apresentada no Quadro 3.

Quadro 3 – Variáveis do conjunto de dados base.

Variável	**Tipo de dado**	**Descrição**
label	*string*	Classificação mapeada a partir da classificação fornecida pelo Politifact. Representa a variável dependente do estudo e sempre possui um dos seguintes valores: *entailment*, *neutral* e *contradiction*.
statement	*string*	Alegação verificada pelo Politifact. Representa uma das variáveis independentes do estudo e também corresponde ao título do artigo.
article_text	*string*	Texto do artigo de verificação do Politifact. Representa uma das variáveis independentes do estudo.

Fonte: Elaboração própria.

Além disso, vale ressaltar que os artigos extraídos abrangem uma grande diversidade de assuntos (embora todos estejam relacionados ao cenário político norte-americano) e que informações quanto à extensão das alegações verificadas e dos textos dos artigos estão apontadas no Quadro 4. Dentre essas informações, vale ressaltar que o texto dos artigos geralmente ultrapassa o limite de extensão do Google BERT (de aproximadamente 2500 caracteres), conforme indicado pela média calculada, mas que isso foi levado em consideração na aplicação das técnicas, conforme descrito a seguir.

Já pensando em contornar a limitação do tamanho dos textos, e ao mesmo tempo alcançar uma eficácia significativa com modelos *Bi-Encoder* (os quais demonstraram ser totalmente ineficazes quando utilizandos com textos de maior extensão), foi gerado um conjunto de dados adicional com base no conjunto original descrito acima. Este conjunto

Quadro 4 – Métricas da extensão dos valores de statement e article_text.

Variável	**Média**	**Valor mínimo**	**Primeiro quartil**	**Mediana**	**Terceiro quartil**	**Valor máximo**
statement	109.69	17.00	76.00	101.00	136.00	437.00
article_text	5121.11	308.00	3747.00	4977.50	6300.75	20286.00

Fonte: Elaboração própria.

adicional mantém as mesmas características do conjunto anterior, com exceção dos textos de cada artigo, que foram subdivididos em parágrafos.

4.3.2 Modelos Bi-Encoder

Os primeiros testes realizados envolveram modelos de codificação *Bi-Encoder*, com o propósito de explorar a sua eficácia em tarefas de análise de similaridade semântica entre textos, dentro do contexto proposto. Nesta etapa, foram realizados experimentos utilizando dois modelos pré-treinados, os quais foram disponibilizados pela biblioteca *sentence-transformers*: o *all-MiniLM-L6-v2*[17] e o *all-mpnet-base-v2*[18]. Ambos os experimentos foram realizados utilizando o conjunto de dados com os textos subdivididos em parágrafos, os quais foram vetorizados em *embeddings* por cada um dos modelos, juntamente com suas alegações correspondentes. Subsequentemente, foram realizadas as análises de similaridade dos *embeddings* através da função de similaridade por cosseno (as funções por distância euclidiana e manhattan também foram utilizadas, mas não apresentaram quaisquer variações significativas em relação à similaridade por cosseno), e seus resultados estão descritos abaixo:

- **all-MiniLM-L6-v2:** os experimentos conduzidos com o modelo *all-MiniLM-L6-v2* visaram validar a hipótese de que um modelo *Bi-Encoder* de alta velocidade seria capaz de alcançar resultados significativos, uma vez que este modelo apresenta a maior velocidade de vetorização de sentenças por segundo, dentre os modelos disponibilizados na biblioteca *sentence-transformers*. Contudo, a hipótese foi rejeitada pois a abordagem não apresentou quaisquer tendências significativas (até mesmo em testes com sentenças simples) e, portanto, foi rapidamente descartada.

- **all-mpnet-base-v2:** o segundo conjunto de experimentos envolveu a utilização do modelo *all-mpnet-base-v2*, que, apesar de sua menor velocidade de vetorização, é conhecido por ter um desempenho superior em tarefas de similaridade semântica. A hipótese por trás deste teste foi a de que um modelo diferente, capaz de produzir *embeddings* mais detalhados, poderia fornecer resultados mais significativos.

[17] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[18] https://huggingface.co/sentence-transformers/all-mpnet-base-v2

Assim como no experimento anterior, foram gerados *embeddings* para todas as alegações e parágrafos do conjunto de dados, seguidos pela análise de similaridade por cosseno de cada alegação com cada um dos parágrafos de seu artigo correspondente. Após essa análise, foram derivadas as variáveis *max_predicted_value*, *min_predicted_value*, *avg_predicted_value* e *median_predicted_value*, que representam os valores máximos, mínimos, médios e medianos das similaridades por cosseno encontradas em cada artigo, respectivamente. Esses valores foram então avaliados por meio de uma análise exploratória dos resultados, almejando entender melhor a sua distribuição.

Inicialmente, foram construídos diagramas de caixa para cada uma das métricas de similaridade geradas, separadas pela classificação do artigo. Esses diagramas estão expostos na Figura 11, e não revelaram quaisquer tendências significativas que poderia ser atribuídas à classificação dos artigos. Em particular, as variáveis *max_predicted_value* e *min_predicted_value* não demonstraram quaisquer tendências de causalidade, enquanto *avg_predicted_value* e *median_predicted_value* apresentaram tendências mínimas, que não parecem ter qualquer efeito significativo na análise.

Além disso, visando compreender melhor o comportamento das métricas de similaridade geradas e suas distribuições, também foram construídos gráficos de distribuição para cada umas das variáveis derivadas por tipo de classificação, conforme exposto na Figura 12.

Observando a figura, foi possível reafirmar a falta de causalidade observada na análise anterior, uma vez que as distribuições apresentaram comportamento muito parecido independentemente da classificação do artigo. Por consequência, concluiu-se que a utilização de modelos Bi-Encoder pré-treinados não é capaz de gerar análises satisfatórias para o escopo do projeto.

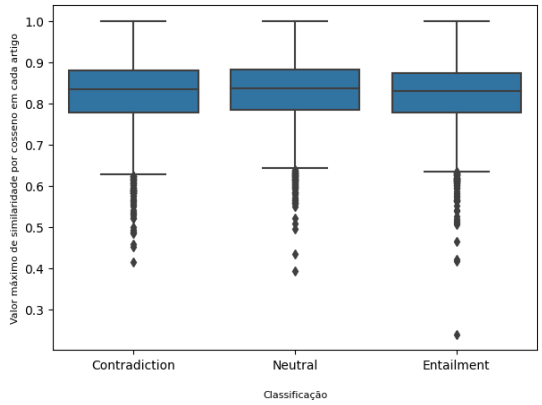
4.3.3 Modelos Cross-Encoder

Após a conclusão de que modelos *Bi-Encoder* não são suficientes para alcançar os objetivos do trabalho, foram selecionados os dois modelos *Cross-Encoder* da biblioteca *sentence-transformers* que obtiveram os melhores resultados de acurácia nos datasets SNLI[19] e MULTINLI[20]: *nli-deberta-v3-base*[21] (primeira colocação) e *nli-distilroberta-base*[22] (segunda colocação).

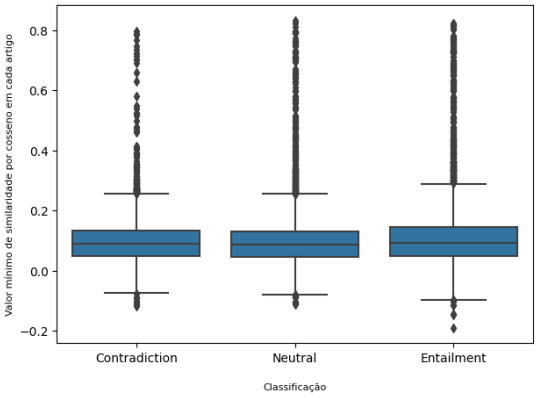
Antes de iniciar os testes, no entanto, fez-se necessário executar um novo processo de preparação de dados na base coletada pelo módulo Coletor de Dados. Esse processo

[19] https://nlp.stanford.edu/projects/snli/
[20] https://cims.nyu.edu/ sbowman/multinli/
[21] https://huggingface.co/cross-encoder/nli-deberta-v3-base
[22] https://huggingface.co/cross-encoder/nli-distilroberta-base

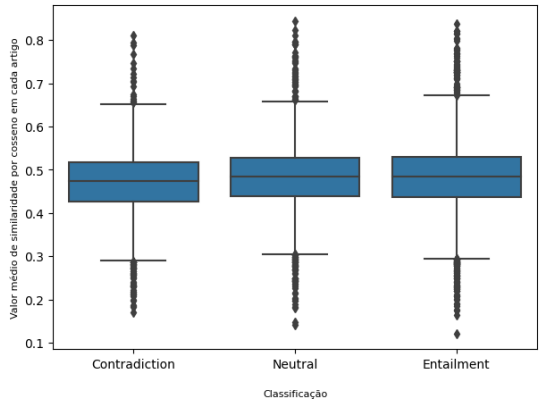
Figura 11 – Diagramas de caixa das métricas de similaridade por cosseno por classificação.



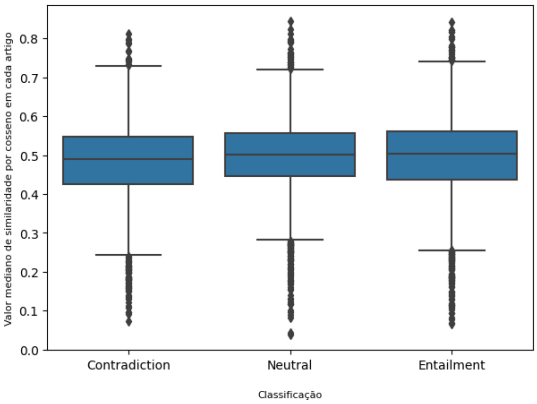
(a) max_predicted_value.



(b) min_predicted_value.



(c) avg_predicted_value.



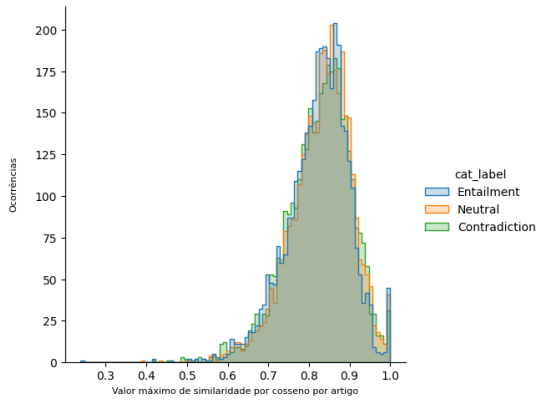
(d) median_predicted_value.

Fonte: Elaboração própria.

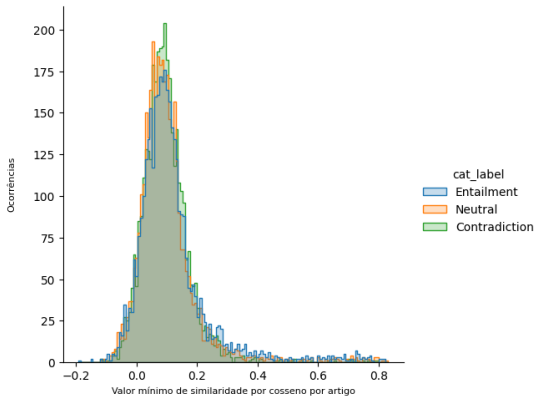
foi motivado pelo tempo elevado de avaliação dos modelos *Cross-Encoder*, bem como pelo ajuste fino dos modelos, que foi realizado posteriormente para melhorar os resultados obtidos. A preparação não aplicou alterações significativas na composição dos dados, tendo apenas separado o conjunto base (com textos completos dos artigos) em três conjuntos menores: o conjunto de treino, com 72% dos dados disponíveis (7866 registros); o conjunto de teste, com 20% dos dados disponíveis (2186 registros); e o conjunto de avaliação, com 8% dos dados disponíveis (874 registros), sendo o último utilizado para avaliar o modelo durante o período de treinamento. Todas as divisões foram realizadas utilizando a função *model_selection.train_test_split* da biblioteca *sklearn*, primeiramente com os parâmetros *test_size* e *random_state* definidos como 0.2 e 42, respectivamente, e então novamente com os parâmetros definidos como 0.1 e 42, respectivamente (na segunda divisão, foi utilizada a base com 80% dos dados extraída na primeira divisão).

Ambos os modelos foram então avaliados utilizando os dados do conjunto de teste, com o auxílio da classe *CESoftmaxAccuracyEvaluator*, disponibilizada no módulo *cross_encoder.evaluation* da biblioteca *sentence-transformers*. Os dois modelos apresentaram acurácias semelhantes, tendo *nli-distilroberta-base* alcançado aproximadamente 31,93%

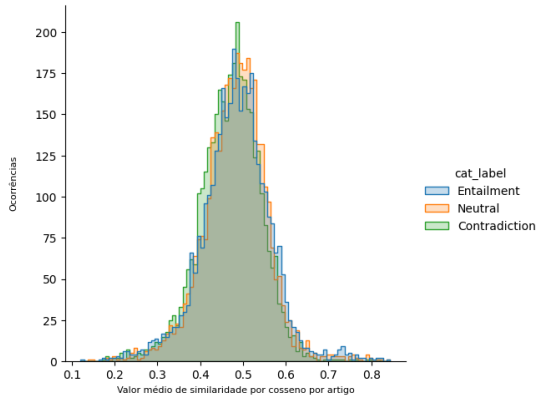
Figura 12 – Diagramas de distribuição das métricas de similaridade por cosseno por classificação.



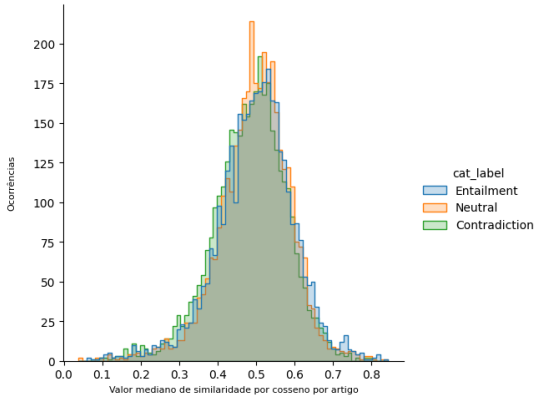
(a) max_predicted_value.



(b) min_predicted_value.



(c) avg_predicted_value.



(d) median_predicted_value.

Fonte: Elaboração própria.

e *nli-deberta-v3-base* alcançado aproximadamente 32,11%.

Em decorrência dos resultados obtidos, foi concluído que o uso de modelos *Cross-Encoders* pré-treinados também não é capaz de apresentar resultados suficientemente satisfatórios para o objetivo do trabalho, tornando necessário um ajuste fino destes modelos com dados contextualizados. Assim sendo, prosseguiu-se com o ajuste fino do modelo *nli-deberta-v3-base* (uma vez que este apresentou melhores resultados) seguindo duas abordagens distintas, as quais estão detalhadas a seguir:

- **Ajuste fino *nli-deberta-v3-base* com o recorte inicial dos artigos:** esta abordagem utiliza as alegações verificadas por cada artigo, juntamente com seus textos completos, para realizar o ajuste fino do modelo. No entanto, devido à limitação de extensão textual do BERT, artigos mais longos acabam sendo recortados, o que limita o treinamento às seções iniciais dos textos.

  Além dos pares de entradas textuais, outros parâmetros também foram utilizados para o ajuste fino do modelo. As descrições de cada parâmetro, juntamente com seus valores utilizados, estão detalhadas a seguir no Quadro 5.

Quadro 5 – Parâmetros de treino nli-deberta-v3-base.

Parâmetro	**Descrição**	**Valor usado**
train_dataloader	Objeto da classe *DataLoader* disponível no módulo *utils.data* da biblioteca *pytorch*.	Objeto *DataLoader* configurado para alterar aleatoriamente a ordem dos artigos e usar conjuntos de 16 artigos por vez (*shuffle=True* e *batch_size=16*).
evaluator	Objeto de alguma das classes disponíveis no módulo *cross_encoder.evaluation* da biblioteca *sentence-transformers*.	Objeto da classe *CESoftmaxAccuracyEvaluator* configurado com a amostra de 874 artigos definida no item anterior.
epochs	Número de épocas do treinamento	3.
evaluation_steps	Representa de quantas em quantas atualizações do modelo a avaliação deve ser realizada.	1000.
warmup_steps	Representa a quantidade inicial de atualizações do modelo que deve ser realizada com menor taxa de aprendizado, uma vez que o modelo ainda não possui muitos exemplos dos dados e pode afetar a assertividade das atualizações.	148.

Fonte: Elaboração própria.

Após o ajuste fino do modelo, a avaliação de sua acurácia foi conduzida utilizando a mesma abordagem da avaliação dos modelos pré-treinados, a fim de possibilitar a comparação do modelo *nli-deberta-v3-base* antes e depois do ajuste fino. Ao final do processo, o modelo retreinado apresentou acurácia de aproximadamente 63,58%, superando o modelo sem treinamento em aproximadamente 31,44% dos casos.

- **Ajuste fino *nli-deberta-v3-base* com o recorte final dos artigos:** apesar da melhoria significativa apresentada na abordagem anterior, levantou-se a hipótese de que, como o Politifact costuma apresentar suas considerações nos parágrafos finais do texto (os quais muitas vezes são utilizados devido a limitação de extensão textual do BERT), o modelo poderia apresentar resultados melhores caso o recorte de textos extensos fosse realizado utilizando as seções finais dos artigos. Para validar essa hipótese, recortou-se os textos de todos os artigos do conjunto de dados para conter apenas os 2560 caracteres finais (para textos menores, não houve qualquer alteração), de acordo com a seguinte lógica: o número máximo de *tokens* (divisões de palavras realizadas pelo BERT) suportado pelo modelo *nli-deberta-v3-base* é de 512 por classificação, e sabe-se que cada token representa, em média, 4 caracteres. Considerando um espaço entre cada *token*, foi definida uma estimativa de que o

modelo seria capaz de lidar com aproximadamente $512 * (4 + 1)$ caracteres, que resulta em 2560. No restante, o processo de treinamento seguiu as mesmas etapas da abordagem anterior.

Após o ajuste fino, a avaliação do modelo apontou acurácia de aproximadamente 78,36%, superando o modelo com recorte inicial em aproximadamente 14,78% dos casos, e alcançando resultados suficientemente satisfatórios para integração do modelo ao CheckBERT.

Por fim, visando facilitar o acompanhamento dos resultados obtidos, todos os modelos Cross-Encoder utilizados, juntamente com suas respectivas acurácias, foram elencados no Quadro 6.

Quadro 6 – Acurácias obtidas em modelos Cross-Encoder.

Modelo	**Acurácia Obtida**
nli-distilroberta-base pré-treinado	31,93%
nli-deberta-v3-base pré-treinado	32,11%
nli-deberta-v3-base após ajuste fino com recorte inicial dos artigos	63,58%
nli-deberta-v3-base após ajuste fino com recorte final dos artigos	78,36%

Fonte: Elaboração própria.

5 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Neste trabalho, apresentou-se a concepção e implementação de um sistema web destinado a combater uma das principais problemáticas da sociedade moderna: a disseminação de informações falsas. A abordagem adotada cobriu desde o desenvolvimento de *pipelines* de extração de dados até a implementação de uma interface gráfica, incluindo, entre essas duas extremidades, a definição e utilização de um banco de dados *ElasticSearch*, a idealização e desenvolvimento de uma API web, e o ajuste fino e utilização de um modelo BERT otimizado para tarefas de NLI.

Além disso, foram abordados conceitos como *web scraping*, transformação de dados, análise exploratória de dados, *machine learning*, *deep learning*, grafos (especificamente DAGs), computação paralela e distribuída, entre outros. Sendo todos esses tópicos essenciais para o desenvolvimento do CheckBERT, bem como para a fundamentação teórica do trabalho.

Já no que se refere ao modelo do Classificador BERT, apesar de seus resultados possuírem espaço para melhorias, acredita-se que o estudo alcançou acurácia suficiente para cumprir seu objetivo, tendo demonstrado melhorias significativas em comparação ao modelo *nli-deberta-v3-base*, quando analisado sob a ótica do contexto de artigos jornalísticos. Entretanto, visando melhorar ainda mais a acurácia do modelo, algumas estratégias que podem auxiliar, além da coleta de mais fontes de dados, são:

- Utilização de sentenças manualmente introduzidas na base de dados do modelo, com o intuito de combater cenários nos quais a sentença fornecida é sintaticamente similar a sentença analisada pelo site verificador, mas semanticamente oposta (o modelo ainda apresenta falhas ocasionais nessa conjectura);
- Adição de uma camada de sumarização de texto antes do treinamento do modelo, visando extrair apenas os fatos e informações relevantes do artigo;
- Aplicação de um maior pré-processamento de dados no treinamento do modelo, aspirando garantir a qualidade de dados e remover caracteres ou sentenças indesejadas;

Finalmente, quanto às sugestões de **Trabalhos Futuros**, pode-se listar as seguintes temáticas não abordadas nesse projeto:

- Adição de outras fontes de dados ao módulo Coletor de Dados.
- Utilização de busca semântica nas consultas de artigos no *ElasticSearch* (viável a partir do armazenamento de *embeddings* do conteúdo textual dos artigos).
- Incorporação de uma rotina de *Machine Learning Operations* (MLOps) para gerenciar e monitorar o processo de melhoramento contínuo do modelo.

- Aplicação de melhorias na Interface Gráfica, tornando-a mais responsiva e facilitando sua navegação por parte do usuário.
- Expansão da funcionalidade do CheckBERT para outras línguas além da inglesa, desde a coleta de dados até o treinamento do modelo.

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.

DEVLIN, Jacob *et al.* BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *In:* Disponível em: https://arxiv.org/abs/1810.04805.

DIAZGRANADOS, Hernan. 70% de los latinoamericanos desconoce cómo detectar una fake news. **Karpersky Daily**, 4 fev. 2020. Disponível em: https://latam.kaspersky.com/blog/70-de-los-latinoamericanos-desconoce-como-detectar-una-fake-news/17015/. Acesso em: 7 dez. 2022.

FIORE, Marcelo; DEVESAS CAMPOS, Marco. The Algebra of Directed Acyclic Graphs. *In:* **Computation, Logic, Games, and Quantum Foundations. The Many Facets of Samson Abramsky: Essays Dedicated to Samson Abramsky on the Occasion of His 60th Birthday**. Edição: Bob Coecke, Luke Ong e Prakash Panangaden. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. P. 37–51. ISBN 978-3-642-38164-5. DOI: 10.1007/978-3-642-38164-5_4. Disponível em: https://doi.org/10.1007/978-3-642-38164-5_4.

KALIYAR, Rohit Kumar; GOSWAMI, Anurag; NARANG, Pratik. FakeBERT: Fake news detection in social media with a BERT-based deep learning approach. **Multimedia Tools and Applications**, v. 80, n. 8, p. 11765–11788, mar. 2021. ISSN 1573-7721. DOI: 10.1007/s11042-020-10183-2. Disponível em: https://doi.org/10.1007/s11042-020-10183-2.

LAZER, David M. J. *et al.* The science of fake news. **Science**, v. 359, n. 6380, p. 1094–1096, 2018. DOI: 10.1126/science.aao2998. eprint: https://www.science.org/doi/pdf/10.1126/science.aao2998. Disponível em: https://www.science.org/doi/abs/10.1126/science.aao2998.

MITCHELL, Ryan. **Web Scraping with Python**: Collecting More Data from the Modern Web. [*S.l.*]: O'Reilly Media, Inc., 2018.

MÜLLER, Martin; SALATHÉ, Marcel; KUMMERVOLD, Per E. **COVID-Twitter-BERT: A Natural Language Processing Model to Analyse COVID-19 Content on Twitter**. [*S.l.*]: arXiv, 2020. DOI: 10.48550/ARXIV.2005.07503. Disponível em: https://arxiv.org/abs/2005.07503.

NEMESLAKI, András; POCSAROVSZKY, Károly. Web crawler research methodology. *In:* (22nd European Regional Conference of the International Telecommunications Society (ITS): "Innovative ICT Applications - Emerging Regulatory, Economic and Policy Issues", Budapest, Hungary, 18th-21st September, 2011). Disponível em: http://hdl.handle.net/10419/52173.

PICKELL, Devin. What Is Web Scraping? How to Automate Web Data Collection. **G2**, fev. 2023. Disponível em: https://www.g2.com/articles/web-scraping.

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.

RIVEIRA, Carolina. O brasileiro está mais alerta sobre fake news — mas ficou paranoico. **Exame**, 11 ago. 2021. Disponível em: https://exame.com/brasil/brasileiro-confianca-fake-news-pesquisa/. Acesso em: 7 dez. 2022.

RUBIO HANCOCK, JAIME. Dicionário Oxford dedica sua palavra do ano, 'pós-verdade', a Trump e Brexit. **El País**, 17 nov. 2016. Disponível em: https://brasil.elpais.com/brasil/2016/11/16/internacional/1479308638_931299.html. Acesso em: 8 jun. 2023.

SCHWARTZ, Barry. Google: BERT now used on almost every English query. **Search Engine Land**, 15 out. 2020. Disponível em: https://searchengineland.com/google-bert-used-on-almost-every-english-query-342193. Acesso em: 7 dez. 2022.

SOUZA, Rodrigo. **Comparação de Métodos para Inferência em Linguagem Natural**. 2020. Master thesis – Universidade de São Paulo.

VEJA. 'Fake news' é eleita palavra do ano por dicionário Collins. **Veja**, 2 nov. 2017. Disponível em: https://veja.abril.com.br/mundo/fake-news-e-eleita-palavra-do-ano-por-dicionario-collins/. Acesso em: 7 dez. 2022.

—

APÊNDICE A – CÓDIGO-FONTE

COLETOR DE DADOS (airflow-fakebert):

- **airflow-fakebert/requirements.txt**

```
alembic==1.9.2
amqp==5.1.1
anyio==3.6.2
apache-airflow==2.5.1
apache-airflow-providers-celery==3.1.0
apache-airflow-providers-common-sql==1.3.3
apache-airflow-providers-ftp==3.3.0
apache-airflow-providers-http==4.1.1
apache-airflow-providers-imap==3.1.1
apache-airflow-providers-sqlite==3.3.1
apispec==3.3.2
argcomplete==2.0.0
arrow==1.2.3
attrs==22.2.0
Babel==2.11.0
beautifulsoup4==4.11.2
billiard==3.6.4.0
blinker==1.5
cachelib==0.9.0
cattrs==22.2.0
celery==5.2.7
certifi==2022.12.7
cffi==1.15.1
charset-normalizer==2.1.1
click==8.1.3
click-didyoumean==0.3.0
click-plugins==1.1.1
click-repl==0.2.0
clickclick==20.10.2
colorama==0.4.6
colorlog==4.8.0
commonmark==0.9.1
ConfigUpdater==3.1.1
connexion==2.14.1
cron-descriptor==1.2.32
```

```
croniter==1.3.8
cryptography==38.0.4
Deprecated==1.2.13
dill==0.3.1.1
dnspython==2.3.0
docutils==0.19
elastic-transport==8.4.0
elasticsearch==8.6.1
email-validator==1.3.0
exceptiongroup==1.1.0
Flask==2.2.2
Flask-AppBuilder==4.1.4
Flask-Babel==2.0.0
Flask-Caching==2.0.2
Flask-JWT-Extended==4.4.4
Flask-Login==0.6.2
Flask-Session==0.4.0
Flask-SQLAlchemy==2.5.1
Flask-WTF==1.1.1
flower==1.2.0
graphviz==0.20.1
greenlet==2.0.1
gunicorn==20.1.0
h11==0.14.0
httpcore==0.16.3
httpx==0.23.3
humanize==4.4.0
idna==3.4
inflection==0.5.1
itsdangerous==2.1.2
Jinja2==3.1.2
jsonschema==4.17.3
kombu==5.2.4
lazy-object-proxy==1.9.0
linkify-it-py==2.0.0
lockfile==0.12.2
Mako==1.2.4
Markdown==3.4.1
markdown-it-py==2.1.0
```

```
MarkupSafe==2.1.2
marshmallow==3.19.0
marshmallow-enum==1.5.1
marshmallow-oneofschema==3.0.1
marshmallow-sqlalchemy==0.26.1
mdit-py-plugins==0.3.3
mdurl==0.1.2
numpy==1.24.2
packaging==21.3
pandas==1.5.3
pathspec==0.9.0
pendulum==2.1.2
pluggy==1.0.0
prison==0.2.1
prometheus-client==0.15.0
prompt-toolkit==3.0.36
psutil==5.9.4
pycparser==2.21
Pygments==2.14.0
PyJWT==2.6.0
pyparsing==3.0.9
pyrsistent==0.19.3
python-daemon==2.3.2
python-dateutil==2.8.2
python-nvd3==0.15.0
python-slugify==7.0.0
pytz==2022.7.1
pytzdata==2020.1
PyYAML==6.0
requests==2.28.2
requests-toolbelt==0.10.1
rfc3986==1.5.0
rich==13.1.0
setproctitle==1.3.2
six==1.16.0
sniffio==1.3.0
soupsieve==2.3.2.post1
SQLAlchemy==1.4.46
SQLAlchemy-JSONField==1.0.1.post0
```

```
SQLAlchemy-Utils==0.39.0
sqlparse==0.4.3
tabulate==0.9.0
tenacity==8.1.0
termcolor==2.2.0
text-unidecode==1.3
tornado==6.1
typing_extensions==4.4.0
uc-micro-py==1.0.1
unicodecsv==0.14.1
urllib3==1.26.14
vine==5.0.0
wcwidth==0.2.6
Werkzeug==2.2.2
wrapt==1.14.1
WTForms==3.0.1
```

- **airflow-fakebert/___init___.py**
- **airflow-fakebert/docker-compose.yaml**

```
# Basic Airflow cluster configuration for CeleryExecutor
    with Redis and PostgreSQL.
#
# NOTE: This configuration is for local development. Do not
    use it in a production deployment.
#
# This configuration supports basic configuration using
    environment variables or an .env file.
# The following variables are supported:
#
# AIRFLOW_IMAGE_NAME           - Docker image name used to
    run Airflow.
#                                Default: apache/airflow
    :2.5.1
# AIRFLOW_UID                  - User ID in Airflow
    containers
#                                Default: 50000
# AIRFLOW_PROJ_DIR             - Base path to which all the
    files will be volumed.
#                                Default: .
```

```
# Those configurations are useful mostly in case of
    standalone testing/running Airflow in test/try-out mode
#
# _AIRFLOW_WWW_USER_USERNAME   - Username for the
    administrator account (if requested).
#                                   Default: airflow
# _AIRFLOW_WWW_USER_PASSWORD   - Password for the
    administrator account (if requested).
#                                   Default: airflow
# _PIP_ADDITIONAL_REQUIREMENTS - Additional PIP requirements
     to add when starting all containers.
#                                   Default: ''
#
# Feel free to modify this file to suit your needs.
---
version: '3'
x-airflow-common:
  &airflow-common
  # In order to add custom dependencies or upgrade provider
      packages you can use your extended image.
  # Comment the image line, place your Dockerfile in the
      directory where you placed the docker-compose.yaml
  # and uncomment the "build" line below, Then run `docker-
      compose build` to build the images.
  image: ${AIRFLOW_IMAGE_NAME:-apache/airflow:2.5.1-python3
      .10}
  # build: .
  environment:
    &airflow-common-env
    AIRFLOW__CORE__EXECUTOR: CeleryExecutor
    AIRFLOW__DATABASE__SQL_ALCHEMY_CONN: postgresql+psycopg2
        ://airflow:airflow@postgres/airflow
    # For backward compatibility, with Airflow <2.3
    AIRFLOW__CORE__SQL_ALCHEMY_CONN: postgresql+psycopg2://
        airflow:airflow@postgres/airflow
    AIRFLOW__CELERY__RESULT_BACKEND: db+postgresql://airflow
        :airflow@postgres/airflow
    AIRFLOW__CELERY__BROKER_URL: redis://:@redis:6379/0
    AIRFLOW__CORE__FERNET_KEY: ''
```

```
    AIRFLOW__CORE__DAGS_ARE_PAUSED_AT_CREATION: 'true'
    AIRFLOW__CORE__LOAD_EXAMPLES: 'false'
    AIRFLOW__API__AUTH_BACKENDS: 'airflow.api.auth.backend.
        basic_auth,airflow.api.auth.backend.session'
    _PIP_ADDITIONAL_REQUIREMENTS: ${
        _PIP_ADDITIONAL_REQUIREMENTS:- arrow pandas numpy
        elasticsearch>=8.6}
  volumes:
    - ${AIRFLOW_PROJ_DIR:-.}/dags:/opt/airflow/dags
    - ${AIRFLOW_PROJ_DIR:-.}/logs:/opt/airflow/logs
    - ${AIRFLOW_PROJ_DIR:-.}/plugins:/opt/airflow/plugins
  user: "${AIRFLOW_UID:-50000}:0"
  depends_on:
    &airflow-common-depends-on
    redis:
      condition: service_healthy
    postgres:
      condition: service_healthy

services:
  postgres:
    image: postgres:13
    environment:
      POSTGRES_USER: airflow
      POSTGRES_PASSWORD: airflow
      POSTGRES_DB: airflow
    volumes:
      - postgres-db-volume:/var/lib/postgresql/data
    healthcheck:
      test: ["CMD", "pg_isready", "-U", "airflow"]
      interval: 5s
      retries: 5
    restart: always

  redis:
    image: redis:latest
    expose:
      - 6379
    healthcheck:
```

```
      test: ["CMD", "redis-cli", "ping"]
      interval: 5s
      timeout: 30s
      retries: 50
    restart: always

  airflow-webserver:
    <<: *airflow-common
    command: webserver
    ports:
      - 8080:8080
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost
         :8080/health"]
      interval: 10s
      timeout: 10s
      retries: 5
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

  airflow-scheduler:
    <<: *airflow-common
    command: scheduler
    healthcheck:
      test: ["CMD-SHELL", 'airflow jobs check --job-type
         SchedulerJob --hostname "$${HOSTNAME}"']
      interval: 10s
      timeout: 10s
      retries: 5
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

  airflow-worker:
```

```
    <<: *airflow-common
    command: celery worker
    healthcheck:
      test:
        - "CMD-SHELL"
        - 'celery --app airflow.executors.celery_executor.
            app inspect ping -d "celery@$${HOSTNAME}"'
      interval: 10s
      timeout: 10s
      retries: 5
    environment:
      <<: *airflow-common-env
      # Required to handle warm shutdown of the celery
          workers properly
      # See https://airflow.apache.org/docs/docker-stack/
          entrypoint.html#signal-propagation
      DUMB_INIT_SETSID: "0"
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

  airflow-triggerer:
    <<: *airflow-common
    command: triggerer
    healthcheck:
      test: ["CMD-SHELL", 'airflow jobs check --job-type
          TriggererJob --hostname "$${HOSTNAME}"']
      interval: 10s
      timeout: 10s
      retries: 5
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

  airflow-init:
```

```
<<: *airflow-common
entrypoint: /bin/bash
# yamllint disable rule:line-length
command:
  - -c
  - |
    function ver() {
      printf "%04d%04d%04d%04d" $${1//./ }
    }
    airflow_version=$$(AIRFLOW__LOGGING__LOGGING_LEVEL=
       INFO && gosu airflow airflow version)
    airflow_version_comparable=$$(ver $${airflow_version
       })
    min_airflow_version=2.2.0
    min_airflow_version_comparable=$$(ver $${
       min_airflow_version})
    if (( airflow_version_comparable <
       min_airflow_version_comparable )); then
      echo
      echo -e "\033[1;31mERROR!!!: Too old Airflow
         version $${airflow_version}!\e[0m"
      echo "The minimum Airflow version supported: $${
         min_airflow_version}. Only use this or higher!"
      echo
      exit 1
    fi
    if [[ -z "${AIRFLOW_UID}" ]]; then
      echo
      echo -e "\033[1;33mWARNING!!!: AIRFLOW_UID not set
         !\e[0m"
      echo "If you are on Linux, you SHOULD follow the
         instructions below to set "
      echo "AIRFLOW_UID environment variable, otherwise
         files will be owned by root."
      echo "For other operating systems you can get rid
         of the warning with manually created .env file
         :"
      echo "    See: https://airflow.apache.org/docs/
         apache-airflow/stable/howto/docker-compose/
```

```
      index.html#setting-the-right-airflow-user"
  echo
fi
one_meg=1048576
mem_available=$$(($$(getconf _PHYS_PAGES) * $$(
   getconf PAGE_SIZE) / one_meg))
cpus_available=$$(grep -cE 'cpu[0-9]+' /proc/stat)
disk_available=$$(df / | tail -1 | awk '{print $$4
   }')
warning_resources="false"
if (( mem_available < 4000 )) ; then
  echo
  echo -e "\033[1;33mWARNING!!!: Not enough memory
     available for Docker.\e[0m"
  echo "At least 4GB of memory required. You have $$
     (numfmt --to iec $$((mem_available * one_meg)))
     "
  echo
  warning_resources="true"
fi
if (( cpus_available < 2 )); then
  echo
  echo -e "\033[1;33mWARNING!!!: Not enough CPUS
     available for Docker.\e[0m"
  echo "At least 2 CPUs recommended. You have $${
     cpus_available}"
  echo
  warning_resources="true"
fi
if (( disk_available < one_meg * 10 )); then
  echo
  echo -e "\033[1;33mWARNING!!!: Not enough Disk
     space available for Docker.\e[0m"
  echo "At least 10 GBs recommended. You have $$(
     numfmt --to iec $$((disk_available * 1024 )))"
  echo
  warning_resources="true"
fi
if [[ $${warning_resources} == "true" ]]; then
```

```
            echo
            echo -e "\033[1;33mWARNING!!!: You have not enough
                resources to run Airflow (see above)!\e[0m"
            echo "Please follow the instructions to increase
               amount of resources available:"
            echo "   https://airflow.apache.org/docs/apache-
               airflow/stable/howto/docker-compose/index.html#
               before-you-begin"
            echo
          fi
          mkdir -p /sources/logs /sources/dags /sources/
             plugins
          chown -R "${AIRFLOW_UID}:0" /sources/{logs,dags,
             plugins}
          exec /entrypoint airflow version
      # yamllint enable rule:line-length
      environment:
        <<: *airflow-common-env
        _AIRFLOW_DB_UPGRADE: 'true'
        _AIRFLOW_WWW_USER_CREATE: 'true'
        _AIRFLOW_WWW_USER_USERNAME: ${
           _AIRFLOW_WWW_USER_USERNAME:-airflow}
        _AIRFLOW_WWW_USER_PASSWORD: ${
           _AIRFLOW_WWW_USER_PASSWORD:-airflow}
        _PIP_ADDITIONAL_REQUIREMENTS: ''
      user: "0:0"
      volumes:
        - ${AIRFLOW_PROJ_DIR:-.}:/sources

    airflow-cli:
      <<: *airflow-common
      profiles:
        - debug
      environment:
        <<: *airflow-common-env
        CONNECTION_CHECK_MAX_COUNT: "0"
      # Workaround for entrypoint issue. See: https://github.
         com/apache/airflow/issues/16252
      command:
```

```
      - bash
      - -c
      - airflow

  # You can enable flower by adding "--profile flower"
     option e.g. docker-compose --profile flower up
  # or by explicitly targeted on the command line e.g.
     docker-compose up flower.
  # See: https://docs.docker.com/compose/profiles/
  flower:
    <<: *airflow-common
    command: celery flower
    profiles:
      - flower
    ports:
      - 5555:5555
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost
         :5555/"]
      interval: 10s
      timeout: 10s
      retries: 5
    restart: always
    depends_on:
      <<: *airflow-common-depends-on
      airflow-init:
        condition: service_completed_successfully

volumes:
  postgres-db-volume:
```

- **airflow-fakebert/dags/politifact_dag.py**

```
import datetime

from airflow.models import DAG

from politifact.crawler import get_rulings,
   get_newer_articles_urls, flat_article_urls_list, \
    filter_already_extracted_urls
```

```
from politifact.scraper import scrap_article

with DAG(
    "politifact_dag",
    schedule_interval=datetime.timedelta(minutes=15),
    start_date=datetime.datetime(2023, 2, 1),
    catchup=False,
    default_args={"depends_on_past": False, "retries": 0},
) as dag:
    urls_list = get_newer_articles_urls.expand(ruling=
       get_rulings())
    filtered_urls = filter_already_extracted_urls(
       flat_article_urls_list(urls_list))
    scrap_article.expand(url=filtered_urls)
```

- **airflow-fakebert/dags/___init___.py**
- **airflow-fakebert/dags/politifact/scraper.py**

```
from datetime import datetime
from os import environ

import arrow
import httpx

from bs4 import BeautifulSoup
from httpx import Response
from airflow.decorators import task

from elastic_search.base import ESManager
from politifact.models import ArticleData



@task
def scrap_article(url: str):
    print(f"URL: {url}")
    article_resp = _get_article(url)
    article_data = _parse_article_data(article_resp, url)
    print(f"Author: {article_data.author}")
    print(f"Statement: {article_data.statement}")
    print(f"Statement date: {article_data.statement_date}")
```

```
    print(f"Label: {article_data.label}")
    print(f"Full text: {article_data.article_text}")
    _store_article_data(article_data)


def _get_article(url: str) -> Response:
    try:
        return httpx.get(url, timeout=None)
    except Exception:
        raise Exception(f"Algo deu errado. URL: {url}")


def _parse_article_data(article_resp: Response, url: str) ->
    ArticleData:
    soup = BeautifulSoup(article_resp.text, "html.parser")
    return ArticleData(
        url=url,
        author=_parse_author(soup),
        statement=_parse_statement(soup),
        statement_date=_parse_statement_date(soup),
        label=_parse_label(soup),
        article_text=_parse_article_text(soup)
    )


def _parse_author(soup: BeautifulSoup) -> str:
    assert (author_link := soup.find('a', {'class': 'm-
       statement__name'}))
    return author_link.text.strip()


def _parse_statement(soup: BeautifulSoup) -> str:
    assert (statement_div := soup.find("div", {"class": "m-
       statement__quote"}))
    return statement_div.text.strip()


def _parse_statement_date(soup: BeautifulSoup) -> datetime:
    assert (article_description := soup.find("div", {"class
```

```
        ": "m-statement__desc"}))

    if "stated on" not in article_description.text:
        raise Exception(f"Unexpected article description: {
            article_description.text}")

    assert (date := arrow.get(article_description.text, "
        MMMM D, YYYY"))
    return date.datetime


def _parse_label(soup: BeautifulSoup) -> str:
    assert (truth_o_meter_div := soup.find('div', {'class':
        'm-statement__meter'}))
    assert (label_image_div := truth_o_meter_div.find('div')
        )
    assert (label_img := label_image_div.find('img'))
    assert (label := label_img.get("alt"))
    return label


def _parse_article_text(soup: BeautifulSoup) -> str:
    assert (article_section := soup.find("article", {"class
        ": "m-textblock"}))
    for featured_fact_check in article_section.select('
        section.o-pick'):
        featured_fact_check.extract()
    return article_section.text


def _store_article_data(data: ArticleData):
    index_name = "politifact_raw"

    es_manager = ESManager(environ.get("ES_HOST", "http://
        localhost:9200"))
    es_manager.create_index(
        index_name=index_name,
        mapping=ArticleData.get_elastic_search_mapping(),
    )
```

```
        es_manager.populate_index(index_name, document=data.dict
            ())
```

- **airflow-fakebert/dags/politifact/models.py**

```
from datetime import datetime
from typing import NamedTuple


class ArticleData(NamedTuple):
    author: str
    statement: str
    statement_date: datetime
    label: str
    article_text: str
    url: str

    def dict(self) -> dict:
        return {
            "url": self.url,
            "author": self.author,
            "statement": self.statement,
            "statement_date": self.statement_date,
            "label": self.label,
            "article_text": self.article_text,
        }

    @classmethod
    def get_elastic_search_mapping(cls) -> dict:
        return {
            "mappings": {
                "properties": {
                    "url": {"type": "keyword"},
                    "author": {"type": "text"},
                    "statement": {"type": "text"},
                    "statement_date": {"type": "date"},
                    "label": {"type": "keyword"},
                    "article_text": {"type": "text"},
                }
```

```
            }
        }
```

- **airflow-fakebert/dags/politifact/crawler.py**

```
from os import environ
from typing import Tuple, List

import httpx
from airflow.decorators import task
from airflow.models import Variable
from bs4 import BeautifulSoup

from elastic_search.base import ESManager
from httpx import RequestError

API_ENDPOINT = "https://www.politifact.com/factchecks/list/?
    page={page}&ruling={ruling}"


@task
def get_rulings() -> Tuple[str, str, str, str, str, str]:
    return (
        "true",
        "mostly-true",
        "half-true",
        "barely-true",
        "false",
        "pants-fire"
    )


@task
def get_newer_articles_urls(ruling: str) -> List[str]:
    print(f"Ruling: {ruling}")
    result = []

    try:
        ruling_metadata = Variable.get(
            f"politifact_raw_metadata_{ruling}",
```

```
            deserialize_json=True
        )
    except KeyError:
        print(f"Creating `politifact_raw_metadata_{ruling}`
            variable")
        ruling_metadata = {
            "is_fully_synced": False,
            "last_page_visited": 0
        }
        Variable.set(
            f"politifact_raw_metadata_{ruling}",
            ruling_metadata,
            serialize_json=True
        )

    print(f"{ruling_metadata=}")

    is_fully_synced = ruling_metadata["is_fully_synced"]
    last_page_visited = ruling_metadata["last_page_visited"]
    if is_fully_synced:
        last_page_visited = 0

    page_limit = 5
    for i in range(page_limit):
        url = API_ENDPOINT.format(page=last_page_visited +
            1, ruling=ruling)
        print(f"Sending request to: {url}")

        try:
            response = httpx.get(url, timeout=60)
            if response.status_code != 200:
                if response.status_code == 500 and "Someone'
                    s up to no good." in response.text:
                    last_page_visited += 1
                    continue
                raise RequestError(f"Unexpected status code:
                    [{response.status_code}]")
        except RequestError as e:
            print(f"RequestError found! Exception: {e}")
```

```
            return result

        last_page_visited += 1

        if "no results found" in response.text.lower():
            print(f"{response.text=}")
            is_fully_synced = True
            break

        result.extend(_parse_urls(response.text))

    Variable.update(
        f"politifact_raw_metadata_{ruling}",
        {
            "is_fully_synced": is_fully_synced,
            "last_page_visited": last_page_visited
        },
        serialize_json=True
    )

    return result


def _parse_urls(html: str) -> List[str]:
    soup = BeautifulSoup(html, "html.parser")
    assert (article_list := soup.find("ul", {"class": "o-
       listicle__list"})), f"Unexpected HTML: {html}"
    return [
        f"https://www.politifact.com{url.get('href')}"
        for url in article_list.find_all("a")
        if "personalities" not in url.get('href')
    ]


@task
def flat_article_urls_list(urls_list: List[List[str]]) ->
   List[str]:
    return [item for sublist in urls_list for item in
       sublist]
```

```
@task
def filter_already_extracted_urls(urls: List[str]) -> List[
    str]:
    es_manager = ESManager(environ.get("ES_HOST", "http://
        localhost:9200"))
    count_resp = es_manager.client.count(index="
        politifact_raw")
    total_count_es = count_resp.body["count"]

    page_size = 5000
    sort = [
        {"statement_date": "asc"},
        {"url": "desc"}
    ]
    resp = es_manager.client.search(
        index="politifact_raw",
        size=page_size,
        sort=sort
    )
    stored_urls = [hit["_source"]["url"] for hit in resp.
        body["hits"]["hits"]]

    while len(stored_urls) < total_count_es:
        sort_values = [
            resp['hits']['hits'][-1]['sort'][0],
            resp['hits']['hits'][-1]['sort'][1]
        ]
        resp = es_manager.client.search(
            index="politifact_raw",
            size=page_size,
            sort=sort,
            search_after=sort_values
        )
        stored_urls = (
            stored_urls
            + [hit["_source"]["url"] for hit in resp.body["
                hits"]["hits"]]
```

```
        )

    result = list(set(urls) - set(stored_urls))
    print(f"Number of articles to scrap: {len(result)}")
    return result
```

- **airflow-fakebert/dags/politifact/___init___.py**
- **airflow-fakebert/dags/elastic_search/base.py**

```
from typing import Any, List

from elastic_transport import ObjectApiResponse
from elasticsearch import Elasticsearch


class ESManager:
    def __init__(self, host: str):
        self.__client = Elasticsearch(host)

    def create_index(self, index_name: str, mapping: dict)
      -> None:
        print(f"Creating index: {index_name}")
        resp = self.__client.indices.create(index=index_name
           , body=mapping, ignore=400)
        print(f"Response: {resp}")

    def delete_index(self, index_name: str) -> None:
        print(f"Deleting index: {index_name}")
        resp = self.__client.indices.delete(index=index_name
           )
        print(f"Response: {resp}")

    def populate_index(self, index_name: str, document: dict
       , id: str | None = None) -> None:
        print(f"Populating index: {index_name}")
        resp = self.__client.index(index=index_name,
           document=document, id=id)
        print(f"Response: {resp}")
```

```
    def delete_from_index(self, index_name: str, id: str) ->
        None:
        print(f"Deleting document #{id} from index `{
            index_name}`")
        resp = self.__client.delete(index=index_name, id=id)
        print(f"Response: {resp}")

    def query_index(self, index_name: str, query: dict) ->
        ObjectApiResponse[Any]:
        print(f"Querying index: {index_name}")
        return self.__client.search(index=index_name, query=
            query)

    def refresh_index(self, index: str | List[str] | None =
        None):
        index = index or '_all'
        print(f"Refreshing indices: {index}")
        resp = self.__client.indices.refresh(index=index,
            allow_no_indices=True)
        print(f"Response: {resp}")

    @property
    def client(self) -> Elasticsearch:
        return self.__client
```

- **airflow-fakebert/dags/elastic_search/___init___.py**

BANCO DE DADOS (elasticsearch-fakebert):

- **elasticsearch-fakebert/docker-compose.yml**

```
version: '2.2'
services:
  es01:
    image: docker.elastic.co/elasticsearch/elasticsearch
        :7.17.9
    container_name: es01
    environment:
      - node.name=es01
      - cluster.name=es-docker-cluster
      - discovery.seed_hosts=es02,es03
```

```
      - cluster.initial_master_nodes=es01,es02,es03
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - data01:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
    networks:
      - elastic
  es02:
    image: docker.elastic.co/elasticsearch/elasticsearch
      :7.17.9
    container_name: es02
    environment:
      - node.name=es02
      - cluster.name=es-docker-cluster
      - discovery.seed_hosts=es01,es03
      - cluster.initial_master_nodes=es01,es02,es03
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - data02:/usr/share/elasticsearch/data
    networks:
      - elastic
  es03:
    image: docker.elastic.co/elasticsearch/elasticsearch
      :7.17.9
    container_name: es03
    environment:
      - node.name=es03
      - cluster.name=es-docker-cluster
```

```
      - discovery.seed_hosts=es01,es02
      - cluster.initial_master_nodes=es01,es02,es03
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    ulimits:
      memlock:
        soft: -1
        hard: -1
    volumes:
      - data03:/usr/share/elasticsearch/data
    networks:
      - elastic

volumes:
  data01:
    driver: local
  data02:
    driver: local
  data03:
    driver: local

networks:
  elastic:
    driver: bridge
```

API BACK-END (api-fakebert):

- **api-fakebert/pyproject.toml**

```
[tool.poetry]
name = "api-fakebert"
version = "0.1.0"
description = ""
authors = ["nevesgmd <gmedeirosneves@gmail.com>"]
readme = "README.md"
packages = [{include = "api_fakebert"}]

[tool.poetry.dependencies]
python = "^3.10"
fastapi = "^0.95.0"
elasticsearch = {version = "^8.6.2", extras = ["async"]}
```

```
uvicorn = "^0.21.1"
python-dotenv = "^1.0.0"
sentence-transformers = "^2.2.2"

[tool.poetry.group.test.dependencies]
flake8 = "*"
mypy = "*"
taskipy = "*"
isort = "*"
black = "*"

[tool.taskipy.tasks]
fmt = "isort . && black ."
lint = "flake8"
check_types = "mypy ."

[tool.isort]
profile = "black"
multi_line_output = 3
lines_between_sections = 0

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

- **api-fakebert/Dockerfile**

```
FROM python:3.10.1-slim-buster

ENV POETRY_VERSION=1.4.1

RUN apt-get update && apt-get install -y build-essential
RUN pip install "poetry==$POETRY_VERSION"

WORKDIR /app

COPY pyproject.toml poetry.lock /app/

RUN poetry install --no-root --no-interaction
```

```
COPY . /app

ENV PYTHONPATH=/app

EXPOSE 8000

CMD ["poetry", "run", "uvicorn", "app.main:app", "--host",
    "0.0.0.0", "--port", "8000"]
```

- **api-fakebert/docker-compose.yml**

```
version: "3.9"
services:
  app:
    build: .
    ports:
      - "8000:8000"
    env_file:
      - .env
```

- **api-fakebert/.flake8**

```
[flake8]
import_order_style = pep8
max-line-length = 88
```

- **api-fakebert/app/main.py**

```
import uvicorn
from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware
from app.routers import search

app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
app.include_router(search.router)
```

```
@app.get("/")
async def root() -> dict[str, str]:
    return {"message": "Hello World!"}


if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

- **api-fakebert/app/___init___.py**
- **api-fakebert/app/services/nli_model.py**

```
from enum import Enum
from os import environ
import numpy
from sentence_transformers.cross_encoder import CrossEncoder
    # type: ignore

MAX_BERT_TOKENS = 512
AVG_TOKEN_LEN = 4
SPACE_PER_TOKEN = 1
MAX_BERT_CHARS = (AVG_TOKEN_LEN + SPACE_PER_TOKEN) *
   MAX_BERT_TOKENS


class PredictionLabel(Enum):
    CONTRADICTION = "contradiction"
    ENTAILMENT = "entailment"
    NEUTRAL = "neutral"


class NLIModel:
    def __init__(self):
        self.__model = CrossEncoder(environ.get("
           NLI_MODEL_PATH", "nli_model"))

    def predict(self, input_search: str, article_text: str):
        pair = [[input_search, self.
           _get_relevant_article_section(article_text)]]
```

```
            scores = self.__model.predict(pair)
            return self._get_predict_label(scores)

        @staticmethod
        def _get_relevant_article_section(article_text: str) ->
            str:
            return article_text[-MAX_BERT_CHARS:]

        @staticmethod
        def _get_predict_label(scores: numpy.ndarray) ->
            PredictionLabel:
            label_mapping = [
                PredictionLabel.ENTAILMENT,
                PredictionLabel.NEUTRAL,
                PredictionLabel.CONTRADICTION,
            ]
            label_idx = scores.argmax(axis=1)[0].item()
            return label_mapping[label_idx]
```

- **api-fakebert/app/services/__init__.py**
- **api-fakebert/app/routers/search.py**

```
import os
from dotenv import load_dotenv
from elasticsearch import Elasticsearch
from fastapi import APIRouter
from app.models.search import (
    ArticleResponse,
    ArticleSearchResult,
    PredictionData,
    SearchAPIResponse,
)
from app.services.nli_model import NLIModel

load_dotenv()
router = APIRouter()

client = Elasticsearch(os.getenv("ES_HOST"))
index_name = "politifact_raw"
```

```
@router.get("/search")
async def search(query: str) -> dict[str, SearchAPIResponse
    | str]:
    es_response = client.search(
        index=index_name,
        query={
            "bool": {
                "should": [
                    {"match": {"statement": {"query": query,
                        "boost": 2}}},
                    {"match": {"article_text": {"query":
                        query}}},
                ]
            }
        },
        size=1,
    )
    if not (
        es_search_results := [
            ArticleSearchResult(**hit["_source"]) for hit in
                es_response["hits"]["hits"]
        ]
    ):
        return {"result": "No matches found."}

    relevant_article = es_search_results[0]
    prediction_label = NLIModel().predict(query,
        relevant_article.article_text)

    response = SearchAPIResponse(
        query=query,
        article=ArticleResponse.from_search_result(
            relevant_article),
        prediction=PredictionData(label=prediction_label),
    )
    return {"result": response}
```

- **api-fakebert/app/routers/__init__.py**

- **api-fakebert/app/models/search.py**

```
from __future__ import annotations
from pydantic import BaseModel
from app.services.nli_model import PredictionLabel


class ArticleSearchResult(BaseModel):
    url: str
    author: str
    statement: str
    statement_date: str
    label: str
    article_text: str


class ArticleResponse(BaseModel):
    url: str
    author: str
    statement: str
    statement_date: str
    label: str

    @classmethod
    def from_search_result(cls, result: ArticleSearchResult)
        -> ArticleResponse:
        return ArticleResponse(
            url=result.url,
            author=result.author,
            statement=result.statement,
            statement_date=result.statement_date,
            label=result.label,
        )


class PredictionData(BaseModel):
    label: PredictionLabel


class SearchAPIResponse(BaseModel):
```

```
    query: str
    article: ArticleResponse
    prediction: PredictionData
```

- **api-fakebert/app/models/___init___.py**

CLASSIFICADOR BERT (experiments-fakebert):

- **experiments-fakebert/pyproject.toml**

```
[tool.poetry]
name = "bert-news-similarity"
version = "0.1.0"
description = ""
authors = ["nevesgmd <gmedeirosneves@gmail.com>"]

[tool.poetry.dependencies]
python = ">=3.10,<3.11"
numpy = "^1.23.2"
pandas = "^1.4.4"
transformers = "^4.21.2"
jupyter = "^1.0.0"
sentence-transformers = "^2.2.2"
httpx = "^0.23.0"
seaborn = "^0.12.1"
opendatasets = "^0.1.22"
loguru = "^0.6.0"
pydantic = "^1.10.2"
arrow = "^1.2.3"
scipy = "^1.9.3"
datasets = "^2.10.1"
elasticsearch = {version = "^8.7.0", extras = ["async"]}
sentencepiece = "^0.1.99"
wordcloud = "^1.9.2"

[tool.poetry.dev-dependencies]
black = "*"
flake8 = "*"
flake8-bugbear = "*"
flake8-import-order = "*"
isort = "*"
```

```
taskipy = "*"

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

[tool.taskipy.tasks]
lint = "flake8"
check_imports = "isort --check ."
check_fmt = "black --check ."
fmt = "black . && isort ."
check_types = "(npm list -g pyright@1.1.245 || npm install -
    g pyright@1.1.245) && pyright"
check = "task lint && task check_imports && task check_fmt
    && task check_types"

[tool.isort]
profile = "black"
multi_line_output = 3
```

- **experiments-fakebert/pyproject.toml**

```
[tool.poetry]
name = "experiments-fakebert"
version = "0.1.0"
description = ""
authors = ["nevesgmd <gmedeirosneves@gmail.com>"]

[tool.poetry.dependencies]
python = ">=3.10,<3.11"
numpy = "^1.23.2"
pandas = "^1.4.4"
transformers = "^4.21.2"
jupyter = "^1.0.0"
sentence-transformers = "^2.2.2"
httpx = "^0.23.0"
seaborn = "^0.12.1"
opendatasets = "^0.1.22"
loguru = "^0.6.0"
pydantic = "^1.10.2"
```

```
arrow = "^1.2.3"
scipy = "^1.9.3"
datasets = "^2.10.1"
elasticsearch = {version = "^8.7.0", extras = ["async"]}
sentencepiece = "^0.1.99"
wordcloud = "^1.9.2"

[tool.poetry.dev-dependencies]
black = "*"
flake8 = "*"
flake8-bugbear = "*"
flake8-import-order = "*"
isort = "*"
taskipy = "*"

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

[tool.taskipy.tasks]
lint = "flake8"
check_imports = "isort --check ."
check_fmt = "black --check ."
fmt = "black . && isort ."
check_types = "(npm list -g pyright@1.1.245 || npm install -
   g pyright@1.1.245) && pyright"
check = "task lint && task check_imports && task check_fmt
   && task check_types"

[tool.isort]
profile = "black"
multi_line_output = 3
```

- **experiments-fakebert/___init___.py**
- **experiments-fakebert/exploration/df_prep_from_elastic.ipynb**

```
#!/usr/bin/env python
# coding: utf-8

## Collecting data from ElasticSearch
```

```
# In[1]:


from typing import Any, List

from elastic_transport import ObjectApiResponse
from elasticsearch import Elasticsearch


class ESManager:
    def __init__(self, host: str):
        self.__client = Elasticsearch(host)

    def create_index(self, index_name: str, mapping: dict)
      -> None:
        print(f"Creating index: {index_name}")
        resp = self.__client.indices.create(index=index_name
           , body=mapping, ignore=400)
        print(f"Response: {resp}")

    def delete_index(self, index_name: str) -> None:
        print(f"Deleting index: {index_name}")
        resp = self.__client.indices.delete(index=index_name
           )
        print(f"Response: {resp}")

    def populate_index(self, index_name: str, document: dict
       , id: str | None = None) -> None:
        print(f"Populating index: {index_name}")
        resp = self.__client.index(index=index_name,
           document=document, id=id)
        print(f"Response: {resp}")

    def delete_from_index(self, index_name: str, id: str) ->
        None:
        print(f"Deleting document #{id} from index `{
           index_name}`")
        resp = self.__client.delete(index=index_name, id=id)
```

```
            print(f"Response: {resp}")

        def query_index(self, index_name: str, query: dict) ->
            ObjectApiResponse[Any]:
            print(f"Querying index: {index_name}")
            return self.__client.search(index=index_name, query=
                query)

        def refresh_index(self, index: str | List[str] | None =
            None):
            index = index or '_all'
            print(f"Refreshing indices: {index}")
            resp = self.__client.indices.refresh(index=index,
                allow_no_indices=True)
            print(f"Response: {resp}")

        @property
        def client(self) -> Elasticsearch:
            return self.__client


    # In[2]:

    from os import environ

    es_manager = ESManager(environ.get("ES_HOST", "http://
        localhost:9200"))
    count_resp = es_manager.client.count(index="politifact_raw")
    total_count_es = count_resp.body["count"]
    print(f"Total records: {total_count_es}")

    page_size = 5000
    sort = [
        {"statement_date": "asc"},
        {"url": "desc"}
    ]
    resp = es_manager.client.search(
        index="politifact_raw",
        size=page_size,
```

```
    sort=sort
)

articles = [
    {
        "label": hit["_source"]["label"],
        "statement": hit["_source"]["statement"],
        "article_text": hit["_source"]["article_text"]
    }
    for hit
    in resp.body["hits"]["hits"]
]

while len(articles) < total_count_es:
    sort_values = [
        resp['hits']['hits'][-1]['sort'][0],
        resp['hits']['hits'][-1]['sort'][1]
    ]
    resp = es_manager.client.search(
        index="politifact_raw",
        size=page_size,
        sort=sort,
        search_after=sort_values
    )
    articles = (
        articles
        + [
            {
                "label": hit["_source"]["label"],
                "statement": hit["_source"]["statement"],
                "article_text": hit["_source"]["article_text
                    "]
            }
            for hit
            in resp.body["hits"]["hits"]
        ]
    )

print(f"Fetched records: {len(articles)}")
```

```
# In[3]:


import pandas as pd

df = pd.DataFrame.from_records(articles)
df.head()


# In[4]:


df.label.value_counts()


### Data wrangling

#### Converting labels

# In[5]:


labels_to_int = {
    "true": 0,
    "mostly-true": 0,
    "half-true": 1,
    "barely-true": 2,
    "false": 2,
    "pants-fire": 2,
}
df["label"] = df['label'].map(labels_to_int)
df.head()


#### Balancing DataFrame based on label count

# In[8]:
```

```
df.label.value_counts()


# In[9]:


sample_size = min(df.label.value_counts())
sample_size


# In[10]:


df_label_0 = df[df['label'] == 0]
df_label_1 = df[df['label'] == 1]
df_label_2 = df[df['label'] == 2]

df_label_0_sampled = df_label_0.sample(n=sample_size,
    random_state=1)
df_label_1_sampled = df_label_1.sample(n=sample_size,
    random_state=1)
df_label_2_sampled = df_label_2.sample(n=sample_size,
    random_state=1)

df_balanced = pd.concat([df_label_0_sampled,
    df_label_1_sampled, df_label_2_sampled])
df_balanced.head()


### Preparing a separate DataFrame with articles
    paragraphs

# In[11]:


paragraphs_balanced_df = df_balanced.copy()
paragraphs_balanced_df.head()
```

```
# In[15]:


paragraphs_balanced_df["paragraphs"] =
    paragraphs_balanced_df.article_text.str.split("\n")
paragraphs_balanced_df.head()


# In[16]:


# Removing empty paragraphs
paragraphs_balanced_df["paragraphs"] = (
    paragraphs_balanced_df["paragraphs"].apply(lambda
        paragraph_list: [p for p in paragraph_list if p.strip
        ()])
)
paragraphs_balanced_df.head()


# In[18]:


# Dropping unnecessary `article_text` column
paragraphs_balanced_df.drop('article_text', axis=1, inplace=
    True)
paragraphs_balanced_df.head()


### Saving DataFrames

# In[26]:


df_balanced.to_csv("politifact_articles.csv", index=False)

# Use JSON for paragraphs to better handle lists
```

```
paragraphs_balanced_df.to_json("politifact_paragraphs.json",
    orient="records")
```

- **experiments-fakebert/exploration/dataset_exploration.ipynb**

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd

df = pd.read_csv("politifact_articles.csv")
df.head()


# In[2]:


df.shape


# In[23]:


df.statement.value_counts()


# In[25]:


df["statement_len"] = df["statement"].str.len()
df["article_text_len"] = df["article_text"].str.len()
df.head()


# In[28]:
```

```
df[df["article_text_len"] == 4977.50]


# In[26]:


df.describe()


# In[14]:


from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt


# In[15]:


def random_color_func(word=None, font_size=None, position=
    None, orientation=None, font_path=None, random_state=
    None):
     h = int(360.0 * 21.0 / 255.0)
     s = int(100.0 * 255.0 / 255.0)
     l = int(100.0 * float(random_state.randint(60, 120)) /
        255.0)

     return "hsl({}, {}%, {}%)".format(h, s, l)


# In[22]:


stopwords = set(STOPWORDS)
stopwords.add("t")
stopwords.add("ki")

wordcloud = WordCloud(
     background_color='white',
```

```
        stopwords=stopwords,
        max_words=1000,
        max_font_size=30,
        random_state=42
    ).generate(str(df.statement))

    plt.figure(figsize=(8, 6))
    plt.imshow(wordcloud.recolor(color_func=random_color_func,
        random_state=42), interpolation="bilinear")
    plt.axis('off')
    plt.show()
```

- **experiments-fakebert/exploration/paragraphs_all_mpnet-base-v2.ipynb**

```
    #!/usr/bin/env python
    # coding: utf-8

    # ## Loading DataFrame

    # In[2]:


    import pandas as pd


    # In[34]:


    df = pd.read_json("politifact_paragraphs.json")
    df.head()


    # ## Encoding articles data with all-mpnet-base-v2 model

    # In[35]:


    import time
    from sentence_transformers import SentenceTransformer
```

```
# In[36]:


model = SentenceTransformer("all-mpnet-base-v2")


### Encoding statements

# In[38]:


print("Starting statements encoding")
start = time.time()

statements_list = df["statement"].tolist()
encoded_statements = model.encode(statements_list)

end = time.time()
print(f"It took {end - start} seconds to finish statements
    encoding")


### Encoding paragraphs

#### Encoding all paragraphs first (faster)

# In[41]:


import itertools

paragraphs_list = df["paragraphs"].tolist()
flattened_list = list(itertools.chain.from_iterable(
    paragraphs_list))
print(f"Paragraphs to encode: {len(flattened_list)}")

print("Starting paragraphs encoding")
```

```
start = time.time()
flat_encoded_paragraphs = model.encode(flattened_list)
end = time.time()
print(f"It took {end - start} seconds to finish statements
    encoding")


### Saving the encoded data

#### Joining the encoded data in a list of dicts

# In[45]:


start = time.time()

encoded_articles = []
current_p_index = 0
for i, row in df.iterrows():
    num_paragraphs = len(row["paragraphs"])

    encoded_articles.append(
        {
            "encoded_statement": encoded_statements[i],
            "encoded_paragraphs": flat_encoded_paragraphs[
                current_p_index:current_p_index +
                num_paragraphs],
            "factchecker_label": row["label"]
        }
    )
    current_p_index += num_paragraphs

end = time.time()
print(f"It took {end - start} seconds to join the encoded
    data in a list of dicts")


# In[50]:
```

```
print(len(encoded_articles))
print("-"*10)
print(len(encoded_articles[0]["encoded_paragraphs"]))
print(len(encoded_articles[1]["encoded_paragraphs"]))
print(len(encoded_articles[2]["encoded_paragraphs"]))
print("-"*10)
print(len(df.iloc[0]["paragraphs"]))
print(len(df.iloc[1]["paragraphs"]))
print(len(df.iloc[2]["paragraphs"]))


# In[47]:


import pickle

with open('all_mpnet_base_v2_paragraphs_embeddings.pkl', "wb
    ") as file:
     pickle.dump(encoded_articles, file, protocol=pickle.
        HIGHEST_PROTOCOL)
```

- **experiments-fakebert/exploration/all-mpnet-base-v2-analysis.ipynb**

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


import pickle
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
import sys
from statistics import median
from time import time
```

```
from scipy.spatial.distance import cityblock, euclidean,
    minkowski, jaccard

from sentence_transformers import util
from pydantic import BaseModel

module_path = os.path.abspath(os.path.join('..'))
if module_path not in sys.path:
    sys.path.append(module_path)
from src.politifact_news_data.transform.models import
    get_articles_from_file, ArticleLabel


### Loading article data and embeddings

# In[3]:


with open("all_mpnet_base_v2_paragraphs_embeddings.pkl", "rb
    ") as file:
    embeddings = pickle.load(file)


# In[6]:


df = pd.read_json("politifact_paragraphs.json")
df.head()


### Computing similarity values

# In[12]:


from enum import Enum


class SimilarityFunction(Enum):
```

```
    COS_SIM = "cos_sim"
    EUCLIDEAN = "euclidean"
    MANHATTAN = "manhattan"
    MINKOWSKI = "minkowski"



def get_df_entries(
    df: pd.DataFrame,
    embeddings: list[dict],
    function: SimilarityFunction = SimilarityFunction.
       COS_SIM
) -> list[dict]:
    result: list[dict] = []

    start = time()
    for i, row in df.iterrows():
        statement_embedding = embeddings[i]["
           encoded_statement"]
        paragraphs_embeddings = embeddings[i]["
           encoded_paragraphs"]

        similarity_values = []
        for p_embedding in paragraphs_embeddings:
            if function == SimilarityFunction.COS_SIM:
                sim_value = float(util.cos_sim(
                   statement_embedding, p_embedding)[0][0])
            elif function == SimilarityFunction.EUCLIDEAN:
                sim_value = euclidean(statement_embedding,
                   p_embedding)
            elif function == SimilarityFunction.MANHATTAN:
                sim_value = cityblock(statement_embedding,
                   p_embedding)
            elif function == SimilarityFunction.MINKOWSKI:
                sim_value = minkowski(statement_embedding,
                   p_embedding)
            similarity_values.append(sim_value)

        result.append(
```

```
                {
                    "label": row["label"],
                    "max_similarity_value": max(
                        similarity_values),
                    "min_similarity_value": min(
                        similarity_values),
                    "avg_similarity_value": sum(
                        similarity_values) / len(
                        similarity_values),
                    "median_similarity_value": median(
                        similarity_values),
                }
            )
        end = time()
        print(f"It took {end - start} seconds to get {len(
            embeddings)} DF entries with function `{function}`")

        return result


# In[13]:


cos_sim_entries = get_df_entries(df, embeddings)
euclidean_entries = get_df_entries(df, embeddings,
    SimilarityFunction.EUCLIDEAN)
manhattan_entries = get_df_entries(df, embeddings,
    SimilarityFunction.MANHATTAN)
minkowski_entries = get_df_entries(df, embeddings,
    SimilarityFunction.MINKOWSKI)


# In[15]:


cos_sim_df = pd.DataFrame.from_dict(cos_sim_entries)
euclidean_df = pd.DataFrame.from_dict(euclidean_entries)
manhattan_df = pd.DataFrame.from_dict(manhattan_entries)
minkowski_df = pd.DataFrame.from_dict(minkowski_entries)
```

```
## Exploring similarities results

### Mapping categorical values

# In[21]:


dow = {
    0: "Entailment",
    1: "Neutral",
    2: "Contradiction",
}
cos_sim_df["cat_label"] = cos_sim_df["label"].map(dow)
euclidean_df["cat_label"] = euclidean_df["label"].map(dow)
manhattan_df["cat_label"] = manhattan_df["label"].map(dow)
minkowski_df["cat_label"] = minkowski_df["label"].map(dow)
cos_sim_df.head()


### Setting base color for the plots

# In[17]:


base_color = sns.color_palette()[0]


### Cosine similarity

# In[22]:


cos_sim_df.describe()


# In[50]:
```

```
plt.figure(figsize=(7, 5))
sns.boxplot(
    data=cos_sim_df,
    x='cat_label',
    y='max_similarity_value',
    order=[
        "Contradiction",
        "Neutral",
        "Entailment"
    ],
    color=base_color
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor máximo de similaridade por cosseno em cada
    artigo', size=8);
plt.savefig("img/cos_sim_box_plot_max.png", format="png",
   bbox_inches="tight")


# In[51]:


plt.figure(figsize=(7, 5))
sns.boxplot(
    data=cos_sim_df,
    x='cat_label',
    y='min_similarity_value',
    order=[
        "Contradiction",
        "Neutral",
        "Entailment"
    ],
    color=base_color
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor mínimo de similaridade por cosseno em cada
    artigo', size=8);
plt.savefig("img/cos_sim_box_plot_min.png", format="png",
```

```
    bbox_inches="tight")


# In[52]:


plt.figure(figsize=(7, 5))
sns.boxplot(
    data=cos_sim_df,
    x='cat_label',
    y='avg_similarity_value',
    order=[
        "Contradiction",
        "Neutral",
        "Entailment"
    ],
    color=base_color
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor médio de similaridade por cosseno em cada
    artigo', size=8);
plt.savefig("img/cos_sim_box_plot_avg.png", format="png",
    bbox_inches="tight")


# In[53]:


plt.figure(figsize=(7, 5))
sns.boxplot(
    data=cos_sim_df,
    x='cat_label',
    y='median_similarity_value',
    order=[
        "Contradiction",
        "Neutral",
        "Entailment"
    ],
    color=base_color
```

```
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor mediano de similaridade por cosseno em
    cada artigo', size=8);
plt.savefig("img/cos_sim_box_plot_median.png", format="png",
     bbox_inches="tight")


# In[54]:


plt.figure(figsize=(7, 5))
sns.displot(cos_sim_df, x="median_similarity_value", hue="
    cat_label", element="step")
plt.ylabel('OcorrÃªncias', size=8, labelpad=15)
plt.xlabel('Valor mediano de similaridade por cosseno por
    artigo', size=8);
plt.savefig("img/cos_sim_displot_median.png", format="png",
    bbox_inches="tight")


# In[55]:


plt.figure(figsize=(7, 5))
sns.displot(cos_sim_df, x="avg_similarity_value", hue="
    cat_label", element="step")
plt.ylabel('OcorrÃªncias', size=8, labelpad=15)
plt.xlabel('Valor médio de similaridade por cosseno por
    artigo', size=8);
plt.savefig("img/cos_sim_displot_avg.png", format="png",
    bbox_inches="tight")


# In[56]:


plt.figure(figsize=(7, 5))
sns.displot(cos_sim_df, x="min_similarity_value", hue="
```

```
    cat_label", element="step")
plt.ylabel('OcorrÃªncias', size=8, labelpad=15)
plt.xlabel('Valor mínimo de similaridade por cosseno por
    artigo', size=8);
plt.savefig("img/cos_sim_displot_min.png", format="png",
    bbox_inches="tight")


# In[57]:


plt.figure(figsize=(7, 5))
sns.displot(cos_sim_df, x="max_similarity_value", hue="
    cat_label", element="step")
plt.ylabel('OcorrÃªncias', size=8, labelpad=15)
plt.xlabel('Valor máximo de similaridade por cosseno por
    artigo', size=8);
plt.savefig("img/cos_sim_displot_max.png", format="png",
    bbox_inches="tight")


### Euclidean

# In[34]:


euclidean_df.describe()


# In[58]:


plt.figure(figsize=(7, 5))
sns.boxplot(
    data=euclidean_df,
    x='cat_label',
    y='max_similarity_value',
    order=[
        "Contradiction",
```

```
            "Neutral",
            "Entailment"
        ],
        color=base_color
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor máximo de similaridade euclideana em cada
    artigo', size=8);
plt.savefig("img/euclidean_box_plot_max.png", format="png",
    bbox_inches="tight")


# In[59]:


plt.figure(figsize=(7, 5))
sns.boxplot(
    data=euclidean_df,
    x='cat_label',
    y='min_similarity_value',
    order=[
        "Contradiction",
        "Neutral",
        "Entailment"
    ],
    color=base_color
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor mínimo de similaridade euclideana em cada
    artigo', size=8);
plt.savefig("img/euclidean_box_plot_min.png", format="png",
    bbox_inches="tight")


# In[60]:


plt.figure(figsize=(7, 5))
sns.boxplot(
```

```
    data=euclidean_df,
    x='cat_label',
    y='avg_similarity_value',
    order=[
        "Contradiction",
        "Neutral",
        "Entailment"
    ],
    color=base_color
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor médio de similaridade euclideana em cada
    artigo', size=8);
plt.savefig("img/euclidean_box_plot_avg.png", format="png",
    bbox_inches="tight")


# In[61]:


plt.figure(figsize=(7, 5))
sns.boxplot(
    data=euclidean_df,
    x='cat_label',
    y='median_similarity_value',
    order=[
        "Contradiction",
        "Neutral",
        "Entailment"
    ],
    color=base_color
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor mediano de similaridade euclideana em cada
    artigo', size=8);
plt.savefig("img/euclidean_box_plot_median.png", format="png
    ", bbox_inches="tight")
```

```
## Manhattan

# In[39]:


manhattan_df.describe()


# In[62]:


plt.figure(figsize=(7, 5))
sns.boxplot(
    data=manhattan_df,
    x='cat_label',
    y='max_similarity_value',
    order=[
        "Contradiction",
        "Neutral",
        "Entailment"
    ],
    color=base_color
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor máximo de similaridade manhattan em cada
    artigo', size=8);
plt.savefig("img/manhattan_box_plot_max.png", format="png",
    bbox_inches="tight")


# In[63]:


plt.figure(figsize=(7, 5))
sns.boxplot(
    data=manhattan_df,
    x='cat_label',
    y='min_similarity_value',
    order=[
```

```
            "Contradiction",
            "Neutral",
            "Entailment"
        ],
        color=base_color
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor mínimo de similaridade manhattan em cada
    artigo', size=8);
plt.savefig("img/manhattan_box_plot_min.png", format="png",
    bbox_inches="tight")


# In[64]:


plt.figure(figsize=(7, 5))
sns.boxplot(
    data=manhattan_df,
    x='cat_label',
    y='avg_similarity_value',
    order=[
        "Contradiction",
        "Neutral",
        "Entailment"
    ],
    color=base_color
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor médio de similaridade manhattan em cada
    artigo', size=8);
plt.savefig("img/manhattan_box_plot_avg.png", format="png",
    bbox_inches="tight")


# In[65]:


plt.figure(figsize=(7, 5))
```

```
sns.boxplot(
    data=manhattan_df,
    x='cat_label',
    y='median_similarity_value',
    order=[
        "Contradiction",
        "Neutral",
        "Entailment"
    ],
    color=base_color
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor mediano de similaridade manhattan em cada
    artigo', size=8);
plt.savefig("img/manhattan_box_plot_median.png", format="png
    ", bbox_inches="tight")


### Minkowski

# In[45]:


minkowski_df.describe()


# In[66]:


plt.figure(figsize=(7, 5))
sns.boxplot(
    data=minkowski_df,
    x='cat_label',
    y='max_similarity_value',
    order=[
        "Contradiction",
        "Neutral",
        "Entailment"
    ],
```

```
    color=base_color
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor máximo de similaridade minkowski em cada
    artigo', size=8);
plt.savefig("img/minkowski_box_plot_max.png", format="png",
    bbox_inches="tight")


# In[67]:


plt.figure(figsize=(7, 5))
sns.boxplot(
    data=minkowski_df,
    x='cat_label',
    y='min_similarity_value',
    order=[
        "Contradiction",
        "Neutral",
        "Entailment"
    ],
    color=base_color
)
plt.xlabel('Classificação', size=8, labelpad=15)
plt.ylabel('Valor mínimo de similaridade minkowski em cada
    artigo', size=8);
plt.savefig("img/minkowski_box_plot_min.png", format="png",
    bbox_inches="tight")


# In[68]:


plt.figure(figsize=(7, 5))
sns.boxplot(
    data=minkowski_df,
    x='cat_label',
    y='avg_similarity_value',
```

```
        order=[
            "Contradiction",
            "Neutral",
            "Entailment"
        ],
        color=base_color
    )
    plt.xlabel('Classificação', size=8, labelpad=15)
    plt.ylabel('Valor médio de similaridade minkowski em cada
        artigo', size=8);
    plt.savefig("img/minkowski_box_plot_avg.png", format="png",
        bbox_inches="tight")


    # In[69]:


    plt.figure(figsize=(7, 5))
    sns.boxplot(
        data=minkowski_df,
        x='cat_label',
        y='median_similarity_value',
        order=[
            "Contradiction",
            "Neutral",
            "Entailment"
        ],
        color=base_color
    )
    plt.xlabel('Classificação', size=8, labelpad=15)
    plt.ylabel('Valor mediano de similaridade minkowski em cada
        artigo', size=8);
    plt.savefig("img/minkowski_box_plot_median.png", format="png
        ", bbox_inches="tight")
```

- **experiments-fakebert/exploration/cross_encoder.ipynb**

```
    #!/usr/bin/env python
    # coding: utf-8
```

```
### Loading articles data

# In[9]:


import pandas as pd

df = pd.read_csv("politifact_articles.csv")
df.head()


### Testing raw CrossEncoder models

#### Setting test set

# In[14]:


from sklearn.model_selection import train_test_split

_, test = train_test_split(df, test_size=0.2, random_state
    =42)
test.head()


# In[15]:


from sentence_transformers.readers import InputExample


def get_input_examples(df: pd.DataFrame) -> list[
    InputExample]:
    records = df.to_dict('records')
    return [
        InputExample(
            texts=[
                record["statement"],
```

```
                record["article_text"],
            ],
            label=record["label"]
        )
        for record in records
    ]


# In[16]:


test_inputs = get_input_examples(test)
len(test_inputs)


### nli-deberta-v3-base

# In[22]:


from sentence_transformers.cross_encoder import CrossEncoder

model_deberta_v3 = CrossEncoder("cross-encoder/nli-deberta-
    v3-base", num_labels=len(df.label.unique()))


# In[24]:


from sentence_transformers.cross_encoder.evaluation import
    CESoftmaxAccuracyEvaluator

evaluator = CESoftmaxAccuracyEvaluator.from_input_examples(
    test_inputs, name='AllNLI-test')
evaluator(model_deberta_v3)


### nli-distilroberta-base
```

```
# In[23]:


from sentence_transformers.cross_encoder import CrossEncoder

model_distilroberta = CrossEncoder("cross-encoder/nli-
    distilroberta-base", num_labels=len(df.label.unique()))


# In[25]:


from sentence_transformers.cross_encoder.evaluation import
    CESoftmaxAccuracyEvaluator

evaluator = CESoftmaxAccuracyEvaluator.from_input_examples(
    test_inputs, name='AllNLI-test')
evaluator(model_distilroberta)


### Training

# In[13]:


from sentence_transformers.cross_encoder import CrossEncoder

# Defining base model (using nli-deberta-v3-base since it
    has best results above)
model = CrossEncoder("cross-encoder/nli-deberta-v3-base",
    num_labels=len(df.label.unique()))


#### Using full articles (which will be truncated due to
    BERT's max token limitation)

# In[4]:
```

```
from sklearn.model_selection import train_test_split

train, test = train_test_split(df, test_size=0.2,
    random_state=42)
train, dev = train_test_split(train, test_size=0.1,
    random_state=42)
train.head()


# In[5]:


train.shape, dev.shape, test.shape


# In[7]:


train_inputs = get_input_examples(train)
dev_inputs = get_input_examples(dev)
test_inputs = get_input_examples(test)

len(train_inputs), len(dev_inputs), len(test_inputs)


# In[9]:


from datetime import datetime

train_batch_size = 16
num_epochs = 3
model_save_path = 'output/cross-encoder-nli-deberta-v3-base
    -'+datetime.now().strftime("%Y-%m-%d_%H-%M-%S")


# In[10]:
```

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(train_inputs, shuffle=True,
    batch_size=train_batch_size)


# In[11]:


from sentence_transformers.cross_encoder.evaluation import
    CESoftmaxAccuracyEvaluator

evaluator = CESoftmaxAccuracyEvaluator.from_input_examples(
    dev_inputs, name='AllNLI-dev')


# In[12]:


import math

warmup_steps = math.ceil(len(train_dataloader) * num_epochs
    * 0.1)  # 10% of train data for warm-up
warmup_steps


# In[ ]:


model.fit(
    train_dataloader=train_dataloader,
    evaluator=evaluator,
    epochs=num_epochs,
    evaluation_steps=1000,
    warmup_steps=warmup_steps,
    output_path=model_save_path
)
```

```
# In[21]:


model = CrossEncoder(model_save_path)
evaluator = CESoftmaxAccuracyEvaluator.from_input_examples(
    test_inputs, name='AllNLI-test')
evaluator(model)


### Using a relevant article selection (cut text by N
    characters from the end)

# In[15]:


MAX_BERT_TOKENS = 512
AVG_TOKEN_LEN = 4
SPACE_PER_TOKEN = 1
MAX_BERT_CHARS = (AVG_TOKEN_LEN + SPACE_PER_TOKEN) *
    MAX_BERT_TOKENS

df.loc[df["article_text"].str.len() < MAX_BERT_CHARS]


# In[16]:


df['article_text'] = df['article_text'].str[-MAX_BERT_CHARS
    :]
df.head()


# In[17]:


df.loc[df["article_text"].str.len() < MAX_BERT_CHARS]


# In[18]:
```

```
from sklearn.model_selection import train_test_split

train, test = train_test_split(df, test_size=0.2,
    random_state=42)
train, dev = train_test_split(train, test_size=0.1,
    random_state=42)
train.head()


# In[19]:


train.shape, dev.shape, test.shape


# In[21]:


train_inputs = get_input_examples(train)
dev_inputs = get_input_examples(dev)
test_inputs = get_input_examples(test)

len(train_inputs), len(dev_inputs), len(test_inputs)


# In[22]:


from datetime import datetime

train_batch_size = 16
num_epochs = 3
model_save_path = 'output/cross-encoder-nli-deberta-v3-base-
    relevant-selection'+datetime.now().strftime("%Y-%m-%d_%H
    -%M-%S")
```

```
# In[23]:


from torch.utils.data import DataLoader

train_dataloader = DataLoader(train_inputs, shuffle=True,
    batch_size=train_batch_size)


# In[24]:


from sentence_transformers.cross_encoder.evaluation import
    CESoftmaxAccuracyEvaluator

evaluator = CESoftmaxAccuracyEvaluator.from_input_examples(
    dev_inputs, name='AllNLI-dev')


# In[25]:


import math

warmup_steps = math.ceil(len(train_dataloader) * num_epochs
    * 0.1) # 10% of train data for warm-up
warmup_steps


# In[92]:


model.fit(
    train_dataloader=train_dataloader,
    evaluator=evaluator,
    epochs=num_epochs,
    evaluation_steps=1000,
    warmup_steps=warmup_steps,
    output_path=model_save_path
```

```
)


# In[93]:


model = CrossEncoder(model_save_path)
evaluator = CESoftmaxAccuracyEvaluator.from_input_examples(
    test_inputs, name='AllNLI-test')
evaluator(model)
```

INTERFACE GRÁFICA (front-end-fakebert):

- **front-end-fakebert/vite.config.js**

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react()],
  server: {
    watch: {
      usePolling: true,
    },
    host: true,
    strictPort: true,
    port: 5173,
  }
})
```

- **front-end-fakebert/package.json**

```
{
  "name": "front-end-fakebert",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
```

```
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-icons": "^4.8.0",
    "styled-components": "^5.3.9"
  },
  "devDependencies": {
    "@types/react": "^18.0.28",
    "@types/react-dom": "^18.0.11",
    "@vitejs/plugin-react": "^3.1.0",
    "esbuild": "^0.17.14",
    "vite": "^4.2.0"
  }
}
```

- **front-end-fakebert/index.html**

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,
        initial-scale=1.0" />
    <title>FakeBERT</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

- **front-end-fakebert/Dockerfile**

```
FROM node:14-alpine
```

```
WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 5173

CMD [ "npm", "run", "dev" ]
```

- **front-end-fakebert/docker-compose.yml**

```
version: "3"
services:
  app:
    build: .
    ports:
      - "5173:5173"
```

- **front-end-fakebert/src/main.jsx**

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App'
import './index.css'

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
)
```

- **front-end-fakebert/src/index.css**

```
* {
  box-sizing: border-box;
  margin: 0;
```

```
    padding: 0;
    font-family: Helvetica, sans-serif;
  }

  img {
    display: block;
    margin-left: auto;
    margin-right: auto;
    padding-top: 50px;
    width: 300px;
    height: auto;
  }
```

- **front-end-fakebert/src/App.jsx**

```
  import { useState } from "react";

  import "./App.css";
  import { SearchBar } from "./components/SearchBar";
  import LoadingSpinner from "./components/LoadingSpinner";
  import SearchResult from "./components/SearchResult";


  function App() {
    const [result, setResult] = useState([]);
    const [loading, setLoading] = useState(false);

    return (
      <div className="App">
        <img src="../checkbert_logo.png" alt="CheckBERT Logo"
           />
        <div className="search-bar-container">
          <SearchBar setResult={setResult} setLoading={
             setLoading} />
          {loading && <LoadingSpinner duration={1000}
             borderColor="#5271FF"/>}
          {!loading && result.hasOwnProperty("prediction") &&
            <SearchResult result={result}/>}
```

```
        </div>
      </div>
    );
  }

  export default App;
```

- **front-end-fakebert/src/App.css**

```
  .App {
    background-color: #eee;
    width: 100vw;
    height: 100vh;
  }

  .search-bar-container {
    padding-top: 20px;
    width: 60%;
    margin: auto;
    display: flex;
    flex-direction: column;
    align-items: center;
    min-width: 200px;
  }
```

- **front-end-fakebert/src/components/SearchResult.jsx**

```
  import styled from 'styled-components';
  import ArticleInfo from './ArticleInfo';
  import PredictionResult from './PredictionResult';


  const SearchResultsWrapper = styled.div`
    display: flex;
    flex-direction: column;
    margin: 16px auto;
  `;

  function SearchResult({ result }) {
```

```
    return (
      <SearchResultsWrapper>
        <PredictionResult apiResult={result}/>
        <ArticleInfo articleData={result.article}/>
      </SearchResultsWrapper>
    );
  }

  export default SearchResult;
```

- **front-end-fakebert/src/components/SearchBar.jsx**

```
  import { useState } from "react";
  import { FaSearch } from "react-icons/fa";

  import "./SearchBar.css";

  export const SearchBar = ({ setResult, setLoading }) => {
    const [input, setInput] = useState("");

    const fetchData = (value) => {
      try {
        setLoading(true);
        // TODO: Replace localhost by the API URL if released
        fetch(`http://localhost:8000/search?query=${
           encodeURIComponent(value)}`)
        .then((response) => response.json())
        .then(
          (json) => {
            setResult(json.result)
            setLoading(false)
          }
        )
      } catch (error) {
        setLoading(false);
        console.error(error);
      }
    };
```

```
    return (
      <div className="input-wrapper">
        <FaSearch id="search-icon" />
        <input
          spellcheck="false"
          placeholder="Search here..."
          value={input}
          onChange={(e) => setInput(e.target.value)}
          onKeyDown={(event) => {
            if (event.key === 'Enter') {
              console.log('User input:', input);
              fetchData(event.target.value)
            }
          }}
        />
      </div>
    );
  };
```

- **front-end-fakebert/src/components/SearchBar.css**

```
.input-wrapper {
  width: 100%;
  height: 2.5rem;
  border: none;
  border-radius: 10px;
  padding: 0 15px;
  box-shadow: 0px 0px 8px #ddd;
  background-color: white;
  display: flex;
  align-items: center;
}

input {
  background-color: transparent;
  border: none;
  height: 100%;
  font-size: 1.25rem;
  width: 100%;
```

```
  margin-left: 5px;
}

input:focus {
  outline: none;
}

#search-icon {
  color: royalblue;
}
```

- **front-end-fakebert/src/components/PredictionResult.jsx**

```
import React from "react";
import styled from "styled-components";

const PredictionResultWrapper = styled.div`
  text-align: center;
  width: 100%;
  height: 45px;
  font-size: 14px;
  color: #2b2d42;
  line-height: 20pt;
  box-sizing: border-box;
`;

const Query = styled.span`
  font-weight: bold;
  color: #2b2d42;
`;

const Label = styled.span`
  text-transform: uppercase;
  color: #2b2d42;
`;

const PredictionResult = ({ apiResult }) => {
  let labelColor;
```

```
    switch (apiResult.prediction.label) {
      case 'entailment':
        labelColor = '#28A745'; // green
        break;
      case 'neutral':
        labelColor = '#FFC107'; // yellow
        break;
      case 'contradiction':
        labelColor = '#e61919'; // red
        break;
      default:
        labelColor = '#000'; // black
    }

    return (
      <PredictionResultWrapper>
          CheckBERT labeled the sentence <Query>"{apiResult.
             query}"</Query> with a <Label style={{ color:
             labelColor }}>{apiResult.prediction.label}</Label
             > relationship based on the following article
             text:
      </PredictionResultWrapper>
    );
  };

  export default PredictionResult;
```

- **front-end-fakebert/src/components/LoadingSpinner.jsx**

```
  import React from 'react';
  import styled, { keyframes } from 'styled-components';

  const SpinnerWrapper = styled.div`
    display: flex;
    flex-direction: row;
    align-items: center;
    justify-content: center;
    grid-gap: 1rem;
    margin-top: 2rem;
```

```
';

const spinnerAnimation = keyframes`
from {
        transform: rotate(0deg);
    }
    to {
        transform: rotate(360deg);
    }
`;

const SpinnerBody = styled.div`
  height: ${(props) => (props.height ? props.height : '4rem
     ')};
  width: ${(props) => (props.width ? props.width : '4rem')};
  border: ${(props) => (props.borderWidth ? props.
     borderWidth : '4px')} solid #d1d5db;
  border-top-color: ${(props) => (props.borderColor ? props.
     borderColor : '#3b82f6')};
  border-radius: 50%;
  animation: ${spinnerAnimation}
    ${(props) => (props.duration ? `${props.duration}ms` :
       '800ms')} linear infinite;
`;

const LoadingSpinner = (props) => {
  return (
    <div className="LoadingSpinner">
        <SpinnerWrapper>
            <SpinnerBody {...props} />
        </SpinnerWrapper>
    </div>
  )
};

export default LoadingSpinner;
```

- **front-end-fakebert/src/components/ArticleInfo.jsx**

```
import React from "react";
import styled from "styled-components";

const ArticleInfoWrapper = styled.div`
  display: flex;
  flex-direction: column;
  justify-content: space-between;
  align-items: center;
  width: 100%;
  height: 160px;
  padding: 10px;
  border-radius: 10px;
  background-color: white;
  margin: 0 0 10px 0;
`;

const Label = styled.span`
  font-size: 12px;
  font-weight: bold;
  text-transform: uppercase;
  color: #2b2d42;
`;

const Statement = styled.div`
  text-align: center;
  margin-top: 10px;
  font-size: 18px;
  font-weight: bold;
  color: #2b2d42;
`;

const Author = styled.div`
  margin-top: 5px;
  font-size: 12px;
  color: #2b2d42;
`;


const StatementDate = styled.div`
```

```
  margin-top: 5px;
  font-size: 12px;
  color: #2b2d42;
`;

const Url = styled.a`
  margin-top: 10px;
  font-size: 12px;
  color: blue;
  text-decoration: none;
  text-align: center;
`;

const ArticleInfo = ({ articleData }) => {
  let labelColor;

  switch (articleData.label) {
    case 'true':
      labelColor = '#28A745'; // green
      break;
    case 'mostly-true':
      labelColor = '#DFEA26'; // mix between green and
         yellow
      break;
    case 'half-true':
      labelColor = '#FFC107'; // yellow
      break;
    case 'barely-true':
      labelColor = '#FF8C00'; // orange
      break;
    case 'false':
      labelColor = '#e61919'; // red
      break;
    case 'pants-fire':
      labelColor = '#f00'; // bright red
      break;
    default:
      labelColor = '#000'; // black
  }
```

```
    return (
      <ArticleInfoWrapper>
        <Label style={{ color: labelColor }}>{articleData.
            label}</Label>
        <Statement>{articleData.statement}</Statement>
        <Author>By: {articleData.author}</Author>
        <StatementDate>Date: {articleData.statement_date.
            substring(0, 10)}</StatementDate>
        <Url href={articleData.url}>{articleData.url}</Url>
      </ArticleInfoWrapper>
    );
  };

  export default ArticleInfo;
```

APÊNDICE B – ARTIGO SBC

CheckBERT: Um Sistema de Identificação de Informações Falsas Através da Comparação Semântica de Sentenças com Notícias de Portais Jornalísticos de Língua Inglesa Utilizando Google BERT

Gabriel Medeiros das Neves[1], **Carina Friedrich Dorneles**[2]

[1]Universidade Federal de Santa Catarina (UFSC)
Departamento de Informática e Estatística (INE) – Florianópolis – SC – Brazil

g.neves@grad.ufsc.br, carina.dorneles@ufsc.br

Abstract. *A survey conducted by Kaspersky in partnership with CORPA between December 2019 and January 2020 revealed that, on average, 62% of Brazilians are unable to detect fake news on the internet. A more recent study by IDEIA and Vero institutes, from July 2021, found that 79% of surveyed Brazilians consider the spread of fake news a serious problem. To address this issue, this study proposes the implementation of a system that can identify false information in textual sentences by comparing them with the articles content from USA news portals. This comparison is made by analyzing the semantic similarity between the provided sentences and previously collected news articles using Google BERT.*

Resumo. *Uma pesquisa realizada pela Kaspersky em parceria com a CORPA revelou que, em média, 62% dos brasileiros são incapazes de detectar notícias falsas na internet. Um estudo mais recente realizado pelos institutos IDEIA e Vero, em julho de 2021, constatou que 79% dos brasileiros entrevistados consideram a disseminação de notícias falsas um problema grave. Para enfrentar essa problemática, este estudo propõe a implementação de um sistema capaz de identificar informações falsas em frases textuais através da sua comparação com o conteúdo de artigos de portais de notícias dos EUA. Processo este que é realizado através da análise de similaridade semântica entre as frases fornecidas e os artigos de notícias previamente coletados, utilizando o Google BERT.*

1. Introdução

A propagação de informações falsas não é exclusividade do século XXI, ainda que o tema seja extremamente relevante nos tempos atuais. Ao longo da história, diversos episódios de rumores falsos acabaram alterando o rumo das esferas políticas e sociais da época, desde o suicídio do general romano Marco Antônio, ao receber a falsa notícia do falecimento de sua amante Cleópatra, até a divulgação massiva de notícias falsas durante eventos marcantes da última década, como as eleições de 2016 nos Estados Unidos e a saída do Reino Unido da União Europeia [Rubio Hancock, Jaime]. Contudo, conforme o acesso e o compartilhamento de informações foi sendo progressivamente facilitado pelas inovações digitais, a disseminação de notícias falsas também tornou-se mais recorrente, ao ponto que, em 2017, a expressão "fake news", utilizada para referir-se a notícias falsas, foi eleita como expressão do ano pelo dicionário Collins, segundo a matéria publicada pela revista [Veja].

Entendendo o crescimento e a notoriedade do tema no meio digital, algumas organizações construíram ferramentas para auxiliar a população a identificar a veracidade dos fatos que lhe são apresentados. Dentre essas ferramentas, podemos citar a FactCheck.org[1], o Washington Post Fact Checker[2] e o Politifact[3]. Além disso, no Brasil, podemos verificar a autenticidade de fatos por portais como Fato ou Fake[4], Comprova[5], Lupa[6] e outros. Entretanto, o funcionamento de todas essas ferramentas depende da investigação por parte de uma equipe de jornalistas, que então publicam seu parecer sobre a exatidão dos fatos e apresentam as fontes e referências utilizadas. Todavia, esse processo frequentemente não consegue acompanhar a velocidade da comunicação na internet e, juntamente com interfaces nem sempre amigáveis ao usuário final e a impraticidade de verificar a precisão dos fatos em diferentes portais, acaba resultando na incerteza, ou ainda na aceitação, de informações incorretas ou falsas.

À vista disso, o presente trabalho visa facilitar o reconhecimento de informações falsas através da comparação com notícias publicadas nos principais portais de notícias de língua inglesa, bem como em algumas das ferramentas identificadas acima. Esse processo de identificação ocorre sem intervenções manuais, fazendo uso de um modelo contextualizado de NLP chamado BERT, que utiliza *Deep Learning* para compreender o significado das palavras em um texto da mesma maneira que humanos o fazem.

Ao final do projeto, espera-se que o software desenvolvido seja capaz de assimilar a sentença fornecida pelo usuário, identificar artigos relacionados e retornar a classificação obtida pelo modelo BERT, juntamente com o URL do artigo utilizado para classificação e outras informações relevantes, como sua data de publicação e título, por exemplo.

1.1. Objetivo

No que diz respeito ao objetivo geral do trabalho, é proposta a criação de um software capaz de identificar se uma sentença fornecida como entrada pelo usuário final da ferramenta é falsa. A identificação se dá através da comparação semântica do conteúdo textual de artigos de portais de notícias de língua inglesa. Para isso, é utilizado um modelo de NLP chamado Google BERT, com a finalidade de identificar contexto e significado nos textos jornalísticos, e assim possibilitar a classificação do relacionamento entre a sentença fornecida pelo usuário e o conteúdo dos artigos, o qual pode ser classificado como concordante, discordante ou neutro.

2. Trabalhos Relacionados

2.1. FakeBERT

O artigo de [Kaliyar et al. 2021] propõe a utilização de uma abordagem de aprendizado profundo baseada em arquiteturas BERT denominada FakeBERT, que visa identificar informações falsas em trechos de publicações em redes sociais. Essa abordagem mescla

[1]https://www.factcheck.org/
[2]https://www.washingtonpost.com/news/fact-checker/
[3]https://www.politifact.com/
[4]https://g1.globo.com/fato-ou-fake/
[5]https://projetocomprova.com.br/
[6]https://lupa.uol.com.br/

a utilização de redes neurais convolucionais profundas e a arquitetura bidirecional do BERT, com o intuito de minimizar problemas de ambiguidade que, segundo os autores, são o maior desafio durante o entendimento de modelos de processamento de linguagem natural. Quanto ao treinamento e testagem do modelo, foram realizados experimentos em um conjunto de dados de notícias publicadas durante o período eleitoral de 2016 nos Estados Unidos, contando tanto com notícias verdadeiras quanto falsas. A testagem comparou o FakeBERT com outros modelos que utilizam redes neurais convolucionais e redes neurais de memória de longo e curto prazo, e os resultados apresentaram uma vantagem notória para o FakeBERT, que alcançou uma acurácia de 98%.

2.2. COVID-Twitter-BERT

Tendo em vista a recente pandemia do coronavírus e seus desdobramentos, bem como a relevância do Twitter na disseminação de notícias e no entendimento da percepção da sociedade sobre determinado tema, o artigo *"COVID-Twitter-BERT: A Natural Language Processing Model to Analyse COVID-19 Content on Twitter"*, de [Müller et al. 2020], visa o desenvolvimento de um modelo de processamento de linguagem natural a partir do enriquecimento do modelo existente BERT-Large, que utiliza a arquitetura BERT e foi pré-treinado com dados da Wikipédia. Para isso, foram extraídos mais de 160 milhões de tweets referentes ao coronavírus, dos quais, após tratamento e filtragem, foram selecionados cerca de 22,5 milhões para o treinamento e testagem do modelo. Ao final, o modelo desenvolvido alcançou seu objetivo em sua principal atribuição - auxiliar tarefas de classificação semântica dentro do contexto do COVID-19, como a identificação de sentimento em uma sentença ou a classificação do conteúdo textual como pessoal ou jornalístico - e foi capaz de obter resultados superiores quando comparado ou modelo pré-treinado BERT-Large, que foi utilizado como base para o estudo.

2.3. Tabela Comparativa

A análise demonstra que os trabalhos relacionados, embora utilizem o Google BERT para atacar o mesmo problema, apresentam estratégias diferentes para combatê-lo. Isso porque tanto o FakeBERT quanto o COVID-Twitter-BERT utilizam conjuntos de dados derivados de postagens de redes sociais para a concepção de um modelo de categorização de textos. Por outro lado, o CheckBERT executa a extração de artigos de portais de notícias e os utiliza para o treinamento de um modelo de categorização do relacionamento entre dois textos: a sentença fornecida pelo usuário final e o texto do artigo mais relevante. Adicionalmente, o CheckBERT também oferece os resultados da classificação e informações sobre o artigo em uma interface web.

Dessa maneira, entende-se que ambos os trabalhos relacionados apresentam contribuições consideráveis para o combate à desinformação na internet, evidenciando resultados notáveis em seus respectivos contextos. Analogamente, o CheckBERT fornece uma solução final web destinada a simplificar o processo de verificação de informações inverídicas, contribuindo para o mesmo propósito, mas sob uma perspectiva distinta.

3. CheckBERT

3.1. Fluxo de atividades

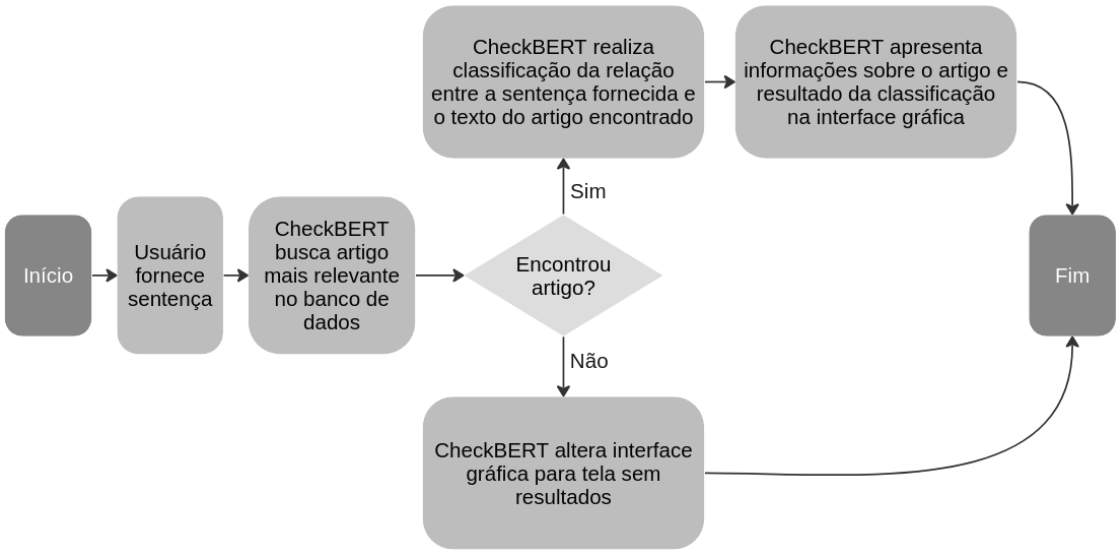
Conforme descrito na Figura 1, o fluxo de atividades durante o uso do CheckBERT é iniciado a partir do fornecimento de uma sentença por parte do usuário, de maneira manual.

Tabela 1. Comparação entre os trabalhos relacionados e o CheckBERT.

Trabalho	Fonte de dados	Contexto dos dados	Estratégia de extração de dados	Taxa de atualização de dados	Modelos utilizados	Resultado final do trabalho
FakeBERT	Redes sociais.	Eleição presidencial norte-americana de 2016.	Não possui.	Não atualiza a base de dados inicial.	BERT + CNN.	Modelo otimizado para detecção de notícias falsas através da análise semântica do conteúdo textual de uma publicação.
COVID-Twitter-BERT	Redes sociais.	COVID.	Não possui.	Não atualiza a base de dados inicial.	BERT.	Modelo otimizado para detecção de notícias falsas através da análise semântica do conteúdo textual de um tweet.
CheckBERT	Portais jornalísticos.	Conteúdo jornalístico diverso.	Coleta dados da web utilizando técnicas de *Web Scraping*.	Configurável (a cada 15 minutos por padrão).	BERT.	Software web capaz de receber uma sentença, identificar um artigo relacionado na base de dados, comparar relacionamento semântico entre a sentença e o texto do artigo, e por fim retornar para o usuário dados do artigo e da classificação realizada pelo modelo BERT.

Essa etapa constitui a única interação exigida do usuário final ao longo de todo o processo de verificação. Logo após, o CheckBERT utiliza a sentença fornecida para efetuar a procura do artigo mais relevante no banco de dados, e então realiza a classificação do relacionamento entre o texto do artigo mais relevante e a sentença fornecida, que pode ser categorizado em um dos seguintes rótulos: concordante, discordante ou neutro. Após realizar a classificação, o CheckBERT atualiza a interface gráfica e apresenta ao usuário os dados do artigo e da classificação.

Figura 1. Fluxo de atividades durante o uso do CheckBERT.



Os dados apresentados variam de acordo com a proveniência dos artigos, uma vez que o CheckBERT aceita artigos de dois gêneros de portais distintos. O primeiro gênero corresponde aos portais de notícia, que divulgam exclusivamente os fatos relacionados a um evento específico. Já o segundo gênero corresponde aos portais verificadores, que averiguam a veridicidade de uma alegação particular e expõem os fatos que conduziram à classificação efetuada.

Qualquer que seja a proveniência do artigo, é esperado que o portal disponibilize ao menos as seguintes informações:

1. Título do artigo;
2. Data de publicação do artigo;
3. URL para leitura gratuita do artigo;

Adicionalmente, artigos de sites verificadores também devem possuir as seguintes informações:

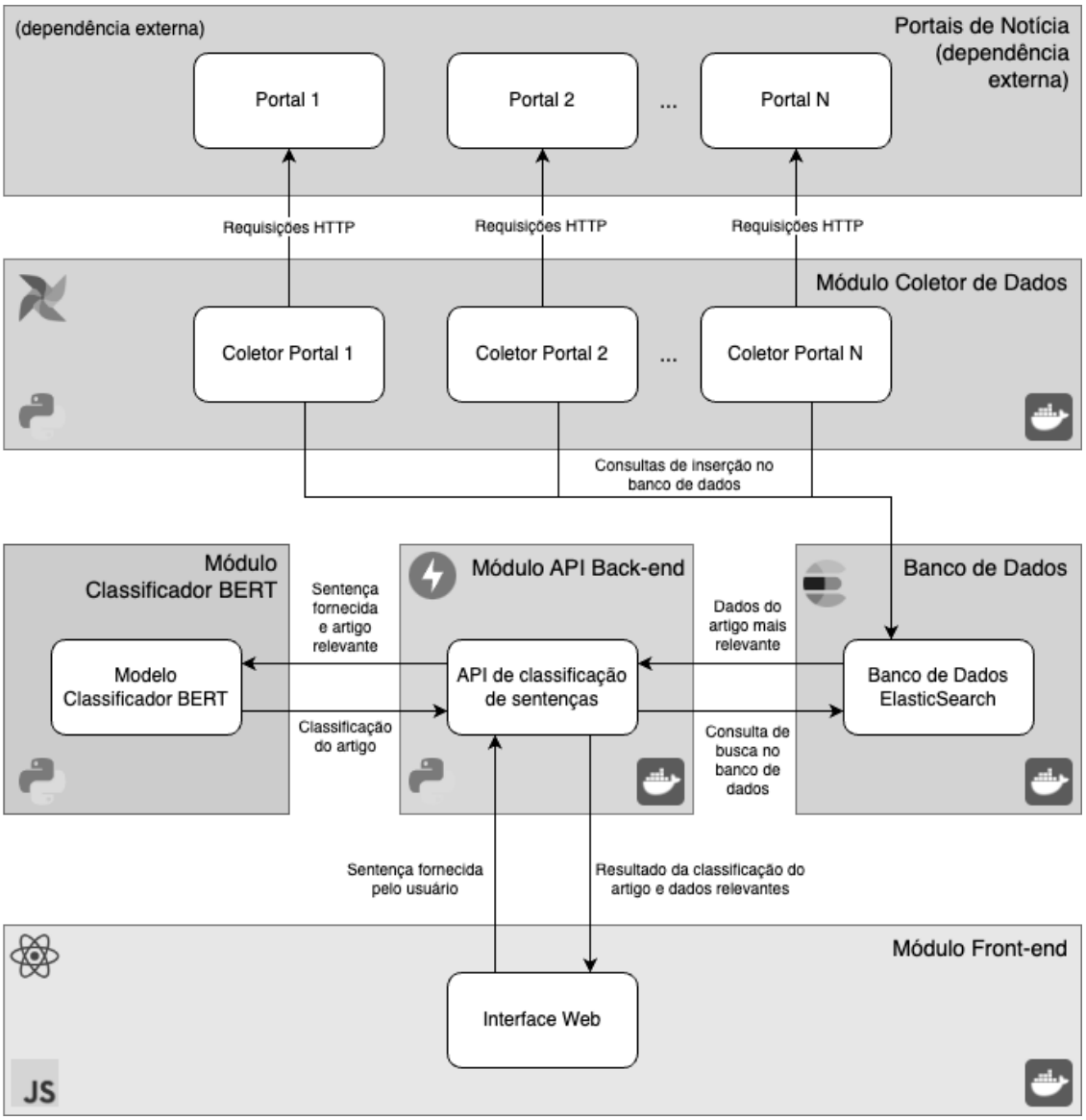
1. Classificação da alegação atribuída pelo portal de notícia (varia de acordo com o portal);
2. Autor da alegação (pode corresponder ao veículo utilizado para publicação, como Facebook ou Twitter, caso o portal decida omitir o autor da publicação);

Por fim, faz-se necessário evidenciar que os artigos de sites verificadores são utilizados para o treinamento do modelo de classificação BERT, mas esse processo não é realizado durante o uso da ferramenta, e está apresentado em mais detalhes nas seções seguintes.

3.2. Arquitetura

Como demonstrado na Figura 2, o CheckBERT é dividido em quatro módulos principais e um banco de dados, além dos portais de notícia com os quais o Módulo Coletor de Dados interage, e que representam dependências externas do projeto. Todos os módulos comunicam-se com ao menos um outro módulo do sistema, mas apenas a módulo Interface Gráfica possui interação com o usuário.

Figura 2. Arquitetura do CheckBERT.



Naturalmente, cada módulo detém funções e finalidades únicas, alinhadas às suas respectivas necessidades para a execução do CheckBERT. Em virtude disso, todos os módulos, juntamente com o banco de dados, foram brevemente descritos abaixo. São eles:

- **Módulo Coletor de Dados:** Utiliza técnicas de *web scraping* para extrair o conteúdo textual dos artigos dos portais jornalísticos selecionados, os quais possuem fundamental importância tanto o para o treinamento do Classificador BERT quanto para a experiência do usuário ao navegar pela ferramenta. Devido ao escopo do trabalho, o único portal selecionado para viabilizar o contexto do sistema foi o Politifact, devido à sua adequação de dados dentro do contexto proposto.
- **Banco de Dados:** Responsável por armazenar todos os dados de artigos coletados pelo Coletor de Dados.
- **Módulo Classificador BERT:** Contém todos os scripts referentes ao treinamento do modelo BERT para tarefas de NLI, que, de acordo com [Souza 2020], consistem na validação de que determinado trecho de texto em língua natural, chamado de premissa, acarreta em outro texto, chamado hipótese. No contexto desse trabalho, o modelo é utilizado para validar se algum trecho de texto do artigo mais relevante acarreta (entende-se como concorda, discorda ou não possui relação) na sentença fornecida pelo usuário.
- **Módulo API Back-end:** Oferece ao Front-end uma interface de acesso às funcionalidades do Classificador BERT e do Banco de Dados, intermediando a comunicação e retornando os dados necessários em formato JSON durante a utilização da ferramenta.
- **Módulo Interface Gráfica:** Representa o único meio de comunicação entre o usuário e o CheckBERT, sendo tanto o ponto de entrada quanto de saída dos dados durante o fluxo de atividades do CheckBERT.

3.3. Interfaces Gráficas

Na Figura 3, foram elencadas as possíveis interfaces do CheckBERT antes e depois da pesquisa realizada pelo usuário, visando documentar o aspecto visual da ferramenta.

3.4. Experimentos Google BERT

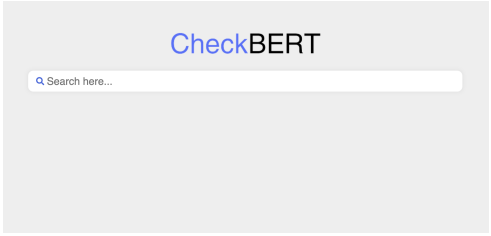
Esta seção visa descrever os experimentos realizados no módulo Classificador BERT para aprimoramento do modelo BERT utilizado.

3.4.1. Conjunto de dados

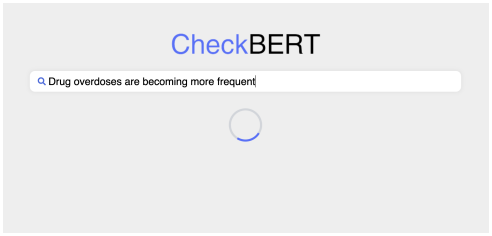
Inicialmente, selecionou-se todos os artigos coletados pelo módulo Coletor de Dados até o dia 13 de março de 2023, totalizando 21620 artigos disponibilizados pelo site Politifact, que foi utilizado para validar a viabilidade do projeto. Esses artigos foram então convertidos para um objeto *DataFrame* da biblioteca Python *pandas*, que foi utilizada em todas as operações subsequentes de transformação dos dados.

No que diz respeito às transformações realizadas, primeiramente as classificações originalmente concedidas pelo site foram convertidas de acordo com a necessidade do

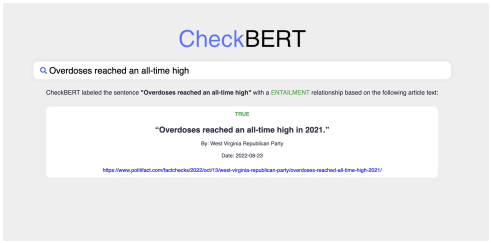
Figura 3. Telas CheckBERT.



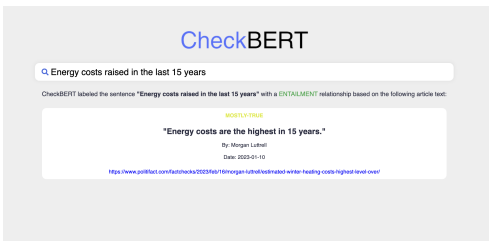
(a) Tela inicial.



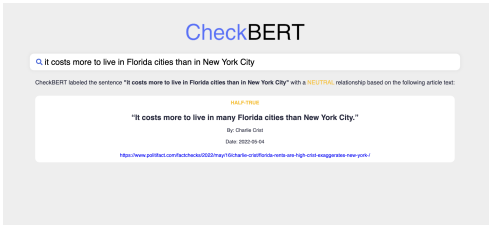
(b) Tela de carregamento.



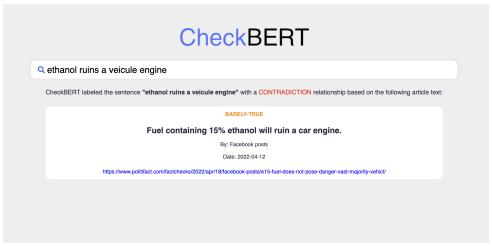
(c) Tela de classificação com alegação classificada como verdadeira pelo Politifact.



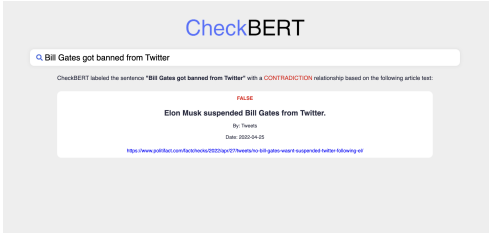
(d) Tela de classificação com alegação classificada como verdadeira em sua maioria pelo Politifact.



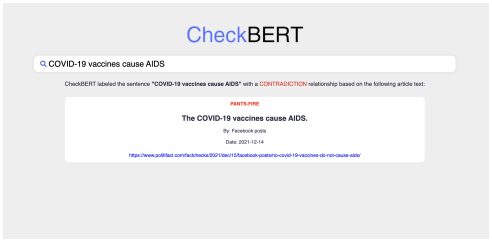
(e) Tela de classificação com alegação classificada como meia-verdade pelo Politifact.



(f) Tela de classificação com alegação classificada como falsa em sua maioria pelo Politifact.



(g) Tela de classificação com alegação classificada como falsa pelo Politifact.



(h) Tela de classificação com alegação classificada como falsa possuindo alegação ridícula pelo Politifact.

desenvolvimento de modelos NLI. Mais especificamente, as classificações *true* e *mostly-true* foram mapeadas para *entailment* (concordância), *half-true* foi convertida para *neutral* (neutralidade), e *mostly-false*, *false* e *pants-fire* foram mapeadas para *contradiction* (discordância).

Em seguida, construiu-se uma base de dados balanceada a partir da fusão de três amostras distintas, cada uma contendo 3642 registros de uma das classificações mapeadas no processo anterior (*entailment*, *neutral* e *contradiction*). Esses registros foram selecionados de maneira aleatória - porém reprodutível - utilizando a função *DataFrame.sample()* da biblioteca *pandas*, com o parâmetro *random_state* definido como 1. Esta abordagem assegura que cada experimento pode ser replicado exatamente da mesma maneira, garantindo assim a integridade e comparabilidade dos resultados.

Ao término do processo de transformação e amostragem, obteve-se um conjunto de dados balanceado com 10926 registros. Cada registro é composto por três variáveis: *label*, que representa a classificação mapeada no processo anterior; *statement*, que representa a alegação verificada pelo Politifact; e *article_text*, que representa o texto do artigo do Politifact. Uma descrição mais detalhada destas variáveis é apresentada na Tabela 2.

Tabela 2. Variáveis do conjunto de dados base.

Variável	Tipo de dado	Descrição
label	*string*	Classificação mapeada a partir da classificação fornecida pelo Politifact. Representa a variável dependente do estudo e sempre possui um dos seguintes valores: *entailment*, *neutral* e *contradiction*.
statement	*string*	Alegação verificada pelo Politifact. Representa uma das variáveis independentes do estudo e também corresponde ao título do artigo.
article_text	*string*	Texto do artigo de verificação do Politifact. Representa uma das variáveis independentes do estudo.

Além disso, vale ressaltar que os artigos extraídos abrangem uma grande diversidade de assuntos (embora todos estejam relacionados ao cenário político norte-americano) e que informações quanto à extensão das alegações verificadas e dos textos dos artigos estão apontadas na Tabela 3. Dentre essas informações, vale ressaltar que o texto dos artigos geralmente ultrapassa o limite de extensão do Google BERT (de aproximadamente 2500 caracteres), conforme indicado pela média calculada, mas que isso foi levado em consideração na aplicação das técnicas, conforme descrito a seguir.

3.4.2. Modelos Bi-Encoder

Os primeiros testes realizados envolveram modelos de codificação *Bi-Encoder*, com o propósito de verificar se modelos com esse tipo de codificação seriam suficientes para as

Tabela 3. Métricas da extensão dos valores de statement e article_text.

Variável	Média	Valor mínimo	Primeiro quartil	Mediana	Terceiro quartil	Valor máximo
statement	109.69	17.00	76.00	101.00	136.00	437.00
article_text	5121.11	308.00	3747.00	4977.50	6300.75	20286.00

tarefas de análise de similaridade semântica propostas. Para isso, foram realizados experimentos utilizando dois modelos pré-treinados, os quais foram disponibilizados pela biblioteca *sentence-transformers*: o *all-MiniLM-L6-v2*[7] e o *all-mpnet-base-v2*[8]. Entretanto, nenhum dos modelos apresentou quaisquer resultados significativos, e portanto foram rapidamente descartados.

3.4.3. Modelos Cross-Encoder

Após a conclusão de que modelos *Bi-Encoder* não são suficientes para alcançar os objetivos do trabalho, foram selecionados os dois modelos *Cross-Encoder* da biblioteca *sentence-transformers* que obtiveram os melhores resultados de acurácia nos datasets SNLI[9] e MULTINLI[10]: *nli-deberta-v3-base*[11] (primeira colocação) e *nli-distilroberta-base*[12] (segunda colocação).

Antes de iniciar os testes, no entanto, fez-se necessário executar um novo processo de preparação de dados na base coletada pelo módulo Coletor de Dados. Esse processo foi motivado pelo tempo elevado de avaliação dos modelos *Cross-Encoder*, bem como pelo ajuste fino dos modelos, que foi realizado posteriormente para melhorar os resultados obtidos. A preparação não aplicou alterações significativas na composição dos dados, tendo apenas separado o conjunto base (com textos completos dos artigos) em três conjuntos menores: o conjunto de treino, com 72% dos dados disponíveis (7866 registros); o conjunto de teste, com 20% dos dados disponíveis (2186 registros); e o conjunto de avaliação, com 8% dos dados disponíveis (874 registros), sendo o último utilizado para avaliar o modelo durante o período de treinamento. Todas as divisões foram realizadas utilizando a função *model_selection.train_test_split* da biblioteca *sklearn*, primeiramente com os parâmetros *test_size* e *random_state* definidos como 0.2 e 42, respectivamente, e então novamente com os parâmetros definidos como 0.1 e 42, respectivamente (na segunda divisão, foi utilizada a base com 80% dos dados extraída na primeira divisão).

Ambos os modelos foram então avaliados utilizando os dados do conjunto de teste, com o auxílio da classe *CESoftmaxAccuracyEvaluator*, disponibilizada no módulo *cross_encoder.evaluation* da biblioteca *sentence-transformers*. Os dois modelos apresentaram acurácias semelhantes, tendo *nli-distilroberta-base* alcançado aproximadamente 31,93% e *nli-deberta-v3-base* alcançado aproximadamente 32,11%.

[7]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2
[8]https://huggingface.co/sentence-transformers/all-mpnet-base-v2
[9]https://nlp.stanford.edu/projects/snli/
[10]https://cims.nyu.edu/ sbowman/multinli/
[11]https://huggingface.co/cross-encoder/nli-deberta-v3-base
[12]https://huggingface.co/cross-encoder/nli-distilroberta-base

Em decorrência dos resultados obtidos, foi concluído que o uso de modelos *Cross-Encoders* pré-treinados também não é capaz de apresentar resultados suficientemente satisfatórios para o objetivo do trabalho, tornando necessário um ajuste fino destes modelos com dados contextualizados. Assim sendo, prosseguiu-se com o ajuste fino do modelo *nli-deberta-v3-base* (uma vez que este apresentou melhores resultados) seguindo duas abordagens distintas, as quais estão detalhadas a seguir:

- **Ajuste fino *nli-deberta-v3-base* com o recorte inicial dos artigos:** esta abordagem utiliza as alegações verificadas por cada artigo, juntamente com seus textos completos, para realizar o ajuste fino do modelo. No entanto, devido à limitação de extensão textual do BERT, artigos mais longos acabam sendo recortados, o que limita o treinamento às seções iniciais dos textos.
  Além dos pares de entradas textuais, outros parâmetros também foram utilizados para o ajuste fino do modelo. As descrições de cada parâmetro, juntamente com seus valores utilizados, estão detalhadas a seguir na Tabela 4.

Tabela 4. Parâmetros de treino nli-deberta-v3-base.

Parâmetro	**Descrição**	**Valor usado**
train_dataloader	Objeto da classe *DataLoader* disponível no módulo *utils.data* da biblioteca *pytorch*.	Objeto *DataLoader* configurado para alterar aleatoriamente a ordem dos artigos e usar conjuntos de 16 artigos por vez (*shuffle=True* e *batch_size=16*).
evaluator	Objeto de alguma das classes disponíveis no módulo *cross_encoder.evaluation* da biblioteca *sentence-transformers.*	Objeto da classe *CESoftmax-AccuracyEvaluator* configurado com a amostra de 874 artigos definida no item anterior.
epochs	Número de épocas do treinamento	3.
evaluation_steps	Representa de quantas em quantas atualizações do modelo a avaliação deve ser realizada.	1000.
warmup_steps	Representa a quantidade inicial de atualizações do modelo que deve ser realizada com menor taxa de aprendizado, uma vez que o modelo ainda não possui muitos exemplos dos dados e pode afetar a assertividade das atualizações.	148.

Após o ajuste fino do modelo, a avaliação de sua acurácia foi conduzida utilizando a mesma abordagem da avaliação dos modelos pré-treinados, a fim de possibilitar a comparação do modelo *nli-deberta-v3-base* antes e depois do ajuste fino. Ao final do processo, o modelo retreinado apresentou acurácia de aproximadamente 63,58%, superando o modelo sem treinamento em aproximadamente 31,44% dos casos.

- **Ajuste fino *nli-deberta-v3-base* com o recorte final dos artigos:** apesar da melhoria significativa apresentada na abordagem anterior, levantou-se a hipótese de que, como o Politifact costuma apresentar suas considerações nos parágrafos finais do texto (os quais muitas vezes são utilizados devido a limitação de extensão textual do BERT), o modelo poderia apresentar resultados melhores caso o recorte de textos extensos fosse realizado utilizando as seções finais dos artigos. Para validar essa hipótese, recortou-se os textos de todos os artigos do conjunto de dados para conter apenas os 2560 caracteres finais (para textos menores, não houve qualquer alteração), de acordo com a seguinte lógica: o número máximo de *tokens* (divisões de palavras realizadas pelo BERT) suportado pelo modelo *nli-deberta-v3-base* é de 512 por classificação, e sabe-se que cada token representa, em média, 4 caracteres. Considerando um espaço entre cada *token*, foi definida uma estimativa de que o modelo seria capaz de lidar com aproximadamente $512 * (4 + 1)$ caracteres, que resulta em 2560. No restante, o processo de treinamento seguiu as mesmas etapas da abordagem anterior.
  Após o ajuste fino, a avaliação do modelo apontou acurácia de aproximadamente 78,36%, superando o modelo com recorte inicial em aproximadamente 14,78% dos casos, e alcançando resultados suficientemente satisfatórios para integração do modelo ao CheckBERT.

Por fim, visando facilitar o acompanhamento dos resultados obtidos, cada um dos modelos Cross-Encoder utilizados, juntamente com suas respectivas acurácias, foram elencados na Tabela 5.

Tabela 5. Acurácias obtidas em modelos Cross-Encoder.

Modelo	**Acurácia Obtida**
nli-distilroberta-base pré-treinado	31,93%
nli-deberta-v3-base pré-treinado	32,11%
nli-deberta-v3-base após ajuste fino com recorte inicial dos artigos	63,58%
nli-deberta-v3-base após ajuste fino com recorte final dos artigos	78,36%

4. Considerações Finais e Trabalhos Futuros

O desenvolvimento desse trabalho resultou na concepção e implementação de um sistema web destinado a combater uma das principais problemáticas da sociedade moderna: a disseminação de informações falsas. A abordagem adotada cobriu desde o desenvolvimento de *pipelines* de extração de dados até a implementação de uma interface gráfica, incluindo, entre essas duas extremidades, a definição e utilização de um banco de dados *ElasticSearch*, a idealização e desenvolvimento de uma API web, e o ajuste fino e utilização de um modelo BERT otimizado para tarefas de NLI.

Já no que se refere ao modelo do Classificador BERT, apesar de seus resultados possuírem espaço para melhorias, acredita-se que o estudo alcançou acurácia suficiente para cumprir seu objetivo, tendo demonstrado melhorias significativas em comparação ao modelo *nli-deberta-v3-base*, quando analisado sob a ótica do contexto de artigos jornalísticos. Entretanto, visando melhorar ainda mais a acurácia do modelo, algumas estratégias que podem auxiliar, além da coleta de mais fontes de dados, são:

- Utilização de sentenças manualmente introduzidas na base de dados do modelo, com o intuito de combater cenários nos quais a sentença fornecida é sintaticamente similar a sentença analisada pelo site verificador, mas semanticamente oposta (o modelo ainda apresenta falhas ocasionais nessa conjectura);
- Adição de uma camada de sumarização de texto antes do treinamento do modelo, visando extrair apenas os fatos e informações relevantes do artigo;
- Aplicação de um maior pré-processamento de dados no treinamento do modelo, aspirando garantir a qualidade de dados e remover caracteres ou sentenças indesejadas;

Finalmente, quanto às sugestões de **Trabalhos Futuros**, pode-se listar as seguintes temáticas não abordadas nesse projeto:

- Adição de outras fontes de dados ao módulo Coletor de Dados.
- Utilização de busca semântica nas consultas de artigos no *ElasticSearch* (viável a partir do armazenamento de *embeddings* do conteúdo textual dos artigos).
- Incorporação de uma rotina de MLOps para gerenciar e monitorar o processo de melhoramento contínuo do modelo.
- Aplicação de melhorias na Interface Gráfica, tornando-a mais responsiva e facilitando sua navegação por parte do usuário.
- Expansão da funcionalidade do CheckBERT para outras línguas além da inglesa, desde a coleta de dados até o treinamento do modelo.

Referências

[Diazgranados] Diazgranados, H. 70% de los latinoamericanos desconoce cómo detectar una fake news. *Karpersky Daily*.

[Kaliyar et al. 2021] Kaliyar, R. K., Goswami, A., and Narang, P. (2021). Fakebert: Fake news detection in social media with a bert-based deep learning approach. *Multimedia Tools and Applications*, 80(8):11765–11788.

[Müller et al. 2020] Müller, M., Salathé, M., and Kummervold, P. E. (2020). Covid-twitter-bert: A natural language processing model to analyse covid-19 content on twitter.

[Riveira] Riveira, C. O brasileiro está mais alerta sobre fake news — mas ficou paranoico. *Exame*.

[Rubio Hancock, Jaime] Rubio Hancock, Jaime. Dicionário oxford dedica sua palavra do ano, 'pós-verdade', a trump e brexit. *El País*.

[Souza 2020] Souza, R. (2020). Comparação de métodos para inferência em linguagem natural. Master thesis, Universidade de São Paulo.

[Veja] Veja. 'fake news' é eleita palavra do ano por dicionário collins. *Veja*.