UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LEONARDO NEVES DA SILVA
SIDNEY RIBEIRO RAMOS JÚNIOR

PROCESSAMENTO DE LINGUAGEM NATURAL

Uma abordagem através do Word2Vec

RIO DE JANEIRO
2024

LEONARDO NEVES DA SILVA
SIDNEY RIBEIRO RAMOS JÚNIOR

PROCESSAMENTO DE LINGUAGEM NATURAL

Uma abordagem através do Word2Vec

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Profa. Valeria Menezes Bastos

RIO DE JANEIRO
2024

Silva, Leonardo Neves da
S586p Processamento de linguagem natural: uma abordagem através do Word2Vec / Leonardo Neves da Silva e Sidney Ribeiro Ramos Júnior. – 2024.

78 f.

Orientadora: Valeria Menezes Bastos.

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação)- Universidade Federal do Rio de Janeiro, Instituto de Computação, Bacharel em Ciência da Computação, 2024.

1. Word2Vec. 2. Processamento de linguagem natural. 3. Língua Portuguesa. 4. Wikipedia. 5. Clueweb. I. Ramos Júnior, Sidney Ribeiro. II. Bastos, Valeria Menezes (Orient.). III. Universidade Federal do Rio de Janeiro, Instituto de Computação. IV. Título.

LEONARDO NEVES DA SILVA
SIDNEY RIBEIRO RAMOS JÚNIOR

PROCESSAMENTO DE LINGUAGEM NATURAL
Uma abordagem através do Word2Vec

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 28 de maio de 2024

BANCA EXAMINADORA:

gov.br Documento assinado digitalmente
VALERIA MENEZES BASTOS
Data: 30/08/2024 10:00:42-0300
Verifique em https://validar.iti.gov.br

Valeria Menezes Bastos
D.Sc. (Instituto de Computação - UFRJ)

gov.br Documento assinado digitalmente
SILVANA ROSSETTO
Data: 29/08/2024 14:57:00-0300
Verifique em https://validar.iti.gov.br

Silvana Rossetto
D.Sc. (Instituto de Computação - UFRJ)

gov.br Documento assinado digitalmente
JOAO ANTONIO RECIO DA PAIXAO
Data: 29/08/2024 14:48:30-0300
Verifique em https://validar.iti.gov.br

João Antonio Recio Da Paixão
D.Sc. (Instituto de Computação - UFRJ)

Dedicamos este trabalho aos membros amados de nossas famílias, cujo amor, apoio e incentivo foram fundamentais em cada etapa de nossas jornadas acadêmicas.

Agradecemos também a nossa coordenadora, Valéria Menezes Bastos, pela orientação dedicada, cobranças, sabedoria e inspiração que tornaram possível a realização deste trabalho. Sem o apoio de cada um de vocês, este projeto não teria sido alcançado.

Muito obrigado por tudo.

AGRADECIMENTOS

Em primeiro lugar, a Deus, pela nossa vida e por nos permitir ultrapassar todos os obstáculos encontrados ao longo da realização deste trabalho.

A minha amada família. Sem o seu apoio incondicional, esse momento não seria possível. Obrigado pelo suporte emocional, por estarem presente e por acreditar em mim.

A professora Valeria Menezes Bastos, por ter sido nossa orientadora e ter desempenhado tal função com dedicação e amizade.

Ao professor Estevam Rafael Hruschka Junior, pelos contatos e materiais que foram fundamentais para o desenvolvimento da pesquisa que possibilitou a realização deste trabalho.

A Doutora Maisa Duarte, pelo fornecimento de dados e materiais que foram fundamentais para o desenvolvimento dos testes que construíram e apoiaram a realização deste trabalho.

À instituição de ensino UFRJ, essencial no meu processo de formação profissional, pela dedicação, e por tudo o que aprendi ao longo dos anos do curso.

Aos professores, por todos os conselhos, pela ajuda e pela paciência com a qual guiaram o meu aprendizado.

Aos meus colegas de curso, com quem convivi intensamente durante anos, pelo companheirismo e pela troca de experiências que me permitiram crescer não só como pessoa, mas também como formando.

A todos aqueles que contribuíram, direta ou indiretamente, para a realização deste trabalho.

Com gratidão, Sidney Ribeiro Ramos Junior

AGRADECIMENTOS

Gostaria de expressar minha sincera gratidão a todas as pessoas e instituições que desempenharam um papel fundamental na realização deste trabalho de conclusão de curso. Este momento não seria possível sem o apoio incondicional de minha amada família. Agradeço por estar sempre presente, fornecendo suporte emocional e acreditando em minha jornada acadêmica, mesmo nos momentos mais desafiadores.

À dedicada coordenadora, Valéria Menezes Bastos, estendo meus agradecimentos especiais. Sua insistência, paciência e orientações valiosas foram cruciais para o desenvolvimento deste projeto. O comprometimento e a paixão demonstrados por ela desempenharam um papel vital em minha trajetória acadêmica, moldando não apenas este trabalho, mas também minha visão como estudante.

Além disso, quero expressar minha gratidão a todos os professores que compartilharam seus conhecimentos e experiências, enriquecendo minha formação acadêmica. Cada conselho, crítica construtiva e incentivo foram elementos essenciais para o meu crescimento como estudante e profissional em formação.

Em relação à parte técnica, gostaria de mencionar pessoas como Rafael Hruschka, Maisa Duarte e Myrian Costa, que nos forneceram não apenas materiais de estudo, mas também experiências e pontos de vista fundamentais para que chegássemos a este resultado satisfatório que apresentamos hoje.

Não posso deixar de mencionar meus colegas de turma, cujo apoio mútuo contribuiu para um ambiente de aprendizado colaborativo e enriquecedor. Juntos, enfrentamos desafios, celebramos conquistas e construímos memórias inestimáveis que levarei comigo ao longo de minha jornada.

Por fim, agradeço a todos que, direta ou indiretamente, contribuíram para a realização deste trabalho. Este é um marco significativo em minha vida acadêmica, e todos vocês desempenharam um papel vital em meu sucesso.

Com gratidão, Leonardo Neves da Silva

"Artificial intelligence would be the ultimate version of Google. The ultimate search engine that would understand everything on the web. It would understand exactly what you wanted, and it would give you the right thing. "

Larry Page

RESUMO

Este trabalho investiga o Processamento de Linguagem Natural (PLN) utilizando diversas técnicas e ferramentas, com ênfase na aplicação Word2Vec. O PLN é uma área de estudo essencial para a interação entre computadores e linguagens humanas, sendo crucial para aplicações como tradução automática, análise de sentimentos e assistentes virtuais.
Além do Word2Vec, o estudo explora diversas ferramentas e técnicas utilizadas para processar e analisar grandes bases de dados textuais. Entre os principais *corpus*, estão a Wikipedia e o ClueWeb. O estudo também implementa e avalia métodos de limpeza de dados e normalização de texto, abordando questões como a remoção de acentuação e a correção de erros ortográficos. Esses processos são essenciais para preparar o texto bruto para a análise subsequente, garantindo que os modelos possam operar com maior eficiência.
O trabalho testa algumas soluções para a análise de similaridade semântica entre palavras e avaliação de contextos linguísticos. Os resultados demonstram se técnicas aplicadas são eficazes para melhorar a compreensão e o processamento automático da linguagem natural. Apesar do NELL ter sido a principal inspiração para o início do desenvolvimento, a proposta central é validar o uso dessas ferramentas no contexto da língua portuguesa, mostrando sua aplicabilidade e eficácia em diversas tarefas de PLN.

Palavras-chave: Word2Vec; Processamento de linguagem natural; Língua Portuguesa. Wikipedia; Clueweb; NLTK; NELL.

ABSTRACT

This work investigates Natural Language Processing (NLP) using various techniques and tools, with a focus on the application of Word2Vec. NLP is a crucial field for the interaction between computers and human languages, being essential for applications such as machine translation, sentiment analysis, and virtual assistants.
In addition to Word2Vec, the study explores various tools and techniques used to process and analyze large textual databases. Among the primary corpora are Wikipedia and ClueWeb. The study also implements and evaluates methods for data cleaning and text normalization, addressing issues such as accent removal and spelling correction. These processes are essential for preparing raw text for subsequent analysis, ensuring that models can operate with greater efficiency.
The work tests several solutions for semantic similarity analysis between words and contextual linguistic evaluation. The results demonstrate whether the applied techniques are effective in improving the understanding and automatic processing of natural language.
Although NELL was the main inspiration for the start of the development, the central proposal is to validate the use of these tools in the context of the Portuguese language, showing their applicability and effectiveness in various NLP tasks.

Keywords: Word2Vec; NLP; Natural Language Processing; Brazilian Portuguese language; Wikipedia; Clueweb; NLTK; NELL.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

CMU	Carnegie Mellon University (Universidade Carnegie Mellon)
PLN	Processamento de Linguagem Natural
NLP	Natural Language Processing (Processamento de Linguagem Natural)
WEKA	Waikato Environment for Knowledge Analysis (Ambiente Waikato para Análise do Conhecimento)
NELL	Never-Ending Language Learning (Aprendizado de Linguagem Sem Fim)
NLTK	Natural Language Toolkit (Kit de Ferramentas de Linguagem Natural)
IA	Inteligência Artificial
DB	Database (Base de Dados)
API	Application Programming Interface (Interface de Programação de Aplicações)
GPU	Graphics Processing Unit (Unidade de Processamento Gráfico)
POS	Part-of-Speech (Classificação Gramatical)
CNN	Convolutional Neural Network (Rede Neural Convolucional)
RNN	Recurrent Neural Network (Rede Neural Recorrente)
CSV	Comma-Separated Values (Valores Separados por Vírgula)
SQL	Structured Query Language (Linguagem de Consulta Estruturada)
KNN	K-nearest neighbors (K-vizinhos mais próximos)
XML	eXtensible Markup Language (Linguagem de Marcação Extensível)
WARC	Web ARChive (Arquivo da Web)
CBOW	Continuous Bag of Words (Bolsa de Palavras Contínua)
BoW	Bag of Words (Bolsa de Palavras)
HTML	Hypertext Markup Language (Linguagem de Marcação de Hipertexto)

SUMÁRIO

1 INTRODUÇÃO

Desde os primórdios da civilização, a escrita tem sido uma tecnologia essencial que não para de evoluir. No começo, a ideia era usar símbolos simples para que as pessoas pudessem se expressar e facilitar a comunicação. Hoje, a escrita é a nossa principal forma de transmitir conhecimento e tem sido cada vez mais aprimorada ao longo dos anos. Vimos a evolução da escrita em vários formatos, desde a criação dos símbolos usados na pré-história, passando pelos hieróglifos, a transformação dos alfabetos e hoje possuímos vários formatos que evoluem de formas independentes e são usados por grupos diferentes de leitores. Os *QR-Codes*, por exemplo, são a evolução dos códigos de barras e são uma forma que transmitir conhecimento de forma rápida e compacta.

A linguagem é uma das principais ferramentas utilizada pelos humanos para se comunicar, ou seja, a forma que um ser humano consegue transmitir uma mensagem para outro. Porém, a maneira como escrevemos pode facilitar ou dificultar a comunicação, mas tudo depende da capacidade do receptor de interpretar a mensagem.

A interpretação de textos e seu contexto é onde muitas vezes uma ideia pode ser distorcida. Palavras e sons podem ter significados diferentes em línguas diferentes. E nem precisamos comparar línguas distintas para ver isso acontecer. No nosso próprio idioma, o português brasileiro, existem variações regionais que fazem com que uma mesma palavra possa ter significados diferentes dependendo de onde você está. Dentro destas variações temos também palavras homônimas, as quais são escritas de forma idêntica, mas possuem significados diferentes.

Estudar a similaridade entre palavras sempre intrigou estudiosos. Grandes pesquisas etimológicas tentam desvendar as origens e evoluções das palavras e entender porque palavras não similares podem ter significados parecidos, ou vice-versa. Quando os computadores surgiram, pudemos dar um salto significativo nessas pesquisas, armazenando vastos dados linguísticos usados em diversos tipos de aplicações. No entanto, uma parte de todo esse conhecimento continua no formato de dado bruto, ou seja, esperando apenas a aplicação certa surgir para que ela possa ser compreendida e tratada. E é neste ponto que entra o aprendizado de máquina.

Com avanços das tecnologias de aprendizado de máquina e a utilização das suas variadas técnicas como análise de sentimentos, análise de discurso e análise morfológica, chegamos a um novo patamar. As máquinas agora podem desenvolver algo similar ao que chamamos de conhecimento empírico. Conhecimento empírico é aquele que se baseia na experiência direta, observação e experimentação do mundo real (PORFIRIO, 2024).

Ainda estamos longe de fazer com que as máquinas alcancem nosso nível de conhecimento, mas elas estão começando a "aprender". E isso é bem interessante. As máquinas aprendem de forma diferente, mas a base é a mesma: repetição, análise e ajuste. E assim

como os humanos, as máquinas também precisam de dados para treinar e aprender de maneira correta.

Neste trabalho, exploraremos como as tecnologias de aprendizado de máquina podem ser aplicadas para avaliar contextos em documentos, considerando as nuances da língua portuguesa usada no Brasil. Vamos tentar entender os desafios específicos que enfrentaremos com o português brasileiro e quais resultados poderemos alcançar. Afinal, linguagem não é só o que conseguimos falar, ouvir, ler e escrever, ela envolve estruturas muito mais complexas como: morfologia, sintaxe, semântica, pragmática, entre outros aspectos que dependem do contexto em que o texto está inserido (Wikipedia, 2024).

1.1 MOTIVAÇÃO

A motivação por trás deste trabalho começou quando conhecemos o NELL (Never-Ending Language Learning)(CARLSON et al., 2010), um projeto inovador da Universidade Carnegie Mellon (CMU), que foi criado para ser uma inteligência artificial que está sempre aprendendo sobre o mundo ao seu redor. Esta IA percorre a internet, lendo e aprendendo novos conteúdos constantemente.

O NELL foi desenvolvido pela equipe da CMU liderada por Tom Mitchell. O sistema usa *crawlers* para vasculhar a Internet em busca de novos textos, aumentando seu conhecimento de forma contínua. Com milhões de exemplos de frases, recursos morfológicos e estruturas de páginas da web, o NELL consegue melhorar sua capacidade de leitura com o tempo. Ele aprende a distinguir entre o que já conhece e o que ainda não viu, otimizando sua busca por novos conteúdos. E mais do que isso, ele consegue adicionar novos itens e propor novas classificações.

O NELL funciona forma autônoma. Ele consegue criar seu próprio vocabulário e entender o que ele significa. Isso tem muitas vantagens, como a rapidez e a eficiência na aprendizagem. Mas também tem desvantagens, como a possibilidade de interpretar conteúdos de forma errada ou aprender informações que não são verdadeiras.

Assim que descobrimos o NELL algumas perguntas se fixaram em nossas cabeças.

"Será que o NELL funcionaria bem na língua portuguesa?"

"O que precisaríamos estudar para entender um projeto desta magnitude?"

"Já existem projetos similares no Brasil?"

Essas questões nos levaram a explorar mais a fundo o campo do processamento de linguagem natural (PLN) e as tecnologias disponíveis. A ideia de ter uma máquina que pudesse aprender e entender português como o NELL faz com o inglês é fascinante. O português, com suas nuances e variações regionais, apresenta um desafio único que nos motivou a investigar mais. Queríamos descobrir se seria possível aplicar tecnologias tão avançadas para funcionar eficientemente em nosso idioma.

Para isso, começamos a estudar as técnicas e métodos de *PLN* que já existem. Buscamos modelos e ferramentas disponíveis no mercado a fim de analisar se eles seriam adequados para nossos testes. Avaliamos também várias bases de dados para verificar se seriam suficientes para treinar uma IA no contexto do português brasileiro.

1.2 OBJETIVOS

Nosso principal interesse é investigar como as técnicas de processamento de linguagem natural (*PLN*) podem ser aplicadas ao português brasileiro. Queremos entender quais são os desafios específicos do nosso idioma, testar as tecnologias e explorar as soluções.

Para isso, temos alguns objetivos claros em mente. Primeiro, estudamos as técnicas e métodos de *PLN* que já existem, tanto em inglês quanto em português, para identificar quais são os mais eficazes. A ideia é compreender bem as ferramentas disponíveis e avaliar como elas podem ser adaptadas para lidar com as particularidades da língua portuguesa brasileira.

Para isso, utilizamos as principais ferramentas de *PLN* disponíveis no mercado para verificar se são adequados para nossos testes. Isso envolve uma análise detalhada das bases de dados que podem ser usadas para treinar uma *IA* no contexto do português brasileiro. Avaliar essas bases de dados é crucial para garantir que temos o material necessário para um aprendizado eficiente.

A cada etapa de testes, percebemos novos desafios e aprendemos mais sobre as particularidades do nosso idioma.

1.3 ORGANIZAÇÃO DO DOCUMENTO

Esse documento está organizado em 5 capítulos e seus respectivos subcapítulos.

No primeiro capítulo se encontram a motivação que nos levou ao presente trabalho e os objetivos que foram traçados para a pesquisa.

No segundo capítulo abordamos o Estado da Arte, mostrando de forma mais objetiva, o que é Processamento de Linguagem Natural e quais ferramentas utilizamos para o processo de pesquisa, desde bibliotecas e *frameworks*, até as bases de dados textuais (*corpus*) que foram utilizadas para treinamento dos modelos utilizados.

No terceiro capítulo abordamos sobre as metodologias utilizadas. Quais foram os critérios para a seleção das bases de dados textuais, como foi feito o preparo e o pré-processamento dessas bases e como foi o processo de integração desses dados.

No quarto capítulo descrevemos como o aprendizado e as técnicas foram utilizadas. Qual foi, dentre tantos existentes, o método e as bibliotecas que utilizamos para treinamento dos modelos. Ainda no capítulo quatro, é descrito todo o processo de treinamento dessas redes, desde a abordagem até a sanitização das bases de texto. Por fim, são apresentados os resultados obtidos.

No quinto e último capítulo são ditas as nossas considerações finais, contendo o que foi descoberto com a pesquisa e a conclusão que obtivemos após toda a jornada que percorremos.

2 ESTADO DA ARTE

Uma das etapas mais importantes ao se elaborar um projeto é a escolha da abordagem e das ferramentas que serão utilizadas. Por isso, grande parte do nosso trabalho foi testar e comprovar a eficácia das diferentes ferramentas de tratamento de texto disponíveis no mercado, para então escolher as mais adequadas a cada tipo de abordagem da pesquisa. Nesse capítulo, são apresentados os conceitos básicos que orientaram o nosso trabalho, as ferramentas avaliadas e utilizadas nas atividades realizadas.

2.1 PROCESSAMENTO DE LINGUAGEM NATURAL

O Processamento de Linguagem Natural (*PLN*) é uma área da Ciência da Computação e Inteligência Artificial dedicada a automatizar a geração e compreensão de línguas humanas naturais. Em um mundo cada vez mais digital, o *PLN* desafia os computadores a extrair significado da linguagem humana, transformando-a em formatos compreensíveis e úteis.

Um dos principais desafios enfrentados pelo *PLN* é a compreensão da Linguagem Natural. Isso envolve capacitar computadores a interpretar contextos e intenções presentes na comunicação humana cotidiana. Além disso, o *PLN* visa não apenas entender a linguagem, mas também gerá-la de forma que seja coerente e relevante para diferentes propósitos computacionais, como a análise de temas de textos ou classificação de palavras.

Ao longo da história, a comunicação humana evoluiu significativamente, refletindo até mesmo na diversidade de alfabetos desenvolvidos ao longo dos séculos. Desde os antigos hieróglifos até os alfabetos modernos como o romano, os seres humanos sempre buscaram aprimorar a forma como se expressam e se comunicam. A diversidade cultural também influencia na maneira como diferentes sociedades desenvolvem e utilizam seus sistemas de escrita.

Embora tenhamos evoluído na capacidade de ensinar a comunicação escrita a qualquer outra pessoa dedicada em aprender, ensinar máquinas a compreender esse tipo de linguagem ainda é um desafio. A meta do *PLN* é equipar computadores com a habilidade de processar e interpretar a linguagem humana assim como os seres humanos fazem.

O desenvolvimento do *PLN* envolve a criação de sistemas capazes de interpretar não apenas palavras isoladas, mas também frases e discursos complexos. Isso requer o uso de algoritmos avançados que podem analisar a estrutura gramatical, o contexto semântico e as intenções por trás das palavras utilizadas em um texto. Além disso, podemos fazer a categorização automática de documentos, a extração de informações relevantes e até mesmo a classificação de conteúdos.

2.1.1 WEKA

O software WEKA (Waikato Environment for Knowledge Analysis) (WITTEN et al., 2016) é uma ferramenta de código aberto amplamente utilizada para mineração de dados e aprendizado de máquina. Foi desenvolvido pela Universidade de Waikato na Nova Zelândia e leva esse nome em homenagem a uma espécie de ave nativa da região, como mostrado na Figura 1.

O WEKA foi uma boa escolha para começar porque ele trata de forma direta a aplicação dos algoritmos e os métodos utilizados para o aprendizado.

Figura 1 – O logotipo do WEKA homenageia uma espécie de ave, nativa da Nova Zelândia.

Uma das características mais notáveis do WEKA é sua interface gráfica amigável, que facilita a utilização por iniciantes. Através dessa interface, os usuários podem carregar conjuntos de dados de variados tipos, como arquivos *CSV*, *TXT* e até bases de dados *SQL*, escolher algoritmos de aprendizado de máquina, configurar parâmetros e avaliar os resultados com facilidade. Isso o torna uma escolha popular para fins educacionais.

Apesar da aparência simples, ele oferece uma ampla gama de algoritmos de aprendizado de máquina. Isso inclui algoritmos de classificação, regressão, *clustering*, associação, entre outros. Em sua interface o usuário pode escolher entre técnicas tradicionais, como árvores de decisão e os k-vizinhos mais próximos ou até abordagens mais avançadas, como redes neurais. A diversidade de algoritmos permite que os usuários escolham os mais adequados aos seus dados e problemas específicos.

Dos algoritmos presentes podemos citar os que achamos mais interessantes e pudemos testar com mais exemplos: O que mais usamos e achamos bem útil e intuitivo é o algoritmo de *Naive Bayes* que é muito eficiente para grandes conjuntos de dados e calcula a probabilidade de classificação conforme o teorema de Bayes. Outro algoritmo simples e interessante é o *KNN* (*K-nearest neighbors*) que classifica uma amostra com base nos exemplos mais próximos dentre as características dadas.

O WEKA também é notável pela capacidade de manipulação e transformação de dados. Ele permite a execução de tarefas de pré-processamento, como normalização, discretização e seleção de atributos, fundamentais para preparar os dados para uma análise,

tornando-os adequados para os algoritmos de aprendizado de máquina. Esse conjunto de ferramentas facilita a limpeza e preparação dos dados, etapas cruciais para a obtenção de resultados precisos e significativos.

Outro ponto forte do WEKA é a possibilidade de visualizar dados e resultados de forma gráfica. A ferramenta inclui vários métodos de visualização, como gráficos de dispersão, matrizes de confusão e gráficos de desempenho, que ajudam os usuários a entender melhor os padrões.

Além disso, o WEKA possui uma comunidade ativa de usuários e desenvolvedores. Isso significa que o usuário pode encontrar recursos adicionais e se necessário, tutoriais e suporte online para auxiliar no uso eficaz da ferramenta. A comunidade contribui para a manutenção e atualização contínua do WEKA, garantindo que ele permaneça relevante e eficaz. Essa rede de suporte é especialmente valiosa para novos usuários que podem enfrentar desafios iniciais ao explorar as capacidades da ferramenta.

O WEKA também é extensível, permitindo que os usuários criem e integrem seus próprios algoritmos e extensões. A capacidade de personalização e extensão faz do WEKA uma ferramenta versátil que pode evoluir junto com as necessidades dos seus usuários.

Nossa avaliação sobre o WEKA é que ele é um software fácil e simples de ser utilizado e por conta disso ele foi escolhido como nossa porta de entrada para realizar os primeiros testes práticos mais complexos com Inteligência Artificial e algoritmos de aprendizado de máquina. Sua acessibilidade, combinada com suas poderosas capacidades, faz dele uma escolha ideal para quem está começando no campo do aprendizado de máquina.

2.1.2 NLTK

O Natural Language Toolkit *NLTK* (BIRD; LOPER, 2009) é uma biblioteca de código aberto em Python, amplamente reconhecida e utilizada para processamento de linguagem natural (*PLN*). Desenvolvida pela Universidade da Pensilvânia, ela é uma ferramenta poderosa para análise de texto e tarefas relacionadas ao *PLN*. Através da Figura 2 podemos observar que o logotipo do NLTK faz clara referência ao logotipo da linguagem *Python*

Figura 2 – Logotipo do NLTK, uma biblioteca de código aberto em Python

Com uma ampla gama de recursos, o *NLTK* fornece um vasto conjunto de bibliotecas e módulos que abrangem desde a transformação em *tokens* até a análise sintática e a classificação de documentos. Essa variedade de recursos permite que os usuários realizem tarefas complexas de processamento de linguagem natural com facilidade.

Além disso, o *NLTK* oferece diversas ferramentas e técnicas de pré-processamento de texto, incluindo "tokenização" (transformação em *tokens*), remoção de *stopwords*, *stemming* (redução de palavras à sua forma base) e lematização (redução de palavras a seus lemas). Essas etapas são fundamentais para preparar o texto bruto para análise, removendo ruídos e padronizando seu conteúdo.

Uma das características notáveis do *NLTK* é seu suporte a diferentes idiomas, incluindo o português brasileiro. Ele oferece modelos e recursos específicos para a língua desejada, tornando-o versátil para análise de texto em várias línguas.

O *NLTK* integra-se perfeitamente com técnicas de aprendizado de máquina, permitindo que os usuários construam modelos de classificação de texto e análise de sentimentos. Também oferece suporte para algoritmos de aprendizado supervisionado e não supervisionado, facilitando a criação de modelos de categorização de documentos, detecção de tópicos e análise de sentimentos.

Nossa escolha em utilizar a biblioteca do *NLTK* ocorreu, pois ela é uma ferramenta muito conhecida e de amplo uso no campo de processamento de linguagem natural, possuindo assim uma estrutura bem consolidada, atualizações frequentes, uma comunidade ativa de desenvolvedores e usuários e uma documentação bastante rica além de recursos on-line. Os usuários podem encontrar tutoriais, exemplos de código e suporte da comunidade para auxiliar na utilização eficaz desta poderosa ferramenta, fomentando discussões e ajudando a resolver pequenos problemas de utilização que podem ocorrer.

2.2 GRANDES BASES DE TEXTO

Para realizar análises e pesquisas na área de *PLN* é necessário possuir uma base de dados para treinamento e teste de acurácia do modelo treinado. Quanto maior e mais variada for sua base, maiores serão as possibilidades de associação entre palavras, pois mais usos e contextos diferentes para uma determinada palavra serão encontrados. A seguir, seguem algumas das bases de dados utilizadas para testes e realização da pesquisa.

2.2.1 Wikipedia

A Wikipedia Dumps (FOUNDATION, 2024), é uma base de dados formada por todos os tópicos e páginas, com artigos, imagens e históricos de revisões, existentes na Wikipedia no formato *wikitext* e *XML*. É atualizada periodicamente no site Wikimedia. Inclusive, existem versões desta base de dados em vários idiomas, tantos quais a quantidade de domínios nos quais a Wikipedia está presente.

Os *dumps* da Wikipedia são de acesso aberto e podem ser facilmente baixados diretamente do site ou de repositórios públicos. Isso os torna acessíveis a pesquisadores e desenvolvedores. Eles estão disponíveis para serem utilizados em pesquisas, projetos educacionais, desenvolvimento de aplicativos ou em qualquer outro uso que se queira. Além disso, os dados estão em formato estruturado, facilitando o processamento e a extração de informações. Os artigos são organizados em páginas, com metadados como título, texto principal, categorias e links para outros artigos. Isso permite uma fácil manipulação dos dados.

Ela foi escolhida, pois a Wikipedia é tido como a maior enciclopédia virtual do mundo e tendo isso em vista, possui uma grande variedade de temas e palavras-chave associadas aos textos ali presentes. Além disso, possui uma comunidade extremamente ativa que garante que as informações relacionadas as suas palavras-chave não estejam erradas, o texto tenha uma probabilidade muito grande de estar ortograficamente correto e seja sempre muito bem organizado.

2.2.2 Clueweb

É uma base de dados proveniente de um projeto inovador criado através do The Lemur Project (HARPOLE et al., 2005) em conjunto com a Universidade de Carnegie Mellon e algumas empresas como o Google, IBM e o Yahoo. O foco do projeto eram pesquisas sobre algoritmos e métodos para busca e indexação de informações relacionadas a tecnologias de Processamento de Linguagem Natural. A base de dados textuais, chamada de *ClueWeb*, consistia, na época que foi apresentada, em um conjunto de 733.019.372 páginas em 10 línguas diferentes, entre elas o português brasileiro. E foi a principal base de dados utilizada no treinamento do *NELL*. Atualmente, já existem diversas versões do *ClueWeb* coletadas durante o passar dos anos.

Utilizar o *ClueWeb* apresentou desafios técnicos significativos devido ao seu tamanho e complexidade. A preparação e o processamento desses dados requerem recursos de hardware substanciais e técnicas eficientes de armazenamento e recuperação. Além disso, é crucial lidar com a heterogeneidade dos dados da web, que inclui diversos formatos de documentos e informações não estruturadas.

O arquivo de dados do *ClueWeb* pode ser obtido por meio do site oficial, realizando um breve cadastro de requisição com os motivadores para utilizá-lo ou através de instituições de pesquisa que mantêm cópias desses dados.

2.2.3 Macmorpho

O Mac-Morpho (ALMEIDA; NUNES, 2003) é uma das maiores bases de dados textuais em português. Esta vasta coleção de textos não apenas contém representações da riqueza de palavras e frases em português, mas também oferece uma profunda análise sobre a

estrutura e o contexto da língua. Ele é composto por um grande apanhado de notícias de jornais e revistas coletados da web.

Ao analisar o Mac-Morpho podemos examinar como as palavras interagem em diferentes contextos, conforme agrupadas em frases e como a sintaxe varia em diversos cenários linguísticos. Imagine poder desvendar detalhes da língua portuguesa por meio de um vasto conjunto de dados que abrange desde o uso informal e cotidiano até textos mais formais e técnicos.

Para os computadores, essa base de dados é como uma mina de ouro de informações. Ao processar e entender os padrões no Mac-Morpho, os algoritmos de processamento de linguagem natural podem aprender como as palavras são usadas em diferentes situações. Isso é crucial para tradução automática, por exemplo, onde o significado de uma palavra muitas vezes depende do contexto em que é usada.

Além disso, o Mac-Morpho é amplamente utilizado para treinamento na construção de *chatbots* e assistentes de voz em português. Compreender os detalhes da língua é essencial para que essas tecnologias possam interagir de forma eficaz e natural com os usuários, entendendo perguntas complexas e elaborando respostas de maneira inteligente e natural.

Existem muitas bases, como a Wikipedia Dumps citada anteriormente, que possuem uma vasta gama de textos, porém foi através do Mac-morpho que percebemos que não é apenas a quantidade e o volume de palavras que importa ao analisarmos um texto, mas que a qualidade dos textos também é fundamental. Podemos dizer isso, pois além dele possuir um grande acervo também possui estruturação e categorização para textos e palavras, o que faz com que o aprendizado evolua de forma mais rápida.

O Mac-morpho foi a nossa base de textos inicial e fundamental para análises e aprendizado, pois devido ao seu tamanho limitado em comparação com as outras executava significativamente mais rápida, o que era fundamental nos primeiros testes. Porém, o mesmo motivo que fez com que usássemos fez também com que deixássemos de usar, pois comparada com as demais o seu volume de texto era ínfimo.

2.3 IDENTIFICADORES DE CONTEXTO

São ferramentas utilizadas dentro do campo de Processamento de Linguagem Natural para que se possa analisar e extrair contextos de frases ou parágrafos do texto. A seguir, algumas ferramentas utilizadas na realização das pesquisas.

2.3.1 Word2vec

O Word2Vec (MIKOLOV et al., 2013) é uma ferramenta de aprendizado de máquina desenvolvida por Tomas Mikolov, no Google, em 2013, e que se tornou amplamente adotada para identificar contexto semântico das palavras em um espaço vetorial. Essa ferra-

menta possui modelos relacionados entre si para produzir contextos entre palavras, consistindo em redes neurais de duas camadas, treinadas para reconstruir contextos linguísticos de palavras. Esse funcionamento é baseado na ideia de que as palavras que ocorrem em contextos semelhantes tendem a ter significados semelhantes. Para criar esses vetores de palavras, o Word2Vec utiliza redes neurais de duas camadas (CBOW: Continuous Bag of Words e Skip-gram) treinadas em grandes corpus de texto para prever as palavras com base na palavra-alvo.

A ferramenta recebe um corpus de texto e produz vetores multidimensionais, onde cada palavra única no texto é associada ao seu próprio vetor. Palavras com contextos semelhantes terão vetores próximos no espaço.

O treinamento do Word2Vec resulta em vetores densos de palavras, onde palavras semanticamente semelhantes ficam próximas no espaço vetorial. Isso permite que o modelo capture relações e analogias entre palavras.

2.3.1.1 Embeddings

As representações vetoriais das palavras são chamadas *embeddings*.

O Word2Vec gera uma matriz de *embeddings* como resultado. Esta matriz contém vetores que representam as palavras do vocabulário. Cada linha da matriz corresponde ao vetor de uma palavra específica.

O Word2Vec por si só, não é considerado um classificador, mas suas *embeddings* podem ser integradas em diversos modelos de machine learning para realizar tarefas de classificação e outras aplicações avançadas de processamento de linguagem natural, ou seja, ele é usado para treinar representações vetoriais de palavras, que são então usadas em diversos modelos para melhorar a precisão dos testes e classificações.

O nosso foco durante o uso do Word2Vec foi analisar palavras, porém ele vai muito além. Ele pode ser utilizado para processar itens de diversos tipos e gerar uma matriz referente aos itens analisados.

Algo interessante a se falar é que com os resultados de cada item avaliado é possível começar a tratá-los como vetores matemáticos, ou seja, podemos fazer operações, comparações e análises entre eles.

Nas imagens abaixo podemos ver dois exemplos que nos permitem comparar a relação entre algumas palavras.

Na Figura 3, podemos perceber que o vetor resultante entre os vetores homem e mulher, é similar ao vetor resultante entre as palavras rei e rainha. Desta forma é possível entender que a relação entre as palavras rei e rainha é bem parecida com a relação homem e mulher.

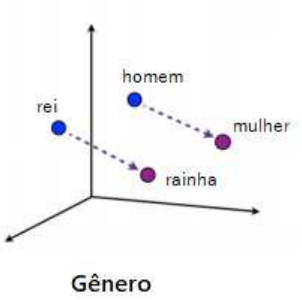

Figura 3 – Comparação entre os vetores homem - mulher e rei - rainha

Na Figura 4 é possível seguir a mesma linha de raciocínio e entender que o mesmo ocorre entre as palavras andar e andando e nadar e nadando, onde a relação entre eles é a conjugação verbal.

Figura 4 – Comparação entre os vetores nadar - nadando - andar - andando.

2.3.1.2 Janela de contexto

Quando o modelo encontra uma palavra em um texto, ele olha para as palavras próximas para entender melhor o significado daquela palavra específica. As janelas de contexto são super importantes porque auxiliam o modelo a capturar o significado das palavras com base em seu uso real no texto. Se uma palavra está sempre perto de outras palavras específicas, o modelo aprende que elas estão relacionadas. Isso é como aprender que "maçã" está frequentemente perto de "fruta" ou "comer", o que nos ajuda a entender que "maçã" é algo que comemos e que é uma fruta. O tamanho da janela de contexto pode variar. Se você usar uma janela pequena, como 2 palavras antes e 2 palavras depois da palavra central, você captura um contexto mais imediato e específico. Com uma janela maior, você pode captar um contexto mais amplo e entender como a palavra se relaciona com mais partes da frase. Podemos ver um exemplo na Figura 5.

Dada a fórmula 2.1:

$$p(w[t+j] \mid w[t]) \tag{2.1}$$

- w[t] É a palavra central na posição t em um texto.

- w[t+j] É a palavra no contexto, localizada a j posições à frente ou atrás da palavra w[t]
- p(w[t+j] | w[t]) É a probabilidade de observar a palavra w[t+j] dada a palavra w[t] como palavra central

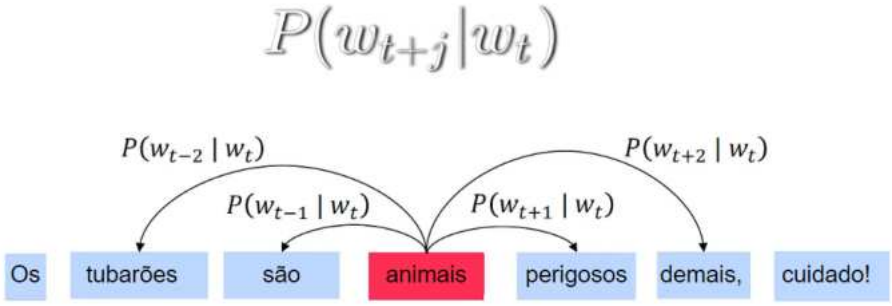

Figura 5 – Representação da formação da janela de contexto.

2.3.1.3 Modelos de representação de palavras

Quando precisamos transformar textos em vetores, qual seria a forma mais simples? O Word2Vec utiliza algumas abordagens principais para aprender essas representações: o modelo Bag of Words (BoW), Continuous Bag of Words (CBOW) e o modelo Skip-gram. Tanto o CBOW e o Skip-gram utilizam janelas de contexto para analisar as palavras ao redor de uma palavra central, auxiliando o modelo a capturar o significado das palavras com base nas palavras que aparecem próximas a elas no texto.

2.3.1.3.1 Bag of words

No modelo *Bag of Words (BoW)*, cada documento é representado como um conjunto de palavras únicas, ignorando a ordem das palavras. Este método é útil para tarefas simples de contagem de palavras e análise de frequência no texto. Como o nome tenta representar, é como se fosse realmente uma bolsa de palavras, onde todas são guardadas e retiradas quando solicitadas para representar uma frase.

O *Bag of Words (BoW)* usa a forma mais simples de representação, também conhecida como *one-hot encoding*. Esta representação vetorial é mais simples, onde cada palavra é representada por um vetor cheio de zeros, exceto por um único "1" que marca a posição da palavra.

Desta forma podemos seguir com alguns exemplos: Se nos basearmos na frase: "O cachorro é um animal doméstico"

- "O cachorro é um animal doméstico"

- “O leão é um animal selvagem”

Primeiro precisamos limpar os textos, ou seja, prepará-los para serem analisados, assim podemos começar explicando o que são as *stopwords*.

As *stopwords* são palavras muito comuns em um idioma que geralmente são filtradas ou removidas durante o processamento de textos porque não contribuem significativamente para o significado de uma frase. Alguns exemplos de *stopwords* da língua portuguesa são: artigos definidos, artigos indefinidos, preposições, conjunções, alguns advérbios, alguns pronomes, etc. Essas palavras são frequentes na linguagem cotidiana, mas raramente transmitem informação discriminativa em tarefas de processamento de linguagem natural.

Ao considerar os exemplos dados, as palavras “o” e “um” são *stopwords*. Elas são tão genéricas e frequentes que podem ser removidas sem perdermos muita informação significativa na compreensão do texto.

- “O”: pode ser removida porque é um artigo definido que não influencia significativamente na descrição do animal.
- “Um”: Também pode ser removida por ser um artigo indefinido que não acrescenta detalhes sobre ser um animal doméstico ou selvagem.

Desta forma, após remover as *stopwords* teremos um texto limpo e pronto para análise. Assim podemos seguir com representação vetorial.

Neste exemplo, podemos adotar um vetor de 6 dimensões para representar o nosso vocabulário. Na Tabela 1 podemos ver como seria a representação de cada palavra vetorialmente.

Palavra	Vetor
cachorro	[1, 0, 0, 0, 0, 0]
é	[0, 1, 0, 0, 0, 0]
animal	[0, 0, 1, 0, 0, 0]
doméstico	[0, 0, 0, 1, 0, 0]
leão	[0, 0, 0, 0, 1, 0]
selvagem	[0, 0, 0, 0, 0, 1]

Tabela 1 – Representação vetorial de palavras

Agora podemos simplificar cada frase em um único vetor, considerando a representação vetorial adotada antes. Somamos os vetores de cada palavra para obter o vetor resultante da frase, obtemos o resultado representado na Tabela 2.

Frase	**Valores**
O cachorro é um animal doméstico	[1, 1, 1, 1, 0, 0]
O leão é um animal selvagem	[0, 1, 1, 0, 1, 1]

Tabela 2 – Tabela de frases e seus valores

A simplificação das frases em vetores únicos permite capturar o significado essencial de cada sentença de maneira compacta, pois cada palavra é representada por um vetor pré-definido e, ao somar os vetores das palavras que compõem a frase, obtemos um vetor resultante que encapsula a informação da frase.

Isso funciona, mas tem um problema grande: é muito ineficiente com vocabulários grandes. Imagina ter um vetor gigante para cada palavra em um dicionário enorme!

2.3.1.3.2 Continuous bag of words

O modelo *Continuous Bag of Words (CBOW)* é um avanço em relação ao *Bag of Words (BoW)*, prevendo uma palavra-alvo com base nas palavras ao redor em um contexto específico. Em vez de contar palavras como o *BoW*, o *CBoW* usa vetores densos para representar cada palavra no vocabulário. Durante o treinamento, o modelo ajusta os parâmetros para avaliar a probabilidade de prever corretamente a palavra-alvo dada as palavras do contexto.

Em vez de usar vetores enormes como os *one-hot encoding*, o Word2Vec cria vetores menores e densos. Esses números não só representam a palavra, mas capturam os significados semânticos. Por exemplo, "cachorro" e "leão" terão vetores próximos no espaço vetorial porque frequentemente aparecem em contextos semelhantes, como ilustrado na Figura 6.

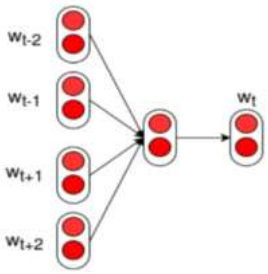

Figura 6 – Exemplo do funcionamento do modelo CBOW. (Fonte: Wikimedia Commons)

2.3.1.3.3 Skip-gram

Por outro lado, o modelo Skip-gram do Word2Vec faz basicamente o contrário do CBOW, ou seja, tenta prever as palavras ao redor de uma palavra-alvo, capturando assim o contexto em que a palavra ocorre, conforme mostrado na Figura 7

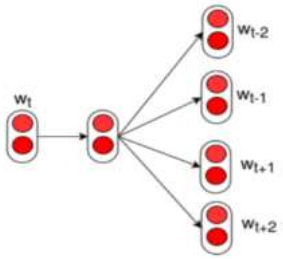

Figura 7 – Exemplo do funcionamento do modelo Skip-gram. (Fonte: Wikimedia Commons)

2.3.1.3.4 Similaridade entre palavras

A principal técnica para comparar a similaridade entre os vetores de duas palavras é a similaridade de cossenos. Para calcular podemos usar a fórmula 2.2.

$$\text{Similaridade de Cossenos} = \cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\|\|\vec{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.2}$$

Dado que o Word2Vec cria vetores para representar cada palavra, essa técnica permite verificar o quão similares os vetores das palavras são.

Os valores extremos de um cosseno são 1 e -1. Desta forma, se o cosseno do ângulo entre os vetores é próximo de 1, a similaridade é alta e as palavras são parecidas.

Conforme o valor obtido, podemos analisar da seguinte forma:

- Valor próximo a 1: Vetores apontam na mesma direção (palavras muito semelhantes).
- Valor próximo a 0: Vetores são ortogonais (palavras não têm relação).
- Valor próximo a -1: Vetores apontam em direções opostas (palavras opostas).

Como mencionado anteriormente, após treinar o modelo Word2Vec, podemos realizar operações vetoriais simples para explorar relações semânticas entre palavras.

No exemplo representado através da Figura 8, podemos supor que queremos completar a frase “A mulher daquele rei foi uma grande ...”, mas não conhecemos a palavra “rainha”. Podemos usar o modelo treinado para descobrir qual palavra tem o sentido desejado neste contexto através de uma operação vetorial.

Primeiro, usamos a palavra “rei” como base. Em seguida, fazemos a operação “rei” menos “homem” para capturar a ideia de uma figura monárquica que não seja do sexo masculino, conforme mostrado na Figura 9. Por fim, adicionamos a palavra “mulher” ao resultado, como visto na Figura 10. Desta forma, o resultado deve se aproximar ou

ser idêntico ao vetor da palavra “rainha”, como ilustrado na Figura 11 e demonstrado na Figura 12.

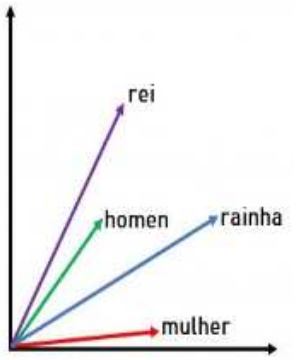

Figura 8 – Podemos avaliar os vetores referentes a cada palavra do nosso dicionário.

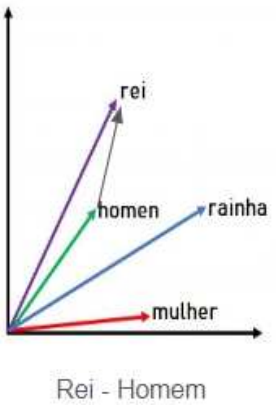

Figura 9 – Primeiro subtraímos o vetor [homem] do vetor [rei], assim teremos uma figura monarca que não se associe a homem.

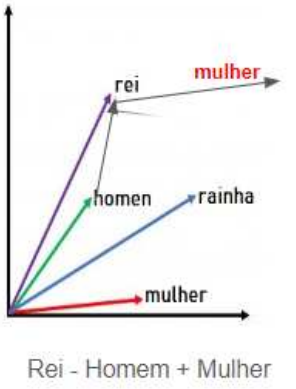

Figura 10 – Em seguida, adicionamos o vetor [mulher] a operação anterior.

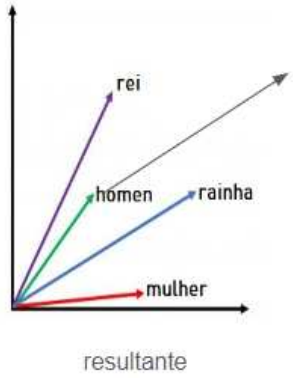

Figura 11 – O vetor resultante se refere a operação de [rei] - [homem] + [mulher].

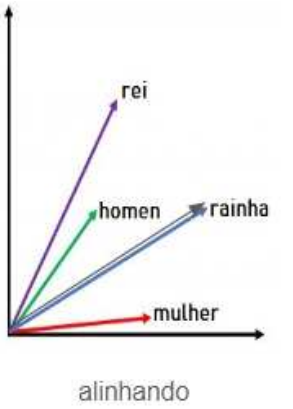

Figura 12 – O vetor resultante das operações, se colocado na origem, se assemelha ao vetor [rainha].

Desta forma, a similaridade de cosseno entre o vetor resultante da operação descrita e o vetor da palavra "rainha" pode ser alta, indicando que a palavra "rainha" é uma boa escolha para completar a frase dada.

2.3.1.4 Comunidade

A utilização do Word2Vec foi escolhida devido à sua ampla adoção por profissionais de Processamento de Linguagem Natural e ao fato de ter sido desenvolvida no centro de pesquisas do Google. É uma biblioteca com uma comunidade forte, que está sempre em processo de atualização e aperfeiçoamento. Além de possuir diversos trabalhos e comunidades de discussão sobre a mesma.

2.3.2 Paragraph2vec

O Paragraph2Vec (LE; MIKOLOV, 2014) é uma ferramenta criada pelo Google que utiliza os mesmos moldes do Word2Vec funcionando como uma extensão do Word2Vec, possuindo modelos relacionados entre si que produzem contextos entre palavras, sendo

esses modelos baseados em redes neurais de três camadas, treinadas para reconstruir contextos linguísticos. Ele foi construído para representar documentos inteiros como vetores, incluindo o contexto semântico do texto.

A principal limitação do Word2Vec é que ele considera apenas uma palavra por vez, sem considerar o contexto mais amplo em que essa palavra aparece. Isso pode ser problemático em situações onde o significado de uma palavra depende fortemente das palavras ao seu redor. O Paragraph2Vec foi desenvolvido para resolver essa limitação.

Funciona organizando frases em vetores de palavras e depois agrupando tais vetores com base em suas características, a fim de formar um parágrafo e adquirir o contexto do mesmo entre outros parágrafos ou mesmo das frases entre si em um mesmo parágrafo. Isso permite que o modelo capture a semântica não apenas ao nível de palavra, mas também ao nível de documento, o que é essencial para tarefas que exigem uma compreensão mais ampla do texto.

Uma das principais aplicações do Paragraph2Vec é na análise de sentimentos. Ele pode, por exemplo, ser usado para avaliar o tom de um conjunto de resenhas de produtos, ajudando a identificar se os clientes estão satisfeitos ou insatisfeitos. Além disso, ele é amplamente utilizado na classificação de documentos, onde pode categorizar automaticamente grandes volumes de texto em diferentes tópicos ou gêneros e também em sistemas de recomendação, onde pode sugerir conteúdos relevantes com base no histórico de leitura ou preferências do usuário.

Ele exige grandes volumes de dados para treinamento eficaz, ainda maiores do que os necessários para treinar modelos no *Word2Vec*. Isso se deve à complexidade adicional de representar contextos ao nível de documento. Além disso, o desempenho do *Paragraph2Vec* é altamente sensível à qualidade dos textos utilizados para treinamento. Textos bem escritos proporcionam melhores resultados, enquanto dados mal estruturados ou com erros ortográficos podem comprometer a eficácia do modelo escolhido.

3 METODOLOGIA

Primeiramente precisávamos entender do que se tratava Processamento de Linguagem Natural. Para entender melhor como funciona o Processamento de Linguagem Natural (*PLN*), começamos utilizando o software WEKA (WITTEN et al., 2016), mas logo percebemos a necessidade de uma ferramenta mais robusta. Foi então que decidimos utilizar o *NLTK*, um ótimo *toolkit* com diversas ferramentas e disponível para Python. No seu *toolkit* existiam ferramentas tanto de análise, como de processamento e avaliação de resultados. Por bastante tempo achávamos que ela era suficiente para o nosso estudo, porém após começarmos a usar o Word2Vec percebemos o quão eficaz os seus modelos são e que ambas, tanto o *NLTK* como o Word2Vec podem ser usadas em paralelo para aprimoramento dos resultados. Ao longo do processo, o tempo todo estávamos testando diversos corpus para avaliar o quão diferente poderiam ser os resultados a partir de fontes diversas. Testamos alguns e apresentaremos alguns resultados comparativos.

A escolha dessas ferramentas foi baseada em suas capacidades específicas e na forma como complementavam nosso projeto. O WEKA foi escolhido pela sua facilidade de uso e ampla gama de algoritmos, o que foi útil para nossos primeiros experimentos. O *NLTK* se destacou na etapa de pré-processamento, essencial para limpar e preparar os dados textuais. Já o Word2Vec foi crucial para a etapa de vetorização, permitindo que nossos modelos aprendessem representações ricas e úteis das palavras. Em resumo, cada ferramenta desempenhou um papel específico e importante em nossa metodologia, contribuindo para a robustez e a eficiência do nosso trabalho. Na próxima parte, vamos discutir os critérios de avaliação que utilizamos para medir a eficácia dos modelos treinados.

3.1 SELEÇÃO DO CORPUS

A seleção de corpus não foi necessariamente o primeiro passo que demos, na verdade, foram vários passos contínuos que fizemos em todas as etapas do trabalho. Pois a cada etapa descobríamos a importância de um corpus robusto e bem estruturado, mas, ao mesmo tempo, como não conhecíamos os melhores corpus disponíveis nem qual usar, acabamos testando alguns.

Ao longo de todo o processo trocávamos de *corpus* por vários momentos, seja para avaliar os resultados, ou mesmo por uma questão de desempenho, pois em corpus menores tinham resultados mais rápidos para testar algum pequeno ajuste na entrada dos dados.

Por isso, apesar da seleção de corpus não ter sido a primeira etapa, ela é um bom assunto para começar explicando os nossos próximos passos e apesar do português não ser tão estudado como outras línguas, existem diversos corpus disponíveis.

Muitas vezes um mesmo corpus possui variações ou atualizações que fazem com que

possamos considerá-lo um novo corpus. Por exemplo, o Corpus do Clueweb sofre atualizações periódicas que alteram seu tamanho significativamente. Já ao acessar o corpus da Wikipedia Dumps vem segmentado em diversos arquivos separados por categorias, o que faz dele vários corpus em um, ou seja, ao quebrar um texto no meio podemos criar um novo corpus, porém a eficiência dele também será diminuída.

Porém, nem todos os corpus são de fácil acesso ou não são tão bem estruturados. Ao longo do processo de aprendizagem e descoberta, testamos e tentamos utilizar alguns corpus que não falaremos em tantos detalhes, pois os resultados foram ínfimos ou em alguns casos nem conseguimos importar os dados devido à estrutura complexa, eis alguns exemplos:

- Floresta Sintática Corpus,
- CRPC - Corpus de Referência do Português Contemporâneo

Dentre os *corpus* que mais utilizamos, o *Mac-Morpho* que consiste em coleções de jornais e revistas brasileiras, foi nossa primeira base de dados que usamos bastante, pois ele estava incluído no *toolkit* do *NLTK* e seu uso era extremamente simples, pois bastava fazer o download e a importação por meio do próprio código Python que estávamos usando.

No entanto, concluímos que, apesar de sua facilidade de acesso, o *Mac-Morpho* apresentava limitações significativas em termos de quantidade e diversidade de conteúdos. Focado em notícias sociais de sua época, o *Mac-Morpho* poderia conter terminologias desatualizadas ou muito específicas, o que não atenderia plenamente às nossas necessidades.

Buscando *corpus* maiores, os *dumps* da Wikipedia nos proporcionaram um excelente ponto de partida. Na época, com aproximadamente 3GB de dados, essa base de artigos em português foi crucial para os nossos experimentos. A diversidade de conteúdos disponíveis na Wikipedia, abrangia uma ampla gama de tópicos e estilos de escrita. Desta forma garantiu que nosso modelo fosse exposto a uma variedade rica de contextos linguísticos.

Após algumas reuniões, conhecemos o professor E. Hruschka e ele nos apresentou ao *ClueWeb*, um extenso corpus com cerca de 40GB de textos em várias línguas, incluindo o português. No entanto, como o acesso ao *ClueWeb* requeria uma solicitação específica e era extremamente pesado para fazer análises, decidimos utilizar também os Wikipedia Dumps para efeitos de comparação. Esses *dumps* são grandes o suficiente e incluem uma vasta quantidade de textos diversificados, o que os torna ideais para os nossos testes com *NLTK* e Word2Vec.

Com o corpus em mãos, começamos os primeiros testes usando o *NLTK*. Inicialmente, focamos em tarefas básicas de pré-processamento, como remoção de *stopwords*, *lematização* e *tokenização*. Essas etapas são essenciais para preparar os dados textuais antes de serem alimentados nos modelos de aprendizado. A simplicidade e eficiência do *NLTK* no pré-processamento nos permitiram limpar e estruturar os textos de maneira eficaz, facilitando o trabalho com as etapas subsequentes.

Paralelamente, exploramos o Word2Vec para criar representações vetoriais das palavras presentes nos textos. O Word2Vec é uma ferramenta poderosa para gerar *embeddings* de palavras, ou seja, transformar palavras em vetores que capturam seus significados semânticos. Ajustamos parâmetros como o número de iterações e a dimensão dos vetores para otimizar a qualidade dessas representações.

Os resultados dos testes iniciais foram promissores. Usando o Wikipedia Dumps e aplicando o Word2Vec, conseguimos criar *embeddings* de alta qualidade, que serviram como base para os modelos de aprendizado de máquina. A riqueza e a diversidade dos textos da Wikipedia se refletiram na qualidade dos *embeddings*, proporcionando uma representação detalhada e precisa das palavras em diversos contextos.

3.2 PRÉ-PROCESSAMENTO DE TEXTO

Após a obtenção do corpus adequado, a próxima etapa foi focar no pré-processamento dos textos para garantir que estivessem prontos para as análises mais aprofundadas. Utilizamos principalmente a biblioteca *NLTK*, que se mostrou essencial para diversas tarefas de limpeza e preparação dos dados textuais.

O pré-processamento dos textos foi dividido em etapas fundamentais para preparar nossos dados para as análises seguintes. A utilização da *NLTK* nos permitiu limpar, estruturar e transformar os textos de maneira eficiente, garantindo que estivessem prontos para serem utilizados em nossos modelos de aprendizado de máquina.

Entre as operações realizadas com o texto, destacamos a remoção de *stopwords*, *lematização*, *stemming*, e a remoção de pontuação e caracteres especiais. A remoção de *stopwords* foi crucial para eliminar palavras comuns que não contribuem significativamente para a análise, como artigos e preposições. A *lematização* nos ajudou a reduzir palavras à sua forma base ou radical, permitindo uma análise mais consistente e reduzindo a redundância. O *stemming* foi aplicado para cortar palavras até suas raízes, o que é particularmente útil para lidar com variações morfológicas.

Além disso, a remoção de pontuação e caracteres especiais garantiu que nossos dados fossem livres de elementos que poderiam distorcer os resultados da análise. Essas operações combinadas nos permitiram trabalhar com um corpus limpo e bem preparado, essencial para a eficácia dos algoritmos de aprendizado de máquina que utilizamos posteriormente.

A combinação das técnicas destas técnicas resultou em textos altamente padronizados e otimizados para o processamento. Essas técnicas permitiram criar um conjunto de dados limpo e organizado, pronto para ser utilizado em diversas aplicações de *PLN*.

3.3 IMPLEMENTAÇÃO DO MODELO

Primeiramente, os textos coletados passaram por um rigoroso processo de pré-processamento. Utilizamos o *NLTK* (Natural Language Toolkit) para realizar a limpeza e normalização dos dados. Este pré-processamento garantiu que os dados estivessem no formato ideal para serem utilizados pelo Word2Vec.

Com os dados limpos e estruturados, começamos a integração com o Word2Vec. Utilizamos a biblioteca *Gensim* (REHUREK; SOJKA, 2010), que oferece uma implementação eficiente do Word2Vec, facilitando o trabalho com grandes volumes de dados textuais. Nossa escolha pelos parâmetros iniciais do Word2Vec, baseados na versão de 2017, foi uma decisão estratégica para começar de forma simples e ajustável. Esses parâmetros incluíam o tamanho dos vetores, a janela de contexto, o número de iterações, o algoritmo de treinamento e a contagem mínima de palavras.

A integração envolveu alimentar o Word2Vec com os textos pré-processados e iniciar o treinamento do modelo. No início, enfrentamos algumas dificuldades com o desempenho, pois os valores padrões dos parâmetros de entrada eram muito altos e causaram lentidão em máquinas menos potentes. A solução foi reduzir os parâmetros a fim de agilizar as primeiras execuções. Com esta forma de abordagem inicial obtivemos resultados rápidos e conseguimos avaliar a eficácia dos parâmetros escolhidos.

Desta forma, aprendemos que a cada iteração devíamos ajustar e refinar os parâmetros, sempre monitorando a convergência do modelo e analisando a qualidade dos vetores de palavras gerados. Este processo iterativo foi essencial para otimizar o desempenho do modelo e alcançar resultados mais precisos. Em resumo, a metodologia adotada para integrar os dados pré-processados com o Word2Vec foi um passo meticuloso e estratégico, crucial para o sucesso do nosso projeto de análise e aprendizado de linguagem natural.

4 EXECUÇÃO E CÓDIGO

4.1 PLATAFORMA

Durante nossos testes, utilizamos alguns computadores para executar os algoritmos, pois como os resultados dos processos eram demorados, por vezes era necessário executar duas variações simultaneamente para ganhar tempo.

Para otimizar o uso dos recursos e evitar conflitos com outras tarefas, rodávamos frequentemente as aplicações em horários em que não precisávamos utilizar os computadores para outras atividades.

Os computadores que usamos para executar o trabalho tinham as seguintes especificações:

- **Computador 1**
  - Processador: Intel Core i5 2500
  - Memória RAM: 4 GB
  - Disco: HD 320 GB
  - Sistema Operacional: Windows 10
- **Computador 2**
  - Processador: Intel Core i7 2600
  - Memória RAM: 16 GB
  - Disco: HD 1 TB
  - Sistema Operacional: Windows 10

Um ponto fundamental do nosso trabalho foi a utilização da funcionalidade de importação e exportação de matrizes. Esta capacidade nos permitiu salvar o progresso e retomar o trabalho posteriormente, sem perder o que já havia sido processado.

Além desses comandos básicos, é importante destacar que ao importar e exportar modelos, é necessário salvar também os parâmetros usados durante o processo. Usar uma matriz previamente treinada com diferentes parâmetros pode gerar resultados inconsistentes nas novas iterações.

Para exportar uma matriz criada, podemos usar a função *save*:

```
model.save("modelName.model")
```

Código 4.1 – Exportar matriz.

É possível exportar os vetores de palavras para o formato de texto. Para exportar, utilize a função *save_ Word2Vec_ format*:

```
model.wv.save_Word2Vec_format("modelName_vectors.txt", binary=False)
```

Código 4.2 – Exportar vetores de palavras.

Para importar e reiniciar o desenvolvimento, podemos utilizar a função *load*:

```
model = gensim.models.Word2Vec.load("modelName.model")
```

Código 4.3 – Importar modelo.

Ao seguir esse procedimento, podíamos interromper a execução a qualquer momento e retomá-la mais tarde, preservando o progresso feito até aquele ponto. Isso foi especialmente útil quando precisávamos liberar os computadores para outras tarefas ou quando os processos se mostravam mais demorados do que o previsto. Desta forma as funções de importação e exportação de modelos através do *Gensim* foram fundamentais na nossa metodologia de trabalho. Elas nos permitiram gerenciar o tempo de execução de forma eficiente, garantindo que pudéssemos continuar nossos experimentos sem interrupções significativas e sem comprometer a integridade dos dados.

4.1.1 Tensorflow ou Gensim

Ao iniciar o uso com o Word2Vec tínhamos uma escolha a fazer: decidir qual biblioteca utilizar: *TensorFlow* (ABADI et al., 2016) ou *Gensim* (REHUREK; SOJKA, 2010).

O *TensorFlow* é uma conhecida biblioteca que vem ganhando cada vez mais reconhecimento e espaço entre a comunidade de IA. Do outro lado, a biblioteca *Gensim* já existe a bastante tempo e tida como uma biblioteca "estável" e forte uso pela comunidade.

Após pesquisas em meio a comunidade e cada uma das bibliotecas, entendemos que o desempenho com o *TensorFlow* era relatado como melhor e mais eficiente, porém para extrair toda a capacidade era preciso utilizar uma *GPU*, algo que não temos à nossa disposição. Um dos grandes trunfos do *TensorFlow* é justamente o suporte otimizado para *GPU*s, que acelera significativamente os processos de treinamento de modelos. Quando utiliza apenas a *CPU*, o desempenho do *TensorFlow* pode ser significativamente reduzido, especialmente para tarefas que exigem alto poder computacional e grandes volumes de dados. Sendo assim, a ausência de uma *GPU* dedicada em nosso ambiente de trabalho se torna um fator limitante para o uso eficiente do *TensorFlow*.

Por outro lado, a biblioteca *Gensim* é projetada para ser altamente eficiente no uso de *CPU*. Sua arquitetura é otimizada para processar grandes volumes de texto utilizando recursos de *CPU*, o que permite realizar operações complexas sem a necessidade de uma *GPU*. Além disso, o *Gensim* é conhecido por sua simplicidade e facilidade de uso, o que foi um ponto positivo para nós, especialmente considerando nosso pouco nível de familiaridade com essas ferramentas.

Apesar da escolha ter sido limitada ao poder de hardware e a facilidade de instalação, o *Gensim* é uma ferramenta robusta e foi muito eficaz para atender as nossas necessidades.

4.2 EXECUÇÃO

Após termos decidido que abordagens, bibliotecas e ferramentas que iriamos utilizar, bem como a decisão de como iriamos abordar cada passo da pesquisa, começamos a executar tudo o que foi previamente pensado para enfim experienciarmos e realizar testes.

4.2.1 Aprendizado Através do WEKA

Primeiramente, houve um esforço para se entender e aprender, de maneira mais teórica, como a máquina aprende. Além de termos como base o que foi ensinado e aprendido na cadeira de I.A presente na ementa curricular do curso, utilizamos como base o livro *Data Mining: Practical Machine Learning Tools and Techniques* (WITTEN et al., 2016), que é uma bibliografia de grande referência dentro da literatura sobre Inteligência Artificial e assuntos relacionados.

Após uma série de estudos sobre a parte teórica, avançamos para o passo prático, com a utilização do software WEKA (WITTEN et al., 2016), sugerido pelo livro, que facilitaria esse primeiro contato, tornando mais familiar o aprendizado de máquina. Junto a ele utilizamos uma base já famosa na literatura de Aprendizado de Máquina para aprendizado e entendimento, *Iris*(FISHER, 1936).

Essa base é composta de uma coleção de três tipos de flores do tipo Iris: Iris Setosa, Iris Versicolor e Iris Virgínica e contém dados sobre as 3 diferentes espécies, como tamanho das pétalas e distância média entre elas.

Nesse primeiro momento o objetivo era adquirir um entendimento um pouco mais profundo do funcionamento de um algoritmo que treina, identifica e categoriza uma determinada entrada de dados.

Treinamos o primeiro modelo com 80% da base deixando assim os 20% restantes para serem classificados e servirem como base para teste da acurácia do modelo. Como resultado, obtivemos que em 96% dos casos as flores foram identificadas de maneira correta, com uma margem média de erro absoluto inferior a 1% como é possível ver na Figura 13.

```
Correctly Classified Instances         144               96      %
Incorrectly Classified Instances         6                4      %
Kappa statistic                          0.94
Mean absolute error                      0.035
Root mean squared error                  0.1586
Relative absolute error                  7.8705 %
Root relative squared error             33.6353 %
Total Number of Instances              150

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure
                 0.98      0          1         0.98      0.99
                 0.94      0.03       0.94      0.94      0.94
                 0.96      0.03       0.941     0.96      0.95
Weighted Avg.    0.96      0.02       0.96      0.96      0.96

=== Confusion Matrix ===

  a  b  c   <-- classified as
 49  1  0 |  a = Iris-setosa
  0 47  3 |  b = Iris-versicolor
  0  2 48 |  c = Iris-virginica
```

Figura 13 – Resultados da análise do WEKA com a base de dados Iris

Em seguida, em um segundo teste, foi gerada uma base própria pseudorrandômica, baseada nos dados provenientes da base Iris, anteriormente utilizada para treinamento e analise, onde o mesmo modelo de treinamento foi utilizado, sendo 80% da base para treinamento do modelo e os 20% restantes para serem utilizados como teste de acurácia. Mais uma vez, foi observado que o resultado adquirido obteve porcentual de acerto de 96,3%, com uma margem média de erro absoluto inferior a 1%, sendo esse resultado muito similar ao anterior, como é possível ver nas Tabelas 3 e 4:

Tipo	Dado
Iris-setosa	[4.6, 3.2, 1.4, 0.2]
Iris-versicolor	[5.7, 3.0, 4.2, 1.2]

Tabela 3 – Exemplo de dados da flor Iris.

Resultado (predição)	Dado
Iris-setosa	[4.9, 3.1, 1.5, 0.1]
Iris-setosa	[5.5, 4.2, 1.4, 0.2]
Iris-setosa	[4.9, 3.1, 1.5, 0.2]
Iris-setosa	[4.6, 3.6, 1.0, 0.2]
Iris-setosa	[5.7, 4.4, 1.5, 0.4]
Iris-setosa	[5.0, 3.5, 1.3, 0.3]
Iris-setosa	[5.2, 3.4, 1.4, 0.2]
Iris-versicolor	[6.5, 2.8, 4.6, 1.5]
Iris-setosa	[5.4, 3.9, 1.3, 0.4]
Iris-versicolor	[5.5, 2.4, 3.7, 1.0]
Iris-versicolor	[6.7, 3.1, 4.7, 1.5]
Iris-versicolor	[5.6, 3.0, 4.1, 1.3]
Iris-setosa	[5.0, 3.6, 1.4, 0.2]
Iris-setosa	[4.8, 3.4, 1.6, 0.2]
Iris-setosa	[5.3, 3.7, 1.5, 0.2]
Iris-setosa	[4.7, 3.2, 1.3, 0.2]
Iris-setosa	[4.4, 3.2, 1.3, 0.2]
Iris-setosa	[5.0, 3.3, 1.4, 0.2]
Iris-setosa	[4.4, 3.0, 1.3, 0.2]
Iris-setosa	[4.7, 3.2, 1.6, 0.2]

Tabela 4 – Resultados da análise do WEKA com a base de dados Iris pseudorrandômica.

A análise dos resultados tanto da primeira rede treinada quanto da segunda, com a base pseudoaleatória, nos ajudaram a entender melhor como funciona o treinamento para o aprendizado de uma rede neural e como ela classifica os dados baseados num treinamento prévio, mesmo utilizando-se uma base de dados muito simples.

4.2.2 Importando Corpus

4.2.2.1 Macmorpho

Primeiro, importamos a base de dados *Mac-Morpho* e realizamos o pré-processamento necessário. Para ambas as tarefas utilizamos o *toolkit* da biblioteca *NLTK*. A forma mais simples de obter uma corpus para testes é fazendo o download diretamente a partir do código em desenvolvimento. O *NLTK* possui acesso a diversas bases pré-organizadas em sua biblioteca, mas o download e a importação são feitas sob demanda para salvar espaço em disco. O download do *corpus* só precisa ser feito uma vez. Após isso ele será salvo no cache do Python e estará disponível para outras aplicações.

Podemos fazer o download e a importação usando a seguinte série de comandos:

```
#importação
import nltk

#definição do corpus que será utilizado
from nltk.corpus import mac_morpho

#download
nltk.download('mac_morpho')

#carregamento das sentenças
texto_corpus = mac_morpho.sents()

# a partir de agora já podemos processar as frases e criar um modelo

# podemos processar o texto com alguma ferramenta ou com código próprio
texto_pre_processado = pre_processar_texto( texto_corpus )

# criamos um modelo com o word2vec
model = Word2Vec (texto_pre_processado, ... )
```

Código 4.4 – Importação e download de corpus.

4.2.2.2 Wikipedia

Para usar os dados da Wikimedia Dumps como *corpus* no Word2Vec utilizando a biblioteca *Gensim*, podemos seguir alguns passos. Primeiro, precisamos baixar os dados da Wikimedia Dumps. Os *dumps* são grandes arquivos que contêm todo o conteúdo da Wikipedia e podem ser baixados facilmente em <https://dumps.wikimedia.org>.

Os arquivos vêm compactados no formato *bzip* em alguns casos, conforme o desempenho do computador, é necessário descompactá-los antes de usar.

Após descompactados, os conteúdos estarão no formato *XML* e precisamos de textos não formatados para iniciar o processamento, para isso, existem algumas formas de extrair o conteúdo dos arquivos após o download.

Dentre as principais bibliotecas para extração de *XML* temos: a *Wikicorpus*, a *BeautifulSoup* e *mwxml*.

Wikicorpus é uma ferramenta do *Gensim* projetada especificamente para processar *dumps* da Wikipedia, convertendo artigos em textos limpos e prontos para uso.

A *mwxml* também é especializado em *dumps* da Wikipedia, permitindo acesso detalhado a todas as informações do *XML* e oferecendo controle granular, mas exige mais configuração para funcionar.

Já *BeautifulSoup* é uma biblioteca genérica para *parsing* de *HTML* e *XML* , possui mais flexibilidade para extrair dados *XML* , mas não é otimizada para a estrutura específica dos *dumps* da Wikipedia.

Escolhemos usar a *Wikicorpus* da biblioteca *Gensim*, por já ser integrada.

A função *Wikicorpus* é usada para processar e transformar *dumps* da Wikipedia em um *corpus* de textos.

```
wiki = WikiCorpus('ptwiki-latest-pages-articles.xml.bz2')
```

Código 4.5 – Carrega transforma e processa base Wikipedia.

Com o atributo wiki.metadata podemos incluir metadados (como títulos) junto aos textos dos artigos.

```
wiki.metadata = True
```

Código 4.6 – Flag para inclusão de metadados.

Podemos então, iterar sobre os textos dos artigos e seus metadados com um *looping*, assim podemos organizar os textos da forma como desejarmos, incluindo fazer a limpeza a organização.

```
from gensim.corpora.wikicorpus import WikiCorpus

wiki = WikiCorpus('ptwiki-latest-pages-articles.xml.bz2')

wiki.metadata = True

for text, (page_id, title) in wiki.get_texts():
    print("ID da pagina:", page_id)
    print("Titulo do artigo:", title)
    print("Texto do artigo:", text)
    texto_corpus += text

# a partir de agora podemos processar as frases e criar um modelo

# podemos processar o texto com alguma ferramenta ou com código próprio
texto_pre_processado = pre_processar_texto( texto_corpus )

# criamos um modelo com o word2vec
model = Word2Vec (texto_pre_processado, ... )
```

Código 4.7 – Iteração principal para organizar o texto.

4.2.2.3 Clueweb

A obtenção do acesso é um pouco diferente dos demais. Como o projeto é mantido pela Carnegie Mellon University, a instituição é responsável pela coleta, processamento e distribuição do *corpus*. Desta forma o acesso aos dados é regulamentado e criterioso. Logo, quem desejar utilizar o *corpus* precisará solicitar acesso e seguir as instruções no site oficial para obter os arquivos.

Os dados do ClueWeb09 estão em formato de arquivo *WARC*, neste formato, cada arquivo é equivalente a um contêiner digital e guarda dezenas de milhares de páginas da web (cerca de 40.000 por arquivo). Para reduzir o tamanho dos arquivos, a compactação é feita em formato *gzip*.

Para iniciar a leitura será necessário descompactar os arquivos ou utilizar a biblioteca *WARC* do Python para ler esses arquivos.

Como o Clueweb é separado em vários arquivos, ao usar a biblioteca *WARC*, será necessário iterar sobre cada arquivo a fim de obter os seus conteúdos. Por exemplo:

```
def read_clueweb09("clueweb09_0001.warc.gz"):
    with open("clueweb09_0001.warc.gz", 'rb') as f:
        for item in warc.WARCFile(arquivo):
            if item['Content-Type'] == 'text/plain':
                texto_corpus += item.payload.read()

# a partir de agora podemos processar as frases e criar um modelo

# podemos processar o texto com alguma ferramenta ou com código próprio
texto_pre_processado = pre_processar_texto( texto_corpus )

# criamos um modelo com o word2vec
model = Word2Vec (texto_pre_processado, ... )
```

Código 4.8 – Iteração principal para obter conteúdos do ClueWeb.

4.2.3 Processando Textos com o NLTK

4.2.3.1 Remoção de stopwords

O primeiro passo no pré-processamento foi a remoção das *stopwords*, que são palavras muito comuns e geralmente "sem importância" para a análise. A biblioteca *NLTK* possui uma lista integrada de *stopwords* para várias línguas, incluindo o português brasileiro, o que facilitou essa tarefa (*utilizamos o método stopwords.words('portuguese') do NLTK para carregar essa lista.*).

A remoção dessas palavras é crucial para reduzir o ruído nos dados e melhorar a eficiência dos algoritmos de aprendizado de máquina dentro do universo de Processamento de Linguagem Natural.

Existem alguns estudos que apontam que dependendo do contexto e do algoritmo utilizado, manter as *stopwords* não afeta a semântica e pode até ser benéfico para manter uma certa relação de distância mínima entre certas palavras. Além disso, algoritimos mais recentes já possuem ferramentas que atribuem valores redundantes as *stopwords* mais conhecidas, o que acaba eliminando a necessidade desta etapa.

No caso desse estudo, a remoção das *stopwords* não foi apenas focando no benefício de gerar um conjunto de dados mais limpo e reduzir o seu ruído semântico, mas como tínhamos um hardware limitado, cada palavra que pudesse ser eliminada ajudaria em um melhor desempenho no processamento da matriz final.

4.2.3.2 Tokenização

A *tokenização* que o processo de dividir o texto em unidades menores, geralmente palavras, foi realizada através do *NLTK*. Esse processo é fundamental para converter o texto bruto em um formato estruturado que possa ser facilmente manipulado e analisado pelos algoritmos.

Foi utilizado o método o nltk.tokenize.word_ tokenize() para dividir o texto em palavras.

4.2.3.3 Lematização e stemming

Para reduzir as palavras às suas formas básicas, utilizamos técnicas de *lematização* e *stemming*. A lematização é o processo de transformar uma palavra em sua forma canônica ou base, enquanto o *stemming* corta os sufixos das palavras para encontrar suas raízes. Ambas as técnicas ajudam a tratar variações morfológicas e reduzir a dimensionalidade dos dados. A *NLTK* oferece ferramentas eficientes para realizar essas tarefas, adaptadas para a língua portuguesa.

Foram utilizados os métodos nltk.stem.RSLPStemmer() e nltk.stem.WordNetLemmatizer() para fazer a lematização e stemming dos textos.

4.2.3.4 Remoção de pontuação e caracteres especiais

Outro passo essencial foi a remoção de pontuações e caracteres especiais dos textos.

Esta é uma parte importante do pré-processamento de texto, pois pontos, vírgulas, pontos de exclamação e interrogação não carregam significado semântico em si e podem interferir no processamento do texto.

Essa limpeza ajuda a focar nos elementos essenciais do texto, evitando que os algoritmos se distraiam com símbolos que não contribuem para o entendimento semântico.

Por exemplo, em vez de processar a frase “Olá, mundo!", o algoritmo lidará apenas com “Olá mundo", simplificando o custo de análise.

Isso foi feito utilizando funções da *NLTK*, que identificam e removem esses elementos, garantindo que o texto fique limpo e padronizado.

Também criamos nossas próprias regras através das funções de manipulação de string do Python.

4.2.3.5 Normalização

A normalização ajuda a garantir que variações na capitalização ou formas de palavras não afetem a análise, enquanto a redução do espaço de termos melhora a eficiência dos modelos de aprendizado de máquina ao focar em termos mais significativos.

Quando necessário utilizamos métodos de manipulação de *string* através das funções do Python como *lower()* e *upper()*

4.2.3.6 Desenvolvimento de ferramentas com o NLTK

Neste capítulo, comentaremos o desenvolvimento de um microprojeto utilizando *Flask*, *Python*, *HTML* e a biblioteca *NLTK*. Este projeto foi concebido não apenas como um exercício acadêmico, mas como uma oportunidade para explorar e aprender sobre diversas tecnologias e técnicas.

Este microprojeto teve como objetivo primário aprender a capturar textos da web através de técnicas de *web scraping*. Isso nos permitiu entender melhor como sistemas como o "Never Ending Language Learning (NELL)" funcionam ao coletar de forma automatizada de dados textuais da internet.

Durante o desenvolvimento, implementamos um sistema de coleta de notícias atuais através de técnicas de *web scraping* com *BeautifulSoup*. Este sistema nos permitiu extrair os conteúdos de notícias de uma fonte confiável e processá-los posteriormente, utilizando as ferramentas de análise de texto.

Neste trecho do trabalho, exploramos não apenas o *NLTK*, mas também a biblioteca *SpaCy* para processamento de linguagem natural. Ambas as ferramentas são robustas em análise de texto, oferecendo funcionalidades avançadas como *tokenização*, análise gramatical, reconhecimento de entidades nomeadas, entre outras. Enquanto o *NLTK* é conhecido por sua flexibilidade e extensa gama de módulos para diferentes tarefas de processamento de linguagem natural, o *SpaCy* se destaca pela eficiência e desempenho, o que seria ideal para testar em um pequeno projeto. A combinação do estudo das ferramentas *NLTK* e *SpaCy* nos permitiu explorar e comparar suas abordagens, ampliando assim nossa compreensão sobre as melhores práticas e técnicas.

Um dos principais resultados do nosso projeto foi a capacidade de realizar uma análise gramatical detalhada das notícias coletadas. Utilizamos *NLTK*/*SpaCy* para categorizar palavras como verbos, substantivos, adjetivos, entre outros, e exibimos essas categorias de forma visualmente informativa na interface em HTML.

Um dos principais resultados do nosso microprojeto foi a capacidade de realizar uma análise gramatical detalhada das notícias coletadas. Utilizamos o *NLTK* para categorizar

palavras como verbos, substantivos, adjetivos, entre outros, e exibimos essas categorias de forma visualmente informativa na interface em HTML do software.

O micro projeto atualmente está hospedado em: <https://noticias-com-nltk.onrender.com/>

No trecho de código abaixo podemos demonstrar como capturar textos de páginas da web:

```
# bibliotecas
import requests
from bs4 import BeautifulSoup

# Função para capturar notícias de um site
def obter_noticias():
    # Exemplo de URL de noticias
    url = 'https://www.JORNALEXEMPLO.com/'
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    # Extrair os títulos das notícias, baseado na estrutura do site
    titulos = []
    for titulo in soup.find_all('h3', class_='hui-premium__title'):
        titulos.append(titulo.text.strip())

    # Retorna os primeiros 10 títulos de notícias
    return titulos[:10]
```

Código 4.9 – Capturar textos de páginas da web.

Após obter trechos da web podemos classificar gramaticalmente as frases, assim como podemos fazemos com qualquer texto de um *corpus*.

```
# Carregar modelo em português do SpaCy
import spacy
nlp = spacy.load('pt_core_news_sm')

#Função que categoriza gramaticalmente
def categorizar_palavras(frase):

    #carrega a frase com o Spacy (poderíamos usar o NLTK também)
    doc = nlp(frase)
    frase_categorizada = []

    for token in doc:
        #analisa cada token da frase e define sua categoria gramatical
        categoria = token.pos_
        categoria_ptbr = traduz_categorias.get(categoria, 'Desconhecid')

        #insere após cada palavra a sua categoria entre parênteses
        palavra_categorizada = f'{token.text} ({categoria_ptbr})'

        #agrega a palavra categorizada na frase de saída
        frase_categorizada.append(palavra_categorizada)

    return ' '.join(frase_categorizada)

# Exemplo de uso
frase_exemplo = "Eu gosto de programar em Python."
frase_categorizada = categorizar_palavras(frase_exemplo)
print(frase_categorizada)
...
SAIDA:
Eu (pronome) gosto (verbo) de (preposicao) programar (verbo) em (
   preposicao) Python (desconhecido).
```

Código 4.10 – Classificar palavras gramaticalmente com o Spacy.

Podemos ver um exemplo do resultado do microprojeto na Figura 14. Na imagem é possível observar as seguintes características:

- Cada categoria gramatical deve ser exibida em uma cor diferente.
- Algumas palavras possuem mais de um sentido e podem aparecer em qualquer uma das categorias.
- Algumas palavras que não conseguiram ser identificadas aparecem em preto.
- A precisão da assertividade pode variar dependendo da qualidade do treinamento.

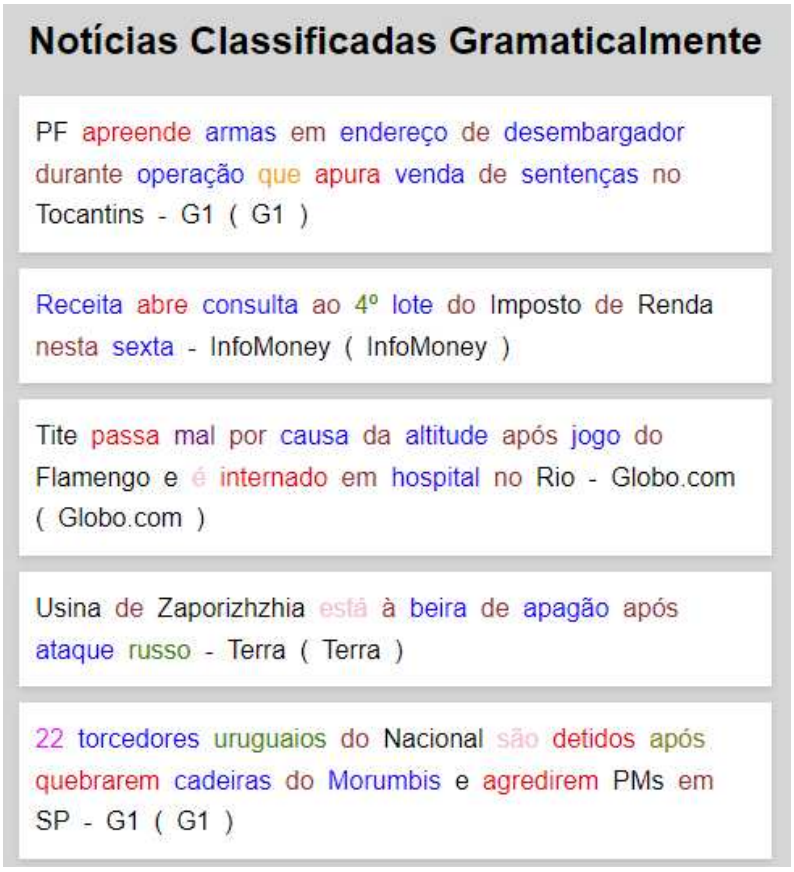

Figura 14 – Categorizando palavras de notícias em tempo real.

4.2.4 Integração de dados com o Word2Vec

Após o extenso pré-processamento e a preparação dos dados, avançamos para a fase de aplicação prática e análise dos resultados obtidos. Utilizamos as técnicas e ferramentas de Processamento de Linguagem Natural (*PLN*) para realizar diversas tarefas, testando e validando nossos modelos para garantir a eficácia e precisão das análises.

Após preparar os dados com o *NLTK*, partimos para a fase de integração com o Word2Vec.

Nossa jornada começou com a conversão dos textos pré-processados em vetores de palavras usando o Word2Vec. Este passo foi crucial para transformar a linguagem natural em uma representação numérica que pudesse ser usada pelos algoritmos de aprendizado de máquina.

No início, enfrentamos várias dificuldades. A primeira foi a seleção dos parâmetros adequados para o Word2Vec. Como ainda não tínhamos experiência suficiente, decidimos usar os parâmetros padrão da versão corrente do Word2Vec (meados de 2018). Esses parâmetros incluem:

- Tamanho dos Vetores: Definido como 100 dimensões. Este parâmetro determina a dimensionalidade dos vetores de palavras gerados. Um vetor maior pode capturar mais nuances de significado, mas também aumenta o tempo de processamento e o uso de memória.

- Janela de Contexto: Configurada para 5 palavras. Este parâmetro especifica o número de palavras à esquerda e à direita de uma palavra-alvo que o modelo deve considerar como contexto. Uma janela maior pode capturar relações de palavras em frases mais longas, mas também pode introduzir mais ruído. Número de Iterações: Estabelecido em 5. Este parâmetro indica quantas vezes o modelo percorre o *corpus* de treinamento completo. Mais iterações podem melhorar a precisão do modelo, mas também aumentam o tempo de treinamento.

- Algoritmo de Treinamento: Utilizado Skip-gram. O Skip-gram tenta prever palavras de contexto a partir da palavra-alvo, o que é eficaz para capturar relações de palavras em dados escassos, mas pode ser mais lento que outras abordagens, como o CBOW (Continuous Bag of Words).

- Min Count: Definido como 5. Este parâmetro faz o modelo ignorar palavras que aparecem com menos frequência do que cinco vezes no *corpus*. Isso ajuda a reduzir o ruído e a focar em palavras mais significativas, mas pode excluir termos raros que poderiam ser importantes em contextos específicos.

A escolha desses parâmetros nos ajudou a iniciar o processo sem complicações adicionais, mas também trouxe algumas limitações. Por exemplo, o desempenho do treinamento

foi ruim em máquinas lentas, o que nos obrigou a otimizar nossos recursos e ajustar os parâmetros conforme ganhávamos mais experiência.

Para deixar mais claro, explicaremos de forma mais técnica o uso do Word2vec em um código Python, por exemplo.

Para criar uma chamada no Word2Vec, precisamos primeiro escolher uma biblioteca a ser importada que ira treinar e processar o nosso modelo. No nosso caso, escolhemos a *Gensim*. Uma das principais vantagens do *Gensim* é sua eficiência em treinamento de modelos em *CPU*s. A biblioteca utiliza operações multitarefas, utilizando os múltiplos núcleos disponíveis da *CPU* para acelerar o processo de treinamento, realizando o processamento paralelo dos mesmos, o que é benéfico para usuários que não possuem acesso a *GPU*s poderosas.

Para demonstrar a criação de um modelo Word2Vec com *Gensim*, aqui está um exemplo básico de como configurar e chamar a função Word2Vec:

```
from gensim.models import Word2Vec
tokenized_sentences = [["exemplo", "de", "frase"], ["outra", "frase", "
    tokenizada"]]

model = Word2Vec(
    sentences=tokenized_sentences,
    vector_size=100,     # Tamanho dos vetores
    window=5,            # Tamanho da janela de contexto
    min_count=5,         # Mínimo de ocorrencias para uma palavra ser
    considerada
    workers=4            # Número de threads
)

# Treinamento do modelo
model.train(tokenized_sentences, total_examples=len(tokenized_sentences)
    , epochs=10)
$
```

Código 4.11 – Modelo criado no Word2Vec usando o Gensim.

A chamada do método *model.train()* no código *Gensim* para Word2Vec é responsável por treinar o modelo de vetores de palavras nas sentenças fornecidas. Nela os três parâmetros são os seguintes:

- tokenized_sentences: Esta é a lista de sentenças *tokenizadas* que serão usadas para treinar o modelo.
- total_examples: Este parâmetro define o número total de exemplos de treinamento, que neste caso é o comprimento da lista de sentenças *tokenizadas*.

- epochs: Este parâmetro define quantas vezes o modelo deve iterar sobre o *corpus* de treinamento. Neste exemplo, o modelo será treinado por 10 épocas.

Ao se chamar o método *model.train()* o Word2Vec segue os seguintes passos:

- Itera sobre as sentenças *tokenizadas* (separadas em *tokens*).
- Para cada época, percorre todas as sentenças e ajusta os vetores de palavras conforme o algoritmo Skip-gram.
- Utiliza o número de *threads* especificado para paralelizar o treinamento e acelerar o processo.
- Ajuste dos vetores de palavras no modelo para refletirem melhor as relações semânticas e contextuais presentes nas sentenças de treinamento.

A função *train()* em si não possui retorno. O principal efeito dessa chamada é o ajuste interno dos vetores de palavras no modelo gerado no Word2Vec.

Após o fim do treinamento, tem se a matriz que contém os vetores de palavras aprendidos no treinamento, aonde palavras que ocorrem frequentemente próximas umas das outras no *corpus* terão vetores mais semelhantes.

4.2.5 Representação Em Formato De Matriz

No Word2Vec, uma matriz é representada pelos vetores de palavras que o modelo aprende durante o treinamento.

Esses vetores de palavras são armazenados internamente no modelo treinado e podem ser acessados e utilizados para diversas tarefas.

O vocabulário é uma coleção de todas as palavras únicas que o modelo Word2Vec encontrou durante o treinamento.

Cada palavra no vocabulário é mapeada para um índice, que é um número inteiro único.

A matriz em si é uma coleção de vetores, onde cada vetor corresponde a uma palavra no vocabulário do modelo.

Cada palavra no vocabulário é associada a um vetor de uma determinada dimensão (definida pelo parâmetro *vector_size*). Também podemos acessar diretamente através do atributo *model.wv.key_to_index*, que é um dicionário mapeando palavras a seus índices.

Os vetores de palavras são armazenados em uma matriz de pesos, onde cada linha da matriz corresponde ao vetor de uma palavra. Essa matriz pode ser acessada através do atributo *model.wv.vectors*.

Por exemplo, se você tem um vocabulário de 10.000 palavras e escolheu *vector_size* igual a 100, a matriz de pesos terá 10.000 linhas e 100 colunas.

Podemos acessar as dimensões da matriz criada pelo modelo com a chamada ao atributo *shape*:

```
dimensoes = model.wv.vectors.shape
```

Código 4.12 – Acessa as dimensões da matriz

4.2.6 Primeiros resultados

Para verificar o resultado do treinamento criado pelo Word2Vec, você pode os usar métodos do próprio modelo criado pelo Word2Vec para analisar os vetores de palavras ou de palavras similares.

Podemos obter o vetor de uma palavra específica buscando-a:

```
word_vector = model.wv['exemplo']
```

Código 4.13 – Obtém o vetor de uma palavra específica.

Podemos também obter uma lista de palavras similares a uma determinada palavra especificada:

```
similar_words = model.wv.most_similar('exemplo')
```

Código 4.14 – Obtém lista de palavras similares.

Também é possível prever a(s) próxima(s) provável(eis) palavra(as) que aparecerá(ão) após iniciar uma sentença:

```
predicted_words = model.wv.predict_output_word(["cidade", "maravilhosa"
    ])
```

Código 4.15 – Obtém possíveis próximas palavras.

E é possível definir o número máximo de resultados de saída através do parâmetro *topn*, como nos exemplos abaixo:

```
similar_words = model.wv.most_similar("exemplo", topn=10)
```

Código 4.16 – Define o número máximo de resultados de saída.

```
predicted_words = model.wv.predict_output_word(["cidade", "maravilhosa"], topn=10)
```

Código 4.17 – Define o número máximo de resultados de saída.

O próximo passo foi realizar um teste com bases mais robustas e pré-classificadas, utilizando-se o *NLTK* e o Word2Vec, que foi escolhido como principal ferramenta de testes neste trabalho. Nessa etapa, a base utilizada foi a *Mac-Morpho*.

É importante ressaltar que no início do projeto todos os textos-base utilizados estavam na língua inglesa. Mesmo assim eles foram utilizados em um primeiro momento, pois a ferramenta foi originalmente treinada para ser utilizada com textos em inglês e o foco principal nessa etapa era adquirir um maior conhecimento do funcionamento das ferramentas.

Nas primeiras rodadas de treinamento não conseguimos bons resultados justamente por não termos um conhecimento mais profundo sobre o funcionamento da ferramenta. Assim os primeiros experimentos foram utilizados como um meio de teste para aprendizado da ferramenta, sem que houvesse qualquer tipo de utilização póstuma com o que foi feito.

Ao fim de uma semana de estudos das ferramentas, ficamos mais íntimos do seu funcionamento e realizamos os mesmos testes da semana anterior, porém agora utilizando a base *Mac-Morpho* (com textos em português) e obtivemos resultados bastante positivos. Como o *Mac-Morpho* é uma base morfologicamente pré-classificada com várias palavras diferenciadas em seus respectivos contextos, isso ajudou o treinamento a ter um melhor resultado.

Após algum tempo de pesquisas, encontramos a base da Wikipedia Dumps (FOUNDATION, 2024), citada no capítulo anterior. Como ela é um grande *dump* de todo texto da Wikipedia, precisava ser tratada antes de ser processada.

Com o texto tratado através do *toolkit* do *NLTK*, o primeiro desafio foi utilizar o Word2Vec para criar um modelo/matriz com estes dados. O processo foi bem mais demorado do que estávamos acostumados até o momento, mas os resultados foram proporcionalmente melhores. Após esta etapa foi a fez de adaptar a estruturas para implementar a leitura dos dados do *ClueWeb*.

4.2.6.1 Resultados

Realizamos testes e comparativos com algumas palavras para avaliar a capacidade e a profundidade dos modelos que desenvolvemos. Esses testes são fundamentais para entender como nossos modelos de Word2Vec se comportam em diferentes contextos e qual a qualidade das representações vetoriais que eles produzem.

Para quantificar a similaridade entre duas palavras, utilizamos uma técnica chamada similaridade do cosseno. Essa métrica nos permite calcular o quão semelhantes duas palavras são com base em suas representações vetoriais.

Cada palavra em nosso modelo Word2Vec é representada por um vetor em um espaço de alta dimensionalidade. Esses vetores são gerados durante o treinamento do modelo e capturam relações semânticas entre as palavras.

Nos nossos resultados, comparamos palavras em dois modelos diferentes: um treinado com dados do *ClueWeb* e outro com dados da *Wikipedia*. Essa comparação nos permite observar como o contexto das palavras pode mudar dependendo do conjunto de dados utilizado para treinar o modelo. Por exemplo, palavras que são altamente similares no modelo do *ClueWeb* podem não ser tão similares no modelo da Wikipedia, e vice-versa.

Iremos comentar alguns exemplos.

Neste primeiro teste, utilizamos a palavra “Recife". Nota-se que na lista de palavras similares, foram retornadas cidades que estão próximas à cidade de Recife e que muitas vezes são referenciadas por ela ou que aparecem em textos em que são citadas cidades da

região. Na listagem de há uma grande similaridade na relação com “Caruaru", muito por conta da cidade ser muito próxima a Recife e a mesma ser rota de passagem e referência para a ida a Caruaru. A baixa similaridade associada a “Pernambuco” se dá, provavelmente, devido à cidade de Recife ser capital do Estado e sendo assim referência do mesmo, logo o nome do Estado possivelmente não viria após o nome da cidade. O resultado pode ser visto na Tabela 5.

Resultados	Similaridade
Pernambuco	0.89907087
Olinda	0.81464592
Caruaru	0.7846946
Recife	0.7771268
Salvador	0.59987654
Jaboatão	0.45765046
Garanhuns	0.4904804
Íbis	0.4819362
Tramways	0.20910993

Tabela 5 – Resultados de similaridade do Word2Vec na base da Wikipedia para a palavra 'Recife'.

O próximo teste foi com a palavra “homem". Na lista de similaridade, podemos ver que a mesma é composta por outros nomes e palavras que são comumente associados a palavra homem, ou ao gênero masculino, em textos e que possuem uma alta porcentagem de relação. Também há uma associação existente a termos associados a nomes dos heróis dos quadrinhos (homem-aranha, homem de ferro, homem-elástico, ...). O mesmo se dá com a alta associação observada com a palavra “aranha", que obteve resultado acima de 90% . O resultado pode ser visto na Tabela 6.

Resultados	Similaridade
senhor	0.96749187
aranha	0.90749187
rapaz	0.86749187
ferro	0.664812
elástico	0.6343275
homem	0.6165179
abominável	0.5126230
honesto	0.412174
neandertal	0.4962586

Tabela 6 – Resultados de similaridade do Word2Vec na base do Clueweb para a palavra 'homem'.

O próximo teste foi realizado com a palavra “arroz” utilizando o modelo Clueweb. Na lista de palavras similares, podemos observar uma forte associação com ingredientes e

alimentos que frequentemente são utilizados em conjunto com arroz em diversas receitas e contextos culinários.

Notamos uma alta similaridade com “manteiga", “frango", “azeite", “tomate", “farinha", “fatias” e “colheres". Essas palavras representam ingredientes comuns, geralmente combinados com arroz em pratos variados, refletindo a capacidade do modelo de identificar relações contextuais frequentes nos textos analisados. Podemos observar a similaridade na Tabela 7.

Resultados	Similaridade
manteiga	0.94026959
frango	0.93998617
azeite	0.93515569
tomate	0.93319941
farinha	0.93296683
fatias	0.93002540
colheres	0.92852443

Tabela 7 – Palavras similares a palavra ’arroz’ segundo o modelo ’Clueweb’.

Nos testes realizados com o modelo Clueweb, observamos que a palavra “Lula” apresentou uma lista de palavras similares que faz sentido dentro do contexto político e social. Entre as palavras similares, encontramos palavras associadas a temas políticos e frequentemente presentes em textos que mencionam “Lula". Essas associações demonstram a capacidade do modelo de capturar contextos e relações semânticas coerentes. Podemos observar o resultado obtido observando a Tabela 8

Resultados	Similaridade
CPMI	0.78719836
Dilma	0.78271896
CPI	0.77512282
PRP	0.76629275
delação	0.75264090
PSC	0.71915197

Tabela 8 – Palavras similares a palavra ’Lula’ segundo o modelo ’Wikipedia’.

No modelo treinado com dados da Wikipedia, a palavra “Bolsonaro” gerou uma lista de palavras similares que refletem o contexto político e social em que ela aparece frequentemente. Estas palavras são comumente encontradas em discussões sobre o cenário político brasileiro e têm uma alta similaridade com “Bolsonaro". Podemos observar o resultado obtido observando a Tabela 9

Resultados	Similaridade
Jair	0.81893593
Dilma	0.67862183
Carlos	0.67554641
inquerito	0.67371958
denuncia	0.67226464
impeachment	0.67164141
STF	0.66814542

Tabela 9 – Palavras similares a palavra 'Bolsonaro' segundo o modelo 'Wikipedia'.

O próximo teste foi realizado com a palavra "carro" utilizando o modelo Wikipedia. Na lista de palavras similares, observamos uma associação interessante com termos que estão ligados a veículos, atividades de velocidade e ação, além de alguns contextos menos esperados.

Notamos uma alta similaridade com "tiro", "correr", "jipe", "piloto", "acorda", "corrida" e "catamarã". Essas palavras refletem tanto o contexto de veículos e velocidade quanto algumas associações mais inusitadas, possivelmente derivadas de usos figurativos ou menos comuns da palavra "carro" em diferentes textos.

É interessante observar que algumas palavras surgem de maneira aparentemente aleatória. Isso pode ser atribuído à qualidade e à diversidade do treinamento do modelo. O modelo Word2Vec aprende associações com base nos contextos em que as palavras aparecem no texto de treinamento. Portanto, se uma palavra aparece frequentemente em contextos específicos ou variados, o modelo aprenderá essas associações, mesmo que não sejam as mais óbvias.

Por exemplo, a presença de palavras como "tiro" e "acorda" pode refletir usos específicos ou menos comuns da palavra "carro" em narrativas ou expressões idiomáticas capturadas no *corpus* de treinamento da Wikipedia. Podemos observar a variação dos resultados observando a Tabela 10.

Resultados	Similaridade
tiro	0.72401720
correr	0.70906347
jipe	0.70357811
piloto	0.70354557
acorda	0.69433481
corrida	0.69160128
catamarã	0.68132305

Tabela 10 – Palavras similares a palavra 'carro' segundo o modelo 'Wikipedia'.

Após os testes realizados, foi possível observar que, embora o Word2Vec tenha sido inicialmente desenvolvido para a língua inglesa e ainda esteja em processo de adaptação

para outras línguas, já é possível obter vetores de similaridade interessantes quando a ferramenta é utilizada na língua portuguesa. Os resultados demonstram que o modelo consegue capturar associações relevantes e contextuais entre palavras, revelando a capacidade do Word2Vec em lidar com o nosso idioma.

No entanto, notamos que algumas palavras inesperadas podem surgir como similares devido à qualidade do treinamento ou ao *corpus* utilizado. Isso pode ocorrer por conta da diversidade e do contexto dos textos que compõem o *corpus*, que influenciam diretamente nas associações aprendidas pelo modelo. Além disso, a quantidade e a variação de dados de entrada podem impactar a precisão e a relevância das associações geradas.

Esses fatores ressaltam a importância de utilizar um *corpus* bem estruturado e representativo para treinamento, garantindo que o modelo capture associações mais precisas e contextuais.

4.2.6.2 Mais resultados do modelo: Wikipedia

Nas Tabelas 11, 12, 13 e 14, podemos ver os resultados obtidos pelo modelo gerado através do *corpus* da Wikipedia para as palavras geografia, escola, arroz e computador, respectivamente.

Palavra similares a (geografia)	Similaridade
ibge	0.75401795
estatistica	0.70915699
domicilios	0.69506538
distritos	0.68976617
habitantes	0.66721928
regioes	0.66559708
populacional	0.66053516

Tabela 11 – Palavras similares a palavra 'geografia' segundo o modelo 'Wikipedia'.

Palavra similares a (escola)	Similaridade
estudante	0.73377419
colegio	0.71455044
faculdade	0.68773448
aulas	0.68326074
escolas	0.68198514
escolar	0.66434401
curso	0.65989906

Tabela 12 – Palavras similares a palavra 'escola' segundo o modelo 'Wikipedia'.

Palavra similares a (arroz)	Similaridade
milho	0.88368642
açúcar	0.87322110
mandioca	0.86615664
secas	0.85888046
encostas	0.85642517
grãos	0.85455650
formando	0.84657019

Tabela 13 – Palavras similares a palavra 'arroz' segundo o modelo 'Wikipedia'.

Palavra similares a (computador)	Similaridade
software	0.85018277
conectividade	0.84964043
computadores	0.83523500
microcontrolador	0.83349830
megabytes	0.82931387
videogames	0.82722884
utiliza	0.81674290

Tabela 14 – Palavras similares a palavra 'computador' segundo o modelo 'Wikipedia'.

4.2.6.3 Mais resultados do modelo: Clueweb

Nas Tabelas 15, 16, 17 e 18, podemos ver os resultados obtidos pelo modelo gerado através do *corpus* da Clueweb para as palavras geografia, escola, arroz e computador, respectivamente.

Palavra similares a (geografia)	Similaridade
biologia	0.88163608
científica	0.86489701
CEBRID	0.85558087
estudos	0.85327876
linguística	0.85280406
promove	0.84759581
educação	0.84544599

Tabela 15 – Palavras similares a palavra 'geografia' segundo o modelo 'Clueweb'.

Palavra similares a (escola)	Similaridade
vila	0.78504574
academia	0.76600093
colégio	0.76449960
presidência	0.76297969
palácio	0.75701278
escoteiro	0.75618106
paulista	0.75571007

Tabela 16 – Palavras similares a palavra 'escola' segundo o modelo 'Clueweb'.

Palavra similares a (arroz)	Similaridade
manteiga	0.94217199
frango	0.93900764
azeite	0.93486339
tomate	0.93362331
farinha	0.93307203
fatias	0.93244678
colheres	0.92873788

Tabela 17 – Palavras similares a palavra 'arroz' segundo o modelo 'Clueweb'.

Palavra similares a (computador)	Similaridade
aparelho	0.86634958
wirelless	0.82823908
visual	0.81746703
externo	0.81315386
usando	0.81221426
instalar	0.81046724
arma	0.79757017

Tabela 18 – Palavras similares a palavra 'computador' segundo o modelo 'Clueweb'.

4.2.6.4 Comparação de resultados

Os testes abaixo representam a diferença de análise e resultados dos dois *corpus* recém analisados, Wikipedia e Clueweb.

A diferença entre eles é a quantidade de texto analisado, devido ao tamanho dos *corpus*. Ambos os *corpus* foram processados durante 20 horas consecutivas. Após este período, todo o conteúdo da Wikipedia foi analisado, enquanto apenas 35 porcento do Clueweb foi analisado.

Em ambos os casos, as matrizes geradas para comparação (tanto a da Wikipedia como a do Clueweb) tiveram os mesmos parâmetros de entrada:

- **Parâmetros**:
  - `vector_size`: 300
  - `window`: 30
  - `min_count`: 10

Dadas estas informações, podemos observar alguns resultados comparativos:

Na Tabela 19 representamos um teste de redundância para verificar a funcionalidade do algoritmo entre palavras idênticas nas quais, como esperado, a similaridade deve ser sempre 1.

Palavras	Modelo	Similaridade
teste - teste	CLUEWEB	1.000000
teste - teste	WIKIPEDIA	1.000000

Tabela 19 – Similaridade de cosseno entre as palavras teste & teste.

Nas Tabelas 20, 21, 22, 23 e 24, observamos uma similaridade alta entre as palavras propostas.

Palavras	Modelo	Similaridade
menino - garoto	CLUEWEB	0.887679
menino - garoto	WIKIPEDIA	0.763955

Tabela 20 – Similaridade de cosseno entre as palavras menino & garoto.

Palavras	Modelo	Similaridade
homem - mulher	CLUEWEB	0.840162
homem - mulher	WIKIPEDIA	0.709823

Tabela 21 – Similaridade de cosseno entre as palavras homem & mulher.

Palavras	Modelo	Similaridade
dois - quatro	CLUEWEB	0.801857
dois - quatro	WIKIPEDIA	0.728029

Tabela 22 – Similaridade de cosseno entre as palavras dois & quatro.

Palavras	Modelo	Similaridade
cachorro - gato	CLUEWEB	0.866906
cachorro - gato	WIKIPEDIA	0.824410

Tabela 23 – Similaridade de cosseno entre as palavras cachorro & gato.

Palavras	Modelo	Similaridade
pai - filho	CLUEWEB	0.842656
pai - filho	WIKIPEDIA	0.789001

Tabela 24 – Similaridade de cosseno entre as palavras pai & filho.

Nas Tabelas 25, 26, 27 e 28, observamos uma variação muito grande entre os dois modelos.

Palavras	Modelo	Similaridade
paz - guerra	CLUEWEB	0.562879
paz - guerra	WIKIPEDIA	0.396921

Tabela 25 – Similaridade de cosseno entre as palavras paz & guerra.

Palavras	Modelo	Similaridade
esporte - futebol	CLUEWEB	0.746216
esporte - futebol	WIKIPEDIA	0.377361

Tabela 26 – Similaridade de cosseno entre as palavras esporte & futebol.

Palavras	Modelo	Similaridade
verde - amarelo	CLUEWEB	0.726658
verde - amarelo	WIKIPEDIA	0.435125

Tabela 27 – Similaridade de cosseno entre as palavras verde & amarelo.

Palavras	Modelo	Similaridade
homem - folha	CLUEWEB	0.480110
homem - folha	WIKIPEDIA	0.136040

Tabela 28 – Similaridade de cosseno entre as palavras homem & folha.

Nas Tabelas 29, 30, 31 e 32, observamos uma variação discreta, mas ainda, sim, com valores próximos.

Palavras	Modelo	Similaridade
carro - moto	CLUEWEB	0.572367
carro - moto	WIKIPEDIA	0.475547

Tabela 29 – Similaridade de cosseno entre as palavras carro & moto.

Palavras	Modelo	Similaridade
chegar - sair	CLUEWEB	0.590257
chegar - sair	WIKIPEDIA	0.628278

Tabela 30 – Similaridade de cosseno entre as palavras chegar & sair.

Palavras	Modelo	Similaridade
amigo - amiga	CLUEWEB	0.704094
amigo - amiga	WIKIPEDIA	0.848854

Tabela 31 – Similaridade de cosseno entre as palavras amigo & amiga.

Palavras	Modelo	Similaridade
palavra - verbo	CLUEWEB	0.821214
palavra - verbo	WIKIPEDIA	0.639660

Tabela 32 – Similaridade de cosseno entre as palavras palavra & verbo.

Após os testes, fica evidente o impacto que uma base grande, rica e complexa pode exercer quando se trata de treinar modelos para captação de similaridades e palavras associadas, utilizando ferramentas como o Word2Vec para o treinamento de modelos de *PLN*.

Para observar casos e realizar mais testes, há um *script* disponível no *git* do projeto que compara as similaridades entre duas palavras dadas como entrada. O link do *git* pode ser encontrado no apêndice deste documento.

4.2.6.5 Acurácia

Para avaliar a qualidade dos resultados obtidos com o modelo Word2Vec, criamos alguns *scripts* específicos que comparam as predições dos modelos com algumas respostas esperadas baseadas em analogias de palavras.

O objetivo é verificar se o modelo consegue preencher corretamente as lacunas em analogias, considerando o contexto da língua portuguesa e aspectos culturais.

Porém, para definir o valor da acurácia pensamos em alguns testes distintos.

4.2.6.5.1 Método 1: acurácia binária entre duplas

Neste método, as analogias são estipuladas com a seguinte linha de pensamento: são dadas duas palavras, sendo esperado que o modelo acerte a segunda palavra buscando palavras similares a primeira.

Utilizamos uma lista com 600 analogias diferentes e verificamos as predições dos modelos para cada uma delas.

Para cada analogia, o modelo gera as 10 palavras similares a palavra teste. Se a palavra esperada estiver dentre as 10, é considerado um acerto.

Tentamos montar uma lista de analogias baseadas em dois tipos de categorias distintas. Alguns exemplos das analogias propostas:

- Analogias de gênero
  - menino → menina
  - príncipe → princesa
  - filho → filha
  - sobrinho → sobrinha
  - neto → neta
- Analogias em geral
  - cachorro → cão
  - aluno → estudante
  - rio → correnteza
  - sol → dia
  - lua → noite

O script verifica se a palavra correta está entre as 10 predições geradas pelo modelo. Se a palavra esperada estiver entre as predições, é considerado um acerto. Caso contrário, é considerado uma falha. Após os resultados de todas as analogias ser gerado, a porcentagem de acertos é definida como a acurácia.

Podemos observar a fórmula 4.1, utilizada para calcular esta acurácia.

$$\text{Acurácia} = \frac{\text{quantidade de acertos}}{\text{quantidade de analogias}} \tag{4.1}$$

Nesta forma de prever a acurácia, a porcentagem costuma ser muito baixa, algumas vezes devido às variações da língua, onde palavras similares são apresentadas no lugar das esperadas, outras vezes a culpa é do treinamento inconclusivo que gera palavras inesperadas e não se aproxima do resultado real.

Com este método tivemos os resultados representados na Tabela 33.

Modelo	**Acurácia**
ClueWeb	5.47%
Wikipedia	12.38%

Tabela 33 – Resultado do método: Acurácia Binária entre duplas

4.2.6.5.2 Método 2: acurácia binária entre quartetos

Neste método, as analogias são estipuladas com a seguinte linha de pensamento: são dadas três palavras, sendo esperado que o modelo acerte a quarta palavra, ou seja, dada a relação entre as palavras 1 e 2, qual seria a palavra 4 que cria a mesma relação com a palavra 3.

Utilizamos uma lista com 350 analogias diferentes e verificamos as predições dos modelos para cada uma delas.

Para cada analogia, o modelo gera as 10 palavras válidas para a relação proposta. Se a palavra esperada estiver dentre as 10, é considerado um acerto.

Tentamos montar uma lista de analogias baseadas em três tipos de categorias distintas (gênero, gramática e geral) para cobrir uma vasta gama de possibilidades.

Veja alguns exemplos:

- Analogias de gênero
  - homem, mulher, menino → menina
  - rei, rainha, príncipe → princesa
  - pai, mãe, filho → filha
  - tio, tia, sobrinho → sobrinha
  - avô, avó, neto → neta
  - marido, esposa, noivo → noiva
  - garoto, garota, rapaz → moça
  - ator, atriz, cantor → cantora

- Analogias gramaticais
  - construir, construiu, destruir → destruiu
  - crescer, cresceu, diminuir → diminuiu
  - entrar, entrou, sair → saiu
  - perder, perdeu, ganhar → ganhou
  - subir, subiu, descer → desceu
  - abrir, abriu, fechar → fechou
- Analogias em geral
  - gato, felino, cachorro → cão
  - escola, professor, aluno → estudante
  - amigo, amiga, homem → mulher
  - lago, mar, rio → correnteza
  - vermelho, azul, rosa → verde
  - luz, noite, sol → dia
  - rei, rainha, homem → mulher
  - pai, mãe, homem → mulher

O *script* verifica se a palavra correta está entre as 10 predições geradas pelo modelo. Se a palavra esperada estiver entre as predições, é considerado um acerto. Caso contrário, é considerado uma falha. Após os resultados de todas as analogias ser gerado, a porcentagem de acertos é definida como a acurácia.

Podemos observar a fórmula 4.2, utilizada para calcular esta acurácia.

$$\text{Acurácia} = \frac{\text{quantidade de acertos}}{\text{quantidade de analogias}} \tag{4.2}$$

Nesta forma de prever a acurácia, a porcentagem costuma ser muito baixa, algumas vezes devido às variações da língua, onde palavras similares são apresentadas no lugar das esperadas, outras vezes a culpa é do treinamento inconclusivo que gera palavras inesperadas e não se aproxima do resultado real.

Com este método tivemos os resultados representados na Tabela 34.

Modelo	**Acurácia**
ClueWeb	2.26%
Wikipedia	4.48%

Tabela 34 – Resultado do método: Acurácia Binária entre quartetos

4.2.6.5.3 Método 3: acurácia de similaridades

Neste método, as analogias são estipuladas com a seguinte linha de pensamento: são dadas duas palavras, sendo esperado que o modelo acerte a segunda buscando palavras similares a primeira.

Utilizamos uma lista com 600 analogias diferentes e verificamos as predições dos modelos para cada uma delas.

Em cada analogia, é calculada a similaridade de cossenos entre cada resultado predito e a palavra esperada, ao final é criada uma média de similaridade para cada analogia.

Ao final das análises, todas as médias de similaridade são somadas e divididas pelo número total de analogias propostas.

Quanto mais o resultado se aproxima de 1.00, melhor ele é.

Podemos observar as fórmulas 4.3 e 4.4, utilizadas para calcular esta acurácia.

$$\text{Média de similaridade} = \text{média das similaridades das palavras preditas com a palavra esperada} \tag{4.3}$$

$$\text{Acurácia} = \frac{\text{soma de todas as médias de similaridades}}{\text{quantidade de analogias}} \tag{4.4}$$

Com este método tivemos os resultados representados na Tabela 35.

Modelo	**Acurácia**
ClueWeb	0.62280
Wikipedia	0.40501

Tabela 35 – Resultado do método: Acurácia de similaridades

5 CONCLUSÃO

Após uma análise detalhada e cuidadosa de todo o conteúdo abordado em nosso trabalho, chegamos a várias conclusões importantes que refletem não apenas a nossa pesquisa, mas também o campo do Processamento de Linguagem Natural aplicado ao português brasileiro. Durante nossa jornada, enfrentamos diversos desafios e descobrimos inúmeras oportunidades que reforçam a relevância e a complexidade do campo de *PLN*.

Inicialmente, vale destacar a escolha do Word2Vec como ferramenta principal para nossos testes. Esta decisão foi fundamentada em sua eficácia na representação vetorial de palavras e na construção de relações contextuais entre elas. O uso da biblioteca Gensim se mostrou particularmente vantajoso, especialmente por sua capacidade de operar de maneira eficiente em ambientes sem a necessidade de GPUs dedicadas. Isso é um ponto extremamente positivo por demonstrar que é possível realizar pesquisas de qualidade em *PLN* mesmo com recursos de hardware limitados, tornando a área mais acessível a instituições e pesquisadores com orçamento reduzido.

Tecnicamente, o Word2Vec se destacou por sua habilidade em capturar similaridades e relações semânticas entre palavras, o que nos permite explorar diversas aplicações, desde a análise de sentimentos até a classificação de documentos. No entanto, notamos que uma das principais limitações do Word2Vec é sua incapacidade de considerar o contexto mais amplo de uma palavra em um documento. Esta limitação nos levou a buscar maneiras de contornar essa deficiência, como a criação de um corpus robusto e representativo da língua portuguesa brasileira.

O uso dessas bases de dados foi essencial para o sucesso do nosso projeto. A base *Mac-Morpho*, por ser uma base morfologicamente pré-classificada, nos proporcionou um ponto de partida sólido para entender melhor como os algoritmos de *PLN* funcionam. Através de testes rigorosos, conseguimos resultados muito positivos, demonstrando a eficácia das ferramentas utilizadas. Enquanto a complexidade e a diversidade dos dados das bases da Wikipedia e do *ClueWeb* nos apresentaram desafios adicionais, pois tivemos que tratar grandes volumes de dados textuais, necessitando de técnicas avançadas de pré-processamento para garantir a qualidade e a relevância das informações extraídas.

A aplicação de técnicas de *PLN* ao nosso idioma apresenta desafios únicos. Diferente do inglês, o qual é amplamente estudado e possui uma vasta quantidade de recursos e ferramentas disponíveis, o português brasileiro ainda carece de bases de dados e modelos tão robustos. Nosso trabalho mostrou a importância de adaptar e, em muitos casos, criar novas ferramentas que atendam às especificidades da nossa língua. Esse esforço é fundamental para melhorar a assertividade e a precisão dos modelos de *PLN* aplicados em contextos reais.

Durante o desenvolvimento deste trabalho, enfrentamos algumas dificuldades. A principal delas foi a necessidade de um grande volume de dados de alta qualidade para o treinamento dos modelos. A coleta e o pré-processamento desses dados demandaram tempo e esforço consideráveis. Além disso, a complexidade dos algoritmos exigiu um entendimento profundo dos conceitos de *PLN* e técnicas de aprendizado de máquina, o que foi desafiador, mas também extremamente enriquecedor para nosso desenvolvimento acadêmico e profissional. Para treinar esses modelos, precisamos de muito poder de processamento e memória, algo que pode ser difícil de conseguir em locais onde não há computadores potentes ou infraestrutura adequada.

Os pontos positivos do nosso projeto são evidentes. Conseguimos demonstrar que é possível obter resultados relevantes em *PLN* utilizando apenas CPU, com a ajuda do Gensim e do Word2Vec. Isso abre portas para muitos outros pesquisadores que talvez não tenham acesso a recursos de hardware avançados. Além disso, nosso trabalho contribuiu para o entendimento das nuances da língua portuguesa, um campo que ainda é menos explorado em comparação com o inglês. No entanto, também identificamos algumas limitações que devem ser consideradas em trabalhos futuros. A sensibilidade dos modelos à qualidade dos dados de treinamento e a necessidade de grandes volumes de dados são aspectos que podem ser melhorados com o avanço da tecnologia.

Socialmente, a aplicação do processamento de linguagem natural tem um impacto significativo em diversas áreas. Conseguimos pensar em algumas direções promissoras para expandir trabalhos como este que se baseiam no estudo do idioma. Desde a automação de atendimento ao cliente até a análise de grandes volumes de dados em redes sociais.

Um aspecto importante seria a criação de parcerias com outras instituições e empresas para desenvolver bases de dados mais representativas do português brasileiro em diversas situações. Essas parcerias poderiam incluir não apenas universidades, mas também empresas de tecnologia, órgãos governamentais e organizações não governamentais. A colaboração poderia ajudar a reunir um conjunto diversificado de dados que refletisse melhor a riqueza e a diversidade do nosso idioma.

Outra área de interesse seria a aplicação das técnicas desenvolvidas em nosso trabalho para resolver problemas específicos em setores como educação, saúde e serviços ao cliente. No setor educacional, por exemplo, poderíamos usar modelos de *PLN* para criar ferramentas que auxiliem no aprendizado de idiomas, oferecendo feedback personalizado e adaptado ao nível de cada aluno. Na área da saúde, esses modelos poderiam ser usados para analisar grandes volumes de dados clínicos e ajudar os profissionais a identificar padrões e tendências que poderiam passar despercebidos em análises manuais. Em serviços ao cliente, a aplicação de *PLN* pode melhorar significativamente a interação entre empresas e consumidores, oferecendo respostas mais rápidas e precisas às dúvidas e problemas apresentados.

Outro ponto crucial é a ética no desenvolvimento e aplicação dessas tecnologias. É fundamental garantir que os modelos de *PLN* sejam desenvolvidos e utilizados de maneira responsável, respeitando a diversidade linguística do português brasileiro e evitando vieses que possam prejudicar determinados grupos de usuários. Ao treinar nossos modelos, precisamos considerar as variações regionais e as gírias que existem em diferentes partes do Brasil. Um modelo treinado apenas com dados de São Paulo pode não entender adequadamente expressões usadas no Nordeste ou no Sul do país, perpetuando estereótipos regionais e linguísticos. Além disso, expressões populares em determinadas comunidades podem ser negligenciadas se não houver uma coleta ampla e representativa de dados. Para resolver isso, devemos incorporar princípios éticos desde a fase de coleta de dados até a implementação final dos modelos, garantindo que todas as variações do português brasileiro sejam representadas de maneira justa e por igual. Isso inclui revisar constantemente os dados de treinamento para detectar e corrigir vieses, além de garantir que as decisões tomadas pelos modelos possam ser explicadas e justificadas de maneira clara para os usuários.

Com isso concluímos que nosso trabalho atingiu seus objetivos e proporcionou uma base sólida para futuras pesquisas em processamento de linguagem natural. As facilidades e dificuldades que enfrentamos ao longo do caminho contribuíram para um aprendizado profundo e significativo. Acreditamos que as metodologias e resultados apresentados não só contribuirão significativamente para o campo de *PLN*, especialmente no contexto da língua portuguesa, mas também servirão como um guia útil para futuros pesquisadores interessados na área. Com o contínuo avanço da tecnologia, esperamos ver desenvolvimentos ainda mais impressionantes que transformarão a forma como interagimos com o texto e a informação.

REFERÊNCIAS

ABADI, M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. **arXiv preprint arXiv:1603.04467**, 2016. Disponível em: <https://arxiv.org/abs/1603.04467>.

ALMEIDA, T. L. de; NUNES, M. das G. V. The Mac-Morpho Brazilian Portuguese corpus. In: **Proceedings of the 2nd Workshop on Linguistic Annotation**. Sapporo, Japan: Association for Computational Linguistics, 2003. p. 65–72. Disponível em: <http://nilc.icmc.usp.br/macmorpho/>.

BIRD, S.; LOPER, E. **Natural Language Processing with Python**. Sebastopol, CA: O'Reilly Media, 2009. ISBN 978-0596516499.

CARLSON, A. et al. Toward an architecture for never-ending language learning. In: . [S.l.: s.n.], 2010. v. 3.

FISHER, R. A. The use of multiple measurements in taxonomic problems. **Annals of Eugenics**, Wiley Online Library, v. 7, n. 2, p. 179–188, 1936. Disponível em: <https://archive.ics.uci.edu/dataset/53/iris>.

FOUNDATION, W. **Wikimedia Dumps**. 2024. Acessado em julho de 2024. Disponível em: <https://dumps.wikimedia.org/>.

HARPOLE, D. C. et al. The lemur project: A new approach to information retrieval. In: **Proceedings of the 2005 ACM SIGIR Conference on Research and Development in Information Retrieval**. Salvador, Brazil: ACM, 2005. p. 521–528.

LE, Q. V.; MIKOLOV, T. Distributed representations of sentences and documents. In: **Proceedings of the 31st International Conference on Machine Learning (ICML)**. Beijing, China: PMLR, 2014. (Proceedings of Machine Learning Research, v. 32), p. 1188–1196. Disponível em: <http://proceedings.mlr.press/v32/le14.pdf>.

MIKOLOV, T. et al. **Distributed representations of words and phrases and their compositionality**. 2013. Disponível em: <https://code.google.com/archive/p/word2vec/>.

PORFIRIO, F. **Empirismo**. 2024. Acesso em: 26 ago. 2024. Disponível em: <https://mundoeducacao.uol.com.br/filosofia/empirismo.htm>.

REHUREK, R.; SOJKA, P. Software framework for topic modelling with large corpora. In: **Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks**. Valletta, Malta: [s.n.], 2010. p. 45–50. Disponível em: <http://www.lrec-conf.org/proceedings/lrec2010/workshops/W14/papers/rehurek.pdf>.

Wikipedia. **Linguística**. 2024. Acesso em: 26 ago. 2024. Disponível em: <https://pt.wikipedia.org/wiki/Lingu%C3%ADstica>.

WITTEN, I. H. et al. **Data Mining: Practical Machine Learning Tools and Techniques**. [S.l.]: Elsevier Inc., 2016. 1-621 p. ISBN 9780128042915.

GLOSSÁRIO

Bag of Words (BoW) Uma representação simplificada de texto em que um documento é descrito pelo número de vezes que cada palavra aparece, sem considerar a ordem das palavras.

BeautifulSoup Uma biblioteca de Python usada para extrair e analisar dados de arquivos HTML e XML.

CPU CPU (Central Processing Unit, ou "Unidade de Processamento Central"): É o processador principal de um computador, responsável por executar as instruções de um programa.

CSV Comma-Separated Values, um formato de arquivo onde os dados são armazenados em texto simples e separados por vírgulas. .

ClueWeb Um grande corpus de dados coletados da web, frequentemente utilizado para pesquisa em recuperação de informação, processamento de linguagem natural e aprendizado de máquina.

Continuous Bag of Words (CBOW) Um modelo de aprendizado de máquina utilizado para treinar representações vetoriais de palavras (embeddings) em processamento de linguagem natural. No modelo CBOW, o objetivo é prever uma palavra-alvo com base no contexto dado pelas palavras adjacentes, utilizando uma abordagem de janela de contexto contínua.

Flask Um microframework de desenvolvimento web em Python.

GPU GPU (Graphics Processing Unit, ou "Processador Gráfico"): É um tipo de processador especializado em realizar cálculos matemáticos de alto desempenho.

Gensim Uma biblioteca de Python para modelagem de tópicos e processamento de linguagem natural. .

HTML HTML (Hypertext Markup Language) é uma linguagem de marcação usada para criar páginas da web.

KNN (K-nearest neighbors) K-nearest neighbors (KNN) é um algoritmo de classificação e regressão que classifica um dado com base na maioria dos votos de seus K vizinhos mais próximos.

Mac-Morpho Um corpus de dados anotados de morfologia para o português brasileiro, utilizado para tarefas de análise linguística e desenvolvimento de ferramentas de processamento de linguagem natural.

NELL Never-Ending Language Learning (NELL) é um projeto de aprendizado de máquina desenvolvido pela Universidade Carnegie Mellon que visa extrair e aprender continuamente informações de texto em linguagem natural.

NLTK Natural Language Toolkit (NLTK) é uma biblioteca de Python para o processamento de linguagem natural.

Naive Bayes Algoritmo de classificação baseado no teorema de Bayes com a suposição de independência entre as características.

PLN Processamento de Linguagem Natural.

Python Uma linguagem de programação de alto nível, interpretada e de uso geral, conhecida por sua sintaxe clara e legibilidade.

QR-Codes A forma plural de *QR-Code*.

QR-Code Um QR-Code (sigla do inglês "Quick Response", que significa "resposta rápida" em português) é um código de barras bidimensional.

SQL Structured Query Language, uma linguagem padrão usada para gerenciar e manipular bancos de dados relacionais.

SpaCy Uma biblioteca de processamento de linguagem natural em Python, projetada para ser rápida e eficiente.

TXT Um formato de arquivo de texto simples que armazena dados sem formatação. .

TensorFlow Uma biblioteca de código aberto desenvolvida pelo Google para o desenvolvimento e treinamento de modelos de aprendizado de máquina e redes neurais.

WARC wARC, sigla para Web ARChive, representa um formato de arquivo utilizado para arquivar e preservar páginas da web e seus metadados.

Wikicorpus Um conjunto de dados extraído da Wikipédia, geralmente utilizado para tarefas de processamento de linguagem natural e aprendizado de máquina.

XML eXtensible Markup Language (XML) é uma linguagem de marcação usada para definir regras de estrutura e formatação de dados em documentos.

chatbots Programas de computador projetados para simular conversas com usuários humanos.

clustering Técnica de aprendizado de máquina que visa agrupar um conjunto de objetos de forma que objetos no mesmo grupo (ou cluster) sejam mais semelhantes entre si do que com aqueles em outros grupos.

corpus Um conjunto de textos ou dados usados para análise linguística ou treinamento de modelos de aprendizado de máquina.

crawlers A forma plural de *crawler*.

crawler Um programa de computador que busca automaticamente informações na internet, geralmente para indexar o conteúdo da web.

dumps A forma plural de *dump*.

dump Um arquivo que contém uma cópia do conteúdo de um banco de dados ou outra forma de armazenamento de dados. .

embeddings As representações vetoriais das palavras são chamadas embeddings.

git Sistema de controle de versão distribuído para rastrear mudanças no código-fonte e colaborar em projetos de software.

lematização Processo de reduzir palavras às suas formas básicas ou lemas. Ao contrário da *tokenização*, que divide o texto em unidades menores, a *lematização* visa normalizar palavras variáveis, como verbos e substantivos, para uma forma base.

looping Looping de programação refere-se à repetição controlada de um bloco de código em um programa. .

mwxml Uma biblioteca de Python para trabalhar com arquivos XML da Wikipédia.

one-hot encoding Uma técnica de codificação utilizada para representar dados categóricos como vetores binários. Cada categoria é transformada em um vetor de comprimento igual ao número de categorias, onde apenas a posição correspondente à categoria presente é marcada com 1, enquanto todas as outras posições são 0.

parsing Análise de uma sequência de símbolos (como palavras) para determinar sua estrutura gramatical e significado. Imagine um explorador desbravando a linguagem.

scripts A forma plural de *script*.

script Um conjunto de instruções escritas em uma linguagem de programação que é executado para realizar uma tarefa específica.

stemming Técnica de processamento de texto que reduz palavras infletidas à sua forma base.

stopwords Stopwords são palavras comuns que são filtradas antes ou depois do processamento de texto, pois são consideradas insignificantes .

string Uma sequência de caracteres usada para representar texto em programação e ciência da computação.

threads A forma plural de *thread*.

thread Uma thread é uma unidade de execução dentro de um processo.

tokenizadas Separar em *token*.

tokenização O processo de dividir um texto em unidades menores chamadas *tokens*, que podem ser palavras, frases ou símbolos.

tokens A forma plural de *token*.

token Token é um valor único que pode ser usado para identificar um indivíduo ou objeto.

toolkit Conjunto de ferramentas.

web scraping Técnica de extração de dados de sites da web por meio de programas que simulam a navegação e a coleta de informações das páginas web.

wikitext Linguagem de marcação usada para formatar e estruturar páginas em wikis, como a Wikipédia. Permite a criação de conteúdos com links, listas e formatação de texto sem a necessidade de codificação HTML.

APÊNDICE A – LINK DO PROJETO NO GITHUB

Ao longo do texto, foram disponibilizados pequenos trechos do código para contextualizar e explicar as seções de forma didática.

Para acessar o código-fonte completo e os materiais adicionais deste projeto acesso o repositório no Github.

O repositório contém todo o código desenvolvido durante a realização deste trabalho de conclusão de curso, incluindo scripts, documentação e exemplos de uso.

O objetivo da disponibilização dos códigos é permitir que outros pesquisadores e desenvolvedores possam reproduzir, avaliar e contribuir com melhorias para o projeto. Visite o nosso repositório pelo link abaixo:

<https://github.com/sevenleo/tcc>

APÊNDICE B – ARQUIVOS FINAIS E ACURÁCIA

Como o *git* do projeto engloba um vasto conteúdo de aprendizado, indicamos a subpasta abaixo para quem desejar avaliar as últimas etapas do projeto e verificar os scripts que geraram os resultados finais:

<https://github.com/sevenleo/tcc/tree/master/word2vec/>

APÊNDICE C – MODELOS E MATRIZES CRIADAS

Como os modelos e as matrizes geradas excedem o tamanho máximo permitido pelo Github, estes arquivos foram disponibilizados através do Google Drive no link abaixo. Os modelos estão organizados em arquivos/pastas com as seguintes características conforme o nome.

Exemplos:

- **Modelo-00313-sem-lematizador.7z**:

  foram analisados 313/1151 arquivos do ClueWeb e não foi aplicado o lematizador nas palavras.

- **Modelo-00615-com-lematizador.7z**:

  foram analisados 615/1151 arquivos do ClueWeb e foi aplicado o lematizador nas palavras.

Link para download dos arquivos:
<https://drive.google.com/drive/folders/1N3Xg54dCJZ9BLCF04JiaGsxZQiZlcynE?usp=sharing>