Arildo Magno de Macedo

Protótipo de aplicação *web* para detecção de plágio

Formiga - MG

2022

Arildo Magno de Macedo

Protótipo de aplicação *web* para detecção de plágio

Monografia do trabalho de conclusão de curso apresentado ao Instituto Federal Minas Gerais - Campus Formiga, como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais

Campus Formiga

Ciência da Computação

Orientador: Me. Roger Santos Ferreira

Formiga - MG

2022

Macedo, Arildo Magno de
M141p Protótipo de aplicação web para detecção de plágio / Arildo Magno de Macedo – Formiga : IFMG, 2022.
86p. : il.

Orientador: Prof. MSc. Roger Santos Ferreira
Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. Plágio. 2. Plágio ofuscado. 3. Lógica Fuzzy. 4. Análise de arquivos 5. Bando de dados léxico. 6. Similaridade. I. Ferreira, Roger Santos. II. Título.

CDD 004

Ficha catalográfica elaborada pela Bibliotecária Msc. Simoni Júlia da Silveira

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS
Campus Formiga
Diretoria de Ensino
Docência Área Acadêmica de Computação
Rua São Luiz Gonzaga, s/n - Bairro São Luiz - CEP 35570-000 - Formiga - MG
- www.ifmg.edu.br

ARILDO MAGNO DE MACEDO

PROTÓTIPO DE APLICAÇÃO PARA DETECÇÃO DE PLÁGIO

Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Minas Gerais - Campus Formiga, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

APROVADO em 25 de novembro de 2022.

BANCA EXAMINADORA

Prof. Roger Santos Ferreira (Orientador)

Profª. Denise Ferreira Garcia Rezende (IFMG)

Prof. Mário Luiz Rodrigues de Oliveira (IFMG)

Formiga, 25 de novembro de 2022.

Documento assinado eletronicamente por **Roger Santos Ferreira**, **Professor**, em 26/11/2022, às 17:57, conforme art. 1º, III, "b", da Lei 11.419/2006.

Documento assinado eletronicamente por **Denise Ferreira Garcia Rezende**, **Professora**, em 28/11/2022, às 11:18, conforme art. 1º, III, "b", da Lei 11.419/2006.

Documento assinado eletronicamente por **Mario Luiz Rodrigues Oliveira**, **Professor**, em 05/12/2022, às 12:00, conforme art. 1º, III, "b", da Lei 11.419/2006.

A autenticidade do documento pode ser conferida no site https://sei.ifmg.edu.br/consultadocs informando o código verificador **1389414** e o código CRC **96A49C26**.

Não há exemplo maior de dedicação do que o da nossa família. À minha querida família, que tanto admiro, dedico o resultado do esforço realizado ao longo deste percurso.

Agradecimentos

Ao professor Roger Ferreira, por ser meu orientador e ter desempenhado tal função com dedicação e amizade. A todos que participaram, direta ou indiretamente, do desenvolvimento deste trabalho, enriquecendo o meu processo de aprendizado.

"Qualquer tecnologia suficientemente avançada é equivalente à mágica"
(Arthur C. Clarke)

Resumo

O plágio é uma forma de má conduta de pesquisa e uma violação ao direito autoral. Neste trabalho, é utilizado um algoritmo de similaridade entre palavras para detectar a probabilidade de plágio entre documentos, baseado em técnicas de processamento de dados, lógica *fuzzy*, bancos de dados e cálculos de similaridade. É possível obter o grau de semelhança entre os documentos, detectando até plágios ofuscados. O presente trabalho apresenta o protótipo de um sistema *web* de análise e inferência para a detecção de plágio em documentos por uma página *web*, bem como resultados de experimentos realizados em ambiente controlado. Foram obtidos resultados relevantes em relação à análise de plágio quando comparados a literatura, além disto, o sistema desenvolvido se mostrou escalável, porém com alguns pontos a se desenvolver em trabalhos futuros, como a utilização de servidores mais robustos.

Palavras-chave: Plágio, Plágio Ofuscado, Lógica *Fuzzy*, Análise de Arquivos, Banco de dados Léxico, Similaridade.

Abstract

Plagiarism is a form of research misconduct and a violation of copyright. In this work, a similarity algorithm between words is used to detect the probability of plagiarism between documents, based on data processing techniques, fuzzy logic, databases and similarity calculations. It is possible to obtain the degree of similarity between documents, even detecting obfuscated plagiarism. The present work presents the prototype of a web analysis and inference system for the detection of plagiarism in documents by a web page, as well as results of experiments carried out in a controlled environment. Relevant results were obtained in relation to the plagiarism analysis when compared to the literature, in addition, the developed system proved to be scalable, but with some points to be developed in future works, such as the use of more robust servers.

Keywords: Plagiarism, Obfuscated Plagiarism, Fuzzy Logic, File Analysis, Lexical Database, Similarity.

Lista de ilustrações

Lista de tabelas

Lista de códigos

Lista de abreviaturas e siglas

HTML	*Hyper Text Markup Language*
CSS	*Cascade Style Sheet*
JS	*JavaScript*
SQL	*Structured Query Language*
API	*Application Programming Interface*
SGBD	*Sistema gerenciador de banco de dados*
LCS	*Least common subsumer*
WUP	*WuPalmer*
ODDS	*Chance de ocorrência de um evento*
MUI	*Material-UI*

Sumário

1 Introdução

O plágio é a deturpação das ideias ou palavras de outra pessoa como se fossem próprias, sem o devido reconhecimento da fonte original (ANDERSON; STENECK, 2011). No sentido de "roubo de propriedade intelectual" existe desde que os humanos produziram obras de arte e pesquisa, no entanto, o fácil acesso à *web*, grandes bancos de dados e telecomunicações em geral, tornou o plágio um sério problema para editores, pesquisadores e instituições de ensino (MAURER; KAPPE; ZAKA, 2006).

A literatura considera a detecção de plágio algo difícil de ser realizado, afinal, uma palavra pode ter vários significados e sentidos possíveis. O plágio mais comum é o literal, que trata apenas de copiar a informação de determinado local e substituir em outro sem atribuir os devidos direitos autorais. Outro tipo de plágio é denominado plágio ofuscado que é mais complexo e por isso mais difícil de ser detectado, pois, os textos plagiados são transformados em palavras e estruturas diferentes (ALZAHRANI; SALIM; PALADE, 2015).

Ademais, a literatura também abrange uma gama enorme de técnicas e meios de detecção de plágio, as quais utilizam bases fundamentais da computação como, lógica booleana e nebulosa, pré-processamento textual e uso de bancos de dados léxicos[1] para análise de similaridade entre documentos, sendo utilizadas para atender ao propósito deste trabalho.

Foram definidos como base e fundamento tecnológico para o desenvolvimento deste trabalho, conhecimento de banco de dados, pré-processamento textual e técnicas de análise de similaridade, além de tecnologias de programação *web* como *Django*, *Python*, *JavaScript*, *React*, *HTML* e *CSS*.

1.1 Justificativa

Segundo Alzahrani, Salim e Palade (2015) a detecção de plágio é uma tarefa difícil e ela pode se tornar pior quando há diversos arquivos para se analisar, embora já existam alguns sistemas de detecção de plágio como os exibidos na seção 2.5 a maioria são de natureza privada e não realizam detecções de plágio em múltiplos arquivos, desta maneira foi idealizado um trabalho que forneça um sistema de análise múltipla de arquivos textuais e gratuito.

[1] Bancos de dados lexicais contêm informações estruturadas sobre palavras de um idioma.

1.2 Objetivos

O objetivo geral deste trabalho de conclusão de curso é desenvolver um protótipo de uma aplicação web para analisar a presença de plágio em múltiplos arquivos, focado na simplicidade de uso e aplicação das tecnologias abordadas. Os seguintes objetivos específicos definem em granularidade mais fina os passos para se atingir o objetivo geral:

- Empregar técnicas de pré-processamento textual.
- Empregar métodos de detecção de similaridade.
- Manipular dados em banco de dados léxicos.
- Modelar uma aplicação.
- Realizar a prototipação de uma aplicação *web* que realize avaliações de plágio.
- Experimentar e analisar os resultados obtidos.

1.3 Estrutura do trabalho

Este trabalho está dividido em seis capítulos: o capítulo 2 apresenta o referencial teórico sobre a base técnica do tema proposto, o capítulo 3 expõe a metodologia, no capítulo 4 são apresentados os resultados obtidos e por fim o capítulo 5 são feitas as considerações finais e trabalhos futuros.

2 Fundamentação teórica

A tabela 1 é fruto de uma revisão literária de trabalhos de detecção de plágio, ela exibe os principais trabalhos estudados e quais as técnicas de detecção de plágio que utilizam, separados em colunas por trabalho, técnica de segmentação e a métrica de detecção de plágio abordada. Foram analisadas as metodologias abordadas nesses trabalhos, como a utilização de segmentação por sentenças e por *Word-Grams*, a detecção de plágio utilizando a métrica de *Wupalmer*, *Lin* e a de *Leacock Chodorow*. Com base nas análises e experimentações utilizando as metodologias desses estudos foi possível traçar a metodologia para a detecção de plágio proposta neste trabalho, além disso, o tópico métrica e método para análise adotados da seção 3 exibe comparações realizadas por outros autores nas métricas de detecção.

Tabela 1 – Trabalhos fundamentalmente relacionados

Autores	Segmentação	Métrica de detecção
Yerra e Ng (2005)	Sentenças	Matriz de correlação de palavras
Alzahrani, Salim e Palade (2015)	Word-Grams e Sentenças	WuPalmer similarity
Ezzikouri, Erritali e Oukessou (2017)	Word-Grams e Sentenças	WuPalmer similarity
Ezzikouri et al. (2018)	Word-Grams	WuPalmer similarity, Leacock Chodorow similarity e Lin similarity

Fonte: Próprio autor

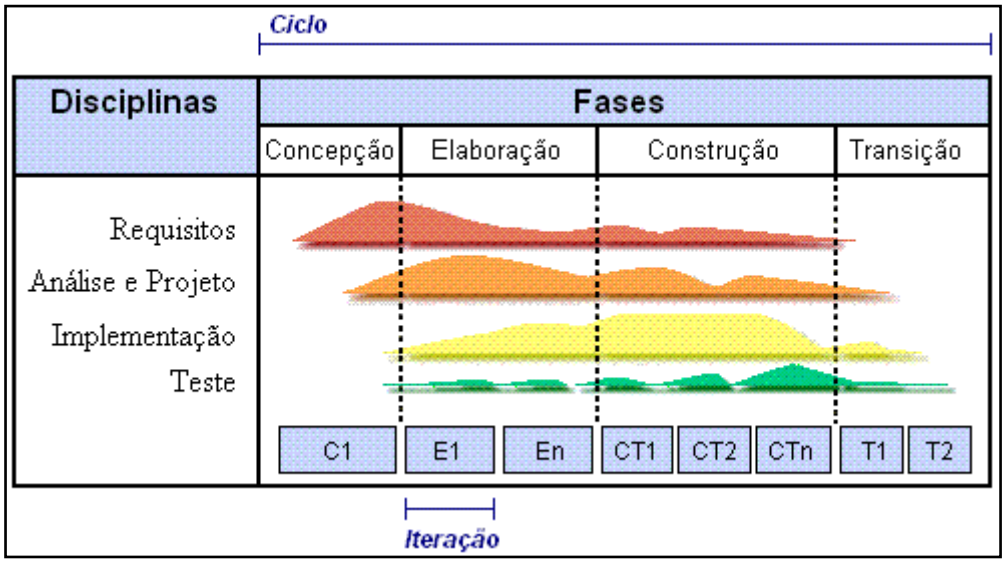
Para o entendimento do assunto abordado neste trabalho uma série de fundamentos e conceitos introdutórios são necessários, estes são descritos abaixo.

2.1 Tecnologias e métodos para desenvolvimento de aplicações *web*

O desenvolvimento deste projeto adotou o modelo do processo de software unificado. De acordo com adonai (2011) este modelo de desenvolvimento de software é interativo e adaptativo, desta forma, consegue produzir um sistema de grande porte como se fossem vários pequenos sistemas diminuindo o risco do projeto. O processo unificado consiste na repetição de uma série de ciclos durante o desenvolvimento de um sistema, por isso,

esse processo é dito como evolucionário subdividido em 4 fases: concepção, elaboração, construção e transição, exibidas na figura 1.

Figura 1 – Modelo de processo de software unificado



Fonte: (ESEMFOCO, 2006)

O *Cross-Origin Resource Sharing* é um mecanismo que permite que recursos restritos em uma página da *web* sejam recuperados por outro domínio fora do domínio ao qual pertence o recurso que será recuperado (WEB.DEV, 2022).

A *WSGI* é a interface de *gateway* do servidor *web*, é uma especificação que descreve como um servidor da *web* se comunica com aplicativos da *web* e como os aplicativos da *web* podem ser encadeados para processar uma solicitação (WSGI.READTHEDOCS.IO, 2022).

2.2 Evolução da análise de plágio

Alguns trabalhos são relevantes para o estudo e evolução da análise plágio, dentre eles pode-se destacar alguns como (ZADEH, 1965) que definiu que um conjunto *fuzzy* é uma classe de objetos que contém graus de pertinência. Kraft e Buell (1983) fizeram um trabalho utilizando estes subconjuntos *fuzzy* para recuperação de informação realizando consultas booleanas em documentos, pouco tempo depois Ogawa, Morita e Kobayashi (1991) propuseram um sistema *fuzzy* de recuperação de documentos utilizando uma matriz de conexão de palavras-chave.

Brin, Davis e García-Molina (1995) apresentaram um sistema chamado *COPS* para registro de documentos e detecção de cópias, sejam cópias completas ou cópias parciais, Shivakumar e Garcia-Molina (1995) apontaram um novo esquema para detecção de cópias

baseado na comparação das ocorrências de frequência de palavras do novo documento com as de documentos registrados chamado de *SCAM*, além disto, Yerra e Ng (2005) definiram uma nova abordagem para detectar documentos *web* semelhantes, especialmente documentos *HTML*.

Alzahrani, Salim e Palade (2015) elaboraram um detector de plágio para a língua inglesa tratando os casos de plágio altamente ofuscados, um modelo de similaridade baseado em semântica *fuzzy* e banco de dados léxico foi apresentado, ademais Ezzikouri et al. (2018) propuseram um detector de similaridade semântica *fuzzy* usando a taxonomia *WordNet* e três abordagens semânticas *WuPalmer*, *Lin* e *Leacock-Chodorow* para documentos árabes.

2.3 Fundamentos da análise de plágio

A tabela 2 ilustra em uma visão macro os fundamentos necessários para a análise de plágio. A tabela está dividida pelos conceitos relacionados com o processamento de linguagem natural, processamento textual, análise de plágio e os cálculos relacionados com a análise de plágio. As descrições dos conceitos são dadas a seguir.

Tabela 2 – Conceitos fundamentais para análise de plágio

Processamento de linguagem natural	**Processamento textual**	**Análise de plágio**	**Cálculos da análise de plágio**
words	part of spech	least common subsummer	relationship words (UanB)
lexicon	stop words	wupalmer similarity	relationship sentences (UA,B)
hyperonymon	wordnet		sentence threshold (EQ)
hyponymon			resemblance between docs (RS(doc1,doc2))
synset			odds
sense			probability
			evaluation measures

Fonte: Próprio autor

Processamento de linguagem natural:

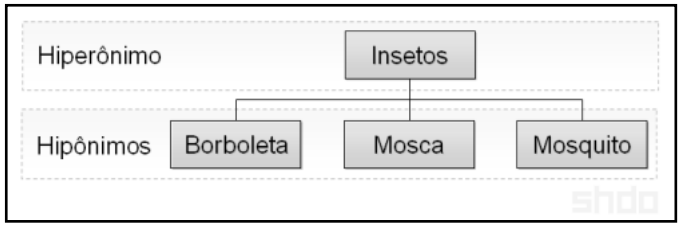
Abaixo são exibidos os principais conceitos relacionados com processamento de linguagem natural para o entendimento deste trabalho.

As *Words* ou palavras são os blocos básicos de construção das línguas constituídas de duas partes, sua forma e seu significado (JURAFSKY et al., 2009). *Lexicon* é um livro contendo um arranjo alfabético das palavras em um idioma e suas definições (WEBSTER,

2022). *Hypernym* ou hiperônimos são palavras de sentido genérico, ou seja, palavras cujos significados são mais abrangentes do que os hipônimos (MUNDOEDUCACAO, 2022) e *Hyponym* ou hipônimos são palavras de sentido específico, palavras cujos significados são hierarquicamente mais específicos do que de outras (BRASILESCOLA, 2022), a figura 2 demonstra uma relação de hiperônimo e hipônimo.

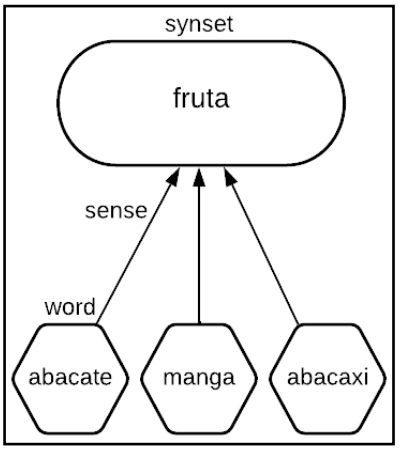
Além disto, *Sense* é uma representação discreta de um aspecto do significado de uma palavra, mas em uma *wordnet* são essencialmente ligações entre palavras e *synsets*, já um *Synset* é formado por grupos de palavras que compartilham o mesmo conceito, formas diferentes de palavras com o mesmo conceito são agrupadas em um *synset* (WN, 2022b), a figura 3 demonstra uma relação de *senses* e *synsets*.

Figura 2 – Relação entre hiperônimo(*hypernym*) e hipônimo(*hyponym*)



Fonte: apsletrasunip (2022)

Figura 3 – Relação entre *word*, *sense* e *synset*



Fonte: Próprio autor

Processamento textual:

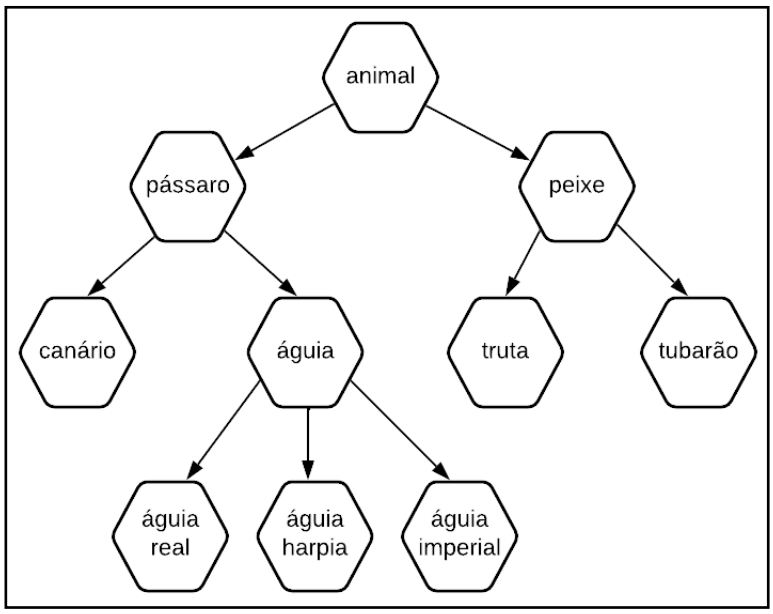
Os principais conceitos relacionados com processamento textual são exibidos a seguir.

O *Part-of-speech* ou parte do discurso é um rótulo especial atribuído a cada *token*(palavra) em um corpus de texto para indicar a parte da fala e muitas vezes também

outras categorias gramaticais, como identificar se o *token* é um substantivo, pronome, verbo, adjetivo, advérbio, preposição, conjunção ou interjeição, as tags *POS* são usadas em pesquisas de corpus e em ferramentas e algoritmos de análise de texto (SKETCHENGINE, 2022). As *Stop Words* são o conjunto de palavras comumente usadas em qualquer idioma (GANESAN, 2022).

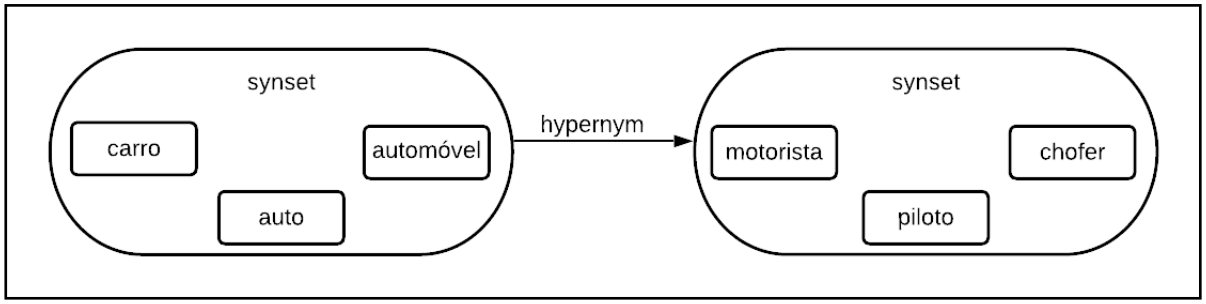
Um *WordNet* é um banco de dados léxico organizado por conceitos como pode ser visualizado na figura 4, a unidade básica de uma *wordnet* são os *synset*, estes são ligados por meio de *senses* para formar a estrutura do *wordnet* (WN, 2022b). No *wordnet* os *synsets* são ligados entre si para formar vários tipos de relações, por exemplo, se o conceito expresso por um *synset* é mais geral do que de outro, então ele está em uma relação de *hypernym*(hiperônimo) com este outro *synset* (WN, 2022b), conforme exibe a figura 5.

Figura 4 – Estrutura de uma *WordNet*



Fonte: Próprio autor

Figura 5 – Relação de *hypernym* entre *synsets*



Fonte: Próprio autor

Análise de plágio:

As definições formais para análise de plágio são exibidas abaixo.

O *Least Common Subsumer(LCS)* é o ancestral comum mais específico de dois conceitos encontrados em uma dada ontologia, semanticamente representa a semelhança do par de conceitos (GLOBAL, 2022). A *WuPalmer Similarity* é uma métrica que calcula o parentesco de dois *synsets* nas taxonomias *WordNet* utilizando o *Least Common Subsumer* (ADVISORS, 2022).

Uma métrica é uma medida para algo, um meio de derivar uma medida quantitativa ou aproximação para fenômenos qualitativos, há alguns tipos de métricas na análise de plágio, as métricas baseadas no conteúdo da informação, como as três medidas (LIN, 1993) (JIANG; CONRATH, 1997) e (RESNIK, 2011) onde o conteúdo de informação de um nó em *WordNet* é a soma de todas as probabilidades de todas as palavras naquele *synset*, quanto maior a probabilidade de encontrar uma instância de um *synset* menor se torna seu conteúdo de informação, enquanto *synsets* contendo palavras raras têm alto conteúdo de informação.

Também há métricas baseadas no comprimento do caminho, como as (LEACOCK; CHODOROW, 1995) e (WU; PALMER, 1994), a maioria das medidas de similaridade ou parentesco baseadas no comprimento do caminho usam algum valor para dimensionar o comprimento do caminho, *Leacock-Chodorow* usa o dobro da profundidade máxima na taxonomia para dimensionar o comprimento do caminho, *WuPalmer* divide a profundidade dupla do *LCS* de dois conceitos pela soma do comprimento do caminho entre os conceitos e a profundidade do *LCS*.

Um método é um processo pelo qual uma tarefa é concluída, há alguns tipos de métodos para realizar uma análise de plágio, como os métodos baseados em caracteres, esse tipo de método explora recursos baseados em caracteres, eles utilizam estes recursos para encontrar semelhanças entre um documento de consulta e documentos existentes. Os baseados em sintaxe, esses métodos exploram recursos sintáticos como parte do discurso *(POS)* de frase e palavras em diferentes declarações para detectar plágio. Os baseados em semântica onde uma frase pode ser definida como um grupo ordenado de palavras.

Como também existem os métodos baseados em *fuzzy* onde a similaridade das palavras de um texto é representada por valores que variam de zero a um. Os baseados em estrutura que se baseiam na semelhança contextual de como as palavras são usadas em todos os documentos. Os baseados em estilometria, esses métodos visam quantificar os estilos de escrita do autor, calculando a pontuação de similaridade entre duas seções ou parágrafos com base nas características estilométricas dos autores.

Cálculos relacionados com a análise de plágio:

Para o desenvolvimento deste trabalho e validação do mesmo são utilizados conceitos e cálculos propostos em outros estudos, conforme apresentados nesta seção.

As chances de ocorrência de um evento, ou *ODDS*, são calculadas com a razão entre o número de eventos que produzem um resultado e o número que não produz (SZUMILAS, 2010), úteis para sumarizar resultados. As *ODDS* podem ser convertidas para probabilidade dividindo o valor das *ODDS* por 1 + *ODDS* (GRAPHPAD, 2021) e a probabilidade pode ser multiplicada por 100 para obter o valor em porcentagem. As equações da *ODDS* e da conversão para probabilidade são exibidas como 2.1 e 2.2.

$$odds = \frac{p}{1-p} \tag{2.1}$$

$$probability = odds/(1 + odds) \tag{2.2}$$

No trabalho de Alzahrani, Salim e Palade (2015) no tópico *Evaluation of rules* foram descritas equações que fornecem um modo de se calcular a similaridade entre duas sentenças utilizando a métrica de *WuPalmer* para avaliar suas palavras. Onde a similaridade entre uma sentença A com outra B é dada avaliando a similaridade *wup(WuPalmer)* de todas as palavras que compõe a sentença A em relação às palavras da sentença B, em geral, a semelhança entre as sentenças A, B e B, A se diferem conforme o tamanho das sentenças, por esta razão é necessário realizar a análise de ambos os lados da relação. As equações são exibidas como 2.3 e 2.4.

$$\mathrm{U}_{\mathrm{a}_n B} = MAX(wup(an, b1), wup(an, b2)..., wup(an, \mathrm{b}_m)) \tag{2.3}$$

$$\mathrm{U}_{\mathrm{b}_n A} = MAX(wup(bn, a1), wup(bn, a2)..., wup(bn, \mathrm{a}_m)) \tag{2.4}$$

Para avaliar o resultado em um único valor de similaridade das sentenças, é calculado a média do resultado da análise de cada palavra, onde UA,B é o lado da relação A, B e UB,A o lado B,A. As equações são exibidas como 2.5 e 2.6.

$$\mathrm{U}_{A,B} = \frac{(\mathrm{U}_{\mathrm{a}_1 B} + \mathrm{U}_{\mathrm{a}_2 B} + ... + \mathrm{U}_{\mathrm{a}_n B})}{n} \tag{2.5}$$

$$\mathrm{U}_{B,A} = \frac{(\mathrm{U}_{\mathrm{b}_1 A} + \mathrm{U}_{\mathrm{b}_2 A} + ... + \mathrm{U}_{\mathrm{b}_n A})}{n} \tag{2.6}$$

No trabalho Alzahrani, Salim e Palade (2015) também foi descrita uma equação no tópico *Interpretation of the result* onde dada a semelhança entre a sentença A e B e de B com A, deve pegar o menor valor entre elas e verificar se este valor é superior a um determinado limiar p, se sim, as duas sentenças são similares. A equação segue como 2.7.

$$EQ = 1\ se\ (MIN(\mathrm{U}_{A,B}, \mathrm{U}_{B,A}) > p)\ 0\ caso\ contrário. \tag{2.7}$$

No trabalho Yerra e Ng (2005) foi descrita uma equação no tópico *detecting html similar documents* onde dada a quantidade de sentenças similares entre os documentos é possível encontrar a similaridade dos documentos, em geral, a similaridade do documento 1 em relação ao documento 2 difere da similaridade do documento 2 em relação ao documento 1 conforme o tamanho dos documentos, assim também sendo necessário analisar os documentos de ambos os lados da relação. As equações seguem como 2.8 e 2.9.

$$RS(\mathrm{doc}_1, \mathrm{doc}_2) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} EQ(\mathrm{S}_{\mathrm{d1}_i}, \mathrm{S}_{\mathrm{d2}_j})}{m} \tag{2.8}$$

$$RS(\mathrm{doc}_2, \mathrm{doc}_1) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} EQ(\mathrm{S}_{\mathrm{d2}_i}, \mathrm{S}_{\mathrm{d1}_j})}{m} \tag{2.9}$$

Os estudos de Alzahrani, Salim e Palade (2015) também forneceram alguns parâmetros e equações no tópico *Evaluation measures* que podem ser utilizados para mensurar a qualidade de uma ferramenta detectora de plágio, os parâmetros são: *precision*/precisão *(Pplag)*, *recall*/cobertura *(Rplag)*, *harmonic-mean*/média harmônica *(Fplag)*, *granularity*/granularidade *(Gplag)* e *score-point*/pontuação *(Scoreplag)*. As equações seguem como 2.10, 2.11, 2.12, 2.13 e 2.14.

$$\mathrm{P}_{plag} = \frac{TP}{TP + FP} \tag{2.10}$$

$$\mathrm{R}_{plag} = \frac{TP}{TP + FN} \tag{2.11}$$

$$\mathrm{F}_{plag} = 2 * \frac{\mathrm{P}_{plag} * \mathrm{R}_{plag}}{\mathrm{P}_{plag} + \mathrm{R}_{plag}} \tag{2.12}$$

$$\mathrm{G}_{plag} = \frac{\mathrm{NP}_{detected}}{\mathrm{NP}_{annotated}} \tag{2.13}$$

$$\mathrm{Score}_{plag} = \frac{\mathrm{F}_{plag}}{\log_2(1 + \mathrm{G}_{plag})} \tag{2.14}$$

Onde TP refere-se ao número de casos de plágio detectados corretamente, FP refere-se ao número de detecções falsas de casos e FN refere-se ao número de casos de plágio que não são detectados como plágio. O *NPdetected* é o número de detecções corretas, *NPannotated* os casos anotados como plágios e as medidas de avaliação são combinadas em um único valor *Scoreplag*, que pode ser usado para fazer uma comparação quantitativa de algoritmos de detecção de plágio.

2.4 Banco de dados léxico

O *OpenWN-PT* é um banco de dados léxico de relações entre palavras. Foi desenvolvido e aprimorado nos trabalhos de Paiva, Rademaker e Melo (2012) e Coelho et al. (2014).

O processo de construção do *OpenWN-PT* utilizou aprendizado de máquina para construir relacionamentos utilizando informações provenientes de várias versões da Wikipedia, bem como dicionários abertos, partindo de uma projeção ao nível dos lemas em português e suas relações, o *OpenWN-PT* tem sido constantemente aprimorado por meio de acréscimos linguísticos, manuais ou semiautomáticos (PAIVA; RADEMAKER; MELO, 2012).

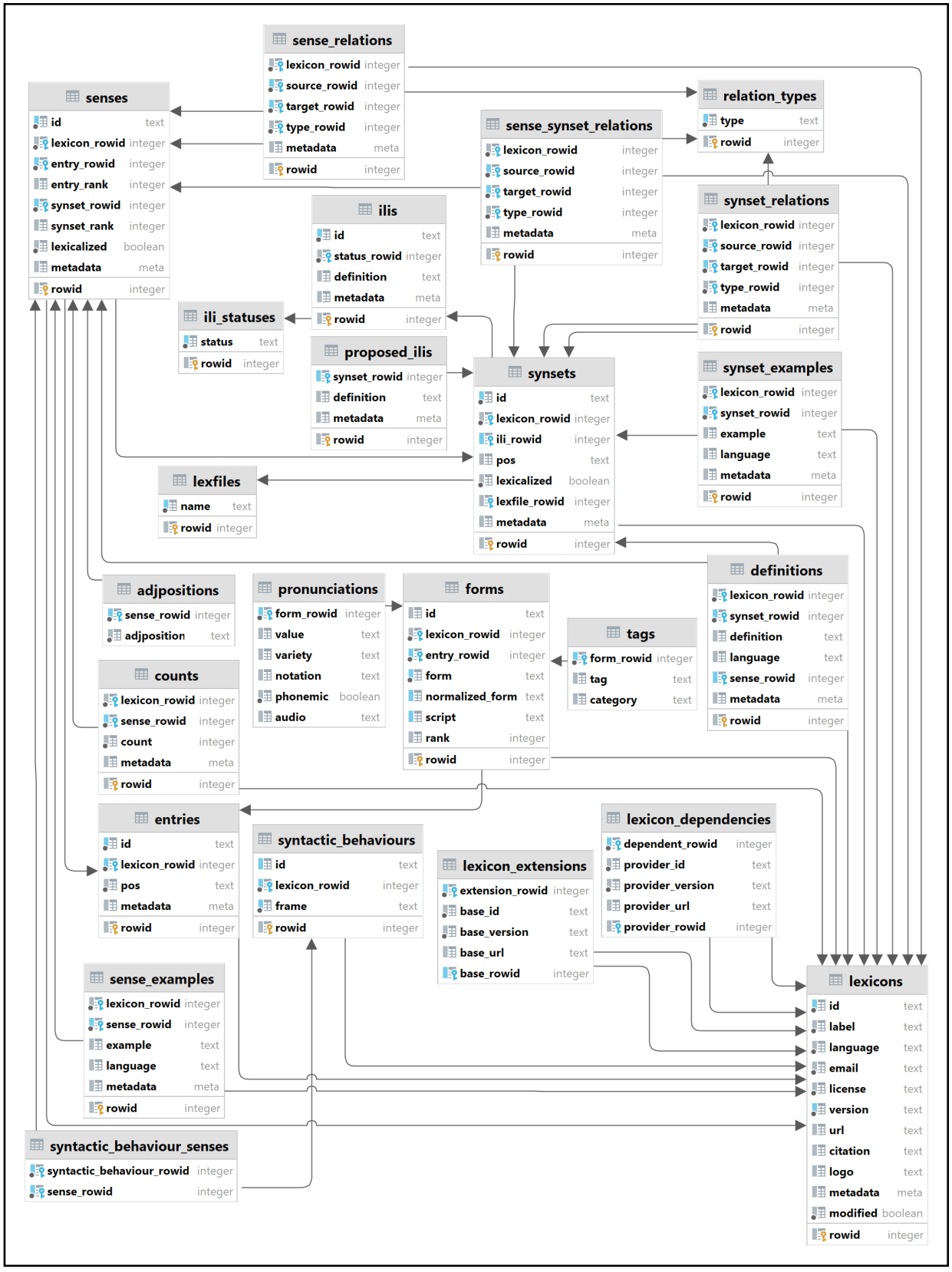
A tabela 3 apresenta as entidades do banco de dados e suas descrições, o *schemma* completo do banco pode ser visualizado na figura 6.

Tabela 3 – Entidades banco de dados

nome	**descrição**
adjpositions	armazena a posição da entidade como adjetivo
counts	armazena a quantidade de ocorrência de uma entidade
definitions	armazena a definição de alguma entidade
entries	armazena as entidades
forms	armazena as formas canônicas das palavras
ili_statuses	armazena o status do identificador interlingual
ilis	armazena o identificador interlingual
lexfiles	armazena o nome do arquivo lexicógrafo da entidade
lexicon_dependencies	armazena dependências do banco léxico
lexicon_extensions	armazena extensões do banco léxico
lexicons	armazena as informações do banco léxico
pronunciations	armazena as informações sobre pronúncias das entidades
proposed_ilis	armazena informações sobre os synsets e sua definição
relation_types	armazena os tipos de relações entre as entidades
sense_synset_relations	armazena as relações entre os senses e os synsets
sense_examples	armazena exemplos de senses
sense_relations	armazena as relações entre os senses
senses	armazena os senses
synset_examples	armazena alguns exemplos de synsets
synset_relations	armazena as relações entre os synsets
synsets	armazena os synsets
syntatic_behaviours	armazena comportamentos syntaticos
syntatic_behavior_senses	armazena comportamentos syntaticos relativo aos senses
tags	armazena as tags

Fonte: Próprio autor

Figura 6 – DER do banco de dados completo



Fonte: Paiva, Rademaker e Melo (2012)

2.5 Projetos semelhantes

Alguns projetos semelhantes ao trabalho aqui proposto podem ser encontrados disponíveis na *web*, a lista abaixo exibe os principais e seus pontos de semelhança:

- *Ithenticate*: este aplicativo compara os documentos de entrada com as fontes de documentos disponíveis na *web* (ITHENTICATE, 2022). Assemelhasse em fornecer uma análise baseada em segmentos do texto, detectar plágio ofuscado e fornecer um relatório com as sentenças possivelmente plagiadas destacadas.

- *CheckPlagiarism*: este sistema verifica a semelhança de texto entre duas *urls* ou em dois arquivos e mostra o conteúdo correspondente (PLAGIARISM, 2022). Assemelhasse em fornecer a possibilidade de comparar o conteúdo de dois arquivos entre si, podendo submetê-los, detectando plágio ofuscado, porém somente entre dois textos.

- *Prepostseo*: este projeto compara dois documentos ou páginas da *web* lado a lado para descobrir o conteúdo plagiado (PREPOSTSEO, 2022). Assemelhasse em comparar o conteúdo de dois textos entre si, no entanto, somente entre dois textos e não detectando plágio ofuscado.

- *CopyLeaks*: este sistema realiza a comparação de dois documentos de texto que podem estar em formatos diferentes e mostra o conteúdo plagiado (COPYLEAKS, 2022). Assemelha em comparar o conteúdo de dois textos entre si que podem estar em formatos diferentes, mas apenas entre dois textos e não detectando plágio ofuscado.

3 Materiais e Métodos

Compõem a metodologia adotada na solução proposta por este trabalho as técnicas e tecnologias apresentadas nesta seção, divididas entre materiais e métodos.

3.1 Materiais

Para o desenvolvimento deste trabalho foi necessário o uso de materiais de desenvolvimento *web*, como ferramentas de prototipação, linguagens de programação, bibliotecas e servidores, abaixo segue uma lista com os materiais utilizados:

- *HyperText Markup Language*: é a linguagem de marcação padrão para documentos projetados para serem exibidos em um navegador da *web*, ele pode ser auxiliado por tecnologias como *Cascading Style Sheets (CSS)* e linguagens de *script* como *JavaScript* (ORG, 2015). Utilizou-se a versão 5.

- *Cascading Style Sheets*: é uma linguagem de folha de estilo usada para descrever a apresentação de um documento escrito em uma linguagem de marcação como *HTML* ou *XML* (MOZILLA, 2015), o *CSS* é projetado para permitir a separação de apresentação e conteúdo, layout, cores e fontes (ORG, 2011). Utilizou-se a versão 4.15.

- *JavaScript*: é uma linguagem compilada de alto nível, muitas vezes *just-in-time* que está conforme o padrão *ECMAScript* (ES, 2020), possui tipagem dinâmica, orientação a objetos baseada em protótipos e funções de primeira classe, multi-paradigma, suportando estilos de programação orientados a eventos, funcionais e imperativos (JAVASCRIPT, 2022). Utilizou-se a versão ES6.

- *Material UI*: é uma biblioteca de componentes React de código aberto que implementa o *Material Design* do Google, ele inclui uma coleção abrangente de componentes pré-construídos que estão prontos para uso na produção imediatamente (MUI, 2022). Utilizou-se a versão 4.12.4.

- *React*: é uma biblioteca *JavaScript* de *Front End* gratuita e de código aberto (REACTJS, 2022) para construir interfaces de usuário baseadas em componentes de *UI*, o *React* pode ser usado como base no desenvolvimento de aplicativos de página única (REACT, 2022). Utilizou-se a versão 17.0.2.

- *Echarts*: é uma biblioteca de gráficos combináveis construída em componentes *React* (ECHARTS, 2022). Utilizou-se a versão 3.0.2.

- *Structured Query Language*: é uma linguagem específica de domínio usada em programação e projetada para gerenciar dados mantidos em um sistema de gerenciamento de banco de dados relacional ou para processamento de fluxo em um sistema de gerenciamento de fluxo de dados relacional (HEUSER, 2009).

- *SQLite*: é um mecanismo de banco de dados escrito na linguagem *C* (SQLITE, 2010). Utilizou-se a versão 3.32.0.

- *OpenWN-PT*: é uma *WordNet* de acesso aberto para o português (PAIVA; RADEMAKER; MELO, 2012). Utilizou-se a versão 1.0.0.

- *wn*: é um pacote que fornece uma interface de comunicação com os dados de uma *WordNet* (WN, 2022a). Utilizou-se a versão 0.9.1.

- *Python*: é uma linguagem de programação interpretada, orientada a objetos e de alto nível com semântica dinâmica, suas estruturas de dados integradas combinadas com tipagem dinâmica e vinculação dinâmica o tornam muito atraente para o Desenvolvimento Rápido de Aplicativos, bem como para uso como *script* ou linguagem de colagem para conectar componentes existentes (PYTHON, 2022). Utilizou-se a versão 3.10.

- *Django*: é um framework para desenvolvimento rápido para *web*, inclui dezenas de extras que podem ser usados para lidar com tarefas comuns de desenvolvimento *web*, o *Django* cuida da autenticação do usuário, administração de conteúdo, mapas do site, *feeds RSS* e muitas outras tarefas (DJANGOPROJECT, 2022). Utilizou-se a versão 3.1.3.

- *Django REST framework*: é um kit de ferramentas construído em cima do *Django web framework* que reduz a quantidade de código necessários à criação de interfaces *REST* (FRAMEWORK, 2022). Utilizou-se a versão 3.12.2.

- *Django Cors Headers*: um aplicativo *Django* que adiciona cabeçalhos *CORS (Cross-Origin Resource Sharing)* 2.1 às respostas, isso permite solicitações no navegador para seu aplicativo *Django* de outras origens (HEADERS, 2022). Utilizou-se a versão 3.5.0.

- *Django Heroku*: é uma biblioteca *Django* para aplicativos Heroku que garante uma experiência de implantação e desenvolvimento perfeita (HEROKU, 2022b), utilizou-se a versão 0.3.1.

- *Heroku*: é uma plataforma de nuvem que suporta várias linguagens de programação, o *Heroku* é considerado uma plataforma poliglota, pois possui recursos para um desenvolvedor criar, executar e dimensionar aplicativos de maneira semelhante na maioria dos idiomas (HEROKU, 2022a). Utilizou-se a versão 7.62.0.

- *Gunicorn*: é um servidor *HTTP Python Web Server Gateway Interface*, o servidor *Gunicorn* é amplamente compatível com vários *frameworks* da *web*, implementado de forma simples, leve nos recursos do servidor e bastante rápido (GUNICORN, 2022). Utilizou-se a versão 20.0.4.

- *WhiteNoise*: com algumas linhas de configuração, o *WhiteNoise* permite que seu aplicativo *web* sirva seus próprios arquivos estáticos, tornando-o uma unidade independente que pode ser implantada em qualquer lugar sem depender de *nginx*, *Amazon S3* ou qualquer outro serviço externo (WHITENOISE, 2022). Utilizou-se a versão 5.2.0.

- *spaCy*: é uma biblioteca de *software* de código aberto para processamento avançado de linguagem natural, escrita nas linguagens de programação *Python* (SPACY.IO, 2022). Utilizou-se a versão 3.2.4.

- *Axios*: é um cliente *HTTP* baseado em *Promises* para fazer requisições, pode ser utilizado tanto no navegador quanto no *Node.js* ou qualquer serviço de *API* (AXIOS, 2022). Utilizou-se a versão 0.21.1.

- *Sendinblue*: oferece um serviço de envio de e-mails (SENDINBLUE, 2022).

- *Pdfminer.six*: é um pacote *Python* para extrair informações dos documentos em *PDF* (PDFMINER, 2022). Utilizou-se a versão 20220524.

- *docx2txt*: um utilitário baseado em *Python* puro para extrair texto e imagens de arquivos *docx* (DOCX2TXT, 2022). Utilizou-se a versão 0.8.

- *Reportlab PDF*: uma biblioteca *Python* para gerar *PDFs* e gráficos (REPORTLAB, 2022). Utilizou-se a versão 3.6.9.

- *Git*: é um software para rastrear alterações em qualquer conjunto de arquivos, geralmente usado para coordenar o trabalho entre programadores que desenvolvem código-fonte colaborativamente durante o desenvolvimento de software. Seus objetivos incluem velocidade, integridade de dados e suporte para fluxos de trabalho distribuídos e não lineares (ARCHIVE, 2005), (MARC, 2007). Utilizou-se a versão 2.34.1.

- *GitHub*: é um provedor de hospedagem na Internet para desenvolvimento de software e controle de versão usando *Git*, ele oferece o controle de versão distribuído e a funcionalidade de gerenciamento de código-fonte do *Git*, além de seus próprios recursos, ele fornece controle de acesso e vários recursos de colaboração, como rastreamento de *bugs*, solicitações de recursos, gerenciamento de tarefas, integração contínua e wikis para cada projeto (GITHUB, 2021). Utilizou-se a versão 2.37.3.

- *Astah*: é uma ferramenta de modelagem *UML* (ASTAH, 2021). Utilizou-se a versão 8.1.
- *Figma*: é uma aplicação *web* colaborativa para design de interface (FIGMA, 2021). Utilizou-se a versão 107.0.

3.2 Métodos

Segue uma lista com os métodos utilizados neste trabalho, bem como a utilização dos materiais:

- *Diagramas UML*: utilizando o *Astah* são criados diagramas *UML* que auxiliam no processo de desenvolvimento do sistema.
- *Mockups*: usando o *Figma* são criados *Mockups* de todas as páginas visuais do sistema para serem seguidas no desenvolvimento das mesmas.
- *Construção da área de interação com o usuário*: feito em *React.js* com a biblioteca *MaterialUI* seguindo os *Mockups* e *UML* definidos, este módulo feito de forma que seja amigável ao usuário empregando conceitos de usabilidade, este módulo irá receber os dados referentes aos arquivos e enviá-los a *API* que irá processá-los e retornar ao *Front End* em formato *Json*, de maneira que o resultado possa ser processado por interfaces gráficas como a biblioteca *Echarts* utilizada para renderizar o resultado ao usuário de forma gráfica.
- Construção da *API*: para a aplicação é criada uma *api* feita em *Django* com utilização do *SGBD SQLite* com o banco de dados *OpenWN-PT* utilizando chamadas em *SQL*. As consultas e requisições ao banco de dados serão feitas nesta camada e seu retorno dos dados requisitados serão transmitidos ao módulo de visão via *JSON*. Dentro da API são utilizados pacotes e bibliotecas que auxiliam no processo de detecção de plágio, sendo eles *docx2txt* e *pdfminer*, para conversão dos arquivos em texto, *wn* para utilizar a wordnet, *spaCy* para obter a lista de *stopwords* em português, *reportlabpdf* para gerar *pdf* com o resultado e *sendinblue* para enviar o resultado ao usuário via servidor de *e-mail*.
- Métrica e Método para análise adotados: dentre as métricas e métodos descritos na seção 2.3 foi escolhido utilizar a *WuPalmer* que utiliza métricas baseadas no comprimento do caminho e utiliza métodos baseados em semântica *fuzzy*.

  Comparações entre as métricas foram realizadas por outros autores e seus resultados são exibidos abaixo e baseado em tais resultados, pode-se concluir que a métrica de Wu e Palmer (1994) apresenta resultados relevantes quando comparada as demais.

Nos estudos de (WARIN; OXHAMMAR; VOLK, 2008) foram analisados métricas de análise de similaridade na ontologia *Common Procurement Vocabulary* usando uma medida de similaridade semântica e *WordNet*, os resultados da comparação entre eles são apresentados na tabela 4, a metade superior da tabela exibe os resultados quando nenhum limiar é usado e a metade inferior mostra com o efeito do limiar.

Tabela 4 – Comparação entre métricas de similaridade

Measure	**Average P**	**Recall**
No threshold		
Resnik	0,705	0,839
Leacock-Chodorow	0,684	0,977
Wu-Palmer	0,677	0,977
Lin	0,616	0,580
Jiang-Conrath	0,626	0,724
Baseline	0,565	0,977
Threshold		
Resnik	0,711	0,667
Leacock-Chodorow	0,711	0,845
Wu-Palmer	0,703	0,770
Lin	0,626	0,483
Jiang-Conrath	0,633	0,655
Baseline	0,592	0,774

Fonte: WARIN, OXHAMMAR e VOLK (2008)

No trabalho Mcinnes e Pedersen (2013) também foram analisados métricas de similaridade e utilizaram um conjunto de dados biomédicos que contém 203 termos e siglas ambíguos, a tabela 5 exibe o resultado da comparação entre as medidas baseadas em caminho propostas por Leacock e Chodorow (1995), Wu e Palmer (1994) e Nguyen e Al-Mubaid (2006), métricas baseadas no conteúdo da informação, taxonomia Jiang e Conrath (1997) e Lin (1993), a medida de relacionamento *lesk* adaptada proposta Lesk (1986) e a medida de parentesco proposta por Patwardhan, Banerjee e Pedersen (2003).

Tabela 5 – Comparação entre métricas de similaridade

WS	**Path-based**				**Corpus-based**			**Taxonomy-based**		
	path	lch	wup	nam	res	jcn	lin	res	jcn	lin
2	0,63	0,63	0,64	0,63	0,64	0,65	0,65	0,65	0,65	0,64
5	0,66	0,66	0,67	0,66	0,68	0,69	0,69	0,68	0,68	0,68
10	0,68	0,68	0,69	0,69	0,70	0,71	0,71	0,70	0,70	0,70
25	0,71	0,69	0,71	0,71	0,73	0,73	0,74	0,73	0,73	0,73
50	0,71	0,69	0,70	0,71	0,73	0,74	0,74	0,73	0,73	0,73
70	0,71	0,69	0,70	0,71	0,73	0,74	0,74	0,73	0,73	0,73

Fonte: Mcinnes e Pedersen (2013)

Por fim, Mcinnes e Pedersen (2013) realizando experimentos com conjuntos de dados biomédicos, foi constatado que a métrica *WuPalmer Similarity* superou os outros três candidatos no *benchmark nDCG*, a tabela 6 exibe o resultado.

Tabela 6 – Comparação entre métricas de similaridade

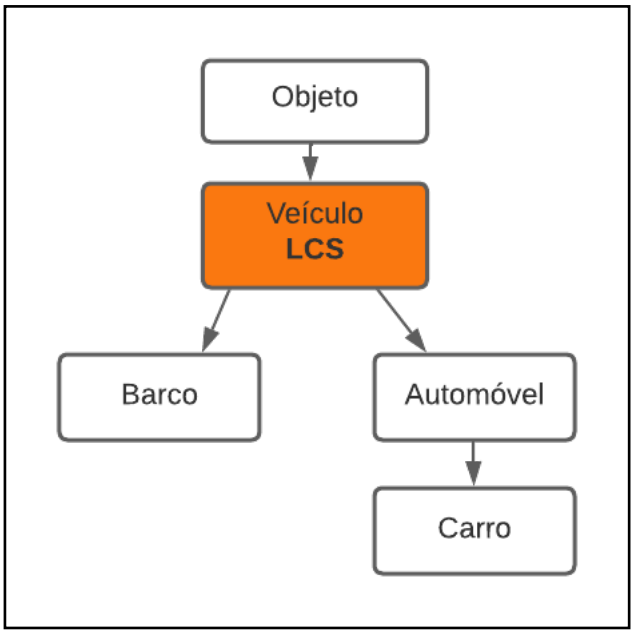
Query	**GoogleDistance**	**WuPalmer**	**Resnik**	**Word2Vec**
E2	0,3969	0,4411	0,4487	0,3911
E7	0,3845	0,4553	0,4504	0,3781
E8	0,4066	0,4476	0,4451	0,4163
E17	0,4429	0,4454	0,448	0,4336
E26	0,2636	0,3911	0,354	0,2796
E28	0,3633	0,4684	0,4678	0,3807
E31	0,433	0,4394	0,4383	0,3866
E35	0,3686	0,4622	0,456	0,3727
E50	0,3345	0,3811	0,4016	0,3535
E54	0,3711	0,48	0,4762	0,3862
E66	0,4609	0,4835	0,4833	0,466
E67	0,3779	0,4493	0,4486	0,3908
E68	0,429	0,4486	0,447	0,4302
E78	0,379	0,4324	0,4383	0,4381
E79	0,4334	0,4234	0,4172	0,3934
E80	0,2572	0,413	0,4144	0,2694
E89	0,3866	0,4202	0,3908	0,3605
E94	0,4305	0,3772	0,373	0,3642

Fonte: Mcinnes e Pedersen (2013)

- *Least Common Subsumer*: também conhecido como *Lowest Common Hypernym* ou *LCS* de dois conceitos X e Y, é o conceito mais próximo que pode ser definido como ancestral de X e Y, um conceito é definido como um ancestral de outro conceito da mesma forma que é definido um ancestral na árvore genealógica humana, onde

os avós são os ancestrais dos pais, em outras palavras, o *LCS* é o ancestral comum mais específico de dois conceitos, semanticamente, representa a semelhança do par de conceitos, quando se trata de *wordnet* os ancestrais são os *hypernym* de uma determinada palavra conforme definido no tópico *hypernym* e *hyponym* na seção 2. A figura 7 demonstra este tipo de relação, onde o veículo é o *hypernym*(ancestral) mais específico de carro e barco, o que o torna o *LCS* de carro e barco.

Figura 7 – *Least Common Subsumer(LCS)*



Fonte: Próprio autor

- *WuPalmer*: o cálculo da métrica de similaridade de *WuPalmer* (ADVISORS, 2022) tem sua concepção baseada em medir a similaridade entre dois *synsets* baseado no quão próximo eles estão semanticamente, para isto utiliza-se o *Least Common Subsumer(LCS)* em seu cálculo. Sua lógica pode ser faseada, sendo elas:

  a) Receber os *synsets* que se deseja analisar.

  b) Calcular a lista de *hypernym* para cada *synset*.

  c) Calcular quais são os *hypernym* que ambos *synsets* possuem em comum.

  d) Calcular o *hypernym* em comum mais distante da raiz, assim encontrando o *LCS*.

  e) Calcular a distância do synset1 ao *LCS* na árvore.

  f) Calcular a distância do synset2 ao *LCS* na árvore.

  g) Calcular a distância do número de nós (distância + 1) do *LCS* ao nó raiz da árvore.

h) Aplicar a equação proposta por *WuPalmer*:

Equação: $wup = \frac{2k}{i+j+2k}$

- **i**: a distância do *synset1* ate o *LCS* na árvore.
- **j**: a distância do *synset2* até o *LCS* na árvore.
- **k**: o número de nós (distância + 1) do *LCS* ao nó raiz da árvore.

- Análise dos documentos: o processo de detecção utiliza os cálculos descritos no tópico cálculos relacionados com a análise de plágio na seção 2.3. Para realizar a análise entre palavras e sentenças é utilizado a metodologia de Alzahrani, Salim e Palade (2015), onde é empregado a métrica de *Wu-Palmer* para analisar a semelhança entre um par de palavras e utilizado a análise das palavras das sentenças para obter a similaridade entre as sentenças.

  Além disto, é utilizada a metodologia abordada em Yerra e Ng (2005) usando um limiar para validar a similaridade entre duas sentenças e calculando a similaridade entre os documentos baseado na quantidade de sentenças semelhantes entre eles.

- Validação da eficácia do protótipo apresentado: para validar a eficácia/eficiência do protótipo apresentado será utilizado o método descrito por Alzahrani, Salim e Palade (2015) que pode ser encontrado no tópico cálculos relacionados com a análise de plágio na seção 2.3, no conjunto de dados composto de 128 arquivos, sendo 50 obtidos da internet e 78 criados pelo autor. As formas de validação serão conduzidas nos tipos de análises rápida e profunda, as quais se diferenciam na quantidade de *synsets* que cada uma analisa nas palavras. Sendo assim, para este conjunto de dados é calculado a quantidade de casos de plágio detectados corretamente, o número de detecções falsas, o número de casos de plágio que não são detectados como plágio, a granularidade e a pontuação da solução proposta no trabalho.

- *Versionamento*: o versionamento é feito utilizando o *Git*, por meio de um repositório no *Github*, ambas aplicações do sistema, *Front End* e API serão versionadas no mesmo projeto visando otimização das consultas.

- *Hospedagem*: é utilizado o *heroku* para hospedar o sistema, para que este sistema seja disponibilizado no *heroku* é necessário utilizar algumas bibliotecas e configurações, sendo elas o *Django Heroku* para a integração do sistema ao heroku, *gunicorn* que é um servidor *HTTP WSGI* e o *whitenoise* que servirá ao servidor os arquivos estáticos do sistema e para fazer a comunicação entre o *Front End* e a *API* será utilizado o *axios*, *django-rest-framework* e o *django-cors-headers*.

4 Resultados e discussões

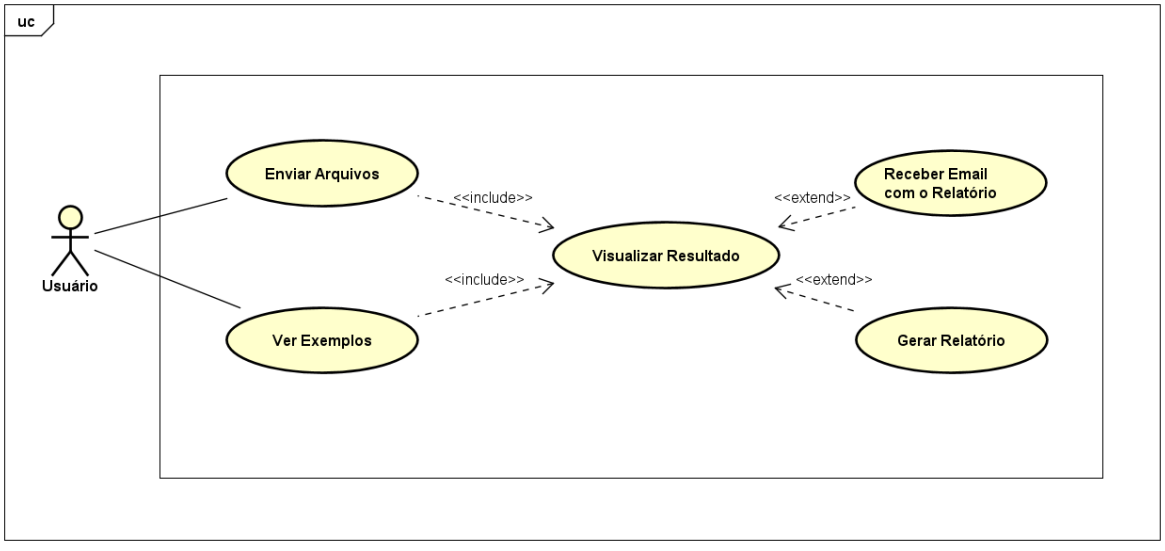
Nesta seção são detalhados os resultados e decisões tomadas ao decorrer do período de desenvolvimento deste trabalho. São apresentadas as etapas de decisão, implementação e integração com outras soluções utilizadas, assim como a execução e comparação dos resultados obtidos. O protótipo desenvolvido no presente trabalho está disponível em detectorplagio.herokuapp.com.

4.1 Casos de uso

O modelo *UML* baseado no (SOMMERVILLE, 2011) foi utilizado para demonstrar como o sistema funcionaria. Assim, foi desenvolvida a sua expansão de uso, sendo divididas todas as suas interações com os usuários e as funções a oferecer. O diagrama demonstrou o fluxo básico de eventos e com base neste diagrama foi traçada a rota de desenvolvimento do sistema por meio do processo unificado.

Para concepção do sistema foi desenvolvido o diagrama de casos de uso apresentado na figura 8 o qual demonstra como é a interação do usuário com a solução proposta. O usuário pode interagir com o sistema de duas formas: a primeira, enviando seus próprios arquivos para serem analisados, obtendo seu resultado através de gráfico e relatórios, ou como segunda possibilidade via aba de exemplos onde o usuário utiliza os próprios documentos contidos no servidor.

Figura 8 – Caso de uso



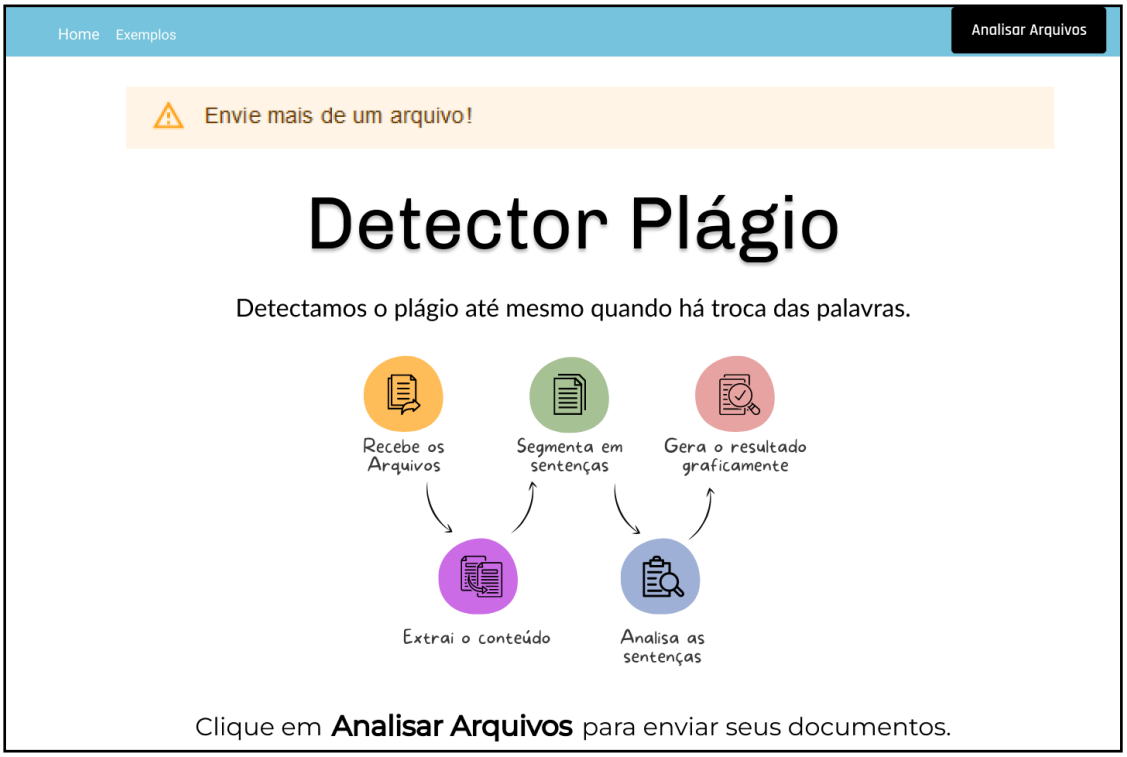
Fonte: Próprio autor

4.2 Mockups

Dados os casos de uso, foi possível gerar um *mockup* para cada uma das telas do sistema. A criação das telas feitas por meio do *Figma*[1] possibilitou a visualização e alteração do *layout* antes de sua implementação.

A figura 9 demonstra como é a ideia da página inicial, que contém uma barra de navegação e a opção para enviar os arquivos para serem analisados, a figura 10 exibe como é feito alerta ao usuário o auxiliando na utilização do sistema, a figura 11 mostra como é a página de envio do e-mail e escolha do tipo de análise rápida ou profunda realizada pelo usuário, a figura 12 mostra como é a confirmação antes de processar os arquivos, a figura 13 mostra como é a página de *loading* enquanto a análise é realizada, a figura 14 exibe a página de exemplos que fornece ao usuário a possibilidade de testar o sistema sem precisar subir seus arquivos utilizando arquivos pré-carregados no servidor e a figura 15 mostra como é exibido o resultado após uma análise, de maneira gráfica contendo a porcentagem de similaridade entre os arquivos como um grafo onde cada nó representa um arquivo, a aresta a ligação entre eles e o peso a similaridade entre os mesmos, exibindo um caso onde a similaridade entre o arquivo *text1* e *text3* é de 33.5%.

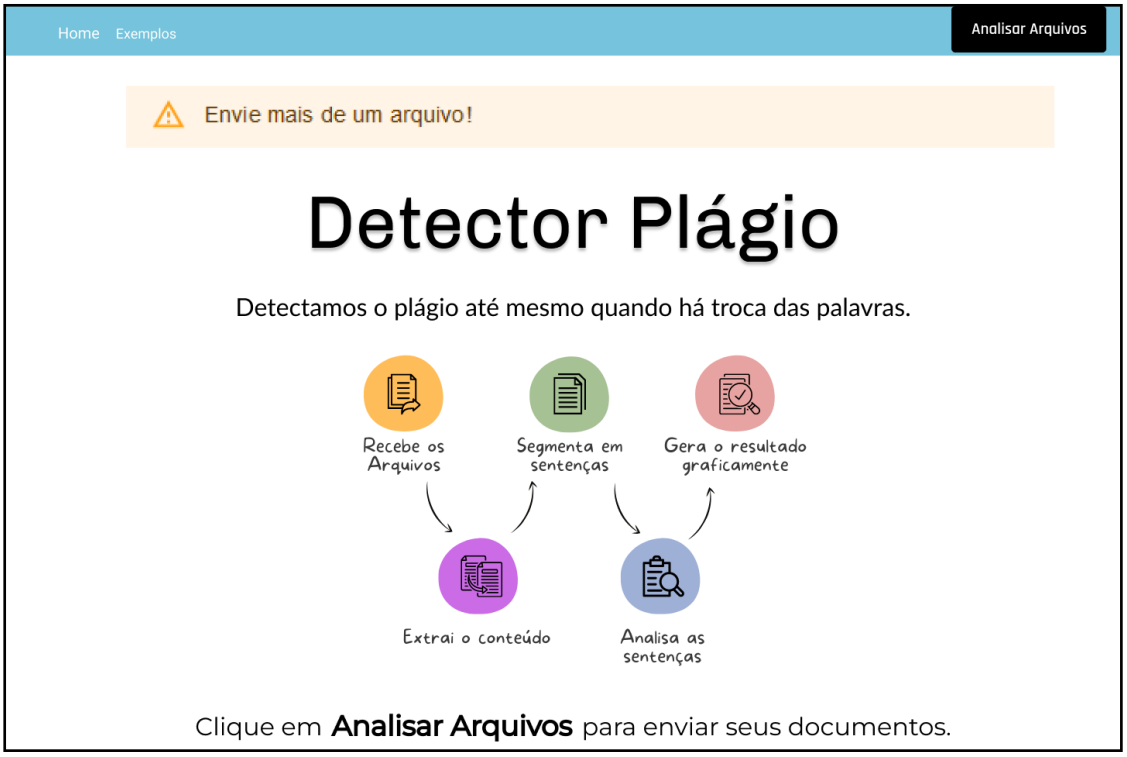
Figura 9 – *Mockup* da tela inicial



Fonte: Próprio autor

[1] *Figma* é um editor gráfico de vetor e prototipagem de projetos de design

Figura 10 – *Mockup* dos avisos ao usuário



Fonte: Próprio autor

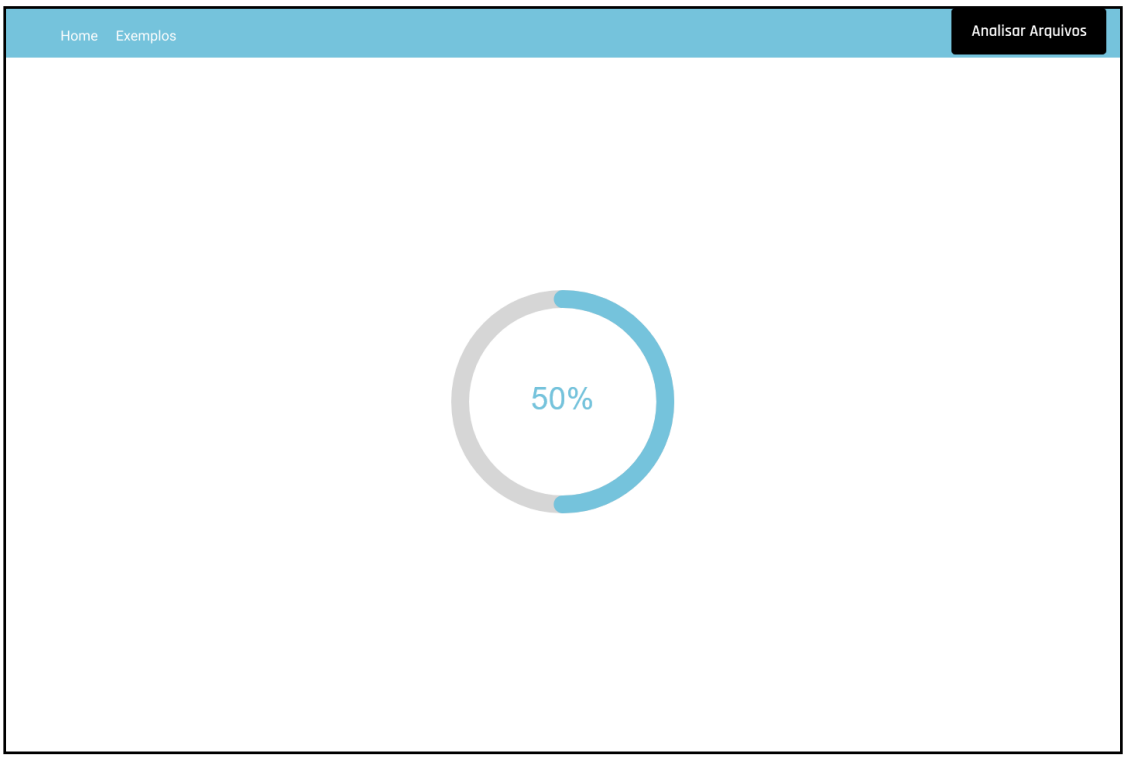
Figura 11 – *Mockup* da escolha de tipo e *email*

Fonte: Próprio autor

Figura 12 – *Mockup* da confirmação de escolha de tipo e *e-mail*

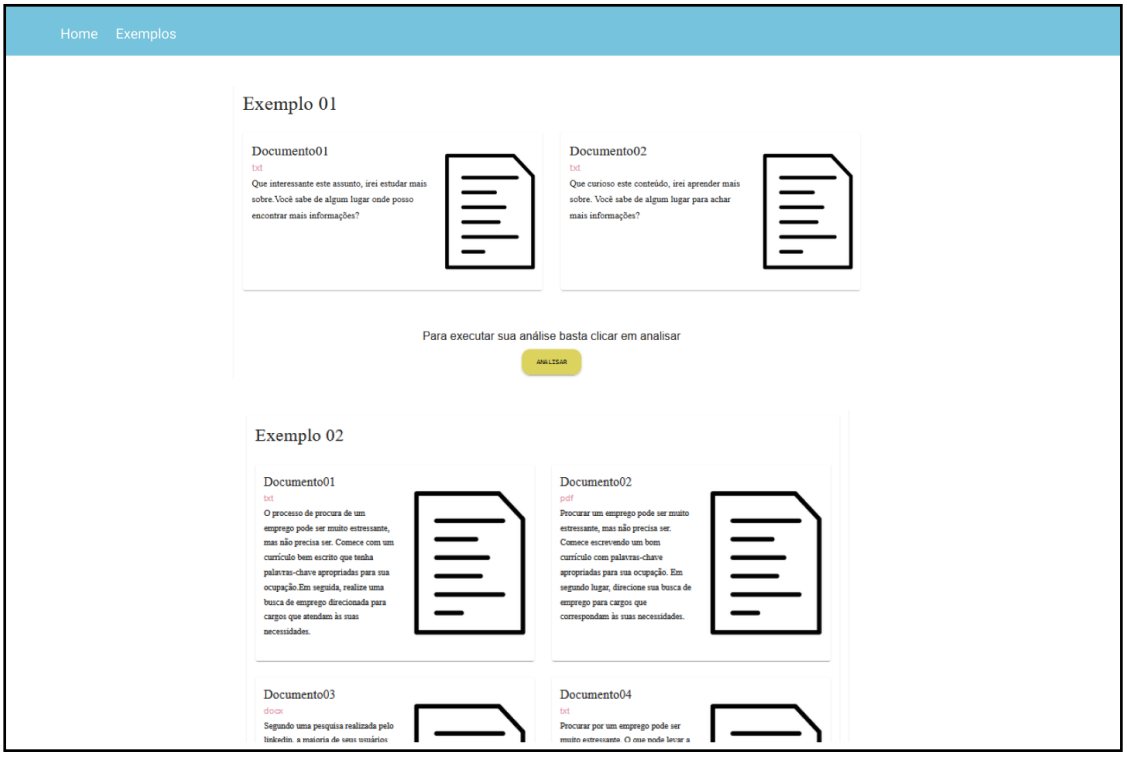
Fonte: Próprio autor

Figura 13 – *Mockup* da tela de *loading*



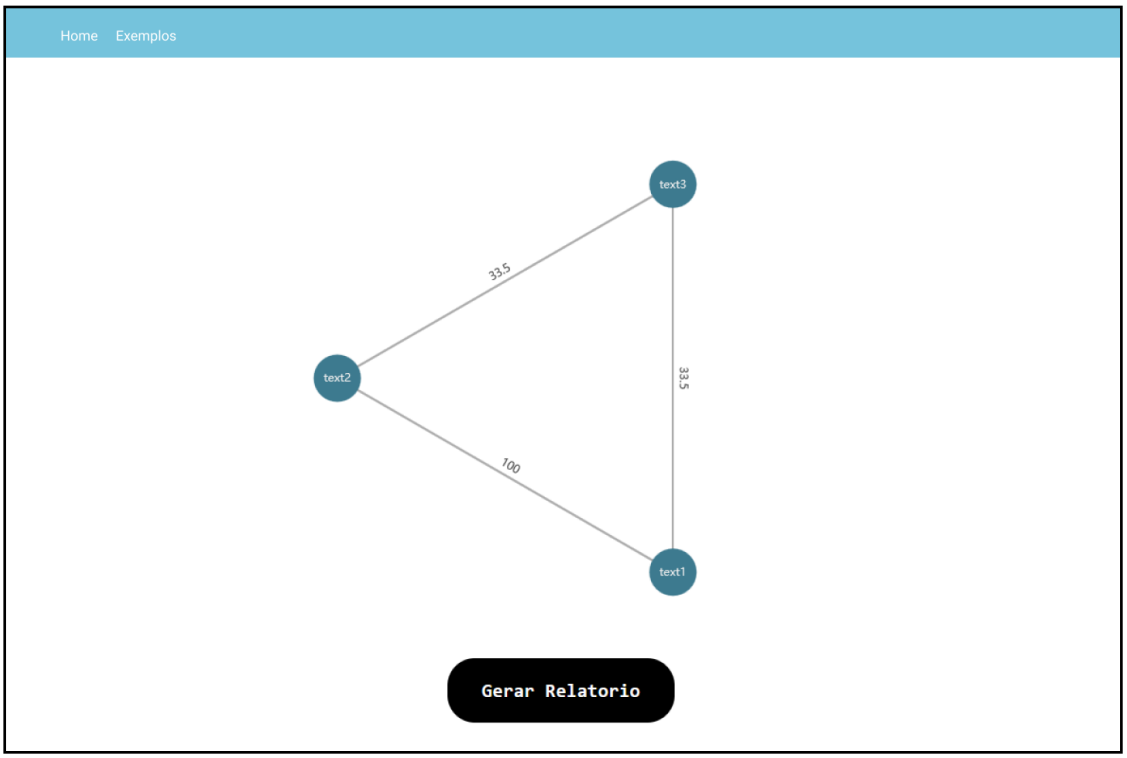
Fonte: Próprio autor

Figura 14 – *Mockup* da tela de exemplos



Fonte: Próprio autor

Figura 15 – *Mockup* da tela de resultados



Fonte: Próprio autor

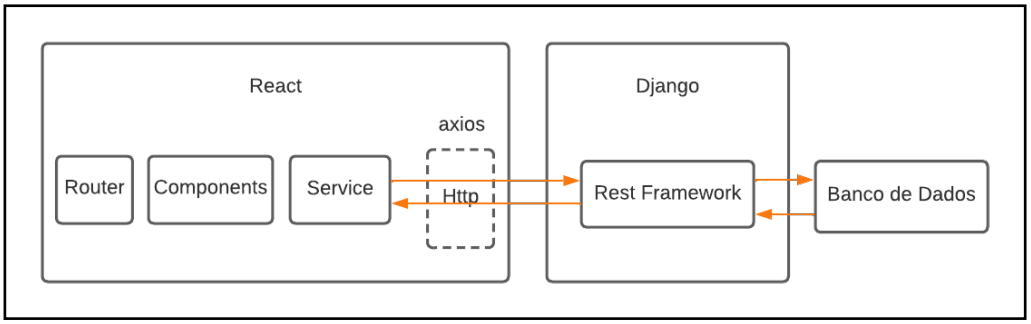
4.3 Estrutura do sistema

A estruturação divide o sistema em duas partes, interface com o usuário e *API*, ambos alocados juntos no *Heroku*[2].

O *Front End* fica responsável por enviar os arquivos do usuário via requisições à *API* para que a mesma realize suas operações, tais como processamento e análise dos dados e então retorne ao *Front End* para serem exibidos graficamente.

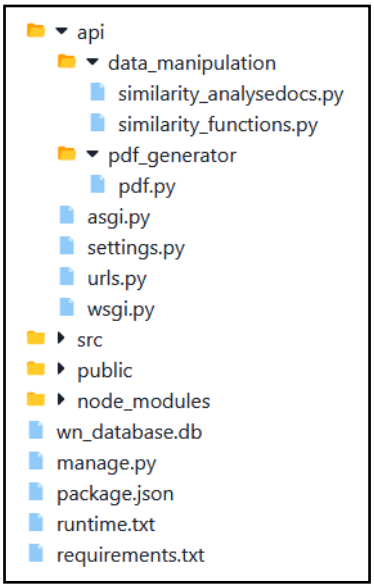
Como pode ser visualizado na figura 16, o *Django* exporta *APIs REST* usando o *Django REST Framework*, o *React Client* envia solicitações *HTTP* e recupera as respostas *HTTP*, a figura 17 apresenta a estrutura de pastas da solução proposta, que oferece o necessário para se adequar ao escopo e requisitos do presente projeto.

Figura 16 – Integração *API* e *Front End*



Fonte: Próprio autor

Figura 17 – Estrutura de diretórios do projeto



Fonte: Próprio autor

[2] *Heroku* é uma plataforma de nuvem como serviço que suporta várias linguagens de programação.

4.3.1 Servidor

Este projeto foi realizado com um servidor *Gunicorn* e o mesmo está hospedado no *Heroku*. O *Heroku* fica responsável por subir o servidor e mantê-lo ativo. Neste servidor foi desenvolvido um projeto com a *stack Django-React*, é utilizado o *whitenoise* para servir os arquivos estáticos do sistema ao servidor, utilizado o *axios*, *django-rest-frameworks* e *django-cors-headers* para fazer a comunicação entre o *Front End* e a *API* do sistema.

4.3.2 *Front End*

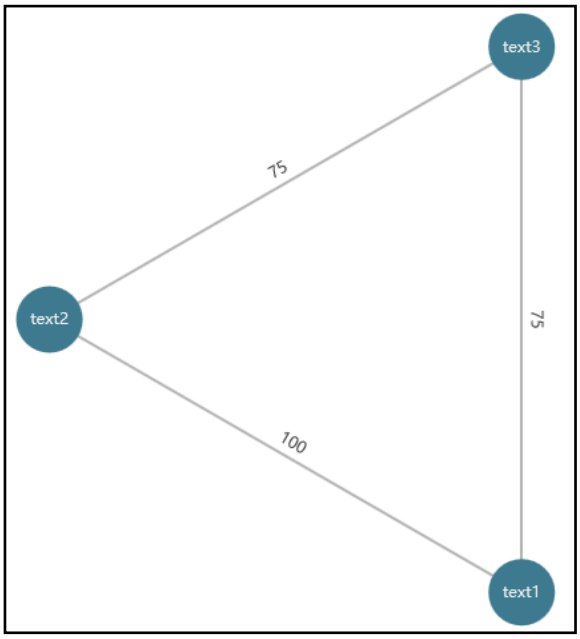
A construção do *Front End* foi baseada nos diagramas *UML* e *Mockups* previamente criados, a parte que se relaciona com o *client* é desenvolvida em *React* e quando é necessário se comunicar com a *API* o *axios* é utilizado chamando a rota de algum método, conforme demonstrado no código 4.1, nos métodos envolvendo arquivos estes são enviados para a *API* como *FormDatas*[3].

Código 4.1 – Axios Front End Chamada

```
axios.get("/api/analise")
```

O *Front End* fica encarregado de exibir o resultado da análise ao usuário, para isto é utilizado a biblioteca gráfica *Recharts* que modela os resultados obtidos em forma de grafo, como exibe a figura 18, o *Front End* também é responsável por disponibilizar as opções de gerar relatório com os resultados ou enviar via *e-mail*.

Figura 18 – Resultado da análise de arquivos exibido via *Recharts*



Fonte: Próprio autor

[3] Os objetos *FormData* são usados para criar formulários HTML e enviá-los em algum método de rede.

4.3.3 *API*

O *Back End* é feito em *Django* e funciona como uma *API* recebendo requisições do *Front End* e retornado os resultados como *JSON* ao mesmo. Os códigos 4.2 e 4.3 exibem um exemplo de uma rota e a invocação do seu método relacionado.

Código 4.2 – *API urls*

```
urlpatterns = [
path('analise', Analise.as_view()),
]
```

Código 4.3 – *API views*

```
class Analise(APIView):
def post(self, request, format=None):
    return Response(analyse(documents))
```

4.4 Banco de Dados

No sistema foi empregado o banco de dados *OpenWN-PT*, porque o sistema requer um banco de dados no modelo *WordNet* em português e construir um não faz parte do escopo do trabalho. Porém, alguns processos foram realizados neste banco, os processos e seus resultados são descritos a seguir.

Para a manipulação deste banco de dados e aplicação do algoritmo de *WuPalmer* foi utilizado o pacote (WN, 2022a), uma vez que o *wn* dispõe em seu conjunto o banco de dados *OpenWN-PT* e o algoritmo da métrica de similaridade de *WuPalmer*. Além de manipular banco de dados e conter algoritmos de análise de similaridade, o pacote *wn* realiza diversas outras operações, por isto o mesmo necessita que os bancos de dados que irão utilizá-lo sigam seu *schemma*[4] padrão para bancos de dados, que pode ser visualizado seção 2.4.

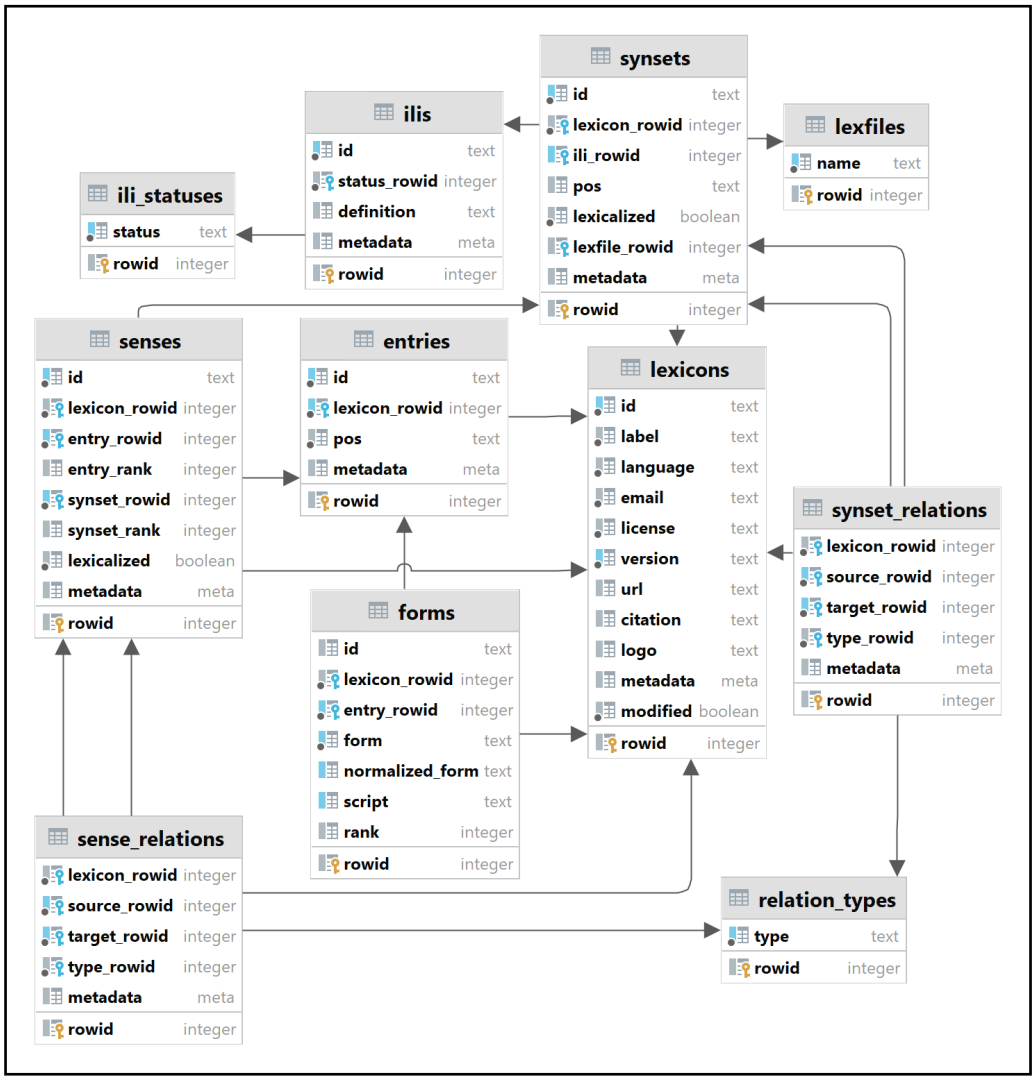
Portanto, o *OpenWN-PT* utilizado segue este *schema*, entretanto, muitas dessas tabelas não serão utilizadas, pois, possuem informações desnecessárias no contexto deste projeto. Ao contactar a comunidade responsável pelo pacote *wn*, os mesmos informaram não haver possibilidade de se alterar o *schema* dos bancos, como deletar uma tabela, pois o código está fortemente acoplado ao design do *schema*, e se o *schema* mudar; o comportamento do código ficará indefinido e pode gerar mensagens enigmáticas de erro.

[4] Um *schemma* de um sistema de banco de dados é sua estrutura descrita em uma linguagem formal suportada pelo sistema de gerenciamento de banco de dados e refere-se à organização de dados como um diagrama de como um banco de dados é construído.

A deleção destas tabelas faz com que seja necessário a construção de um novo software terceiro utilizado como dependência, mas, uma alternativa foi adotada: as tabelas que não necessárias tiveram seus conteúdos deletados, apenas restando suas estruturas utilizadas na verificação de *schemma* do pacote *wn*.

Para a análise da similaridade neste projeto é necessário apenas os dados contidos nas tabelas: *lexfiles*, *synsets*, *synsets_relations*, *relation_types*, *senses*, *sense_relations*, *entries*, *lexicons*, *forms*, *ilis* e *ili_statuses*. Sendo assim, o conteúdo das demais tabelas foram deletados. O diagrama do banco de dados com as entidades utilizadas no sistema pode ser visualizado na figura 19, seu conteúdo e sua interligação serão explicadas posteriormente.

Figura 19 – DER banco de dados das tabelas utilizadas



Fonte: Adaptado de (PAIVA; RADEMAKER; MELO, 2012)

A figura 19 exibe quais são as tabelas do sistema, elas são compostas de poucos atributos, como valor de identificação, referências as outras tabelas e valores relacionados a elas próprias; como, por exemplo, a tabela *synsets* que possui os *rowid* indicando sua

posição na tabela e o atributo id que armazena o valor do *synset*, além disto, as tabelas estão interligadas entre si conforme descrito na seção 4.4.1, informações detalhadas sobre cada uma das tabelas e seus atributos podem ser encontradas no apêndice A.

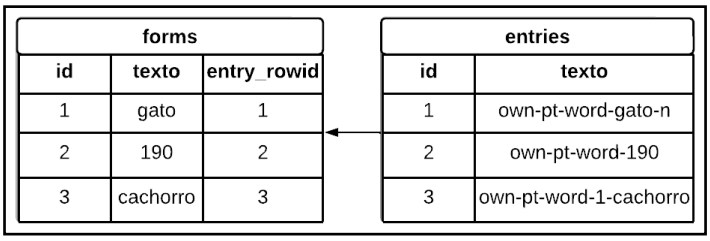
4.4.1 Organização do banco de dados

O conteúdo de cada tabela e como estão conectadas é descrito a seguir.

4.4.1.1 *Forms, entries, senses e synsets*

A tabela *forms* possui a forma canônica de uma entidade, sendo utilizada para começar as buscas por um dado informado pelo usuário, como pode ser visualizado na figura 20, a tabela *forms* possui uma referência a tabela *entries* assim referenciando a *entry* equivalente a ela naquela tabela.

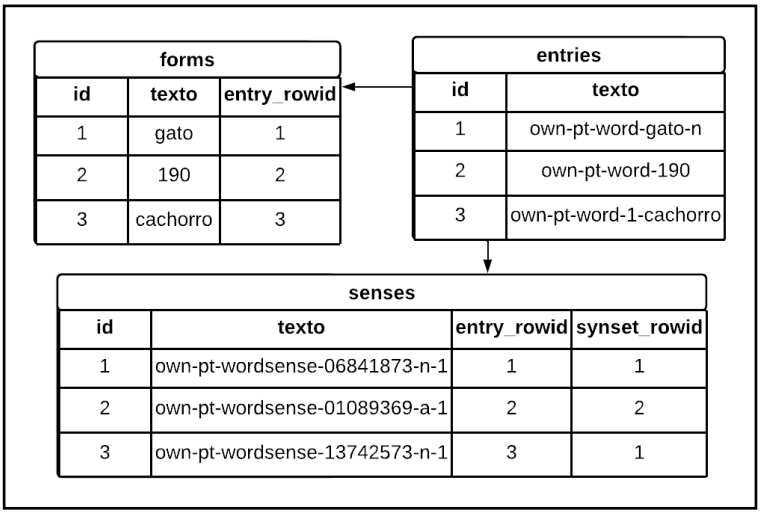
Figura 20 – Relação *forms* e *entries*



Fonte: Próprio autor

Para obter quais são os *senses* de determinada *entry*, basta ir à tabela *senses* e buscar quais são os *senses* relacionados com esta determinada *entry*.

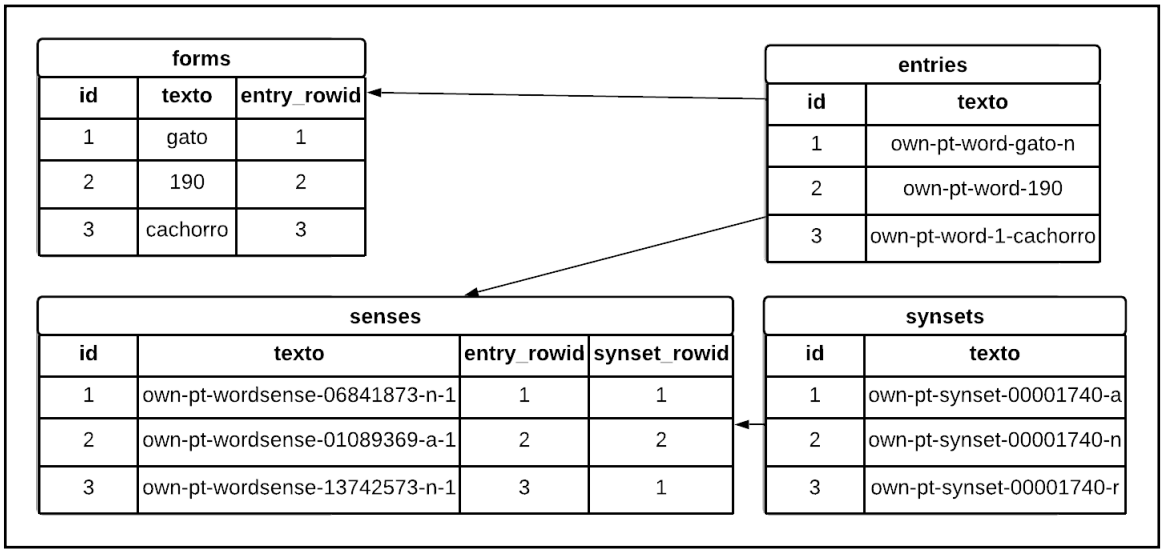
Figura 21 – Relação *senses* e *entries*



Fonte: Próprio autor

Para obter quais são os *synsets* de determinada *entry*, primeiro deve-se buscar quais são os *senses* relacionados com esta determinada *entry* e então encontra-se quais são os *synsets* relacionados com estes *senses*.

Figura 22 – Relação *sense* e *synset*



Fonte: Próprio autor

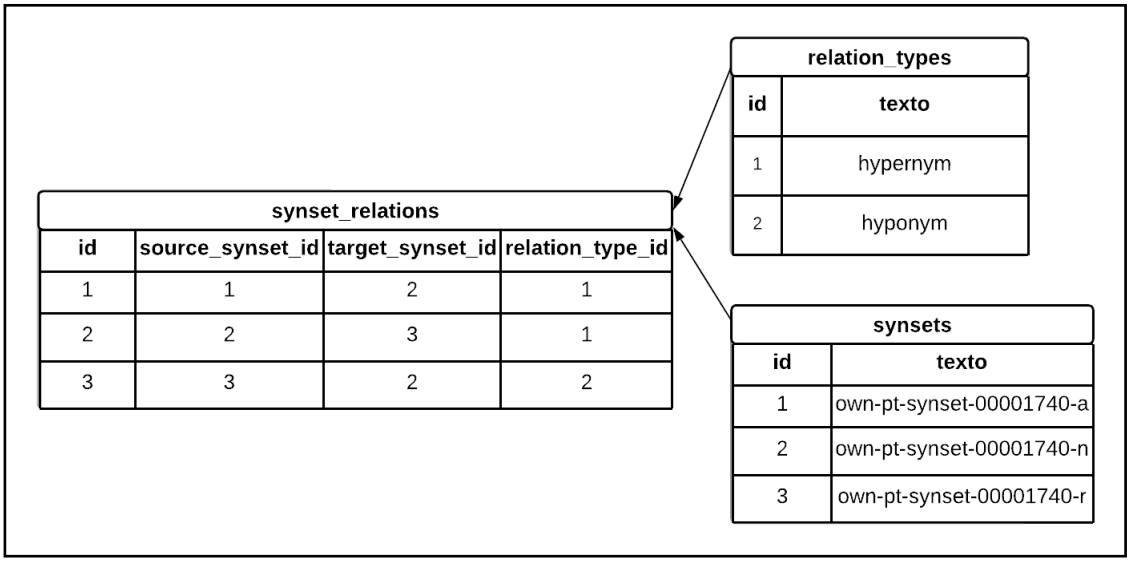
4.4.1.2 Synsets relations e senses relations

Os *synsets* e os *senses* estão relacionados entre si, devido a estes relacionamentos é possível realizar o cálculo de similaridade de *WuPalmer*. Neste banco os tipos de relacionamentos entre os *senses* e os *sysnsets* são diversos, mas os relevantes para este trabalho são os relacionamentos de:

- *hypernym*: relação de *hypernym* de um *synset/sense* sobre outro.
- *hyponym*: relação de *hyponym* de um *synset/sense* sobre outro.

que estão contidos na tabela *relation_types*, já que são estes os relacionamentos utilizados para calcular o *Least Common Subsumer* utilizado no cálculo da métrica de similaridade de *WuPalmer*, a tabela *synset_relations* possui referência as relações entre os *synsets* e os tipos de relações, como pode ver visualizado na figura 23, com isto é possível traçar uma árvore de um determinado *synset* utilizando suas relações de *hypernym* e *hyponym*.

Figura 23 – Relações entre os *synsets*



Fonte: Próprio autor

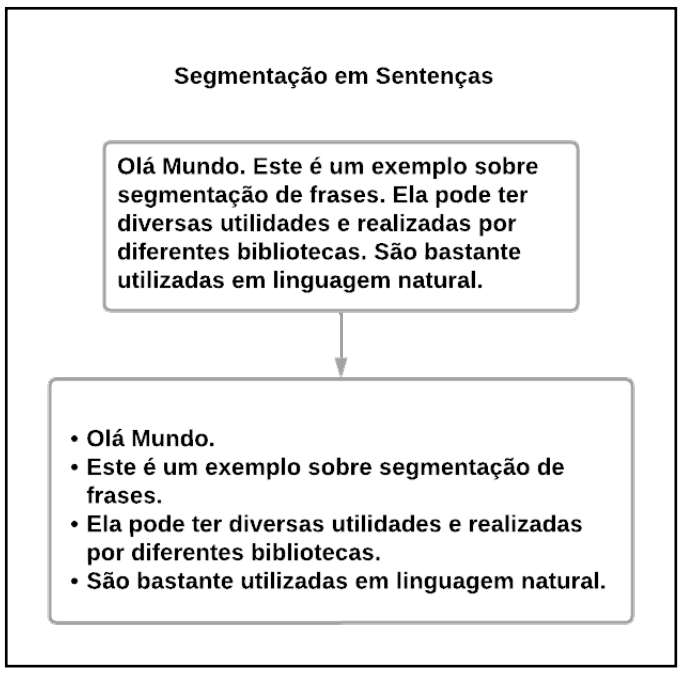
4.5 Processo de Análise dos documentos

O processo de análise dos documentos é divido em duas etapas, etapa textual envolvendo manipulação dos arquivos de entrada e etapa de análise envolvendo a análise do conteúdo destes arquivos.

Etapa Textual:

A primeira etapa consiste em receber as entradas do usuário e convertê-las em textos puros, para isto elas são enviadas para o módulo textual do sistema que invoca as bibliotecas *pdfminer* e *docx2txt* para extrair o conteúdo dos documentos caso estejam em outro formato diferente de txt. Na conclusão desta fase é obtido os textos das entradas, o texto é convertido para caixa baixa, então é feito a remoção das *stopwords* deles iterando o mesmo e verificando se possui alguma palavra da lista de *stopwords* fornecida pela biblioteca *spaCy*, após a remoção das *stopwords*, o texto é segmentado em sentenças utilizando a regex: [·!?] + [.!?]?, separando o texto pelos sinais de pontuação ponto, exclamação e interrogação, assim gerando uma segmentação por frases. A figura 24 ilustra um processo de segmentação utilizado nesta etapa.

Figura 24 – Processo de Segmentação em Sentenças



Fonte: Próprio autor

Etapa de Análise:

Há dois tipos de análise no projeto, análise rápida e profunda, elas se diferem no processo da análise das palavras, alterando a quantidade de synsets que cada uma verifica, na análise profunda a análise é aplicada a cada um dos synsets retornados de ambas sentenças, assim, realizando uma comparação NxN entre todos os synsets. Já na análise rápida é aplicada apenas para o primeiro synset de cada uma.

Acessando o banco de dados do sistema é possível montar a árvore de relações de uma determinada palavra, com esta árvore pode-se realizar o cálculo de *Least Common Subsumer* e então aplicar a métrica de similaridade de *WuPalmer* para calcular a similaridade entre um par de palavras. Obtendo a similaridade entre as palavras é possível realizar o cálculo da similaridade entre sentenças baseado nas palavras que as compõe e com a similaridade das sentenças pode-se realizar o cálculo da similaridade dos documentos.

O método de análise dos documentos recebe as sentenças dos documentos pela etapa textual e invoca o método de análise de sentenças enviando a ele o conjunto de sentenças recebidas para cada documento. O método de análise de sentenças analisa pares de sentenças buscando encontrar a similaridade entre os pares, para este processo é invocado o método de análise das palavras, enviando a ele as palavras que compõe o par de sentenças.

O método de análise das palavras ocorre analisando pares de palavras e armazenado o resultado dos mais semelhantes, para analisar um par de palavras primeiro é verificado

se elas possuem o mesmo *part of speech(POS)*, pois se uma palavra for um verbo e outra um substantivo ambas não são similares, assim não sendo necessário analisá-las, se as palavras possuem o mesmo *POS* deve-se acessar o banco de dados do sistema e pesquisar por elas na tabela *forms*, já que esta tabela possui a forma canônica da palavra, a tabela *forms* possui referência a tabela *entries*, assim referenciando a entidade equivalente a ela, como pode ser visualizado na figura 20, após encontrar a entidade equivalente à palavra é preciso buscar quais são os *senses* e *synsets* relacionados com esta entidade, isto pode ser feito acessando a tabela *senses* como exibe as figuras 21 e 22.

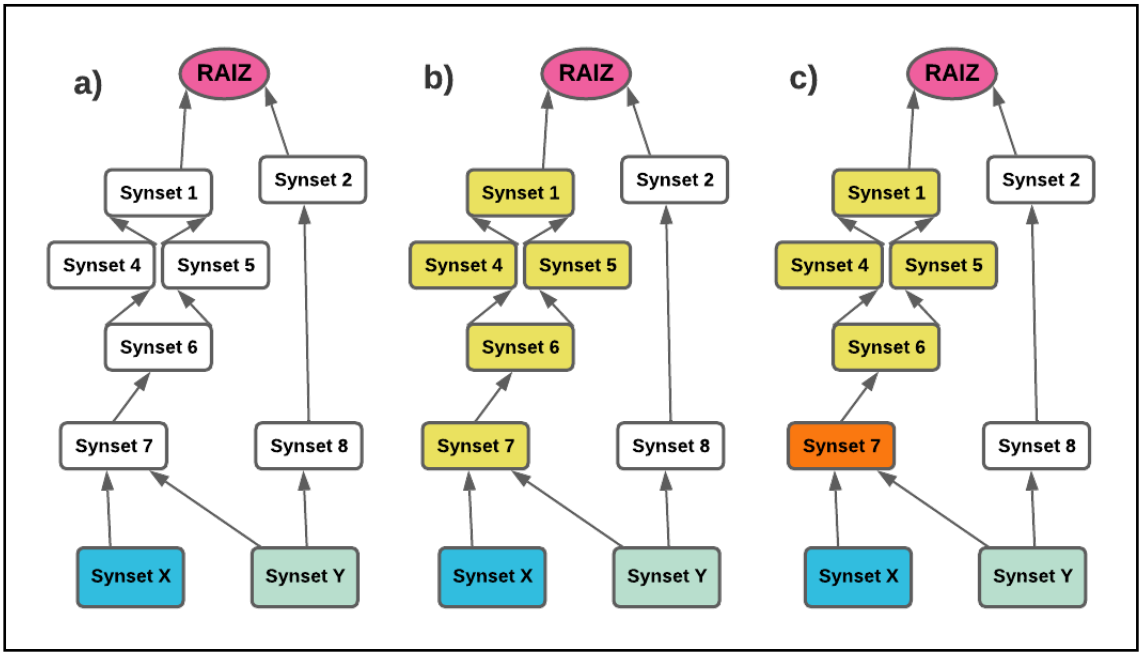
Obtido os *synsets* relacionados com uma determinada entidade, deve-se fazer o mapeamento dos hiperônimos dos *synsets*, isto é feito acessando a tabela *synset relations* e obtendo as relações de hiperônimo*(hypernym)* dos *synsets* em questão, como pode ser visualizado na figura 23, ao obter uma relação de hiperônimo de um *synset* com outro, navega-se ao outro e subsequente até chegar a raiz da árvore.

Com os hiperônimos dos *synsets* de ambas palavras calculados, é necessário calcular quais são os hiperônimos que eles possuem em comum, com os hiperônimos em comum calculados deve-se selecionar aquele mais distante da raiz, este será o *Least Common Subsumer*. A figura 25 exibe um exemplo deste processo, com os synsets analisados nas cores verde e azul, a raiz de vermelho, os hiperônimos em comum de amarelo e de laranja, o *Least Common Subsumer*.

Uma vez obtido o *Least Common Subsumer (LCS)* é possível aplicar a métrica de similaridade de *WuPalmer* descrita na seção 3.2. Onde i é a distância do synset1 ao *LCS*, j a do synset2 ao *LCS* e k a distância do *LCS* a raiz da árvore, feito isso é possível obter a similaridade entre estes dois *synsets*, também interpretado como a similaridade entre o par de palavras que pertencem a estes *synsets*.

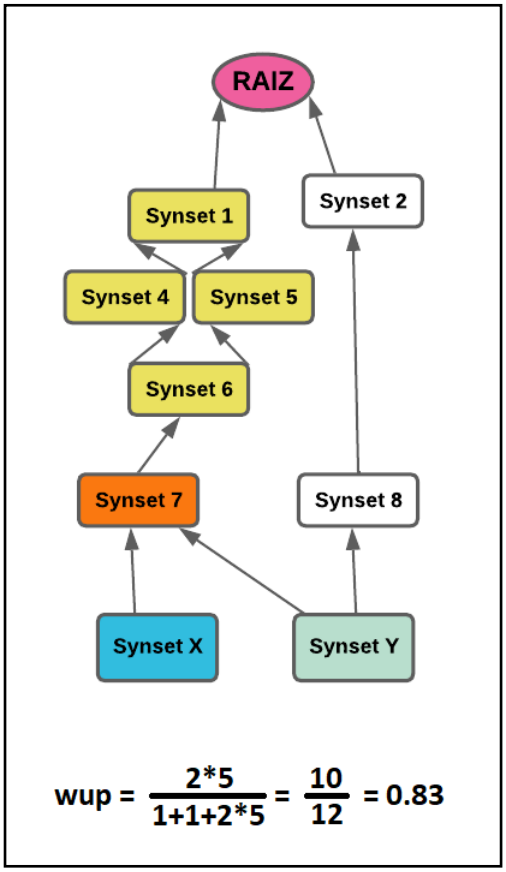
A figura 26 exibe o resultado da aplicação da métrica de similaridade de *WuPalmer* no exemplo dado na figura 25, onde i e j são 1 e k é 5, aplicando a equação da métrica resulta na similaridade de 0.83 para o exemplo dado. A análise rápida e profunda se diferem neste ponto, na rápida a análise é realizada apenas para o primeiro *synset* retornado do banco de cada uma das palavras, já na profunda ela é repetida para todos os *synset* retornados e armazenado o que gerar o maior resultado de similaridade da métrica de *WuPalmer*.

Figura 25 – Cálculo do *Least Common Subsumer*



Fonte: Próprio autor

Figura 26 – Cálculo da similaridade de *WuPalmer*



Fonte: Próprio autor

A análise da similaridade das palavras é realizada para todas as palavras da sentença A com as palavras da sentença B, como dito na equação 2.3 e após isto é aplicado a equação 2.5, que realiza a média da maior similaridade de cada palavra da sentença A em relação à sentença B, assim obtendo a similaridade da sentença A em relação à sentença B. Este processo é repetido para o outro lado da relação, para todas as palavras da sentença B comparadas com as palavras da sentença A aplicando a equação 2.4 e então a equação 2.6, obtendo a similaridade entre a sentença B em relação à sentença A, ao fim desta etapa é obtido tanto a relação da sentença A com a B quando da B com à A.

Com o resultado gerado na análise das sentenças é aplicada a equação 2.7, dado p como valor de um limiar, definido como p = 0,6, se ambos resultados das similaridades das sentenças obtiveram um valor superior a 0,6 as sentenças são contabilizadas como similares e armazenadas para serem exibidas futuramente no relatório de plágio.

Quando a análise de todas as sentenças é finalizada, o resultado é retornado ao método de análise de documentos e este aplica a equação 2.8, onde a semelhança entre o documento 1 e o documento 2 é dada por quantas sentenças similares o documento 1 possui em relação ao documento 2, após obter a relação do documento 1 com o documento 2, é aplicada a equação 2.9, desta vez analisando a relação do documento 2 com o documento 1.

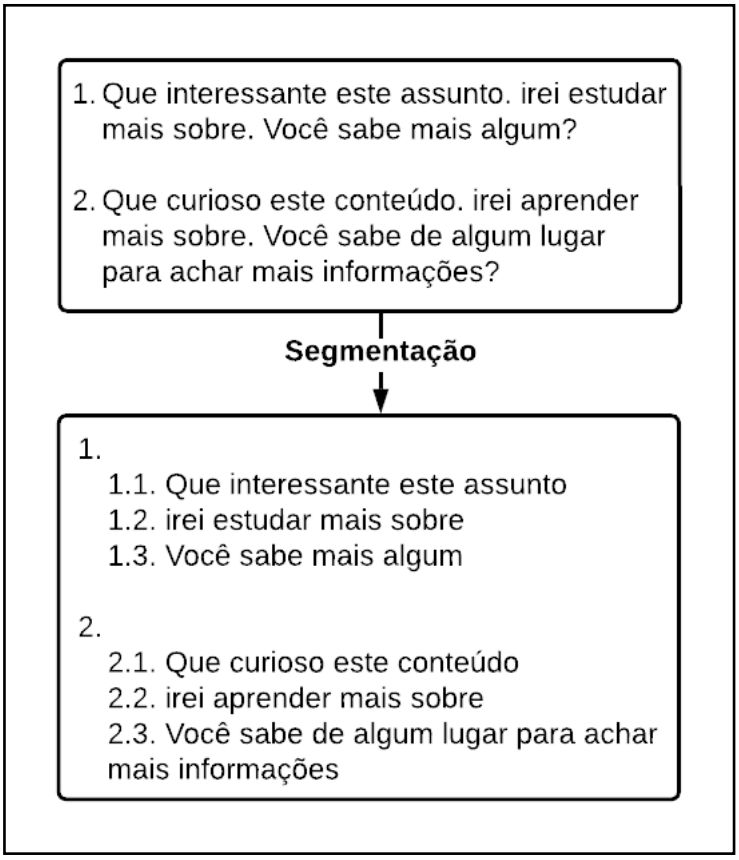
Obtido a similaridade dos documentos de ambos lados da relação, para gerar quão similar os documentos são, é aplicado a equação 2.1, ela irá sumarizar os resultados da semelhança entre os documentos em um só, onde P é dado pela multiplicação da similaridade dos documentos, com o resultado da equação é obtido o valor das *ODDS* então é utilizada a equação 2.2 e o resultado é convertido para probabilidade e então multiplicado por 100 para obter o valor em porcentagem, fornecendo a porcentagem da chance de existir plágio nos documentos.

Com esta metodologia é possível realizar uma análise entre diversos documentos entre si, invocando o método de análise de documentos para cada par de documentos. A seção abaixo demonstra um processo de análise seguindo os passos citados nesta seção.

Demonstração de um processo de análise:

A seguir é exibido um processo de análise rápida entre dois documentos utilizando os passos descritos acima, como a análise profunda repete o mesmo processo apenas alterando a quantidade de verificações ela não será demonstrada. Além disto, o exemplo a seguir visa dar um exemplo claro de como é o processo da análise da detecção de plágio, por isso foi utilizado um exemplo com arquivos pequenos, porém o sistema também trabalha com arquivos grandes. A figura 27 exibe dois documentos, seu conteúdo e o resultado da etapa de segmentação por frases neles.

Figura 27 – Conteúdo e segmentação dos textos

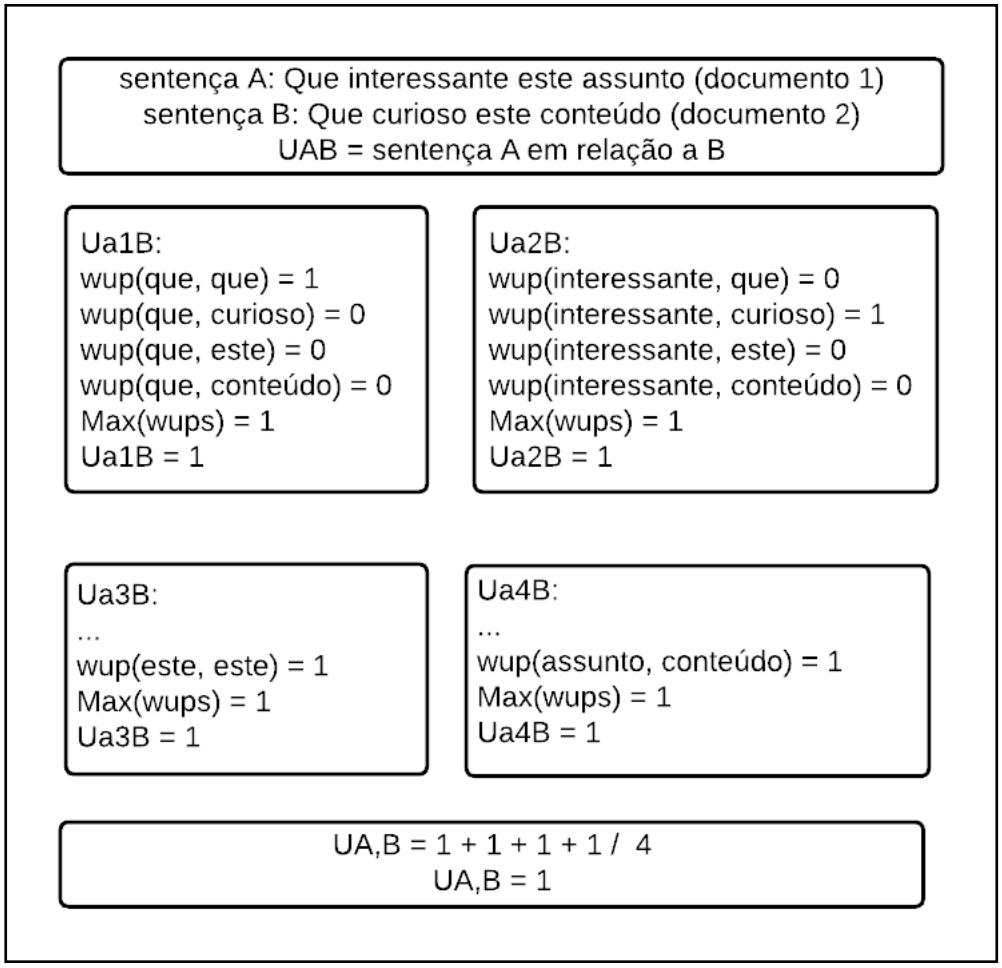


Fonte: Próprio autor

Pela figura 28 é demonstrado como é realizado o processo de análise de duas sentenças, uma pertencente ao documento 1 e outra ao documento 2, onde a sentença A é referente ao documento 1 e a sentença B ao documento 2. É aplicado o cálculo da métrica de similaridade *WuPalmer(wup)* para cada palavra de uma sentença e obtido a palavra que possuir a maior similaridade em relação a ela na outra sentença.

O *wup*(palavra1, palavra2) demonstra o processo da aplicação da métrica *WuPalmer* nas palavras 1 e 2 e o resultado obtido, *Ua1B* indica a maior similaridade da palavra 1 da sentença A em relação a todas as palavras da sentença B, *Ua2B* da palavra 2 da sentença A e o mesmo para as demais palavras, conforme dito na equação 2.3. Ao realizar o cálculo de cada palavra da sentença A em relação à sentença B é aplicada a equação 2.5 fornecendo o *UA,B*, este sendo a similaridade da sentença A em relação à B baseada nas palavras que as compõe.

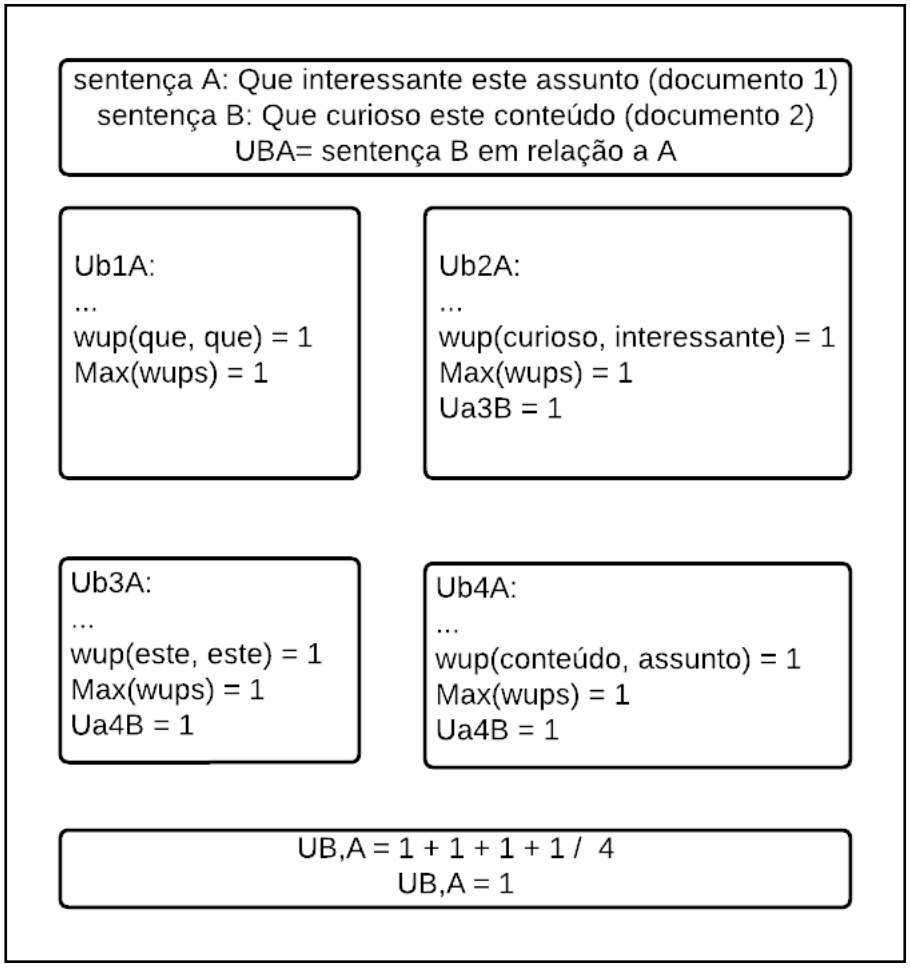
Figura 28 – Análise da sentença A em relação à B pelas suas palavras



Fonte: Próprio autor

Também é necessário realizar a análise da sentença B em relação à A, para isto são aplicadas as equações 2.4 e 2.6, a figura 29 exibe a análise da sentença B em relação à A e o *UB,A* é a similaridade da sentença B em relação à A.

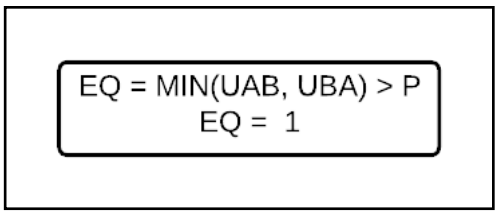
Figura 29 – Análise da sentença B em relação à A pelas suas palavras



Fonte: Próprio autor

Obtidos os UA,B e UB,A, é aplicado a equação 2.7, caso a similaridade de ambas sentenças for superior ao limiar P que é 0,6, elas são contabilizadas como similares, como exibe a figura 30.

Figura 30 – EQ

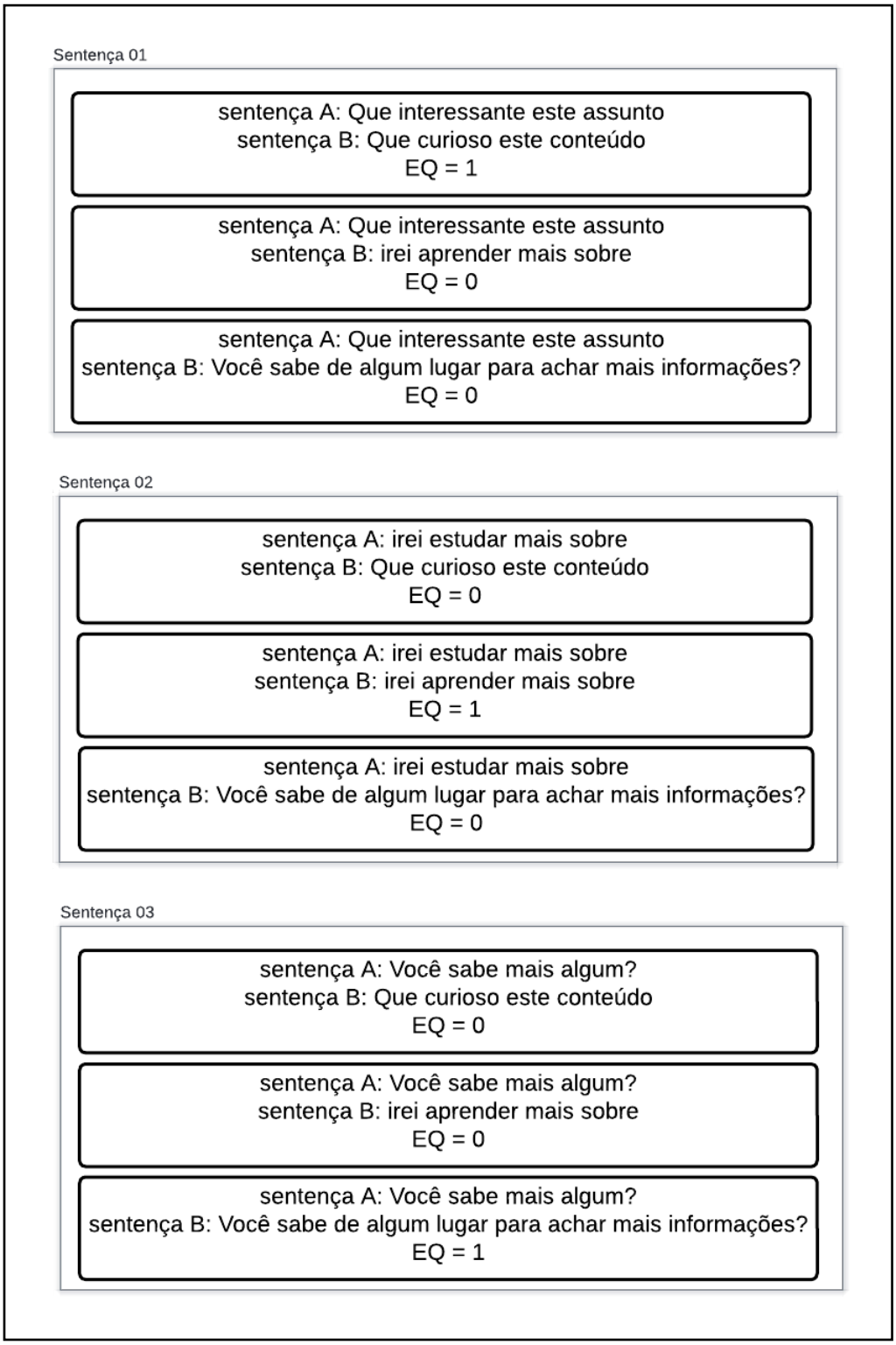


Fonte: Próprio autor

Sendo assim, é necessário realizar este processo para as demais sentenças que compõe os documentos, a figura 31 já exibe o resultado EQ da equação 2.7, nas sentenças

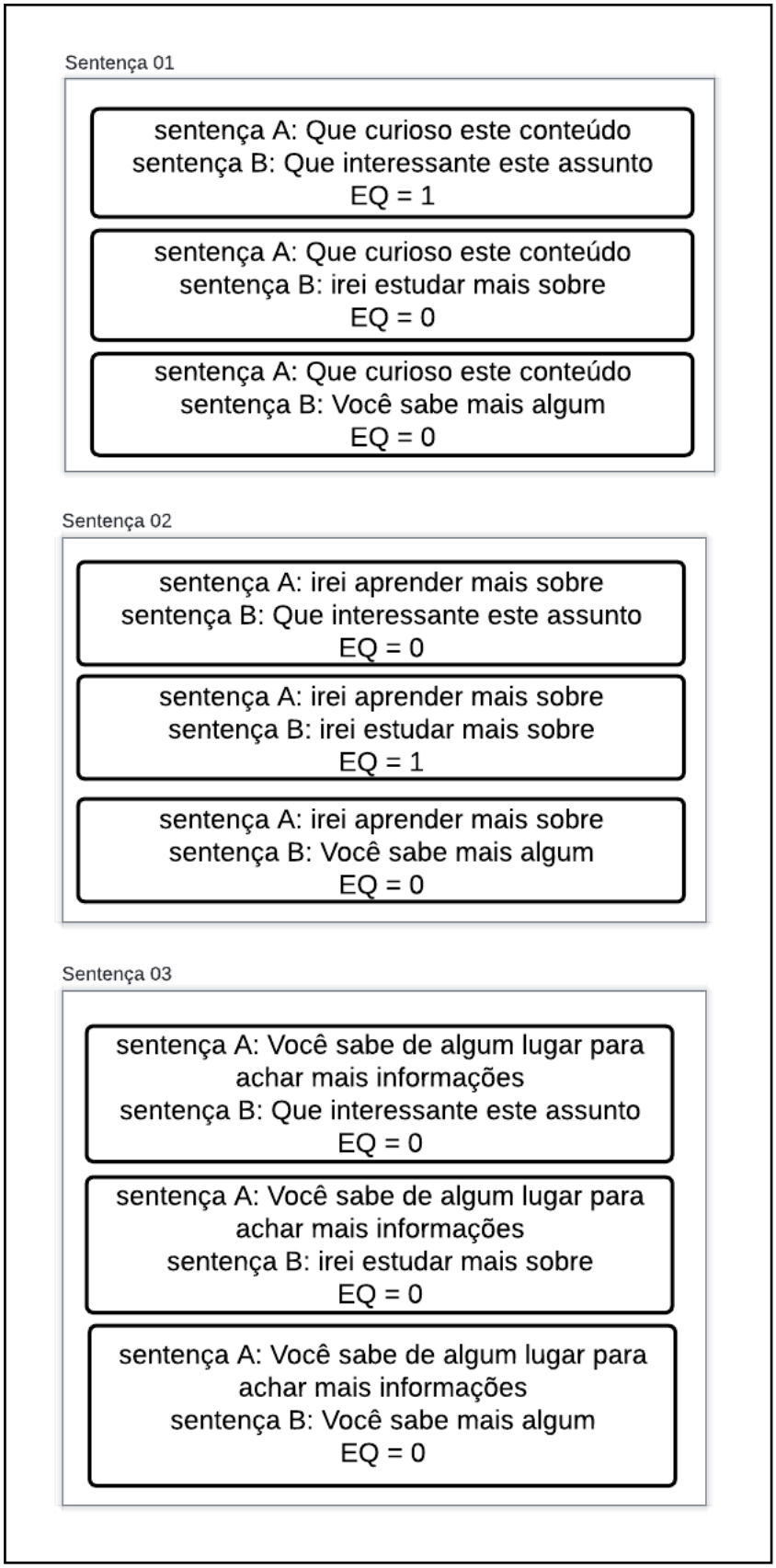
do documento 1 em relação às sentenças do documento 2 e a figura 32 exibe o resultado *EQ* das sentenças do documento 2 em relação às sentenças do documento 1.

Figura 31 – Resultado das demais sentenças do documento 1 em relação ao 2



Fonte: Próprio autor

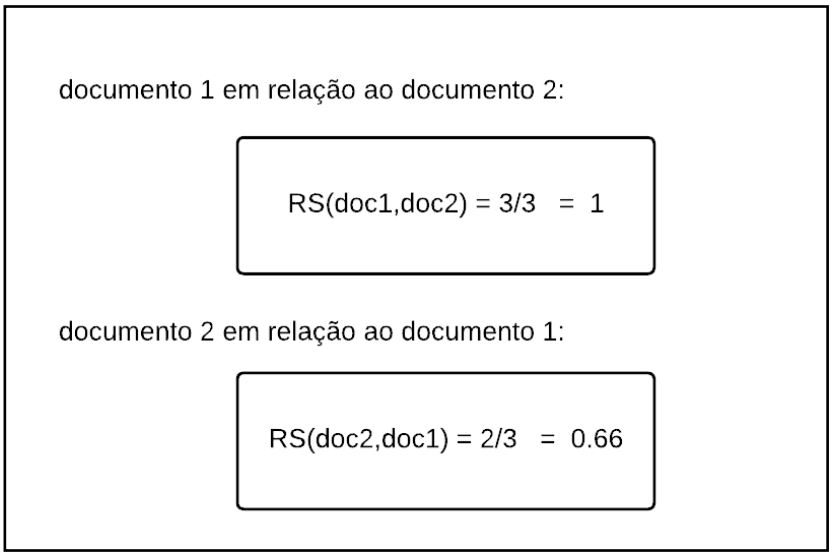
Figura 32 – Resultado das demais sentenças do documento 2 em relação ao 1



Fonte: Próprio autor

Calculado as similaridades entre as sentenças dos documentos, deve-se aplicar as equações 2.8 e 2.9 que contabilizaram a quantidade de sentenças semelhantes do documento 1 em relação do documento 2, dado por *RS(doc1,doc2)* e do documento 2 em relação ao documento 1 dado por *RS(doc2,doc1)*, exibidas pela figura 33.

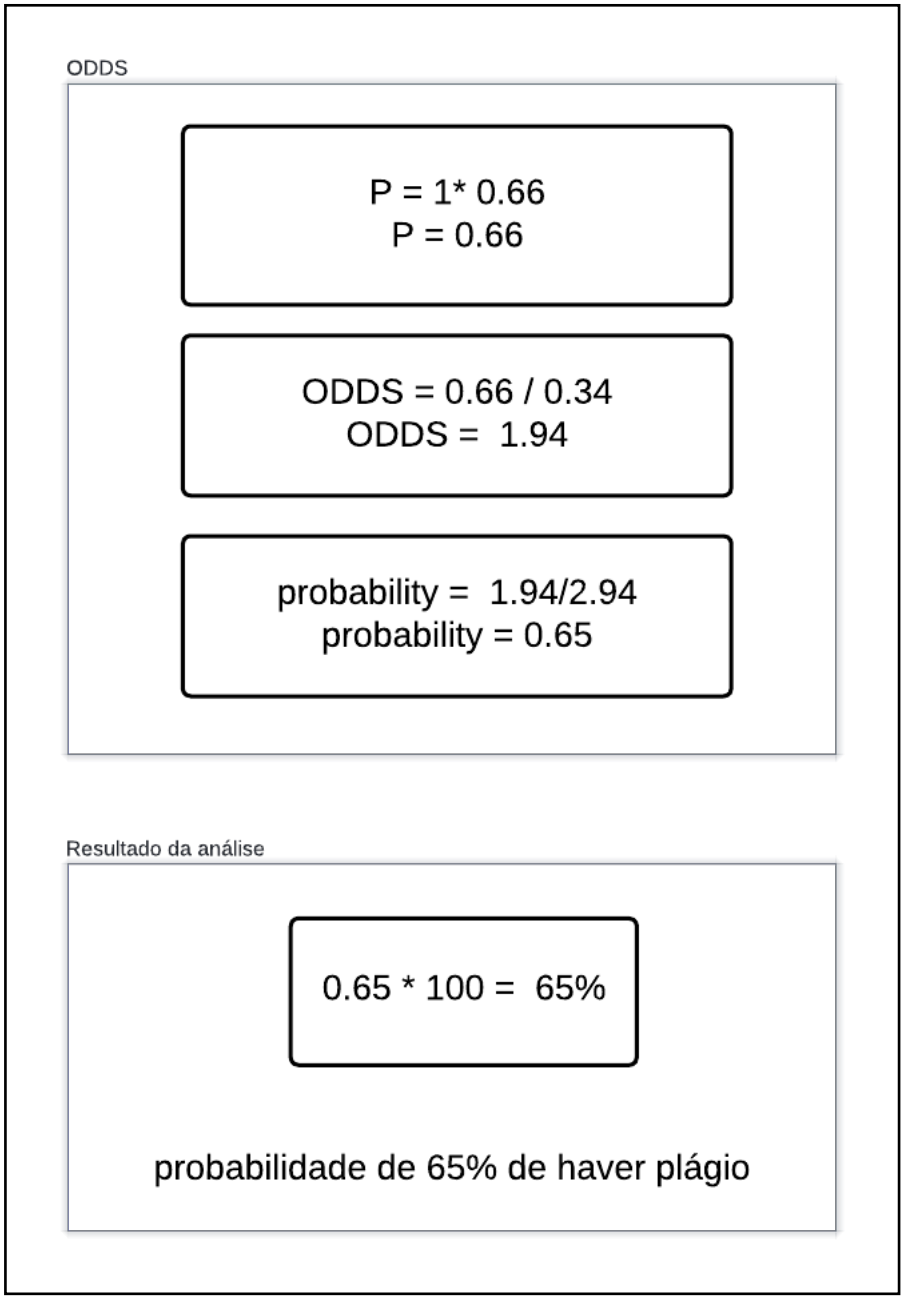
Figura 33 – Calculando a similaridade entre os documentos com base nas sentenças



Fonte: Próprio autor

Após esta etapa, basta aplicar a equação 2.1 para obter a probabilidade de plágio entre os documentos, o P é dado pela multiplicação da semelhança entre os documentos fornecida em *RS(doc1,doc2)* e *RS(doc2,doc1)*, ao aplicar a equação é obtido o valor das *ODDS*, então é utilizada a equação 2.2 e o resultado é convertido para probabilidade e multiplicado por 100 para obter o valor em porcentagem, assim gerando a probabilidade de haver plágio nestes documentos, a figura 34 exibe estes passos.

Figura 34 – Resultado da análise de plágio entre os dois documentos



Fonte: Próprio autor

4.6 Avaliação do método de análise

A validação do método de detecção abordado utilizou a metodologia descrita em 3.2, a validação é realizada nos tipos de detecção rápida e profunda utilizados em um conjunto de dados composto de 128 arquivos. A maioria dos experimentos nas validações do método de detecção utilizaram arquivos pequenos, uma vez que o processo de análise é o mesmo para arquivos grandes ou pequenos. Sendo assim, a análise em arquivos pequenos

é suficiente para medir a habilidade da detecção do sistema, mas também foram realizados experimentos em arquivos grandes. Também vale destacar que o tamanho dos arquivos impactam no tempo de um processo de análise, porém, este parâmetro não foi medido, uma vez que o foco dos experimentos estava na habilidade do sistema detectar o caso de plágio.

Os dados da validação podem ser visualizados na tabela 7, onde A é a quantidade de análises realizada, N a quantidade de arquivos analisados, D plágios detectados corretamente, ND a quantidade de plágios não detectados, DF a quantidade de detecções falsas e P a quantidade de casos anotados como plágio. O resultado da validação podem ser visualizados na tabela 8 onde P indica a precisão da detecção analisando a quantidade de detecções corretas e as detecções falsas, R indica a cobertura alisando a quantidade de detecções corretas e as não detectadas, G indica a granularidade da detecção, F é a média harmônica e pontuação o resultado da qualidade da detecção.

Tabela 7 – Dados para validação das técnicas de detecção de plágio

Tipo de Análise	**A**	**N**	**D**	**ND**	**DF**	**P**
Rápida	64	128	51	12	1	63
Profunda	64	128	59	4	2	63

Tabela 8 – Resultados da validação das técnicas de detecção de plágio

Tipo de Análise	**P**	**R**	**G**	**F**	**Pontuação**
Rápida	0,9807	0,8095	1.0000	0,8868	**0,8868**
Profunda	0,9672	0,9365	1,0000	0,9515	**0,9515**

4.7 Discussões sobre os resultados

Realizando análises em um conjunto de 128 arquivos utilizando a metodologia proposta na seção 3.2 foi obtido para o valor de precisão na análise rápida e profunda, respectivamente, 0,9807 e 0,9672, o que indica que a análise rápida possui uma precisão maior que a profunda, isto é causado, pois a precisão é dada pela quantidade de análises corretas sobre a quantidade de análises corretas e detecções falsas, a análise profunda gera mais falsos positivos por analisar cada *synset* que compõe a palavra, assim obtendo um resultado pior no quesito precisão.

Já em relação à cobertura os resultados para análise rápida e profunda foram 0,8095 e 0,9365, este é dado pela quantidade de detecções corretas sobre a quantidade de corretas e as não detectadas, como a análise profunda realiza uma verificação em mais *synsets* que a rápida, a profunda obtêm um resultado superior no quesito cobertura.

A granularidade de ambas detecções é de 1, visto que há um controle no sistema para fornecer uma granularidade de 1. A média harmônica da análise rápida e profunda é

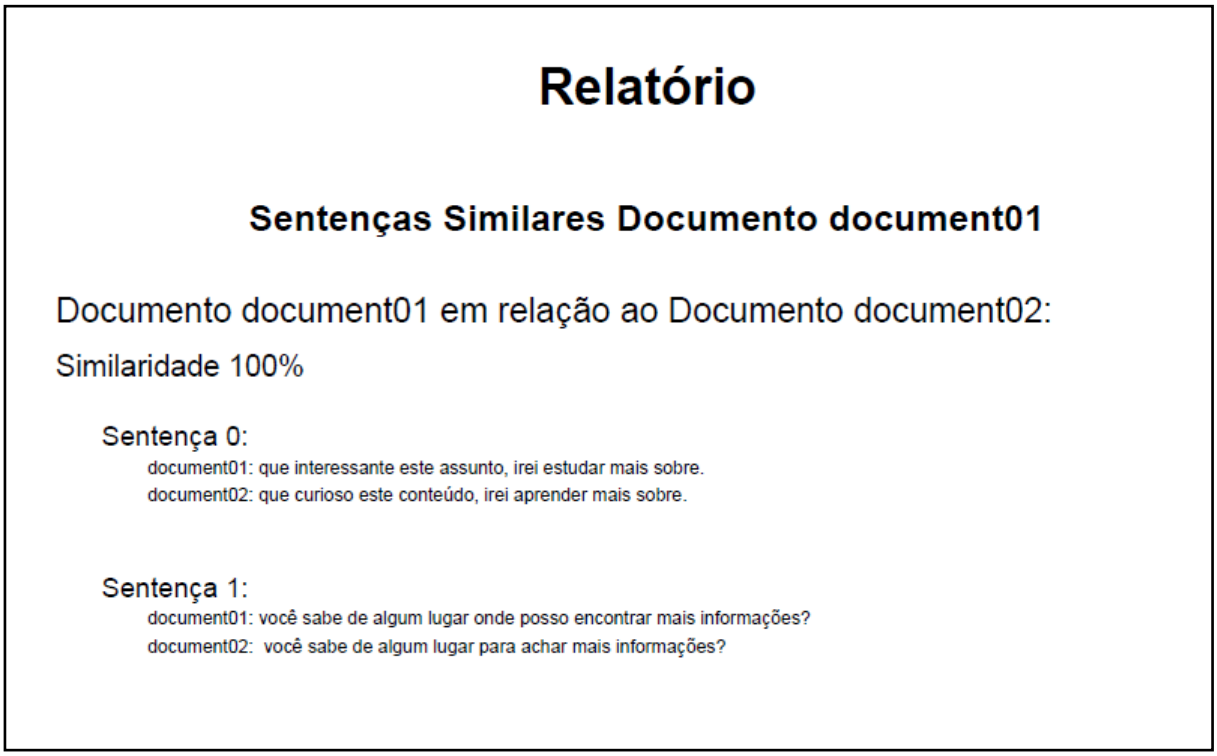
dada pelos valores de precisão e cobertura foram de 0,8868 e 0,9515, por consequente como a granularidade é de 1, a pontuação das análises também foi de 0,8868 e 0,9515. Assim, baseado nos valores das pontuações, podemos concluir que a análise profunda fornece uma qualidade de detecção melhor em relação a rápida.

Ademais, comparado a literatura, a pontuação obtido para ambas análises se mostra satisfatório, pois no trabalho Alzahrani, Salim e Palade (2015) que utiliza a mesma metodologia para validação da qualidade de detecção, o resultado da pontuação para métodos de segmentação baseado em sentenças, que utilizam metodologias *semânticas fuzzy* como no caso de *WuPalmer* foi de 0,7850. Entretanto, deve haver ressalvas na comparação desta pontuação com as pontuações obtidas no trabalho aqui presente, uma vez que esta comparação não é direta, já que estas validações foram realizadas em conjuntos de dados distintos, pois não havia a possibilidade de se utilizar os mesmos conjuntos de dados.

4.8 Geração de *PDF*s

Com a análise realizada, os *pdfs* com os relatórios relativos aos documentos são criados, utilizando a *reportlabpdf* e enviados ao sistema, ficando disponíveis ao usuário para baixar ou receber via *e-mail*, a figura 35 demonstra um exemplo de relatório gerado.

Figura 35 – Relatório PDF



Fonte: Próprio autor

4.9 Envio de *e-mails*

Com o relatório da análise gerado, caso o usuário deseje, o mesmo é enviado ao seu *e-mail*, utilizando o servidor *SMTP Sendinblue*.

5 Conclusão

Neste trabalho, foi desenvolvida e apresentada um protótipo de uma aplicação detectora de plágio, que visa fornecer aos usuários uma detecção dos seus arquivos de forma amigável. O protótipo apresentado consegue efetuar a conversão de arquivos de entrada em formatos *pdf*, *txt*, *docx* para texto, de realizar análises de múltiplos arquivos simultaneamente e de enviar o resultado ao *e-mail* do usuário. Somando essas características, há uma interface intuitiva que visa facilitar todos os processos para os usuários, disponibilizando de opções de exemplos, bem como o resultado da análise exibida como um gráfico e um relatório com o conteúdo da análise.

Pelo fato de ser um sistema *web*, a utilização deste sistema pode ser realizada em qualquer lugar ou dispositivo, visto que basta apenas um navegador e acesso à internet. Além disso, a utilização de novas tecnologias, como *React* e *Django*, reforçam a segurança do sistema e melhoram o desenvolvimento do código dado as facilidades que cada um apresenta.

5.1 Escalabilidade do sistema

O sistema está moldado para ser escalável, na versão atual devido aos planos dos servidores utilizados, *Heroku* e *Sendinblue*, o sistema está atendendo a 50 usuários, suportando análises de arquivos de até 50 *megabytes* totais e até 3 análises simultâneas, podendo enviar 300 *e-mails* diários, o que é suficiente para este protótipo. Esta informação consta na documentação dos respectivos servidores utilizados, além das mesmas, terem sido validadas pelo autor. Sendo assim, para escalar o sistema basta adquirir servidores mais robustos.

5.2 Pontos fracos e possíveis melhorias

Existem caminhos alternativos para o desenvolvimento deste trabalho que podem resultar em uma melhor análise de plágio, porém não há garantias que sejam melhores, a seguir são exibidos os principais.

O método e métrica utilizados para a análise: a escolha de um método baseado em semântica *fuzzy* usando a métrica *WuPalmer* se deu baseado em comparações entre os métodos realizados em outros estudos como exibido em 3.2 e sua disponibilidade, porém existem outros métodos de análise e métricas além do aqui abordado, alguns baseados em semântica *fuzzy* como o (LEACOCK; CHODOROW, 1995) e outros baseados em outros

métodos como os baseados em caracteres, esta alteração pode influenciar no resultado da análise, bem como no tempo de processamento da mesma.

O método de processamento textual: neste projeto foi utilizado a segmentação por sentenças, porém existem outros tipos de segmentações textuais como as por *W3G*, *W5G* e *W8G* que podem ser encontradas nos trabalhos de (EZZIKOURI ERRITALI, 2017), esta alteração também pode influenciar no resultado da análise e no tempo de processamento desta.

O banco de dados utilizado: foi utilizado o banco de dados léxico (PAIVA; RADEMAKER; MELO, 2012) devido a sua disponibilidade, uso público e documentação, entretanto existem outros bancos léxicos para a língua portuguesa e a utilização de outro pode influenciar no desenvolvimento do projeto e no resultado da análise.

Além disto, o protótipo desenvolvido possui alguns pontos fracos relacionados com os servidores utilizados e o banco de dados detalhados na seção 5.3.

5.3 Trabalhos futuros

O protótipo desenvolvido possui alguns pontos que podem sem desenvolvidos em trabalhos futuros, esta seção exibe os principais.

Adição de servidores mais robustos: o protótipo possui uma limitação relacionada com o servidor que ele utiliza para realizar os cálculos da análise, este não suporta mais que três análises simultâneas e arquivos maiores que 50 *megabytes* no total por não possuir memória disponível para tal, sendo assim um possível ponto de desenvolvimento no futuro.

Inserção de novas palavras no banco de dados: como a construção de um banco de dados léxico não fez parte do escopo deste trabalho, algumas palavras podem não estar contidas no banco utilizado, o que pode influenciar em alguma análise, fazendo com que seja um possível ponto de desenvolvimento no futuro.

Referências

ADONAI. *Processo Unificado*. 2011. <https://www.adonai.eti.br/2011/06/processo-unificado-pu-unified-process/>. Acessado em: 2022-08-20. Citado na página 18.

ADVISORS quick. *What is Wu Palmer similarity*. 2022. Acessado em: 2022-08-21. Disponível em: <https://quick-advisors.com/what-is-wu-palmer-similarity/>. Citado 2 vezes nas páginas 23 e 35.

ALZAHRANI, S. M.; SALIM, N.; PALADE, V. Uncovering highly obfuscated plagiarism cases using fuzzy semantic-based similarity model. *Journal of King Saud University - Computer and Information Sciences*, v. 27, n. 3, p. 248–268, 2015. ISSN 1319-1578. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1319157815000361>. Citado 6 vezes nas páginas 16, 20, 24, 25, 36 e 61.

ANDERSON, M. S.; STENECK, N. H. The problem of plagiarism. *Urologic Oncology: Seminars and Original Investigations*, v. 29, n. 1, p. 90–94, 2011. ISSN 1078-1439. Plagiarism. Disponível em: <https://www.sciencedirect.com/science/article/pii/S107814391000270X>. Citado na página 16.

APSLETRASUNIP. *Você sabe o que são hiperônimos e hipônimos?* 2022. Acessado em: 2022-08-01. Disponível em: <https://apsletrasunip.files.wordpress.com/2019/05/hiperonimo.png>. Citado na página 21.

ARCHIVE web. *So I'm writing some scripts to try to track things a whole lot faster*. 2005. <https://web.archive.org/web/20190701210808/https://marc.info/?l=linux-kernel>. Acessado em: 2022-07-07. Citado na página 31.

ASTAH. *Astah*. 2021. <https://astah.net>. Acessado em: 2022-07-05. Citado na página 32.

AXIOS. *Axios*. 2022. <https://axios-http.com/>. Acessado em: 2022-09-18. Citado na página 31.

BRASILESCOLA. *Hiponimos*. 2022. <https://brasilescola.uol.com.br/gramatica/hiponimos-hiperonimos.htm>. Citado na página 21.

BRIN, S.; DAVIS, J.; GARCÍA-MOLINA, H. Copy detection mechanisms for digital documents. In: *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*. New York, NY, USA: Association for Computing Machinery, 1995. (SIGMOD '95), p. 398–409. ISBN 0897917316. Disponível em: <https://doi.org/10.1145/223784.223855>. Citado na página 19.

COELHO, L. et al. Embedding nomlex-br nominalizations into openwordnet-pt. *GWC 2014: Proceedings of the 7th Global Wordnet Conference*, p. 378–382, 01 2014. Citado na página 26.

COPYLEAKS. *CopyLeaks*. 2022. <https://app.copyleaks.com/text-compare>. Acessado em: 2022-08-06. Citado na página 28.

DJANGOPROJECT. *Django Project*. 2022. <https://www.djangoproject.com/start/overview/>. Acessado em: 2022-07-02. Citado na página 30.

DOCX2TXT. *docx2txt*. 2022. <https://pypi.org/project/docx2txt/>. Acessado em: 2022-09-18. Citado na página 31.

ECHARTS. *Echarts*. 2022. <https://echarts.apache.org>. Acessado em: 2022-09-18. Citado na página 29.

ES tc39. *ECMAScript 2020 Language Specification*. 2020. <https://tc39.es/ecma262/#sec-overview>. Acessado em: 2022-07-01. Citado na página 29.

ESEMFOCO. *Estrutura do Processo Unificado*. 2006. <http://es-emfoco.blogspot.com/2006/08/estrutura-do-processo-unificado.html>. Acessado em: 2022-08-20. Citado na página 19.

EZZIKOURI ERRITALI, O. Fuzzy-semantic similarity for automatic multilingual plagiarism detection. *International Journal of Advanced Computer Science and Applications*, 2017. Citado na página 64.

EZZIKOURI, H. et al. Fuzzy cross language plagiarism detection (arabic-english) using wordnet in a big data environment. In: *Proceedings of the 2018 2nd International Conference on Cloud and Big Data Computing*. New York, NY, USA: Association for Computing Machinery, 2018. (ICCBDC'18), p. 22–27. ISBN 9781450364744. Disponível em: <https://doi.org/10.1145/3264560.3264562>. Citado na página 20.

FIGMA. *figma*. 2021. <https://www.figma.com>. Acessado em: 2022-10-05. Citado na página 32.

FRAMEWORK django-rest. *django-rest-framework*. 2022. <https://www.django-rest-framework.org/>. Acessado em: 2022-09-18. Citado na página 30.

GANESAN kavita. *What are Stop Words*. 2022. Acessado em: 2022-08-21. Disponível em: <https://kavita-ganesan.com/what-are-stop-words/#.YwJRBxzMLGI>. Citado na página 22.

GITHUB. *GitHub Copyright*. 2021. <https://github.com/github>. Acessado em: 2022-07-05. Citado na página 31.

GLOBAL igi. *What is Least Common Subsumer*. 2022. Acessado em: 2022-08-21. Disponível em: <https://www.igi-global.com/dictionary/least-common-subsumer-lcs/41765>. Citado na página 23.

GRAPHPAD. *What is the difference between odds and probability*. 2021. <https://www.graphpad.com/support/faq/probability-vs-odds/>. Acessado em: 2022-08-20. Citado na página 24.

GUNICORN. *gunicorn*. 2022. <https://gunicorn.org/>. Acessado em: 2022-09-18. Citado na página 31.

HEADERS django-cors. *django-cors-headers*. 2022. <https://pypi.org/project/django-cors-headers/>. Acessado em: 2022-09-18. Citado na página 30.

HEROKU. *Documentation - Getting Started*. 2022. Heroku. Disponível em: <https://devcenter.heroku.com/categories/reference>. Acesso em: 2 fev 2022. Citado na página 30.

HEROKU django. *django-heroku*. 2022. <https://pypi.org/project/django-heroku/>. Acessado em: 2022-09-18. Citado na página 30.

HEUSER, C. A. *Projeto de Banco de Dados*. Porto Alegre: Bookman, 2009. Citado na página 30.

ITHENTICATE. *Ithenticate*. 2022. <https://www.ithenticate.com/>. Acessado em: 2022-08-06. Citado na página 28.

JAVASCRIPT. *JavaScript*. 2022. <https://www.javascript.com/>. Acessado em: 2022-09-18. Citado na página 29.

JIANG, J. J.; CONRATH, D. W. Semantic similarity based on corpus statistics and lexical taxonomy. In: *Proceedings of the 10th Research on Computational Linguistics International Conference*. Taipei, Taiwan: The Association for Computational Linguistics and Chinese Language Processing (ACLCLP), 1997. p. 19–33. Disponível em: <https://aclanthology.org/O97-1002>. Citado 2 vezes nas páginas 23 e 33.

JURAFSKY, D. et al. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. Second edition, pearson international edition. Upper Saddle River, NJ: Prentice Hall, Pearson Education International, 2009. (Prentice Hall series in artificial intelligence). Citado na página 20.

KRAFT, D. H.; BUELL, D. A. Fuzzy sets and generalized boolean retrieval systems. *Int. J. Man Mach. Stud.*, v. 19, p. 45–56, 1983. Citado na página 19.

LEACOCK, C.; CHODOROW, M. Filling in a sparse training space for word sense identification. In: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*. Montreal Quebec Canada: Mathematics, 1995. p. 448–453. Citado 3 vezes nas páginas 23, 33 e 63.

LESK, M. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In: *Proceedings of the 5th Annual International Conference on Systems Documentation*. New York, NY, USA: Association for Computing Machinery, 1986. (SIGDOC '86), p. 24–26. ISBN 0897912241. Disponível em: <https://doi.org/10.1145/318723.318728>. Citado na página 33.

LIN, D. Principle-based parsing without overgeneration. In: *Proceedings of the 31st Annual Meeting on Association for Computational Linguistics*. USA: Association for Computational Linguistics, 1993. (ACL '93), p. 112–120. Disponível em: <https://doi.org/10.3115/981574.981590>. Citado 2 vezes nas páginas 23 e 33.

MARC. *serious inflate inconsistency*. 2007. <https://marc.info/?l=git&m=118143549107708>. Acessado em: 2022-07-07. Citado na página 31.

MAURER, H.; KAPPE, F.; ZAKA, B. Plagiarism – a survey. *Journal of Universal Computer Science*, Verlag der Technischen Universität Graz, v. 12, n. 8, p. 1050–1084, 2006. ISSN 0948-695X. Citado na página 16.

MCINNES, B.; PEDERSEN, T. Evaluating measures of semantic similarity and relatedness to disambiguate terms in biomedical text. *Journal of biomedical informatics*, v. 46, 09 2013. Citado 2 vezes nas páginas 33 e 34.

MOZILLA developer. *CSS developer guide*. 2015. <https://developer.mozilla.org/en-US/docs/Learn/CSS>. Acessado em: 2022-06-23. Citado na página 29.

MUI. *Material UI*. 2022. <https://mui.com/pt/material-ui/getting-started/overview/>. Acessado em: 2022-07-07. Citado na página 29.

MUNDOEDUCACAO. *Hiperonimos*. 2022. <https://mundoeducacao.uol.com.br/gramatica/hiponimos-hiperonimos-holonimos-meronimos.htm>. Acessado em: 2022-06-22. Citado na página 21.

NGUYEN, H.; AL-MUBAID, H. New ontology-based semantic similarity measure for the biomedical domain. In: *2006 IEEE International Conference on Granular Computing*. Atlanta, GA, USA: IEEE, 2006. p. 623–628. Citado na página 33.

OGAWA, Y.; MORITA, T.; KOBAYASHI, K. A fuzzy document retrieval system using the keyword connection matrix and a learning method. *Fuzzy Sets and Systems*, v. 39, n. 2, p. 163–179, 1991. ISSN 0165-0114. Applications of fuzzy systems theory. Disponível em: <https://www.sciencedirect.com/science/article/pii/016501149190210H>. Citado na página 19.

ORG w3. *What is CSS*. 2011. <https://www.w3.org/standards/webdesign/htmlcss#whatcss>. Acessado em: 2022-07-05. Citado na página 29.

ORG w3. *HTML 4.0 Specification W3C Recommendation Conformance: requirements and recommendations*. 2015. <https://www.w3.org/TR/REC-html40-971218/conform.html#deprecated>. Acessado em: 2022-07-05. Citado na página 29.

PAIVA, V. de; RADEMAKER, A.; MELO, G. de. Openwordnet-pt: An open Brazilian Wordnet for reasoning. In: *Proceedings of COLING 2012: Demonstration Papers*. Mumbai, India: The COLING 2012 Organizing Committee, 2012. p. 353–360. Published also as Techreport http://hdl.handle.net/10438/10274. Disponível em: <http://www.aclweb.org/anthology/C12-3044>. Citado 5 vezes nas páginas 26, 27, 30, 45 e 64.

PATWARDHAN, S.; BANERJEE, S.; PEDERSEN, T. Using measures of semantic relatedness for word sense disambiguation. In: *Proceedings of the 4th International Conference on Computational Linguistics and Intelligent Text Processing*. Berlin, Heidelberg: Springer-Verlag, 2003. (CICLing'03), p. 241–257. ISBN 3540005323. Citado na página 33.

PDFMINER. *pdfminer*. 2022. <https://pypi.org/project/pdfminer/>. Acessado em: 2022-09-18. Citado na página 31.

PLAGIARISM check. *CheckPlagiarism*. 2022. <https://www.check-plagiarism.com/plagiarism-comparison-search>. Acessado em: 2022-08-07. Citado na página 28.

PREPOSTSEO. *Prepostseo*. 2022. <https://www.prepostseo.com/plagiarism-comparison-search>. Acessado em: 2022-08-06. Citado na página 28.

PYTHON. *What is Python*. 2022. <https://www.python.org/doc/essays/blurb/>. Acessado em: 2022-07-02. Citado na página 30.

REACT. *React*. 2022. <https://reactjs.org/>. Acessado em: 2022-09-18. Citado na página 29.

REACTJS. *React A JavaScript library for building user interfaces*. 2022. <https://reactjs.org/>. Acessado em: 2022-07-07. Citado na página 29.

REPORTLAB. *reportlab*. 2022. <https://www.reportlab.com/dev/docs/>. Acessado em: 2022-09-18. Citado na página 31.

RESNIK, P. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *CoRR*, abs/1105.5444, 2011. Disponível em: <http://arxiv.org/abs/1105.5444>. Citado na página 23.

SENDINBLUE. *sendinblue*. 2022. <https://sendinblue.com>. Acessado em: 2022-09-18. Citado na página 31.

SHIVAKUMAR, N.; GARCIA-MOLINA, H. Scam: A copy detection mechanism for digital documents. *proc DL*, 1995. Citado na página 19.

SKETCHENGINE. *Pos Tags*. 2022. <https://www.sketchengine.eu/blog/pos-tags/>. Acessado em: 2022-08-03. Citado na página 22.

SOMMERVILLE, I. *Engenharia de software*. 9. ed. ed. Sao Paulo: Pearson Prentice Hall, 2011. OCLC: 940079598. ISBN 9788579361081. Citado na página 37.

SPACY.IO. *spacy*. 2022. <https://spacy.io/>. Acessado em: 2022-09-18. Citado na página 31.

SQLITE. *SQLite Copyright*. 2010. <https://www.sqlite.org/copyright.html>. Acessado em: 2022-06-25. Citado na página 30.

SZUMILAS, M. Explaining odds ratios. *J. Can. Acad. Child Adolesc. Psychiatry*, v. 19, n. 3, p. 227–229, ago. 2010. Citado na página 24.

WARIN, M.; OXHAMMAR, H.; VOLK, M. Enriching an ontology with wordnet based on similarity measures. *MEANING-2005 Workshop*, 08 2008. Citado na página 33.

WEB.DEV. *cross-origin-resource-sharing*. 2022. <https://web.dev/cross-origin-resource-sharing/>. Acessado em: 2022-09-18. Citado na página 19.

WEBSTER, M. *Definition of lexicon*. 2022. Acessado em: 2022-08-21. Disponível em: <https://www.merriam-webster.com/dictionary/lexicon>. Citado na página 21.

WHITENOISE. *whitenoise*. 2022. <https://pypi.org/project/whitenoise/>. Acessado em: 2022-09-18. Citado na página 31.

WN. *WN documentation*. 2022. <https://wn.readthedocs.io/en/latest/>. Acessado em: 2022-07-21. Citado 2 vezes nas páginas 30 e 44.

WN. *wordnet words senses and synsets*. 2022. <https://wn.readthedocs.io/en/latest/guides/wordnet.html?highlight=forms#words-senses-and-synsets>. Acessado em: 2022-07-23. Citado 2 vezes nas páginas 21 e 22.

WSGI.READTHEDOCS.IO. *wsgi*. 2022. <https://wsgi.readthedocs.io/en/latest/what.html>. Acessado em: 2022-09-18. Citado na página 19.

WU, Z.; PALMER, M. Verb semantics and lexical selection. In: *32nd Annual Meeting of the Association for Computational Linguistics*. Las Cruces, New Mexico, USA: Association for Computational Linguistics, 1994. p. 133–138. Disponível em: <https://aclanthology.org/P94-1019>. Citado 3 vezes nas páginas 23, 32 e 33.

YERRA, R.; NG, Y.-K. Detecting similar html documents using a fuzzy set information retrieval approach. In: *2005 IEEE International Conference on Granular Computing*. USA: IEEE, 2005. v. 2, p. 693–699 Vol. 2. Citado 3 vezes nas páginas 20, 25 e 36.

ZADEH, L. Fuzzy sets. *Information and Control*, v. 8, n. 3, p. 338–353, 1965. ISSN 0019-9958. Disponível em: <https://www.sciencedirect.com/science/article/pii/S001999586590241X>. Citado na página 19.

Apêndices

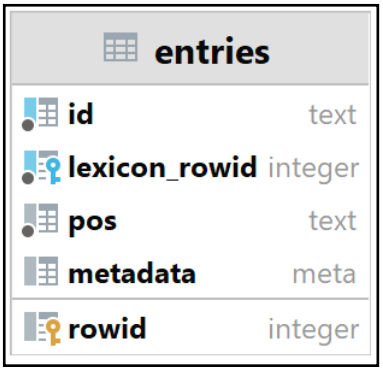
APÊNDICE A – Modelos do banco de dados

A.1 *Model Entries*

Esta tabela armazena as entidades no banco de dados, entidades são compostas de palavras.

- *id*: O seu identificador como uma palavra.
- *lexicon_rowid*: referencia a seu banco léxico.
- *pos*: *(Part-of-speech)* indica sua classificação como palavra.
- *metadata*: armazena informações sobre a estrutura.

Figura 36 – *Model Entries*



Fonte: Próprio autor

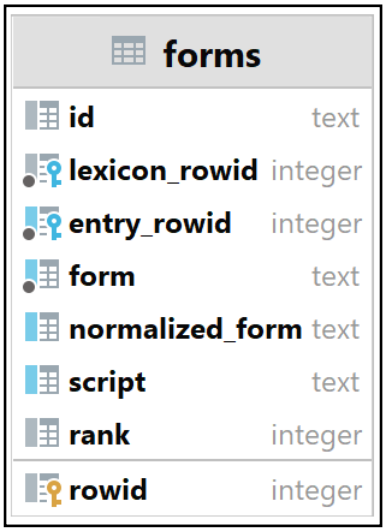
A.2 *Model Forms*

Esta tabela armazena as formas canônicas das palavras *(entries)*.

- *id*: O seu identificador como um form.
- *lexicon_rowid*: referencia a seu banco léxico.
- *entry_rowid*: referência a qual palavra*(entry)* o *sense* é ligado.
- *form*: a palavra na forma canônica.
- *normalized_form*: a forma normalizada do *form*.

- *script*: indica qual *script* o *form* deva seguir caso existir.
- *rank*: pode ser usado para indicar a preferência por um tipo de *form*, *rank* 0 é indica que é o lema o preferido.
- *rowid*: indica sua posição de id nas linhas da tabela.

Figura 37 – *Model Forms*



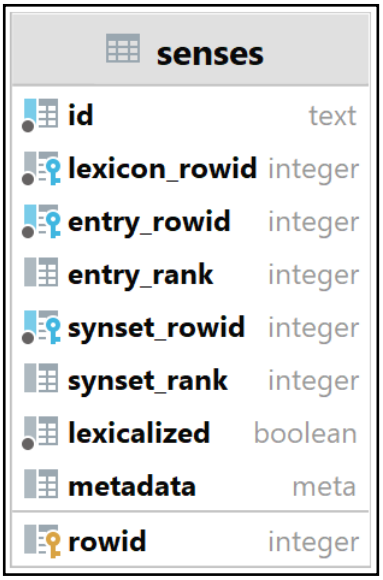
Fonte: Próprio autor

A.3 *Model Senses*

Esta tabela armazena os *senses*.

- *id*: O seu identificador como um sentido.
- *lexicon_rowid*: referencia a seu banco léxico.
- *entry_rowid*: referência a qual palavra*(entry)* o *sense* é ligado.
- *entry_rank*: pode ser usado para indicar a preferência por um tipo de *entry*.
- *synset_rowid*: referência a qual *synset* ele é ligado.
- *synset_rank*: pode ser usado para indicar a preferência por um tipo de *synset*.
- *lexicalized*: um booleano que indica se o sentido está lexicalizado.
- *metadata*: armazena informações sobre a estrutura.

Figura 38 – *Model Sense*



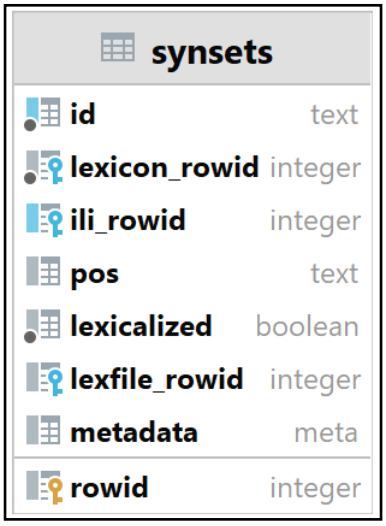
Fonte: Próprio autor

A.4 *Model Synsets*

Esta tabela armazena os *synsets* (conjunto de sinônimos).

- *id*: O seu identificador como um *synset*.
- *lexicon_rowid*: referencia a seu banco léxico.
- *ili_rowid*: referência se é interlingual ou não
- *pos*: *(Part-of-speech)* indica sua classificação como palavra.
- *lexicalized*: boolean que indica se esta lexicalizado ou nao.
- *lexfile_rowid*: referência a tabela externa de arquivos lexicos.
- *metadata*: armazena informações sobre a estrutura.
- *row id*: indica o seu número de linha na tabela.

Figura 39 – *Model Synsets*



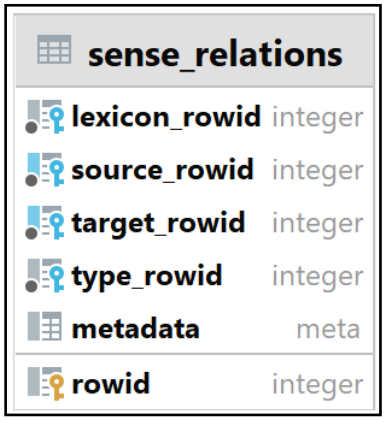
Fonte: Próprio autor

A.5 *Model Sense Relations*

Esta tabela armazena as relações entre os *senses*.

- *row id*: indica o seu número de linha na tabela.
- *lexicon_rowid*: O seu identificador do *lexicon*.
- *source_rowid*: é a referência ao *sense* origem da relação.
- *target_rowid*: é a referência ao *sense* destino da relação.
- *type_rowid*: é a referência a tabela *type*.
- *metadata*: armazena informações sobre a estrutura.

Figura 40 – *Model Sense Relations*



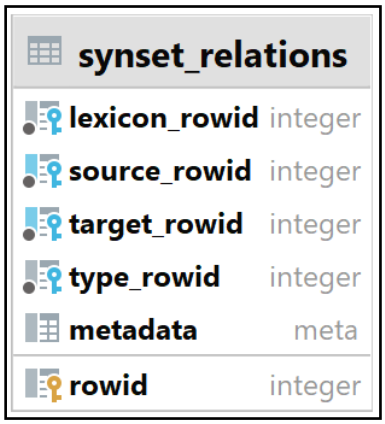
Fonte: Próprio autor

A.6 *Models Synset Relations*

Esta tabela armazena as relações entre os *synsets*.

- *lexicon_rowid*: O seu identificador do *lexicon*.
- *source_rowid*: é a referência ao *synset* origem da relação.
- *target_rowid*: é a referência ao *synset* destino da relação.
- *type_rowid*: id referência a tabela *relation types*.
- *metadata*: armazena informações sobre a estrutura.
- *row id*: indica o seu número de linha na tabela.

Figura 41 – *Model Synsets Relations*



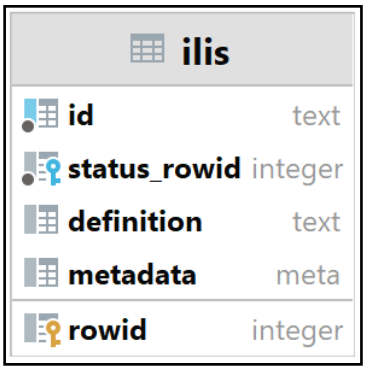
Fonte: Próprio autor

A.7 *Model Ilis*

Esta tabela armazena o identificador interlingual dos *synsets*.

- *id*: O seu identificador como um interlingual.
- *status_rowid*: referência a tabela externa *status row id* com o seu tipo de *status*.
- *definition*: definição do *status*.
- *metadata*: armazena informações sobre a estrutura.
- *row id*: indica o seu número de linha na tabela.

Figura 42 – *Model Ilis*



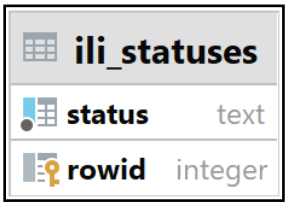
Fonte: Próprio autor

A.8 *Model Ilis Statuses*

Esta tabela armazena o *status* do identificador interlingual.

- *status*: texto que informa seu *status*.
- *row id*: indica o seu id.

Figura 43 – *Model Ilis Status*



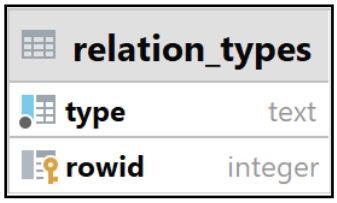
Fonte: Próprio autor

A.9 *Model Relation Types*

Esta tabela armazena tipos de relações

- *type*: texto que indica o tipo da relação.
- *row id*: indica o seu id.

Figura 44 – *Model Relation Types*



Fonte: Próprio autor

APÊNDICE B – Algoritmos

B.1 Manipular textos

B.1: usando as bibliotecas *pdfminer* e *docx2txt* este método é responsável por extrair o texto dos arquivos, sendo *pdf*, *txt* ou *docx* e retorna o seu conteúdo em formato texto.

Código B.1 – Extrair texto dos arquivos

```
def extract_text(file):
    name = file.name
    obj = file.file
    extracted_text = ""
    if name.endswith('.pdf'):
        extracted_text = high_level.extract_text(obj, "")
    elif name.endswith('.txt'):
        for line in obj:
            extracted_text += line.decode() + ' '
        extracted_text = extracted_text[:-1]
    elif name.endswith('.docx'):
        extracted_text = docx2txt.process(obj)
    extracted_text = " ".join(extracted_text.split())
    return extracted_text
```

B.2: utilizando a biblioteca *spaCy* e recebendo como entrada o texto, este método responsável por realizar a segmentação do texto em sentenças. Além disto, também são realizadas algumas etapas de pré-processamento textual, como a remoção de *stop words*.

Código B.2 – Segmentar texto em sentenças

```
def segmentation_based_sentences(self, text):
    originaltext_segmented = []
    doc = re.findall(r'[^.!?]+[.!?]?', text)
    for sentence in doc:
        sentence_source = self.remove_only_spaces(sentence)
        sentence_clean = self.remove_stop_words_puncts_spaces(sentence)
        if len(sentence_clean) > 0 and not
           self.verify_sentence_already_segmented(originaltext_segmented,sentence_cl
```

```
        originaltext_segmented.append((sentence_source, sentence_clean))
    return originaltext_segmented
```

B.2 Calcular similaridade

B.3: por questões de segurança do servidor *heroku* não é permitido requisições longas e como o processamento dos arquivos pode ser demorado, foi optado por utilizar *threads* para processar os arquivos e então requisições posteriores para verificar se o resultado está completo. Sendo assim, este método é responsável pro iniciar as *threads*, bem como criar as flags que indicam o *status* da análise dos documentos, realizar a manutenção das mesmas e invocar os métodos que aplicam a análise sobre os documentos.

Código B.3 – Calcular similaridade

```
class CalculateSimilarity(APIView):
def post(self, request, format=None):
    data = request.data
    if not os.path.exists("api/analyse_flags"):
        os.makedirs("api/analyse_flags")
    if not os.path.exists("api/analyse_flags/processing-lock.txt"):
        with open("api/analyse_flags/processing-lock.txt", "w") as f:
            f.write(str(0))
            f.close()
        info_files = text_data.get_info_from_files(request)
        thread_long = ThreadLong(data=info_files)
        thread_long.start()
        if not
           os.path.exists("api/analyse_flags/result_analyse.dictionary"):
            with open('api/analyse_flags/processing-lock.txt', 'rb') as f:
                data_file = int(f.read())
                calc = data_file / len(info_files)
                f.close()
            return Response(calc)
        else:
            with open('api/analyse_flags/result_analyse.dictionary',
               'rb') as config_dictionary_file:
                config_dictionary = pickle.load(config_dictionary_file)

            delete_flags()
            return Response(config_dictionary)
```

```
    else:
        if not
            os.path.exists("api/analyse_flags/result_analyse.dictionary"):
            with open('api/analyse_flags/processing-lock.txt', 'rb') as f:
                data_file = int(f.read())
                dits = dict(data.lists())
                files = dits.get('file')
                calc = data_file / len(files)
                f.close()
            return Response(calc)
        else:
            with open('api/analyse_flags/result_analyse.dictionary',
                'rb') as config_dictionary_file:
                config_dictionary = pickle.load(config_dictionary_file)
            delete_flags()
            return Response(config_dictionary)
```

B.4: este método realiza a análise entre os dois documentos, segmentando o conteúdo dos arquivos, analisando suas sentenças.

Código B.4 – Analisar os documentos

```
def analyse_docs(file_name1, file_name2, doc1, doc2):
    text_manipulation = text_data.TextManipulation()
    similarity = similarity_data.Similarity()
    doc1_segmented = text_manipulation.segmentation_based_sentences(doc1)
    doc2_segmented = text_manipulation.segmentation_based_sentences(doc2)
    qntd_similar_sets1, similar_sets_log1, sentences_analysed1 =
        similarity.calculate_similar_sentences_in_docs(
        doc1_segmented, doc2_segmented, [])
    degree_resemblance1 =
        similarity.degree_resemblance(qntd_similar_sets1,
        len(doc1_segmented))
    qntd_similar_sets2, similar_sets_log2, sentences_analysed2 =
        similarity.calculate_similar_sentences_in_docs(
        doc2_segmented, doc1_segmented, sentences_analysed1)
    degree_resemblance2 =
        similarity.degree_resemblance(qntd_similar_sets2,
        len(doc2_segmented))
    percent_plagiarism =
```

```
        similarity.odds_ratio_in_percent(degree_resemblance1,
        degree_resemblance2)
    analyse = AnalyseResult()
    analyse.name_file1 = file_name1
    analyse.name_file2 = file_name2
    analyse.similar_sets_log1 = similar_sets_log1
    analyse.similar_sets_log2 = similar_sets_log2
    analyse.percent_plagiarism = percent_plagiarism
    return analyse
```

B.7: este método realiza a análise entre as sentenças dos documentos percorrendo suas sentenças.

Código B.5 – Calcula as sentenças similares nos documentos

```
def calculate_similar_sentences_in_docs(self, doc_segmented1,
    doc_segmented2, sentences_already_analysed, type_analyse):

    qntd_similar_sets = []
    similar_sentences_log = []
    sentences_analysed = sentences_already_analysed

    for sentence1 in doc_segmented1:
        lock_similar_found = False
        for sentence2 in doc_segmented2:
            verify_set = self.verify_sentence_analysed(sentences_analysed,
                self.convert_token_to_str(sentence1[1]),
                self.convert_token_to_str(sentence2[1]))

            if verify_set == None:
                similar_sets_temp =
                    self.similarity_between_sentences(sentence1[1],
                sentence2[1], type_analyse)

                uAB = similar_sets_temp[0]
                uBA = similar_sets_temp[1]

                sentence_result = {
                    "sentence_doc1":
                        self.convert_token_to_str(sentence1[0]),
```

```
                    "sentence_doc2":
                        self.convert_token_to_str(sentence2[0]),
                    "sentence_trated_doc1":
                        self.convert_token_to_str(sentence1[1]),
                    "sentence_trated_doc2":
                        self.convert_token_to_str(sentence2[1]),
                    "percentage_doc1_doc2": similar_sets_temp[0],
                    "percentage_doc2_doc1": similar_sets_temp[1],
                }

                if self.sentences_similar_threshold(uAB, uBA):
                    threshold_high = True
                    if not lock_similar_found:
                        qntd_similar_sets.append(1)
                        lock_similar_found = True
                    similar_sentences_log.append(sentence_result)
                else:
                    threshold_high = False
                sentences_analysed.append((sentence_result,
                    threshold_high))

            elif verify_set != None and verify_set != False:
                if not lock_similar_found:
                    qntd_similar_sets.append(1)
                    lock_similar_found = True
                similar_sentences_log.append(verify_set)

    return qntd_similar_sets, similar_sentences_log, sentences_analysed
```

B.6: este método realiza a análise entre duas sentenças, analisando suas palavras.

Código B.6 – Analisar as sentenças

```
def similarity_between_sentences(self, set1, set2, type_analyse):
    anB = []
    bmA = []
    words_analyesd = []
    for word1 in set1:
        similarity = 0
```

```
        temp_similarity = []
        for word2 in set2:
            verify = self.verify_word_analysed(words_analyesd, word1,
                word2)
            if verify == None:
                if similarity < 1 or similarity == SYNONYMGROUPNOTFOUND:
                    similarity = self.wu_palmer_similarity(word1, word2,
                        type_analyse)
                    temp_similarity.append(similarity)
                    words_analyesd.append((word1, word2, similarity))
                else:
                    temp_similarity.append(similarity)
            else:
                similarity = verify
                temp_similarity.append(similarity)

        if SYNONYMGROUPNOTFOUND in temp_similarity:
            while SYNONYMGROUPNOTFOUND in temp_similarity:
                temp_similarity.remove(
                SYNONYMGROUPNOTFOUND)

        if len(temp_similarity) > 0:
            anB.append(max(temp_similarity))

    for word2 in set2:
        similarity = 0
        temp_similarity = []
        for word1 in set1:
            verify = self.verify_word_analysed(words_analyesd, word2,
                word1)
            if verify == None:
                if similarity < 1 or similarity == SYNONYMGROUPNOTFOUND:
                    similarity = self.wu_palmer_similarity(word1, word2,
                        type_analyse)
                    temp_similarity.append(similarity)
                    words_analyesd.append((word1, word2, verify,
                        similarity))
            else:
                similarity = verify
```

```
                temp_similarity.append(similarity)

        if SYNONYMGROUPNOTFOUND in temp_similarity:
            while SYNONYMGROUPNOTFOUND in temp_similarity:
                temp_similarity.remove(
                SYNONYMGROUPNOTFOUND)
        if len(temp_similarity) > 0:
            bmA.append(max(temp_similarity))
    if len(anB) > 0:
        uAB = sum(anB) / len(anB)
    else:
        uAB = 0
    if len(bmA) > 0:
        uBA = sum(bmA) / len(bmA)
    else:
        uBA = 0
    return uAB, uBA
```

B.7: este método realiza a análise entre duas palavras, encontrando seus synsets e aplicando o *wup* do pacote *wn* neles.

Código B.7 – Analisar as palavras

```
def wu_palmer_similarity(self, word1, word2, type_analyse):
    if word1 == word2:
        return 1

    synsets1 = wn.synsets(word1)
    synsets2 = wn.synsets(word2)
    analyse_synsets_wup = [('', '', 0)]
    value_similarity = 0

    if len(synsets1) > 0 and len(synsets2) > 0:
        if type_analyse == "profunda":
            for synset1 in synsets1:
                for synset2 in synsets2:
                    if synset1.pos == synset2.pos:
                        result = wn.similarity.wup(synset1, synset2,
                            simulate_root=True)
                        analyse_synsets_wup.append((synset1, synset2,
```

```
                        result))
            value_similarity = max(analyse_synsets_wup, key=lambda x:
                x[2])[2]
        elif type_analyse == "rapida":
            if synsets1[0].pos == synsets2[0].pos:
                value_similarity = wn.similarity.wup(synsets1[0],
                    synsets2[0], simulate_root=True)
    else:
        return SYNONYMGROUPNOTFOUND
    return value_similarity
```

B.3 Pacote *wn*

No projeto foram utilizados alguns do pacote *wn* o principal é descrito abaixo *wn.wup()*.

B.3.1 *WuPalmer*

B.8: retorna a similaridade *WuPalmer* entre dois *synsets*. Para isto recebe os *synsets*, calcula seu *Least Common Subsumers* e por fim, aplica a equação da métrica de similaridade de *WuPalmer*.

Código B.8 – Cálculo *WuPalmer*

```
def wup(synset1: Synset, synset2: Synset, simulate_root=False) -> float:
    _check_if_pos_compatible(synset1.pos, synset2.pos)
    lcs_list = _least_common_subsumers(synset1, synset2, simulate_root)
    lcs = lcs_list[0]
    i = len(synset1.shortest_path(lcs, simulate_root=simulate_root))
    j = len(synset2.shortest_path(lcs, simulate_root=simulate_root))
    k = lcs.max_depth() + 1
    return (2*k) / (i + j + 2*k)
```

Todos os algoritmos, bem como todo o código do projeto, pode ser encontrado no repositório GitHub ArildoMagno/prototipo-aplicacao-web-deteccao-plagio.