UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Rafael Denipote Ricci

Análise de sentimentos no Twitter sobre a Reforma da Previdência no ano de 2019

Uberlândia, Brasil

2020

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Rafael Denipote Ricci

Análise de sentimentos no Twitter sobre a Reforma da Previdência no ano de 2019

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Paulo Henrique Ribeiro Gabriel

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2020

Agradecimentos

Aos meus pais por me apoiar e incentivar nesta importante etapa em minha vida. Aos amigos que fiz durante a faculdade, passamos juntos por momentos difíceis, mas também por bons momentos com churrascos, cerveja e muitas risadas. À minha namorada que me incentivou neste trabalho e em sua conclusão. E por fim e não menos importante ao meu orientador, Paulo Henrique, pois sem ele a criação deste Trabalho de Conclusão de Curso não seria possível. Acreditou na minha ideia, me orientou e teve paciência para que fosse possível a sua finalização.

Resumo

Redes sociais são ambientes virtuais muito populares, sendo presente no cotidiano de milhões de pessoas ao redor do mundo. Existem diversas redes cada uma com suas especializações, como compartilhamento de vídeos, imagens ou texto. O *Twitter* é uma entre as mais populares, focada em compartilhamento de textos curtos, os chamados de *tweets*. Este trabalho de conclusão de curso visa analisar sentimentos em relação à proposta da Reforma da Previdência no Brasil, aprovada em 2019. A análise de sentimentos une conceitos de aprendizagem de máquinas, processamento de linguagem natural, linguísticas e análise textual e visa extrair opiniões de textos através de uma classificação, tradicionalmente relacionado a um sentimento "positivo" ou "negativo". Nesse contexto, foram realizados experimentos para apurar o impacto de diferentes técnicas de pré-processamento, uma importante etapa que procura melhorar o desempenho da classificação. Também foram feitas comparações de quatro algoritmos de aprendizagem de máquina, utilizados para classificar os textos: Naïve Bayes, Regressão Logística, Máquinas de Vetores de Suporte (SVM) e Floresta Aleatória. Os resultados do pré-processamento revelaram que diferentes técnicas podem afetar de diferentes formas cada algoritmo. Os algoritmos também obtiveram desempenhos considerados adequados, com o Naïve Bayes tendo a acurácia mais baixa, cerca de 84% com a aplicação de técnicas de pré-processamento de texto. Analisando os resultados das classificações, o Naïve Bayes foi o que classificou mais como positivo, já os demais como neutros. Também foram feitas comparações entre resultados das classificações, demonstrando que os mais semelhantes foram o Regressão Logística e o SVM, com 91% de classificações iguais. Já os mais distintos foram o Naïve Bayes e o Floresta Aleatória com apenas 46% de classificações iguais.

Palavras-chave: Análise de Sentimentos; Aprendizagem de Máquina; Twitter; Reforma da Previdência; Pré-processamento.

Abstract

Social networks are prevalent virtual environments, being present in the daily lives of millions of people around the world. There are several networks, each with their specializations, such as sharing videos, images, or text. Twitter is one of the most popular, focused on sharing short texts, the so-called tweets. This course conclusion paper aims to perform a sentiment analysis regarding the Brazilian Social Security Reform, approved in 2019. The sentiment analysis combines machine learning concepts, natural language processing, linguistics, and textual analysis. It aims to extract opinions from texts through a classification traditionally related to a "positive" or "negative" feeling. In this context, we conducted a set of experiments to determine the impact of different pre-processing techniques, an important step that aims to improve the classification's performance. We perform a comparison of four machine learning algorithms used to classify the texts: Naïve Bayes, Logistic Regression, Support Vector Machines (SVM), and Random Forest. The results of the pre-processing revealed that different techniques could affect each algorithm differently. The algorithms also obtained performances considered adequate, with Naïve Bayes having the lowest accuracy, 84% with the application of text pre-processing techniques. Analyzing the classification results, Naïve Bayes was the one that classified more as positive, while the others as neutral. Comparisons were also made between the classifications' results, showing that the most similar was Logistic Regression and SVM, with 91% of equal classifications. The most distinctive was Naïve Bayes and Random Forest, at only 46% equal ratings.

Keywords: Sentiment Analysis; Machine Learning; Twitter; Brazilian Social Security Reform; Pre-processing.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

Redes sociais virtuais são utilizadas por milhões de pessoas e estão presentes em quase todos os países ao redor do mundo. Constituem um ambiente de interação social online e existem muitos tipos, cada uma com suas particularidades e foco como; compartilhamento de notícias, expressão de opiniões, relacionamentos, interesses profissionais, compartilhamento de imagens e vídeos, divulgação de serviços e produtos, entre outros. As primeiras redes surgiram no final da década de 1990 e, desde então, vêm ganhando cada vez mais popularidade (CORRÊA, 2017). Esse ganho pode estar associado diretamente ao aumento de acessibilidade à internet, tanto por políticas de inclusão digital, como por avanços de tecnológicos.

Existem diversas redes, com grande volume de usuários, como o *Facebook*, *Twitter*, *Instagram*, *YouTube*, *LinkedIn* que são utilizadas para diferentes fins. Neste trabalho de conclusão de curso (TCC) a rede social que foi explorada é o *Twitter*[1]. O *Twitter* surgiu em 2006 com a ideia de compartilhar textos curtos, inicialmente limitados a 140 caracteres, valor aumentado posteriormente para 280, textos esses que são chamados de *tweets*. É uma rede muito popular pela sua ideia simples de compartilhamento de notícias, ideias ou opiniões a respeito de diversos assuntos por qualquer pessoa de forma fácil e rápida (ASLAM, 2020).

Com o grande número de usuários da rede (mais de 340 milhões), o volume de informações geradas é também muito grande, informações essas que podem ser exploradas para diversos fins. Essas informações podem ser exploradas por empresas para recomendar produtos de acordo com dados de navegação de usuários ou por mensagens compartilhas. Também podem ser utilizada para avaliar a satisfação de usuários a respeito de um determinado produto, ou até mesmo para extração de opiniões a respeito de temas políticos.

Seguindo essa ideia de explorar redes sociais para extrair opiniões de um determinado assunto, surgiu a linha de pesquisa chamada Análise de Sentimentos. Também conhecida como Mineração de Opiniões, visa extrair de textos uma atitude, opinião, emoção, sentimentos em ralação a um tópico (SILVA, 2016). Essas informações extraídas a partir dos textos são feitas através de um conjunto de técnicas, algoritmos e modelos que são responsáveis por realizar o tratamento de opiniões (SILVA, 2016). O significado de opiniões é bastante abrangente, mas para Análise de Sentimentos é comumente associado a um sentimento "positivo" ou "negativo". Essa classificação foi a utilizada para este trabalho, com a inclusão da classe "neutro", ou ausência de sentimento.

Neste trabalho, aplicou-se a Análise de Sentimentos em *tweets* a respeito da pro-

[1] <https://twitter.com>

posta da "Reforma da Previdência" no ano de 2019. Trata-se de mudanças em regras de aposentadoria, como fixação da idade mínima de 65 anos para homens e 62 para mulheres, tempo mínimo de contribuição de 15 anos para homens que trabalham no setor privado e 20 para mulheres. No setor público, o tempo mínimo de contribuição foi estipulado para 20 anos para os dois sexos. As mudanças foram aprovadas na Câmera dos Deputados em agosto de 2019 e votada no Senado Federal do Brasil em outubro do mesmo ano (G1, 2019).

Em Análise de Sentimentos, existem diversos passos que devem ser seguidos; um que se destaca é a escolha do algoritmo que vai classificar os textos. Usualmente é feita a escolha de apenas um algoritmo. Neste trabalho, no entanto, foram selecionados quatro algoritmos e, através de experimentos que calcularam os desempenhos dos quatro, os seus desempenhos foram comparados. Ao final dos experimentos os algoritmos foram utilizados para classificar os textos referente a "Reforma da Previdência" coletados e tiveram os resultados gerados também comparados. Além disso, foram realizados testes no processamento de texto, uma etapa igualmente importante na Análise de Sentimentos. No pré-processamento, uma sequência de técnicas foram empregadas e avaliadas de modo a observar como afetam os algoritmos de classificação, além de seus efeitos nos textos. Os algoritmos utilizados foram: *Naïve Bayes*, *Regressão Logística*, *Máquinas de Vetores de Suporte* (SVM) e *Floresta Aleatória*. Esses algoritmos foram selecionados por serem muito populares em projetos que tratam com aprendizagem de máquinas.

1.1 Motivação

Devido ao formato dos *tweets*, com textos curtos, rápidos de serem publicados e de fácil acesso, a rede gera diariamente um volume de textos muito elevado por seus usuários (ASLAM, 2020). Essas características são muito favoráveis para Análise de Sentimento, combinadas com o fato de ser relativamente simples extrair dados de *tweets*. Por esses motivos, a rede tem sido muito explorada em mineração de dados e de análise de sentimentos.

O *Twitter* também tem se demonstrado muito importante durante eventos políticos, como em 2011 durante a chamada "Primavera Árabe", em que se desencadeou uma série de protestos em países do Oriente Médio e norte da África. Já em no Brasil, em 2013 a rede foi muito utilizada durante os protestos a respeito das tarifas de ônibus, e, entre 2015 e 2016, durante o *impeachment* da ex-presidenta Dilma Rousseff (CORRÊA, 2017). Por esses motivos foi escolhido o *Twitter* para desenvolvimento deste trabalho.

Em particular a Reforma da Previdência, também de cunho político, foi muito discutido ao longo do ano de 2019. Isso se deve pelo fato de mudar regras de aposentadoria como fixação de idade mínima para se aposentar, tempo mínimo de contribuição,

regras de transição para trabalhadores que já estão no mercado de trabalho (G1, 2019). Essas mudanças vão afetar diretamente milhões brasileiros inseridos no mercado de trabalho, tanto no formal quanto no informal, que num futuro irão se aposentar. O tema foi amplamente abordado por mídias de comunicação e também foi muito popular no *Twitter*.

Além disso, não há um consenso sobre qual algoritmo de aprendizagem de máquina é melhor para lidar com Análise de Sentimentos e existam poucos trabalhos com a ideia de comparação entre métodos (ARAÚJO; GONÇALVES; BENEVENUTO, 2013). Por essa razão, avaliações de comparação entre algoritmos foram realizadas.

1.2 Objetivos

O principal objetivo deste TCC é desenvolver um projeto de análise de sentimentos, seguindo as etapas convencionais (coleta, processamento de texto, seleção de algoritmo, treinamento do algoritmo selecionado, classificação e análise dos resultados). Porém, devido ao fato de existirem muitos trabalhos que seguem exatamente essas etapas, bem como por algumas questões levantadas, foram realizados experimentos complementarem ao desenvolvimento tradicional da Análise de Sentimentos.

Dessa forma, alguns objetivos secundários foram levantados. O primeiro é verificar o impacto do pré-processamento, não só como isso afeta os textos, mas também os algoritmos de classificação. Outro objetivo é a comparação entre algoritmos de aprendizagem de máquina no contexto de análise de sentimentos. Tradicionalmente é selecionado um algoritmo em Análise de Sentimentos para classificar textos. Essa seleção é feita por meio testes de desempenho em que os candidatos vão ter os resultados comparados e o eleito vai ser o que melhor desempenhar. Existem diversos testes que podem ser aplicados nesta etapa, e entre os mais populares está a validação cruzada, e que se faz necessário uma base de textos previamente classificada. Já para este Trabalho de Conclusão de curso foram escolhidos quatro algoritmos, dos quais foram aplicados os testes de comparação com diversas bases de dados geradas a partir da aplicação de diferentes técnicas de pré-processamento de texto.

No final dos experimentos os algoritmos estavam treinado e prontos para serem utilizados. Os algoritmos classificaram uma base de dados coletada no *Twitter* a respeito da proposta da reforma da previdência no ano de 2019. Uma base composta por 980.577 *tweets* num período correspondente ao mês de janeiro a novembro de 2019. A base coletada foi classificada e os resultados gerados por cada algoritmo foram analisados e comparados.

Por fim, o ambiente de desenvolvimento deste trabalho foi também uma questão. Existem muitas ferramentas que lidam com mineração de dados, aprendizagem de máquina e análise de sentimentos. É possível desenvolver um trabalho utilizando várias ao

mesmo tempo, mas optou-se por apenas uma, a linguagem de programação *Python*. Esse também foi um objetivo secundário levantado pelo projeto: explorar a linguagem no contexto de Análise de Sentimentos. O projeto foi todo desenvolvido em *Python*, da coleta realizada por uma ferramenta desenvolvida por Camelo (2017), e as demais etapas que foram desenvolvidas pelo autor[2].

1.3 Organização do Trabalho

Os próximos Capítulos estão divididos da seguinte forma.

Capítulo 2 - Referencial Teórico : Fundamentação teórica base para desenvolvimento do trabalho. Apresenta conceitos como Aprendizagem de Máquina e Análise de Sentimentos, além formas de transformações textuais para possibilitar que textos escritos por humanos possam ser interpretados por algoritmos de classificação. Apresenta também os algoritmos de classificação que vão ser utilizados, ferramentas de desenvolvimento e, ao final, trabalhos correlatos.

Capítulo 3 - Desenvolvimento e Resultados : Descreve as etapas de desenvolvimento do projeto as quais foram; coleta de dados, criação da base classificada para o treinamento dos algoritmos e experimentos, pré-processamento da base coletada e da classificada, treinamento e testes dos modelos, experimentos com as técnicas de pré-processamento aplicadas e por fim é feita a classificação da base coletada pelos quatro algoritmos. Também em conjunto com a apresentação da etapa já é apresentado os seus resultados e, no caso dos resultados dos experimentos e da classificação, são feitas analises e comparações.

Capítulo 4 - Conclusão : Por fim, na conclusão é feito um breve resumo do Trabalho, depois as contribuições são destacadas e ao final são propostas ideias para trabalhos futuros.

[2] As ferramentas utilizadas são apresentadas na Seção 2.6, o desenvolvimento no Capítulo 3, os códigos estão disponíveis em: <https://github.com/RafaelDRicci/PythonSentimentAnalysis>

2 Referencial Teórico

Neste capítulo, apresenta-se a fundamentação teórica para o desenvolvimento deste trabalho: os principais conceitos, as ferramentas utilizadas e os trabalhos na mesma área de pesquisa.

A Seção 2.1 apresenta brevemente o que é aprendizagem de máquina e os dois problemas que são tratados por essa área. A Seção 2.2 apresenta o que é análise de sentimentos, como a área que vem crescendo recentemente e um de seus principais desafios. A Seção 2.3 apresenta formas de transformação de textos escritos em linguagem humana, de modo a criar uma estrutura que possa ser interpretada pelos algoritmos. Já a Seção 2.4 apresenta como as categorias de classificadores são divididos e discorre a respeito dos que foram utilizados no desenvolvimento deste trabalho. Na Seção 2.5, discute-se o conceito de avaliação de classificadores, apresentando técnicas e métricas de validação. A Seção seguinte a 2.6 discorre a respeito das ferramentas utilizadas para o desenvolvimento do projeto. Por fim, a Seção 2.7 apresenta outros trabalhos desenvolvidos na área de análise de sentimentos e que são relevantes para este TCC.

2.1 Aprendizagem de Máquina

O aprendizagem de máquina (do inglês, *Machine Learning*) é a capacidade de um computador "aprender" a partir de um conjunto de dados e, com isso, fazer previsões ou classificações, sem que tenha sido previamente programado (BISHOP, 2006; SHALEV-SHWARTZ; BEN-DAVID, 2014). Ferramentas de aprendizado de máquina estão presentes em diversas áreas da Computação, como reconhecimento de padrões e inteligência artificial, podendo, também, serem exploradas em diversas aplicações, como a recomendação de produtos por lojas eletrônicas com base nos dados de navegação de clientes.

Problemas que envolvem aprendizagem de máquina podem ser divididos em diversas categorias (PEDREGOSA et al., 2011). Neste texto, foca-se em duas delas: regressão e de classificação. Técnicas de regressão são os que envolvem a predição de uma ou mais variáveis contínuas, as quais podem ter flutuação em seu valor. Um exemplo é tentar prever o consumo de uma bebida dado um conjunto de fatores. Neste caso, é difícil prever exatamente quantos litros vão ser consumidos, então a previsão vai ser uma aproximação. Já a classificação envolve prever uma categoria; por exemplo, pode-se inferir se vai ou não chover em um dia, se um paciente está ou não doente são alguns exemplos de problemas de classificação (BISHOP, 2006; SHALEV-SHWARTZ; BEN-DAVID, 2014).

2.2 Análise de Sentimentos

Análise de sentimentos mescla conceitos de mineração de dados, aprendizagem de máquina, linguística, processamento de linguagem natural e análise textual (SILVA, 2016). Essa área procura extrair opiniões, sentimentos, avaliações, atitudes e emoções sobre determinadas entidades, como produtos, serviços, organizações, indivíduos, problemas, eventos, tópicos e seus atributos (LIU, 2012). No que diz respeito ao tipo de tarefas de aprendizagem de máquina, pode ser considerado como um problema de classificação.

Pesquisas em análise de sentimento vêm crescendo cada vez mais nas últimas décadas e ganhando diversas aplicações, sendo uma forte ferramenta para entender opiniões de um grande volume de pessoas. Isso se deve pelo fato de que, atualmente, um grande volume de dados está disponível — principalmente devido ás mídias sociais. Assim, o crescimento das pesquisas nessa área coincide com o surgimento e o crescimento dessas mídias. De fato, segundo Liu (2012), análise de sentimentos é uma linha de pesquisa ligada diretamente com mídias sociais.

O principal desafio da análise de sentimento é o "entendimento" por parte do computador sobre textos escritos na linguagem natural, de modo a extrair opiniões. Por "opinião", entende-se como a polaridade de uma sentença. Comumente, associa-se um sentimento a essa polaridade, ou seja: "positivo" ou "negativo". Eventualmente, aceita-se a classificação "neutro", que significa a ausência de sentimento.

Para exemplificar como atribuir sentimentos, duas frases vão ser utilizadas, a primeira; "Adoro a comida daquele restaurante" e a segunda "Não gostei da comida do restaurante". A primeira recebe uma classificação "positiva" e a segunda "negativa". Isso pode se justificar pela interpretação do sentido da frase, já que na primeira a palavra "Adoro" traz uma ideia de aprovação aproximando a classificação para a "positiva". Na segunda a combinação das palavras "Não" e "gostei" faz com que a frase transmita a ideia de reprovação o que aproxima a classificação como "negativa". No segundo exemplo o que torna a frase "negativa" é a combinação de duas palavras, já que se apenas "gostei" fosse considerando a classificação poderia mudar. Uma observação é que a palavra "Adoro" e a combinação "Não gostei" foram as que definiram qual o sentimento que a frase transmite, ou seja, tem um maior valor semântico para a classificação.

2.3 Representação dos Atributos

Para os classificadores poderem interpretar as sentenças, elas devem passar por uma etapa de transformação. Nesta Seção, descreve-se a técnica de saco de palavras (do inglês, *bag-of-words*) para representar sentenças, além de variações para melhorar sua classificação. É preciso frisar que essas técnicas não são, necessariamente, as mesmas de

pré-processamento: tal etapa é necessária para treinamento, teste e classificação, ao passo que o pré-processamento visa melhorar os seus resultados.

2.3.1 Saco de Palavras (*bag-of-words*)

Para representar os atributos o modelo saco de palavras é um dos mais utilizados em aprendizagem de máquina e também em análise de sentimentos (SILVA, 2016). Trata-se de uma representação das frases em uma matriz, cujas colunas representam cada termo[1] de todo conjunto de dados e as linhas as frases. A Tabela 1 representa a estrutura de um saco de palavras.

Tabela 1 – Representação de um saco de palavras (*bag-of-words*).

-	T1	T2	T3	...	Tm
Sentença 1	a11	a12	a13	...	a1m
Sentença 2	a21	a22	a23	...	a2m
...	...	...	...	...	...
Sentença n	an1	an2	an3	...	anm

Fonte: adaptada de Silva (2016).

Os valores na matriz podem representar, por exemplo, a presença (ou até a quantidade) do termo na frase ou a ausência do mesmo. Por exemplo, caso o valor $a11$ seja 1 significa que a Sentença 1 possui uma ocorrência do termo $T1$; analogamente, $a12 = 2$ significa que o termo $T2$ ocorre duas vezes na Sentença 1 e $a13 = 0$ significa que o termo não ocorre na referida sentença. Esse exemplo ilustra um caso de matriz de frequências (SILVA, 2016).

Os termos T foram considerados, em um primeiro momento, como sinônimos de palavras, de modo a facilitar a explicação; porém, eles recebem uma nomenclatura diferente, sendo chamados de "características" (ou, em um termo comumente empregado em inglês, "*features*"). Essa mudança de nomenclatura é importante porque esses termos não são, necessariamente, compostos por uma única palavra, como é o caso dos bigramas.

2.3.2 Bigramas

Em bigramas as características são representadas por dois termos em sequência (BIRD; KLEIN; LOPER, 2009). Para exemplificar esse conceito, seja a frase "Não gostei da comida daquele restaurante". Em bigramas as características seriam: "Não gostei", "gostei da", "da comida", "comida daquele" e "daquele restaurante".

Se fosse utilizado apenas um termo por característica (unigrama), os termos "Não" e "gostei" constituiriam uma característica cada, o que pode levar o classificador a inter-

[1] Nesta monografia, "termo" está sendo utilizado como sinônimo de "palavra".

pretar a sentença como "positiva". Já em bigramas os dois termos pertencerão à mesma característica ("Não gostei"); assim, há uma maior probabilidade do classificador interpretar a mesma sentença como "negativo".

2.3.3 TF-IDF

Além da forma como as características são definidas, existem diferentes maneiras de definir os valores na matriz. A mais simples é definida pela presença ou a ausência dessa característica na frase, recebendo valores binários de 0 (ausência) e 1 (presença). Outra maneira, é pela frequência com a qual ela ocorre, ou seja, um valor referente à quantidade de vezes que o termo é presente na frase, ou 0 caso não ocorra.

Existe, ainda a frequência ponderada TF-IDF (do inglês, *Term Frequency - Inverse Documente Frequency*), que considera a relação do termo em uma frase com todo *corpus textual*[2] (RIBEIRO, 2015). Um termo que é muito frequente em todo *corpus* não é muito relevante, com baixo valor descritivo, já que por ocorrer em muitas frases não é um termo que tenha um peso para definir qual o sentimento que a frase expressa. Por outro lado, um termo que aparece poucas vezes tem um valor descritivo melhor (RIBEIRO, 2015), pois por ocorrer poucas vezes ele tem um peso maior quando esta presente em uma frase. O valor do TF-IDF é computado pela Equação (2.1).

$$TFIDF = TF \cdot \log\left(\frac{N}{DF}\right) \tag{2.1}$$

Nessa equação, o valor TF é a frequência do termo na frase, N é o total de frases em todo o *corpus* e DF é a frequência do termo em todo o *corpus*. É notável que, pela Equação (2.1), quanto maior o valor de DF, ou seja, quanto mais comum aquele termo for em todo *corpus* menor vai ser sua relevância. Analogamente, quanto menor for a seu valor, ou seja, quanto menos comum for o termo, sua capacidade descritiva vai aumentar.

2.4 Classificadores

Entre os métodos para abordar a classificação de texto, destacam-se os baseados em léxicos e aprendizagem supervisionada. Métodos baseados em léxicos requerem uma lista de palavras para as quais um sentimento já foi atribuído previamente por especialistas (SILVA, 2016). Não demandam treinamento do modelo, pois a classificação vai usar a lista de palavras (que também recebe o nome de dicionário léxico) para basear a sua classificação (BORDIN JUNIOR, 2018).

Já no caso dos métodos supervisionados, há a necessidade de uma base classificada para o treinamento (SILVA, 2016). Essa base pode ser chamada de base treino e os métodos

[2] *Corpus textual* ou *corpus linguístico* se refere a todos o conjunto de textos.

utilizam conceitos de aprendizagem de máquina para fazer a classificação. Assim, quando uma nova sentença é apresentada, tenta-se prever sua polaridade com base no treinamento.

No contexto da análise de sentimentos, as polaridades são "positivo" ou "negativo". A base de treino pode ser criada a partir de uma amostra dos dados a serem classificados, a qual vai receber uma classificação por especialistas humanos. É possível, também, utilizar outras bases já classificadas por outros trabalhos, desde que tenha um contexto semelhante (BORDIN JUNIOR, 2018). Neste trabalho, foram utilizados algoritmos de aprendizado de máquina supervisionados, os quais foram treinados com uma base de dados provenientes de outros trabalhos. Em seguida, esses métodos foram comparados entre si e utilizados para classificar a base deste trabalho. Os algoritmos utilizados são descritos a seguir.

2.4.1 Naïve Bayes

No Naïve Bayes é um classificador com uma abordagem probabilística, baseado no *Teorema de Bayes*, e considerado ingênuo por assumir que as variáveis não são dependentes entre si (RIBEIRO, 2015). Por ter um bom desempenho computacional na classificação de texto o *Naïve Bayes* vem sendo muito utilizado para tratar problemas de análise de sentimentos (CORRÊA, 2017).

O Teorema de Bayes, em linhas gerais, define que, dado um conjunto de evidências B, qual a probabilidade P de A ocorrer (SCHMITT, 2013). Isso é mostrado na Equação (2.2) (GONÇALVES, 2020).

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \tag{2.2}$$

em que:

- $P(B \mid A)$ é probabilidade de B acontecer dado que A ocorreu;
- $P(A)$ é a probabilidade de A ocorrer;
- $P(B)$ é probabilidade de B ocorrer, $P(B) \neq 0$.

2.4.2 Regressão Logística

A regressão logística (do inglês, *Logistic Regression*) também é um algoritmo muito utilizado para lidar com classificação de texto[3]. Esse algoritmo utiliza modelos de regressão para calcular ou prever a probabilidade de um evento ocorrer (FIGUEIRA, 2006).

[3] Embora o algoritmo tenha "regressão" em seu nome, ele é muito aplicado em problemas de classificação e não de regressão.

A ocorrência do evento é definida pela variável dependente, que dado um conjunto de variáveis explanatórias, estima a sua probabilidade de ocorrer. O modelo pode ser definido pela Equação (2.3):

$$P(Y) = \frac{1}{1 + e^{-g(x)}} \tag{2.3}$$

em que:

$$g(x) = B_0X_0 + B_1X_1 + \ldots + B_pX_p \tag{2.4}$$

No caso, Y representa a variável dependente e $P(Y)$ retorna a probabilidade do evento ocorrer e $\{X_0, X_1, \ldots, X_p\}$ representa o conjunto de p variáveis explanatórias ou independentes. Os coeficientes B são estimados por um conjunto de dados de treino.

Para classificação, quando se estuda a probabilidade do evento ocorrer, se $P(Y) > 0{,}5$ então $Y = 1$ (o evento ocorre) ou, se $P(Y) < 0{,}5$, então $Y = 0$ (o evento não ocorre). Além disso, que o valor de $P(Y)$ representa a probabilidade daquele evento estudado acontecer.

2.4.3 Florestas Aleatória

O algoritmo florestas aleatórias (do inglês, *Random Forest*) é composto por uma coleção de árvores de decisão. Para eleger qual será a classe de um novo elemento, todas as árvores classificam-no e a classe que tiver mais votos corresponderá àquele elemento (OSHIRO, 2013). Antes de apresentar o algoritmo em si, é necessário um entendimento prévio do que é uma árvore de decisão.

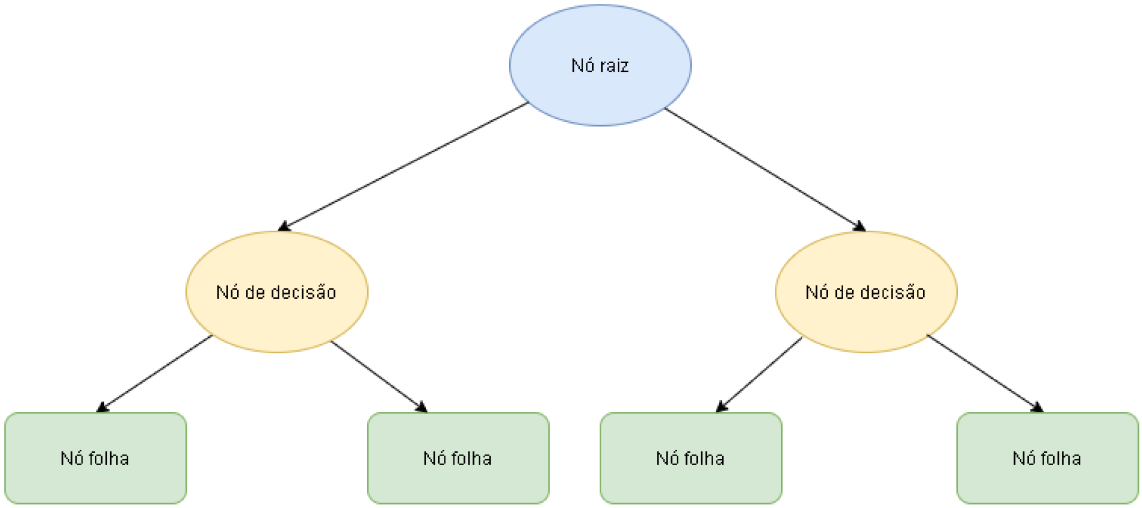
2.4.3.1 Árvores de Decisão

Uma árvore de decisão é um classificador simples composto por nós folhas que correspondem a uma classe ou nós de decisão que testam algum atributo. Cada resultado dos testes levam a uma subárvore. As subárvores são compostas pela mesma estrutura da árvore (MONARD; BARANAUSKAS, 2003).

A Figura 1 exemplifica a estrutura de uma árvore de decisão. O nó mais ao topo corresponde ao nó raiz; esse nó é um de decisão como os demais, mas por ser o início da árvore recebe um nome próprio. Cada nó de decisão corresponde a um atributo dos dados analisados como, por exemplo, se um paciente sente dor e dependendo da resposta (sim ou não para este exemplo) leva para uma subárvore ou outra. Caso a resposta leva a um nó folha aquela amostra recebe uma classificação correspondente a folha (STARMER, 2018).

A partir de uma base de dados, é possível construir uma árvore e, com ela, fazer predições de novos elementos. São fáceis de construir, fáceis de usar e de interpretar, no

Figura 1 – Estrutura básica de uma árvore de decisão.



Fonte: o autor, a partir de um exemplo apresentado por Starmer (2018).

entanto, possuem uma baixa capacidade preditiva e uma acurácia baixa. Elas conseguem ter uma boa acurácia apenas com os dados usados em sua criação, tornando assim pouco úteis para serem utilizadas em problemas reais de aprendizagem de máquina. Para solucionar esta questão que foi criado o algoritmo conhecido como floresta aleatória (STARMER, 2018).

2.4.3.2 Classificador Florestas Aleatórias

Como já foi descrito anteriormente, esse classificador é composto por um conjunto de várias árvores de decisão, com intuito de manter suas facilidades e melhorar sua acurácia, capacidade generativa e preditiva (STARMER, 2018). Também é gerado a partir de uma base de dados e, para explicar como são criadas, será utilizada a base mostrada pela Tabela 2.

Tabela 2 – Base de dados que vai ser utilizada para exemplificar como é a criação de um algoritmo floresta aleatória (*Random Forest*). Corresponde a dados de dias que choveu baseado em determinadas características do dia. A coluna "Índice" é para identificação do elemento, a coluna "Chuva" representa a classe e as demais são suas variáveis. Vale ressaltar que estes dados não corresponde a uma base real, foi criada apenas com intuito de demonstração.

Índice	Temperatura	Nublado	Vento	Umidade	Chuva
0	Elevada	Sim	Sim	Alta	Choveu
1	Baixa	Sim	Não	Alta	Choveu
2	Elevada	Não	Sim	Baixa	Não Choveu
3	Baixa	Sim	Não	Baixa	Não Choveu

Fonte: o autor, a partir de um exemplo apresentado por Starmer (2018).

A primeira etapa para construir uma Floresta de Aleatória é selecionar aleatoriamente uma amostragem dos dados originais. Assim construindo uma base de dados com os originais e de forma aleatória. A Tabela 3 é um exemplo de como fica uma amostragem baseado nos dados da Tabela 2. O primeiro elemento selecionado aleatoriamente corresponde ao terceiro (Índice 3) dos dados originais e assim por diante até ter uma nova base de dados. Essa nova base também é conhecida como *Bootstrapped Dataset* (MONARD; BARANAUSKAS, 2003).

A próxima etapa é selecionar algumas variáveis (colunas que correspondem a características do dia) da amostragem para criar uma árvore de decisão. Por exemplo, escolher apenas as variáveis "Temperatura" e "Nublado". Depois determinar qual entre às duas melhor separa as classes ("Choveu" ou "Não Choveu"). Existem diversas formas de determinar qual atributo melhor separa os elementos, como escolher de forma aleatória, menos valores, mais valores, ganho máximo, índice gini e razão de ganho (MONARD; BARANAUSKAS, 2003). Após escolher qual será o primeiro nó da árvore (nó raiz), é selecionadas novamente duas variáveis de forma aleatória, mas sem repetir a escolhida anteriormente. Por exemplo, supondo que "Nublado" foi selecionado para ser o nó raiz, então selecionar duas entre as demais variáveis para os outros nós (STARMER, 2018). Continuar com está seleção de atributos até não sobrar mais variáveis ou até os elementos estarem completamente divididos (nó folha).

Tabela 3 – Uma amostragem dos dados da Tabela 2 em que os dados foram selecionados aleatoriamente até compor uma nova base (*Bootstrapped Dataset).*

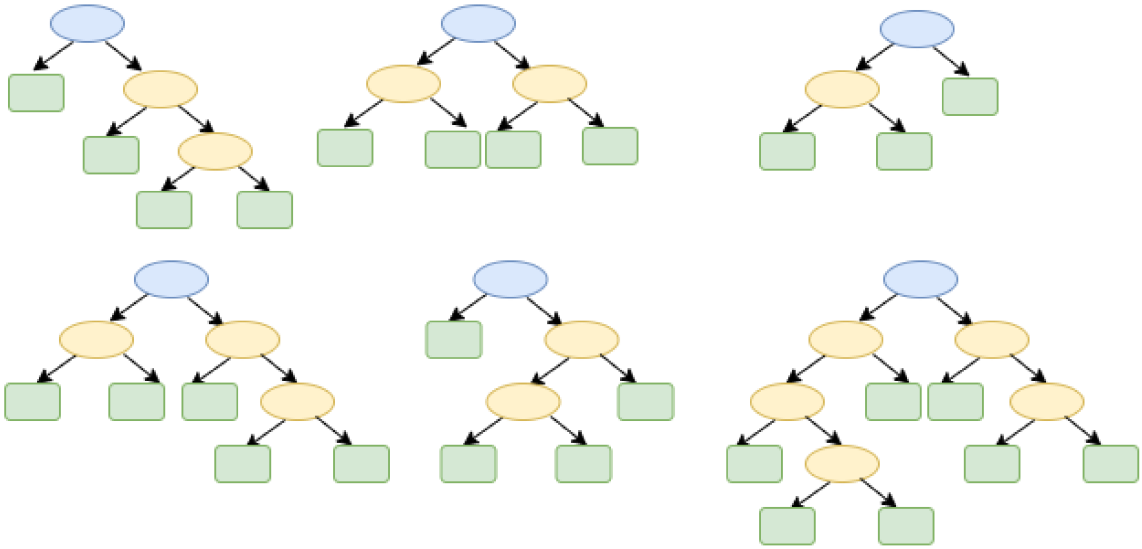
Índice	Temperatura	Nublado	Vento	Umidade	Chuva
3	Baixa	Sim	Não	Baixa	Não Choveu
0	Elevada	Sim	Sim	Alta	Choveu
2	Elevada	Não	Sim	Baixa	Não Choveu
0	Elevada	Sim	Sim	Alta	Choveu

Fonte: o autor, a partir de um exemplo apresentado por Starmer (2018).

Dessa forma uma árvore vai ser construída. Essas duas etapas devem ser repetidas até que se construa a quantidade de árvores desejadas (STARMER, 2018). A Figura 2 ilustra uma pequena floresta aleatória composta por seis árvores.

Já a classificação é feita por uma votação, na qual todas as árvores classificam um novo elemento e a classe que esse recebe corresponderá à de mais votos. Por exemplo, utilizando a mesma base de dados da Tabela 2, porém com 100 árvores em vez de apenas seis, se um novo elemento é apresentado ao classificador e 80% das árvores o classificam como "Choveu", então essa será sua classificação final. É importante destacar que uma quantidade baixa de árvores permite uma maior chance de ocorrência de empates (portanto, tem menor precisão de classificação).

Figura 2 – Ilustração de uma floresta aleatória composta por seis Árvores.



Fonte: o autor, a partir de um exemplo apresentado por Starmer (2018).

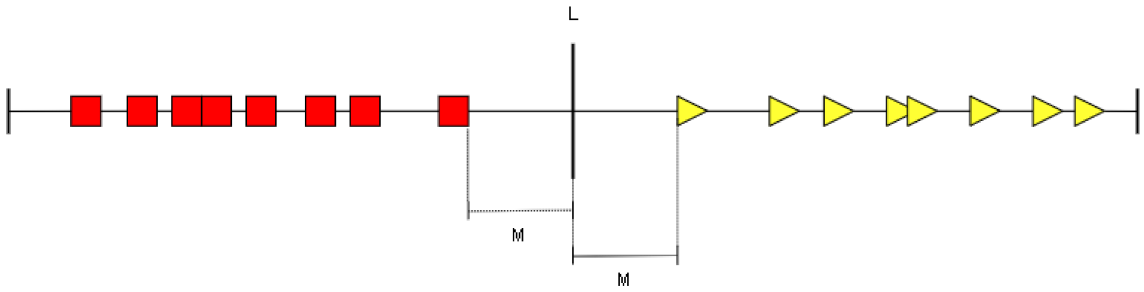
2.4.4 Máquinas de Vetores de Suporte

Máquinas de vetores de suporte (do inglês, *Support Vector Machine*, SVM) é um classificador que divide duas classes no espaço vetorial através de um hiperplano (RIBEIRO, 2015). O algoritmo utiliza um limiar de separação para dividir as classes, porém, como é possível que exista mais de um limiar, o algoritmo tem que decidir qual deles separa as classes de maneira mais adequada. Essa decisão é tomada com base na distância entre um elemento de uma classe com um de outra classe. A distância entre um elemento e a limiar recebe o nome de margem.

A Figura 3 representa um primeiro cenário simples com apenas uma característica (uma dimensão) dividida em duas classes: quadrados e triângulos. O valor de L corresponde a limiar de separação e corresponde a um único ponto e M são as margens. A posição de L é definida durante a etapa de treinamento e pode ser calculada com base na metade da distância entre o último elemento da classe dos quadrados com o primeiro da classe triângulo (como no caso da Figura 3). Após o treinamento, todos os novos elementos que forem apresentados ao algoritmo, e que estiverem à esquerda de L, serão classificado como quadrado e todos que estiverem a direita como triângulo. Assim, as novas entradas que serão classificadas vão estar mais próximas aos elementos da classe em que pertencem do que de elementos da outra classe (STARMER, 2019).

Seja, agora, outro cenário em que a disposição dos elementos das classes está distribuída de uma forma diferente, de modo que o primeiro elemento da classe triângulo pertença agora à classe quadrado, como representado na Figura 4. Nesse caso, se a mesma técnica for empregada, o limiar de decisão estará deslocado para direita. Após a etapa de treinamento, pode ser que novas entradas sejam classificadas como quadrados, mesmo

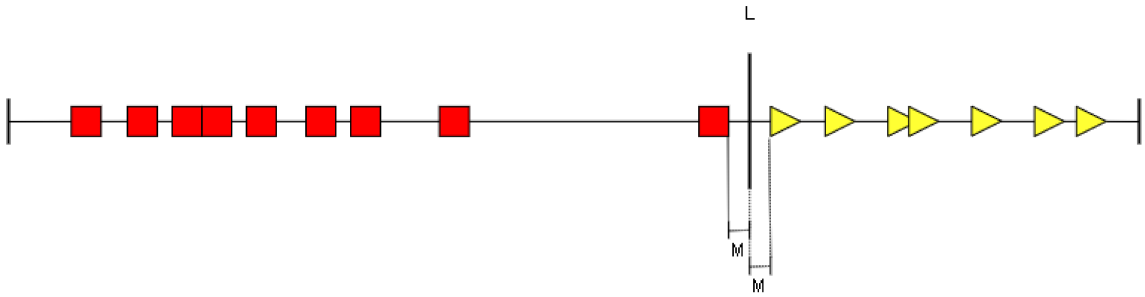
Figura 3 – Representação com uma dimensão que exemplifica uma possibilidade de separação entre classes pelo algoritmo SVM.



Fonte: o autor, a partir de um exemplo apresentado por Starmer (2019).

que estando mais próximas dos elementos da classe dos triângulos. Para resolver esse problema, a escolha da posição de L não precisa, necessariamente, utilizar os valores das extremidades, e sim outros elementos. Isso é ilustrado na Figura 5, na qual foi escolhido o penúltimo quadrado e o segundo triângulo para compor os valores de L e M. Dessa forma, mesmo que suponha que um elemento pertencente aos quadrados fique dentro da área dos triângulos, novos elementos vão receber classificações que mais se aproximam dos elementos de uma determinada classe (STARMER, 2019).

Figura 4 – Representação com uma dimensão que exemplifica uma segunda possibilidade de divisão pelo algoritmo SVM com os elementos distribuídos de forma diferente.

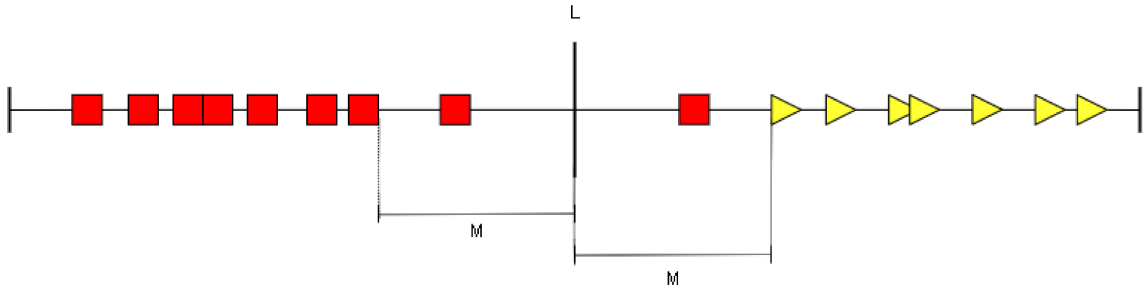


Fonte: o autor, a partir de um exemplo apresentado por Starmer (2019).

Nesses cenários, foram ilustradas maneiras de calcular a posição de L com base em dois elementos de cada classe, contudo existem mais formas, e o número de possibilidades só aumento com base no tamanho dos dados de treino. Assim o algoritmo precisa escolher qual a melhor posição do limiar de separação e dos valores das margens. E para isso, durante a etapa de treinamento, é realizada a técnica de validação cruzada[4] para escolher qual a melhor posição de L e M, de moda a otimizá-los. Assim, será escolhida uma posição em que ele acerte o máximo possível de classificação sem que muitos elementos fiquem fora da área em que pertençam.

[4] A técnica de validação cruzada é descrita na Seção (2.5.1)

Figura 5 – Representação com uma dimensão que exemplifica uma terceira possibilidade de divisão pelo algoritmo SVM com os elementos distribuídos da mesma forma que na Figura 4.



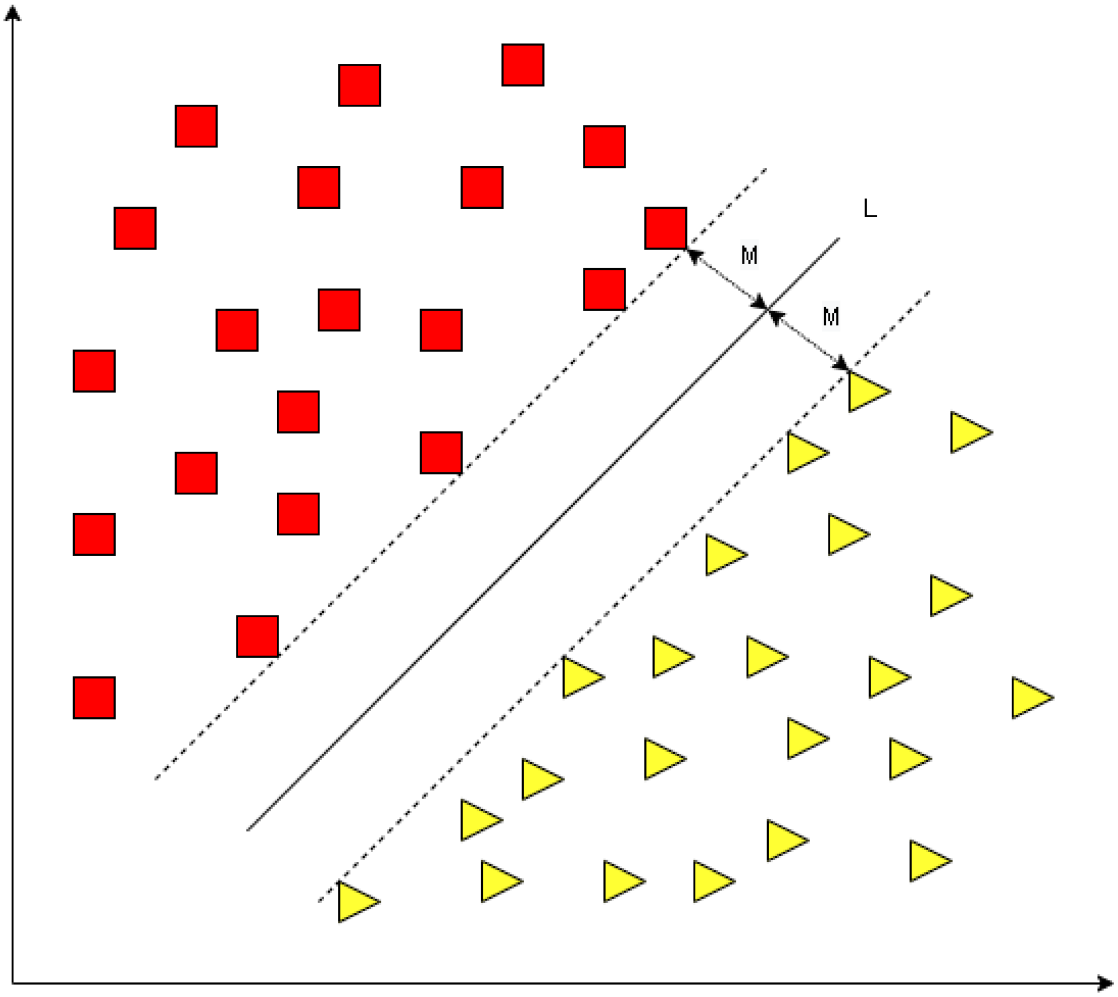
Fonte: o autor, a partir de um exemplo apresentado por Starmer (2019).

Os cenários anteriores exemplificaram classes que possuem apenas um atributo, o que implica apenas uma dimensão; porém, é provável que ao lidar com dados reais eles possuam mais de uma característica. A Figura 6 demonstra como o SVM divide duas classes em um cenário com duas dimensões. Agora L não é mais um ponto, mas sim uma linha divisória e as margens correspondem a distância entre elementos de cada classe com L. A mesma lógica é aplicada nos cenários com uma dimensão, em que é possível várias linhas divisórios e a escolhida vai ser por validação cruzada. Já com três dimensões o limiar de separação vai ser um plano que divide os elementos das classes. Conforme o número de dimensões dos dados aumentam, o limiar de separação também muda mas é possível observar um padrão. O limiar de separação corresponde a um hiperplano de $D-1$ dimensões em que D representa o número de dimensões dos dados analisados.

Os exemplos até aqui representam classes lineares, ou seja, classes que podem ser separadas por um ponto ou reta (hiperplano), como no caso do exemplo da Figura 3. No entanto, existem casos em que não é possível dividi-las dessa maneira. Um exemplo é ilustrado na Figura 7, no qual, independente da posição do limiar de separação, não é possível uma separação das classes de uma forma satisfatória. Para lidar com esses casos, o SVM precisa realizar uma adaptação nos dados e uma solução possível é aumentar o número de dimensão dos próprios dados. Como exemplo, a Figura 8 representa os dados da Figura 7 em que ocorre o aumento de uma dimensão para duas (e, assim, o algoritmo consegue dividi-las). Essa transformação é feita através de uma função de *kernel*. Existem diversas funções de *kernel* que podem ser utilizada para solucionar o problema de dados não linearmente separáveis, sendo que a solução de aumentar o número de dimensões é apenas uma delas (STARMER, 2019). No momento da criação do modelo, o *kernel* a ser utilizado fica a cargo do programador[5].

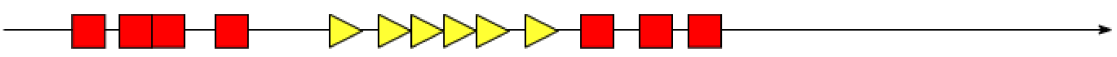
[5] Para mais informações sobre essas funções *kernel*, o leitor pode acessar a documentação do *sklearn* (PEDREGOSA et al., 2011).

Figura 6 – Representação com duas dimensão da separação de duas classes pelo algoritmo SVM.



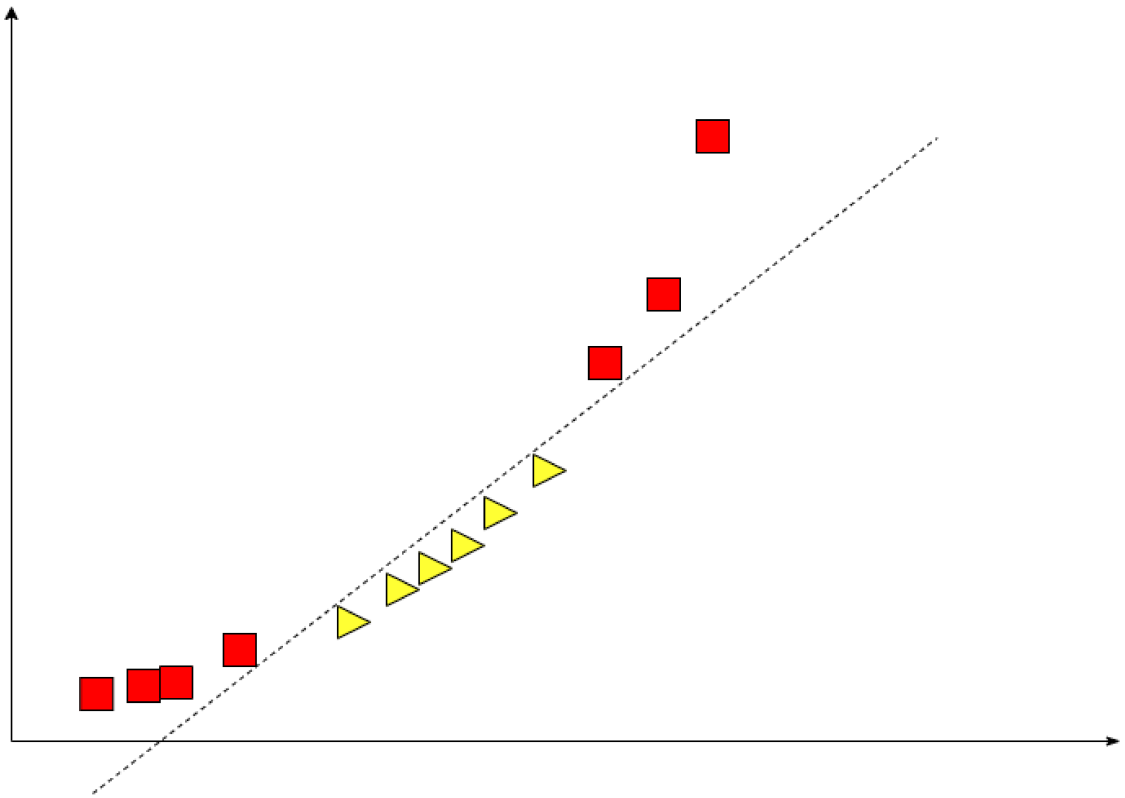
Fonte: o autor, a partir de um exemplo apresentado por Starmer (2018).

Figura 7 – Dados distribuídos de forma em que não é possível dividir as classes de forma linear pelo algoritmo SVM



Fonte: o autor, a partir de um exemplo apresentado por Starmer (2019).

Figura 8 – Transformação do conjunto de dados apresentados na Figura 7 para que o algoritmo SVM seja capaz de realizar a divisão das classes.



Fonte: o autor, a partir de um exemplo apresentado por Starmer (2019).

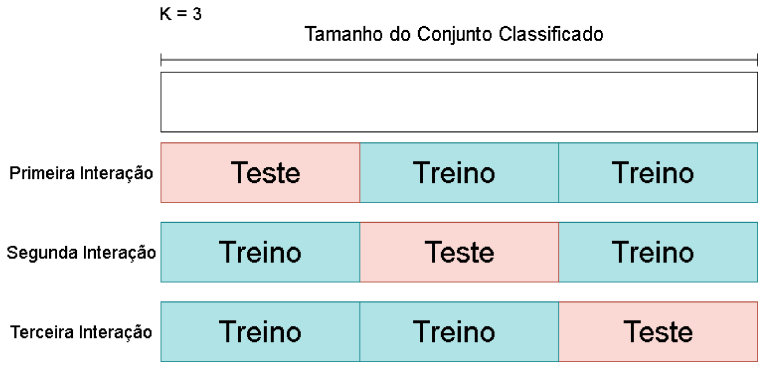
2.5 Avaliação dos Classificadores

A etapa de avaliação é consiste em estimar o quanto os modelos conseguem classificar corretamente amostras, com o objetivo de, por exemplo, escolher qual algoritmo utilizar para um determinado conjunto de dados, ou então com intuito de comparação entre modelos. Uma técnica simples é obter um conjunto de dados já classificados e dividi-lo em dois subconjuntos, um para o treinamento e o outro para teste. O treinamento é realizado com o primeiro subconjunto e o segundo é utilizado para a classificação, ou seja, os elementos do subconjunto de teste são classificados pelo modelo. Em seguida, é feita uma comparação entre a classificação, já que os dados de teste já tinham uma classificação prévia. Finalmente, calcula-se qual a sua taxa de acerto. Por exemplo, se no subconjunto de teste existem 100 objetos e um modelo classificou corretamento 80, esse teve uma taxa de acerto de 80%. Existem outras formas de realizar a validação e entre as mais utilizadas está a técnica conhecida como validação cruzada (CORRÊA, 2017).

2.5.1 Validação Cruzada

O uso de diferentes amostras de dados e medidas de validação é uma boa maneira de avaliar a capacidade preditiva dos modelos (CORRÊA, 2017). Por esse motivo a técnica da validação cruzada foi escolhida. A técnica consiste em dividir o conjunto de dados classificado em k subconjuntos de dados e serão realizados k interações (SCHMITT, 2013). Essa técnica também é chamada de *k-fold cross validation* e o seu funcionamento está esquematizado na Figura 9.

Figura 9 – *k-fold Cross Validation* com $k = 3$



Fonte: o autor

A Figura 9 ilustra um exemplo com $k = 3$; assim o conjunto original será dividido em três subconjuntos de tamanhos iguais. Na primeira interação, o primeiro subconjunto compreende os dados para teste e outros dois para treino. Na segunda interação o segundo subconjunto vai ser o de testes e os demais os de treino. Por fim, na terceira etapa, o último subconjunto será o de teste, e o primeiro e o segundo de treino. Para calcular a taxa de acerto, é feita uma média das taxas de acerto pelo valor de k.

2.5.2 Matriz de confusão

Após a validação cruzada, é possível criar uma matriz que explora melhor os dados classificados. Essa matriz é chamada de matriz de confusão e se trata de uma forma de visualizar os resultados a partir de uma série de medidas de qualidade (SCHMITT, 2013). Uma matriz de confusão é ilustrada na Tabela 4.

Quanto ao conteúdo da matriz, esse representa as medidas de avaliação dos resultados, que são:

- Verdadeiro Positivo (VP): Classificações corretas para classe "*positivo*"
- Verdadeiro Negativo (VN): Classificações corretas para classe "*negativo*"
- Verdadeiro Neutro (VNE): Classificações corretas para classe "*neutro*"

Tabela 4 – Exemplo de matriz de confusão. Colunas representam as classes reais e linhas as preditas pelo algoritmo de classificação.

Classe	Positivo Real	Negativo Real	Neutro Real
Positivo Predito	VP	FP	FP
Negativo Predito	FN	VN	FN
Neutro Predito	FNE	FNE	VNE

Fonte: o autor.

- Falso Positivo (FP): Classificações incorretas que receberam classificação "*positivo*"
- False Neutro (FN): Classificações incorretas que receberam classificação "*negativo*"
- Falso Neutro (FNE): Classificações incorretas que receberam classificação "*neutro*"

Essa sequência de medidas faz referência à quantidade de sentenças que receberam uma classe específica e expõe quantas foram (in)corretamente classificadas. Também fica em evidência para classificação incorreta quais foram as classes que aquelas sentenças receberam. No caso de usar matriz de confusão em conjunto com a validação cruzada, para cada iteração, uma matriz será gerada e todas elas terão seus valores somados.

Além de expor os resultados dos testes, também é possível calcular medidas de avaliação. Uma medida de avaliação de classificadores é a acurácia (Equação 2.5), que se trata de uma medida qualitativa responsável por computar a proporção de classificações corretas (CORRÊA, 2017).

$$\text{Acurácia} = \frac{VP + VN + VNE}{VP + VN + VNE + FP + FN + FNE} \tag{2.5}$$

O que essa equação faz é formalizar a quantidade de classificações corretas pelo total de textos. Desta maneira, é possível realizar um comparativo entre os algoritmos e determinar qual obteve um melhor resultado nos experimentos.

2.6 Ferramentas

Nesta Seção, são apresentadas ferramentas empregadas no desenvolvimento deste TCC. Existem diversas ferramentas para abordar problemas de aprendizagem de máquina como o Weka[6], ferramenta que está entre as mais utilizadas para abordar o problema (HALL et al., 2009). Essa ferramenta fornece uma coleção de algoritmos de aprendizagem de máquina e pré-processamento de dados além de ser possível avaliar e comparar diferentes algoritmos (HALL et al., 2009).

Outra ferramenta muito utilizada é o *Scikit Learn*[7], uma biblioteca de aprendi-

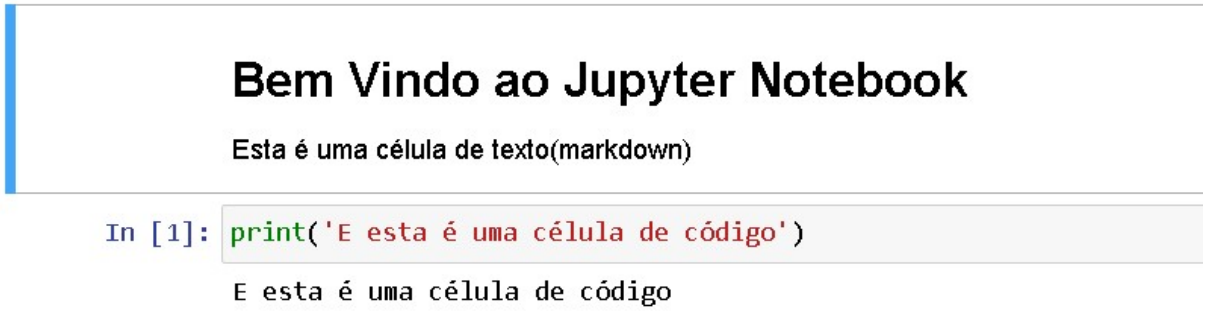
[6] O Weka pode ser acessado em: <https://www.cs.waikato.ac.nz/ml/weka/>
[7] A biblioteca *Sciket Learn* pode ser acessada em: <https://scikit-learn.org/stable/>

zagem de máquina com código aberto para linguagem Python (SCIKIT-LEARN, 2019). Como um dos objetivos deste projeto é explorar a linguagem *Python* para o campo de análise de sentimentos, a biblioteca *Scikit Learn* foi escolhida para criar os modelos em conjunto com outras ferramentas auxiliares. Nas seções seguintes, as bibliotecas e ferramentas que foram utilizadas são apresentadas.

2.6.1 Jupyter Notebook

Este trabalho foi desenvolvido no ambiente de desenvolvimento chamado *Jupyter Notebook*[8]. Trata-se de uma aplicação web *open-source* para criar e compartilhar arquivos que integram o código com textos, equações, visualizações e gráficos[9]. Pode ser utilizado para desenvolver projetos em diversas linguagens, entre elas o *Python*. Além disso, permite desenvolver o código em conjunto textos de maneira bem organizada o que facilita muito a documentação, além do que o *GitHub*[10] consegue renderizar os arquivos, permitindo assim sua visualização sem a necessidade de baixá-lo. Os arquivos *Jupyter Notebook* são particionados em células, as quais podendo ser compostas por código ou textos. A Figura 10 apresenta brevemente o formato das células.

Figura 10 – Apresentação do *Jupyter Notebook*.



Fonte: o autor.

A estilização dos textos é feita por meio do *Markdown*, uma linguagem de marcação simples que converte o texto em HTML. Já as células de códigos são responsáveis pela execução de trechos de códigos. Elas podem ser executadas independentemente, como se fossem uma execução no terminal. Caso tenha algum retorno, esse será mostrado logo abaixo da célula. Além disso, é possível gerar gráficos com bibliotecas como a *mathplotlib* (HUNTER, 2007), com o gráfico sendo desenhado como saída de uma célula. Além disso, as variáveis criadas em uma célula ficarão armazenadas em memória, podendo ser utilizadas posteriormente em outra célula.

[8] Disponível em <https://jupyter.org/>
[9] Para mais informações e um guia de uso a repeito do *Jupyter Notebook* a documentação pode ser consultada em: <https://jupyter-notebook.readthedocs.io/en/stable>
[10] Disponível em <https://github.com/>

Ao abrir um projeto *Jupyter*, ele é apresentado como se já estivesse sido executado, com todas as saídas. Isso se trata de apenas uma forma de visualização, o arquivo não está sendo executado. Será, portanto, necessário executar todas as células no caso de criar novas células que utilizem variáveis ou funções do projeto, por exemplo. Além disso, a ordem de execução das células é importante, embora seja possível executar qualquer célula em qualquer ordem, isso pode gerar resultados completamente diferentes.

2.6.2 *pandas*

O *pandas*[11] é uma ferramenta muito utilizada em trabalhos que lidam com grande volume de dados na linguagem *Python*. Trata-se de uma biblioteca que lida com conjuntos de dados tabulares, como dados armazenados em planilhas ou banco de dados (MCKINNEY, 2020), sendo capaz de interpretar diferentes formatos de arquivos (*csv*, *xls*, *sql*, *json*, etc.). Depois da leitura dos arquivos, eles ficam organizados em tabelas que recebem o nome de *Data Frame*. Um *Data Frame* tem um formato muito semelhante a uma tabela*Excel*, o que facilita sua apresentação e organização. A Figura 11 ilustra o formato de um *Data Frame*.

Figura 11 – Exemplo de um *Data Frame*.

	Coluna1	Coluna2	Coluna3	Coluna4
Linha1	X_{11}	X_{12}	X_{13}	X_{14}
Linha2	X_{21}	X_{22}	X_{23}	X_{24}
Linha3	X_{31}	X_{32}	X_{33}	X_{34}

Fonte: o autor.

Um *Data Frame*, como uma tabela tradicional, é organizado por linhas e colunas e ambos possuem índices que podem ser explicitamente nomeados. Para dados utilizados em aprendizagem de máquina, cada linha representa um elemento ou classe e as colunas suas características. Possui, também, recursos de busca por condições lógicas (método *query*), de forma semelhante das que são feitas em bancos de dados. Permite, também, inserir e modificar dados ou realizar cálculos matemáticos, recursos que podem ser utilizados de forma combinados como, por exemplo, somar todos os valores de uma determinada coluna ou fazer uma busca e realizar uma média dos valores. Também é possível concatenar duas tabelas, copiar todos os valores de uma tabela e criar gráficos com a tabela ou parte dela.

O *pandas* possui diversos recursos que foram utilizados para desenvolvimento deste trabalho e para a apresentação dos resultados. Muitas tabelas presentes nesta monografia foram criadas diretamente com o *pandas* em conjunto com outros recursos de estilização.

[11] Disponível em <https://pandas.pydata.org/>

2.6.3 Scikit Learn

Como já foi brevemente apresentado, o *Scikit Learn*[12] é uma biblioteca de aprendizagem de máquina para a linguagem *Python* (PEDREGOSA et al., 2011). Com ela, é possível criar modelos de classificação, regressão e agrupamento de dados, além de oferecer recursos auxiliares como redução de dimensionalidade, seleção e avaliação de modelos e pré-processamento de dados (SCIKIT-LEARN, 2019).

O *Scikit Learn* fornece diversos algoritmos, tanto supervisionados quanto não-supervisionados, assim sendo, é possível utilizá-lo para abordar diferentes problemas que envolvem aprendizagem de máquina. Para problemas de análise de sentimentos, existem recurso para lidar com linguagem natural em que o texto é convertido em uma estrutura com a qual algoritmos possam lidar, como um saco de palavras[13]. Para realizar essa tarefa exite o módulo *feature_extraction*, que extrai informações de dados brutos, tanto de textos quanto de imagens. Existe, também, o módulo *model_selection* para avaliação e seleção de modelos, através de diversos métodos entre eles o de validação cruzada[14]. Com essa biblioteca, ainda é possível realizar cálculos de métricas de qualidade como a acurácia, funcionalidade que é desempenhada através do método *metrics*.

Já para os algoritmos de aprendizagem de máquina, o *Scikit Learn* oferece diversos métodos para a criação de modelos. Neste trabalho, foram utilizados quatro métodos, um para cada algoritmo. O primeiro é o *linear_model*, com o submétodo *LogisticRegression*, para criar o modelo de regressão logística (Seção 2.4.2). O segundo método é o *neive_bayes* com o submétodo *MultinomialNB*, para o modelo baseado em *Naïve Bayes* multinomial (Seção 2.4.1). O terceiro é o *SVM* com o submétodo *LinearSVC* para o modelo de máquinas de vetores de suporte (Seção 2.4.4). Finalmente, foi utilizado o método *ensemble* com o submétodo *RandomFlorest* para o florestas aleatórias (Seção 2.4.3). Existem diversos outros métodos e submétodos para outros algoritmos e para outras funcionalidades; assim o *Scikit Learn* é uma ferramenta adequada para se trabalhar com aprendizagem de máquina. Além disso, possui uma ampla documentação[15] que, além de disponibilizar e explicar todos os métodos, traz também um tutorial, não só da biblioteca, mas também da área de aprendizagem de máquinas em geral.

2.6.4 Ferramentas Auxiliares

Nesta Seção é feita uma breve apresentações das demais ferramentas empregadas para o desenvolvimento do projeto.

Natural Liguagem Toolkit (NLTK) : Trata-se de uma biblioteca de código aberto

[12] Disponível em <https://sklearn.org/>
[13] Saco de palavras foi apresentado na Seção (2.3.1)
[14] Método de validação cruzada é explicado na Seção (2.5.1)
[15] Disponível em: <https://scikit-learn.org/stable/modules/classes.html>

em *Python* para o processamento de linguagem natural. Entende-se como linguagem natural aquela que é utilizada na comunicação entre humanos; idiomas como inglês e português. Existem também as linguagens artificiais, que pode ser entendidas como linguagens que não são utilizadas para comunicação entre humanos, como linguagens de programação, linguagem de máquina ou notações matemáticas. De forma ampla o processamento de linguagem natural pode ser entendido como uma forma de manipular a linguagem natural através de computadores (BIRD; KLEIN; LOPER, 2009). As transformações expostas na Seção 2.3 são os processamentos de linguagem natural, em que um texto é transformado em uma estrutura que os algoritmos consigam interpretar. O *nltk* foi utilizado neste projeto no processo de separação dos textos em palavras (*tokenize*) com o intuito de auxiliar na etapa de pré-processamento, através dos métodos *tokenize* e *TweetTokenize*. Foi um grande facilitador, pois caso não fosse empregada a biblioteca, teria que ser criado um trecho de código apenas para essa função.

matplotlib : Biblioteca *Python* para criação de gráficos e visualização de dados [16]. Foi empregada no projeto para criar gráficos da acurácia dos algoritmos em relação aos pré-processamentos e também os gráficos com os resultados das classificações, tanto o total de cada algoritmo quanto a classificação por mês.

seaborn : Trata-se de uma biblioteca de visualização de dados em *Python*, baseado no *matplotlib*. Também foi utilizado para a criação de gráficos, mais especificamente os do tipo pareto que demonstra as palavras mais frequentes nas bases de dados.

punctuation : Um método da biblioteca *string*, nativa do *Python*, que retorna uma lista contendo caracteres de pontuação. Utilizado na criação da lista de *stopwords* utilizada na função que remove caracteres específicos no pré-processamento[17].

re : Módulo *Python* de expressões regulares[18]. Expressões regulares são essencialmente mini linguagens de programação especializadas, inclusa dentro do *Python* por meio do módulo *re*. Tem a função de manipulação de *strings* e foi empregada no projeto no pré-processamento de dados para remover *urls*, *hashtags* e menções dos *tweets*.

wordcloud : Biblioteca empregada na criação das nuvens de palavras[19]

2.7 Trabalhos Relacionados

Análise de sentimentos tem sido bastante explorada em pesquisas acadêmicas, bem como por empresas com interesses comerciais. Isso se deve ao fato de ter um grande

[16] Documentação do *matplotlib* disponível em :<https://matplotlib.org/>
[17] A explicação do que é e como foi criada a lista e a função para remover *stopwords* foi feita na Seção 3.3
[18] Documentação do módulo *re* disponível em: <https://docs.python.org/3.8/howto/regex.html>
[19] Biblioteca *wordcloud* disponível em: <https://pypi.org/project/wordcloud/.>

potencial dado pelo grande volume de mensagens escritas por usuários na internet, em diversos portais como: redes sociais, sites de notícias, blogs, etc. Embora muitos trabalhos foram feitos e estão em andamento, essa ainda é uma área pouco explorada em relação ao volume de dados que é criado diariamente. Nesta Seção em particular são apresentados alguns trabalhos feitos na área e utilizados como base para este TCC.

Corrêa (2017) apresenta em sua monografia um estudo de análise de sentimentos com *tweets* referentes aos filmes que concorreram ao Oscar no ano de 2017. Os *tweets* foram coletados utilizando a mesma ferramenta deste trabalho (*getoldtweets*), inclusive foi por meio desse trabalho o primeiro contato com a ferramenta pelo autor deste. Foi utilizada uma abordagem supervisionada, em que foi feita manualmente a classificação de uma parcela dos dados coletados para compor a base de dados de rotulados. Por meio da etapa de avaliação, o algoritmo *Naïve Bayes* multinomial foi escolhido para classificar a base completa. Com os resultados da classificação, tentou-se prever de quais filmes seriam os prováveis a ganhar o Oscar daquele ano e quais seriam os de menor chance. Não foi obtida uma relação matemática significativa entre o resultado da classificação com os filmes vencedores daquele ano; porém, foi observada uma maior incidência de *tweets* classificados como positivos sobre alguns títulos.

Caracteres especiais que simulam expressões humanas, denominados de *emoticons*, são amplamente utilizados em redes sociais e pode revelar qual o sentimento expresso pelo usuário. Seguindo essa ideia, Gonçalves, Benevenuto e Almeida (2013) propõem um estudo de como *emoticons* são utilizados no *Twitter*, realizando uma análise com diversos *emoticons*. Nesse estudo, uma série de *emoticons* são utilizados com seu significado e suas variações. Em seguida, é feita uma análise que demonstra a quantidade de *emoticons* utilizada por *tweet*. Finalmente, são feitas associações entre pares de *emoticons* (por exemplo, em quantos *tweets* os *emoticons* "A" e "B" ocorrem juntos?).

Os autores também fazem um estudo em relação à frequência de cada *emoticons* com o intuito de revelar os mais utilizados e também uma associação entre eles com *hastag*, demonstrada por nuvens de palavras de modo a revelar qual o sentimento está associado. Por fim, observam como esses *emoticons* aparecem em postagens referente a grandes eventos como tópicos relacionado com tragédias, lançamentos, política, saúde e esporte. O estudo revela que, embora haja limitações, a técnica pode ser utilizada para captar sentimentos expressos no *Twitter*.

Existem diversos métodos para medir sentimentos com diferentes abordagens, tanto baseados em léxico quanto supervisionados, de modo a definir a polaridade de uma sentença. No entanto, não existe um consenso na literatura de qual é o mais eficiente para o problema. Por esse motivo, Araújo, Gonçalves e Benevenuto (2013) apresentam oito diferentes métodos de análise de sentimentos e fazem uma comparação entre eles, levando em consideração a cobertura e a classificação correta. Além disso, desenvolvem um

método que combina sete dos oito avaliados, obtendo bons resultados. Com o trabalho, foi desenvolvida uma aplicação web que permite comparações entre resultados de diferentes métodos de análise de sentimentos, denominado de *iFell*.

Já Silva (2016) explora em sua tese diversos classificadores com abordagem supervisionado através de agregadores de classificadores, também conhecidos como *ensembles*. A ideia é combinar múltiplos classificadores para resolver o mesmo problema, atuando em conjunto como se fossem um único classificador consolidado. Além de realizar um estudo com aprendizagem não supervisionado, que pode melhorar a capacidade de generalização. A combinação de agregadores com abordagens não supervisionada se demonstrou experimentalmente promissora. Para refinar a classificação de sentimentos foi feito o uso de um algoritmo denominado de C^3E (*Consensus between Classification and Clustering Ensemble*), por um procedimento de autotreinamento. Finalmente, realiza-se uma comparação entre os resultados desse método com o classificador SVM, bem como com os melhores resultados da literatura correlata. A aplicação da técnica apresenta-se promissora, pois supera em muitos casos o SVM e alguns trabalhos da literatura.

3 Desenvolvimento e Resultados

Neste capítulo são descritas as etapas de desenvolvimento deste trabalho. Optou-se, nesta monografia, por apresentar o desenvolvimento em conjunto com os resultados de cada etapa, de modo a organizar e justificar as decisões de projeto tomadas. As etapas desenvolvidas estão organizadas em seções da seguinte forma:

Seção 3.1 Coleta de dados — descreve como foram extraídos os dados a serem classificados.

Seção 3.2 Criação da base classificada — descreve como foi criada a base utilizada para treinar e testar os classificadores.

Seção 3.3 Pré-processamento de dados — detalha as técnicas de processamento de texto às quais a base de dados foi submetida, tanto a de treino e teste quanto a coletada. Também apresenta como as técnicas de processamento empregadas afetam as bases de dados.

Seção 3.4 Treinamento e teste dos modelos — descreve como os classificadores foram treinados e testados. Apresenta, também, resultados de acurácia de todos os algoritmos e faz uma comparação entre eles.

Seção 3.5 Experimentos com abordagens de pré-processamento — apresenta experimentos feitos a de modo a verificar o quanto a etapa de pré-processamento influencia no desempenho dos classificadores.

Seção 3.6 Classificação e resultados — após todas as etapas, detalha-se como foi feita a classificação dos dados coletados e o resultado da análise de sentimentos.

3.1 Coleta de dados

Para a coleta de dados da plataforma *Twitter*, foi utilizada a ferramenta *GetOldTweets* (CAMELO, 2017), escrita em linguagem *Python* e que permite extrair *tweets* a partir de uma data específica. Embora o próprio *Twitter* disponibilize uma *API*[1] para fazer essa coleta, a mesma é limitada a *tweets* publicados nos últimos sete dias, o que impossibilitaria coletar todos os dados necessários para este trabalho. O *GetOldTweets* supera a limitação de tempo da *API* do *Twitter*, apresentando uma interface em linha de comando e compatível com versões 2.x do *Python*. Embora seja possível utilizá-la com

[1] <https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets>

as versões 3.x, ela não é oficialmente suportado pelo *Python* (CAMELO, 2017). Como os modelos de análise de sentimentos empregados neste trabalho foram todos desenvolvidos em *Python3*, e o suporte ao *Python2* está oficial suspenso desde janeiro de 2020 (PETERSON, 2008), optou-se por utilizar uma nova versão da ferramenta, denominada *GetOldTweets3* (MOTTL, 2019).

O *GetOldTweets3* é um aprimoramento do *GetOldTweets* original que corrige problemas, adiciona recursos e tem suporte somente ao *Python3* (MOTTL, 2019). Assim, por meio de sua interface por linha de comando, foi possível fazer a coleta de dados. Para a instalação é necessário utilizar o *pip*[2], por meio do comando *"pip install GetOldTweets3"*. Depois, basta acessar a pasta *bin*, na pasta em que a *GetOldTweets3* foi instalado. Diversos argumentos podem ser passados durante a execução do comando *GetOldTweets3*, de modo a refinar o processo de busca. Dentre esses argumentos, destacam-se:

querysearch : Texto para realizar uma busca, sendo que todos os *tweets* que contiverem a *string* informada serão recuperados. Neste trabalho, a *string* utilizada foi; "reforma da previdencia", em todas as buscas. Não é necessário utilizar acentuação no texto do argumento, uma vez que os *tweets* que possuírem a frase da consulta, tendo ou não acentuação, serão ser extraídos.

since : Data inicial da consulta, sendo que a coleta é feita a partir da data informada. Neste trabalho, foram feitas 11 coletas correspondentes aos meses de janeiro a novembro do ano de 2019. Foram, portanto, executadas buscas utilizando 11 diferentes argumentos: "2019-01-01", "2019-02-01", "2019-03-01", etc.

until : Data final da consulta, de modo que a coleta é feita até o dia anterior à data informada. Também foram utilizados 11 argumentos: "2019-02-01", "2019-03-01", "2019-04-01", etc. Ao se utilizar *since* e *until* em conjunto, os primeiros *tweets* extraídos vão ser os do dia anterior em *until* e os últimos da data em *since*, ou seja, a coleta é feita começando da data mais recente (*until*) até a data mais antiga (*since*).

emoji : Argumento relacionado com a extração ou não de *emoticons*. Por padrão, os *emoticons* são descartados; porém, com o argumento "*unicode*", eles são preservados. Aqui, "*unicode*" foi utilizado como parâmetro para todas as consultas, visto que deseja-se realizar uma análise de sentimentos, portanto, *emoticons* são necessários.

output : Nome do arquivo que será criado para armazenar os dados coletados. Por padrão, é salvo um arquivo intitulado "*output.csv*"; porém, por motivos de organização, foi utilizado um parâmetro diferente para cada consulta, referente ao mês da coleta (por exemplo, "*janeiro.csv*").

[2] *pip* é um gerenciador e instalador de pacotes *Python*

Na Figura 12 é mostrado um exemplo de como executar uma busca utilizando esses argumentos. O arquivo gerado após a execução do programa contém diversas informações úteis referentes aos *tweets* coletados, como, por exemplo, data, nome do usuário e o texto compartilhado. Apenas o texto foi utilizado no desenvolvimento deste trabalho.

Figura 12 – Interface de linha de comando da *GetOldTweets3* para coleta de dados referente ao mês de janeiro de 2019.

```
 python3 GetOldTweets3 --querysearch "reforma da previdencia" --since 2019-01-01
--until 2019-02-01 --emoji unicode --output "janeiro.csv"
```

Fonte: o autor

3.1.1 Problemas na fase de coleta de dados

Ao verificar os arquivos criados durante a coleta, observou-se que algumas não estavam completo. Datas referentes a determinados meses terminavam em dias anteriores ao informado no campo "*since*", ou seja, a coleta tinha se encerrado antes do estipulado. Para contornar esse problema, novas coletas foram necessárias. Especificamente, foram realizadas mais quatro coletas, uma para os meses de maio e novembro, e duas para setembro. Antes de realizar as novas consultas, os arquivos foram renomeados manualmente acrescentando o carácter "1" (um) antes da extensão *.csv*, ficando "*maio1.csv*", "*setembro1.csv*" e "*novembro1.csv*".

A coleta referente ao mês de maio foi interrompida no dia 14 e a nova foi realizada tendo esse dia como argumento para os campos "*until*" (ou seja, "2019-05-14"). No campo "*output*" foi utilizado o nome "*maio2.csv*" e os demais argumentos permaneceram iguais. Para setembro, a consulta foi interrompida no dia 26; assim, para foi utilizada a data "2019-09-26" e gerado o arquivo "*setembro2.csv*". Ainda assim, não foram coletados todos os *tweets* restantes de setembro, terminando a busca no dia 22; consequentemente, foi necessário alterar o parâmetro "*until*" para "2019-09-22", armazenando o resultado em "*setembro3.csv*". Já em relação a novembro, apenas mais uma consulta foi necessária: a primeira coleta parou no dia 03 e efetuou-se uma nova busca com os parâmetros "2019-11-03" e "*novembro2.csv*".

Finalmente, para criar um único arquivo para cada um desses meses, implementou-se um pequeno programa *Python*[3], dentro do qual os arquivos são lidos, em um *Data Frame* e concatenados, gerando um novo arquivo *csv* com todos os *tweets* do mês. Um trecho do código referente ao mês de maio é mostrado na Figura 13.

A quantidade de *tweets* de cada mês, após as coletas extras, bem como o total de *tweets* coletados, é apresentada na Tabela 5. A base final possui um total de 980.577 *tweets*

[3] Alguns trechos de códigos e arquivos, como é o caso do código que soluciona o problema da coleta, não estarão disponíveis por conterem dados pessoais de usuários do *Twitter*

Figura 13 – Trecho do código para a criação do arquivo contendo os *tweets* do mês de maio.

```
In [1]: import pandas as pd
        mes = pd.read_csv("dados\\coletados\\maio1.csv")

In [2]: maio = mes
        maio.shape

Out[2]: (94476, 12)

In [3]: mes = pd.read_csv("dados\\coletados\\maio2.csv")
        mes.shape

Out[3]: (51886, 12)

In [4]: maio = pd.concat([maio, mes], ignore_index=True)
        maio.shape

Out[4]: (146362, 12)

In [5]: maio.to_csv("dados\\coletados\\maio.csv", index=False)
```

Fonte: o autor

coletados. Observa-se que o período com maior quantidade de postagens vai de fevereiro a julho de 2019; por outro lado, os meses com menos referências ao tema são os de janeiro de 2019 (com menos de 30.000) e novembro de 2019 (menos de 20.000). É importante observar que a Reforma da Previdência foi aprovada na Câmara dos Deputados, em primeiro turno, no dia 10 de julho de 2019; a votação em segundo turno foi finalizada em 7 de agosto. O período que antecede fim da votação é o que mais concentra *tweets* e após o termino houve uma queda no volume. Uma evidência de como eventos políticos refletem na rede.

Tabela 5 – Quantidade de *tweets* coletados em cada mês e o total.

Janeiro	Fevereiro	Março	Abril	Maio	Junho	Julho	Agosto	Setembro	Outubro	Novembro	Total
29716	104575	148101	134716	146362	104015	141055	40999	46371	65202	19465	980577

Fonte: o autor

3.2 Criação da base classificada

Como os modelos de classificação de dados utilizados neste trabalho são supervisionados, é necessário empregar uma base classificada para as etapas de treinamento e validação. Em vez de classificar manualmente uma amostra dos *tweets* coletados, foram utilizadas bases já classificadas provenientes de outros trabalhos encontrados na literatura. Os *tweets* classificados foram extraídos de dois conjuntos de dados, ambos com três classes para representar os seguintes sentimentos: "positivo", "negativo" e "neutro".

O primeiro conjunto de dados, intitulado *Portuguese Tweets for Sentiment Analysis*[4], foi disponibilizado na plataforma *Kaggle*[5] e possui uma abundância de *tweets* em língua portuguesa já classificados. O conjunto de dados é dividido em quatro arquivos, dois classificados como positivos/negativos e dois classificados como neutros. Os arquivos do tipo positivo/negativo possuem *emoticons*, sendo que um deles se refere a um tema político e o outro não segue um tema específico. Dos neutros, um é proveniente de veículos de comunicação e o outro foi criado a partir de *hashtag*.

Neste trabalho, foram utilizados três dos quatro arquivos (optou-se por descartar o arquivo sem tema). Essa escolha se deve pelo fato de que os *tweets* a serem classificados neste trabalho terem um tema parecido com o conjunto de dados de política. Além disso, não foi utilizado todo o conjunto de dados, apenas uma amostra aleatória foi extraída de cada arquivo. Foi feita uma extração de 2450 amostras positivas e 2450 negativas no arquivo com tema político. Já dos arquivos com neutros, foram utilizados 1225 de cada arquivo, somando também 2450.

Esses valores de amostra foram definidos por dois motivos. Primeiramente, o conjunto total de dados é muito grande e levariam muito tempo para treinar e testar quatro algoritmos diversas vezes; além disso, foram realizados vários experimentos (explicado posteriormente neste capítulo), e em cada um é necessário treinar e testar os modelos. O segundo motivo é que, para manter a proporção de cada classe, a próxima base de dados escolhida para compor a base de treino e teste deve possuir uma proporção de cada classe próxima a este valor.

A segunda base utilizada é proveniente do portal *Minerando Dados*[6] e é composta por *tweets* referentes ao Estado de Minas Gerais, contendo termos políticos e de educação[7]. Essa segunda base foi escolhida por não possuir *emoticons* em todos os "positivos" e "negativos". Existe uma pequena desproporção nos positivos nesse arquivo (ao todo 3300) em relação as demais classes e, para manter a proporcionalidade, foi selecionada, aleatori-

[4] Disponível em: <https://www.kaggle.com/augustop/portuguese-tweets-for-sentiment-analysis>

[5] *Kaggle* é uma comunidade online de cientistas de dados e aprendizado de máquina onde é possível publicar e baixar bases de dados de diversos tipos e modelos de aprendizagem de máquinas

[6] *Minerando dados* é um blog que oferece cursos de mineração de dados e aprendizagem de máquina.

[7] Essa base está disponível em: <https://github.com/minerandodados/mdrepo/blob/master/Tweets_Mg.csv>

amente, uma amostra de 2450 entre os positivos. Para os neutros e negativos, todos foram utilizados. Um código *Python* foi desenvolvido para extrair os *tweets* de cada arquivo[8] e, ao final, criar um novo arquivo que será utilizado para treino e teste dos modelos. A quantidade de *tweets* de cada classe em cada arquivo que foi utilizada para compor a base classificado do projeto, além do arquivo criado com a base classificada, é sumarizada na Tabela 6. A última linha da tabela corresponde ao arquivo criado e utilizado nas próximas etapas, com um total de 14.699 amostras, sendo que cada classe tem uma quantidade similar de dados.

Tabela 6 – tabela com o total de *tweets* classificados de cada arquivo. A coluna "Arquivo" corresponde ao nome do arquivo utilizado, em que os três primeiros faz parte da base *Portuguese Tweets for Sentiment Analysis*, o quarto a base *Tweets_MG* e o quinto a base que criada a partir dos demais arquivos e que vai ser utilizada.

Arquivo	Positivo	Negativo	Neutro	Total
TweetsNeutralHash.csv	0	0	15727	15727
TweetsNeutralNews.csv	0	0	37556	37556
TweetsWithTheme.csv	32744	28847	0	61591
Tweets_Mg.csv	3300	2446	2453	8199
tweets_treino.csv	4900	4896	4903	14699

Fonte: o autor

3.3 Pré-processamento de dados

As técnicas de pré-processamento de dados alteram a base original, com o objetivo de padronizar o texto, eliminar palavras, pontuações ou termos que não têm valor semântico para a classificação (SILVA, 2016). A implementação de todo pré-processamento foi feita através de um código *Python*[9]. Também foi criado um código para realizar uma avaliação nos resultados do processamento de texto, contendo neste amostras, tabelas e nuvens de palavras[10].

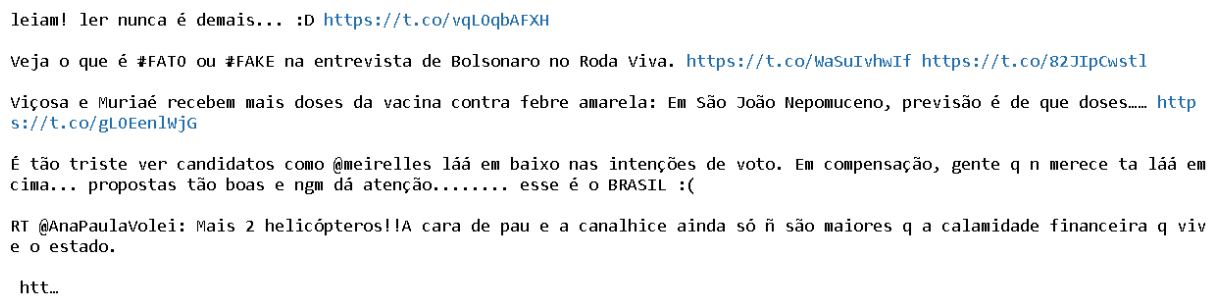
[8] O código de criação da base de treino e teste está disponível em: <https://github.com/RafaelDRicci/PythonSentimentAnalysis/blob/main/BaseTreino.ipynb>

[9] Código referente ao pré-processamento de dados disponível em: <https://github.com/RafaelDRicci/PythonSentimentAnalysis/blob/main/ProcessamentoDados.ipynb>

[10] Código de avaliação do pré-processamento disponível em: <https://github.com/RafaelDRicci/PythonSentimentAnalysis/blob/main/VisualizandoBases.ipynb>

As técnicas de pré-processamento são aplicadas de forma incremental, ou seja, uma técnica é aplicada em cima de uma base já modificada pelas técnicas anteriores. Ao final dois arquivos são gerados, um correspondente com a base de treino e teste, e um com as dos *tweets* coletados. Da base de treino e teste o resultado de todas as etapas são salvos, já na coletada apenas o último resultado é salvo. Para ilustrar como cada técnica afeta a base foi coletada uma amostra aleatória com cinco *tweets* como mostra a Figura 14. As técnicas utilizadas estão listadas adiante.

Figura 14 – Uma amostra de cinco *tweets* da base de dados classificada.



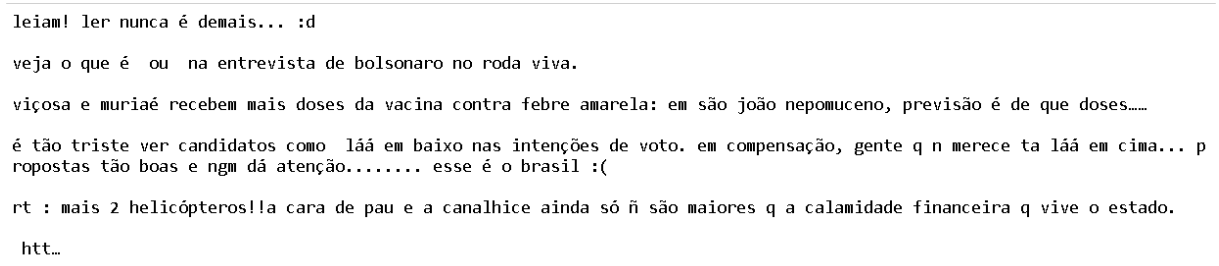
Fonte: o autor

1. Conversão do texto para minúsculo de forma que palavras escritas de formato diferente possam ser reconhecida como iguais.

2. Remoção de *hyperlinks*, *urls*, *hashtags* e menções, visto que não possuem valor semântico para classificação.

   Foram necessárias três funções, uma para remoção de *links*, uma para *urls* e uma para *hashtags*. A Figura 15 apresenta como a amostra da Figura 14 ficou após passar pelas quatro primeiras etapas do pré-processamento.

Figura 15 – Amostra de cinco *tweets* após passar pelas duas primeiras etapas do pré-processamento (*lower case*, *remoção de links*, *urls*, *hashtags* e *menções*).



Fonte: o autor

3. *Tokenização* ou transformar uma frase em uma lista de palavras. Essa função não é exatamente uma técnica de pré-processamento, mas ela auxilia as demais. Após *tokenizar* e realizar a manipulação desejada, é necessário transformar o texto novamente

em uma frase; para isso, foi criada uma função. Foi utilizado o *TweetTokenizer*[11], um modulo do *NLTK* que foi desenvolvido para transformar em *tokens* frases do *Twitter* que reconhece caracteres especiais como *emoticons*. A Figura 16 exibe um exemplo da primeira frase da amostra *tokenizada*.

Figura 16 – Exemplo da primeira frase da amostra, após as primeiras etapas do pré-processamento e *tokenizada*.

```
['leiam', '!', 'ler', 'nunca', 'é', 'demais', '...', ':d']
```

Fonte: o autor

4. Padronização de abreviações. É muito comum a escrita informal no *Twitter* e, com isso, utilizam-se muitas abreviações. Assim, é necessário padronizar os termos para que o classificador não diferencie duas palavras por uma estar abreviada. Por exemplo, a palavra "não" é comumente encontra abreviada como "n" ou "ñ". Apenas uma função foi necessária, empregando um dicionário[12] em que as abreviação são mapeadas de forma que a chave é a abreviação e o valor é a palavra que ela se referente.

   O dicionário foi construído com a experiência do autor deste trabalho, com algumas consultas à base de dados, em conjunto com um artigo informal que lista várias abreviações mais usadas no *Whatsapp* (STEIN, 2019). A função criada pode aceitar qualquer dicionário além do que foi criado para este trabalho.

5. Tratamento de *emoticons*. *Emoticons* são usados para demonstrar ou ressaltar algum sentimento e o uso deles pode alterar o sentimento expresso (GONÇALVES; BENEVENUTO; ALMEIDA, 2013). Para o tratamento de um *emoticon*, de forma parecida com o de abreviações, foi criada uma função com o auxílio de um dicionário. A diferença está na estrutura do dicionário, em que possui apenas três chaves, correspondente a sentimentos positivos, negativos ou neutro e seus valores são três listas de *emoticons*. O que vai alterado na base é que os *emoticon* vão ser substituídos pelas chaves que são; *"emoticon_positivo"* para "Positivos", *"emoticon_negativo"* para "Negativos" e *"emoticon_neutro"* para os "Neutros".

   As listas foram criadas utilizando o trabalho de Araújo, Gonçalves e Benevenuto (2013), que apresenta técnicas de análise de sentimentos. Os autores apresentaram uma tabela com *emoticons* e suas respectivas polaridades. Toda a tabela foi utilizada para compor as listas; além disso, novos *emoticons* foram incluídos, de acordo

[11] <https://www.nltk.org/api/nltk.tokenize.html>

[12] Dicionário (do inglês, *Dictonary*) é um tipo de dado do *Python*, que indexa uma chave a um valor. O valor pode ser acessado diretamente se tiver a chave, sem a necessidade de percorrer todo o dicionário, como em uma lista. Todos dicionários e listas para etapa de pré-processamento pode ser visualizados junto ao código de processamento de texto, disponível em:<https://github.com/RafaelDRicci/PythonSentimentAnalysis/blob/main/ProcessamentoDados.ipynb>.

com a experiência do autor deste trabalho. Foi também necessário incluir novos *emoticons* com caracteres minúsculos, correspondente aos que possuem caracteres maiúsculo. Isso foi necessário devido à primeira etapa deste pré-processamento, ou seja, a transformação de todo o texto para minúsculo.

Também foi utilizado o *ranking* de *emoticons*, tendo como base o trabalho de Kralj Novak et al. (2015), no qual um grupo de pessoas foi selecionado para classificar um grande número de *tweets* em 13 idiomas diferentes. A partir dos *tweets* classificados em "positivos", "negativos" e "neutros", foi possível construir um *ranking* com seus respectivos *emoticons*. O ranking possui ao todo 751 *emoticons*[13] e uma série de análises foi feita para criá-lo. O arquivo utilizado para este trabalho é uma versão sem as análises, possuindo apenas a frequência dos *emoticons* e também o número de ocorrências em cada classes. A Figura 17 exibe os cinco primeiros elementos do *ranking*. Para complementar as listas, o critério escolhido foi o maior número de aparições em uma determinada classe; por exemplo, o primeiro é mais frequente em *tweets* positivos, então vai ser inserido na lista correspondente aos positivos.

Figura 17 – Cinco primeiros *emoticons* do ranking.

	Emoji	Unicode codepoint	Occurrences	Position	Negative	Neutral	Positive	Unicode name	Unicode block
0	😂	0x1f602	14622	0.805101	3614	4163	6845	FACE WITH TEARS OF JOY	Emoticons
1	❤	0x2764	8050	0.746943	355	1334	6361	HEAVY BLACK HEART	Dingbats
2	♥	0x2665	7144	0.753806	252	1942	4950	BLACK HEART SUIT	Miscellaneous Symbols
3	😍	0x1f60d	6359	0.785292	329	1390	4640	SMILING FACE WITH HEART-SHAPED EYES	Emoticons
4	😭	0x1f62d	5526	0.803352	2412	1218	1896	LOUDLY CRYING FACE	Emoticons

Fonte: o autor

6. Remoção de *stopwords*, palavras sem valor semântico para os classificadores. Foi necessária uma função e uma lista. Para a criação da lista foi utilizado o módulo *NLTK* que possui um método para *stopwords* da língua portuguesa. A lista foi incrementada com pontuações, utilizando o *punctuation* do módulo *string* do *Python*, o qual é composto por vários símbolos de pontuação. Além disso, foram adicionadas diversas palavras e símbolos que o autor julgou necessárias, como "reforma" e "previdência". A Figura 18 exibe como a amostra ficou após passar por todas as etapas do pré-processamento.

Comparando a Figura 19, uma nuvem de palavras de toda a base coletada, com a Figura 20, nuvem da mesma base depois do processamento de texto, é possível visualizar o efeito do processamento de texto. A nuvem da Figura 19 apresenta termos com maior evidência como "reforma", "previdência", "da", "de", "que", "na", entre outros. Todas essas palavras são consideradas *stopwords* e são removidas no pré-processamento. Nenhum

[13] O *ranking* de *emojis* pode ser acessado em <http://kt.ijs.si/data/Emoji_sentiment_ranking/>

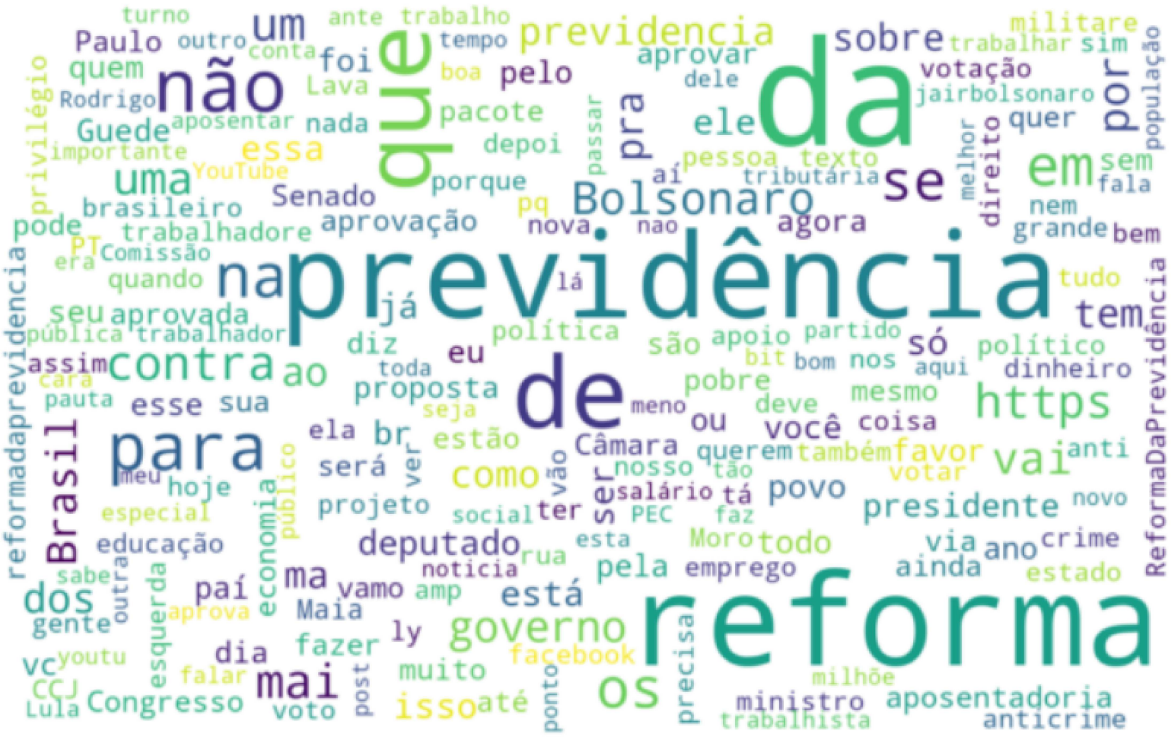
Figura 18 – Amostra aleatória de cinco *tweets* da base de treino após passar por todas as etapas do pré-processamento.

```
leiam ler nunca demais emoticon_positivo

veja entrevista bolsonaro roda viva

viçosa muriaé recebem doses vacina contra febre amarela joão nepomuceno previsão doses

triste ver candidatos láá baixo intenções voto compensação gente merece láá cima propostas boas ninguém dá atenção brasil
emoticon_negativo

2 helicópteros cara pau canalhice ainda maiores calamidade financeira vive estado
```

Fonte: o autor

emoticons aparece na nuvem e termos com mais relevância para a classificação ficam em menor evidência. Já na nuvem da figura 20, termos com mais relevância aparecem com mais destaque como: “contra”, “governo”, “vai”, “aprovar”, “brasil”, etc. Nessa nuvem, aparecem *emoticons* da forma como foram tratados no pré-processamento. É possível observar que termos com mais relevância para o classificador ficam mais em evidência após o pré-processamento. Vale ressaltar que as palavras “reforma” e “previdência” foram incluídas na lista de *stopwords*, pois, como a coleta foi feita utilizando esses termos, todos os *tweets* vão apresentá-los.

Figura 19 – Nuvem de palavras de todos conjunto de dados coletados antes do pré-processamento.



Fonte: o autor

Figura 20 – Nuvem de palavras de todos conjunto de dados coletados após do pré-processamento.

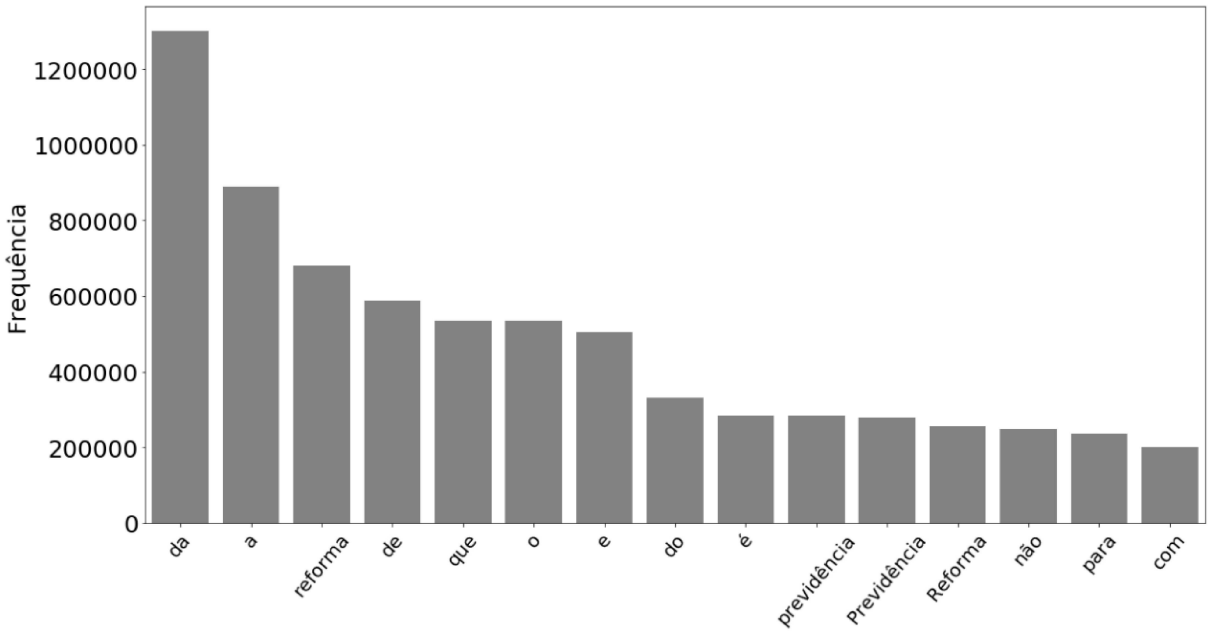
Fonte: o autor

Outra comparação que pode ser feita entre as bases é visualizada nas figuras 21 e 22. Em ambas, são apresentados gráficos do tipo Pareto que demonstram os 15 termos mais frequentes na base: a primeira com os dados brutos e a segunda, com dados processados. Para criar as nuvens de palavras e os gráficos, todas as bases coletadas de cada mês foram concatenados em um único arquivo, na forma de um *Data Frame*.

Analisando o gráfico da Figura 21 fica ainda mais explícita a necessidade do pré-processamento, pois, entre os quinze termos mais frequentes, apenas o "não" tem valor semântico para os classificadores. Outra observação interessante sobre o gráfico, é que a palavra "previdência" apareceu duas vezes, uma com a primeira letra em maiúsculo e outra em minúsculo; assim, o simples fato de converter todo texto para minúsculo já tem um efeito sobre a base. Por outro lado, no gráfico da Figura 22 quase todos, se não todos, os termos mais frequentes possuem importância para o classificador, sendo o que mais se destacou foi a palavra "contra".

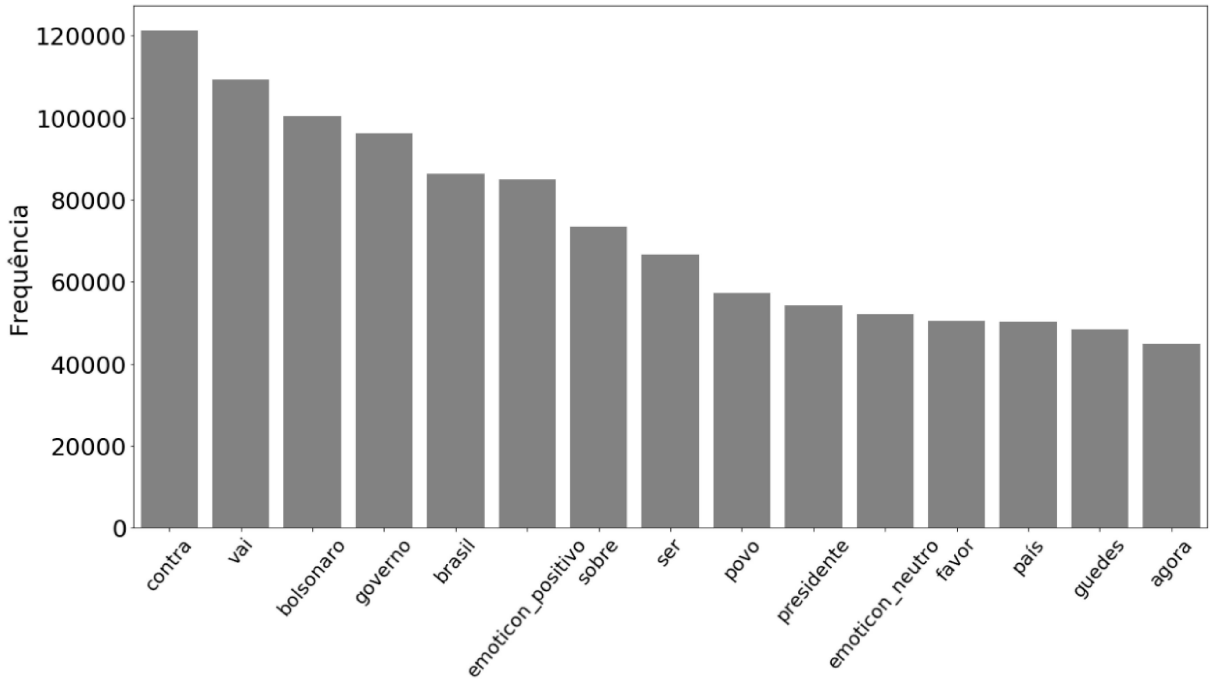
Os quinze termos mais frequentes também estão demonstrados na Tabela 7. Nessa tabela fica mais evidente a quantidade de palavras que foram removidas, já que o termo mais frequente antes do processamento ("da") possui 1.300.416 aparações e o menos frequente, 199.303. Após o pré-processamento, o termo mais frequente passa a ocorrer 121.317 vezes. Deve-se destacar, porém, que essa é uma redução drástica na quantidade

Figura 21 – Gráfico do tipo Pareto dos 15 termos mais frequentes antes do pré-processamento da base coletada.



Fonte: o autor

Figura 22 – Gráfico do tipo Pareto dos 15 termos mais frequentes após do pré-processamento da base coletada.



Fonte: o autor

total de palavras na base de dados, o que pode afetar não só a precisão dos classificadores, mas também o seu desempenho.

Tabela 7 – tabela com os 15 termos mais frequentes, direita antes e a esquerda depois do pré-processamento de texto.

	Palavra	Frequencia
6	da	1300416
17	a	890001
4	reforma	681446
33	de	586116
14	que	535204
37	o	534216
63	e	504002
13	do	331557
9	é	283378
18	previdência	283047
26	Previdência	279466
320	Reforma	254684
252	não	248884
172	para	235515
43	com	199303

	Palavra	Frequencia
178	contra	121317
162	vai	109206
55	bolsonaro	100339
10	governo	96097
182	brasil	86262
0	emoticon_positivo	85040
32	sobre	73502
197	ser	66598
102	povo	57188
260	presidente	54137
739	emoticon_neutro	52030
41	favor	50498
312	país	50341
62	guedes	48384
268	agora	44832

Fonte: o autor

3.4 Treinamento e validação

Como já foi exposto anteriormente, é necessário transformar o texto em uma estrutura que os algoritmos consigam interpretar. Para isso, a biblioteca *Scikit-learn* foi empregada por meio do pacote *feature-extraction* que lida com a extração de recursos em dados brutos. Para este trabalho, utilizou-se o sub-pacote *text*, que é específico para textos;

esse pacote possui alguns métodos interessantes como o *CountVectorizer*, que transforma textos em uma matriz de ocorrências, e o *TfidfTransformer*, que converte uma matriz de ocorrências em uma matriz TF–IFD, além do método *TfidfVectorizer*, que transforma o conjunto de texto diretamente em uma matriz TF–IFD. O *TfidfVectorizer* foi utilizado para transformar as bases, pois já realiza a transformação direta dos textos em TF–IFD, além de ser possível definir o *n*-grams (utilizou-se *n_range* = (1,2) para que uma *feature* corresponde a duas palavras).

A Figura 23 ilustra como é a transformação de um texto em uma matriz de frequência utilizando *CountVectorizer*. Na primeira célula, é criada a matriz de frequências, instanciando-se uma variável que recebeu o nome de "*vectorize*". Esssa variável não é a matriz, mas será utilizada para criá-la. Após isso o método *fit_transform* é invocado para definir as *features*, que ficam armazenadas na variável criada anteriormente. Esse método também cria a matriz, mas é necessário instanciar uma nova variável, no caso matriz. Existem outros métodos, como o *fit*, que apenas define as *features*, e o *transform*, que retorna uma matriz, mas nesse último caso se faz necessário que as *features* já tenham sido definidas anteriormente. O método *transform* vai ser importante para classificar futuras bases com o modelo treinado com a matriz. As células seguintes mostram como fica a estrutura das variáveis criadas: a segunda mostra é a matriz e a terceira, as *features* que a compõem. Cada linha representa um texto e cada valor na matriz a sua frequência, como explicado na Seção 2.3.1.

Figura 23 – Trecho de código que exemplifica como é a criação de um vetor de frequências. Também esta demonstrado como fica a estrutura da matriz após a sua criação.

```
In [1]: from sklearn.feature_extraction.text import CountVectorizer

        corpus = [ 'This is the first document.',
                   'This document is the second document.',
                   'And this is the third one.',
                   'Is this the first document?']

        vectorize = CountVectorizer()
        matriz = vectorize.fit_transform(corpus)

In [2]: matriz.toarray()

Out[2]: array([[0, 1, 1, 1, 0, 0, 1, 0, 1],
               [0, 2, 0, 1, 0, 1, 1, 0, 1],
               [1, 0, 0, 1, 1, 0, 1, 1, 1],
               [0, 1, 1, 1, 0, 0, 1, 0, 1]], dtype=int64)

In [3]: vectorize.get_feature_names()

Out[3]: ['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

Fonte: o autor

Para este trabalho, foi utilizada a matriz TF–IDF; a lógica se mantém praticamente

igual, mudando apenas o módulo utilizado. Outra mudança é que, no momento da criação da primeira variável, é necessário já informar que serão utilizados bigramas. A Figura 23 traz o trecho de código que faz essa transformação.

Figura 24 – Trecho do código que transforma um conjunto de texto em uma matriz TF–IDF n-grams.

```
tfidf_transformer = TfidfVectorizer(ngram_range = (1, 2), analyzer = "word")
X_tfidf = tfidf_transformer.fit_transform(X_treino)
```

Fonte: o autor

Conforme já mencionado, neste trabalho, utilizou-se quatro classificadores (Seção 2.4). O primeiro classificador é o Naïve Bayes (MANNING; RAGHAVAN; SCHÜTZE, 2008; PEDREGOSA et al., 2011), o segundo é um modelo linear de regressão logística (BISHOP, 2006; PEDREGOSA et al., 2011), o terceiro baseia-se em SVM (BISHOP, 2006; PEDREGOSA et al., 2011) e, por fim, o quarto classificador e uma floresta aleatória (MONARD; BARANAUSKAS, 2003; PEDREGOSA et al., 2011). Para treinar os modelos, é necessário informar um conjunto de dados em um formado padronizado, permitindo a interpretação — no caso, uma matriz TF–IDF —, bem como suas classes. Em muitos modelos de aprendizagem de máquina, o conjunto de dados de treino é conhecido como X de treino e as classes desse conjunto como Y de treino.

Após o treinamento, é necessário validar os classificadores. Para isso, foi realizada uma validação cruzada (Seção 2.5.1) *k-fold* com k igual a 10 em todos os modelos. O pacote *scikit-learn* também fornece ferramentas para a validação cruzada por meio do módulo *cross_val_ predict* do pacote *model_selection*, sendo necessário, apenas, passar como parâmetros: 1) o modelo criado e treinado, 2) a base de treino e as classes utilizadas, e 3) o valor de k.

Foram implementadas quatro funções, uma para cada algoritmo, as quais criam, treinam e realizam validação cruzada. Elas possuem uma estrutura bem similar, sendo que recebem como parâmetro os conjuntos X e Y de treino. O X de treino são os textos já transformados e o Y consiste nas classes dos dados de X. Vale ressaltar X terá uma estrutura de uma matriz e Y uma estrutura de lista, sendo que o número de elementos em X (as linhas) tem que ser o mesmo que o número de elementos em Y.

As funções implementadas neste trabalho retornam os modelos criados, com uma variável com os resultados que comprimem o retorno da validação cruzada. As funções em *Python* são exibidas nas figuras 25, 26, 27 e 28.

O diferencial das funções está na de criação dos modelos, sendo que, para o Naïve Bayes e o SVM, basta instanciá-los com seus respectivos métodos. Já para a Regressão Logística, fez-se necessário informar alguns parâmetros como *solver*, que corresponde a um

Figura 25 – Função que cria, treina e realiza validação cruzada em um modelo baseado em Naïve Bayes.

```
#Baseado em Naive Bayes Multinomial (MultinomoalNB)
def _MNB(X_treino, y_treino):

    modelo_MNB = MultinomialNB()
    modelo_MNB.fit(X_treino, y_treino)

    resultados_MNB = cross_val_predict(modelo_MNB, X_treino, y_treino, cv = 10)

    return  modelo_MNB, resultados_MNB
```

Fonte: o autor

Figura 26 – Função que cria, treina e realiza validação cruzada em um modelo baseado em Regressão Logística.

```
#Regressão Logistica (LosgisticRegression)
def _LR(X_treino, y_treino):

    modelo_LR = LogisticRegression(solver='lbfgs', max_iter=200, random_state=322, multi_class="multinomial")
    modelo_LR.fit(X_treino, y_treino)

    resultados_LR = cross_val_predict(modelo_LR, X_treino, y_treino, cv = 10)

    return modelo_LR, resultados_LR
```

Fonte: o autor

Figura 27 – Função que cria, treina e realiza validação cruzada em um modelo baseado em SVM.

```
#Máquinas de Vetores de Suporte(LinearSVC)
def _LSVM(X_treino, y_treino):

    modelo_LSVM = LinearSVC()
    modelo_LSVM.fit(X_treino, y_treino)

    resultados_LSVM = cross_val_predict(modelo_LSVM, X_treino, y_treino, cv = 10)

    return modelo_LSVM, resultados_LSVM
```

Fonte: o autor

Figura 28 – Função que cria, treina e realiza validação cruzada em um modelo baseado em Floresta aleatória.

```
#Florestas Aleatórias(RancomForest)
def _RF(X_treino, y_treino):

    modelo_RF = RandomForestClassifier(n_estimators=100, random_state=142)
    modelo_RF.fit(X_treino, y_treino)

    resultados_RF = cross_val_predict(modelo_RF, X_treino, y_treino, cv = 10)

    return modelo_RF, resultados_RF
```

Fonte: o autor

algoritmo de otimização, no caso o *lbfgs*[14] que lida bem com problemas envolvendo mais de duas classes e com penalidades. O próximo parâmetro é o *max_iter* que corresponde ao número máximo de interações do algoritmo de otimização e *multi_class* como *multinomial* para deixar explícito que se trata de uma problema com mais de duas classes.

Finalmente, para o algoritmo de Floresta Aleatória, foi necessário utilizar o parâmetro *n_estimators*, para o número de árvores, e *random_state*, que é um número escolhido aleatoriamente, usado caso o código seja executado mais de uma vez de modo a gerar a mesma floresta. Esse valor se demonstrou necessário, pois, sem ele, os resultados ficavam diferentes a cada vez que o código era executado, tornando a análise dos resultados mais complexa.

Com os resultados da validação cruzada, é possível extrair diversas métricas de validação. Neste trabalho, utilizou-se apenas a acurácia[15]. Após todo o processamento dos dados de treino e teste, e utilizando bigramas, o Naïve Bayes obteve uma acurácia de 86%, a Regressão Logística chegou a 94%, o SVM e Floresta Aleatória obtiveram ambos 95%. Todos os algoritmos tiveram uma taxa de acerto relativamente alta considerando apenas os valores de suas acurácias, sendo Naïve Bayes a de menor valor. Ainda é possível construir matrizes de confusão para uma análise ainda mais aprofundada das validações. A Figura 29a apresenta a matriz do Naïve Bayes, a Figura 29b, da Regressão Logística, a Figura 29c a de Máquinas de Vetores de Suporte e a Figura 29d a Floresta Aleatória.

Figura 29 – Matrizes de confusão.

(a) Naïve Bayes

Predito / Real	Negativo	Neutro	Positivo	All
Negativo	4617	142	137	4896
Neutro	598	3673	632	4903
Positivo	332	255	4313	4900
All	5547	4070	5082	14699

(b) Regressão Logística

Predito / Real	Negativo	Neutro	Positivo	All
Negativo	4491	377	28	4896
Neutro	141	4594	168	4903
Positivo	9	195	4696	4900
All	4641	5166	4892	14699

(c) SVM

Predito / Real	Negativo	Neutro	Positivo	All
Negativo	4559	314	23	4896
Neutro	102	4620	181	4903
Positivo	10	159	4731	4900
All	4671	5093	4935	14699

(d) Floresta Aleatória

Predito / Real	Negativo	Neutro	Positivo	All
Negativo	4558	322	16	4896
Neutro	108	4641	154	4903
Positivo	15	160	4725	4900
All	4681	5123	4895	14699

Fonte: o autor.

[14] Esse método é conhecido como *Limited-memory BFGS*, um método de otimização não linear (XIAO; WEI; WANG, 2008).

[15] Outras métricas estão disponíveis no arquivo que contém o código-fonte, como *precision*, *recall*, *f1-score*. Essas não foram utilizadas, mas podem ser exploradas em trabalhos futuros para uma comparação mais detalha dos resultados. Código disponível em: <https://github.com/RafaelDRicci/PythonSentimentAnalysis/blob/main/Algoritmos.ipynb>

Para uma melhor interpretação dos valores das matrizes de confusão, foi criada a Tabela 8. O Naïve Bayes foi o que obteve os maiores valores de falso negativo e falso positivo, e os menores verdadeiros neutros e positivos. Por outro lado, foi o que classificou mais como verdadeiro negativo. Já considerando o total de erros e acertos, ele foi o que teve o pior desempenho, já que foi o que menos classificou corretamente — consequentemente, o que mais classificou incorretamente. A Regressão Logística foi o método que obteve o menor valor de verdadeiro negativo e os valores de verdadeiro neutros e positivos foram maiores que os do Naïve Bayes, e menores que os outros dois. Observando os valores totais, também só supera o Naïve Bayes. O SVM teve o praticamento o mesmo valor de verdadeiro negativo que o Floresta Aleatória. Também teve vantagem nos verdadeiros positivos. Observando os totais de acerto e erro, o SVM fica atrás somente do Floresta aleatória. Já o Floresta Aleatória obteve os maiores valores de verdadeiro neutro. Foi o que teve os melhores valores de acerto e erros totais.

Tabela 8 – tabela criado a partir das matrizes de confusão de cada algoritmo. Foram somado os valores de todos as matrizes e concatenados em uma nova tabela.

	Naïve Bayes	Regressão Logística	SVM	Floresta Aleatória
Verdadeiro Negativo	4617	4491	4559	4558
Falso Negativo	930	150	112	123
Verdadeiro Neutro	3673	4594	4620	4641
Falso Neutro	397	572	473	482
Verdadeiro Positivo	4313	4696	4731	4725
Falso Positivo	769	196	204	170
Total de Acertos	12603	13781	13910	13924
Total de Erros	2123	948	789	775

Fonte: o autor.

Observando a Figura 29 e a Tabela 8, fica mais evidente quais algoritmos obtiveram os melhores resultados, sendo possível criar uma ordem, com o Naïve Bayes em quarto, o Regressão em terceiro, SVM em segundo e Floresta Aleatória em primeiro (embora esses três últimos chegaram a resultados bem próximo). Outra observação que se pode extrair das matrizes é como os algoritmos se comportam, observando o total de classificação para cada classe, sem considerar se elas foram corretas ou não. No caso do Naïve Bayes ele foi o que mais classificou *tweets* como positivo e negativo e menos como neutro, e menos como neutros. Já os demais classificaram menos como negativo, com uma distribuição mais homogênea entra as classes, com destaque para a Regressão que foi o que mais classificou neutros.

3.5 Experimentos com abordagens de pré-processamento

Já foi descrito como o pré-processamento afeta a base de dados, com diminuição na quantidade total de palavras e com termos com maior relevância mais frequente após o processamento, mas não como afeta a precisão dos algoritmos. Para isso um experimento foi realizado de moda a constatar qual o efeito do processamento de texto sobre a acurácia dos classificadores. Este experimento ocorreu em conjunto com a etapa de treinamento e todos os passos descritos na Seção 3.4 foram seguidos. As acurácias apresentadas na Seção 3.4 são resultadas da aplicação de todas as técnicas de processamento de texto juntamente com bigramas. Para demonstrar o impacto nos modelos, foi utilizada novamente a acurácia sobre as cada etapa do processamento de texto.

Foram utilizados os arquivos treino e teste gerados na etapa de pré-processamento (Seção 3.3), como visto na Figura 30. Os algoritmos foram treinados, avaliados e suas acurácias extraídas diversas vezes, uma para cada conjunto de texto salvo no arquivo. Todos os quatro algoritmos foram submetidos aos testes. Sempre foi utilizado a estrutura TF–IDF para representar os atributos e só na última etapa que foi utilizado bigramas.

Figura 30 – Arquivo gerado pelo processamento de texto, mostrando os cinco primeiros elementos. A coluna "Text" representa os dados brutos e a "Classificação" a classe daquele *tweet*. As demais colunas correspondem aos resultados das técnicas de processamento.

	Text	Classificacao	URL	Hashtags	Mencoes	Abreviacoes	Emoticons	Stopwords
0	Trio é detido após assaltar pizzaria em Uberlâ...	Positivo	trio é detido após assaltar pizzaria em uberlâ...	trio é detido após assaltar pizzaria em uberlâ...	trio é detido após assaltar pizzaria em uberlâ...	trio é detido após assaltar pizzaria em uberlâ...	trio é detido após assaltar pizzaria em uberlâ...	trio detido após assaltar pizzaria uberlândia ...
1	@PMMG190 - 42º BPM: POLICIAIS MILITARES DE CUR...	Positivo	@pmmg190 - 42º bpm: policiais militares de cur...	@pmmg190 - 42º bpm: policiais militares de cur...	- 42º bpm: policiais militares de curvelo pre...	- 42º bpm : policiais militares de curvelo pre...	- 42º bpm : policiais militares de curvelo pre...	42º bpm policiais militares curvelo prendem tr...
2	Presídio em Minas adota novo modelo e consegue...	Positivo	presídio em minas adota novo modelo e consegue...	presídio em minas adota novo modelo e consegue...	presídio em minas adota novo modelo e consegue...	presídio em minas adota novo modelo e consegue...	presídio em minas adota novo modelo e consegue...	presídio minas adota novo modelo consegue recu...
3	Um jovem é preso, um menor apreendido por tráf...	Positivo	um jovem é preso, um menor apreendido por tráf...	um jovem é preso, um menor apreendido por tráf...	um jovem é preso, um menor apreendido por tráf...	um jovem é preso , um menor apreendido por trá...	um jovem é preso , um menor apreendido por trá...	jovem preso menor apreendido tráfico drogas zo...
4	Trio é preso suspeito de roubo, tráfico e abus...	Positivo	trio é preso suspeito de roubo, tráfico e abus...	trio é preso suspeito de roubo, tráfico e abus...	trio é preso suspeito de roubo, tráfico e abus...	trio é preso suspeito de roubo , tráfico e abu...	trio é preso suspeito de roubo , tráfico e abu...	trio preso suspeito roubo tráfico abuso sexual...

Fonte: o autor

Primeiramente, foram utilizados os dados brutos para treinar e realizar a validação cruzada. Depois computou-se a acurácia e a armazenou seus valores em uma tabela. O mesmo processo de treino, validação cruzada, extração e armazenamento da acurácia foi realizado nas bases das colunas *URL*, *Hashtags*, *Mencoes*, *Abreviacoes*, *Emoticons* e *Stopwords*[16]. Ao final, é repetido o mesmo processo, utilizando novamente a última base, "*stopwords*", que contém o resultado do processamento, mas dessa vez utilizando bigramas. Assim, cria-se uma tabela com a acurácia para cada etapa do processamento para

[16] A acurácia após a conversão do texto para minúsculo não foi calculada, pois, o método que cria a matriz TF–IDF já faz essa conversão automaticamente

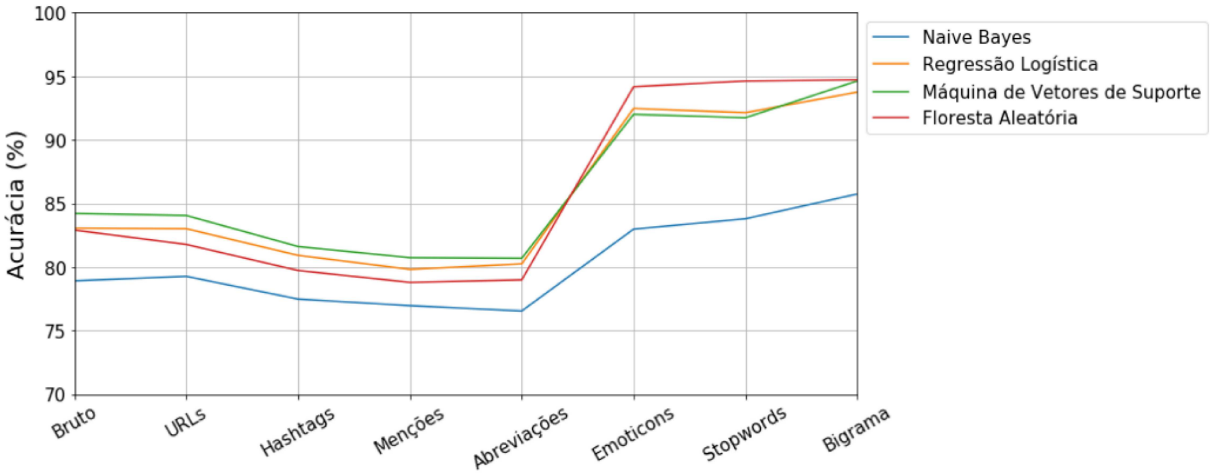
cada algoritmo. A evolução no valor da acurácia pode ser vista na Tabela 9 em que os valores vão de 0 a 1, sendo que, quanto mais próximo de 1, mais *tweets* foram classificados de forma correta, em média. A Figura 31 apresenta um gráfico dos mesmos valores de acurácia, convertidos para porcentagem. É observado, novamente, o mesmo padrão da Seção de treinamento, no qual o Naïve Bayes teve uma desvantagem em relação aos demais. Em todas as etapas do pré-processamento os valores da sua acurácia e sua curva do gráfico ficaram sempre abaixo dos outros algoritmos. Os demais algoritmos obtiveram acurácias e curvas muito próximas. De qualquer modo, todos chegaram a valores satisfatórios no decorrer dos experimentos.

Tabela 9 – tabela com os score dos modelos das etapas de pré-processamento, que por sua vez foram aplicadas de forma cumulativa.

	Naive Bayes	Regressão Logística	Máquina de Vetores de Suporte	Floresta Aleatória
Bruto	0.789101	0.830601	0.842166	0.829036
URLs	0.792639	0.830124	0.840533	0.817743
Hashtags	0.774747	0.809171	0.81611	0.797265
Menções	0.769576	0.79815	0.807334	0.787877
Abreviações	0.765426	0.802504	0.806858	0.78985
Emoticons	0.829784	0.924689	0.920063	0.941765
Stopwords	0.837948	0.921355	0.917409	0.946323
Bigrama	0.857405	0.937547	0.946323	0.947275

Fonte: o autor.

Figura 31 – Gráfico por modelo por processamento. Demonstra como cada etapa do pré-processamento de dados afeta a acurácia dos modelos.



Fonte: o autor.

Outra observação interessante é que os classificadores já possuem uma acurácia razoável com os dados brutos, sendo que os valores foram próximos ou acima de 80%, sendo que o melhor é o SVM, com 84%. Após os primeiros processamentos, todos sofreram uma queda em suas acurácias, chegando a um valor mínimo na etapa de tratamento de abreviação (respectivamente, 76,50%, 80%, 80% e 79% de acurácia).

O tratamento de *emoticon*, por outro lado, surtiu um efeito muito positivo: as acurácias subiram para 83%, 92,50%, 92% e 94%, respectivamente, sendo esse um ponto de virada para o Floresta Aleatória, já que nele o algoritmo supera o Regressão Logística e o SVM. Esse tratamento foi o que mais surtiu efeito nas acurácias geral, com os maiores ganhos para todos os algoritmos, ganhos de cerca de; 7%, 12%, 12% e 15%. Esses ganhos pode ser visto também no gráfico (Figura 31), com uma curva mais acentuada entre os pontos de tratamento de abreviações com o de *emoticon*. Isso evidencia como *emoticon* são muito utilizados para expressar sentimentos, além do fato de serem muito presentes nos dados de treino.

A próxima etapa é a remoção de *stopwords*: esse tratamento levou a um sutil alteração nas acurácias, como uma pequena queda no Regressão e SVM, e um aumento no Naïve Bayes e na Floresta Aleatória. Isso é uma constatação interessante, pois indica que, dependendo da técnica de processamento, o resultado pode diferir para diferentes algoritmos.

Por fim, foi calculada a acurácia utilizando bigramas. Embora não seja, necessariamente, um pré-processamento, ela é uma técnica que visa melhorar a capacidade preditiva dos algoritmos. Isso, porém, não acontece em todos os casos, como mostra a Floresta Aleatória que não teve mudança significativa em sua acurácia. Já os demais tiveram ganho, com destaque para o Naïve Bayes que teve um ganho de quase 2% e para o SVM, cujo ganho foi de quase 3%

3.6 Classificação e Resultados

Com os modelos criados e treinados, focou-se em classificar os *tweets* coletados (Seção 3.1), os quais já passaram por todas as etapas do pré-processamento descritas na Seção 3.3. Com isso, basta carregar o arquivo com o *tweets* processados e os modelos e o vocabulário (*features*) da matriz TF–IDF utilizada em seus treinos. Em seguida, invoca-se o método *transform* para criar uma matriz com a base de dados carregadas e, finalmente, utiliza-se o método *predict* dos modelos, informando a matriz como parâmetro. O método retorna uma lista que cada elemento corresponde a uma classes que, corresponde a uma linha da matriz, ou no caso, um *tweet*. Todo esse procedimento está ilustrado na Figura 32. Fez-se necessário utilizar o método *fillna*, que substitui os elementos vazios por uma *string* sem nenhum caractere (conforme mostrado na terceira linha da segunda célula).

Figura 32 – Trecho de código que carrega os modelos e classifica a base de dados coletada.

```
In [3]: naive_bayes = load('naive_bayes.joblib')
        logistic_regression = load('logistic_regression.joblib')
        linear_SVM = load('linear_SVM.joblib')
        random_forest = load('random_forest.joblib')

In [4]: tfidf_transformer = load('vectorizer.joblib')

        base_coletada = pd.read_csv('dados\\resultados\\base_coletada_processada.csv')
        base_coletada = base_coletada.fillna('')

        X_tfidf = tfidf_transformer.transform(base_coletada['Texto Processado'])

In [5]: y_previsto_MNR =  naive_bayes.predict(X_tfidf)
        y_previsto_LR = logistic_regression.predict(X_tfidf)
        y_previsto_LSVM = linear_SVM.predict(X_tfidf)
        y_previsto_RF = random_forest.predict(X_tfidf)
```

Fonte: o autor

Os resultados foram salvos na forma de um arquivo *csv*, contendo o mês daquele *tweets*, o texto antes e depois do processamento e a classe que cada algoritmo estimou. Na Tabela 10 é mostrado o formato do arquivo salvo com seus cinco primeiros elementos.

Para uma melhor visualização dos resultados, foi criado um *Data Frame* concatenando os resultados de cada classificação com o arquivo contendo os *tweets* coletados. A Tabela 6 ilustra com os cinco primeiros elementos.

Tabela 10 – *Data Frame* criado com os resultados de cada algoritmo com os cinco primeiros elementos.

	Mes	Texto	Texto_Processado	Naive_Bayes	Logistic_Regression	SVM	Random_Forest
0	Janeiro	👍E jornalista sabichão apoiando reforma fasci...	emoticon_positivo jornalista sabichão apoian...	Positivo	Positivo	Positivo	Positivo
1	Janeiro	Isso é só uma demostração do que virar quando ...	demostração virar chegar congresso	Negativo	Neutro	Neutro	Neutro
2	Janeiro	Governo diz que espera aprovar reforma da Prev...	governo diz espera aprovar 1º semestre	Neutro	Neutro	Neutro	Neutro
3	Janeiro	Joice, mudadando de assunto, é verdade q o gov...	joice mudadando assunto verdade governo pedind...	Negativo	Negativo	Negativo	Negativo
4	Janeiro	meu pai foi muito bolsominion arrepedindo fala...	pai bolsominion arrepedindo falando discurso m...	Positivo	Positivo	Positivo	Neutro

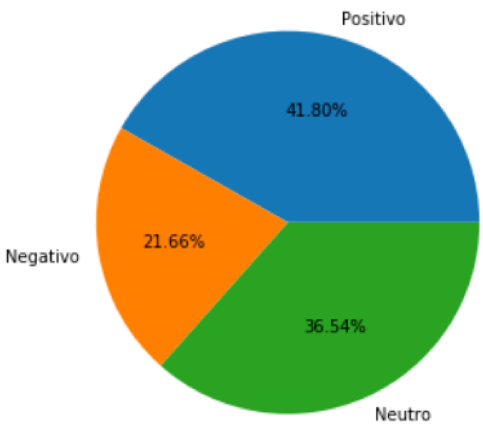
Fonte: o autor

Deve-se ressaltar que, ao todo, 980.577 *tweets* foram classificados e que foi feita uma contagem da quantidade de cada classe por classificador. No total, o modelo baseado em Naïve Bayes classificou 409.855 (41,80% das amostras) como positivos, 358.335 (36,54%) neutros e 212.387 (21,66%) negativos. O Regressão Logística classificou 209.452 (21,36%) como positivos, 724.700 (73,91%) como neutros e 46.425 (4,73%) como negativos. Máquinas de Vetores de Suporte classificou 222.859 (22,73%) como positivos, 704.257 (71,82%) como neutro e 53.461 (5,45%) como negativo. Por fim o Floresta Aleatória classificou como positivo 69.066 (7,04%), 842.722 (85,94%) como neutro e 68.789 (7,02%)

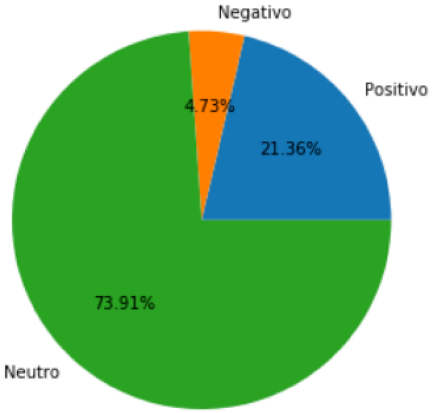
como negativo. As Figuras 33a, 33b, 33c e 33d apresentam gráficos de cada modelo com a porcentagem do total dos *tweets* classificados.

Figura 33 – Gráficos da porcentagem de cada classificador.

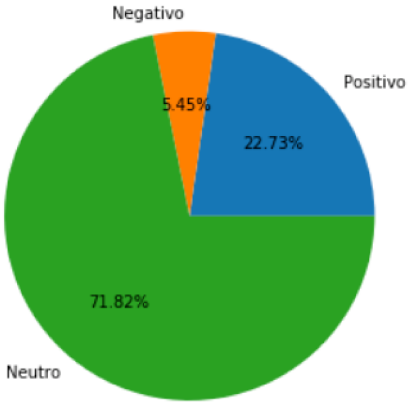
(a) Naïve Bayes



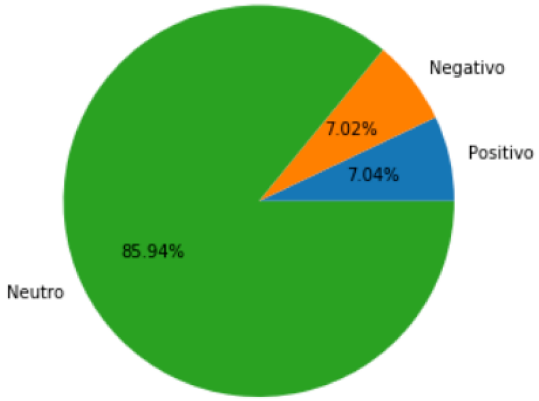
(b) Regressão Logística



(c) SVM



(d) Floresta Aleatória



Fonte: o autor.

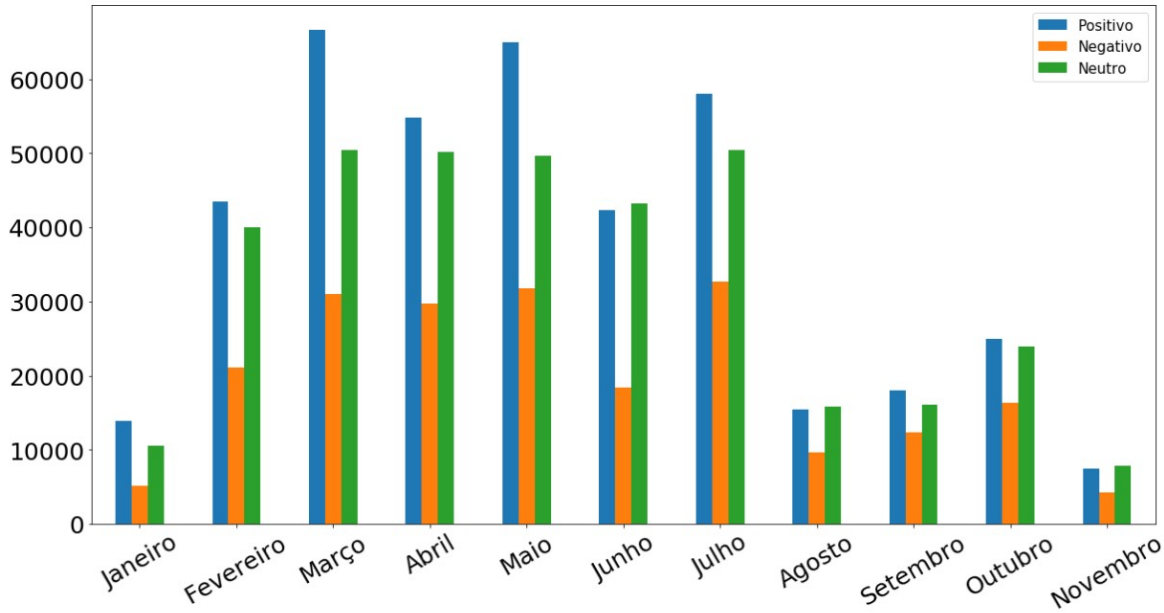
Comparando os resultados dos classificadores, é notável o quão diferentes ficaram alguns dos resultados. O Naïve Bayes classificou mais *tweets* como positivos e os demais como neutros. A classificação pelo Naïve Bayes teve uma melhor distribuição entre as classes, já os demais predominou neutros. Isso é uma grande diferença, considerando que foi utilizada a mesma base para treino e que o Naïve Bayes obteve a menor acurácia nos testes. Comparando a Regressão com o SVM, percebe-se resultados muito próximos, com ambos tendo uma maior porcentagem de positivos que negativos. Já a Floresta Aleatória teve uma maior proporção entre negativos e positivos, uma porcentagem que pode ser considerado empate técnico. Esse último também foi o que mais classificou neutros. Isso deixa claro como os classificadores diferem em sua estrutura, culminando em resultados distintos. Não é possível, neste trabalho, definir qual classificou mais corretamente, visto que esses *tweets* não possuem uma classificação prévia, mas levando em consideração os

experimentos da Seção 3.5, em que o Regressão Logística, o Máquinas de Vetores de Suporte e o Floresta Aleatória obtiveram os maiores valores de acurácia e classificaram mais como "neutros", pode-se concluir que houve uma predominância de *tweets* "neutros".

Para uma melhor visualização da classificação, elas foram dividas por mês com gráficos, um para cada algoritmo. O gráfico da Figura 34 representa o Naïve Bayes, a Figura 35 o Regressão Logística, já a Figura 36 Máquinas de Vetores de Suporte e por fim a Figura 37 Floresta Aleatória. As tabelas 11, 12, 13 e 14 também apresentam os *tweets* classificados em cada mês.

Com os gráficos e as tabelas, é possível extrair mais algumas informações. Primeiro, os meses de fevereiro a julho concentram a maior quantidade de *tweets*; isso demonstra que esse período do ano de 2019 foi o que mais os usuários do *Twitter* comentaram sobre o tema. Outra observação é que o padrão do total de classes se repetem para na maior parte dos meses em ambos os algoritmos. Uma observação é o quão semelhante foi a classificação do Regressão Logística com o SVM. Outra interessante é que no caso do Floresta Aleatória, ouve em alguns meses uma predominância de negativos sobre os positivos, em outros positivos sobressaíram e empate em outros, mas na média a proporção foi muito próxima.

Figura 34 – Gráfico de barras com a quantidade a classificação separados por mês Naïve Bayes.



Fonte: o autor

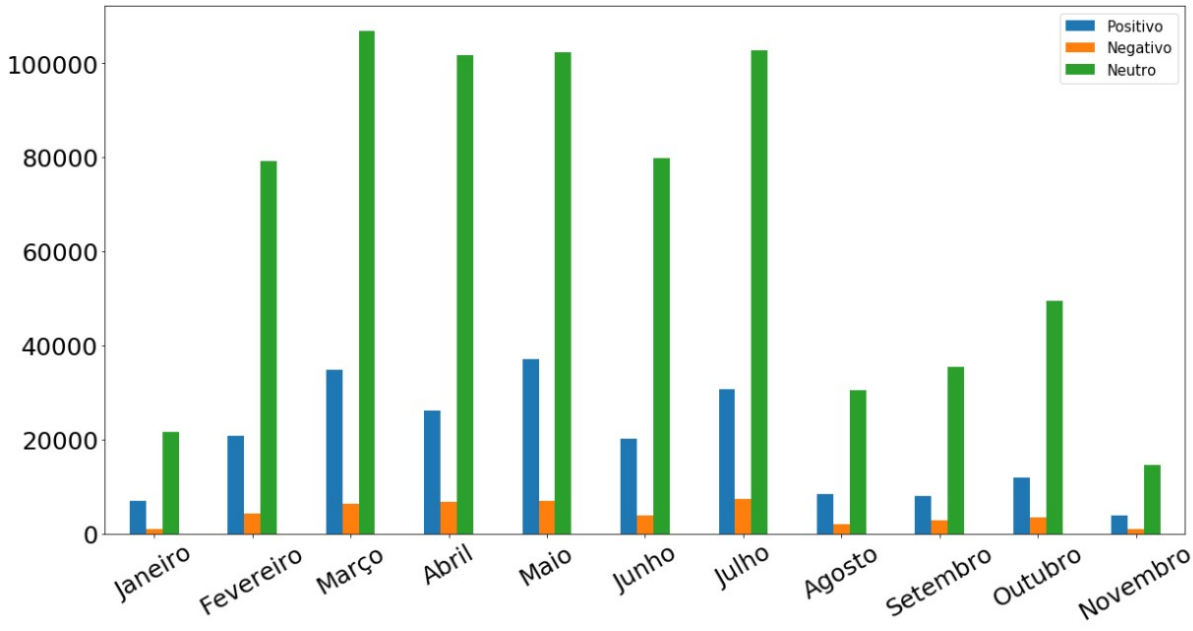
Para uma comparação ainda mais direta entre os algoritmos foram criadas algumas tabelas no formato de matriz (muito com uma matriz de confusão) que compara classificações entre dois algoritmos. Tomando a Tabela 15a como exemplo, que compara

Tabela 11 – tabela com a classificação por mês pelo Naïve Bayes.

	Positivo	Negativo	Neutro
Janeiro	13942	5180	10594
Fevereiro	43479	21083	40013
Março	66616	31014	50471
Abril	54775	29773	50168
Maio	64925	31758	49679
Junho	42337	18447	43231
Julho	57968	32676	50411
Agosto	15464	9671	15864
Setembro	17950	12309	16112
Outubro	24940	16293	23969
Novembro	7459	4183	7823

Fonte: o autor

Figura 35 – Gráfico de barras com a quantidade a classificação separados por mês pelo Regressão Logística.



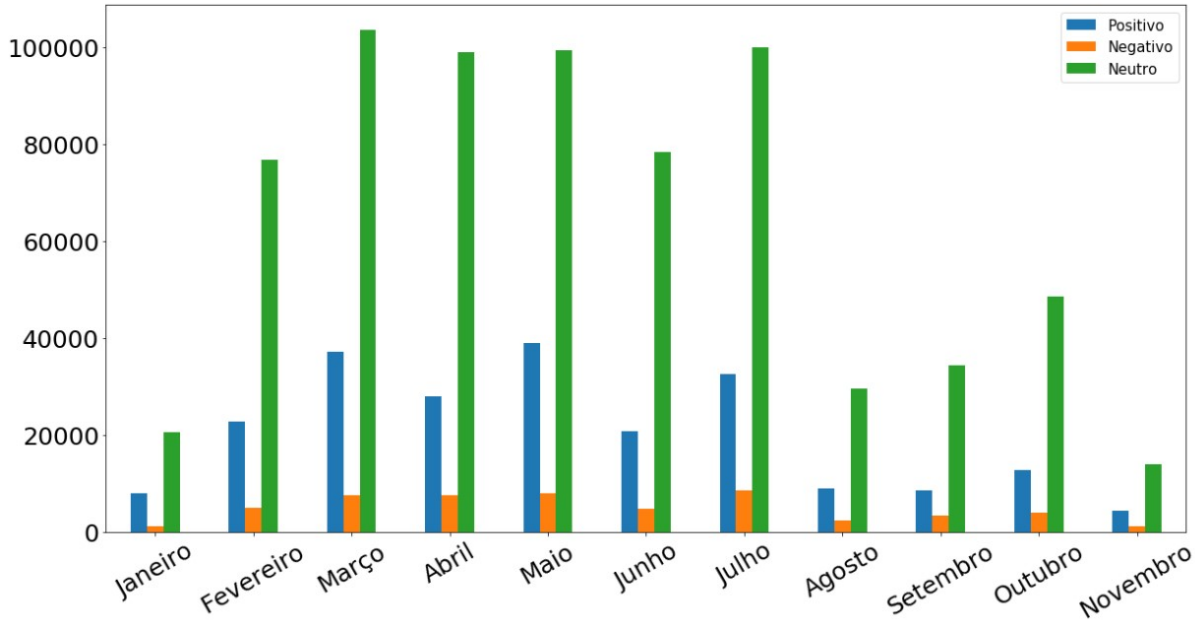
Fonte: o autor

Tabela 12 – tabela com a classificação por mês pelo Regressão Logística.

	Positivo	Negativo	Neutro
Janeiro	7070	978	21668
Fevereiro	20903	4382	79290
Março	34876	6425	106800
Abril	26272	6770	101674
Maio	37052	7051	102259
Junho	20110	3995	79910
Julho	30825	7419	102811
Agosto	8481	2043	30475
Setembro	7976	2855	35540
Outubro	12046	3561	49595
Novembro	3841	946	14678

Fonte: o autor

Figura 36 – Gráfico de barras com a quantidade a classificação separados por mês pelo Máquinas de Vetores de Suporte.



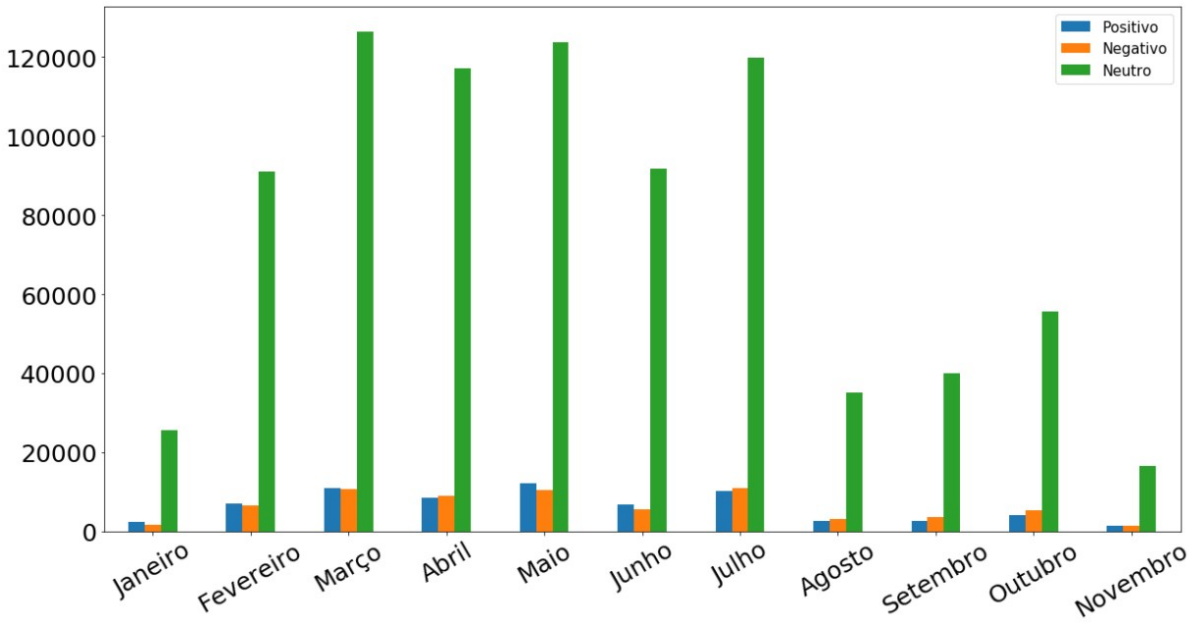
Fonte: o autor

Tabela 13 – tabela com a classificação por mês pelo Máquinas de Vetores de Suporte.

	Positivo	Negativo	Neutro
Janeiro	7891	1142	20683
Fevereiro	22760	5075	76740
Março	37098	7496	103507
Abril	28050	7656	99010
Maio	38984	7973	99405
Junho	20856	4725	78434
Julho	32506	8593	99956
Agosto	9078	2337	29584
Setembro	8579	3399	34393
Outubro	12718	3919	48565
Novembro	4340	1146	13979

Fonte: o autor

Figura 37 – Gráfico de barras com a quantidade a classificação separados por mês pelo Floresta Aleatória.



Fonte: o autor

Tabela 14 – tabela com a classificação por mês Floresta Aleatória.

	Positivo	Negativo	Neutro
Janeiro	2347	1627	25742
Fevereiro	6978	6509	91088
Março	10942	10773	126386
Abril	8545	9064	117107
Maio	12254	10422	123686
Junho	6742	5596	91677
Julho	10253	11061	119741
Agosto	2757	3211	35031
Setembro	2726	3595	40050
Outubro	4163	5428	55611
Novembro	1359	1503	16603

Fonte: o autor

a classificação gerada pelo Naïve Bayes com a do Regressão Logística[17]. Não só o total de cada algoritmo fica evidenciado mas também a relação das classes, como quantos foram classificados como positivo por um e como negativo pelo outro, ou quantos foram classificados igualmente por ambos.

Na Tabela 15a, as linhas representam os classificados pelo Naïve Bayes e as colunas pelo Regressão Logística. As posições da matriz são as quantidades classificadas por algoritmo, por exemplo, a primeira posição da matriz ("*Positivo*", "*Positivo*") significa que ambos classificaram os mesmo *tweets* (160291) como positivos, a próxima posição ("*Positivo*", "*Negativo*") significa que o Naïve Bayes classificou 2682 como positivos e o Regressão Logística classificou os mesmo como negativo. Os elementos com "*Total*" representa a quantidade total daquela classe de um classificador, por exemplo, o elemento ("*Total*", "*Positivo*") é a quantidade de elementos positivos classificados pelo Regressão Logística. Para uma melhorar a comparação entre os algoritmos, os *tweets* classificados da mesma forma por dois, ou seja, as posições ("*Positivo*" "*Positivo*"), ("*Negativo*" "*Negativo*") e ("*Neutro*" "*Neutro*") foram somadas em cada tabela e calculada a porcentagem em ralação ao total de *tweets*. Já para a quantidade de classificações diferentes entre dois algoritmos, bastou subtrair a quantidade de classificações iguais do total de *tweets*.

Analisando a Tabela 15a, ao todo 549.005 (56%) *tweets* receberam as mesmas classes pelos algoritmos Naïve Bayes e Regressão Logística, enquanto 431.572 (44%) receberam classificações distintas. Já a Tabela 15b compara o Naïve Bayes com o Máquinas

[17] Essas tabelas que comparam os resultados entre dois algoritmos são referenciadas como matriz de resultados.

de vetores de Suporte. Os *tweets* classificados da mesma forma somam-se 555.789 (57%), e de forma diferente 424.788 (43%). Comparando o Navve Bayes com o Floresta Aleatória, por meio da Tabela 15c, classificados iguais somam-se 447.657 (46%) e diferentes 532.920 (54%). A Tabela 15d compara o Regressão Logística com Máquinas de Vetores de Suporte, que classificaram ao todo 893.725 (91%) de forma igual e 86.852 (9%) diferentes. Regressão com Florestas, demonstrada na Tabela 15e, classificaram de forma igual 789.320 (80%) e de forma diferente 191.257 (20%). Por fim a Tabela 15f compara o Máquinas de Vetores com o Floresta que classificaram da mesma forma 764.953 (78%) e de forma distinta 215.624 (22%).

Com essas tabelas, ficam nítidas as semelhanças e diferenças entre os resultados gerados pelos algoritmos. O Naïve Bayes foi o que gerou os resultados mais distintos entre os outros, com destaque com o Floresta Aleatória em que mais da metade (54%) dos *tweets* receberam classificações diferentes pelos algoritmos. Já a semelhança nas classificações entre os demais foi maior, com destaque para o Regressão Logística e o Máquinas de Vetores de Suporte, em que uma quantidade muito alto de *tweets* (91%) receberam a mesma classificação pelos dois algoritmos.

Pelos resultados das classificações, com três dos quatro algoritmos predominando mais classes "neutra" e apenas o Naïve Bayes com predominância de "positivos", levando em consideração que ele foi o algoritmo que teve os valores de acurácia mais baixo nos experimentos na Seção 3.5, conclui-se que a maior parte dos *tweets* são "neutros". Isso pode ser uma evidência de que a rede foi mais utilizada para publicação de notícias de cunho informativo do que de cunho opinativo.

Tabela 15 – Matrizes de comparação.

(a) Naïve Bayes (linhas) × Regressão Logística (colunas).

	Positivo	Negativo	Neutro	Total
Positivo	160291	2682	246882	409855
Negativo	38175	42554	131658	212387
Neutro	10986	1189	346160	358335
Total	209452	46425	724700	980577

(b) Naïve Bayes (linhas) × SVM (colunas).

	Positivo	Negativo	Neutro	Total
Positivo	166287	6648	236920	409855
Negativo	44257	45148	122982	212387
Neutro	12316	1665	344354	358335
Total	222860	53461	704256	980577

(c) Naïve Bayes (linhas) × Floresta Aleatória (colunas).

	Positivo	Negativo	Neutro	Total
Positivo	51814	20386	337655	409855
Negativo	11665	45749	154973	212387
Neutro	5587	2654	350094	358335
Total	69066	68789	842722	980577

(d) Regressão Logística (linhas) × SVM (colunas).

	Positivo	Negativo	Neutro	Total
Positivo	178959	7411	23082	209452
Negativo	6923	36547	2955	46425
Neutro	36978	9503	678219	724700
Total	222860	53461	704256	980577

(e) Regressão Logística (linhas) × Floresta Aleatória (colunas).

	Positivo	Negativo	Neutro	Total
Positivo	58823	26811	123818	209452
Negativo	2966	27526	15933	46425
Neutro	7277	14452	702971	724700
Total	69066	68789	842722	980577

(f) SVM (linhas) × Floresta Aleatória (colunas).

	Positivo	Negativo	Neutro	Total
Positivo	56341	29348	137171	222860
Negativo	3299	26612	23551	53462
Neutro	9426	12829	682000	704255
Total	69066	68789	842722	980577

Fonte: o autor.

4 Conclusão

Este trabalho de conclusão de curso teve como objetivo explorar conceitos da Análise de Sentimentos em redes sociais. Nesse cotexto, o *Twitter* foi escolhido para compor a base de dados a ser classificada por suas vantagens e facilidades. Um objetivo secundário foi explorar a linguagem de programação *Python* para lidar com problemas de análise de sentimentos e aprendizagem de máquina.

Em paralelo ao desenvolvimento tradicional de um projeto de análise de sentimentos, foram feitos diversos experimentos com o objetivo de explorar e testar etapas de desenvolvimento. O primeiro que se destaca é a comparação de algoritmos de aprendizagem de máquina, utilizados para classificar os textos. Outro conceito que foi explorado é o pré-processamento de texto, que tradicionalmente pretende otimizar a classificação. Por fim, os algoritmos foram utilizados para classificar a base extraída do *Twitter* e os resultados gerados comparados.

Ao final da coleta 980.577 *tweets* foram extraídos no ano de 2019, entre os meses de janeiro a novembro. Os meses de fevereiro a julho foram os que mais concentraram *tweets*, sendo que nesse mesmo período a reforma foi votada na Câmara dos Deputados, evidenciando que assuntos políticos se refletem na rede.

Em relação à comparação entre algoritmos, algumas observações interessantes foram constatada. A primeira é que nem sempre a técnica de pré-processamento aplicada surte um efeito positivo se analisada isoladamente. Algumas técnicas como a remoção de *URLs* surtiram efeitos negativos nos quatro algoritmos utilizados, já outras técnicas como o tratamento de *emoticons* surtiram um efeito positivo, com ganho significativo nas acurácias de todos. Vale ressaltar que os dados utilizados no treinamento não correspondem a uma amostra dos dados coletados, mas sim foram criados a partir de outras bases já classificadas. Essa é uma observação importante, pois uma dessas bases contém elevado número de *emoticons*. Isso revela que determinados tratamentos de textos vão surtir efeitos variados dependendo da base.

Os testes das técnicas de processamento de texto também revelaram que diferentes técnicas surtem efeitos distintos nos algoritmos. Como, por exemplo, no tratamento de *emoticons* que surtiu efeitos positivos em todas as acurácias, em alguns casos o ganho foi maior que em outros. O algoritmo Floresta Aleatória teve o maior ganho de todos (cerca de 12,2%) com esta técnica, sendo este um ponto que a acurácia do algoritmo supera as dos demais. Já no caso da remoção de *stopwords*, os algoritmos Regressão Logística e SVM tiveram queda de acurácia já o Naïve Bayes e o Floresta Aleatória tiveram ganho. Isso revela que também o tipo de tratamento específico afeta de formas diferentes algoritmos

diferentes.

Após a classificação dos *tweets* coletados, os resultados dos algoritmos foram analisados e comparados. Dessa forma foi possível realizar uma comparação entre os algoritmos através de quantos foram classificados igualmente por dois algoritmos. As comparações dos resultados revelaram que entre os algoritmos o que gerou os resultados mais diferentes foi o Naïve Bayes, com uma distribuição regular entre as classes com uma pequena vantagem da classe "positivo". Os outros 3 geraram resultados muito próximos, com maior predominância de "neutros", com destaque para o SVM e o Regressão que tiveram resultados muito semelhantes, com 91% dos *tweets* receberam a mesma classificação.

4.1 Contribuições

Como principais contribuições deste trabalho, destacam-se:

1. Criação de uma base de dados composta por 980.577 *tweets* a respeito da "reforma da previdência" do ano de 2019. A base foi classificada por quatro algoritmos diferentes. Além disso, a base contem dois conjuntos de textos, um antes e um depois do processamento de texto. Também possui o mês correspondente a publicação.

2. Criação de um base classificada a partir de bases de terceiros, para compor a base utilizada no treinamento e nos testes. A base é composta ao todo de 14.699 *tweets*.

3. Estudo e apresentação da linguagem de programação *Python* para lidar com problemas relacionado a aprendizagem de máquina, mineração de dados e análise de sentimentos. Diversas de ferramentes e bibliotecas da linguagem foram apresentadas e utilizadas.

4. Uma série de testes com o objetivo de demonstrar o efeito de técnicas e etapas a respeito da análise de sentimentos. Com os testes fica em evidência como o processamento de texto afeta algoritmos e a bases de dados. Também foi testado e comparados algoritmos, que revelaram o com maior eficiência e como cada técnica de processamento afeta cada um. Por fim os algoritmos classificaram a base coletada e seus resultados comparados, que revelaram semelhanças e diferenças entre eles.

5. Criação de um projeto de análise de sentimentos em conjunto com teste, disponível para ser utilizados em trabalhos futuros.

4.2 Trabalhos Futuros

Neste trabalho foram aplicadas diversas técnicas de processamento de texto, mas existem outras que podem ser exploradas. Uma que não foi explorada é a remoção de *tweets*

duplicados: é muito comum, na rede, que usuários republicarem ou compartilharem *tweets* já postados. Outra ideia é identificação de robôs que postam mensagens (os chamados *bots*). Programas como esses podem influenciar os classificadores; pode-se, assim, remover suas postagens e reavaliar os classificadores.

Também é interessante realizar uma análise aprofundada nas técnicas de processamento de textos que surtiram efeitos negativos em todos os algoritmos, com o intuito de avaliar o motivo dessa piora. Finalmente, pretende-se classificar manualmente uma amostragem dos dados coletados. Dessa forma, cria-se uma nova base de treinamento dos algoritmos, substituindo a utilizada, e torna-se possível uma nova comparação entre os algoritmos. Também é possível utilizar outros algoritmos e métodos, como abordagens não-supervisionadas, para enriquecer a comparação. Nesse contexto, é possível ainda utilizar mais métricas de validação para comparações e avaliações mais aprofundadas, pois apenas a acurácia foi considerada aqui como medida de avaliação de desempenho. Algumas já estão disponíveis no arquivo de código que treina e testa os modelos, métricas como *precision*, *recall* e *f1-score*, estão todas disponíveis para serem exploradas.

Finalmente, espera-se fazer outras classificações, diferentes da tradicionalmente usada em análise de sentimento ("positivo" ou "negativo"). Classes que transmita a ideia do texto ser "contra" ou "a favor" a respeito da "reforma da previdência" por exemplo. Isso porque, ao analisar os *tweets* coletados, existem muitos casos que o sentimento transmitido é "negativo", mas não necessariamente transmite a ideia que o usuário é "contra" a proposta. Essas duas categorias de classificação não transmitem a mesma ideia. Para isso seria também necessário uma classificação manual de uma amostragem da base coletada. Assim, a análise de sentimentos pode ser utilizar forma como pesquisas de opiniões.

Referências

ARAÚJO, M.; GONÇALVES, P.; BENEVENUTO, F. Measuring sentiments in online social networks. In: *Proceedings of the 19th Brazilian Symposium on Multimedia and the Web*. New York, NY, USA: Association for Computing Machinery, 2013. p. 97–104. Citado 3 vezes nas páginas 12, 33 e 42.

ASLAM, S. *Twitter by the Numbers: Stats, Demographics & Fun Facts*. 2020. Omnicore. Disponível em: <https://www.omnicoreagency.com/twitter-statistics/>. Acesso em: 12 nov. 2020. Citado 2 vezes nas páginas 10 e 11.

BIRD, S.; KLEIN, E.; LOPER, E. *Natural Language Processing with Python*: Analyzing text with the natural language toolkit. Sebastopol, CA, USA: O'Reilly Media, 2009. 803 p. Citado 2 vezes nas páginas 16 e 32.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer Science+Business Media, 2006. 738 p. (Information Science and Statistics). Citado 2 vezes nas páginas 14 e 49.

BORDIN JUNIOR, A. *Aplicação de programação genética na análise de sentimentos*. 142 f. Dissertação (Mestrado) — Universidade Federal de Goiás, Goiânia, GO, 2018. Citado 2 vezes nas páginas 17 e 18.

CAMELO, J. H. *GetOldTweets-python*. 2017. Disponível em: <https://github.com/Jefferson-Henrique/GetOldTweets-python>. Acesso em: 13 dez. 2019. Citado 3 vezes nas páginas 13, 35 e 36.

CORRÊA, I. T. *Análise dos sentimentos expressos na rede social Twitter em relação aos filmes indicados ao Oscar 2017*. 72 f. Monografia (Graduação em Sistemas de Informação) — Universidade Federal Uberlândia, Uberlândia, MG, 2017. Citado 7 vezes nas páginas 10, 11, 18, 26, 27, 28 e 33.

FIGUEIRA, C. V. *Modelos de regressão logística*. 138 p. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, 2006. Citado na página 18.

G1. *Saiba o que muda com a reforma da Previdência*. Globo Comunicação e Participações S.A., 2019. G1 Economia. Disponível em: <https://g1.globo.com/economia/noticia/2019/11/12/saiba-o-que-muda-com-a-reforma-da-previdencia.ghtml>. Acesso em: 06 nov. 2020. Citado 2 vezes nas páginas 11 e 12.

GONÇALVES, P.; BENEVENUTO, F.; ALMEIDA, V. O que tweets contendo emoticons podem revelar sobre sentimentos coletivos? In: *Anais do II Brazilian Workshop on Social Network Analysis and Mining*. Porto Alegre, RS: Sociedade Brasileira de Computação, 2013. p. 128–139. Citado 2 vezes nas páginas 33 e 42.

GONÇALVES, T. *Teorema de Bayes: o que é e qual sua aplicação?* 2020. Voitto. Disponível em: <https://www.voitto.com.br/blog/artigo/teorema-de-bayes>. Acesso em: 05 nov. 2020. Citado na página 18.

HALL, M.; FRANK, E.; HOLMES, G.; PFAHRINGER, B.; REUTEMANN, P.; WITTEN, I. H. The Weka data mining software: An update. *SIGKDD Explor. Newsl.*, Association for Computing Machinery, New York, NY, USA, v. 11, n. 1, p. 1018, nov. 2009. ISSN 1931-0145. Citado na página 28.

HUNTER, J. D. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, v. 9, n. 3, p. 90–95, 2007. Citado na página 29.

KRALJ NOVAK, P.; SMAILOVIĆ, J.; SLUBAN, B.; MOZETIČ, I. Sentiment of emojis. *PLoS ONE*, v. 10, n. 12, p. e0144296, 2015. Citado na página 43.

LIU, B. *Sentiment Analysis and Opinion Mining*. San Rafael, CA: Morgan & Claypool, 2012. 168 p. Citado na página 15.

MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. *Introduction to Information Retrieval*. Cambridge, MA, USA: Cambridge University Press, 2008. 544 p. Citado na página 49.

MCKINNEY, W. *pandas: powerful Python data analysis toolkit*. 2020. Disponível em: <https://pandas.pydata.org/docs/pandas.pdf>. Acesso em: 03 nov. 2020. Citado na página 30.

MONARD, M. C.; BARANAUSKAS, J. A. Indução de regras e árvores de decisão. In: REZENDE, S. O. (Ed.). *Sistemas Inteligentes: Fundamentos e Aplicações*. 1. ed. Barueri,SP: Manole Ltda, 2003. p. 115–140. ISBN 85-204-168. Citado 3 vezes nas páginas 19, 21 e 49.

MOTTL, D. *GetOldTweets3*. 2019. Disponível em: <https://github.com/Mottl/GetOldTweets3>. Acesso em: 13 dez. 2019. Citado na página 36.

OSHIRO, T. M. *Uma abordagem para a construção de uma única árvore a partir de uma Random Forest para classificação de bases de expressão gênica*. 101 p. Dissertação (Mestrado) — Universidade de São Paulo, Ribeirão Preto, SP, 2013. Citado na página 19.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, nov. 2011. Citado 4 vezes nas páginas 14, 24, 31 e 49.

PETERSON, B. *Python 2.7 Release Schedule*. 2008. Disponível em: <https://www.python.org/dev/peps/pep-0373/>. Acesso em: 13 dez. 2019. Citado na página 36.

RIBEIRO, L. B. *Análise de sentimento em comentários sobre aplicativos para dispositivos móveis: estudo do impacto do pré-processamento*. 83 f. Monografia (Bacharelado em Ciência da Computação) — Universidade de Brasília, Brasília, DF, 2015. Citado 3 vezes nas páginas 17, 18 e 22.

SCHMITT, V. F. *Uma análise comparativa de técnicas de aprendizagem de máquina para prever a popularidade de postagens no Facebook*. 57 p. Monografia (Graduação em Ciência da Computação) — Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, 2013. Citado 2 vezes nas páginas 18 e 27.

SCIKIT-LEARN DEVELOPERS. *User Guide*. 2019. Disponível em: <https://scikit-learn.org/stable/user_guide.html>. Acesso em: 02 fev. 2020. Citado 2 vezes nas páginas 29 e 31.

SHALEV-SHWARTZ, S.; BEN-DAVID, S. *Understanding Machine Learning*: From theory to algorithms. Cambridge, MA: Cambridge University Press, 2014. 449 p. Citado na página 14.

SILVA, N. F. F. da. *Análise de sentimentos em textos curtos provenientes de redes sociais*. 112 p. Tese (Doutorado) — Universidade de São Paulo, São Carlos, SP, Brasil, 2016. Citado 6 vezes nas páginas 10, 15, 16, 17, 34 e 40.

STARMER, J. *StatQuest: Random Forests Parte 1 - Construindo, Usando e Avaliando*. 2018. Publicado no canal StatQuest with Josh Starme. 1 vídeo (9 min 53 s). Disponível em: <https://youtu.be/J4Wdy0Wc_xQ>. Acesso em: 09 nov. 2020. Citado 5 vezes nas páginas 19, 20, 21, 22 e 25.

_____. *Máquinas de Vetores de Suporte, Claramente explicadas*. 2019. Publicado no canal StatQuest with Josh Starme. 1 vídeo (20 min 30 s). Disponível em: <https://youtu.be/efR1C6CvhmE>. Acesso em: 09 nov. 2020. Citado 5 vezes nas páginas 22, 23, 24, 25 e 26.

STEIN, T. *Gírias e abreviações mais usadas do Whatsapp*. 2019. Dicionário Polular. Disponível em: <https://www.dicionariopopular.com/girias-abreviacoes-siglas-mais-usadas-do-whatsapp/>. Acesso em: 20 fev. 2020. Citado na página 42.

XIAO, Y.; WEI, Z.; WANG, Z. A limited memory BFGS-type method for large-scale unconstrained optimization. *Computers & Mathematics with Applications*, v. 56, n. 4, p. 1001–1009, ago. 2008. Citado na página 51.