Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Estatística

Processamento de linguagem natural aplicado à análise de sentimentos em opiniões de usuários

Tayná Ferreira Santana

Trabalho de Conclusão de Curso

Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Estatística

Processamento de linguagem natural aplicado à análise de sentimentos em opiniões de usuários

Tayná Ferreira Santana
Orientadora: Profa. Dra. Andressa Cerqueira

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do título de Bacharel em Estatística.

São Carlos
Julho de 2025

Federal University of São Carlos
Exact and Technology Sciences Center
Department of Statistics

Natural Language Processing applied to sentiment analysis in user opinions

Tayná Ferreira Santana
Advisor: Profa. Dra. Andressa Cerqueira

Bachelors dissertation submitted to the Department of Statistics, Federal University of São Carlos - DEs-UFSCar, in partial fulfillment of the requirements for the degree of Bachelor in Statistics.

São Carlos
July 2025

Tayná Ferreira Santana

Processamento de linguagem natural aplicado à análise de sentimentos em opiniões de usuários

Este exemplar corresponde à redação final do trabalho de conclusão de curso devidamente corrigido e defendido por Tayná Ferreira Santana e aprovado pela banca examinadora.

Aprovado em 03 de julho de 2025.

Banca Examinadora:

- Profa. Dra. Andressa Cerqueira
- Profa. Dra. Daiane Aparecida Zuanetti
- Prof. Dr. Gustavo Henrique de Araujo Pereira

Resumo

Processamento de Linguagem Natural (PLN), do inglês *Natural Language Processing* (NLP), é uma área da inteligência artificial que permite a interação entre computador e linguagem natural entre humanos, seja por meio de áudios ou textos, considerado como um avanço tecnológico significativo e atual. Com um grande conjunto de dados, surge uma crescente necessidade de utilizar aprendizado de máquina para processar textos e torná-lo capaz de compreender os sentimentos expressos pelo autor. Através de técnicas de análise de sentimentos, classificam-se essas expressões em opiniões positivas ou negativas, não sendo uma tarefa fácil considerando haver desafios como idiomas, sarcasmos, ambiguidade, uso de gírias e expressões, além de diferentes contextos culturais. Assim, este trabalho foca na aplicação de análise de sentimentos de opiniões de usuários sobre produtos de beleza, utilizando modelos de aprendizado de máquina para a classificação e interpretação de textos.

Palavras-chave: *análise de sentimentos, Bag-of-Words, processamento de linguagem natural, TF-IDF, Word2Vec.*

Abstract

Natural Language Processing (NLP) is a field of artificial intelligence that enables interaction between computers and human natural language, whether through audio or text. It is considered a significant and current technological advancement. With the availability of large datasets, there is a growing need to use machine learning to process text and enable systems to understand the sentiments expressed by the author. Through sentiment analysis techniques, these expressions are classified into positive or negative opinions. However, this is not an easy task, given the challenges such as different languages, sarcasm, ambiguity, the use of slang and expressions, as well as varying cultural contexts. Therefore, this work focuses on applying sentiment analysis to user opinions about beauty products, using machine learning models for text classification and interpretation.

Keywords: *Bag-of-Words, natural language processing, sentiment analysis, TF-IDF, Word2Vec.*

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

O Processamento de Linguagem Natural (PLN) é uma área de pesquisa antiga, que surgiu praticamente em conjunto com os primeiros computadores em torno de 1940, tendo como problema inicial a tradução automática entre línguas, segundo Caseli e Nunes (2024). O objetivo do PLN torna-se descobrir e compreender métodos de processamento computacional da linguagem humana.

Atividades como encontrar documentos inteiros ou fragmentados em bases de dados que satisfaçam a consulta do usuário (*Information Retrieval*), encontrar informações específicas e estruturadas em documentos não estruturados (*Information Extraction*), responder a perguntas feitas pelo usuário (*Question Answering*) e tradução de um idioma para outro (*Machine Translation*), são descritas em Indurkhya e Damerau (2010) e Jurafsky e Martin (2024), e pertencem ao universo PLN. Contudo, abordaremos neste trabalho a área de análise de sentimentos, do inglês chamado de *Sentiment Analysis* ou *Opinion Mining*.

Em análise de sentimentos, temos como objeto de estudo as opiniões e seus conceitos relacionados, como sentimentos, apreciações, avaliações, atitudes e emoções, que são analisados em relação a entidades como produtos, serviços, eventos, indivíduos e empresas, por exemplo, focando em opiniões que ocasionam sentimentos negativos ou positivos. Esse campo de pesquisa evoluiu significativamente no início dos anos 2000, se tornando uma das áreas mais procuradas em PLN, como visto em Liu (2022). Apesar da identificação de um sentimento em uma opinião aparentar ser uma tarefa simples, enfrentamos o desafio de analisar imensas quantidades de dados, tornando necessário o uso de métodos computacionais.

Para muitos, detectar sentimentos positivos ou negativos é somente uma etapa. Logo,

surgem objetivos e motivações mais ambiciosas, como utilizar a análise de sentimentos para então prever o desempenho de vendas (Liu *et al.*, 2007), classificar produtos e comerciantes (McGlohon *et al.*, 2010), compreender a relação entre a linha de apostas da NFL e opiniões públicas de usuários no Twitter (Hong e Skiena, 2010), prever resultados eleitorais (Tumasjan *et al.*, 2010) ou até mesmo prever o mercado de ações (Bollen *et al.*, 2011). Nesse sentido, compreender o sentimento de seus usuários se torna um ponto importante para direcionar tomadas de decisões no ramo empresarial, político, social ou econômico.

Sendo assim, propomos neste Trabalho de Conclusão de Curso o estudo e a aplicação de análise de sentimentos usando dados abertos da Amazon, empresa multinacional norte-americana, atualizados no ano 2023. Os dados em estudo descrevem *reviews* (opiniões) de usuários sobre produtos de beleza que compraram no site da Amazon. As implementações foram realizadas utilizando a linguagem Python. Dessa forma, os objetivos deste trabalho são:

- Aplicação de inteligência artificial em textos a partir de dados reais com enfoque no estudo do processamento computacional de textos e sua aplicação em dados de opiniões de usuários sobre produtos de beleza da Amazon;
- Estudo teórico e implementação de métodos de pré-processamento de textos;
- Aplicação de análise de sentimentos, implementando modelos de aprendizado de máquina para classificação e interpretação de textos;
- Estudo preditivo e interpretativo sobre opiniões de usuários.

Como organização do trabalho, informamos que traremos os principais fundamentos e nomenclaturas do pré-processamento de textos no Capítulo 2. No Capítulo 3, estudaremos diferentes metodologias, como Bag-of-Words, TF-IDF e Word2Vec, que visam identificar e/ou interpretar sentimentos positivos e negativos, através de algoritmos de aprendizado de máquina supervisionado. Na sequência, o Capítulo 4 apresenta os resultados obtidos após serem postos em prática todos os conceitos abordados nos capítulos antecedentes, para, por fim, as considerações finais serem apresentadas no Capítulo 5.

Capítulo 2

Pré-Processamento

Este capítulo aborda os principais fundamentos e nomenclaturas do pré-processamento de dados no contexto de processamento de linguagem natural. Inicialmente, discutimos conceitos essenciais como *documentos*, *corpus*, *tokens*, e técnicas como *tokenization*, *lemmatization*, *stemming* e a remoção de *stop words*. Essas etapas são fundamentais para preparar o texto de maneira adequada para uma análise com qualidade interpretativa. Em seguida, apresentamos o banco de dados utilizado, o qual contém milhares de opiniões de usuários sobre produtos de beleza, e que servirá como base para a aplicação das técnicas discutidas.

2.1 Fundamentos do Pré-Processamento

Para tratarmos do pré-processamento, primeiramente temos que estabelecer o conceito de algumas nomenclaturas populares no meio de PLN. Caseli e Nunes (2024) descrevem um *documento* como um único texto, podendo ser um livro, uma música ou até uma breve opinião, representado na forma de uma observação em um conjunto de dados. Dessa maneira, temos grandes conjuntos de textos, também chamados de *corpus*, que consistem em uma coleção de documentos. Jurafsky e Martin (2024) designa também o termo *expressão regular*, comumente abreviada como *regex*, sendo uma notação algébrica para caracterizar cadeias de caracteres que podem ser extraídas de um documento. *Token* também é um conceito popular em PLN, definido como um termo que representa uma sequência de caracteres, comumente associado à palavra escrita, visto em Caseli e Nunes (2024). Logo, um *token* pode ser uma palavra, um sinal de pontuação, um e-mail ou um número de telefone, por exemplo, basta atribuirmos significado a ele.

Na implementação de algoritmos para a resolução dos problemas, é necessário que haja uma tradução da linguagem humana (texto ou fala) para a computacional, sendo essa a etapa inicial conhecida como pré-processamento, a qual modifica o *corpus* de determinada maneira para que possa ser implementado por tal método, visto que diferentes métodos demandam diferentes estruturações dos dados.

Pelos dados (*reviews*) se tratarem de textos que, além de palavras, contêm pontuações, símbolos ou números, normalizamos o texto, ou seja, o transformamos em um formato padronizado e conveniente para a análise. Por isso, autores como Brownlee (2017), Caseli e Nunes (2024) e Jurafsky e Martin (2024) sugerem que apliquemos técnicas no *corpus* como:

- *Tokenization*: transformar os documentos em diversos *tokens*. Exemplo: no documento "Fui ao mercado e comprei pão, leite e água.", temos os *tokens* "Fui", "ao", "mercado", "e", "comprei", "pão", ",", "leite", "e", "água", ".";
- *Lemmatization*: transforma a palavra em seu lema, ou seja, em sua base. Exemplo: as palavras "correr", "corre" e "corremos" são convertidas para a palavra "correr";
- *Stemming*: retiramos os prefixos e sufixos da palavra, deixando apenas seu radical. Exemplo: as palavras "caminhando", "caminhou" e "caminha" são convertidas para "caminh";
- Retirar *Stop Words*: retiramos palavras que não possuem grande importância para a compreensão e significado de uma frase. Exemplos: "a", "em", "com" e "na".

Bird *et al.* (2009) e Brownlee (2017) abordam uma série de maneiras de utilizar a linguagem Python em PLN, inclusive na realização do pré-processamento, embora Paaß e Giesselbach (2023) diga não ser mais necessário, visto que na maior parte dos casos já existam métodos modernos que derivam e implementam tais ações automaticamente.

Para facilitar a consulta dos principais conceitos abordados no pré-processamento de textos, a Tabela 2.1 reúne os termos definidos anteriormente, acompanhados de uma breve descrição.

Tabela 2.1: Glossário de termos utilizados em Processamento de Linguagem Natural.

Termo	Descrição
Documento	Único texto representado na forma de uma observação no conjunto de dados.
Corpus	Coleção de documentos.
Regex	Notação algébrica para caracterizar cadeias de caracteres que podem ser extraídas de um documento.
Token	Sequência de caracteres, comumente associado à palavra escrita.
Tokenization	Técnica para transformar os documentos em diversos *tokens*.
Lemmatization	Técnica para transformar a palavra em seu lema, ou seja, em sua base.
Stemming	Técnica para retirar os prefixos e sufixos da palavra, deixando apenas seu radical.
Stop Words	Palavras que não possuem grande importância para a compreensão e significado de uma frase.

2.2 Banco de Dados

Os dados que serão analisados nesse trabalho referem-se a mais de 700 mil opiniões de usuários sobre produtos de beleza que compraram no site da Amazon. Os dados foram coletados em 2023, possuindo opiniões de vários anos. O conjunto de dados conta com avaliação, data de avaliação e opinião (em texto) do produto feita pelo usuário, por exemplo, além do nome, descrição, preço e imagem do item. Os dados são públicos e estão disponíveis em https://amazon-reviews-2023.github.io/.

Capítulo 3

Métodos

Uma vez pré-processados, os documentos serão submetidos a diferentes metodologias que visam representar quantitativamente os textos. A primeira que abordaremos é o *Bag-of-Words* (BoW), seguido do segundo método chamado de *Term Frequency - Inverse Document Frequency* (TF-IDF). Por fim, também exploraremos o método *Word2Vec* (W2V). As matrizes geradas por essas três metodologias servem de base para a criação de covariáveis em modelos estimados por algoritmos de aprendizado de máquina supervisionado, destinados à identificar e/ou interpretar sentimentos positivos e negativos.

3.1 Bag-of-Words

Esse método consiste em criar covariáveis para cada documento do *corpus*, possuindo uma representação vetorial, segundo Izbicki e dos Santos (2020). Para isso, catalogamos todas as palavras existentes no *corpus* e contabilizamos a frequência com que cada palavra (*token* padronizado) aparece em cada documento. Um ponto destacado por Manning e Schutze (1999), é que a ordem e estruturação das palavras é ignorada, afetando palavras dependentes entre si, apesar de haver uma grande quantidade de casos em que BoW obtém bons resultados.

Como o próprio nome indica, o método pode ser visto como um saco de palavras (tradução de *bag of words*), pois as palavras que constroem o texto se embaralham fazendo com que a ordem delas deixe de importar, perdendo assim a referência de ordem. Isso acontece porque o Bag-of-Words se preocupa apenas com as palavras e a frequência com que elas aparecem. Por exemplo, temos 3 documentos (sem pré-processamento) a seguir representados na Tabela 3.1:

- Documento 1: “O cachorro corre. O cachorro dorme.”;
- Documento 2: “O gato dorme.”;
- Documento 3: “O cachorro e o gato brincam.”.

Tabela 3.1: Bag-of-Words para os documentos.

Documento	**O**	**cachorro**	**corre**	**dorme**	**gato**	**e**	**brincam**	**.**
Documento 1	2	2	1	1	0	0	0	2
Documento 2	1	0	0	1	1	0	0	1
Documento 3	2	1	0	0	1	1	1	1

Assim, a representação do BoW seria uma matriz esparsa (que contém muitos zeros), como indica Zhao e Mao (2017), onde cada linha indica um documento e cada coluna indica um *token*. Ao final, utilizamos essa matriz e tratamos cada *token* como uma covariável para a implementação de algoritmos como Naive Bayes, Redes Neurais, Floresta Aleatória, entre outros, gerando um modelo capaz de classificar novos documentos e avaliar a influência de cada *token* para cada classe disponível.

Desse modo, a proposta do BoW se baseia em reconhecer que se determinadas palavras aparecem em determinados tipos de textos, já é suficiente para tomar a decisão de classificação. Nesse sentido, BoW acaba sendo uma abordagem simplória, mas que ainda exige vários cuidados adicionais. A depender do objetivo, a aparição de algumas palavras não deveria ser, intuitivamente, responsável pela classificação do texto. Por exemplo, se queremos classificar o texto como sendo da área de física ou de biologia, a palavra “ciclo” não seria suficiente, visto que em ambas as áreas temos a presença dela em vários processos. Assim, por mais que tivesse elevada frequência nos dois tipos de texto, não seria tão razoável utilizar tal palavra como critério de decisão. O contrário ocorreria com a palavra “metabolismo”, pois é bem mais provável surgir em textos de biologia do que de física.

3.2 TF-IDF

O *Term Frequency - Inverse Document Frequency* (TF-IDF) aparece como um método alternativo em Caseli e Nunes (2024) e Brownlee (2017). Ele ajuda a avaliar a importância das palavras dentro de um *corpus*, identificando termos significativos que podem ser usados para algoritmos de busca ou de classificação, por exemplo. Em vez de contagem de

palavras, usamos um peso acoplado a respectiva frequência, pois um problema com contagens simples, como em BoW, é que algumas palavras, como *stop words*, surgirão em grandes quantidades, mesmo que não sejam muito significativas para a compreensão da opinião. Portanto, TF-IDF sugere que, sendo t determinado termo e d um documento,

- *Term Frequency* (TF): frequência relativa que determinada palavra (termo) aparece dentro de um documento:

$$tf(d,t) = \frac{\text{número de ocorrências de } t \text{ em } d}{\text{total de termos em } d};$$

- *Inverse Document Frequency* (IDF): avalia raridade (importância relativa) da palavra dentro do *corpus*, dando peso maior a palavras menos frequentes:

$$idf(t) = \log_{10}\left(\frac{\text{número de documentos}}{\text{número de documentos que contém } t}\right);$$

- *Term Frequency - Inverse Document Frequency* (TF-IDF): avalia frequência do termo no documento e raridade do termo no *corpus*:

$$tf\text{-}idf(t,d) = tf(d,t) \cdot idf(t).$$

O domínio e intuição de cada componente é:

- $0 \leq tf(d,t) \leq 1$: a intuição por trás do TF é que, se uma palavra aparece frequentemente em um documento, ela provavelmente é importante para o conteúdo dele. Então, quanto mais próximo de 0, menos importante a palavra é para seu documento. O contrário acontece quando nos aproximamos do valor 1;

- $0 \leq idf(t) \leq \log_{10}(\text{número de documentos})$: em IDF, se uma palavra aparece em muitos documentos, ela provavelmente é menos importante para identificar documentos, pois é considerada como uma palavra comum, não ajudando a distinguir os documentos uns dos outros. Logo, quanto mais próximo de 0, mais recorrente a palavra é dentre os documentos e, quanto mais próximo de $\log_{10}(\text{número de documentos})$, mais a palavra será exclusiva de um pequeno conjunto de documentos;

- $0 \leq tf\text{-}idf(t,d) \leq \log_{10}(\text{número de documentos})$: como resultado, o TF-IDF se torna uma mescla dos conceitos de frequência e raridade das palavras. A me-

dida se aproxima de 0 quando a palavra é pouco frequente no documento e/ou aparece em grande proporção de documentos. Em contrapartida, se aproxima de $\log_{10}$(número de documentos) quando a palavra é bem frequente no documento e/ou aparece em um número restrito de documentos, entendendo que essas palavras são mais importantes para distinção dos documentos.

Como exemplo, usamos os mesmos documentos (sem pré-processamento) a seguir, e aplicamos sobre eles o TF-IDF, representado na Tabela 3.2:

- Documento 1: "O cachorro corre. O cachorro dorme.";
- Documento 2: "O gato dorme.";
- Documento 3: "O cachorro e o gato brincam.".

Tabela 3.2: TF-IDF para os documentos.

Documento	**O**	**cachorro**	**corre**	**dorme**	**gato**	**e**	**brincam**	**.**
Documento 1	0	0.0440	0.0596	0.0220	0	0	0	0
Documento 2	0	0	0	0.0440	0.0440	0	0	0
Documento 3	0	0.0252	0	0	0.0252	0.0681	0.0681	0

Note que assim como em BoW, o TF-IDF não considera a ordem das palavras, tomando a suposição de que as palavras são independentes entre si. Também é representado por matriz esparsa, tendo suas linhas e colunas a mesma função de representação definidas na seção anterior (3.1). Por fim, também utilizamos tal matriz para aplicação de algoritmos de aprendizado de máquina, como feito em Akuma *et al.* (2022).

3.3 Word2Vec

Desenvolvido por Mikolov *et al.* (2013), o Word2Vec possui como princípio que palavras usadas em contextos parecidos tendem a ter significados parecidos (Caseli e Nunes, 2024). Diferente do BoW e do TF-IDF, que se limitam à contagem de palavras, ignorando a ordem de aparição no documento e a relação entre elas, o W2V consegue capturar a ordem e proximidade entre palavras em um texto, aprendendo relações semânticas entre elas e representando-as em vetores densos com valores contínuos. Esses vetores são frequentemente denominados de *embeddings* estáticos, pois são densas representações numéricas

fixas de dados, como palavras, em um espaço de alta dimensão, como relata Jurafsky e Martin (2024).

Nesse sentido, temos duas possíveis arquiteturas para a construção do vetor de cada palavra: Continuous Bag-of-Words (CBOW) e Skip-Gram. Ambas as arquiteturas utilizam redes neurais com, geralmente, uma camada oculta para modelar as representações vetoriais de cada termo. A diferença entre elas é que, no CBOW, o propósito do modelo é prever uma determinada palavra utilizando palavras de contexto ou vizinhas (palavras que antecedem e sucedem a palavra alvo), enquanto que no Skip-Gram, o objetivo é o inverso: o modelo prevê as palavras vizinhas utilizando uma única palavra. Como exemplo temos a frase "Esse sabonete facial é cheiroso". No CBOW, se as palavras de contexto fossem "Esse", "sabonete", "é" e "cheiroso", tentaríamos prever a palavra alvo "facial". No Skip-Gram, através da palavra "facial" buscaríamos prever as palavras de contexto: "Esse", "sabonete", "é" e "cheiroso".

A Figura 3.1 ilustra a estrutura dos algoritmos. Na camada de entrada ($INPUT$) temos um neurônio para cada palavra vizinha no CBOW, e um único neurônio para a palavra alvo no Skip-Gram. Na camada de saída ($OUTPUT$) temos o contrário. Na camada oculta ($PROJECTION$) temos uma quantidade fixa de neurônios definidas previamente ao treinamento do modelo, conforme explica Caseli e Nunes (2024). Note que nas camadas de entrada e de saída, podemos utilizar a representação *one-hot* para as palavras do *corpus*, ou seja, um vetor binário que possui 1 na posição correspondente à palavra associada e 0 nas demais. Nesse sentido, para obtermos o *embedding* de cada palavra, basta que, após o treinamento do modelo, selecionemos os pesos da respectiva palavra na camada de entrada conectados à camada oculta, isto é, os valores conectados à sua representação *one-hot*, como indica Chang *et al.* (2017).

Contudo, para o treino é necessário a definição de um tamanho para a janela de contexto, a qual refere-se ao número de palavras vizinhas que antecedem e sucedem a palavra alvo. A Figura 3.2 apresenta uma ilustração do funcionamento do treinamento da arquitetura Skip-Gram com tamanho de janela igual a 2. Dharma *et al.* (2022) e Chang *et al.* (2017) explicam que, para um determinado documento, inicialmente identificamos a palavra alvo (em vermelho) como sendo sua primeira palavra, seguida das palavras vizinhas (em azul), a depender do tamanho da janela. Posteriormente, a janela é deslizada ao longo do texto, deslocando-se em uma palavra por vez, alterando a palavra alvo para a próxima palavra e mudando as palavras de contexto do momento. Executamos esse

procedimento percorrendo todas as palavras do documento, até que chegue na última. Fazemos isso para todo o *corpus*, para então finalizarmos o treino. De forma análoga, o mesmo processo ocorre no CBOW.

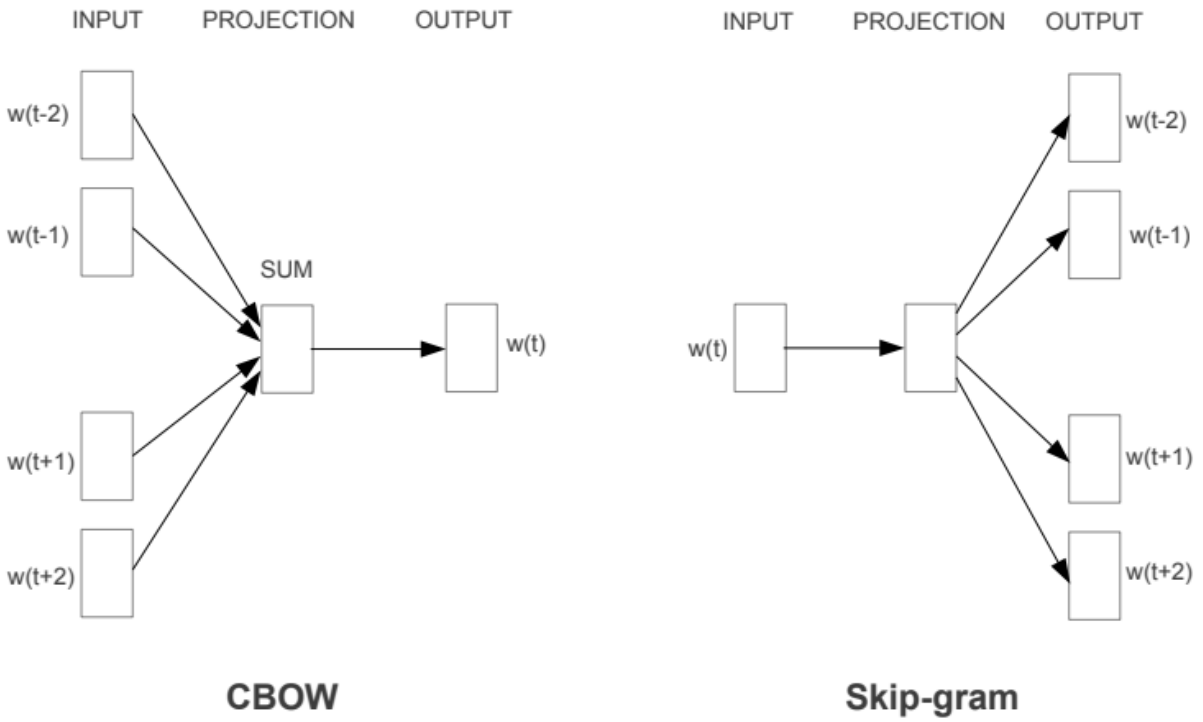


Figura 3.1: Arquitetura CBOW e Skip-Gram. Fonte: Mikolov *et al.* (2013).

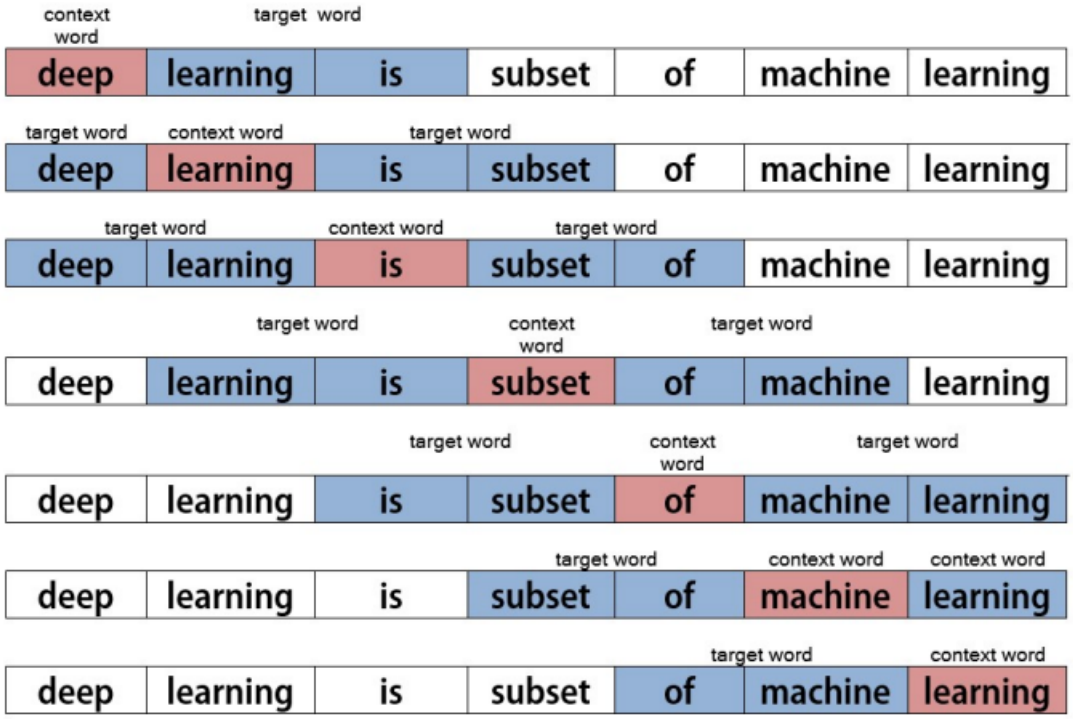


Figura 3.2: Ilustração do treinamento de um documento na arquitetura Skip-Gram com tamanho de janela igual a 2. Fonte: Dharma *et al.* (2022).

Por isso, o Word2Vec é considerado como sendo um algoritmo de aprendizado de máquina auto supervisionado, pois embora não precise de uma variável resposta externa para aprender, ele utiliza o próprio *corpus* como supervisão para aprender (Jurafsky e Martin, 2024).

Desse modo, conseguimos definir a dimensão dos *embeddings* a partir da quantidade de neurônios da camada oculta que treinamos o modelo, uma vez que ambos os números são equivalentes. Como exemplo, definimos o documento 1 como sendo o texto "Esse sabonete facial é cheiroso". Suponha que o modelo tenha sido estimado a partir de uma rede neural cuja camada oculta possui 100 neurônios. Logo, o *embedding* de cada palavra do documento 1 possui 100 elementos/posições. Veja uma simulação na Tabela 3.3.

Tabela 3.3: Matriz de *embeddings* com 100 dimensões para as palavras do documento 1.

Palavra	**Elemento**						
	1	2	3	...	98	99	100
Esse	0.12	0.55	-0.31	...	0.22	-0.44	0.10
sabonete	0.88	-0.21	0.36	...	0.49	0.17	-0.72
facial	0.35	0.19	0.25	...	0.05	0.60	-0.41
é	-0.15	0.40	-0.08	...	-0.02	-0.10	0.33
cheiroso	0.76	-0.13	0.50	...	0.31	0.45	-0.59

Todavia, surge um novo desafio: estamos conduzindo textos com mais de uma palavra, sendo que o Word2Vec fornece um *embedding* para cada palavra, assim, devemos encontrar uma maneira para que as opiniões dos usuários sejam representadas por um vetor. Diante de uma situação similar, Acosta *et al.* (2017) sugere fazer a média entre os vetores das palavras contidas em determinado documento, resultando em uma representação vetorial de mesma ordem para cada opinião. Essa sugestão será aplicada neste trabalho. A ideia de usar somente a soma de vetores é descartada, devido às diferentes quantidades de palavras em cada documento. Dessa forma, podemos usar cada dimensão/posição do vetor como covariável de um modelo de aprendizado de máquina supervisionado. A Tabela 3.4 apresenta um exemplo simulado dos vetores que representam cada documento.

Tabela 3.4: Matriz de *embeddings* com 100 dimensões para as médias das palavras de cada documento.

Documento	**Elemento**						
	1	2	3	...	98	99	100
Documento 1	0.392	0.16	0.144	...	0.21	0.136	-0.258
Documento 2	0.278	0.195	0.122	...	0.185	0.098	-0.214
Documento 3	0.412	0.247	0.159	...	0.229	0.134	-0.198

Com isso, uma desvantagem do Word2Vec é a falta de interpretabilidade sob as covariáveis, visto que as dimensões do vetor não possuem uma clara interpretação, conforme Jurafsky e Martin (2024). Apesar disso, o W2V oferece um interessante recurso. Se tivermos um modelo bem treinado, esperamos que *embeddings* semelhantes possuam palavras semelhantes semanticamente. Essa semelhança é calculada a partir da distância de cosseno (Li e Han, 2013), assim, quanto mais próxima de 1, maior é a similaridade entre os vetores. Também é possível realizar operações algébricas simples entre os *embeddings*, com a finalidade de encontrarmos o vetor da palavra com maior representatividade semântica. Por exemplo, espera-se que a operação vetor(Rei) - vetor(Homem) + vetor(Mulher) resulte em um vetor que seja o mais próximo possível do *embedding* da palavra "Rainha", segundo Mikolov *et al.* (2013).

3.4 Algoritmos de Aprendizado de Máquina

Como mencionado, algoritmos de aprendizado de máquina podem ser empregados após a construção da matriz BoW, TF-IDF ou W2V. Primeiro, é necessário contextualizar o presente problema de classificação. Recorde que um dos objetivos deste trabalho é prever se um determinado texto será classificado como positivo ou negativo perante o sentimento do autor. Dito isso, considere a variável aleatória Y_i (que também será chamada de variável resposta), a qual é qualitativa e assume duas possíveis classes:

$$Y_i = \begin{cases} 1, & \text{sentimento positivo expresso no } i\text{-ésimo documento} \\ 0, & \text{sentimento negativo expresso no } i\text{-ésimo documento} \end{cases}$$

Seja $\mathbf{X}_i$ o conjunto de covariáveis do i-ésimo documento, sendo considerado como um vetor esparso, em BoW e TF-IDF, dado que tende a conter uma baixa variedade de palavras em relação ao *corpus* inteiro, e um vetor denso, em W2V, dado que por construção possui valores reais em cada componente. Nessa direção, definimos $\mathbf{X}_{i,j}$ como a j-ésima covariável do conjunto do i-ésimo documento, de forma que $\mathbf{X}_i = (\mathbf{X}_{i,1}, \mathbf{X}_{i,2}, ..., \mathbf{X}_{i,d})$. Tome $i = 1, ..., n$ e $j = 1, ..., d$. Por característica dos métodos empregados, temos uma grande quantidade de covariáveis, já que nesse caso, se tratando de BoW e TF-IDF, as covariáveis adotadas são os *tokens* (palavras), de preferência pré-processados para melhores resultados. No W2V temos previamente fixada a quantidade de covariáveis. Por fim, temos uma amostra de observações independentes $(Y_1, \mathbf{X}_1), ..., (Y_n, \mathbf{X}_n)$, a qual

será utilizada para treinar os diversos modelos.

Assim, com o conjunto de covariáveis e variável resposta, queremos que o algoritmo informe corretamente a classe (positiva ou negativa) de sentimento expressa pelo autor. Matematicamente, definimos como $g(\mathbf{x})$ a predição do modelo estimado pelo algoritmo, a qual esperamos que seja da mesma categoria que seu respectivo y. Então, o cenário ideal para a predição de m novas observações seria:

$$g(\mathbf{x}_{n+1}) = y_{n+1}, ..., g(\mathbf{x}_{n+m}) = y_{n+m}.$$

Cada algoritmo pode proporcionar uma ampla gama de possíveis modelos preditivos que estimam $g(\mathbf{x})$, e para que isso seja evitado a fim de que o algoritmo retorne apenas o modelo mais adequado ao problema, devemos definir alguns critérios. O primeiro critério auxiliador será o uso da função de risco $R(g(\mathbf{X}))$. Ela busca quantificar o quanto estamos errando na predição de classes. Logo, quanto menor o risco (ou erro), melhor é a função de predição g. Embora tenhamos mais opções, empregamos a função de risco *log-loss* definida em Géron (2022):

$$R(g(\mathbf{X})) = \mathbb{E}\left[\mathbb{I}(Y = 1)(-\log(\mathbb{P}(Y = 1 \mid \mathbf{X}))) + \mathbb{I}(Y = 0)(-\log(\mathbb{P}(Y = 0 \mid \mathbf{X})))\right].$$

Como queremos minimizar $R(g(\mathbf{X}))$, por consequência temos que encontrar a classe $c \in \mathcal{C}$ que oferece maior probabilidade de ser a correta predição, veja Izbicki e dos Santos (2020). Portanto, a melhor função de classificação g será:

$$g(\mathbf{x}) = \arg\max_{c \in \mathcal{C}} \left(\mathbb{P}(Y = c \mid \mathbf{x})\right).$$

Todavia, usamos o estimador *plug-in* para realizar a estimação da probabilidade de Y ser determinado sentimento dado as covariáveis, logo:

$$g(\mathbf{x}) = \arg\max_{c \in \mathcal{C}} \left(\hat{\mathbb{P}}(Y = c \mid \mathbf{x})\right).$$

Assim, o estimador de $R(g(\mathbf{X}))$ é:

$$\hat{R}(g(\mathbf{X})) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(Y_i = 1)(-\log(\hat{\mathbb{P}}(Y_i = 1 \mid \mathbf{x}_i))) + \mathbb{I}(Y_i = 0)(-\log(\hat{\mathbb{P}}(Y_i = 0 \mid \mathbf{x}_i))).$$

3.4.1 LASSO

Desenvolvido por Tibshirani (1996), o primeiro algoritmo de aprendizado de máquina que utilizamos é o LASSO (*Least Absolute Shrinkage and Selection Operator*). A rigor, para o caso binário com função de ligação logito (Liao, 1994), é considerado como uma regressão logística com regularização através de uma penalização, para o problema de classificação, como explica Meier *et al.* (2008). Sua grande vantagem em relação a regressão logística sem penalização, é que é rápida computacionalmente e faz seleção de covariáveis. No caso em que temos apenas duas possíveis classes (caso binário), Wu *et al.* (2009) sugere que a regressão logística busque encontrar a predição do sentimento através de:

$$\mathbb{P}(Y = 1|\mathbf{x}) = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_d x_d}}{1 + e^{\beta_0+\beta_1 x_1+\beta_2 x_2+...+\beta_d x_d}},$$

em que cada β_j, para $j = 0, 1, ..., d$, são os coeficientes a serem ajustados, estando eles associados a alguma covariável. Dessa forma, o objetivo passa a ser minimizar o risco $R(g)$ via $\hat{R}(g)$, de modo que sejam escolhidos os melhores coeficientes, logo temos:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \left[\frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(Y_i = 1)(-\log(\mathbb{P}(Y_i = 1 \mid \mathbf{x}_i))) + \mathbb{I}(Y_i = 0)(-\log(\mathbb{P}(Y_i = 0 \mid \mathbf{x}_i))) \right].$$

Entretanto, para o método LASSO (L_1) ainda precisamos inserir uma penalização, que conta com uma constante λ, assim:

$$\hat{\boldsymbol{\beta}}_{L_1,\lambda} = \arg\min_{\boldsymbol{\beta}} \left[\frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(Y_i = 1)(-\log(\mathbb{P}(Y_i = 1 \mid \mathbf{x}_i))) + \mathbb{I}(Y_i = 0)(-\log(\mathbb{P}(Y_i = 0 \mid \mathbf{x}_i))) + \lambda \sum_{j=1}^{d} |\beta_j| \right].$$

Ao final, esperamos que o algoritmo faça uma boa predição de novas observações e que vários coeficientes estimados sejam iguais a zero, indicando que a respectiva covariável não é influente ou necessária para a predição das classes.

3.4.2 Naive Bayes

O método Naive Bayes recebe tal nome por utilizar o Teorema de Bayes para estimar a $\mathbb{P}(Y = c|\mathbf{x})$ para todas as classes c, como em Rish *et al.* (2001). Para o caso em que as covariáveis sejam consideradas discretas, o Teorema de Bayes é dado por:

$$\mathbb{P}(Y = c|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{X} = \mathbf{x}|Y = c)\mathbb{P}(Y = c)}{\sum_{s\in\mathcal{C}} \mathbb{P}(\mathbf{X} = \mathbf{x}|Y = s)\mathbb{P}(Y = s)}.$$

Izbicki e dos Santos (2020) aponta que conseguimos estimar a $\mathbb{P}(Y = c)$ usando, simplesmente, a proporção amostral de cada classe $c \in \mathcal{C}$. Todavia, necessitamos da

suposição de independência, condicional à classe de Y, entre as covariáveis $(X_1, ..., X_d)$ de modo que:

$$\mathbb{P}(\mathbf{X} = \mathbf{x}|Y = c) = \mathbb{P}((x_1, ..., x_d)|Y = c) = \prod_{j=1}^{d} \mathbb{P}(X_j = x_j|Y = c).$$

Note que, é necessário possuir um modo de estimação para $\mathbb{P}(X = x|Y = c)$, assim, tomamos $X_j|Y = c \sim Discreta(\boldsymbol{\theta}_{j,c})$, sendo $\boldsymbol{\theta}_{j,c}$ um vetor de probabilidades, cuja dimensão é o número de categorias que X_j pode assumir, ou seja, cada probabilidade está associada a uma classe. Para estimar $\boldsymbol{\theta}_{j,c}$, podemos calculá-lo com base nas frequências observadas no conjunto de dados. Nesse sentido, escolher a predição $g(\mathbf{x})$, satisfaz:

$$\begin{aligned}
g(\mathbf{x}) &= \arg\max_{c \in \mathcal{C}} \left(\hat{\mathbb{P}}(Y = c|\mathbf{x}) \right) \\
&= \arg\max_{c \in \mathcal{C}} \left(\hat{\mathbb{P}}(\mathbf{X} = \mathbf{x}|Y = c) \hat{\mathbb{P}}(Y = c) \right) \\
&= \arg\max_{c \in \mathcal{C}} \left(\left(\prod_{k=1}^{d} \hat{\mathbb{P}}(X_k = x_k|Y = c) \right) \hat{\mathbb{P}}(Y = c) \right) \\
&= \arg\max_{c \in \mathcal{C}} \left(\sum_{k=1}^{d} \log \left(\hat{\mathbb{P}}(X_k = x_k|Y = c) \right) + \log \left(\hat{\mathbb{P}}(Y = c) \right) \right).
\end{aligned}$$

Para covariáveis contínuas, temos uma situação análoga. Entretanto, nesse caso, pode-se assumir que $X_j|Y = c \sim Normal(\mu_{j,c}, \sigma^2_{j,c})$, proposto por Murphy *et al.* (2006). Assim, podemos utilizar o estimador de máxima verossimilhança para estimar os parâmetros da distribuição, resultando em $\hat{\mu}_{j,c} = \frac{1}{n_c} \sum_{k \in \mathcal{C}_c} X_{j,k}$ e $\hat{\sigma}^2_{j,c} = \frac{1}{n_c} \sum_{k \in \mathcal{C}_c} (X_{j,k} - \hat{\mu}_{j,c})^2$, onde n_c é a quantidade de observações da classe c e $\mathcal{C}_c = \{j : Y_j = c\}$ é o conjunto das observações da classe c.

3.4.3 Árvore de Classificação

A árvore de classificação é considerada um método extremamente interpretativo. Ela é formada por nós/vértices que se ligam uns aos outros. Cada nó representa uma covariável, porém, o último nó de um ramo representa a classificação de uma observação (um nó que não se conecta a nenhum outro, somente recebe a conexão de outro nó) e comumente é chamado de folha. Quando um vértice B recebe conexão de um vértice A, entendemos que o vértice A é pai do vértice B.

Dito isso, consideramos a árvore de classificação como uma hierarquia (Hansen *et al.*,

2000). Basicamente, para fazer a predição de uma observação começamos do topo, com o nó que não tem pai (nó A). A partir dele, vemos se uma determinada condição é atendida, se for, somos direcionados a um nó B, se não for, vamos ao nó C. Repetimos o mesmo procedimento para os demais nós até que encontre um nó folha. Assim, obtemos a predição.

A Figura 3.3 exibe um exemplo didático de uma árvore de classificação, onde em azul estão as palavras acompanhadas de setas indicando a condição (no caso, verificando se uma palavra existe no documento). Em vermelho e verde estão as folhas, que remetem a predição do documento, classificando-o como sentimento negativo e sentimento positivo, respectivamente.

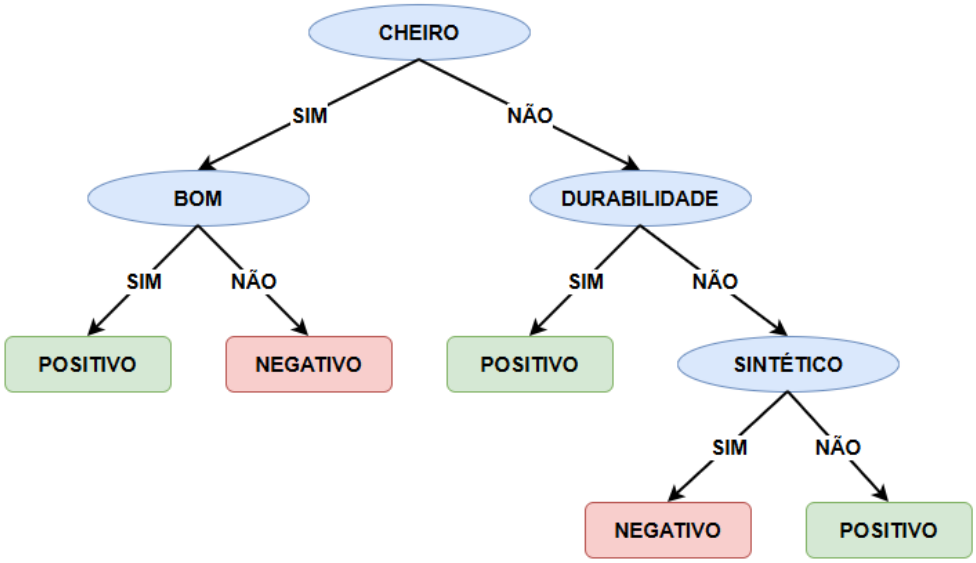


Figura 3.3: Exemplo visual de uma árvore de classificação.

Segundo Izbicki e dos Santos (2020), a construção de uma árvore inicia-se com a decisão das condições, para isso é definida uma partição do espaço das covariáveis, onde essas regiões formadas $(R_1, ..., R_j)$ são distintas e disjuntas, com a finalidade de que tais regiões representem a predição da variável resposta Y e sejam alcançadas através das covariáveis $\mathbf{x}$ e condições. Para que uma região R_k seja representativa em relação às predições de Y, o algoritmo busca que haja a existência de homogeneidade de todos os possíveis Y que pertençam àquela R_k, ou seja, que R_k abrigue observações iguais, como reportado em Morgan (2014). Por esta razão, as regiões são conhecidas como partições puras.

A predição para uma observação y com covariáveis $\mathbf{x}$, a qual está dentro de uma região

R_k, é dada pela moda das observações do conjunto de dados que pertencem à essa região:

$$g(\mathbf{x}) = \text{moda}\{y_i : \mathbf{x}_i \in R_k\}.$$

Buscando definir as regiões puras que melhor se adaptem ao problema, avaliamos o quão razoável é determinada árvore T, já particionada e com os respectivos cálculos de probabilidade (ou proporção) estimada de cada região, com base em seu índice de Gini, $\mathcal{P}(T)$. Assim, para cada região $R_k \in R$ e cada classe $c \in C$, Izbicki e dos Santos (2020) define:

$$\mathcal{P}(T) = \sum_{R_k \in R} \sum_{c \in C} \hat{p}_{R_k,c}(1 - \hat{p}_{R_k,c}).$$

Queremos então minimizar $\mathcal{P}(T)$, visto que quando temos $\hat{p}_{R_k,c}$ muito baixo ou muito alto, situação ideal, a árvore é praticamente pura, já que a maioria das observações são de uma mesma classe.

Outra questão presente que temos que solucionar refere-se a encontrar a árvore T. Existe um custo computacional bem alto caso desejemos testar todas as possibilidades. Com isso, partimos do princípio de que é razoável construirmos uma árvore binária de tal modo que, a árvore comece particionando o espaço das covariáveis em duas divisões (regiões), onde escolherá dentre todas as covariáveis e possíveis condições a melhor combinação de covariáveis e condições que minimize o índice de Gini, formando duas regiões de predição, R_1 e R_2. Assim, fixo a covariável (nó inicial) e a condição escolhida. Em seguida, fazemos o mesmo com as regiões R_1 e R_2, ou seja, para cada região escolhemos dentre todas as covariáveis e condições, a combinação que minimize o índice de Gini da árvore até então construída. O processo se repete enquanto cada folha atingir pelo menos a quantidade mínima, estabelecida previamente, de observações.

3.4.4 Floresta Aleatória

Apesar da alta interpretabilidade das árvores de classificação, elas tendem a apresentar um poder preditivo abaixo dos demais modelos estimados por outros algoritmos. Assim surge a floresta aleatória (Breiman, 2001), com a finalidade de superar esse obstáculo, ela constrói e combina B árvores distintas a fim calcular a predição da observação.

Para a construção da floresta aleatória, inicialmente criamos B amostras *bootstrap* da

amostra original e para cada uma criamos uma árvore $g^b(\mathbf{x})$, resultando em

$$g(\mathbf{x}) = \text{moda}\{g^b(\mathbf{x}), b = 1, ..., B\}.$$

Vale ressaltar que uma amostra bootstrap é uma amostra retirada com reposição do conjunto de dados original, e que possui a mesma quantidade de observações que tal conjunto original.

Para diminuir a correlação entre as diferentes árvores, Géron (2022) diz que são escolhidas aleatoriamente m covariáveis, dentre todas as covariáveis existentes, para o cálculo de $\mathcal{P}(T)$ de um nó na árvore T, com $m = 1, 2, ..., d - 1$, onde d é a quantidade total de covariáveis e, após escolhido, m é um valor fixo para a seleção de cada nó. Em outro nó, novamente são escolhidas aleatoriamente m covariáveis dentre todas as covariáveis existentes. E assim por diante até que a árvore seja construída.

3.4.5 Data-splitting e Overfitting

Dois assuntos intimamente conectados são o data-splitting (divisão de dados) e o overfitting (super-ajuste). Identificamos o overfitting como um problema e o data-splitting como a solução. Imagine que quando montamos vários modelos, escolhemos aquele que acerte o maior número possível de predições. Entretanto, recorde que o modelo escolhido foi baseado nas mesmas observações que medimos o número de acerto. Isso acaba não sendo justo, já que o modelo estudou e buscou predizer corretamente aquele conjunto de dados que foi treinado (conjunto treinamento). Nessa situação, o modelo não se importa com novas observações que gostaríamos de predizer no futuro, já que ele não as conhece e que inclusive podem ser relativamente distintas. Com isso, temos o fenômeno chamado de overfitting, que é quando o modelo se ajusta demais ao conjunto de treinamento, retornando um risco bem baixo e sendo considerado um modelo complexo, expresso em Dietterich (1995). O contrário também pode acontecer. O underfitting (ou sub-ajuste), ocorre quando temos um modelo simplista demais. Acaba não se ajustando bem ao conjunto treinamento e gera, por consequência, uma grande parcela de predições erradas.

Esses são dois problemas que não queremos que ocorra em nossos modelos, pois ao tentar classificar novas observações, possivelmente não teremos um resultado desejado em termos de acertos. Com isso surge uma solução: data-splitting. Data-splitting é um processo de divisão aleatória dos dados, onde dividimos aletoriamente o conjunto de dados

em 3 partes: treinamento, validação e teste. Como o próprio nome sugere, o conjunto de treinamento é aquele que vamos usar para treinar o modelo, é com ele que ajustamos os parâmetros ou estrutura do modelo.

Após treinar vários modelos para cada algoritmo, usamos o conjunto validação para escolher os *tuning parameters* (parâmetros de sintonização) a fim de observar a predição desses novos dados no modelo desenvolvido através de alguma métrica. Esses modelos incluem variações com diferentes combinações de parâmetros, ajustadas com o objetivo de encontrar o melhor balanço entre viés e variância, conforme Izbicki e dos Santos (2020). Dessa maneira, verificamos se o modelo se comporta adequadamente a dados que não conhece. Com a métrica encontrada via dados de validação, decidimos qual é a melhor e escolhemos seu respectivo modelo de determinado algoritmo. Com isso, esperamos que evitemos o overfitting e underfitting, visto que temos um modelo que se adequou às novas observações, possuindo um risco menor do que os demais modelos estimados pelo mesmo algoritmo.

Por último, o conjunto teste é empregado em cada modelo já selecionado pelo conjunto validação para cada algoritmo. Mais uma vez, observamos a predição desses dados de teste nos modelos já selecionados e decidimos qual será o modelo final e a qual algoritmo pertence. O conjunto de teste é importante para que se evite novamente o overfitting, impedindo de tomar providências com base no conjunto validação, que selecionou modelos que também se adaptaram melhor aos seus dados.

Uma variação de data-splitting é a validação cruzada, definida em Berrar (2025). Ela tem o mesmo propósito e essência do data-splitting, mas é realizada de uma maneira um pouco diferente. Com ela temos a junção do conjunto treinamento com o de validação. Tal junção será denominada de conjunto treinamento, e o conjunto teste permanece o mesmo. Nesse procedimento, separamos aleatoriamente o conjunto treinamento em k grupos com observações disjuntas (k-*folds*) e aproximadamente a mesma quantidade de observações. Para treinar o modelo utilizamos todos os grupos de observações com exceção do k-ésimo grupo. Em seguida, utilizamos esse k-ésimo grupo retirado para avaliar o modelo como se fosse um conjunto de validação. Fazemos isso com todos os grupos. Ao final temos k métricas estimadas, calculamos a média delas, e como resultado temos nosso risco estimado. O conjunto de teste é aplicado da mesma maneira que vimos anteriormente.

3.4.6 Medidas de Desempenho de Modelos

É importante calcular diferentes medidas de desempenho de um modelo, além do risco já obtido, porque cada métrica captura um aspecto distinto do comportamento do modelo. Ademais, ao compararmos os modelos entre si, teremos diferentes perspectivas quando observado as medidas. Certo modelo pode ser superior em todas as métricas em relação a outro, como também pode apresentar somente algumas superiores e as restante inferiores. Para o cálculo de algumas métricas, quando a variável resposta é binária, temos a Tabela 3.5. Ela apresenta os erros e acertos das predições realizadas pelo modelo em relação a cada classe. Tais predições são obtidas a partir da estimação da probabilidade $\hat{\mathbb{P}}(Y = 1|\mathbf{x}) \geq K$, sendo $K \in (0, 1)$ considerado como um ponto de corte.

Tabela 3.5: Matriz de confusão.

Real	**Predito**	
	$\hat{Y} = 0$	$\hat{Y} = 1$
$Y = 0$	VN	FP
$Y = 1$	FN	VP

Através da Tabela 3.5, conseguimos formular algumas medidas de desempenho para o modelo e que serão utilizadas e avaliadas posteriormente, veja Izbicki e dos Santos (2020). Em termos de comparação, diferente do risco, desejamos que as métricas a seguir sejam as maiores possíveis, para assim preferir um modelo em relação a outro. As métricas são:

- Acurácia $= \dfrac{VN + VP}{VN + VP + FP + FN}$: proporção de predições corretas;
- Precisão (VPP) $= \dfrac{VP}{VP + FP}$: proporção das predições positivas que realmente são positivas;
- Sensibilidade $= \dfrac{VP}{VP + FN}$: proporção das observações positivas corretamente classificadas como positivas;
- Especificidade $= \dfrac{VN}{VN + FP}$: proporção das observações negativas corretamente classificadas como negativas;
- VPN $= \dfrac{VN}{VN + FN}$: proporção das predições negativas que realmente são negativas;
- F1-Score $= \dfrac{2 \cdot \text{VPP} \cdot \text{Sensibilidade}}{\text{VPP} + \text{Sensibilidade}}$: média harmônica entre Sensibilidade e VPP;

- AUC (Área Sob a Curva ROC): calculada com base na curva ROC, que é construída utilizando Sensibilidade e 1 - Especificidade como eixos, a partir de inúmeros pontos de corte. Analisa-se a AUC da seguinte maneira:
  - AUC $= 1$: modelo é perfeito, sempre classificando corretamente as classes;
  - AUC $= 0.5$: modelo classifica aleatoriamente;
  - AUC < 0.5: modelo é pior que uma classificação aleatória.

3.4.7 Dados desbalanceados

Dados desbalanceados é uma situação comum em muitos casos, ocorrendo quando há uma distribuição desigual entre as classes. Em problemas de classificação, pode ser que uma classe de Y tenha muitos mais observações do que a outra. Isso pode levar a um modelo que tem uma tendência a favorecer a classe majoritária, resultando em um desempenho preditivo ruim, especialmente para a classe minoritária, como relatado em Bermejo *et al.* (2011). Logo, se a quantidade/proporção de determinada classe for pequena dentro do conjunto de dados e as covariáveis não forem suficientemente informativas, a probabilidade de tal classe ocorrer tende a ser menor do que deveria, podendo o modelo errar a predição de Y quando for igual a classe minoritária.

Note que ter dados desbalanceados não é o problema em questão, mas sim o modelo classificar errado devido a probabilidade baixa. Para contornar esse obstáculo, podemos mudar o corte K em classificadores probabilísticos, como sugere Fluss *et al.* (2005). Por padrão, os algoritmos computacionais utilizam o corte de 50% (ou 0.5) para ajustar a classificação de uma observação quando temos 2 classes, ou seja, se $\hat{\mathbb{P}}(Y = 1|\mathbf{x}) \geq 0.5$, então classificamos a observamos como sendo da classe 1 (sentimento positivo).

Agora, o próximo desafio se torna escolher K, tal que $\hat{\mathbb{P}}(Y = 1|\mathbf{x}) \geq K$. Uma forma de encontrar o ponto ótimo de corte é encontrar K que maximize o Índice de Youden (J), conforme Schisterman *et al.* (2005):

$$J = \text{Sensibilidade} + \text{Especificidade} - 1.$$

Portanto para cada corte K, fazemos a classificação das observações através da probabilidade estimada pelo modelo e calculamos a Sensibilidade, Especificidade e J. Por fim, escolhemos K correspondente ao maior J.

Capítulo 4

Resultados

Neste capítulo são colocados em prática todos os conceitos abordados anteriormente, possuindo como diferencial a inserção de instruções e detalhamento de como conduzir, de maneira adequada, a implementação da análise de sentimentos, através de Bag-of-Words, TF-IDF e Word2Vec via modelos de aprendizado de máquina.

Iniciamos com o pré-processamento no banco de dados seguido do processo de divisão de dados, para então aplicarmos os algoritmos LASSO, Naive Bayes, Árvore de Classificação e Floresta Aleatória nas matrizes geradas pelo BoW, TF-IDF e W2V, com a finalidade de obter boas predições e identificar palavras-chave para a classificação de sentimentos. Utilizamos medidas de desempenho para avaliar a performance preditiva dos métodos e gráficos que retratam a importância de algumas covariáveis.

4.1 Pré-processamento

O pré-processamento é uma etapa crucial para lidarmos com textos, pois traduz os documentos em formato de vetores numéricos para que modelos de aprendizado de máquina consigam interpretá-los corretamente. Na prática, o ponto de partida é bem trivial: obter a base de dados que vamos trabalhar. A base de dados das opiniões de usuários sobre produtos de beleza originalmente conta com as variáveis:

- Classificação: avaliação dada pelo usuário ao produto adquirido (notas válidas são 1, 2, 3, 4 e 5);
- Título: título da opinião atribuído pelo usuário (tende a ser mais enxuto);
- Texto: texto da opinião atribuído pelo usuário (tende a ser mais detalhado).

Entretanto, foram necessárias algumas modificações. Tornamos a classificação uma variável binária, ou seja, avaliações iguais a 1, 2 ou 3 atribuímos à classe *negativa* (sentimentos negativos) e as avaliações iguais a 4 ou 5 atribuímos à classe *positiva* (sentimentos positivos). Designou-se a avaliação 3 ao sentimento negativo em busca de diminuir a diferença do desbalanceamento de classes (Figura 4.1), além de, intuitivamente, a nota 3 ser mais associada à experiências negativas do que positivas. Para o *review* completo, fizemos a junção do título e do texto, como se fosse um único documento. Desse modo, temos um documento (opinião do usuário) associado a uma classificação (nota do produto).

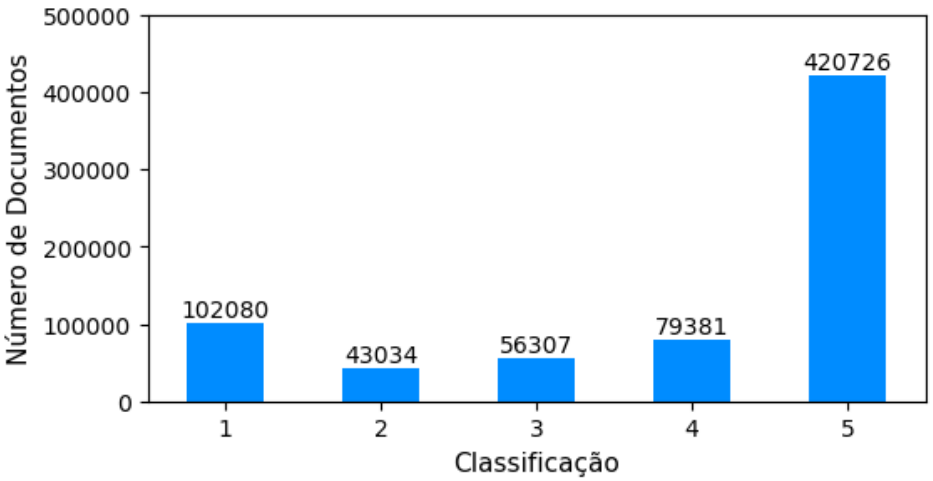


Figura 4.1: Gráfico de barras da distribuição das classificações dos usuários.

Após as devidas alterações, podemos dar início ao pré-processamento de texto. O primeiro passo dado foi fazer a retirada de caracteres não imprimíveis (que não possuem uma representação visível na tela) e de caracteres especiais, como emojis e símbolos, sobrando apenas letras, números e pontuações, além de substituir pontuações estilizadas (outro tipo de formatação) por pontuações padrão.

O idioma inglês predomina no *corpus*, todavia, também temos outros idiomas como o espanhol e o português. Presumimos ser mais interessante trabalhar com uma única língua, por isso, fizemos a tradução de todos os documentos para o inglês, a qual detectava automaticamente outros possíveis idiomas previamente não identificados. A implementação da tradução deve-se ao fato de que queremos capturar o mesmo pensamento expresso pelo autor, consagrando a palavra com uma única representação, afinal, não faz sentido compreender “love” (amor em inglês) e “amor” como palavras diferentes sendo que expressam o mesmo sentimento. Portanto, a barreira linguística deixa de ser um problema.

O próximo passo está relacionado ao processo de retirar todos os dígitos (do zero ao nove) e pontuações (vírgula, ponto final, ponto de exclamação, entre outros), seguido pela realização do *lowercasing* (deixar todas as letras em minúsculas) e da tokenização de todos os documentos. Com os *tokens* definidos, podemos retirar do *corpus* as stop words da língua inglesa (*stopwords.words('english')* da biblioteca *nltk.corpus*), com exceção de "no", "not", "doesn't", "don't", palavras de negação que podem ser importantes para indicação de sentimentos.

Por fim, optamos por aplicar a lemmatização (função *WordNetLemmatizer()* da biblioteca *nltk.stem*) como alternativa ao *stemming*, pelo motivo de ser mais razoável a interpretação de palavras completas em vez de somente os radicais. Ressaltamos que todos os caracteres retirados antes da tokenização, foram substituídos por um espaço, a fim de que não houvesse a possibilidade de junção de palavras vizinhas. Após o pré-processamento, 564 documentos foram excluídos do *corpus* em virtude de não restar nenhum *token* neles, ou seja, estarem nulos, o que impossibilitaria futuras análises.

Todo esse procedimento tende a ser exaustivo computacionalmente, em decorrência da quantidade de documentos. Por este motivo, o uso de paralelismo (múltiplas tarefas executadas ao mesmo tempo por diferentes núcleos de um processador) se tornou essencial para a otimização do tempo neste trabalho.

As Figuras 4.2 e 4.3 exibem a *Word Cloud* (traduzido como nuvem de palavras) do antes e depois do pré-processamento para classificações negativas e positivas, respectivamente, uma representação gráfica das palavras mais frequentes do *corpus*, com o tamanho de cada palavra sendo proporcional à sua frequência de ocorrência, com as palavras mais recorrentes expostas em maior escala. Observe que nas duas figuras houve mudanças quando comparado o antes e depois do pré-processamento, fazendo com que palavras irrelevantes para o entendimento do contexto fossem retiradas. Apesar de ambas as classes terem destaques de nomes de produto de beleza ou parte do corpo, note que depois do pré-processamento a classe negativa possui palavras de desaprovação mais evidenciadas como "money" (dinheiro) e "disappointed" (decepcionado), enquanto a classe positiva conta com o contrário como "love" (amei) ou "great" (ótimo).

a) Word Cloud para sentimentos negativos antes do pré-processamento

b) Word Cloud para sentimentos negativos depois do pré-processamento

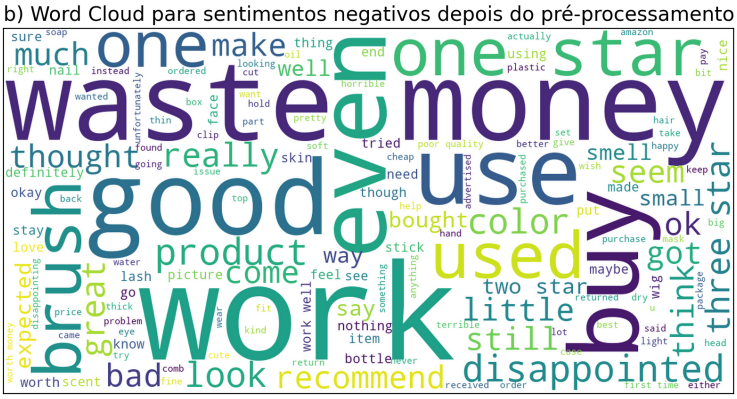
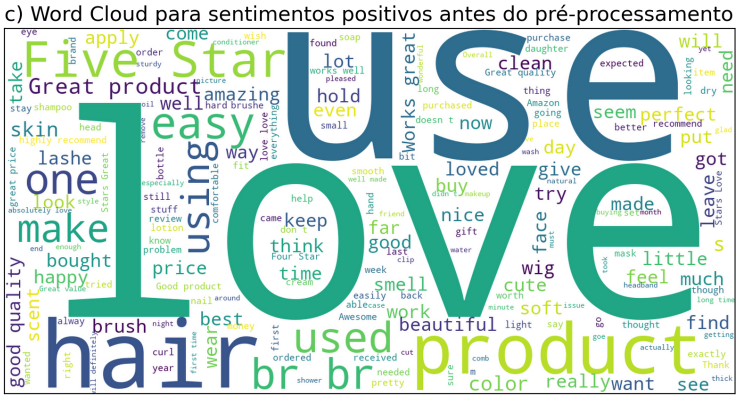


Figura 4.2: Word cloud para a classe de sentimentos negativos: a) antes do pré-processamento; b) depois do pré-processamento.

c) Word Cloud para sentimentos positivos antes do pré-processamento



d) Word Cloud para sentimentos positivos depois do pré-processamento

Figura 4.3: Word cloud para a classe de sentimentos positivos: c) antes do pré-processamento; d) depois do pré-processamento.

4.2 Algoritmos de Aprendizado de Máquina

Recorde que para começarmos o desenvolvimento dos modelos de aprendizado de máquina, temos que realizar a divisão do conjunto de dados, *data-splitting*. Por termos cerca de 700 mil observações, atribuímos 90% delas ao conjunto de treinamento e, consequentemente, 10% ao conjunto de teste. Realizamos validação cruzada com 5 *folds* para todos os algoritmos. Escolhemos usar validação cruzada em vez de *data-splitting* (com 3 conjuntos distintos), devido ao fato de que isso nos proporcionaria mais documentos para o treino do modelo. Perceba que mesmo após a divisão de dados temos a mesma proporção de classes (positivo e negativo) entre os conjuntos sem divisão, treinamento e teste, Figura 4.4. Por último, asseguramos a reprodutibilidade dos resultados, pois todas as etapas que envolvem aleatoriedade utilizam uma semente (valor inicial que define como a aleatoriedade será gerada) definida.

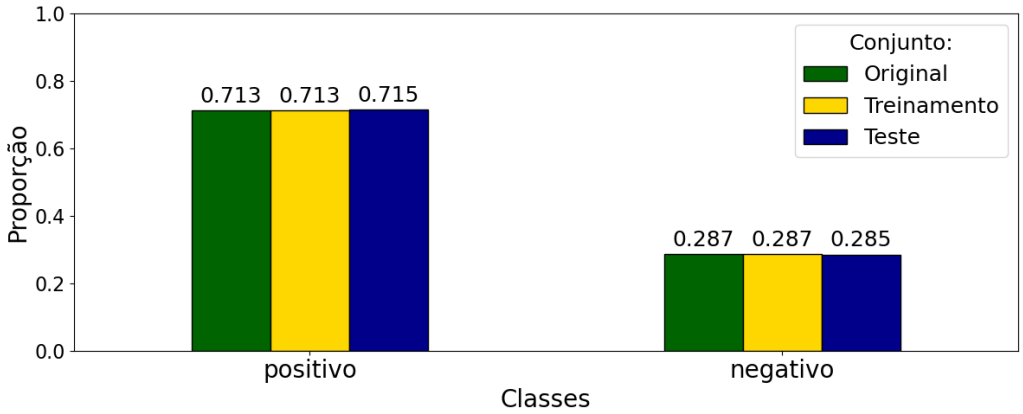


Figura 4.4: Distribuição de classes nos 3 conjuntos de dados.

Após a limpeza de palavras dentro dos documentos, construímos as matrizes esparsas do BoW (*CountVectorizer()* da biblioteca *sklearn.feature_extraction.text*) e TF-IDF (*TfidfVectorizer()* da biblioteca *sklearn.feature_extraction.text*) e a matriz densa do W2V (*Word2Vec()* da biblioteca *gensim.models*) de maneira que *tokens* raros não fossem exibidos, logo, se o *token* não aparecesse em pelo menos 500 documentos, o excluiríamos. Tomamos essa decisão em decorrência de que teríamos cerca de 75 mil covariáveis, sendo que dentre elas haveria palavras com ruídos (abreviações não identificadas, gírias impopulares, entre outros), trazendo inviabilidade computacional e interpretativa para o trabalho. Ao final, foram obtidas 2134 covariáveis (ou palavras distintas).

A Tabela 4.1 exibe algumas estatísticas do conjunto de dados completo sobre a quantidade de palavras por documento antes e depois do pré-processamento, e após a remoção de palavras raras. Com essas medidas, é possível observar o impacto de cada etapa sobre

a distribuição do número de palavras por documentos. Note que, conforme refinamos o processamento das palavras, a média e a variabilidade desses valores diminui consideravelmente. O mesmo acontece com as demais estatísticas, de tal modo que possuímos opiniões sem qualquer palavra. Nesse contexto, temos a Figura 4.5 trazendo o histograma do comportamento da quantidade de palavras por documento, após o descarte dos valores acima do 99º percentil, para melhor visualização. Note que as três distribuições são parecidas, variando somente a escala de valores. Ressaltamos que o descarte dos valores acima do 99º percentil foi realizado apenas na Figura 4.5, e não para a Tabela 4.1.

Tabela 4.1: Medidas descritivas da quantidade de palavras por documento.

	Mín	10%	20%	30%	40%	50%	60%	70%	80%	90%	99%	Máx	Média	DP
Antes do pré-processamento	1	6	9	13	18	23	30	39	52	79	222	2594	36.68	46.89
Depois do pré-processamento	1	4	6	8	10	13	16	20	27	40	110	1559	19.16	23.10
Pós retirada de palavras raras	0	4	5	7	9	11	14	18	24	35	94	1044	16.99	19.45

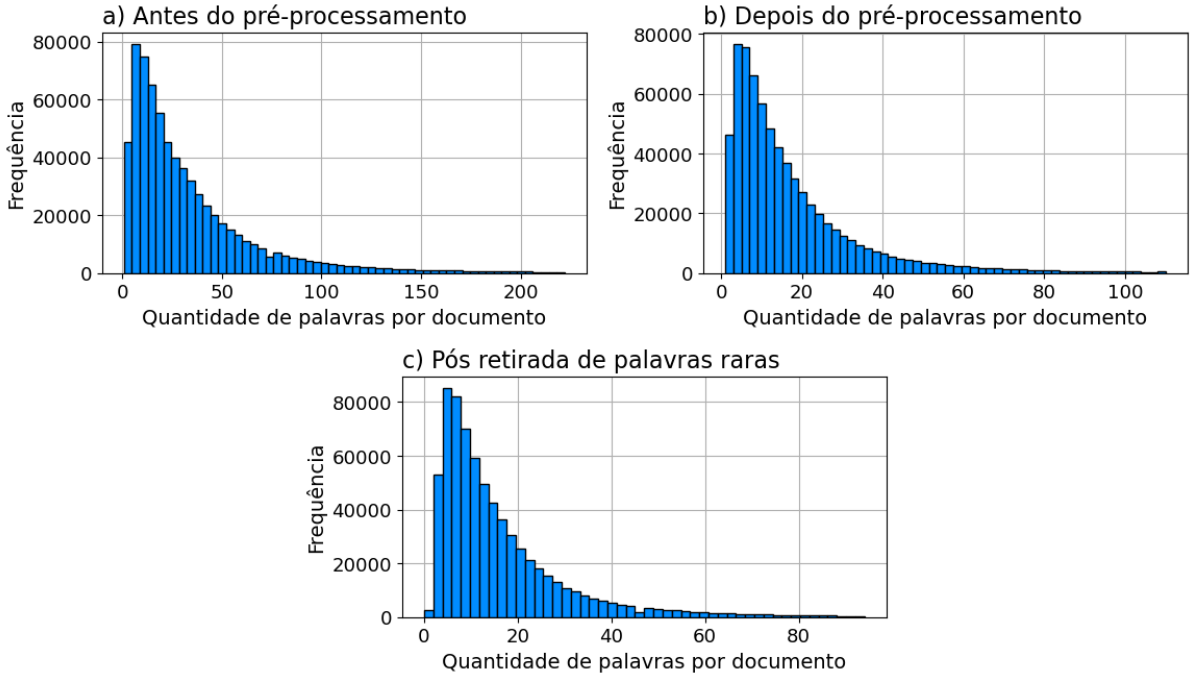


Figura 4.5: Histograma da quantidade de palavras por documento, após o descarte dos valores acima do 99º percentil.

Diferente do BoW e do TF-IDF, o W2V necessita que sejam definidos alguns critérios para a criação dos *embeddings* de cada palavra. O primeiro é o tipo de arquitetura empregada. Embora o Skip-Gram seja mais demorado, ele apresenta melhores resultados em tarefas semânticas, conforme Mikolov *et al.* (2013). Isso é reforçado por Acosta *et al.* (2017) e Muhammad *et al.* (2021), os quais obtiveram melhores resultados em atividades de análise de sentimentos utilizando o Skip-Gram em vez do CBOW. Por isso, seguimos

com o Skip-Gram. Além disso, é preciso que a dimensão do vetor seja definida. Um valor comumente adotado na literatura, e que também será aplicado neste trabalho, é a dimensão 100, como em Lilleberg *et al.* (2015), Zhang e Zhang (2024) e Muhammad *et al.* (2021). Por fim, temos o tamanho da janela de contexto. Ela percorre sequencialmente cada palavra do documento, de modo que, a partir dela que é definida a então palavra alvo e suas palavras vizinhas. Assim, a quantidade de palavras vizinhas, anteriores e posteriores à palavra-alvo, é estabelecida a depender do tamanho da janela, afirma Dharma *et al.* (2022). Por exemplo, se o tamanho da janela for 3, serão consideradas como palavras de contexto (vizinhas) as 3 anteriores e as 3 posteriores à palavra alvo. Nesse caso, usamos o valor 5 como o tamanho da janela, embora essa quantidade varie de literatura para literatura: Acosta *et al.* (2017) usa 10, enquanto Jatnika *et al.* (2019) cita testes com tamanho 3, 5, 6 e 9, por exemplo. Após o treinamento do modelo, como demonstração, trouxemos as palavras mais similares, via distância de cosseno, semanticamente à *amazing* e *horrible*, Tabela 4.2. Veja que palavras positivas estão fortemente associadas à *amazing*, como esperado. Analogamente para as palavras negativas associadas à *horrible*. Além disso, como as dimensões dos vetores representantes de cada documento carecem de uma interpretação, não apresentamos gráficos para interpretabilidade das covariáveis via W2V.

Tabela 4.2: Palavras mais semelhantes semanticamente à *amazing* e *horrible*.

amazing		*horrible*	
Palavra	Similaridade	Palavra	Similaridade
incredible	0.839	terrible	0.943
awesome	0.813	awful	0.918
fantastic	0.811	bad	0.694
fabulous	0.759	worst	0.685
wonderful	0.750	garbage	0.666
love	0.728	trash	0.659
wow	0.703	waste	0.659
obsessed	0.673	poor	0.645
omg	0.656	worse	0.631
great	0.650	unhappy	0.592

É importante ressaltar que a criação das matrizes é um processo posterior ao data-splitting, onde aplicamos a transformação BoW, TF-IDF e W2V apenas no conjunto treinamento e, com base nas covariáveis ou *embeddings* criados por ele, aplicamos a conversão no conjunto de teste, visto que não queremos que haja *Data-Leakage* (vazamento de dados), também chamado de *Pattern Leakage*. Basicamente, o *Data-Leakage* ocorre quando informações do conjunto de teste acabam influenciando o processo de treina-

mento, fazendo com que o modelo pareça mais preciso do que realmente é, já que tem acesso a todos os dados durante o treinamento, se tornando um modelo com dificuldade de generalização, como detalha Apicella *et al.* (2024).

4.2.1 LASSO

Como esperado, o LASSO via método BoW obteve um desempenho computacional ágil, considerando a quantidade de covariáveis. Testamos vários valores para o parâmetro λ, a fim de selecionar, via validação cruzada, o λ que retornasse o menor erro preditivo com base na função *log loss*.

Para melhor visualização, selecionamos apenas os 20 maiores e menores coeficientes estimados, Figura 4.6. No item *a)*, temos as palavras que mais representam a classe negativa, de modo que quanto menor for o coeficiente, mais importante será a palavra para a predição dessa classe. Note que a palavra "meh" (expressão informal que transmite indiferença ou desinteresse) é a que mais contribui para a classe de sentimentos negativos, com ela presente em um texto tendemos a esperar que a opinião do autor seja negativa em relação ao produto. Em contrapartida, no item *b)*, temos as palavras que mais representam a classe positiva, de modo que quanto maior for o coeficiente, mais importante será a palavra para a predição dessa classe. Perceba que "five" (cinco) e "four" (quatro) estão intimamente associados com opiniões positivas sobre produtos de beleza no site da Amazon, representando a nota que atribuíram ao produto, o que é razoável, visto que demonstra uma tendência dos usuários em citar a nota dada (como também evidenciado na *word cloud*, Figura 4.3 *d)*).

Assim como o BoW, o TF-IDF possui o mesmo modo de desenvolvimento do modelo LASSO, alterando somente os resultados. Note que apesar deles serem diferentes, existe uma grande similaridade no top 20 maiores e menores coeficientes de ambas as metodologias, veja as Figuras 4.6 e 4.7. As palavras podem não estar na mesma posição, mas existe um conjunto delas que estão presentes nas duas figuras, indicando que o BoW e o TF-IDF estão trazendo resultados semelhantes com o LASSO, na questão interpretativa.

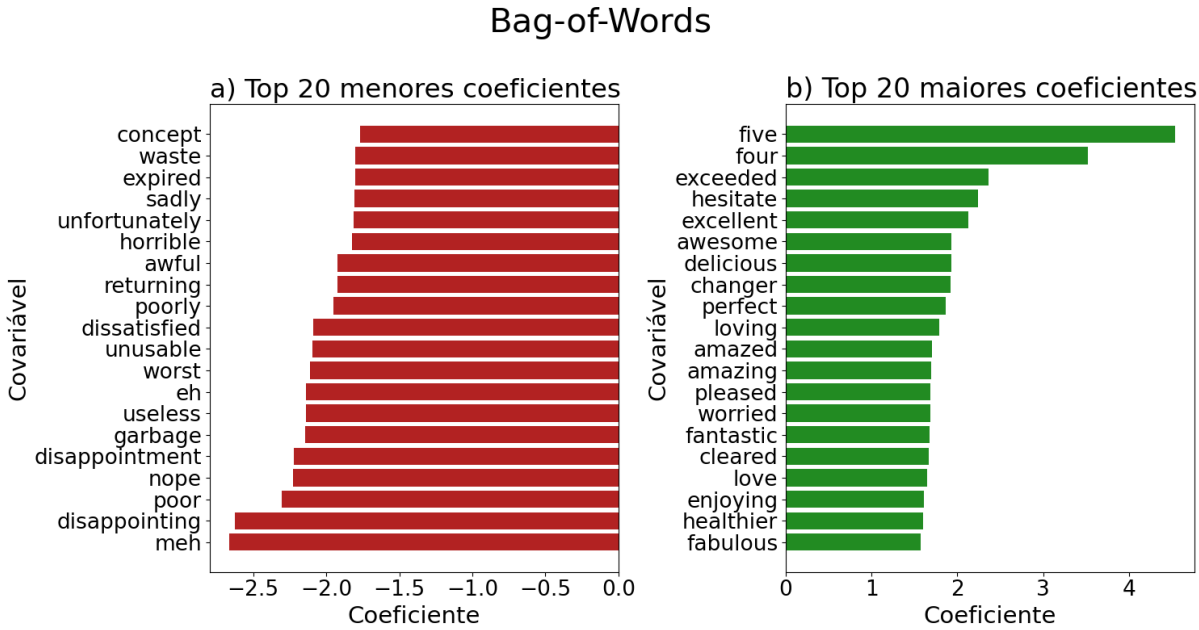


Figura 4.6: Vinte maiores e menores coeficientes estimados pelo LASSO no método BoW.

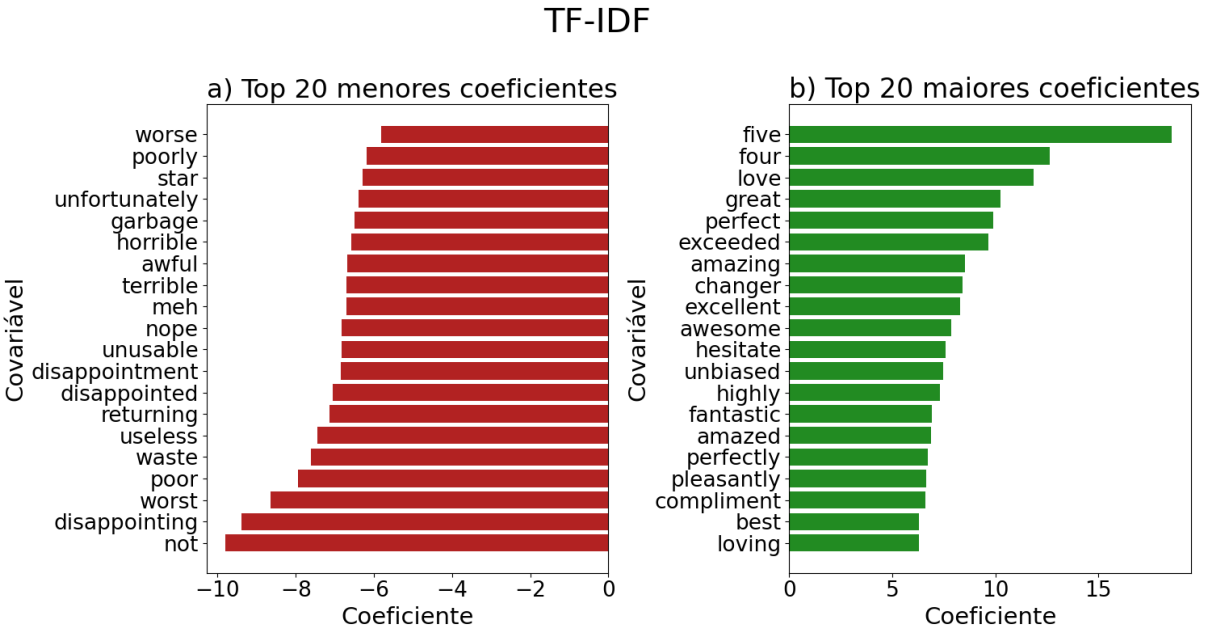


Figura 4.7: Vinte maiores e menores coeficientes estimados pelo LASSO no método TF-IDF.

Se lembre que temos dados desbalanceados, por isso trazemos a Figura 4.8 para exibir a curva ROC juntamente com o ponto de corte ótimo calculado pela maximização do índice de Youden. Assim, *reviews* que possuem probabilidades estimadas acima de 0.683 pelo LASSO no método BoW, são classificados como positivos, caso contrário, são classificados como negativos. Para o TF-IDF e W2V também é necessária a curva ROC, mas dessa vez o ponto ótimo de corte é 0.714 e 0.682, Figuras 4.9 e 4.10, sendo próximos do corte calculado para o Bag-of-Words.

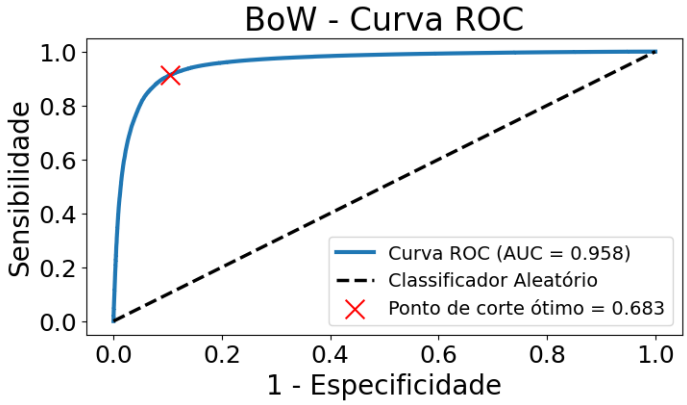


Figura 4.8: Curva ROC estimada pelo LASSO, utilizando o método BoW, no conjunto de treino.

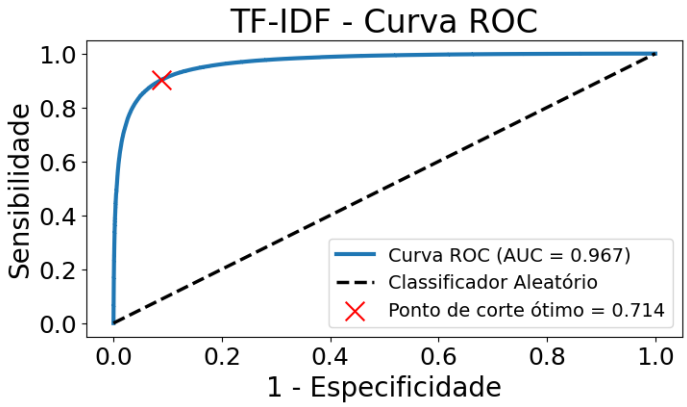


Figura 4.9: Curva ROC estimada pelo LASSO, utilizando o método TF-IDF, no conjunto de treino.

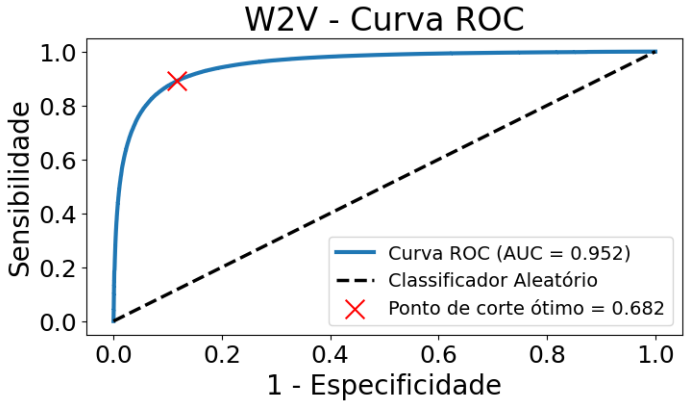


Figura 4.10: Curva ROC estimada pelo LASSO, utilizando o método W2V, no conjunto de treino.

Posteriormente às classificações do conjunto de teste, analisamos o quanto o BoW, o TF-IDF e o W2V acertaram e erraram perante novas observações com o algoritmo LASSO, Tabela 4.3. Observe que os métodos obtiveram desempenho parecido. Enquanto o BoW acertava um pouco mais de opiniões positivas, o TF-IDF acertava um pouco mais de opiniões negativas.

Tabela 4.3: Matriz de confusão do conjunto teste do LASSO (proporção).

Real	**Predito (BoW)**		**Predito (TF-IDF)**		**Predito (W2V)**	
	$\hat{Y}=0$	$\hat{Y}=1$	$\hat{Y}=0$	$\hat{Y}=1$	$\hat{Y}=0$	$\hat{Y}=1$
$Y=0$	17874 (0.894)	2128 (0.106)	18183 (0.909)	1819 (0.091)	17629 (0.882)	2363 (0.118)
$Y=1$	4421 (0.088)	45674 (0.912)	4959 (0.099)	45136 (0.901)	5407 (0.108)	44663 (0.892)

Para facilitar a comparação entre os algoritmos de aprendizado de máquina e os métodos de PLN, foi inserida no final do capítulo uma tabela comparativa contendo as medidas de desempenho de cada combinação.

4.2.2 Naive Bayes

Como puderam perceber, o Naive Bayes não possui um parâmetro para podermos otimizá-lo. Consequentemente, tornou-se neste trabalho o algoritmo mais rápido. Para analisar as palavras mais influentes para ambas as classes, temos as Figuras 4.11 e 4.12 para BoW e TF-IDF, respectivamente. Como na construção do algoritmo usamos apenas probabilidades, uma forma de medir a importância de uma covariável é calculando o logaritmo da probabilidade de ocorrência da covariável X_j nos documentos, condicional à classe c, com $j = 1, 2, ..., 2134$ e c representando a classe positiva ou negativa:

$$\log(\mathbb{P}(X_j|Y = c)).$$

Esses valores são úteis para entender como o modelo está distribuindo as probabilidades de ocorrência das palavras entre as classes. Devido ao fato de que temos muitas covariáveis, a probabilidade de cada uma ocorrer dada uma classe é bem pequena, e por essa razão aplicamos a função logarítmica com o objetivo de aumentar a magnitude do valor, gerando assim números negativos, já que probabilidade possui valor entre 0 e 1. Logo, quanto maior o logaritmo da probabilidade, maior é a probabilidade. Portanto, estamos interessados em analisar os maiores logaritmos da probabilidade condicional a cada classe.

Semelhante ao LASSO, também notamos BoW e TF-IDF apresentando intersecções entre o conjunto das 20 maiores probabilidades de ocorrência de palavras, tanto para a classe positiva quanto para a classe negativa. No entanto, há algumas incongruências interpretativas, como por exemplo no BoW, onde “time” aparece como sendo importante para ambas as classes, ou no TF-IDF, que a palavra “product” é considerada como muito influente nas duas classes.

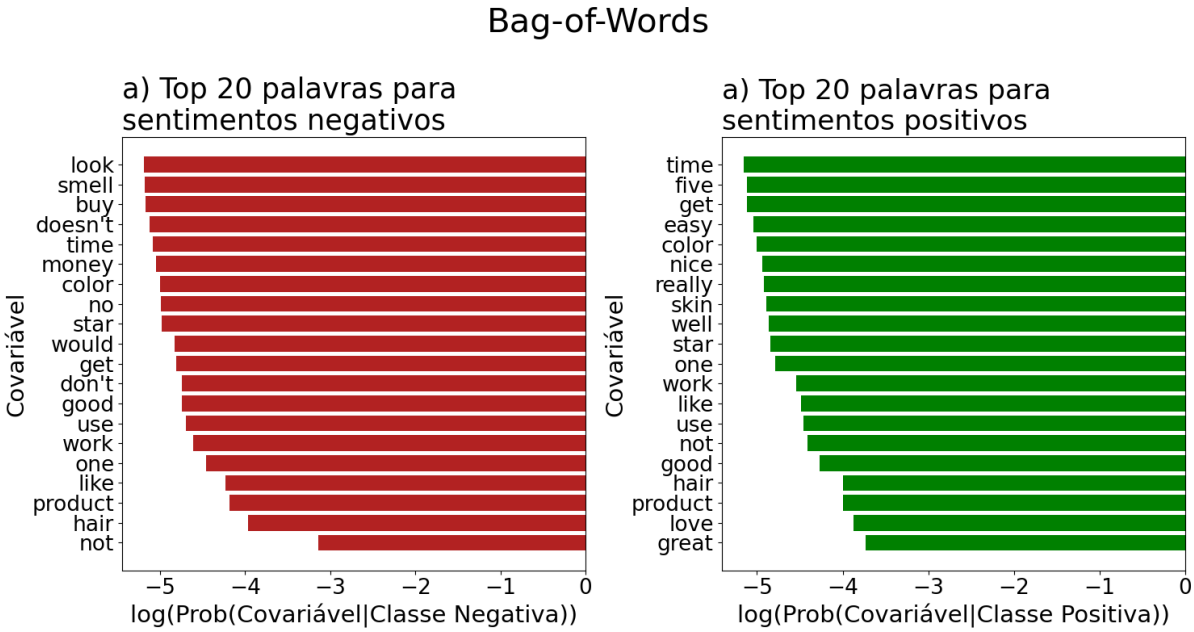


Figura 4.11: As vinte palavras mais importantes estimadas pelo Naive Bayes no método BoW.

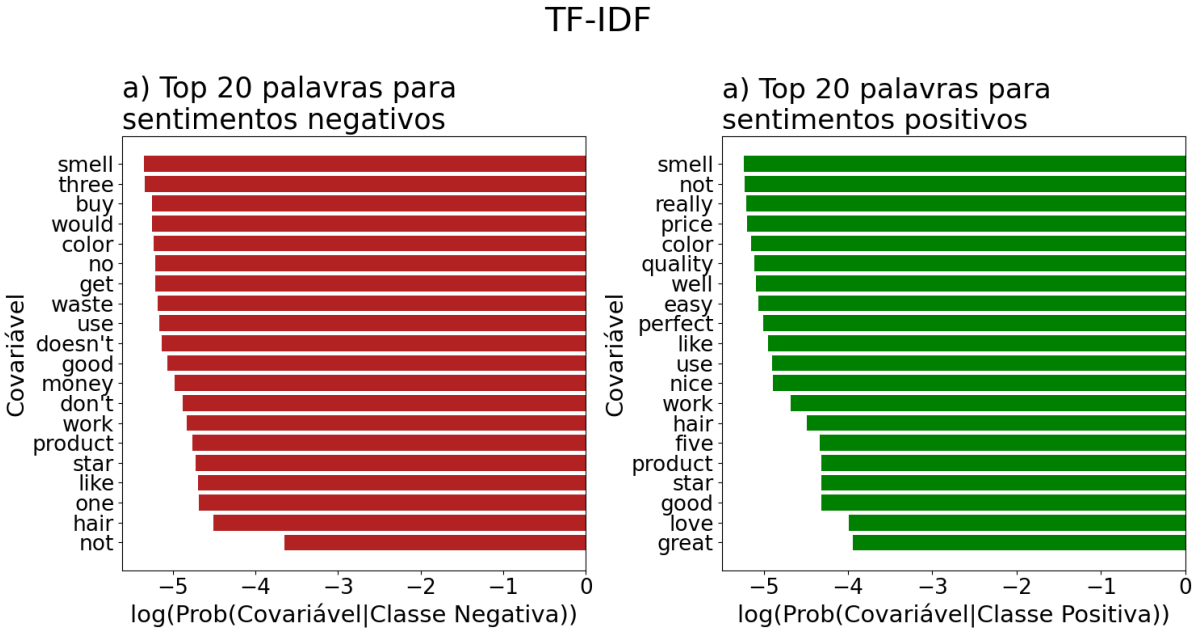


Figura 4.12: As vinte palavras mais importantes estimadas pelo Naive Bayes no método TF-IDF.

Nas Figuras 4.13, 4.14 e 4.15, a curva ROC dos métodos mostram um ponto de corte maior do que os definidos no LASSO para o BoW (0.771) e o TF-IDF (0.721). Note que o ponto de corte para o W2V é 0.099, um valor bem abaixo dos demais. A Figura 4.16 apresenta o histograma das probabilidades estimadas por esse modelo para a classe positiva no conjunto teste, separadas pela verdadeira classe. Observe que a maior parte dos documentos com opiniões verdadeiramente negativas está concentrada entre as probabilidades 0 e 0.1, sendo coerente com o valor 0.099 adotado para determinação das classes, uma vez

que os documentos associados a essas probabilidades são classificados como sentimentos negativos.

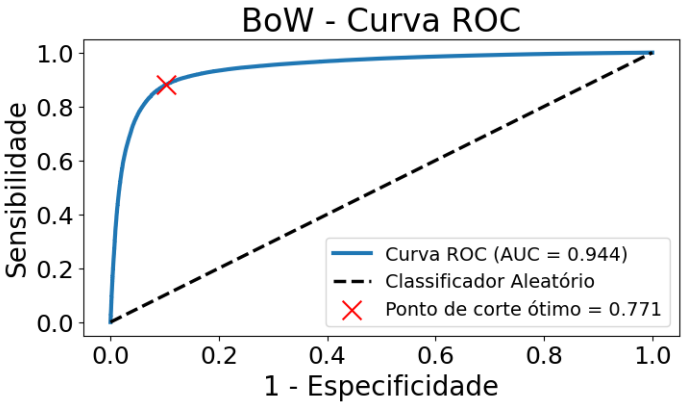


Figura 4.13: Curva ROC estimada pelo Naive Bayes, utilizando o método BoW, no conjunto de treino.

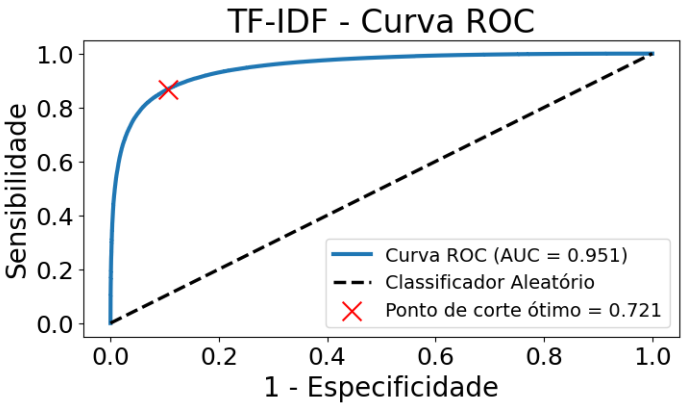


Figura 4.14: Curva ROC estimada pelo Naive Bayes, utilizando o método TF-IDF, no conjunto de treino.

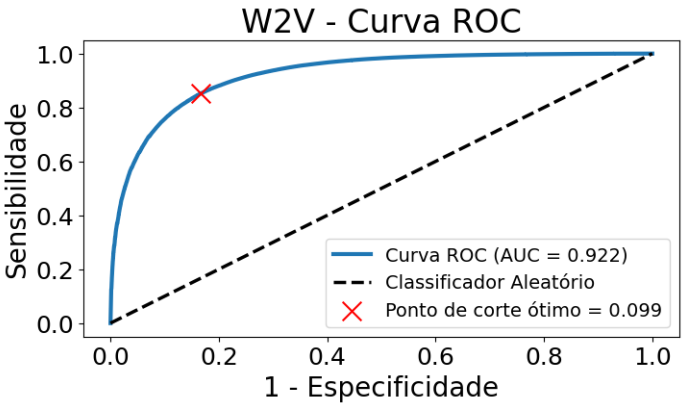


Figura 4.15: Curva ROC estimada pelo Naive Bayes, utilizando o método W2V, no conjunto de treino.

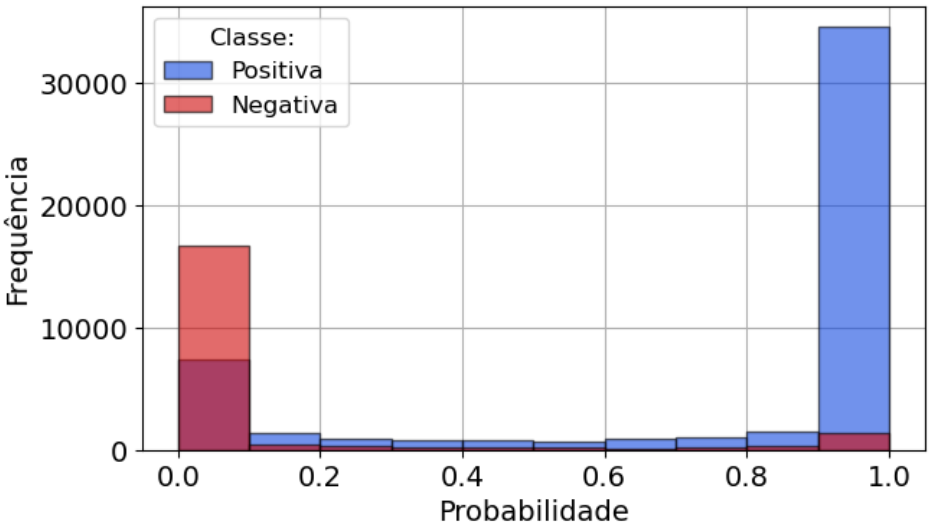


Figura 4.16: Histograma das probabilidades estimadas para a classe positiva pelo modelo Naive Bayes via W2V no conjunto teste, separadas pela verdadeira classe.

Em termos preditivos também tivemos resultados satisfatórios para as metodologias PLN, sendo eles parecidos entre si. Novamente, o BoW leva uma leve vantagem sobre o TF-IDF e W2V quando se trata de predizer a categoria positiva, mas empata na categoria negativa com o TF-IDF, com ambos superando o W2V em número de acertos.

Tabela 4.4: Matriz de confusão do conjunto teste do Naive Bayes (proporção).

Real	**Predito (BoW)**		**Predito (TF-IDF)**		**Predito (W2V)**	
	$\hat{Y}=0$	$\hat{Y}=1$	$\hat{Y}=0$	$\hat{Y}=1$	$\hat{Y}=0$	$\hat{Y}=1$
$Y=0$	17897 (0.895)	2105 (0.105)	17897 (0.895)	2105 (0.105)	16700 (0.835)	3292 (0.165)
$Y=1$	5940 (0.119)	44155 (0.881)	6610 (0.132)	43485 (0.868)	7398 (0.148)	42672 (0.852)

4.2.3 Árvore de Classificação

Já para a Árvore foi necessário a definição de algo novo: um *grid* de parâmetros, que nada mais é do que uma estrutura usada para testar diferentes combinações de valores para múltiplos parâmetros nos algoritmos de aprendizado de máquina, com o objetivo de encontrar a melhor configuração possível com base no risco definido. Desse modo, usamos os seguintes parâmetros para a construção da Árvore:

- *max depth*: limita a capacidade de crescimento da árvore, definindo sua profundidade máxima. Isso impede que a árvore cresça de forma descontrolada, evitando *overfitting*;

- *ccp alpha*: realiza uma espécie de *poda* na árvore, removendo ramos/galhos dela. Quanto maior o valor de ccp alpha, maior será a poda da árvore;

- *min samples leaf*: quantidade mínima de observações em um nó folha, garantindo que as folhas não sejam criadas com poucos dados.

Em BoW e TF-IDF, utilizamos o *grid* de parâmetros de [7, 10, 13, 16, 20] para *max depth* e apenas [5] para *min samples leaf*, de modo que o *ccp alpha* fosse [0.00000386, 0.00000402, 0.0000059, 0.0000083, 0.0000146, 0.000306280] e [0.00000988, 0.00001005, 0.0000259, 0.00006545, 0.00011854, 0.00067785] para cada uma das metodologias, respectivamente. Por conta de termos menos covariáveis para o W2V, testamos o *grid* de [5, 7, 10, 13] para *max depth*, [5] para *min samples leaf* e [0.00000804, 0.00001034, 0.00001372, 0.00001627, 0.00002292, 0.0000297] para o *ccp alpha*. O conjunto de *ccp alpha* é diferente para cada árvore, pois cada árvore cresce de forma distinta com os três conjuntos de dados. Além disso, para termos noção de quais valores selecionar para cada conjunto, estimamos previamente uma árvore e coletamos alguns dos *ccp alpha* fornecidos por ela, assim, preenchemos o *grid* de parâmetros e iniciamos o treinamento do modelo com a validação cruzada. A melhor combinação selecionada para cada método foi:

- BoW: ccp alpha = 0.0000146, max depth = 20 e min samples leaf = 5;
- TF-IDF: ccp alpha = 0.0000259, max depth = 20 e min samples leaf = 5;
- W2V: ccp alpha = 0.00002292, max depth = 13 e min samples leaf = 5.

As Figuras 4.17 e 4.18 trazem as 20 palavras mais importantes para a distinção das classes em cada método na Árvore de Classificação. Essas importâncias indicam o quanto cada palavra contribui para a precisão do modelo, ou seja, o impacto de cada covariável como um nó da árvore. A medida é construída com base na redução de impureza da árvore. Os valores das importâncias são normalizados, e por consequência a soma de todas as importâncias das covariáveis é igual a 1, sendo que cada um varia entre 0 e 1. Logo, quanto mais alto o valor, mais a palavra tem um papel relevante na tomada de decisões do modelo.

Note que o Bag-of-Words e o TF-IDF identificam palavras como covariáveis importantes, que, em outros algoritmos, estavam atribuídas a diferentes classes. Isso ocorre porque essa medida de importância não segrega as palavras por classes, mas as consideram como nós que ajudam a manter a homogeneidade dos dados quando divididos com base nelas, seja para dados preditos como sentimento positivo ou como sentimento negativo. Assim, palavras como “not”, “great” e “broke” se tornam importantes para ajudar a prever o sentimento do autor quando contidas em seu texto.

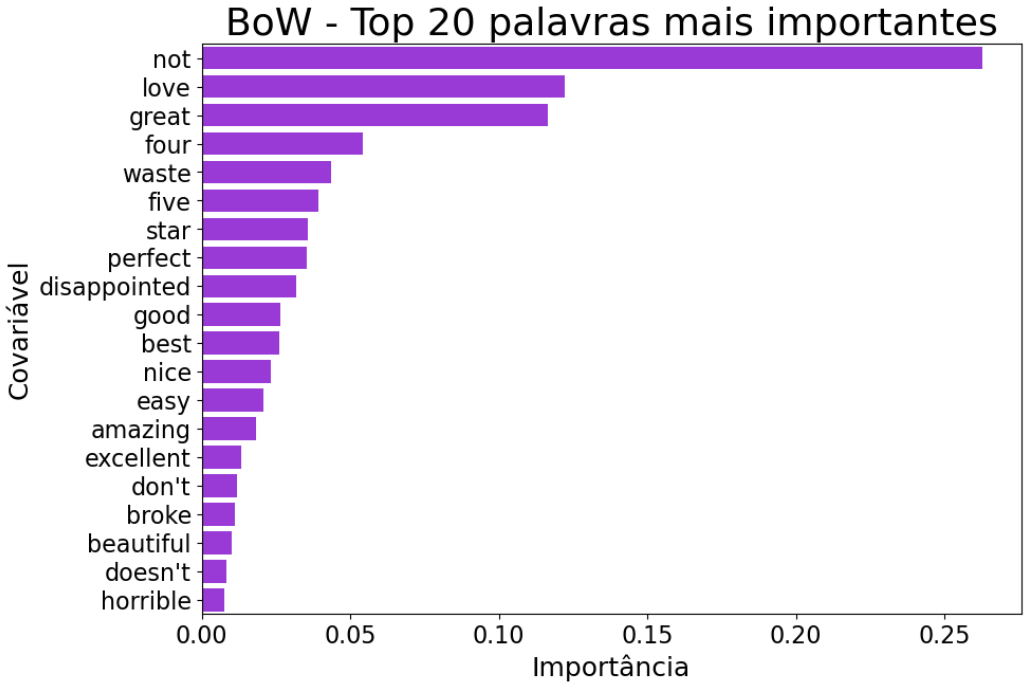


Figura 4.17: As vinte palavras mais importantes estimadas pela Árvore de Classificação no método BoW.

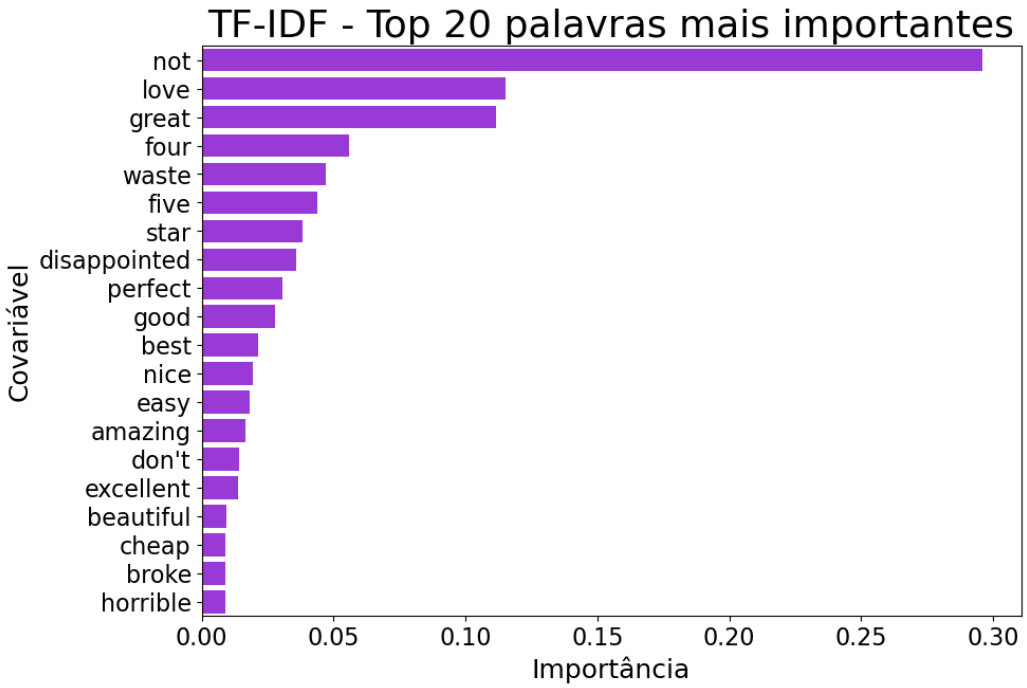


Figura 4.18: As vinte palavras mais importantes estimadas pela Árvore de Classificação no método TF-IDF.

Nas Figuras 4.19, 4.20 e 4.21 temos os cortes próximos e acima de 0.7 nos três métodos.

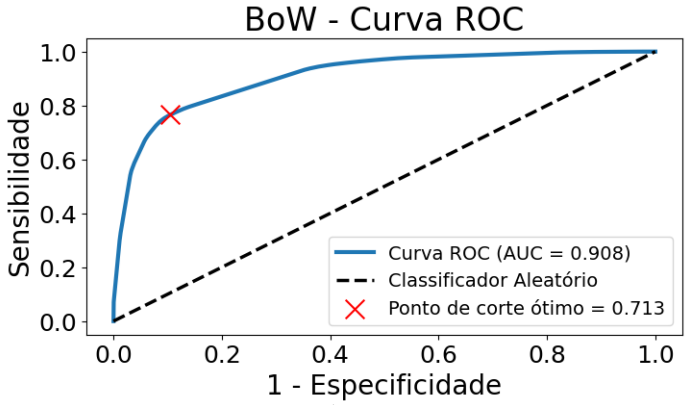


Figura 4.19: Curva ROC estimada pela Árvore de Classificação, utilizando o método BoW, no conjunto de treino.

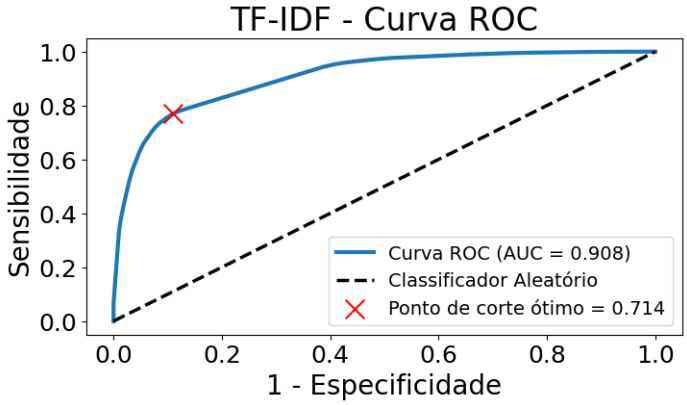


Figura 4.20: Curva ROC estimada pela Árvore de Classificação, utilizando o método TF-IDF, no conjunto de treino.

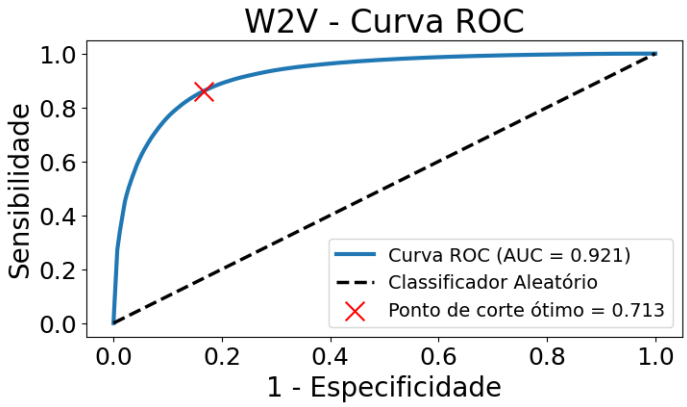


Figura 4.21: Curva ROC estimada pela Árvore de Classificação, utilizando o método W2V, no conjunto de treino.

A Tabela 4.5 apresenta a matriz de confusão calculada. As proporções de valores corretos estão relativamente altas, mas quando comparadas com os demais algoritmos, notamos uma maior dificuldade da Árvore em classificar corretamente a classe positiva ($Y = 1$) nas metodologias BoW e TF-IDF, enquanto o W2V mantém seu nível de desempenho. Por outro lado, o Word2Vec apresenta uma eficiência bem abaixo dos demais

métodos PLN para a classificação correta da classe negativa.

Tabela 4.5: Matriz de confusão do conjunto teste da Árvore de Classificação (proporção).

Real	**Predito (BoW)**		**Predito (TF-IDF)**		**Predito (W2V)**	
	$\hat{Y}=0$	$\hat{Y}=1$	$\hat{Y}=0$	$\hat{Y}=1$	$\hat{Y}=0$	$\hat{Y}=1$
$Y=0$	17855 (0.893)	2147 (0.107)	17769 (0.888)	2233 (0.112)	16247 (0.813)	3745 (0.187)
$Y=1$	11885 (0.237)	38210 (0.763)	11699 (0.234)	38396 (0.766)	7396 (0.148)	42674 (0.852)

Importante registrar que não foi possível trazer a representação visual das árvores estimadas, devido ao elevado número de covariáveis presentes nelas e ao espaço disponível para exibição.

4.2.4 Floresta Aleatória

Assim como a Árvore, a Floresta também necessita de um *grid* de parâmetros. Entretanto, há novas opções:

- *max depth*: limita a capacidade de crescimento da árvore, definindo sua profundidade máxima. Isso impede que a árvore cresça de forma descontrolada, evitando *overfitting*;
- *B estimators*: quantidade de árvores a serem usadas no modelo;
- *max features*: número de covariáveis selecionadas aleatoriamente que são usadas para dividir os nós de cada árvore.

Tanto em BoW quanto em TF-IDF utilizamos o mesmo *grid*: [5, 7, 10] para *max depth*, [100, 250, 500] para *B estimators* e [10, 30, 50] para *max features*. Ambos os métodos selecionaram *max depth* = 10, *B estimators* = 100 e *max features* = 50 e necessitaram de um tempo de desenvolvimento maior do que o LASSO e Naive Bayes. Para o W2V usamos [5, 7] para *max depth*, [100, 250] para *B estimators* e [7, 10, 15] para *max features*. A melhor combinação foi *max depth* = 7, *max features* = 15 e *B estimators* = 100.

Utilizando a medida de importância baseada na redução de impureza das árvores, e com a mesma interpretação das Figuras 4.17 e 4.18, as Figuras 4.22 e 4.23 apresentam muita semelhança entre as palavras selecionadas entre as 20 mais importantes, inclusive havendo uma considerável intersecção entre as palavras dos quatro gráficos, como “easy” e “horrible”, por exemplo.

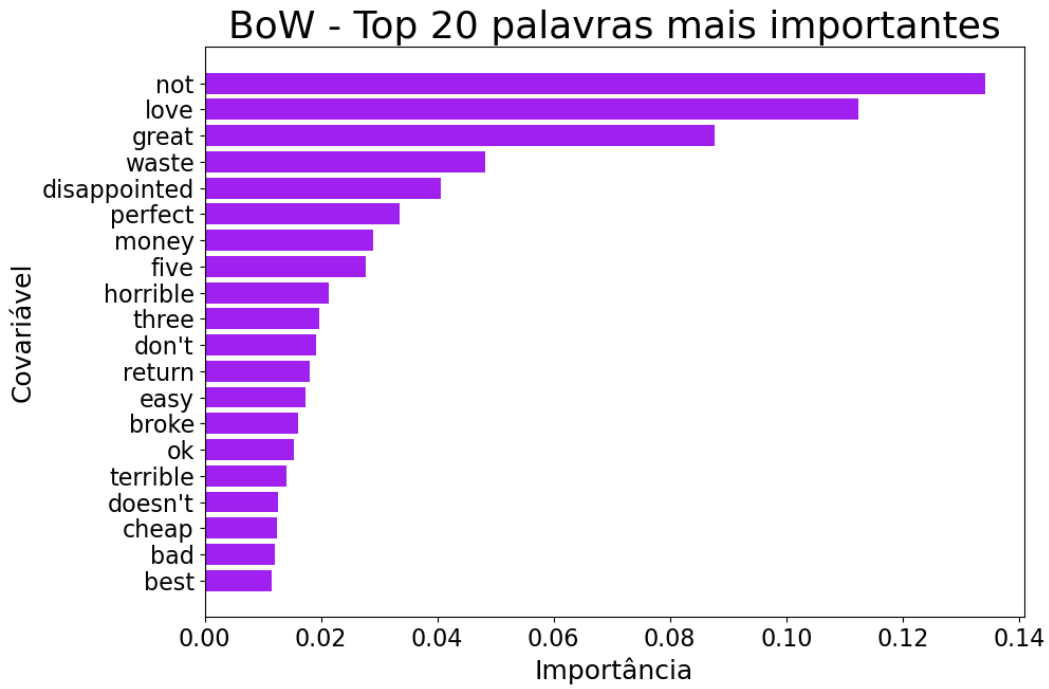


Figura 4.22: As vinte palavras mais importantes estimadas pela Floresta Aleatória no método BoW.

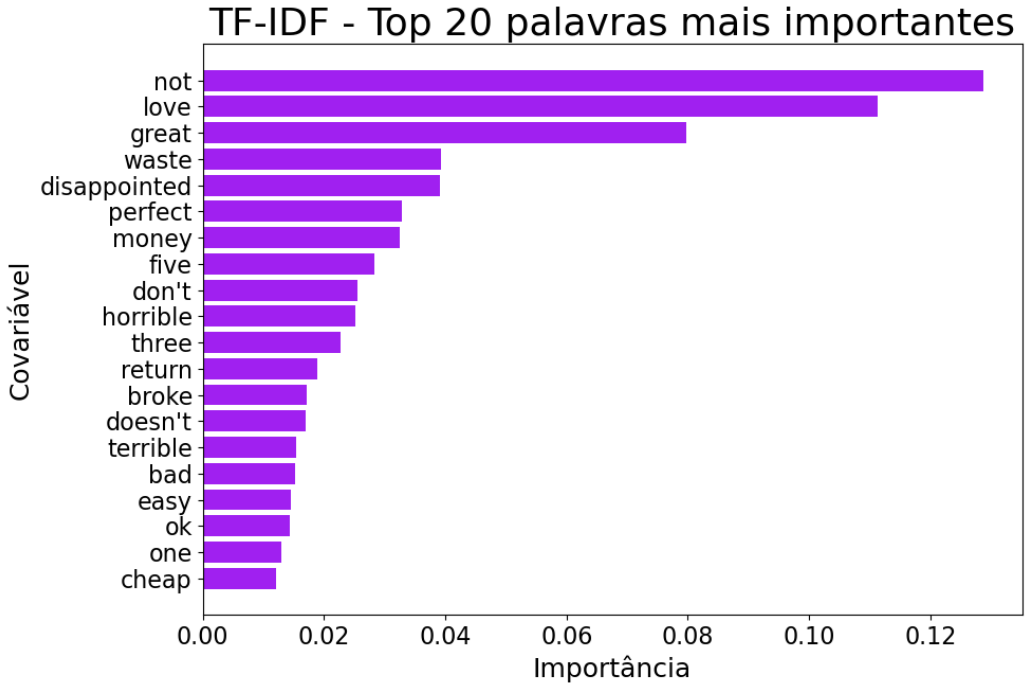


Figura 4.23: As vinte palavras mais importantes estimadas pela Floresta Aleatória no método TF-IDF.

Nas Figuras 4.24, 4.25 e 4.26 temos os cortes próximos de 0.7 em todos os métodos.

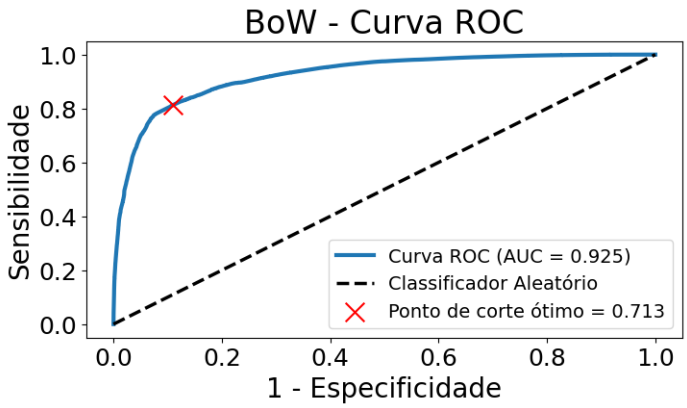


Figura 4.24: Curva ROC estimada pela Floresta Aleatória, utilizando o método BoW, no conjunto de treino.

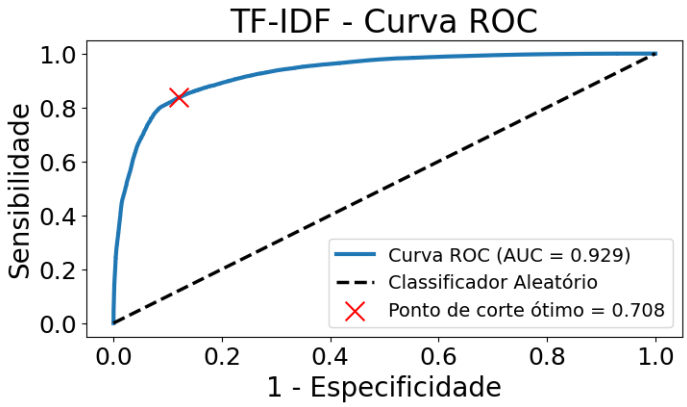


Figura 4.25: Curva ROC estimada pela Floresta Aleatória, utilizando o método TF-IDF, no conjunto de treino.

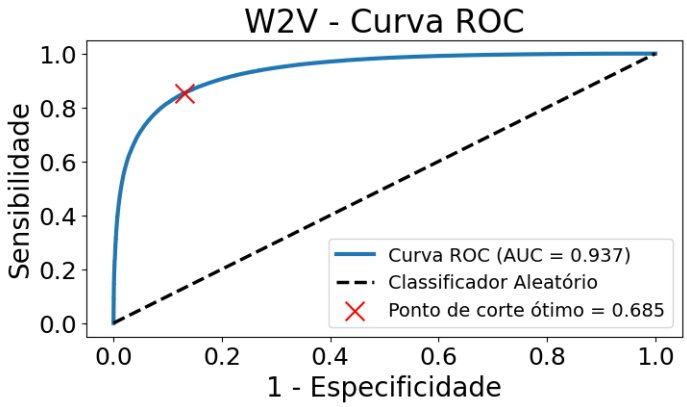


Figura 4.26: Curva ROC estimada pela Floresta Aleatória, utilizando o método W2V, no conjunto de treino.

Novamente, temos uma similaridade entre a Árvore e a Floresta, mas dessa vez entre as Tabelas 4.5 e 4.6, que contam com menores valores proporcionais quando queremos classificar corretamente a classe positiva ($Y = 1$) no BoW e TF-IDF. No geral, houve um aumento de predições corretas nos métodos, quando comparado com a Árvore de Classificação.

Tabela 4.6: Matriz de confusão do conjunto teste da Floresta Aleatória (proporção).

Real	**Predito (BoW)**		**Predito (TF-IDF)**		**Predito (W2V)**	
	$\hat{Y} = 0$	$\hat{Y} = 1$	$\hat{Y} = 0$	$\hat{Y} = 1$	$\hat{Y} = 0$	$\hat{Y} = 1$
$Y = 0$	17809 (0.890)	2193 (0.110)	17531 (0.876)	2471 (0.124)	17311 (0.866)	2681 (0.134)
$Y = 1$	9514 (0.190)	40581 (0.810)	8319 (0.166)	41776 (0.834)	7380 (0.147)	42690 (0.853)

4.2.5 Comparação

Após a implementação de todos os algoritmos, medidas de desempenho foram coletadas para cada um. Através da Tabela 4.7, podemos analisar diferentes perspectivas de comparação entre os doze modelos.

Note que os valores de cada métrica estão bem próximos quando comparados o BoW e o TF-IDF em um único algoritmo. Assim, para ressaltar as maiores distinções entre eles, adotamos o critério de que a diferença nos desempenhos deveria ser igual ou superior a 0.01. No modelo LASSO, a sensibilidade e o VPN usando a representação Bag-of-Words apresentaram melhor desempenho do que em TF-IDF. No entanto, quando analisada a especificidade, o modelo com a transformação TF-IDF se destaca. Já em Naive Bayes, a diferença significativa foi que o VPN e a sensibilidade utilizando a abordagem BoW foi superior (novamente). No caso da Árvore de Classificação, não houve uma discrepância significativa no desempenho entre as diferentes transformações de texto. Por fim, a Floresta Aleatória apresentou melhor eficácia com a especificidade no BoW e com o TF-IDF quando analisados os critérios de acurácia, sensibilidade, VPN e F1-Score.

Observando as métricas dos modelos W2V, percebemos um desempenho inferior em todas elas nos algoritmos LASSO e Naive Bayes, quando comparadas com as medidas do BoW e TF-IDF. Contudo, na Árvore e na Floresta seu comportamento é o contrário: todas as métricas são consideravelmente maiores que as métricas dos outros modelos do respectivo algoritmo, com exceção somente da precisão e da especificidade.

Com isso, é possível observar que, em geral, o LASSO apresenta as medidas mais elevadas e a Árvore possui as mais inferiores em BoW, TF-IDF e W2V. Para o Bag-of-Words e TF-IDF, o Naive Bayes assume o segundo posto de melhor algoritmo, enquanto a Floresta Aleatória fica na terceira posição. Já para o Word2Vec, o segundo e terceiro lugares são invertidos.

Tabela 4.7: Medidas de desempenho dos algoritmos via BoW, TF-IDF e W2V, calculadas no conjunto teste.

MEDIDAS	LASSO			NAIVE BAYES			ÁRVORE			FLORESTA		
	BoW	TF-IDF	W2V	BoW	TF-IDF	W2V	BoW	TF-IDF	W2V	BoW	TF-IDF	W2V
Acurácia	0.907	0.903	0.889	0.885	0.876	0.847	0.800	0.801	0.841	0.833	0.846	0.856
Precisão	0.955	0.961	0.950	0.954	0.954	0.928	0.947	0.945	0.919	0.949	0.944	0.941
Sensibilidade	0.912	0.901	0.892	0.881	0.868	0.852	0.763	0.766	0.852	0.810	0.834	0.853
Especificidade	0.894	0.909	0.882	0.895	0.895	0.835	0.893	0.888	0.813	0.890	0.876	0.866
VPN	0.802	0.786	0.765	0.751	0.730	0.693	0.600	0.603	0.687	0.652	0.678	0.701
F1-Score	0.933	0.930	0.920	0.917	0.909	0.889	0.845	0.846	0.885	0.874	0.886	0.895
AUC	0.956	0.965	0.952	0.944	0.951	0.922	0.906	0.904	0.911	0.922	0.926	0.935

Portanto, em razão das diferenças entre as medidas e considerando que o BoW é mais veloz computacionalmente, elegemos o modelo LASSO via BoW como o mais adequado, tanto em termos preditivos quanto interpretativos, em função de sua rapidez e por apresentar as melhores métricas. Todavia, a AUC é a única medida, dentre as apresentadas, que não depende do ponto de corte escolhido. Se optarmos pela AUC como critério de decisão, seguimos com o LASSO, mas dessa vez com o modelo estimado pelo TF-IDF, já que fornece a maior AUC dentre os doze modelos.

Observe que escolher um ponto de corte é uma tarefa importante em um problema de classificação. Apesar de utilizarmos o Índice de Youden, poderíamos optar por outro critério, o qual mudaria os valores das métricas obtidas. Note que os pontos de corte de todos os modelos ajustados, exceto o Naive Bayes para o Word2Vec, estão consideravelmente próximos da proporção de opiniões positivas (Figura 4.4), sendo assim, 0.713 um possível critério de adoção para a definição do ponto de corte de modelos, embora não empregado nesse trabalho.

Capítulo 5

Considerações Finais

Durante este trabalho, compreendemos os principais conceitos de processamento de linguagem natural, reconhecemos sua importância e exploramos a análise de sentimentos como uma de suas principais áreas de aplicação. Na sequência, entendemos o pré-processamento de textos como sendo uma etapa indispensável para garantir a qualidade de qualquer tipo de análise textual. Após toda a base de conhecimento adquirida, matrizes de dados construídas a partir do Bag-of-Words, TF-IDF e Word2Vec foram empregadas em quatro algoritmos de aprendizado de máquina supervisionado (LASSO, Naive Bayes, Árvore de Classificação e Floresta Aleatória) e coletadas métricas de desempenho de cada modelo estimado, a fim de observarmos o comportamento preditivo deles. Além disso, a interpretabilidade das covariáveis foi analisadas nos modelos associados ao Bag-of-Words e TF-IDF.

Os resultados indicaram que o LASSO proporcionou as medidas de desempenho mais elevadas, sendo então o que melhor se adaptou quanto à predição de novas observações, enquanto a Árvore de Classificação foi considerada como o pior algoritmo, dentre os presentes, para classificação de sentimentos positivos e negativos em opiniões de usuários sobre produtos de beleza. A respeito das metodologias PLN, tivemos BoW e TF-IDF com métricas muito próximas quando comparados em um único algoritmo. Além disso, o W2V somente se destacou positivamente na Árvore de Classificação e na Floresta Aleatória, embora o desempenho não fosse superior ao obtido pelo BoW e TF-IDF no LASSO e Naive Bayes. Portanto, elegemos o modelo LASSO via BoW como o mais indicado quando o foco está na maximização das métricas de desempenho em geral, e o modelo LASSO via TF-IDF quando a preferência se baseia na maior AUC, pois é a única medida, dentre as apresentadas, que não depende do ponto de corte escolhido.

Referente à interpretabilidade, podemos considerar que tivemos alguns consensos entre os modelos, já que forneceram palavras similares ditas como as mais importantes para a predição do sentimento. Em contraste às demais metodologias, o ponto negativo do Word2Vec foi a perda de interpretabilidade das covariáveis, já que as dimensões dos *embeddings* não possuem um significado claro.

Dentre as possíveis e futuras otimizações deste trabalho, incluímos o ajuste de novas covariáveis, a utilização de novos algoritmos e a melhoria dos parâmetros. Novas covariáveis podem ser criadas a partir de mesclas de metodologias. Lilleberg *et al.* (2015) sugere a combinação de TF-IDF e Word2Vec, por exemplo. Também podemos usar novos algoritmos para a classificação de textos, tanto de aprendizado de máquina quanto para a criação de *embeddings*. Por último, é possível que haja uma melhora dos resultados apenas selecionando outros parâmetros para estimar os modelos, tal como no Word2Vec, que poderíamos ter gerado os vetores com mais dimensões, ou até mesmo aumentado o tamanho de sua janela.

Referências Bibliográficas

Acosta, J., Lamaute, N., Luo, M., Finkelstein, E. e Andreea, C. (2017). Sentiment analysis of twitter messages using word2vec. *Proceedings of student-faculty research day, CSIS, Pace University*, **7**, 1–7.

Akuma, S., Lubem, T. e Adom, I. T. (2022). Comparing bag of words and tf-idf with different models for hate speech detection from live tweets. *International Journal of Information Technology*, **14**(7), 3629–3635.

Apicella, A., Isgrò, F. e Prevete, R. (2024). Don't push the button! exploring data leakage risks in machine learning and transfer learning. *arXiv preprint arXiv:2401.13796*.

Bermejo, P., Gámez, J. A. e Puerta, J. M. (2011). Improving the performance of naive bayes multinomial in e-mail foldering by introducing distribution-based balance of datasets. *Expert Systems with Applications*, **38**(3), 2072–2080.

Berrar, D. (2025). Cross-validation. Em *Encyclopedia of Bioinformatics and Computational Biology, 2nd edition*, páginas 638–644. Elsevier.

Bird, S., Klein, E. e Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. "O'Reilly Media, Inc.".

Bollen, J., Mao, H. e Zeng, X. (2011). Twitter mood predicts the stock market. *Journal of computational science*, **2**(1), 1–8.

Breiman, L. (2001). Random forests. *Machine learning*, **45**, 5–32.

Brownlee, J. (2017). Deep learning for natural language processing. *Machine Learning Mystery, Vermont, Australia*, **322**.

Caseli, H. M. e Nunes, M. G. V., editors (2024). *Processamento de Linguagem Natural: Conceitos, Técnicas e Aplicações em Português*. BPLN, second edition.

Chang, C.-Y., Lee, S.-J. e Lai, C.-C. (2017). Weighted word2vec based on the distance of words. Em *2017 International Conference on Machine Learning and Cybernetics (ICMLC)*, volume 2, páginas 563–568. IEEE.

Dharma, E. M., Gaol, F. L., Warnars, H. e Soewito, B. (2022). The accuracy comparison among word2vec, glove, and fasttext towards convolution neural network (cnn) text classification. *J Theor Appl Inf Technol*, **100**(2), 31.

Dietterich, T. (1995). Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR)*, **27**(3), 326–327.

Fluss, R., Faraggi, D. e Reiser, B. (2005). Estimation of the youden index and its associated cutoff point. *Biometrical Journal: Journal of Mathematical Methods in Biosciences*, **47**(4), 458–472.

Géron, A. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc.

Hansen, M. C., DeFries, R. S., Townshend, J. R. e Sohlberg, R. (2000). Global land cover classification at 1 km spatial resolution using a classification tree approach. *International journal of remote sensing*, **21**(6-7), 1331–1364.

Hong, Y. e Skiena, S. (2010). The wisdom of bookies? sentiment analysis vs. the nfl point spread. Em *Proceedings of the International AAAI Conference on Web and Social Media*, volume 4, páginas 251–254.

Indurkhya, N. e Damerau, F. J. (2010). *Handbook of natural language processing*. Chapman and Hall/CRC.

Izbicki, R. e dos Santos, T. M. (2020). *Aprendizado de máquina: uma abordagem estatística*. Rafael Izbicki.

Jatnika, D., Bijaksana, M. A. e Suryani, A. A. (2019). Word2vec model analysis for semantic similarities in english words. *Procedia Computer Science*, **157**, 160–167.

Jurafsky, D. e Martin, J. H. (2024). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Third edition draft edition.

Li, B. e Han, L. (2013). Distance weighted cosine similarity measure for text classification. Em *International conference on intelligent data engineering and automated learning*, páginas 611–618. Springer.

Liao, T. F. (1994). *Interpreting probability models: Logit, probit, and other generalized linear models*. Number 101. Sage.

Lilleberg, J., Zhu, Y. e Zhang, Y. (2015). Support vector machines and word2vec for text classification with semantic features. Em *2015 IEEE 14th international conference on cognitive informatics & cognitive computing (ICCI* CC)*, páginas 136–140. IEEE.

Liu, B. (2022). *Sentiment analysis and opinion mining*. Springer Nature.

Liu, Y., Huang, X., An, A. e Yu, X. (2007). Arsa: a sentiment-aware model for predicting sales performance using blogs. Em *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, páginas 607–614.

Manning, C. e Schutze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

McGlohon, M., Glance, N. e Reiter, Z. (2010). Star quality: Aggregating reviews to rank products and merchants. Em *Proceedings of the International AAAI Conference on Web and Social Media*, volume 4, páginas 114–121.

Meier, L., Van De Geer, S. e Bühlmann, P. (2008). The group lasso for logistic regression. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **70**(1), 53–71.

Mikolov, T., Chen, K., Corrado, G. e Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Morgan, J. (2014). Classification and regression tree analysis. *Boston: Boston University*, **298**.

Muhammad, P. F., Kusumaningrum, R. e Wibowo, A. (2021). Sentiment analysis using word2vec and long short-term memory (lstm) for indonesian hotel reviews. *Procedia Computer Science*, **179**, 728–735.

Murphy, K. P. *et al.* (2006). Naive bayes classifiers. *University of British Columbia*, **18**(60), 1–8.

Paaß, G. e Giesselbach, S. (2023). *Foundation models for natural language processing: Pre-trained language models integrating media*. Springer Nature.

Rish, I. *et al.* (2001). An empirical study of the naive bayes classifier. Em *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, páginas 41–46. Seattle, WA, USA;.

Schisterman, E. F., Perkins, N. J., Liu, A. e Bondell, H. (2005). Optimal cut-point and its corresponding youden index to discriminate individuals using pooled blood samples. *Epidemiology*, **16**(1), 73–81.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **58**(1), 267–288.

Tumasjan, A., Sprenger, T., Sandner, P. e Welpe, I. (2010). Predicting elections with twitter: What 140 characters reveal about political sentiment. Em *Proceedings of the international AAAI conference on web and social media*, volume 4, páginas 178–185.

Wu, T. T., Chen, Y. F., Hastie, T., Sobel, E. e Lange, K. (2009). Genome-wide association analysis by lasso penalized logistic regression. *Bioinformatics*, **25**(6), 714–721.

Zhang, X. e Zhang, L. (2024). The comparison and analysis of skip-gram and cbow in creating financial sentimental dictionary. *Applied and Computational Engineering*, **44**, 56–67.

Zhao, R. e Mao, K. (2017). Fuzzy bag-of-words model for document representation. *IEEE transactions on fuzzy systems*, **26**(2), 794–804.

Apêndice A

Códigos

O trabalho foi implementado utilizando a linguagem de programação Python, contemplando o pré-processamento dos textos, a aplicação de metodologias PLN, a implementação de algoritmos de aprendizado de máquina supervisionado, bem como a avaliação dos modelos por meio de métricas, gráficos e demais análises. Os códigos utilizados se encontram no repositório do *GitHub*: https://github.com/taynaferr/TCC.git.