A blue square with white lines

AI-generated content may be incorrect.

**Os Segredos da Análise de Sentimentos:**

**Um Estudo Aplicado às Lojas Americanas**

Érica Ribeiro[[1]](#footnote-0), Júnior Fernandes Marques[[2]](#footnote-1), Luís Vogel[[3]](#footnote-2), Marlon José Martins[[4]](#footnote-3), Raphael Franco Chaves[[5]](#footnote-4), Thiago Ambiel[[6]](#footnote-5)

ICMC-USP

**RESUMO**

A análise de sentimentos é crucial para entender a percepção do consumidor no e-commerce. Este trabalho investiga as avaliações de consumidores do corpus "B2W-Reviews01.csv" para classificar sentimentos e explorar nuances nas opiniões, visando identificar padrões e tendências relevantes para estratégias de marketing e desenvolvimento de produtos. O estudo se propõe a comparar a eficácia de diferentes abordagens na classificação de sentimentos neste corpus específico. Serão testadas e comparadas três abordagens distintas baseadas em modelos simbólicos, três diferentes algoritmos de aprendizado de máquina supervisionado e um modelo de aprendizado profundo. Espera-se que esta análise comparativa forneça insights sobre as dinâmicas de consumo na era digital, beneficiando empresas e consumidores ao aprofundar a compreensão das opiniões expressas online.

**Sumário**

[**1 INTRODUÇÃO** 3](#_6zt66isgvaue)

[**2 OBJETIVO GERAL** 3](#_vkqkor4tkwxi)

[**2.1 Usuários da Aplicação** 3](#_y6q4t9hlhsyr)

[**2.2 Dados de Entrada** 3](#_z75hj6fiqvl0)

[**2.3 Resultados Fornecidos** 4](#_x3avnyr84pno)

[**2.4 Finalidade** 4](#_9qj9k3qs755h)

[**3 INFORMAÇÕES SOBRE O CORPUS ESCOLHIDO** 5](#_7zspjv1sij19)

[**4 ARQUITETURA DA SOLUÇÃO PROPOSTA NA FASE 1** 5](#_l477wbh8hf6)

[**5 COLETA DE DADOS** 6](#_jxjd4bvj2osj)

[**6 PRÉ-PROCESSAMENTO DOS DADOS** 6](#_ms2gy56kz1o4)

[**6.1 Limpeza dos Dados** 6](#_weo5ia91viqq)

[**6.2 Nuvem de Palavras** 7](#_3b9q0jyhy5cu)

[**6.3 Stop Words** 7](#_kzi5qkv77lxx)

[**6.4 Tokenização** 8](#_1ses1076utwu)

[**7 ANÁLISE EXPLORATÓRIA DOS DADOS** 8](#_ygq1p4r45ywv)

[**8 PROCESSO DE LEMATIZAÇÃO** 10](#_8g16bld6mobh)

[**9 DEFINIÇÃO DA VARIÁVEL DEPENDENTE** 11](#_9s91l6swzbl4)

[**10 AMOSTRA DE DESENVOLVIMENTO** 11](#_ghbquc8tti2b)

[**11 MODELO SIMBÓLICO** 11](#_kwy6g5nu0o8a)

[**11.1 Análise de Performance do Modelo Simbólico** 12](#_hpkvbfabv3h1)

[**11.2 Exemplo de Execução do Modelo Simbólico** 13](#_55vjwqroiw3)

[**12 MODELO SUBSIMBÓLICO** 14](#_4w4plpfouk2a)

[**12.1 Arquitetura da Solução Proposta na Fase 2** 15](#_qfdhytn9vad1)

[**12.2 Modelos de Aprendizado de Máquina** 15](#_i0mn9o72fcyv)

[**12.2.1 Análise de Desbalanceamento da Variável Dependente** 15](#_srh9yclx2mod)

[**12.2.2 Processo de Vetorização** 16](#_kodeova38fwd)

[**12.2.3 Divisão dos Dados** 16](#_2fcunw5iw9nn)

[**12.2.4 Balanceamento de Classes no Conjunto de Treino** 16](#_1azqdivahmhu)

[**12.2.5 Treino dos Modelos de Aprendizado de Máquina Supervisionado** 17](#_vmshkt46uq99)

[**12.2.6 Análise de Performance dos Modelos Supervisionados** 17](#_tq0jsg7dzndx)

[**12.3 Modelo de Aprendizado Profundo** 19](#_3yd9gvong088)

[**12.3.1 Seleção do Modelo Pré-Treinado** 19](#_fefauc4g7tav)

[**12.3.2 Pré-Processamento** 20](#_im4m4ruav40t)

[**12.3.3 Divisão dos Dados** 20](#_vfbym13lwna)

[**12.3.4 Tokenização** 20](#_bh4zc6cgqo1i)

[**12.3.5 Treino do Modelo de Aprendizado Profundo** 21](#_3zt6vwq10d4z)

[**12.3.6 Análise de Performance do Modelo de Aprendizado Profundo** 22](#_gopms4f09y90)

[**12.3.7 Publicação do Modelo** 23](#_i2r5a4vewu7x)

[**12.3.8 Ilustração de Como o Processo Será Executado** 23](#_c2rhy49yrmtd)

[**13 PSEUDOCÓDIGO DO MODELO SIMBÓLICO** 25](#_8qpkpruthvt)

[**16 REFERÊNCIAS BIBLIOGRÁFICAS** 27](#_15poxikv2bxw)

# **1 INTRODUÇÃO**

A análise de sentimentos tem se tornado uma ferramenta essencial para compreender as opiniões e percepções dos consumidores em relação a produtos e serviços, especialmente no contexto do e-commerce. Neste trabalho, será realizada uma investigação aprofundada utilizando o Corpus "B2W-Reviews01.csv", disponibilizado pelas Lojas Americanas. Através da análise das avaliações contidas neste Corpus, buscamos não apenas classificar os sentimentos expressos pelos consumidores, mas também explorar as nuances dessas opiniões, a fim de identificar padrões e tendências que podem informar estratégias de marketing e desenvolvimento de produtos. Este estudo pretende contribuir para uma compreensão mais profunda das dinâmicas de consumo na era digital, revelando insights que podem beneficiar tanto as empresas quanto os consumidores.

# **2 OBJETIVO GERAL**

O objetivo geral do nosso projeto corresponde ao desenvolvimento de uma aplicação automatizada de análise de sentimentos, permitindo uma melhor compreensão das opiniões dos consumidores. A seguir, apresentamos os principais elementos da aplicação, incluindo os tipos de usuários, os dados de entrada requeridos, os resultados fornecidos, bem como a finalidade geral da aplicação.

# **2.1 Usuários da Aplicação**

Os usuários da nossa aplicação são gestores e profissionais de marketing de lojas de varejo, como plataformas de e-commerce ou marketplaces, que desejam compreender as opiniões dos clientes sobre seus produtos. A ferramenta foi projetada para ser simples e acessível, não exigindo conhecimento técnico em computação, permitindo que qualquer profissional utilize os resultados para tomar decisões estratégicas.

# **2.2 Dados de Entrada**

Os dados de entrada que deverão ser fornecidos incluem duas (2) diferentes alternativas, por exemplo:

1. **Análise Individual:** Correspondem as avaliações em formato textual providas pelos consumidores, que poderão ser digitadas num campo textual provido pela aplicação, os quais serão processadas para identificar os sentimentos expressos; e
2. **Análise em Lote:** Possibilidade de importar um arquivo em formato “csv” ou “xlsx” contendo as avaliações em formato textual providas pelos consumidores, para que a aplicação efetue uma análise de forma massificada.

# **2.3 Resultados Fornecidos**

Inicialmente, nosso grupo considerou utilizar o rating de avaliação fornecido pelos consumidores como variável dependente para a análise. Nesse cenário, os ratings 1 e 2 seriam classificados como sentimentos negativos, o rating 3 como sentimento neutro, e os ratings 4 e 5 como sentimentos positivos. No entanto, identificamos diversas divergências na variável "overall\_rating", conforme ilustrado nos exemplos abaixo:

| site\_category\_lv2 | review\_title | review\_text | overall\_rating | recommend\_to\_a\_friend |
| --- | --- | --- | --- | --- |
| Impressora | nao veio cabo usb estou fazendo a reclamaçao | produto não veio com o cabo USB , se não e wifi - tem que vir com o cabo e nada no anuncio especifica que ele não vem | 2 - Regular | No |
| Acessórios para Celular | Bom | A capinha de silicone, é ótima. Aproveitei que ela é incolor, e coloquei fotos nela. Ficou legal. | 2 - Regular | Yes |
| Jardineira | Gostei do produto! | Eu gostei do produto ! da para organizar os armarios . | 3 - Bom | Yes |

Ao analisarmos os exemplos ilustrados acima, ficou evidente que o sentimento do cliente no primeiro caso é negativo, enquanto os dois seguintes são positivos, sem nenhuma indicação de sentimento neutro nos ratings 2 e 3. Ao término desta análise exploratória entre as variáveis “overall\_rating” e “recommend\_to\_a\_friend“, optamos por desenvolver um modelo de análise de sentimentos com classificação binária, categorizando as avaliações como "Positivo" e "Negativo", dado que o principal objetivo da nossa solução é o de apoiar gestores e profissionais de marketing na tomada de decisões estratégicas. A escolha da variável dependente "recommend\_to\_a\_friend" se deve ao seu reflexo direto na intenção do consumidor de recomendar ou não o produto a um amigo. Convertendo essa variável para um formato binário, onde "Yes" representa avaliações positivas e "No" representa avaliações negativas, garantimos uma correspondência clara e prática entre a opinião do consumidor e os resultados do modelo. Adicionalmente, a não inclusão da categoria "Neutro" visa eliminar ambiguidades interpretativas e aumentar a objetividade da análise, tornando os insights mais diretos e acionáveis para os usuários da aplicação. Isso torna a ferramenta mais eficaz ao fornecer informações polarizadas, facilitando a identificação de pontos fortes e fracos percebidos pelos clientes. O resultado final fornecido pela aplicação corresponde a uma classificação do sentimento (positivo ou negativo), acompanhada de uma probabilidade de 0 a 100% que quantifica a intensidade do sentimento detectado.

# **2.4 Finalidade**

Entendemos que tal solução beneficiará tanto as empresas, ao ajustar suas estratégias de marketing e desenvolvimento de produtos, quanto os consumidores, ao melhorar a qualidade dos produtos e serviços oferecidos. Adicionalmente, tal solução será de código aberto, resultando em algumas vantagens significativas, por exemplo: transparência dos códigos, custo-efetividade e personalização.

# **3 INFORMAÇÕES SOBRE O CORPUS ESCOLHIDO**

O Corpus escolhido para o desenvolvimento do presente trabalho corresponde à base de dados e informações “B2W-Reviews01.csv” disponibilizada pelas Lojas Americanas. Tal base de dados e informações encontra-se disponível no Github oficial das Americanas-tech[[7]](#footnote-6), e corresponde a um Corpus aberto de avaliações de produtos ofertados pelas Lojas Americanas na sua plataforma de e-commerce. Este Corpus contempla mais de 130 mil avaliações de clientes, as quais foram coletadas do site Americanas.com entre os meses de janeiro e maio de 2018. Este corpus também contempla 52 diferentes categorias de site de consulta, e mais de 700 diferentes tipos de produtos classificados pelos consumidores. Adicionalmente, este Corpus oferece informações importantes para a análise exploratória de dados contemplando o perfil do avaliador, como gênero, idade e localização geográfica. O Corpus também apresenta duas diferentes taxas de avaliação providas por clientes, dentre as quais se destacam:

1. Uma escala de avaliação entre 1 e 5 pontos, em que: 1 = Ruim, 2 = Regular, 3 = Bom, 4= Ótimo e 5 = Excelente; e
2. Uma pergunta com as alternativas "Sim“ ou “Não" que representa a disposição do cliente em recomendar o produto a outra pessoa.

# **4 ARQUITETURA DA SOLUÇÃO PROPOSTA NA FASE 1**

O diagrama ilustrado abaixo, contempla a arquitetura da solução proposta pelo grupo para a Fase 1:

| A screenshot of a computer  AI-generated content may be incorrect.  Fonte - Elaborado pelos autores. |
| --- |

É possível observar que o diagrama acima contemplou um resumo da arquitetura planejada referente a Fase 2, a qual será explicada em maiores detalhes no capítulo 12.1.

# **5 COLETA DE DADOS**

A coleta de dados corresponde a um processo de extrema importância para o desenvolvimento de qualquer modelo simbólico eficaz. Neste capítulo, detalharemos o processo através do qual os dados foram obtidos, processados e preparados para análise. Com base no que foi discutido no capítulo anterior, utilizamos o repositório oficial da Americanas-tech no Github como a principal fonte de dados. Para garantir a precisão e a eficiência na extração dos dados, foram desenvolvidos códigos específicos na linguagem Python, proporcionando uma coleta automatizada e sistemática. Esses códigos, juntamente com a documentação detalhada, estão disponíveis no material anexo, permitindo a reprodução e a verificação dos métodos utilizados.

# **6 PRÉ-PROCESSAMENTO DOS DADOS**

O pré-processamento dos dados pode ser considerada uma das fases mais importantes no desenvolvimento de modelos preditivos, a qual envolve várias atividades essenciais para garantir a qualidade e a eficácia do modelo. Os capítulos a seguir, detalham as principais etapas efetuadas.

# **6.1 Limpeza dos Dados**

No processo de limpeza dos dados, visamos garantir que as informações utilizadas no desenvolvimento do nosso modelo simbólico estivessem em um formato adequado para análise, sem ruídos que pudessem comprometer os resultados finais da nossa análise. Dentre os processos de limpeza de dados efetuados, destacam-se:

1. Valores Nulos: Primeiramente, removemos todas as observações que continham valores nulos em qualquer uma das colunas. A função dropna() foi utilizada para este propósito, garantindo que apenas registros completos fossem considerados na análise.
2. Dados Duplicados: Em seguida, valores duplicados foram removidos com o objetivo de evitar a redundância de informações. Isso foi feito utilizando a função drop\_duplicates(), mantendo apenas a primeira ocorrência de cada registro duplicado.
3. Padronização: Para padronizar os textos das avaliações dos clientes, todos os caracteres foram convertidos para minúsculas. Isso ajuda a evitar a distinção entre palavras que deveriam ser consideradas iguais, independentemente de estarem em maiúsculas ou minúsculas.
4. Pontuações: Os caracteres de pontuação dos textos das avaliações foram substituídos por espaços. Isso foi feito para garantir que apenas palavras e espaços fossem mantidos, facilitando a análise subsequente.
5. Números: Por fim, todos os números presentes nos textos das avaliações foram removidos. Números geralmente não contribuem para a análise de sentimentos e podem introduzir ruídos nos dados.

# **6.2 Nuvem de Palavras**

A Nuvem de Palavras corresponde a uma representação visual que destaca as palavras mais frequentes em um conjunto de textos, onde o tamanho de cada palavra é proporcional à sua frequência. A imagem a seguir, contribuiu para que o nosso grupo pudesse identificar rapidamente as palavras mais comuns nos comentários dos clientes, proporcionando insights visuais sobre os temas e sentimentos predominantes:

A close up of words

AI-generated content may be incorrect.

Fonte - Elaborado pelos autores.

Para o desenvolvimento da nuvem de palavras ilustrada acima utilizamos a biblioteca WordCloud do Python.

# **6.3 Stop Words**

Em projetos de análise de sentimentos, a etapa de remoção de stop words é uma etapa fundamental do pré-processamento de texto. Via de regra, as stop words correspondem a palavras que, em geral, não agregam significado relevante ao contexto do texto quando analisado em um modelo de análise de sentimentos. No Corpus utilizado para o desenvolvimento deste trabalho foram identificadas 177 diferentes stop words. No processo de remoção de stop words foram consideradas as funcionalidades providas pela biblioteca NLTK. Vale destacar que, embora a maioria das stop words tenha sido removida, palavras com papel semântico relevante como intensificadores (ex: “muito”, “pouco”) e termos de negação (ex: “não”, “nunca”, “jamais”) foram preservadas por desempenharem um papel relevante na polaridade das sentenças. Esses termos são tratados diretamente nas regras do modelo simbólico, como detalhado na seção 11.

# **6.4 Tokenização**

A Tokenização corresponde ao processo de dividir um texto em unidades menores, chamadas de "tokens". Esses tokens podem ser palavras, frases ou até mesmo caracteres, dependendo do nível de tokenização que se deseja aplicar. No nosso trabalho utilizamos a função word\_tokenize da biblioteca NLTK para dividir os comentários providos pelos clientes em palavras individuais. Tal processo permitiu ao nosso grupo manipular e analisar os comentários dos clientes de maneira mais granular, o que é fundamental para extrair informações significativas e relevantes.

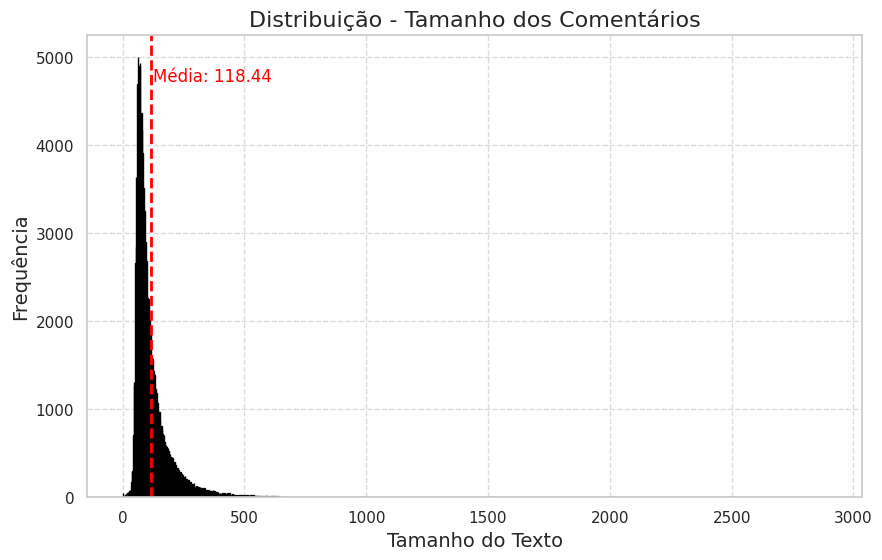
# **7 ANÁLISE EXPLORATÓRIA DOS DADOS**

O intuito dessa etapa é permitir uma compreensão inicial do conjunto de dados disponível. Primeiramente, duas visualizações de dados foram desenvolvidas, as quais nos auxiliaram a entender a distribuição dos ratings atribuídos pelos clientes, bem como as dez palavras utilizadas com maior frequência, conforme ilustrado nos gráficos a seguir:

| A graph with different colored bars  AI-generated content may be incorrect. |  |
| --- | --- |

Fonte - Elaborado pelos autores.

A próxima etapa diz respeito ao desenvolvimento de um histograma contendo a distribuição do comprimento dos comentários após o pré-processamento. Essa visualização nos permitiu identificar como os tamanhos dos comentários variam, fornecendo uma visão clara da densidade e dispersão dos dados. Ao visualizarmos a distribuição dos tamanhos dos comentários, pudemos identificar padrões, como a prevalência de comentários curtos ou longos. Isso pode indicar a natureza das interações dos clientes, como comentários mais curtos talvez indicando feedback rápido e direto, enquanto os mais longos podem conter opiniões mais detalhadas.



Fonte - Elaborado pelos autores.

Adicionalmente, 2 novas visualizações de dados foram desenvolvidas conforme recomendações dos professores, as quais visam ilustrar as recomendações positivas e negativas dos clientes em diferentes estados, bem como no que diz respeito as categorias do site. O gráfico a seguir, ilustra que grande parte das recomendações e classificações está concentrada em 7 diferentes estados brasileiros, sendo São Paulo o responsável pela maioria das classificações:

A graph with numbers and a number on it

AI-generated content may be incorrect.

Fonte - Elaborado pelos autores.

Ao analisarmos tal segmentação no nível de categorias, observa-se uma grande concentração em Celulares e Smartphones, Eletroportáteis, bem como Beleza e Perfumaria, conforme ilustrado no gráfico a seguir:

A graph with different colored squares

AI-generated content may be incorrect.

Fonte - Elaborado pelos autores.

# **8 PROCESSO DE LEMATIZAÇÃO**

A lematização tem como principal objetivo reduzir as palavras em um texto à sua forma base ou raiz, conhecida como *lemma*, enquanto preserva o contexto e a integridade semântica das palavras. Esse processo consiste em identificar termos que efetivamente pertencem à língua, levando em conta sua respectiva classe gramatical (como verbos, substantivos ou adjetivos), e convertê-los para sua forma canônica (ex: "correndo" → "correr"). Diferentemente do *stemming*, que simplesmente trunca as palavras sem considerar significado ou estrutura gramatical, a lematização garante que o resultado seja sempre um termo válido na língua. Em projetos de análise de sentimentos, como o nosso, essa precisão semântica é crucial, pois evita distorções na interpretação dos sentimentos expressos (ex: "ruim" e "ruins" devem ser tratados como a mesma entidade negativa). Para implementar essa etapa, utilizamos a biblioteca spaCy do Python, que aplica modelos de linguagem para identificar a classe gramatical das palavras e extrair seus lemas de forma otimizada.

# **9 DEFINIÇÃO DA VARIÁVEL DEPENDENTE**

No contexto da análise de sentimentos, a variável dependente binária adotada neste modelo é o atributo "recommend\_to\_a\_friend", que classifica as respostas em duas categorias: "Yes" e "No". Para a construção do modelo, essas categorias foram mapeadas para valores numéricos binários, onde "Yes" foi codificado como 1 e "No" como 0. Essa abordagem permite que o modelo identifique e analise a polaridade das opiniões dos usuários, facilitando a interpretação dos resultados. A nova variável 'target' foi então incorporada ao conjunto de dados, possibilitando a realização de análises mais aprofundadas sobre a disposição dos consumidores em recomendar um produto ou serviço a um amigo. A seguir, apresentamos a distribuição desta variável:

|  |
| --- |

Fonte - Elaborado pelos autores.

# **10 AMOSTRA DE DESENVOLVIMENTO**

Essa é uma etapa crucial no processo de construção de modelos de análise de sentimentos, pois define a base sobre a qual o modelo será treinado e testado. Ao término de todas as etapas mencionadas nos capítulos anteriores, dispomos agora de uma base de dados e informações processada e pronta para ser consumida, a qual representa um passo significativo na construção do nosso modelo de análise de sentimentos. Esta base de dados e informações foi utilizada para a construção do modelo simbólico, bem como para a construção do modelo de aprendizado de máquina supervisionado e do modelo de aprendizado profundo referente a fase II.

# **11 MODELO SIMBÓLICO**

O modelo simbólico elaborado pelo nosso grupo combinou regras léxicas e o processamento linguístico para análise de sentimentos. Tal modelo foi desenvolvido com adaptações do VADER para português, inspirando-se no código disponibilizado no repositório do NLTK (Hutto, 2021,<https://github.com/nltk/nltk/blob/develop/nltk/sentiment/vader.py>) e utilizando léxicos especializados obtidos do projeto LeIA (Araujo, 2021,<https://github.com/rafjaa/LeIA>). Destacam-se os seguintes léxicos utilizados:

* **vader\_lexicon\_ptbr.txt**: 7.458 termos com polaridades validadas (ex: "abandono" = -1.9, "amoroso" = +2.3);
* **booster.txt**: Intensificadores como "muito" (+0.293) e "pouco" (-0.293), ajustando a força dos sentimentos;
* **negate.txt**: 60 termos de negação (ex: "nunca", "nem") que invertem polaridades dos termos seguintes;
* **emoji\_utf8\_lexicon\_ptbr.txt**: 3.570 emojis traduzidos (ex: 😊 = "rosto sorridente").

O processamento de um determinado texto pelo modelo simbólico ocorre em uma sequência estruturada de etapas interligadas, conforme ilustrado abaixo:

Primeiramente, o texto é normalizado através da remoção de acentos e pontuação, padronizando a entrada para análise léxica. Em seguida, o algoritmo verifica bigramas para identificar expressões idiomáticas pré-mapeadas (ex: "mó ruim") e termos compostos no léxico, garantindo que combinações específicas não sejam fragmentadas. Paralelamente, intensificadores ("muito", "pouco") são detectados e acumulam valores de *boosting* (B\_INCR / B\_DECR) que ajustam a intensidade do sentimento subsequente. Quando uma negação é identificada (ex: "não", "jamais"), um sinalizador é ativado para inverter a polaridade do próximo termo relevante. Conjunções adversativas, como "mas", disparam um *reset* imediato do contexto, zerando o *score* acumulado para evitar contradições.

Finalmente, a valência total é calculada integrando os efeitos cumulativos de *boosters*, negações e *resets*, com ajustes empíricos para casos onde múltiplas regras interagem (ex: "nunca foi completamente ruim" → negação dupla + *booster*). Essa *pipeline* garante que interações complexas entre palavras sejam modeladas sem dependência de treinamento estatístico.

# **11.1 Análise de Performance do Modelo Simbólico**

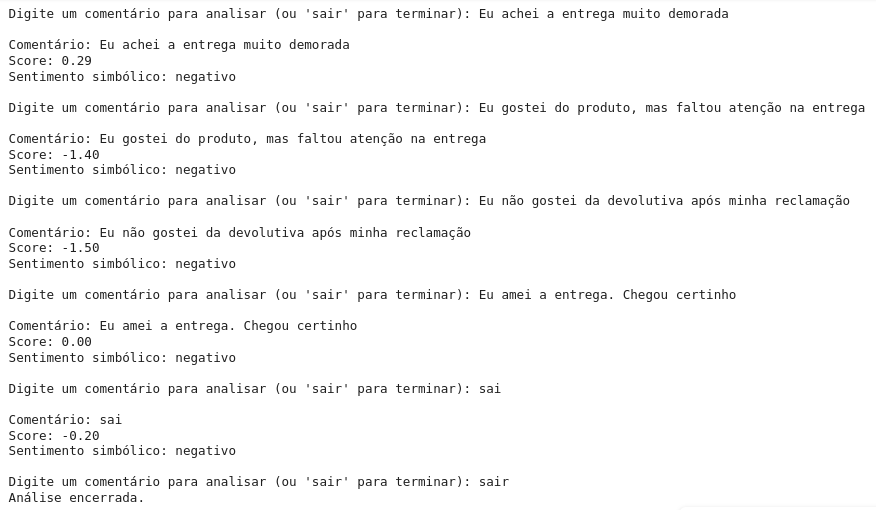
O modelo simbólico baseado no VADER, desenvolvido pelo nosso grupo, alcançou uma acurácia global de 82% em 119.756 avaliações, com F1-score macro de 78%, demonstrando capacidade robusta para classificação de sentimentos em larga escala. Para a classe majoritária, positiva, com 88.997 casos, obteve precisão de 91% e recall de 85%, refletindo eficiência na identificação de padrões claros, como adjetivos positivos, por exemplo, "excelente" e "fantástico", além de expressões idiomáticas mapeadas. No entanto, para a classe minoritária, negativa, com 30.759 casos, as métricas foram menores, com precisão de 61% e recall de 75%, indicando desafios na captura de nuances contextuais. A matriz de confusão revela 75.598 verdadeiros positivos, 13.399 falsos negativos, 7.676 falsos positivos e 23.083 verdadeiros negativos, confirmando o desempenho sólido, mas com erros mais frequentes na classe negativa. A AUC de 0.8787 demonstra boa capacidade de discriminação entre as classes. As principais limitações incluem dificuldades em interpretar textos com linguagem informal, erros gramaticais e vocabulário não padrão, como em avaliações que descrevem "propaganda enganosa", "produto de qualidade inferior" ou "lixo", frequentemente classificadas incorretamente. Além disso, o modelo enfrenta desafios com sarcasmo, ironia e o viés causado pelo desbalanceamento das classes, com proporção aproximada de 3.7:1. Apesar disso, o modelo mostrou-se robusto para aplicações práticas em varejo, apontando caminhos para melhorias futuras, como expansão do léxico para lidar com linguagem coloquial e tratamento de contextos semânticos ambíguos.

|  |  |
| --- | --- |

Fonte - Elaborado pelos autores.

# **11.2 Exemplo de Execução do Modelo Simbólico**

A aplicação do modelo ocorre por meio da classe *SentimentIntensityAnalyzer*, que analisa textos para identificar sentimentos expressos em avaliações de consumidores. A função *analise\_sentimento* processa textos pré-processados, classificando-os como positivos, negativos ou neutros com base no índice de sentimento do VADER. No dataset, a função é aplicada à coluna de avaliações lematizadas, gerando uma classificação automática para milhares de reviews, o que permite aos gestores de varejo entender rapidamente as opiniões dos clientes. Por exemplo, uma avaliação como "preço imbatível e qualidade excelente" é classificada como positiva, enquanto "produto de qualidade inferior, material frágil" é identificada como negativa. Além disso, a função interativa *analisar\_comentario* possibilita que usuários insiram comentários em tempo real, recebendo a classificação instantânea, ideal para análises pontuais. Contudo, o modelo enfrenta dificuldades com textos em linguagem informal ou com erros gramaticais, como em avaliações que mencionam "propaganda enganosa e produto de péssima qualidade", que podem ser incorretamente classificadas devido a vocabulário não padrão ou estruturas confusas. Essas limitações destacam a necessidade de expandir o léxico para lidar com a linguagem coloquial. Ainda assim, a aplicação é eficaz para identificar padrões claros de sentimento, atendendo ao objetivo de fornecer insights valiosos para estratégias de marketing e melhoria de produtos no varejo.



Fonte - Elaborado pelos autores.

# **12 MODELO SUBSIMBÓLICO**

O objetivo deste capítulo é fornecer uma visão detalhada no que diz respeito as propostas de soluções subsimbólicas desenvolvidas pelo nosso grupo. Dentre os fatores que serão abordados no decorrer deste capítulo, destacam-se:

1. a arquitetura da solução;
2. uma breve explicação sobre cada processo da arquitetura;
3. uma breve descrição dos recursos e ferramentas que serão utilizados;
4. uma breve explicação sobre a proposta de treino e teste
5. ilustração de como o processo seria executado;
6. uma análise dos resultados obtidos, comparando a segunda etapa com a abordagem feita na primeira etapa.

Ressaltamos que duas diferentes abordagens de modelos subsimbólicos foram analisadas pelo nosso grupo. Primeiramente, foi efetuada uma análise de 3 diferentes algoritmos de aprendizado de máquina. Por fim, um modelo pré-treinado de redes neurais também foi analisado. As análises e os resultados obtidos serão apresentados no decorrer dos próximos capítulos.

# **12.1 Arquitetura da Solução Proposta na Fase 2**

O diagrama ilustrado abaixo, ilustra a arquitetura da solução proposta pelo grupo para a Fase 2, a qual também contempla a publicação do modelo em ambiente de produção:

A screenshot of a computer

AI-generated content may be incorrect.

Fonte - Elaborado pelos autores.

# **12.2 Modelos de Aprendizado de Máquina**

A análise de sentimentos utilizando modelos de aprendizado de máquina supervisionados envolve diversas etapas cruciais que garantem a eficácia e a precisão do modelo. Dentre essas etapas, destacam-se a análise de desbalanceamento de classes, o processo de vetorização, a divisão dos dados, o balanceamento de classes, o processo de treinamento e teste do modelo, bem como todas as etapas já mencionadas no decorrer da Fase 1. O presente capítulo explora a importância de cada uma das etapas referente a Fase 2, e como elas contribuíram para o sucesso do nosso projeto.

### **12.2.1 Análise de Desbalanceamento da Variável Dependente**

Conforme ilustrado no gráfico abaixo, é possível observar que a classe majoritária (1) representa ~75% do total da distribuição, enquanto a classe minoritária (0) representou 25% desta distribuição:

|  |  |
| --- | --- |

A análise exploratória de desbalanceamento entre as classes ilustrada acima é crucial para projetos de análise de sentimentos com a utilização de modelos de aprendizado de máquina supervisionados. Modelos treinados em dados desbalanceados tendem a ser enviesados em favor da classe majoritária, levando a um desempenho ruim na classe minoritária. Destacamos que algumas técnicas para o tratamento dos dados desbalanceados ilustrados acima foram utilizadas, as quais serão explicadas mais detalhadamente no capítulo.

### **12.2.2 Processo de Vetorização**

Para que os algoritmos de aprendizado de máquina possam processar dados textuais, eles precisam ser convertidos em representações numéricas (vetores). Nesta etapa, os textos lematizados foram tokenizados (divididos em palavras) e usados para treinar um modelo Word2Vec. O modelo Word2Vec foi escolhido pelo grupo dado que o mesmo corresponde a uma técnica de embedding que é capaz de capturar as relações semânticas entre palavras, que entendemos ser crucial em projetos de análise de sentimentos.

### **12.2.3 Divisão dos Dados**

O processo de divisão dos dados em conjuntos de treino e teste corresponde a uma etapa fundamental em qualquer projeto de aprendizado de máquina. Esta prática garante que o modelo seja treinado e avaliado de maneira justa e eficaz. A divisão adequada dos dados ajuda a evitar problemas como o overfitting, onde o modelo aprende muito bem os detalhes e ruídos do conjunto de treino, mas falha em generalizar para novos dados. No presente trabalho, a divisão dos dados em conjuntos de treino e teste foi efetuada utilizando a função “train\_test\_split” da biblioteca “scikit-learn”. A matriz de features (X) e a variável alvo (y) foram divididas com uma proporção de 80% para o treino e 20% para o teste, especificada pelo parâmetro test\_size=0.2. Para garantir a reprodutibilidade dos resultados, o parâmetro random\_state=42 foi utilizado, assegurando que a divisão dos dados seja consistente em diferentes execuções do código.

### **12.2.4 Balanceamento de Classes no Conjunto de Treino**

Em muitos projetos de aprendizado de máquina, especialmente em análise de sentimentos, é comum encontrar conjuntos de dados desbalanceados. Isso significa que uma classe (ou categoria) é muito mais representativa do que a outra. Conforme ilustrado no capítulo 12.2.1, é possível observar à existência de muito mais avaliações positivas do que negativas. Com o propósito de evitar que o nosso modelo de aprendizado de máquina apresentasse previsões enviesadas, com dificuldades em reconhecer e aprender padrões associados à classe minoritária, optamos pelo balanceamento de classes com a utilização da técnica SMOTE (*Synthetic Minority Over-sampling Technique*), a qual corresponde a uma das abordagens mais populares para tratar o desbalanceamento de classes. Através da biblioteca “imblearn”, a técnica de SMOTE foi aplicada exclusivamente ao conjunto de treinamento para gerar novas amostras sintéticas da classe minoritária, igualando sua proporção com a da classe majoritária, conforme ilustrado abaixo:

| **Amostra de Treino Desbalanceada** | **Amostra de Treino Balanceada (SMOTE)** |
| --- | --- |

### **12.2.5 Treino dos Modelos de Aprendizado de Máquina Supervisionado**

Para este projeto, selecionamos três algoritmos de classificação supervisionada amplamente utilizados devido à sua eficácia em diversas tarefas de aprendizado de máquina, os quais encontram-se detalhados a seguir:

1. **Regressão Logística:** A Regressão Logística é um modelo linear simples e eficaz para problemas de classificação binária. Configuramos o modelo com um máximo de 200 iterações e um random\_state=42 para garantir reprodutibilidade.
2. **Random Forest:** O Random Forest é um modelo ensemble que utiliza múltiplas árvores de decisão para melhorar a precisão e reduzir o overfitting. Configuramos o modelo com 100 estimadores e um random\_state=42.
3. **Gradient Boosting:** O Gradient Boosting é outro modelo ensemble que constrói árvores de decisão sequencialmente, corrigindo os erros das árvores anteriores. Configuramos o modelo com 100 estimadores e um random\_state=42.

Cada um dos três modelos ilustrados acima foi treinado utilizando os dados de treinamento rebalanceados (X\_train\_resampled, y\_train\_resampled) visando assegurar que os modelos aprendam de forma justa e equilibrada, sem favorecimentos para a classe majoritária. Para garantir a persistência e reutilização dos modelos treinados, os mesmos foram salvos utilizando a biblioteca joblib. O modelo Word2Vec (model\_w2v) também foi salvo para futuras utilizações.

### **12.2.6 Análise de Performance dos Modelos Supervisionados**

Este capítulo apresenta uma análise detalhada no que diz respeito ao desempenho dos três modelos de aprendizado de máquina supervisionado estudados referente a tarefa de classificação de sentimentos positivos e negativos. Tais modelos foram avaliados com base em métricas padrão de classificação, escores AUC e tempo de processamento. Conforme ilustrado na tabela abaixo, é possível observar que os três modelos demonstraram um desempenho geral robusto e notavelmente similar, alcançando uma acurácia de 0.91. Isso indica que aproximadamente 91% das avaliações no conjunto de teste foram classificadas corretamente por cada um dos modelos.

|  |  |
| --- | --- |
|  |  |

Os valores de F1-score ponderado (weighted avg F1-score) também foram idênticos para os três modelos (0.91), assim como a precisão ponderada (0.92) e o recall ponderado (0.91). Adicionalmente, os três modelos de aprendizado de máquina demonstraram alta capacidade de generalização e resultados competitivos. O Random Forest oferece uma ligeira vantagem na precisão da classe negativa, enquanto a Regressão Logística se sobressai no recall para a mesma classe. O Gradient Boosting, embora muito competente, ficou marginalmente atrás nos F1-scores para a classe negativa e macro average. A Regressão Logística apresentou o maior AUC (0.9589), sugerindo uma capacidade ligeiramente superior de discriminação entre as classes em comparação com o Random Forest (0.9563) e o Gradient Boosting (0.9568). No entanto, as diferenças são mínimas, indicando que todos os três modelos possuem excelente poder de separação. Já com relação ao tempo de processamento, é possível observar que a Regressão Logística demonstrou uma vantagem significativa em eficiência, sendo consideravelmente mais rápida que os outros dois modelos. O Random Forest é substancialmente mais lento, e o Gradient Boosting é o mais custoso computacionalmente dos três.

|  |  |
| --- | --- |

Considerando o equilíbrio entre performance preditiva e custo computacional, a Regressão Logística nos parece ser a escolha mais recomendada entre os modelos de aprendizado de máquina analisados para este problema específico. Tal técnica oferece um desempenho comparável aos modelos baseados em árvores (Random Forest e Gradient Boosting) com uma fração do tempo de treinamento. Se o tempo de processamento não for uma restrição crítica e houver a necessidade de otimizar ao máximo a precisão para a classe Negativa, o Random Forest poderia ser considerado. No entanto, para a maioria das aplicações práticas onde a eficiência é valorizada, a Regressão Logística parece ser a opção mais vantajosa, sem sacrificar significativamente a qualidade da classificação.

# **12.3 Modelo de Aprendizado Profundo**

A análise de sentimentos utilizando modelos de aprendizado profundo, especialmente aqueles baseados em *Transformer*, envolve diversas etapas cruciais que garantem a eficácia e a precisão do modelo. Para o desenvolvimento do nosso trabalho consideramos o modelo pré-treinado de aprendizado profundo conhecido como “*DistilBERT Portuguese Cased*”, com o objetivo de capturamos nuances específicas da língua portuguesa. Tal modelo corresponde a uma versão compacta do BERT, a qual foi otimizada para ser mais rápida e eficiente, sem sacrificar significativamente a precisão. Neste capítulo, serão detalhadas as etapas executadas no decorrer da Fase 2, as quais foram fundamentais para que pudéssemos desenvolver uma solução robusta e eficiente para a tarefa de análise de sentimentos.

### **12.3.1 Seleção do Modelo Pré-Treinado**

A utilização de modelos pré-treinados (*transfer learning*) corresponde a uma prática padrão em projetos de Processamento de Linguagem Natural relacionados a análise de sentimentos. Via de regra, esses modelos pré-treinados já aprenderam representações ricas da linguagem a partir de grandes volumes de texto e podem ser adaptados (*fine-tuned*) para tarefas específicas (como análise de sentimentos) com base de dados menores, resultando em melhor desempenho e treinamento mais rápido do que treinar um modelo do zero. Nosso grupo optou pela escolha de um modelo menor como DistilBERT, que corresponde a uma boa alternativa para ambientes com recursos limitados, como o Colab sem GPU potente. A seguir, destacamos as principais atividades desempenhadas nesta etapa:

1. **Definição do Modelo:** O modelo Transformer pré-treinado a ser utilizado foi definito através do parâmetro “MODEL\_NAME = 'adalbertojunior/distilbert-portuguese-cased'”;
2. **Tokenizador:** Cada modelo Transformer pré-treinado dispõe de um tokenizador específico que foi utilizado durante o seu pré-treinamento. Para carregar o tokenizador associado ao modelo escolhido foi utilizado o parâmetro “tokenizer = AutoTokenizer.from\_pretrained(MODEL\_NAME)”. Destacamos que é essencial utilizar o tokenizador correspondente para que o texto de entrada seja processado da maneira que o modelo espera.
3. **Arquitetura do Modelo:** Nesta etapa a arquitetura do modelo pré-treinado é carregada através do parâmetro “AutoModelForSequenceClassification.from\_pretrained(MODEL\_NAME, num\_labels=2)“. A opção “AutoModelForSequenceClassification” corresponde a uma classe da biblioteca transformers que facilita o uso de modelos pré-treinados para tarefas de classificação de texto. Tal classe já carrega os pesos pré-treinados do corpo do Transformer e inicializa aleatoriamente os pesos da nova camada de classificação. Durante o processo de *fine-tuning*, todos esses pesos, ou uma parte deles, serão ajustados. O parâmetro “num\_labels=2” informa ao modelo pré-treinado de que a tarefa de classificação possui duas classes de saída: Positivo e Negativo. Isso inicializa a camada de classificação com o número correto de neurônios de saída.

### **12.3.2 Pré-Processamento**

É importante destacar que modelos de aprendizado profundo, especialmente os *Transformers*, transferem grande parte do trabalho de engenharia de dados para o próprio modelo, o qual aprende as representações relevantes diretamente dos dados textuais com um mínimo pré-processamento. Um pré-processamento de dados mais robusto, conforme o que foi efetuado para os modelos de aprendizado de máquina apresentados no decorrer do capítulo anterior, poderia ser prejudicial para os modelos de aprendizado profundo, dado que informações relevantes que o modelo de aprendizado profundo utilizaria para entender melhor o contexto, poderiam ser perdidas. Com base no exposto, apenas algumas etapas básicas de pré-processamento de dados foram efetuadas através de expressões regulares. Por fim, é importante destacar que uma das principais funcionalidades dos *Transformers* corresponde a sua capacidade de entender o significado das palavras com base no contexto em que aparecem. A remoção de *“stopwords”* ou a *“lematização”* agressiva poderia remover nuances contextuais importantes.

### **12.3.3 Divisão dos Dados**

O processo de divisão de dados seguiu os mesmos princípios já mencionados no capítulo 12.2.3, na qual a divisão dos dados em conjuntos de treino e teste foi efetuada utilizando a função “train\_test\_split” da biblioteca “scikit-learn”. A matriz de features (X) e a variável alvo (y) foram divididas com uma proporção de 80% para o treino e 20% para o teste, especificada pelo parâmetro test\_size=0.2. Para garantir a reprodutibilidade dos resultados, o parâmetro random\_state=42 foi utilizado, assegurando que a divisão dos dados seja consistente em diferentes execuções do código.

### **12.3.4 Tokenização**

A tokenização corresponde a uma etapa crucial no processamento de texto para modelos de aprendizado profundo, especialmente para aqueles baseados em *Transformers*, como o BERT. Nesta seção, abordaremos como aplicar a tokenização aos textos dos conjuntos de treino e validação, explicando os parâmetros utilizados, bem como a importância desse processo para a análise de sentimentos. Inicialmente, utilizamos um tokenizador previamente carregado para converter os textos dos conjuntos de treino e validação através dos parâmetros abaixo:

* train\_encodings = tokenizer(train\_texts, truncation=True, padding=True, max\_length=512)
* val\_encodings = tokenizer(val\_texts, truncation=True, padding=True, max\_length=512)

Onde:

* **train\_texts / val\_texts**: Correspondem as listas de strings a serem tokenizadas. Estes são os textos dos conjuntos de treino e validação, respectivamente.
* **truncation=True**: Se um texto for mais longo que o comprimento máximo definido (max\_length), ele será truncado. Isso garante que todos os textos se ajustem ao limite de tokens que o modelo pode processar.
* **padding=True**: Se um texto for mais curto que o texto mais longo no batch (ou max\_length se padding='max\_length' for usado), ele será preenchido com tokens especiais de padding. Isso assegura que todas as sequências em um batch tenham o mesmo comprimento, o que é necessário para um processamento em batch eficiente.
* **max\_length=512**: Corresponde ao comprimento máximo da sequência que o modelo pode processar.

### **12.3.5 Treino do Modelo de Aprendizado Profundo**

Para o processo de treinamento do modelo de aprendizado profundo alguns parâmetros chaves foram definidos, dentre os quais destacam-se:

* **output\_dir:** Corresponde ao diretório do google drive utilizado para salvar os checkpoints do modelo, bem como os resultados finais;
* **num\_train\_epochs=1**: Corresponde ao número de vezes que o modelo terá acesso a todo o conjunto de treinamento. Foi definido um valor de 1 dado as limitações de GPU. Tal limite poderia ser revisto e aumentado para 3 ou 6 em estudos futuros visando melhorar o desempenho, entretanto, existira um risco de *overfitting*;
* **per\_device\_train\_batch\_size=8**: Corresponde ao número de amostras de treinamento processadas em cada GPU/CPU por passo de otimização;
* **per\_device\_eval\_batch\_size=16**: Corresponde ao número de amostras de validação processadas em cada GPU/CPU por passo de avaliação;
* **warmup\_steps=500**: Corresponde ao número de passos iniciais de treinamento durante os quais a taxa de aprendizado aumenta linearmente de 0 até seu valor normal. O valor de 500 definido visa estabilizar o treinamento no início;
* **weight\_decay=0.01**: Corresponde ao parâmetro de regularização L2 para prevenir *overfitting*;
* **logging\_dir**: Diretório de saída definido no Google drive para salvar logs para o TensorBoard;
* **logging\_steps=150**: Frequência (em passos) com que as métricas de treinamento são logadas;
* **eval\_strategy="epoch**: Realiza a avaliação no conjunto de validação ao final de cada época;
* **save\_strategy="epoch":** Salva um checkpoint do modelo ao final de cada época;
* **load\_best\_model\_at\_end=True**: Ao final do treinamento, carrega automaticamente os pesos do checkpoint que teve o melhor desempenho na métrica especificada;
* **metric\_for\_best\_model="eval\_loss":** A métrica usada para determinar o "melhor" modelo, no nosso caso, a menor perda de validação;
* **greater\_is\_better=False**: Indica que um valor menor para “metric\_for\_best\_model” é melhor (verdadeiro para perda, falso para acurácia/F1);
* **report\_to="tensorboard"**: Habilita o logging para o TensorBoard, uma ferramenta de visualização.
* **fp16=True**: Tal parâmetro visa habilitar o treinamento com precisão mista (float16). Tal estratégia foi adotada visando acelerar o processo de treinamento, bem como reduzir o uso de memória em GPUs compatíveis, com pouca ou nenhuma perda de acurácia.

Destacamos que o modelo foi treinado por uma única época (Epoch 1/1) dada as limitações em termos de GPU mencionadas em capítulos anteriores, processando 11.976 amostras/passos em aproximadamente 25 minutos e 58 segundos, conforme ilustrado abaixo:

A screenshot of a computer

AI-generated content may be incorrect.

A tabela acima ilustra que a perda de validação (0.179979) é ligeiramente inferior à perda de treinamento (0.199500) ao final da primeira época. Isso pode ser interpretado como um fator positivo dado que sugere que o modelo, até este ponto, está generalizando bem para dados não vistos, e não apresenta sinais imediatos de overfitting significativo após esta única época.

### **12.3.6 Análise de Performance do Modelo de Aprendizado Profundo**

Neste capítulo, será detalhado o desempenho do modelo de aprendizado profundo treinado para a tarefa de classificação de sentimentos. A análise de performance deste modelo foi baseada nas métricas obtidas durante e após o processo de treinamento, incluindo perdas (loss), métricas de classificação padrão e a Área Sob a Curva ROC (AUC). A seguir, são apresentamos os resultados obtidos:

|  |  |
| --- | --- |

Conforme ilustrado na tabela acima, o modelo apresentou as seguintes métricas de classificação:

* Accur**acy**: Este valor indica que o modelo classificou corretamente aproximadamente 94.6643% das amostras no conjunto de validação.
* **Precision**: Este valor de precisão de 96.7198% indica que, quando o modelo previu uma amostra como "Positiva", ele estava correto em 96.7198% das vezes.
* **Recall**: Também para a classe positiva, o recall de 96.0787% indica que o modelo foi capaz de identificar corretamente 96.0787% de todas as amostras verdadeiramente "Positivas" no conjunto de validação.
* **F1-Score**: O F1-Score pode ser descrito como a média harmônica da precisão e do recall. O valor de 96.3982% indica um excelente equilíbrio entre precision e recall, indicando que o modelo tem bom desempenho em ambas as frentes, sem sacrificar excessivamente uma pela outra.

As métricas de validação apresentadas acima são bastante elevadas, sugerindo que o modelo de aprendizado profundo apresenta uma forte capacidade preditiva no conjunto de dados de validação, mesmo após apenas uma única época de treinamento. Adicionalmente, os valores de AUC de 0.97 apresentados tanto para o conjunto de treino quanto para o de teste são excelentes, indicando um alto poder discriminatório do modelo de aprendizado profundo. Destacamos que a proximidade e igualdade dos valores de AUC de treino e teste sinalizam que o modelo está generalizando bem para dados completamente novos, que correspondem ao conjunto de teste., sugerindo que o *overfitting* não é uma preocupação significativa com base nesta métrica.

### **12.3.7 Publicação do Modelo**

A próxima etapa consiste na publicação do modelo de aprendizado profundo em um ambiente produtivo, com o principal propósito de permitir aos usuários finais interagir com tal solução, que stakeholders visualizem seu valor, e que tal modelo seja integrado a fluxos de trabalho mais amplos. Para esta finalidade, o nosso grupo optou pela utilização do pacote *“Streamlit”* do Python como plataforma de publicação, dado que a tal pacote é capaz de aliar simplicidade, rapidez e poder de interatividade para os usuários finais da aplicação.

### **12.3.8 Ilustração de Como o Processo Será Executado**

As imagens apresentadas a seguir, visam ilustrar a execução do processo de análise de sentimentos em ambiente de produção, as quais contemplam as principais funcionalidades da solução desenvolvida pelo nosso grupo:

### **Raphael Irá Inserir Telas da Aplicação!**

# **13 PSEUDOCÓDIGO DO MODELO SIMBÓLICO**

**CONSTANTES**:

  AUMENTO\_BOOSTER ← +0.293

  DIMINUI\_BOOSTER ← –0.293

  CONJUNCOES\_ADVERSATIVAS ← {“mas”, “porém”, “contudo”, “entretanto”, “todavia”}

  CASOS\_ESPECIAIS ← {

“top”: +2, “daora”: +1,

“massa”: +1,

“zoado”: –2, “mó ruim”: –2,

“de boa”: +1, “tenso”: –1, “imbatível”: +2.5

}

**MAPAS PRÉ-CARREGADOS**:

  LEXICO ← mapa de (palavra ou bigrama) → valência

  DICIONARIO\_BOOSTER ← mapa de palavra → (AUMENTO\_BOOSTER ou DIMINUI\_BOOSTER)

  CONJUNTO\_NEGACOES ← conjunto de palavras de negação

**FUNÇÃO** *ajustar\_valencia*(valencia\_base, acumulador\_booster, sinalizar\_negacao):

// Inverte boost se valência base for negativa

se valencia\_base < 0 então

acumulador\_booster ← –acumulador\_booster

fim se

// Aplica boost e zera acumulador

valencia\_tratada ← valencia\_base + acumulador\_booster

acumulador\_booster ← 0

// Inverte sinal se havia negação pendente

se sinalizar\_negacao então

valencia\_tratada ← –valencia\_tratada

sinalizar\_negacao ← FALSO

fim se

retorna (valencia\_tratada, acumulador\_booster, sinalizar\_negacao)

**FIM FUNÇÃO**

**FUNÇÃO** *calcular\_sentimento\_sentenca*(texto\_entrada):

// 1. Normalização

texto\_normalizado ← remover\_acentos\_e\_pontuacao(texto\_entrada)

lista\_tokens ← dividir\_por\_espaco(texto\_normalizado)

// 2. Estado inicial

pontuacao\_total ← 0

acumulador\_booster ← 0

sinalizar\_negacao ← FALSO

ultima\_valencia ← 0

// 3. Iteração sobre tokens

i ← 0

enquanto i < tamanho(lista\_tokens) faça:

token\_atual ← minuscula(lista\_tokens[i])

token\_seguinte ← se i+1 < tamanho(lista\_tokens) então minuscula(lista\_tokens[i+1]) senão nulo

padrao\_bigram ← se token\_seguinte ≠ nulo então token\_atual + " " + token\_seguinte senão nulo

// 3.a Detecta booster

se token\_atual ∈ DICIONARIO\_BOOSTER então

acumulador\_booster ← acumulador\_booster + DICIONARIO\_BOOSTER[token\_atual]

i ← i + 1

continue

fim se

// 3.b Detecta conjunção adversativa (reset + reaplica booster)

se token\_atual ∈ CONJUNCOES\_ADVERSATIVAS então

pontuacao\_total ← 0

se acumulador\_booster ≠ 0 então

se ultima\_valencia < 0 então

acumulador\_booster ← –acumulador\_booster

fim se

pontuacao\_total ← pontuacao\_total + acumulador\_booster

acumulador\_booster ← 0

fim se

i ← i + 1

continue

fim se

// 3.c Detecta negação

se token\_atual ∈ CONJUNTO\_NEGACOES então

sinalizar\_negacao ← VERDADEIRO

i ← i + 1

continue

fim se

// 3.d Detecta léxico ou casos especiais

se token\_atual ∈ LEXICO OU padrao\_bigram ∈ LEXICO OU

token\_atual ∈ CASOS\_ESPECIAIS OU padrao\_bigram ∈ CASOS\_ESPECIAIS então

// Escolhe a valência base e consome token extra se for bigrama

se padrao\_bigram ∈ LEXICO então

valencia\_base ← LEXICO[padrao\_bigram]

i ← i + 1

senão se token\_atual ∈ LEXICO então

valencia\_base ← LEXICO[token\_atual]

senão se padrao\_bigram ∈ CASOS\_ESPECIAIS então

valencia\_base ← CASOS\_ESPECIAIS[padrao\_bigram]

i ← i + 1

senão

valencia\_base ← CASOS\_ESPECIAIS[token\_atual]

fim se

// Ajusta valência com boost e negação

(valencia\_tratada, acumulador\_booster, sinalizar\_negacao) ←

ajustar\_valencia(valencia\_base, acumulador\_booster, sinalizar\_negacao)

pontuacao\_total ← pontuacao\_total + valencia\_tratada

ultima\_valencia ← valencia\_tratada

fim se

i ← i + 1

fim enquanto

// 4. Aplica booster residual ao final

se acumulador\_booster ≠ 0 então

se ultima\_valencia < 0 então

acumulador\_booster ← –acumulador\_booster

fim se

pontuacao\_total ← pontuacao\_total + acumulador\_booster

fim se

retorna pontuacao\_total

**FIM FUNÇÃO**

# **16 REFERÊNCIAS BIBLIOGRÁFICAS**

* Hutto, C. J., & Gilbert, E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. *Eighth International Conference on Weblogs and Social Media (ICWSM-14)*.
* Araujo, R. J. (2021). LeIA – Lexicon-based Sentiment Analysis for Portuguese. [*https://github.com/rafjaa/LeIA*](https://github.com/rafjaa/LeIA)
* Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python*. O’Reilly Media Inc. (sobre NLTK)
* spaCy – Industrial-Strength Natural Language Processing in Python. [*https://spacy.io*](https://spacy.io)

1. ericaribeiro@usp.br [↑](#footnote-ref-0)
2. junior.marques@usp.br [↑](#footnote-ref-1)
3. luisvlopes@usp.br [↑](#footnote-ref-2)
4. mjmartins@alumni.usp.br [↑](#footnote-ref-3)
5. raphaelchaves@usp.br [↑](#footnote-ref-4)
6. thiago.ambiel@usp.br [↑](#footnote-ref-5)
7. Dados disponíveis em: <https://github.com/americanas-tech/b2w-reviews01/blob/main/B2W-Reviews01.csv> [↑](#footnote-ref-6)