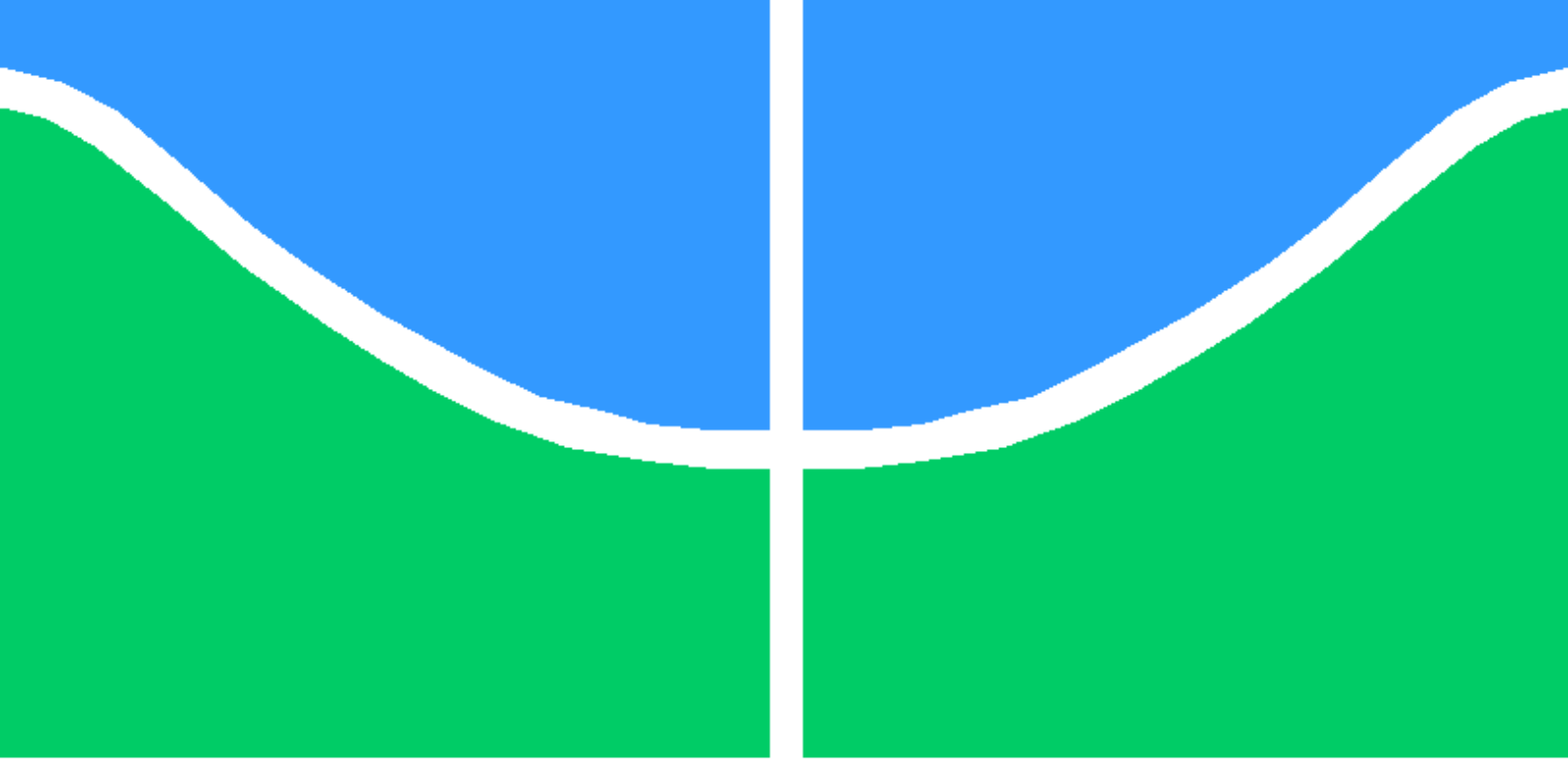

Trabalho Final de Graduação

Estudo e Aplicação de SVMs na Detecção de *Fake-News*

Rodrigo Andrade da Silva

Daniela Prass de Cabral Fagundes

Brasília, Maio de 2021

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

Trabalho Final de Graduação

Estudo e Aplicação de SVMs na Detecção de *Fake-News*

Rodrigo Andrade da Silva

Daniela Prass de Cabral Fagundes

Relatório submetido ao Departamento de Engenharia
Elétrica como requisito parcial para obtenção
do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Dr. Daniel Guerreiro e Silva, ENE/UnB _____
Orientador

Prof. Dr.Alexandre Solon Nery, ENE/UnB _____
Examinador

Prof. Dr. Alexandre Ricardo Soares Romariz, _____
ENE/UnB
Examinador

Agradecimentos

A **minha mãe Lucienne e ao meu pai Aroldo** que me educaram e me guiaram para o meu ingresso na Universidade. Aos **meus irmãos Orí e Juliana e a minha prima Fernanda** que sempre serviram de base e apoio quando encontrei obstáculos. Aos meus colegas de curso **João Fiuza, Pedro Paolo, Matheus Ferreira, Luiz Felipe, Bruno Matias e Daniel Duncan** que sempre me incentivaram à estudar e me auxiliaram com as matérias. Minha colega **Daniela Prass** que dividiu essa jornada comigo, e não menos importante ao professor **Daniel Guerreiro e Silva** que sem sua ajuda e orientação, não seria possível a realização de tal trabalho.

Rodrigo Andrade da Silva

Dedico em primeiro lugar à minha mãe, **Andréa**, que tanto me motiva e me apoia, sempre com afeto. Aos meus avós, **Dora** e **Egon**, por me ensinarem o valor do esforço, da resiliência e do cuidado. Ao meu pai, **Marcelo**, e à minha madrastra, **Marcella**, por me proporcionarem as oportunidades que me trouxeram até aqui. Agradeço especialmente ao meu amigo **Rodrigo**, com quem compartilhei as alegrias e desafios de desenvolver esse projeto, e ao professor **Daniel Guerreiro**, pela orientação excepcional e por todo conhecimento que compartilhou conosco no decorrer desses semestres. Por fim, sou grata a todos os amigos que se fazem presentes em minha vida, em especial à **Mila P.**, à **Renata M.**, à **Bárbara M.**, ao **Daniel A.**, ao **Lucas R.**, ao **Arthur G.** e ao **Matheus F.**.

Daniela Prass de Cabral Fagundes

,

Resumo

As *fake-news* são definidas como o compartilhamento de desinformações que não possuem base factual mas são apresentadas como fatos e não sátira, distribuídas através de jornais, portais de notícia, da televisão e, mais frequentemente, pelas redes sociais. A reprodução dessas notícias falsas sempre foi um grande problema na sociedade e o combate a elas já é, há tempos, uma preocupação recorrente e discutida amplamente no meio acadêmico. Este trabalho apresenta o impacto potencialmente negativo dessas notícias, mostrando alguns exemplos na sociedade atual e citando algumas soluções mais simples para a detecção das mesmas. Com isso, propõe-se uma solução baseada em Máquinas de Vetor de Suporte para executar a detecção automática dessas notícias falsas, porém, antes disso é elucidado toda a base teórica necessária para a total compreensão das técnicas utilizadas. Ao final, é feita uma análise detalhada dos hiperparâmetros C, $gamma$, $degree$ e $Coef0$ e como eles influenciaram na qualidade do modelo, dependendo de qual *Kernel* é empregado. Constatou-se que dentre os modelos treinados, o modelo com o *Bag Of Words* utilizando as ferramentas de Processamento de Linguagem Natural das *stopword* e *stemming* foi o que obteve os melhores resultados. Por fim, conclui-se que SVM foi uma boa técnica empregada para realizar a detecção de *fake-news* obtendo valores próximos de 97% de F1-Score.

Palavras-chaves: Aprendizado de Máquina, Máquinas de Vetor Suporte, detecção, *Fake-News*, *Kernels*, *F1-Score*.

Abstract

Fake-news are defined as the sharing of misinformation that has no factual basis but are presented as facts and not satire, distributed through newspapers, news portals, television and more often, social medias. Fake-news has always been a problem in society, and fighting it is a long time concern in the academia. The following work presents an approach to the fake-news problem, by showing all the harmfull impact that they can bring to the society with examples and giving some advices on how someone can detect them, and also, propose a solution based on Support Vector Machines to detect them in an autonomous way. First, all the theory behind the modules is discribed for the full undestanding of the techniques used. At the end, a detailed analysis is made of the hyperparameters C, gamma, degree and Coef0 and how they influence the quality of the model, depending on which Kernel is employed, showing that among all the trained models, the one with the Bag Of Words using Natural Language Processing of stopwords and stemming was the one that obtained the best results. Finally we conclude that SVM was a good technique used to perform the detection of fake-news getting values close to 97% on the F1-Score.

Keywords: Machine Learning, Support Vector Machine, Fake-News, Kernels, F1-Score.

SUMÁRIO

LISTA DE FIGURAS

1 Introdução

Um tema muito atual e que se configura como um problema ao redor do mundo todo é a disseminação desenfreada de notícias falsas, fenômeno conhecido como *Fake-News*. Trata-se do compartilhamento de desinformações através de jornais, de portais de notícia, da televisão e mais frequentemente, das redes sociais, ou pela definição de Paskin, como [30] "Artigos de notícias específicos que se originam na mídia convencional (online ou offline) ou nas mídias sociais e não têm base factual, mas são apresentados como fatos e não sátira". Este tipo de notícia é escrita e publicada com alguma segunda intenção, seja política ou financeira, já que elas possuem potencial para impactar tanto o governo, quanto a economia. Além disso, as próprias mídias conseguem monetizar a partir de cliques e acessos ao seus sites.

A proliferação de informações errôneas ou mal intencionadas vem crescendo desenfreadamente, e com o auxílio das redes sociais e *blogs* de notícias essa proliferação nunca se deu de maneira tão rápida e atingindo um número tão grande de pessoas como agora. Isso se dá pela natureza apelativa das *Fake-News*, que normalmente possuem títulos chamativos e manchetes sensacionalistas, também chamados de *click baits*, a fim de que um grande número de pessoas cliquem nessas notícias para que haja a monetização fácil de seus portais. As *fake-news* tem a facilidade de alterar a percepção das pessoas a respeito de um determinado assunto, seja por meio de falsos vereditos de supostos médicos ou policiais, ou então de trechos de falas de políticos e de pessoas influentes retiradas de contexto ou que simplesmente nunca aconteceram. Essa disseminação de notícias falsas não é um fenômeno recente da sociedade, data-se que desde os tempos dos romanos o uso de informações falsas, como quando Marco Antônio se encontrava com frequência com a Rainha Cleópatra do Egito, e Otaviano, com o intuito de manchar a reputação de Marco Antônio, fez diversas propagandas enganosas sobre sua pessoa [32].

Apesar do fenômeno das *fake-news* não se algo recente, nos últimos anos isso se tornou algo muito prejudicial para a sociedade como um todo. Seja no âmbito político, como foi visto nas eleições presidenciais do Brasil em 2018, em que o uso do aplicativo de conversa WhatsApp para o espalhamento de notícias falsas foi uma das grandes armas usadas a favor do candidato de extrema direita, Jair Messias Bolsonaro [1], e também nas eleições de 2016 dos Estados Unidos, com o candidato Donald Trump, em que a disseminação de *fake-news* se deu a partir da rede social Twitter, onde, na época, cerca de 6% de todo o consumo de notícias era proveniente de notícias falsas e que apenas 0.1% dos usuários da rede eram responsáveis por compartilhar 80% de todo o conteúdo de *fake-news* [14]. Isso preocupou bastante o cenário político global, pois, por exemplo, candidatos que antes eram julgados com uma pequena chance de ganhar as eleições, com o favorecimento do uso de *fake-news*, foram justamente os candidatos vitoriosos.

Porém, as *fake-news* são um problema ainda maior na área da saúde, o que pode ser observado com grande facilidade na atual pandemia mundial do Corona Vírus. Diferente do âmbito político, na área da saúde o compartilhamento de notícias falsas pode acarretar em complicações de doenças ou até mesmo na morte de indivíduos. Isso se dá pois a grande maioria das pessoas ficam ansiosas

por novas informações acerca do vírus e de seus malefícios recebendo e compartilhando diversos textos e vídeos sobre tratamentos duvidosos sobre o vírus sem confirmar a veracidade dessas notícias. Por fim acabam por causar desinformação e medo aos receptores daquelas mensagens [38] [39] [24].

Com isso, há uma movimentação da sociedade acadêmica no mundo todo em diversas áreas do conhecimento, a fim de combater e estudar os efeitos e as consequências das *fake-news*, especialmente nos últimos anos, quando este se tornou um problema sério em diversas regiões do mundo. Dentro do cenário brasileiro, destaca-se o trabalho do Corpus FakeBR[36], que serviu de ponto de partida para este projeto. Muitos outros trabalhos acadêmicos foram publicados, em diferentes áreas do conhecimento, para auxiliar no combate às *fake-news*. Alguns deles buscam conceptualizar o fenômeno das fake-news propondo e justificando uma definição de fake-news [25], outros buscam avaliar esse fenômeno por uma visão do jornalismo investigativo e avaliar as consequências que a propagação dela acarreta nas pessoas [34]. Porém neste trabalho será dada uma maior ênfase para os estudos realizados na área de aprendizado de máquina.

1.1 Justificativa

O compartilhamento de notícias falsas é um problema muito sério em todos os lugares do mundo, principalmente no Brasil. Com o intuito de contribuir para essa área de pesquisa, o grupo decidiu trabalhar na detecção das *fake-news* utilizando uma ferramenta muito importante para a área: o aprendizado de máquina.

1.2 Objetivos

O presente trabalho tem como objetivo:

- Criar uma classificador de *Fake-News* baseado em Aprendizado de Máquina utilizando Máquinas de Vetor Suporte;
- Propor uma comparação entre os tipos de *Kernels* utilizados pelas *Support Vector Machines* e seus diferentes hiperparâmetros e apresentar o modelo com os melhores resultados;

1.3 Metodologia

O presente trabalho está organizado da seguinte forma:

1. O Capítulo 1 traz um resumo do problema que as *Fake-News* trazem para a sociedade moderna e o que este trabalho pode oferecer para auxiliar neste combate.
2. O Capítulo 2 apresenta toda a fundamentação teórica do trabalho, explicando detalhadamente cada uma delas;
3. O Capítulo 3 mostra os experimentos realizados para o desenvolvimento da parte prática;

4. O Capítulo 4 reúne os resultados obtidos e disponibiliza eles em forma de tabelas e gráficos;

5. No Capítulo 5 são apresentadas as conclusões retiradas do trabalho e as possíveis contribuições futuras que podem ser feitas na área.

2 Fundamentação Teórica

O objetivo deste capítulo é apresentar as motivações para a escolha do tema e as ideias e conceitos envolvidos na implementação do classificador de *Fake News*. Com o intuito de facilitar o entendimento, os temas seguem uma ordem lógica de aprendizado e há uma preocupação em apresentar alto grau de detalhe para cada um dos processos. Espera-se que, ao concluir o estudo, o leitor tenha todas as ferramentas teóricas necessárias para compreender o funcionamento prático discutido no capítulo 3.

2.1 Detecção de *Fake-News*

Como dito anteriormente, as *fake-news* são um problema bem preocupante para a sociedade moderna e uma das grandes dificuldades é a de identificar se uma notícia é verdadeira ou se ela realmente é uma notícia falsa.

Algumas dicas simples de como averiguar a veracidade da notícia são descritas abaixo [27].

- Verificar a data da publicação da notícia, já que muitas das notícias falsas compartilhadas são notícias reais que foram apenas tiradas de sua época original, o que as tornam falsas dependendo do contexto;
- Procurar pela notícia em outras mídias confiáveis para checar se todos os lugares estão reportando o ocorrido da mesma maneira;

Grandes empresas como Instagram, Facebook e Twitter usam de outras ferramentas para realizar essa verificação. O Facebook, por exemplo, usa de técnicas de *machine learning* para identificar notícias falsas ou sensacionalistas e, assim, fazer com que elas apareçam menos nos *feeds* de notícias dos usuários e junto disso, também compartilha dicas de como essa identificação pode ser feita pelos próprios usuários [15]. Já o Twitter garante que toda busca por determinado assunto resulte em artigos com certa credibilidade [14], enquanto no Instagram, postagens sobre determinado assunto que são dadas como falsas, recebem um selo avisando sobre o conteúdo, e *links* para *sites* de confiança que trata sobre aquele assunto [22], como pode ser visto na Figura 2.1.

Essas medidas são possíveis por conta das diferentes vertentes de estudo que podem ser dispor de ferramentas para auxiliar na detecção dessas notícias falsas. As plataformas que se utilizam de técnicas de aprendizado de máquina fazem uma coleta de *fake-news* provenientes de portais de comunicação famosos, as adicionam aos seus bancos de dados e a partir delas, refinam o algoritmo utilizado [4]. Outras abordagens detectam *fake-news* utilizando metadados para realizar a comparação da data de publicação do artigo e o cronograma de divulgação do artigo, bem como de onde é proveniente a história [4].

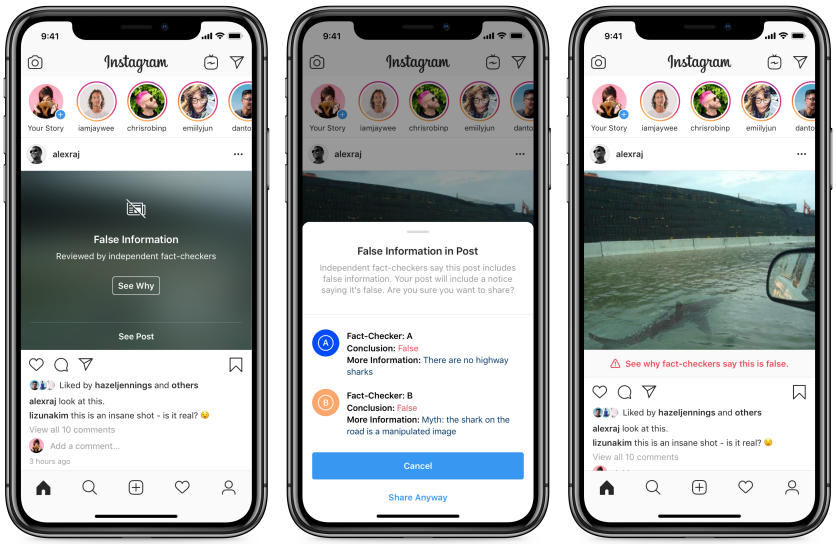

Figura 2.1 – Exemplo da técnica utilizada pelo Instagram Fonte: [22]

2.2 Codificação

Os conceitos computacionais discutidos nessa seção foram baseados na obra de French, C.[9].

O primeiro passo na direção da compreensão de como o computador pode ser usado para identificar uma notícia falsa é entender como ele interpreta informação. É possível fazer um paralelo entre a forma com que os seres humanos processam informação, de acordo com [37], e a forma com que os computadores o fazem. Processamento consiste na transformação de dados de entrada em dados de saída. No caso dos seres humanos, o órgão responsável por tal função é o cérebro, enquanto nos computadores, essa tarefa é executada pela unidade central de processamento. Estímulos externos são percebidos pelo Sistema Sensorial do corpo humano e levados ao cérebro em forma de sinais elétricos pelo Sistema Nervoso. No computador, informações externas são captadas por sensores e levadas em forma de sinais elétricos para a unidade de processamento, onde são convertidos para sinais digitais. No cérebro, a informação de entrada é processada e transformada em uma saída, que pode ser uma ação, um pensamento, uma emoção etc. No computador, os dados de entrada são processados e gera-se uma saída, que normalmente é a impressão de alguma informação visual na tela.

Na unidade de processamento, a menor unidade de informação digital é o dígito binário, ou *bit*. O *bit* é a representação lógica de um estado e pode assumir apenas dois valores: um, que indica verdadeiro, e zero, que indica falso. Os *bits* são agregados em uma unidade maior chamada *byte*. Cada *byte* possui 8 *bits* e, consequentemente, pode representar, no máximo, 256 valores binários. É em termos de *bits* que se mede o tamanho das informações digitais e, em geral, a memória ocupada por elas é levada em conta em qualquer aplicação computacional, visto que os recursos de armazenamento são limitados.

A menor unidade de texto é o caractere, que, por definição, consiste em um símbolo qualquer, seja ele uma letra, um algarismo, um sinal de pontuação, dentre outros. O computador entende

cada caractere como uma sequência de *bits*. O conjunto de múltiplos caracteres recebe o nome de *string* e ela pode ser tanto uma palavra, quanto uma frase, ou até um texto inteiro. É necessário que seja especificado ao computador quais são as regras de conversão de um caractere em uma sequência de bits, para que haja processamento. A essa especificação, dá-se o nome de codificação, do inglês *encoding*. Essa codificação, na prática, é uma tabela que mapeia caracteres às suas respectivas representações numéricas decimais ou hexadecimais, posteriormente convertida em seus correspondentes binários.

A codificação mais usada hoje em dia, podendo ser considerada padrão em tecnologia da informação, é o Conjunto Universal de Caracteres (Unicode, do inglês *Universal Character Set*). Ele inclui todos os caracteres de qualquer sistema de escrita, motivo pelo qual se popularizou tanto em detrimento de outras codificações, mais limitadas. Opera com três tipos distintos - UTF-8 [41], UTF-16 [29] e UTF-32 - cujos sufixos numéricos representam a quantidade de *bits* utilizados no mapeamento. Como utiliza um bloco maior de *bits*, os arquivos codificados com UTF ocupam bastante espaço em comparação a outros menos robustos, porém, em compensação, são capazes de representar um número muito maior de caracteres.

2.3 Aprendizado de Máquina (ML)

Os conceitos apresentados nessa seção fundamentam-se em [2].

A partir do entendimento de como funciona o computador, a nível de processamento, parte-se para um nível mais alto de abstração, que se refere à resolução de problemas. A partir de um conjunto de dados de entrada, buscam-se formas de trabalhá-los para atingir um objetivo específico. Quando o procedimento para gerar a saída desejada é conhecido, utiliza-se algoritmos para realizar a tarefa de transformar os dados de entrada. Um algoritmo consiste em uma sequência de instruções bem definidas que indicam ao computador, por meio de comandos escritos em linguagem de programação, o que deve ser executado.

Porém, existem tarefas para as quais não há um procedimento específico conhecido e, portanto, não há como utilizar um algoritmo para descrevê-lo. Quando o objetivo é fazer predições ou identificar padrões, por exemplo, não há um conjunto de passos específicos a serem seguidos para implementá-lo. Além disso, é preciso levar em conta que os dados podem se modificar ao longo do tempo e de acordo com o contexto em que estão inseridos.

Hoje em dia, a quantidade de dados disponíveis é enorme, motivo pelo qual o termo *Big Data* se popularizou. E partindo do pressuposto de que o mundo não funciona de maneira completamente aleatória, pode-se inferir que esses dados também possuem padrões intrínsecos. Ao explorar esses padrões, é possível entender quais são os processos envolvidos em cada cenário e utilizá-los para fazer predições, assumindo que o futuro próximo se mantenha na tendência evidenciada pelo padrão até o momento.

O Aprendizado de Máquina (ML, do inglês *Machine Learning*) é um dos ramos de estudo da ampla área de inteligência artificial (IA). Na psicologia, a aprendizagem humana pode ser definida como uma "mudança de comportamento resultante do treino ou da experiência"[10]. Esse conceito

é semelhante no contexto de IA. Tom Michell define ML como "um computador que aprende a partir de uma experiência E, alguma tarefa T com uma performance P. Se a performance em T, medida por P, melhora com E, há aprendizado"[26].

A ideia, então, é que o modelo seja capaz de extrair algoritmos automaticamente dos dados, dispensando a necessidade de se criar uma solução para todas as situações possíveis e para as que nem aconteceram ainda. Esse algoritmos podem não descrever perfeitamente o processo em análise, mas acredita-se que eles produzam boas aproximações, das quais se podem tirar conclusões assertivas. Espera-se que essas aproximações sejam cada vez melhores à medida em que o tamanho do conjunto de dados aumenta. As soluções são desenvolvidas em cima das teorias estatísticas, já que o objetivo principal é a fazer inferências com base nas amostras. O desempenho de um modelo de ML depende do quão generalista ele é, ou seja, do quão capaz ele é de fazer predições tanto para os dados que se possui, quanto para novos dados.

Segundo Alpaydin [2], existem quatro principais abordagens dentro do aprendizado de máquina, que variam de acordo com o problema. A associação busca pela criação de relações de frequência com que os fatos acontecem e geralmente é aplicada em análises de cesta de compras. O aprendizado não-supervisionado ataca cenários em que não há exemplos e o seu objetivo é encontrar similaridades entre os dados de entrada, caracterizando-se como uma boa forma de seccionar instâncias. Ele é muito utilizado em segmentações de mercado, por exemplo. O aprendizado por reforço tem o propósito de criar comportamento por meio de punições e recompensas, determinadas por uma política de ação em resposta a um conjunto de estímulos. Os robôs dos jogos virtuais, por exemplo, podem ser programados dessa forma. O aprendizado supervisionado, por sua vez, é utilizado em cenários em que os dados possuem exemplos do que se espera na saída do modelo.

Como neste trabalho, o objetivo é utilizar um conjunto de dados rotularizado para fazer predições acerca da veracidade das notícias, tem-se que a melhor abordagem é a de aprendizado supervisionado.

2.4 Aprendizado Supervisionado

Quando o conjunto de dados a ser utilizado para treinamento possui uma variável que representa a saída desejada do modelo, também chamada de rótulo, ou no caso da regressão, um valor esperado, tem-se um cenário de aprendizado supervisionado. Pode-se pensar no modelo como um aluno, nos dados de treino como exercícios e nos rótulos como respostas das questões. Nessa analogia, o aluno pode verificar a resposta de cada questão após resolvê-la, diminuindo, assim, sua probabilidade de errar a questão seguinte. Ao terminar a lista de exercícios, o aluno é submetido a um teste, que pode ser entendido como o conjunto de dados de validação, cuja nota equivale à métrica de qualidade do modelo. O aluno resolve todo o teste e, ao final, recebe o gabarito e a sua nota, ou seja, seu número de acertos. Ao final do período, o aluno faz uma prova, que pode ser entendida como o conjunto de teste, e a sua nota expressa o quão bem ele aprendeu o conteúdo.

Todo projeto de ML supervisionado tem, como entrada inicial, um conjunto de dados a respeito de algum assunto específico, obtido através de uma coleta de dados. A primeira decisão a ser

tomada é a pergunta a ser feita a esses dados, para que se possa entender o problema e saber eleger candidatos para solucioná-lo. Em seguida, é preciso tratar esses dados, para que se tenha um conjunto no formato de entrada do modelo. Essa etapa recebe o nome de pré-processamento e ao final dela, o conjunto de dados é dividido em três subconjuntos: treino, validação e teste.

Na etapa seguinte, todos os modelos candidatos são treinados a partir do conjunto de treino. Assim, tem-se uma porção de modelos treinados, capazes de responder à pergunta inicial. O dilema passa a ser escolher um entre eles. Deseja-se que o candidato escolhido seja o melhor e para fazer essa verificação, é preciso escolher uma métrica de qualidade. Então, todos os modelos candidatos fazem predições sobre o conjunto de validação e são submetidos a essa métrica para que sejam ranqueados de acordo com seus *scores*, ou seja, suas notas na avaliação. Aquele que receber a maior nota é assinalado como melhor modelo dentre os candidatos. A esse processo de verificação, dá-se o nome de validação cruzada.

Em seguida, o melhor modelo é treinado novamente, dessa vez, utilizando tanto o conjunto de treino, quanto o conjunto de validação. A saída desse processo será o modelo final treinado, capaz de responder à pergunta inicial da melhor forma dentre as formas concorrentes. A última etapa consiste em medir o quão bom ficou esse modelo final, utilizando a mesma métrica escolhida para ranquear os candidatos, no conjunto de teste. A figura 2.2 ilustra como cada etapa conecta-se com as demais.

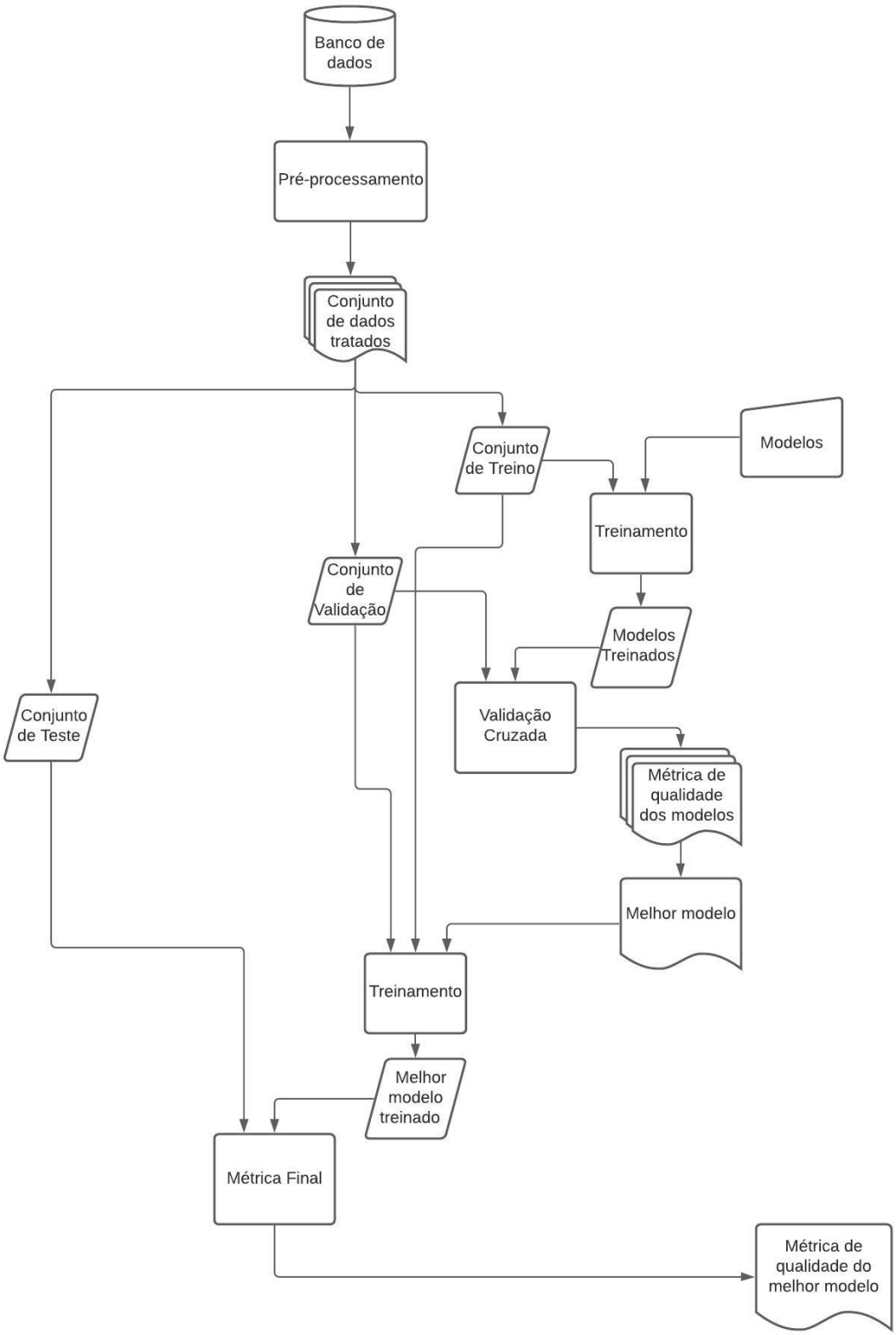

Figura 2.2 – Fluxograma de Projeto de Aprendizado de Máquina Supervisionado.
Fonte: Autores.

O aprendizado supervisionado pode ser dividido em dois tipos de problema: classificação e regressão. A diferença entre eles está no tipo de previsão que pretendem produzir. Caso se deseje agrupar as entradas dentro de categorias específicas, tem-se um modelo supervisionado de classificação. Por outro lado, caso o objetivo seja identificar qual é a saída numérica produzida por uma certa entrada, tem-se um modelo supervisionado de regressão. Portanto, a saída da classificação é um valor discreto, enquanto a saída da regressão é um valor contínuo. Como no caso deste projeto, deseja-se que a saída indique se a notícia é verdadeira ou falsa, tem-se um problema de classificação.

2.4.1 Modelagem de um Classificador

O objetivo principal de um classificador de texto é categorizar documentos em número fixo pré-definido de categorias, tal que cada um só pode ser atribuído a uma das classes [18]. Utiliza-se o aprendizado supervisionado para que a máquina seja capaz de realizar essa tarefa automaticamente. A metodologia e formulação utilizadas nessa seção foram propostas por Alpaydin [2].

Quando se tem um conjunto de dados e se deseja classificá-lo por meio de um modelo de ML, toma-se como pressuposto que esses dados seguem regras não conhecidas. Assim, pode-se dizer que as regras que regem o comportamento dos dados são dadas por $f(x)$, tal que x é o conjunto de todas as observações possíveis em relação a essa essa função desconhecida. O resultado de todas as observações, submetidas a $f(x)$, é expresso por y, conforme mostrado na equação 2.1.

$$y = f(x) \tag{2.1}$$

Em um cenário real, não é possível conhecer todas as observações que compõem x. É possível, apenas, obter uma amostra desses dados, (X, Y), composto por um conjunto de dados de entrada, X, representados pela equação 2.2 e por um conjunto de dados de saída esperada, Y, expressos pela equação 2.3. Cada coluna d de X, x_d, $d = 1, ..., D$, representam atributos, também chamados de *features* ou de variáveis, tal que D representa o número total de atributos da amostra.

$$X = \begin{bmatrix} x_1^1 & x_2^1 & ... & x_d^1 & ... & x_D^1 \\ x_1^2 & x_2^2 & ... & x_d^2 & ... & x_D^2 \\ ... & ... & ... & ... & ... & ... \\ x_1^N & x_2^N & ... & x_d^N & ... & x_D^N \end{bmatrix} \tag{2.2}$$

$$Y = \begin{bmatrix} y_1^1 & y_2^1 & ... & y_i^1 & ... & y_K^1 \\ y_1^2 & y_2^2 & ... & y_i^2 & ... & y_K^2 \\ ... & ... & ... & ... & ... & ... \\ y_1^N & y_2^N & ... & y_i^N & ... & y_K^N \end{bmatrix} \tag{2.3}$$

Cada linha t de X, denotada por x^t, $t = 1, ..., N$, representa uma observação, também chamada de exemplo ou instância, e cada linha t de rótulos, também chamados de *labels* Y, denotada por y^t, $t = 1, ..., N$, representa a classe real da amostra, em forma de vetor binário. Esse é um vetor de tamanho K, referente à quantidade de classes do modelo, denotadas por $C_i, i = 1, ..., K$. Como

cada amostra só pode ter uma classe, é a posição do único valor preenchido no vetor K-dimensional de saída que indica a qual classe ele pertence, conforme expresso pela equação 2.4.

$$y_i^t = \begin{cases} 1 & \text{se } x^t \in C_i \\ 0 & \text{se } x^t \in C_j, j \neq i \end{cases} \tag{2.4}$$

No processo de modelagem, busca-se pela função discriminante que melhor classifica X, $h(X|\Theta)$. Inclusive, $h(X|\Theta)$ é a definição matemática de modelo no contexto de ML. No processo de determiná-la, a primeira suposição feita diz respeito à classe de hipóteses, H, ou seja, ao tipo da função $h(X)$. Em seguida, supõe-se qual é a hipótese h, dentre as contidas por H, que melhor ajusta os dados. Por fim, são escolhidos os parâmetros Θ da função $h(X|\Theta)$. Essa função é utilizada para fazer as predições a respeito de qual classe cada amostra pertence, conforme mostrado na equação 2.5. O objetivo da modelagem é que esses resultados se aproximem o máximo possível das classes reais às quais as amostras pertencem, indicadas por Y.

$$h_i(x^t|\Theta) = \begin{cases} 1 & \text{se } x^t \in C_i \\ 0 & \text{se } x^t \in C_j, j \neq i \end{cases} \tag{2.5}$$

Os responsáveis pelas diferenças entre os valores previstos e os valores reais são os erros irreduzíveis, de bias e de variância. O somatório do efeito de cada um desses erros compõem o erro empírico, que pode ser definido conforme a equação 2.6.

$$E(X) = \sum_{t=1}^{N} \sum_{i=1}^{K} 1(h_i(X^t) \neq y_i^t). \tag{2.6}$$

Os erros irreduzíveis são ruídos consequentes do fato de que algumas amostras podem não ter sido coletadas de maneira correta e de não ser possível obter todas as variáveis que influenciam um problema e sequer todas as observações das variáveis conhecidas. Já o *bias* e a variância surgem a partir de cada uma das suposições de modelagem e, assim, podem ser controlados.

Em um primeiro momento, a função discriminante é aplicada ao conjunto de dados de treino, resultando em predições de treino. O *bias* é definido matematicamente como a diferença entre a média dessas predições e os seus respectivos valores reais, conforme mostrado na equação 2.7.

$$bias = E[h(X)] - Y. \tag{2.7}$$

Quanto menor é essa distância, mais próxima a função está de descrever os dados de treino. Quanto maior é essa distância, mais longe está a função dos dados de treino. A cada suposição feita a respeito dos dados que simplifica a forma de descrevê-los, dá-se o nome de viés indutivo. Ao fazer suposições que simplificam demais o modelo em relação aos dados, essa distância fica maior e se diz que o modelo possui um *bias* alto. Quando são feitas menos suposições, a função se adapta melhor aos dados e nesse caso, diz-se que há um *bias* baixo.

Já a variância é definida como a distância entre cada uma das previsões e a média delas, conforme mostrado na equação 2.8.

$$variancia = E[(h(X) - E[h(X)])^2]. \tag{2.8}$$

Logo, trata-se de uma medida de variabilidade das predições. Quanto maior é esse valor, maior é a influência de um único ponto no conjunto de treino sobre a função final. Quanto menor é esse valor, menor é a influência. É importante lembrar que um conjunto de dados possui ruído e não é interessante que esse ruído seja incluído no modelo como determinante na hora de uma classificação. Quando um modelo possui alta variância, o que ocorre é que a função discriminante descreve tão bem os dados de treino, que esses ruídos são interpretados como dados e causam uma distorção grande nos resultados. Assim, ao utilizar esse modelo para fazer predições em um conjunto diferente, como o de validação, por exemplo, haverá um índice alto de erro. Isso porque o modelo deixa de generalizar bem e, como o objetivo da classificação é ser capaz de fazer predições a partir de novos dados, o modelo deixa de ser considerado bom.

De maneira geral, busca-se por um modelo que possua pouco viés e pouca variância. Há, ainda, um terceiro fator a ser levado em conta: o custo computacional. Quanto mais simples é um modelo, ou seja, se ele possui alto bias e baixa variância, menor é a complexidade envolvida em implementá-lo. Se ele for simples demais, o erro de predição no conjunto de treino será alto e diz-se que houve *Underfitting*. O oposto também é verdadeiro. Quanto mais complexo é o modelo, ou seja, se ele possui uma alta variância e baixo viés, mais custosa é a sua implementação. Se um modelo é complexo demais, o erro no conjunto de treino se aproxima de zero, mas em compensação, o erro no conjunto de validação pode ser alto. A esse cenário, dá-se o nome de *Overfitting*. Esse *trade off* é bastante conhecido na área de ML e recebe o nome dilema viés-variância. Portanto, o ideal é que seja feito um balanceamento entre esses três fatores na hora de modelar o problema.

2.5 Pré-Processamento: Processamento de Linguagem Natural (NLP)

Segundo o dicionário, linguagem é "qualquer meio sistemático de comunicar ideias ou sentimentos através de signos convencionais, sonoros, gráficos, gestuais etc", ou ainda, "qualquer sistema de símbolos ou sinais ou objetos instituídos como signos; código"[21]. Fica claro, então, que a linguagem é a base para todo e qualquer tipo de comunicação. Uma linguagem pode surgir de forma natural, como é o caso dos idiomas, ou ser criada, como é o caso das linguagens de programação. Para ser considerada linguagem, basta que um grupo de regras seja conhecido por um grupo de pessoas e que essas sejam capazes de entender e enviar uma mensagem utilizando-se dessas regras.

Conforme discorrido em detalhes na obra de Harari [16], a linguagem natural, ou apenas língua, trata-se de uma das características mais complexas dos seres humanos e talvez a mais importante. A origem dela pode ser rastreada até mais de 50 mil anos atrás e desde os primórdios da história, ela teve um papel muito importante na evolução da espécie. Ela possibilita que os seres humanos cooperem entre si, aspecto que levou a espécie ao topo da cadeia alimentar. Os idiomas são completamente fluidos e mudam conforme a sociedade que lhes reproduz se modifica. Ao longo

do tempo, já surgiram e desapareceram diversos idiomas diferentes e ainda hoje, existem mais de sete mil deles sendo falados pelo mundo. Essa pluralidade de línguas representa um dos maiores desafios da comunicação de máquinas inteligentes.

NLP pode ser definido como um campo de estudo dentro de Inteligência Artificial cujo objetivo é traduzir a linguagem humana para a linguagem de máquina e vice versa [35]. O objetivo de qualquer ciência linguística é explicar e caracterizar a imensidade de informações que circulam em forma de conversas e textos, dentre outras mídias. O caminho que se segue para tal é o de estudo tanto das capacidades cognitivas humanas de adquirir, produzir e entender linguagem, quanto das estruturas linguísticas que compõem a linguagem [23]. Por um lado, isso pode ser tão simples quanto contar a frequência de palavras e por outro, envolve o entendimento de complexas declarações ao ponto de ser capaz de, no mínimo, responder a elas [5].

O processo de aprendizagem de uma língua, no caso dos seres humanos, ocorre através da assimilação natural e do estudo formal [12]. A primeira acontece de maneira inconsciente e intuitiva ao longo do tempo, resultado de interações entre o indivíduo e ambientes de línguas e culturas diferentes. Já a segunda, diz respeito a uma abordagem supervisionada e acontece conscientemente, por meio do estudo das estruturas gramaticais de um idioma. Os computadores não possuem a capacidade de aprender por assimilação natural, mas podem ser treinados para que conheçam as regras linguísticas de construção de linguagem e sejam capazes de entender e se comunicar. Raskin [33] demonstra a importância de se conhecer aspectos da linguagem humana a fim de implementar soluções de NLP. Um dos objetivos principais da linguística é estudar os mecanismos mentais implícitos na linguagem para encontrar as maneiras como a comunicação é estruturada. É notável o quanto esse conhecimento pode ser útil ao modelar os meios pelos quais o computador entende linguagem.

Parece intuitivo pensar que uma forma de traduzir linguagem seria armazenar todas as palavras em suas diversas formas diferentes e assim, obter um dicionário a ser utilizado para encontrar as palavras de um texto. Porém, para isso, seria necessário dispor de um dicionário estático contendo todas as combinações de palavras e suas variantes gramaticais armazenadas. Tal solução não faz sentido, em termos de custo computacional. Nesse sentido, é mais viável implementar técnicas de aprendizado de máquina, para que ela seja treinada para ler e reconhecer textos de acordo com o objetivo do modelo. Assim, o problema se torna encontrar a melhor forma de treinamento para que o resultado desejado seja alcançado [23].

A compreensão de uma frase por uma máquina ocorre de maneira hierárquica, em termos de processamento. À princípio, a informação é quebrada em unidades menores para que, ao final, possa ser compreendida por completo. A principal diferença é que, pelo menos no contexto deste projeto, a máquina não está a todo momento aprendendo. Ela é treinada utilizando um conjunto fixo de textos e, a partir deles, forma um vocabulário. Quando uma informação nova é inserida, o modelo a quebra em unidades menores e busca por correspondências entre cada uma delas com o que se conhece de vocabulário, ou seja, com o que se tem armazenado na memória do modelo. Cada parte da informação que a máquina não consegue associar é considerada como desconhecida no entendimento daquele conteúdo. Essa quebra do texto em pedaços menores varia de acordo com

o método utilizado para tal e ao conjunto de técnicas com esse tipo de objetivo, dá-se o nome de Tokenização. Existem, também, estratégias para lidar com complexidades linguísticas, típicas da linguagem natural, como a remoção de *stopwords* e o *Stemming*. Os conceitos abordados a seguir seguem a metodologia de Manning [23].

2.5.1 Tokenização

Ao ler um texto, os humanos primeiro leem uma palavra e interpretam seu significado, para depois ler a próxima palavra. Esse processo é muito rápido e torna-se ainda mais rápido à medida que se tem mais fluência em um idioma. Porém, dependendo do contexto, é mais interessante analisar um texto não pelos significados individuais das palavras, mas pelos blocos de palavras que, juntas, expressam uma ideia. Quando se deseja analisar uma oração, por exemplo, é interessante que o bloco de análise contenha o verbo e os demais termos que a compõem. Já no caso de uma análise de período, o bloco contém todas as orações que o compõem. Dessa forma, os blocos variam de acordo com a informação que se deseja obter a partir de um texto. É relativamente recente o reconhecimento do contexto como uma peça essencial na compreensão de linguagem [35].

Os conceitos abordados nessa seção baseiam-se na obra de Manning [23]. No contexto de NLP, o conjunto de textos analisados recebe o nome de *corpus*. Quando o computador lê um texto do *corpus*, realiza um procedimento similar ao da leitura humana. Primeiro, ele separa o texto em blocos de informação para depois processá-los. Esses blocos são chamados de *tokens* e a técnica de separação recebe o nome de Tokenização. Bem como acontece na gramática, os *tokens* podem ser palavras, frases ou blocos de palavras. Nesse contexto, n-grama consiste em uma sequência de n palavras.

Na gramática, a função de separar blocos de palavras é desempenhada pelas pontuações. No caso da Tokenização, essa separação pode ser feita através de qualquer conjunto de símbolos, inclusive pontuações, ou de regras. O mais comum é que esses blocos sejam separados por espaços em branco ou por expressões regulares. As expressões regulares são sequências de caracteres que definem um padrão e são utilizadas, principalmente, para encontrar correspondências entre *strings*. Nem toda linguagem de programação possui essa funcionalidade, porém, dentre as que o fazem, cada uma define qual sequência de caracteres define o quê.

Cada linguagem possui a sua própria construção gramatical e por isso, o idioma do *corpus* precisa ser conhecido previamente para que se possa tomar decisões a respeito de como lidar com cada tipo de particularidade, a depender do objetivo do modelo. Palavras heteronomias, por exemplo, são palavras soletradas de maneira semelhante, porém que possuem significados completamente diferentes. Para o computador, o código que descreve essas palavras é o mesmo e, portanto, são lidas como contendo o mesmo significado. Ainda que seja especificado ao computador todos os significados distintos de uma mesma palavra, ele pode não ser capaz de distingui-las. Um outro problema que pode surgir é o uso de palavras com um significado que não é o original, como no caso das metáforas, por exemplo.

Depois de escolher os *tokens*, eles são utilizados para formar o vocabulário do *corpus*. O vocabulário consiste no conjunto de *tokens* únicos e é utilizado pelo modelo como base para identificar

termos. Ele é construído na fase de treinamento e se torna uma espécie de memória, podendo ser entendida como a linguagem aprendida pelo computador. Quando é inserido um novo texto no modelo treinado, ele primeiro quebra esse texto em *tokens* para depois procurar pela correspondência de cada um deles no vocabulário. Caso essas correspondências sejam encontradas, a máquina entende o *token*, podendo utilizá-lo em suas previsões. Caso a correspondência não ocorra, são aplicadas regras diferentes ao *token* desconhecido, a depender do problema. Por exemplo, ele pode ser descartado ou então pode haver um *token* no vocabulário para identificar termos desconhecidas e, nesse caso, elas podem ser inseridas nas previsões como desconhecidas.

É notável, então, que o grande desafio da Tokenização está na escolha de quais *tokens* utilizar. De maneira intuitiva, pode-se pensar que o melhor seria utilizar todos os *tokens* do conjunto de treino, mas o motivo para não adotar tal abordagem é semelhante ao motivo pelo qual não é criado um dicionário estático contendo todas as palavras de um idioma. Primeiro que o custo computacional é diretamente proporcional ao tamanho do *corpus* e, assim, um corpus grande requer muita capacidade de processamento para treinar o modelo e fazer inferências a partir dele. Além disso, um treinamento que utiliza todas *tokens* do *corpus* pode acabar causando um *overfitting* do modelo. Foi pensando nesse problema que surgiram diversas técnicas de normalização dos dados.

2.5.1.1 Normalização

Uma abordagem para simplificar o vocabulário, que se mostra muito eficaz em problemas em que não é necessário gerar um *output* em forma de linguagem natural, é a remoção de *stopwords*. As *stopwords* são as palavras mais frequentes em um idioma e que não agregam muito ao sentido do texto. É interessante que elas sejam removidas, pois isso diminui o tamanho do conjunto de dados e, consequentemente, do vocabulário e do tempo de processamento do modelo, especialmente na fase de treinamento. Além disso, apresenta-se como uma maneira de garantir que o vocabulário seja composto somente por palavras realmente relevantes.

Normalmente, as típicas candidatas para se tornarem *stopwords* são palavras com função de conexão, tais como pronomes, preposições, conjunções e artigos. Como as regras gramaticais variam muito de um idioma para o outro, suas respectivas listas de *stopwords* também variam. Nesse sentido, não há uma convenção universal de quais *stopwords* devem ser removidas, pois isso varia de um problema para outro. Uma estratégia comumente utilizada para gerar essa lista é ordenar as palavras do *corpus* pela frequência com que aparecem e selecionar manualmente aquelas que não causam problemas ao serem deletadas.

É preciso cuidado na hora de elaborar essa lista, pois a remoção imprópria de palavras pode mudar completamente o sentido do texto ou então dificultar, e até impossibilitar, o entendimento dele pelo modelo. Tais incidentes provocam uma diminuição da qualidade das predições do modelo em que o vocabulário é inserido. Além disso, esse tipo de solução não é recomendada para modelos cuja saída é um texto, pois para um leitor humano, o texto fica desconexo sem essa classe de palavras.

Além de simplificar o vocabulário, é interessante, também, agrupar as palavras de uma maneira mais eficiente, afinal, existem muitas palavras que são escritas de formas diferentes, mas que

possuem um mesmo significado. Essa situação se dá tanto por questões de estilo de escrita, que se relacionam com a preferência do autor, quanto por inflexões existentes na gramática. Existem famílias de palavras que remetem a um mesmo sentido, porém variam de acordo com o contexto. Os verbos, por exemplo, possuem diversas derivações em função do tempo verbal em que são expressos, mas em sua essência, indicam uma ação específica. Para um vocabulário, é interessante que essas palavras diferentes sejam representadas por uma única palavra que represente o conjunto que as agrega. Pode-se criar *tokens* que representem toda uma classe de variações para que a máquina identifique todas elas pelo seu significado único.

Novamente, não é eficiente armazenar cada uma das variações no vocabulário por motivos de custo computacional e de *overfitting*. Uma alternativa melhor é buscar uma forma de retirar apenas o significado das palavras e usá-los para criar os *tokens*. Segundo a morfologia, campo da gramática dedicado ao estudo das estruturas das palavras, as palavras são formadas por um radical, antecedido por prefixos e precedido por sufixos. O radical é a parte da palavra que contém sua significação básica, enquanto os sufixos e prefixos foram acrescentados aos vocábulos ao longo do tempo, conforme a língua foi se desenvolvendo. É possível, inclusive, identificar a origem inicial da palavra a partir da análise dos seus radicais, que podem ser gregos ou latinos, por exemplo.

A partir desse conceito da gramática, surgiram técnicas cujo objetivo é a obtenção de formas base das palavras, não necessariamente seus radicais gramaticais. Essas formas base são usadas para criar o vocabulário e, assim, o modelo é capaz de associar palavras diferentes a um mesmo *token*, sem redundâncias. Uma dessas técnicas é o *Stemming*, que reduz, de maneira bruta, as palavras a uma forma base chamada de *stem*. No caso desse método, uma inflexão, definida assim segundo regras específicas de linguagem do algoritmo, é cortada, restando apenas o que o algoritmo considera como sendo a palavra base.

Apesar de ser eficiente, essa abordagem pode gerar problemas. Caso seja retirada uma parte grande demais das palavras, pode acontecer de palavras com significados diferentes acabarem sendo associadas a um mesmo *token*. Assim, perde-se informação a respeito do texto e tem-se o fenômeno chamado de *Overstemming*. O oposto também pode acontecer. Pode ser que a parte retirada das palavras não seja grande o suficiente, fazendo com que palavras semelhantes sejam associadas a *tokens* diferentes. Assim, o processo de *Stemming* caracteriza-se como sendo simples demais para o problema, a que se dá nome de *Understemming*.

2.5.2 Extração de Atributos de Texto

Seja qual for a aplicação de ML, é necessário que sejam extraídos do conjunto de dados, os atributos a serem considerados para ensinar uma tarefa à máquina. O motivo de não serem inseridos os dados em sua totalidade ao modelo, especialmente em aplicações de texto, é que é justamente essa entrada que define a complexidade do classificador. Por complexidade, leia-se tempo e custo computacional necessário para que o computador aprenda a executar uma tarefa. Esse aspecto merece uma atenção redobrada no cenário de NLP, pois uma pequena frase já pode conter muitos *tokens*, que dirá um conjunto de textos. Levando em conta que a unidade de informação é uma palavra, por exemplo, é possível que se tenha milhares delas ao longo de um *corpus* inteiro.

Conforme elucidado por Alpaydin [2], a estratégia para lidar com conjuntos de dados com muitos atributos é a redução de dimensionalidade. Ela é necessária quando a memória e os recursos computacionais dificultam o processamento de um conjunto de dados. Quanto menor é o conjunto de dados, mais robusto se tornam os modelos mais simples, pois o risco de se estar incluindo ruído no ajuste da função discriminante é menor. É ideal que *features* irrelevantes sejam removidas e que os dados possam ser representados em dimensões menores. Em geral, há duas abordagens para se fazer isso, são elas: seleção de atributos e extração de atributos.

No caso da seleção de atributos, o objetivo principal é que sejam encontradas as *features* que descrevem a maioria da informação de entrada, de maneira que a perda não seja significativa para o entendimento dos dados. Já a extração de atributos consiste na criação de novas *features*, como uma combinação das originais, com o intuito de descrever os dados de maneira mais simplificada. Em ambos os métodos, deseja-se que os dados sejam processados em uma dimensão reduzida em relação à dimensão original, para que se tenha uma performance melhor.

2.5.2.1 *Frequência do Termo-Inverso da Frequência (TF-IDF)*

Após tokenizar e normalizar o texto, é preciso definir a forma do vetor que irá representá-lo na entrada do modelo. O *Bag of Words* (BoW) é uma maneira muito comum de se fazer quando se trata de categorização de textos em NLP. Primeiramente, é construído um vocabulário utilizando os *tokens* normalizados na etapa de pré-processamento. Esse vocabulário consiste em um vetor de tamanho D, tal que D representa o número total de *tokens* encontrados no *corpus*, após a normalização.

Cada documento do *corpus* é representado por um vetor, também de tamanho D, em que cada posição se refere a um *token* do vocabulário. Assim, pode-se definir a entrada do modelo como uma matriz X de tamanho $N \times D$, tal que N representa a quantidade total de documentos que compõem o *corpus*. A essa matriz, dá-se o nome de BoW. Os N vetores horizontais são preenchidos conforme a aparição do *token* a que cada coluna se refere nos documentos. Utilizando uma representação como essa, é possível identificar quais termos estão contidos dentro de cada documento. Fala-se em termos, pois não necessariamente os *tokens* serão palavras. Porém, essa técnica não leva em conta a frequência e a ordem com que essas palavras aparecem no texto [23].

Ou seja, caso o vocabulário possua 10 (dez) palavras e 5 (cinco) delas estão presentes em um determinado documento, o vetor correspondente a ele será preenchido com 5 (cinco) uns e 5 (cinco) zeros. Os '1's serão preenchidos nas posições referentes às palavras encontradas no documento. Essa técnica de escrever dados categóricos em termos de vetores de uns e zeros recebe o nome de *One-Hot Encoding* [11].

Uma boa melhoria desta representação é o uso da Frequência do Termo-Frequência Inversa do Documento (TF-IDF, do inglês *Frequency-Inverse Document Frequency*). Segundo a literatura [20], ele se mostra eficiente especialmente em aplicações de classificação, pois cria um vetor de pesos para os termos, de acordo com a sua relevância no *corpus*. O vocabulário construído é similar ao do BoW, a diferença está na forma com que ele é preenchido. Para cada documento do texto, primeiro, mede-se a frequência de cada termo e depois, a importância dele.

A frequência do Termo, do inglês *Term Frequency*(TF), é medida calculando o quantidade relativa de vezes que um termo aparece em um documento, em relação aos demais termos presentes. Assim, o valor dela aumenta à medida que um termo aparece mais. Então, é incorporado ao cálculo, o Inverso da Frequência do Documento, do inglês *Inverse Document Frrequency*(IDF), para que termos excessivamente comuns no *corpus* não sejam entendidos como relevantes pelo modelo. O motivo disso é que se um termo aparece muito ao longo do *corpus*, é provável que ele não seja um bom parâmetro a se utilizar para distinguir classes. Termos que se encaixam nessa categoria têm seu peso reduzido pelo IDF.

Conforme discutido anteriormente, dependendo do tamanho do *corpus*, computar todos os *tokens* únicos existentes, ainda que removendo alguns na etapa de pré-processamento, pode ser inviável em termos de recursos computacionais. Uma boa alternativa nesses casos é computar o TF-IDF para todos os documentos do *corpus*, ranquear os termos de acordo com a sua relevância e remover os menos relevantes. Porém, é preciso tomar cuidado para não remover informações que podem ser importantes para descrição das classes, pois isso ocasiona em perda de desempenho do modelo.

2.5.3 *Linguistic-Based Features*

As *Linguistic-Based Features* são todas as características de um texto ou uma frase, que não levam em conta as palavras propriamente ditas e sim o que aquelas palavras sintaticamente representam, ou seja, quais as funções sintáticas(sujeito, predicado, objeto, predicativo, etc.) de cada palavra dentro de cada oração, não só isso como também levam em conta detalhes gramaticais, como erro de português e ponto final [8][7]. Um exemplo de como a extração das *Linguistic-Based Features* funciona é o seguinte:

- Hoje o céu está muito nublado! Acho que vai chover e ocê?

Enquanto que a ideia do *Bag-of-Words* seria levar em conta todas as palavras individualmente, como 'Hoje', 'o', 'céu' e 'nublado' e com isso realizar todo um pré-processamento em cima dessas palavras, a das *Linguistic-Based Features* seriam a contabilização da quantidade de aparições de termos semânticos e gramaticais que foram escolhidos.

Em 2019, os estudos de Gravanis [13] mostraram que, se bem selecionadas, as *Linguistic-Based Features* são capazes de detectar com uma alta precisão notícias falsas. Como o treinamento utilizando todos os atributos é algo bem custoso computacionalmente falando, então para uma melhor eficiência do tempo de treinamento e para que não haja perda de optimização em relação aos resultados, Zhou, Burgoon, Twitchell, Qin, e Nunamaker [43] propuseram levar em conta as seguintes informações:

1. Número de verbos, advérbios e adjetivos;
2. Complexidade do Texto: média do tamanho das sentenças e do tamanho das palavras);
3. Pausalidade: quantidade de aparições de sinais de pontuações em uma sentença;

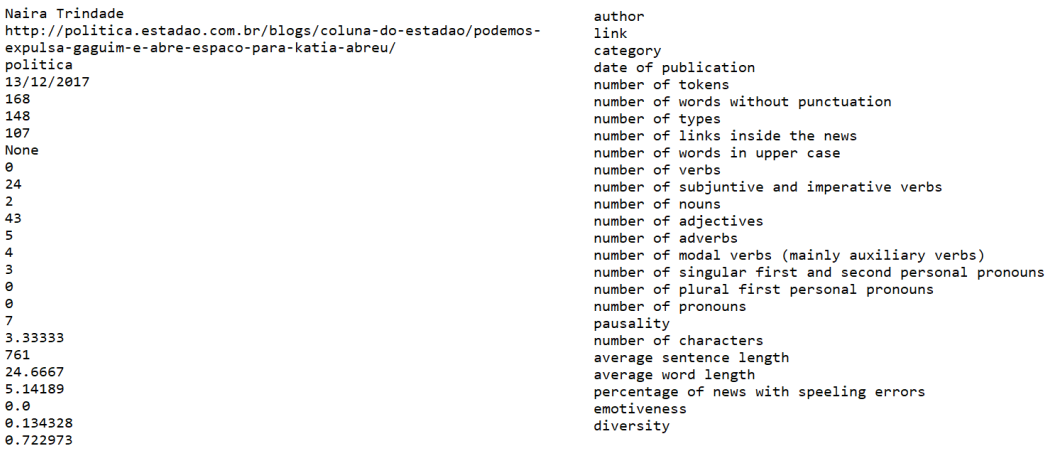

Figura 2.3 – Exemplo de uma notícia utilizando as *linguistic-based features*. A parte da direita são os valores dos atributos e a direita quais são os atributos

4. Incerteza: número de verbos modais e voz passiva;
5. *Non-Immediacy:* frequência de uso de pronomes pessoais;
6. Emotividade: é calculado como a soma do número de adjetivos e advérbios divido pela soma do número de substantivos e verbos [42];
7. Diversidade: é medido como a quantidade de diferentes termos de conteúdo sob o total de termos de conteúdo. Termos de conteúdo são todos os termos que carregam valor semântico;
8. *Informality features:* quantidade de erros ortográficos;

Com isso, utilizando o exemplo anterior as *Linguistic-Based Features* seriam as quantidades de cada um dos atributos acima citados. Um exemplo prático pode ser visto na figura 2.3.

2.6 Modelo: Máquina de Vetor de Suporte (SVM)

Existem muitas abordagens diferentes ao problema de classificação, a depender da classe de hipóteses H escolhida. A máquina de vetor de suporte (SVM), do inglês *Support Vector Machine* (SVM), é uma delas. A SVM surgiu do princípio da minimização de risco estrutural [40] e é baseada em discriminante, ou seja, assume um modelo diretamente para o discriminante ao invés de calcular verossimilhança, posteriores e densidades, como acontece em outras abordagens. A função discriminante, descrita pela equação 2.9, é explicitamente parametrizada pelo conjunto de parâmetros Θ_i.

$$h_i(x|\Theta_i) \tag{2.9}$$

Esse método se popularizou, em detrimento dos baseados em verossimilhança, pois se diz que é considerado mais fácil estimar o discriminante do que as densidades de probabilidades, como é

feito em outras abordagens. Vapnik (1995) é um defensor dessa ideia e afirma, ainda, que não faz sentido utilizar resolver um problema utilizando um outro problema, mais complexo que o original. A metodologia e formulação utilizadas nessa seção foram propostas por Alpaydin [2].

O viés indutivo desse tipo de modelagem está na suposição feita em relação à forma da fronteira que separa as classes. Aprender, no contexto de SVM, significa otimizar os parâmetros Θ_i a fim de maximizar a qualidade da estimação da fronteira que separa as regiões de cada classe. Essa fronteira é especificada por um hiperplano, no caso, um subplano cuja dimensão m é menor do que a dimensão original $m + 1$ [2]. Uma dimensão pode ser entendida como o número mínimo de coordenadas necessárias para especificar um ponto. Assim, o hiperplano de um espaço bidimensional é uma reta, de um espaço tridimensional é um plano, e assim por diante. Esse hiperplano é definido a partir de um vetor de pesos aplicado a um subconjunto do conjunto de treino, que origina os vetores de suporte.

Na classificação, os vetores de suporte referem-se às instâncias que estão próximas da fronteira e, por isso, ao conhecê-los, é possível conhecer o hiperplano. Através deles, é possível estimar o erro de generalização, pois por se tratar de uma região localizada entre duas classes, as amostras posicionadas ali podem estar erroneamente classificadas. As demais instâncias, fora dessa região de incerteza, provavelmente estão corretamente classificadas e, assim sendo, não precisam ser levadas em conta na estimação da fronteira.

Considerando um classificador binário, ou seja, que só possui duas classes, tem-se uma amostra $X = \{x^t, y^t\}$, tal que $y^t = 1$ se $x^t \in C_1$ e $y^t = -1$ se $x^t \in C_2$. O objetivo é encontrar o quanto cada uma das variáveis de entrada influenciam na saída. Para isso, procura-se por um vetor de custo, w, e por um custo inicial, w_0 tais que

$$y^t(w^T x^t + w_0) \geqslant 1 \tag{2.10}$$

Além de querer que as instâncias estejam do lado certo do hiperplano, é desejado que elas estejam a uma certa distância dele. A essa distância, dá-se o nome de margem. Para que haja uma melhor generalização, é interessante que essa margem seja maximizada. Quanto maior é a distância das instâncias mais próximas para o hiperplano, menor é a chance de que um novo dado seja classificado de maneira errônea. Quando os dados são separáveis, para maximizar a margem, basta atender à especificação da equação 2.11 [2].

$$\min \frac{1}{2}||w||^2 \text{ sujeito a } y^t(w^T x^t + w_0) \geqslant 1 \tag{2.11}$$

Nesse caso, trata-se de uma otimização quadrática, cuja complexidade depende de d e pode ser resolvida diretamente para encontrar w e w_0. Assim, cada lado do hiperplano, há instâncias a $\frac{1}{||w||}$ de distância do hiperplano e a margem total é de $\frac{2}{||w||}$.

Quando os dados não são linearmente separáveis, ao invés de tentar ajustar uma função não linear, um truque é mapeá-los para um novo espaço através de uma transformação não linear e, então, usar um modelo linear nesse novo espaço. O modelo linear no novo espaço corresponde

a um modelo não linear no espaço original. As novas dimensões são calculadas a partir de uma função base z.

$$z = \Theta(x) \text{ em que } z_j = \Theta_j(x), j = 1, ..., k \tag{2.12}$$

Ao mapear os dados de um espaço original d-dimensional x para um espaço k-dimensional z, a função discriminante pode ser representada conforme a equação 2.13 [2];

$$h(x) = w^T \Theta(x) = \sum_{j=0}^{k-1} w_j \Theta_j(x). \tag{2.13}$$

Assume-se que $z_1 = \Theta_1(x) \equiv 1$ para que não seja utilizado um custo inicial separado w_0. O problema desse mapeamento é que ele depende de k, que normalmente é muito maior do que d e pode ser maior até do que N. Assim, é interessante que seja utilizada uma função discriminante que seja dependente apenas quantidade de amostras N, e não da dimensionalidade dos dados.

A melhor forma de se lidar com dados não separáveis linearmente, especialmente quando estão em espaços de dimensionalidade alta, é utilizando funções *kernel*. Elas são utilizadas no lugar do cálculo de produto interno entre instâncias nas funções bases, de forma a mapear as duas instâncias para um espaço z e calcular o produto interno nesse novo espaço. Assim, a fórmula que expressa o discriminante passa a ser a equação 2.14 [2], que já está generalizada para os casos em que os dados são lineares ou não.

$$h(x) = w^T \Theta(x) = \sum_{t=1}^{N} \alpha^t y^t \Theta(x^t)^T \Theta(x) = \sum_{t=1}^{N} \alpha^t y^t K(x^t, x). \tag{2.14}$$

Joachims [18] foi um dos pioneiros na utilização de SVM para classificação de texto e fala sobre as vantagens de se implementar essa técnica. Dados de texto possuem um espaço dimensional muito grande e a maioria das *features* são relevantes, o que pode tornar o processamento muito custoso dependendo da técnica escolhida. Como as SVMs não dependem da dimensionalidade dos dados de entrada, são uma boa opção para lidar com esse tipo de *input*.

2.6.0.1 Funções *Kernel*

A função *kernel* é definida, segundo [17], como sendo a equação 2.15, que satisfaz, para todo $a, b \in X$, a equação 2.16.

$$K : X \times X \to \mathbb{R}, \quad (a, b) \mapsto K(a, b) \tag{2.15}$$

$$K(a, b) =< \phi(a), \phi(b) > \tag{2.16}$$

O $\phi(a)$ é uma função que mapeia a para um determinado espaço de produto vetorial z, chamado de espaço de $feature$. A medida de similaridade, K, é chamada de kernel, e ϕ é chamado de mapa

de *features*. A vantagem de usar um *kernel* como medida de similaridade é que isso permite que sejam construídos algoritmos em espaços de produtos vetoriais. Em essência, utilizar a função *kernel* torna o mapeamento para dimensões maiores uma tarefa mais eficiente e menos custosa computacionalmente. A razão disso é que ela chega ao mesmo resultado escalar que o cálculo de produto interno, sem obrigatoriamente obter todas as coordenadas das instâncias para todas as dimensões do novo espaço.

Existem diferentes tipos de funções de *kernel*. A mais simples delas é o *Kernel* Linear, descrito na equação 2.17.

$$K(a, b) = a^T b + 1. \tag{2.17}$$

Ele é definido simplesmente como o produto interno entre as instâncias a e b, somado a uma constante, normalmente considerada como sendo 1. Utilizar *kernel* linear equivale a utilizar a fórmula de discriminante expressa na equação 2.13, pois o custo computacional será o mesmo. Para que seja eficaz, é necessário que os dados sejam linearmente separáveis.

Uma segunda função muito usada é o *kernel* polinomial, descrito na equação 2.18.

$$K(a, b) = (a^T b + 1)^q. \tag{2.18}$$

A ideia por trás dele é criar *features* fazendo combinações polinomiais das entradas. O grau dos polinômios é escolhido previamente e quanto maior ele é, mais complexa se torna a função. A tendência é que ele se adapte cada vez melhor aos dados de treino, porém, por conta da variância que vai crescendo conforme o grau aumenta, pode ser que ele perca capacidade de generalização.

Há, também, a função de base radial (RBF), do inglês *radial-basis function*, descrita na equação 2.19.

$$K(a, b) = exp\left[-\frac{||b - a||^2}{2s^2}\right] \tag{2.19}$$

Ela calcula a distância euclidiana entre os pontos para definir o quão similares eles são. Quanto mais distantes, menor é a similaridade e mais próximo de zero está a função. Quanto mais próximos, maior é a similaridade e mais próximo de um está a função. O valor máximo que ela pode assumir é 1 e isso só ocorre caso a distância seja 0, ou seja, caso os pontos sejam iguais.

2.7 Métricas de Desempenho

Após tratar os dados e treinar os modelos, é preciso determinar uma métrica para indicar se os resultados foram bons ou ruins, para que se possa tomar decisões a respeito deles. No caso de aplicações de classificação binária, o rótulo pode assumir apenas dois valores: negativo e positivo. Entende-se como erro as predições que são diferentes dos rótulos, ou seja, quando um rótulo positivo é previsto como negativo ou um rótulo negativo é previsto como positivo. Os nomes dados a cada uma dessas situações estão descritos abaixo e estão de acordo com as definições de Alpaydin [2].

- Falso Positivo (fp): rótulo negativo e predição positiva;
- Verdadeiro Positivo (vp): rótulo positivo e predição positiva;
- Falso Negativo (fn): rótulo positivo e predição negativa;
- Verdadeiro Negativo (vn): rótulo negativo e predição negativa;

A partir dessas grandezas, é possível estabelecer diferentes métricas para determinar o desempenho de um modelo. A mais intuitiva delas é a acurácia, que consiste no número de acertos, $(vp + vn)$, dentre o número total de amostras, N, conforme mostrado na equação 2.20.

$$Acuracia = \frac{(vp + vn)}{N} = \frac{(vp + vn)}{fp + vp + fn + vn} \tag{2.20}$$

Ela só é considerada uma boa métrica nos casos em que o conjunto de dados é simétrico, ou seja, que contém o mesmo número de rótulos negativos e positivos. Quando isso não ocorre, pode ser que essa medida não reflita a real capacidade do modelo de fazer predições. Um conjunto de dados em que 80% dos rótulos é positivo e os demais são negativos submetido a uma regra que classifica todas as amostras como positivas, a acurácia é de 80%. Pode parecer um bom modelo, quando na verdade, jamais identifica rótulos negativos.

A precisão mede o número de verdadeiros positivos, vp, dentre todos as predições positivas, $vp + fp$, conforme mostrado na equação 2.21.

$$Precisao = \frac{vp}{vp + fp} \tag{2.21}$$

Assim, o número de vezes em que o modelo faz uma predição positiva correta é balanceado pelo número de vezes em que ele faz uma predição positiva incorreta. Quanto maior for o número de predições positivas incorretas, menor é a precisão do modelo.

A sensibilidade, ou *recall* no inglês, mede o número de verdadeiros positivos, vp em relação a todos os rótulos positivos, $vp + fn$, conforme mostrado na equação 2.22.

$$Sensibilidade = \frac{vp}{vp + fn} \tag{2.22}$$

Dessa forma, o número de predições positivas é balanceado pelo número de rótulos positivos previstos incorretamente. Quanto mais rótulos positivos são previstos como sendo negativos, menor é a sensibilidade. Quando menos esse tipo de erro ocorre, maior é a sensibilidade.

Tanto a precisão quanto a sensibilidade são alternativas para se levar em conta a distribuição dos rótulos no conjunto de dados na qualificação de um modelo. Foi criada, ainda, uma métrica que une essas duas medidas para se ter um indicador de desempenho ainda mais assertivo. O

F1 score consiste em uma média harmônica da precisão e da sensibilidade, conforme mostrado na equação 2.23.

$$F1score = \frac{2}{\frac{1}{sensibilidade} + \frac{1}{precisao}} = \frac{2 \cdot (sensibilidade \cdot precisao)}{sensibilidade + precisao} \tag{2.23}$$

Através dessa métrica, pode-se tomar uma decisão mais acertada em relação à qualidade do modelo, pois ela leva em conta diferentes aspectos dos erros e acertos das predições. Assim, não é enviesada, como é o caso da acurácia.

3 Experimentos

3.1 Visão Geral dos Experimentos

A princípio, parte-se de um conjunto de dados rotulado, que consiste nas notícias disponibilizadas pelo *Corpus FAKE.BR*[36], conforme mostrado na figura 3.1. O resultado que se espera ao final do projeto é um modelo capaz de classificar uma notícia como sendo falsa ou verdadeira e sua respectiva métrica de qualidade. Os experimentos podem ser divididos em três blocos principais, sendo eles: separação dos dados, extração de *features* e treinamento do modelo.

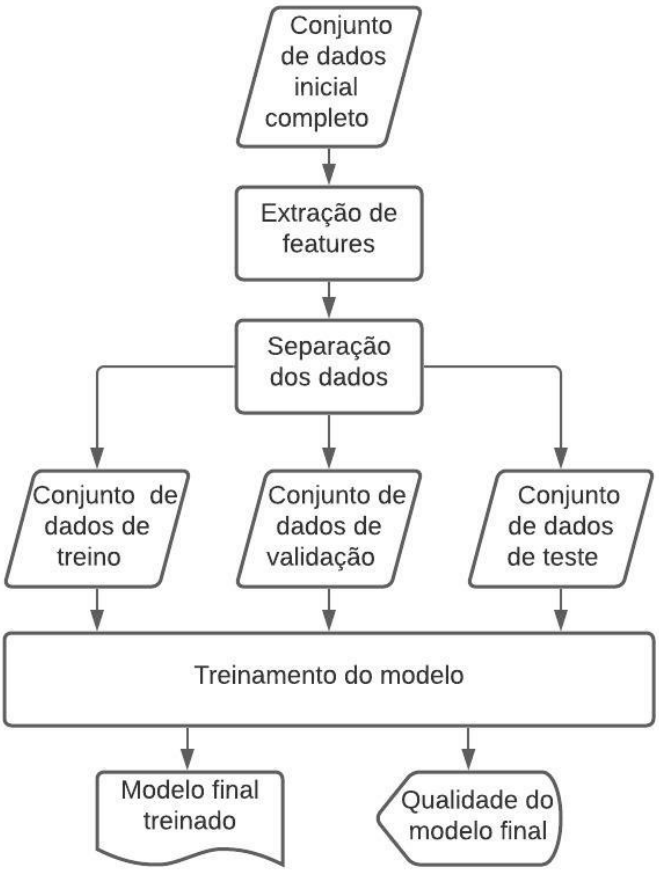

Figura 3.1 – Fluxograma das etapas de experimentos.

A primeira etapa diz respeito à extração dos atributos a serem inseridos nas SVMs, depois é separado esses dados em três subconjuntos: treino, validação e teste. Essa separação é feita para evitar que as métricas de qualidade, aplicadas para comparar os modelos, estejam enviesadas. Então o conjunto de treino é utilizado para treinar o modelo e os demais conjuntos são utilizados para medir a qualidade das predições resultantes do aprendizado.

A extração de *features* ocorre de acordo com dois métodos: *Bag of Words* (BoW) e *Linguistic Features*. Além disso, o BoW foi concebido de seis maneiras, cada uma aplicando uma combinação diferente de técnicas de NLP. Os conjuntos de atributos resultantes do processo de extração estão descritos abaixo.

- *Bag of Words*
  - BoW completo baseado em TF-IDF com tokenização granulada a nível de palavra;
  - *BoW* completo baseado em TF-IDF com tokenização granulada a nível de palavra e remoção de *stopwords*;
  - *BoW* completo baseado em TF-IDF com tokenização granulada a nível de palavra, remoção de *stopwords* e *stemming*;
  - *BoW* truncado baseado em TF-IDF com tokenização granulada a nível de palavra;
  - *BoW* truncado baseado em TF-IDF com tokenização granulada a nível de palavra e remoção de *stopwords*;
  - *BoW* truncado baseado em TF-IDF com tokenização granulada a nível de palavra, remoção de *stopwords* e *stemming*.
- Atributos linguísticos;
  - Número de verbos, advérbios e adjetivos;
  - Complexidade do Texto;
  - Pausalidade;
  - Incerteza;
  - *Non-Immediacy*;
  - Emotividade;
  - Diversidade;
  - *Informality features*.

A última etapa, de treinamento do modelo, envolve a utilização de SVMs para treinar o classificador e métricas de qualidade para validá-lo e testá-lo. Foram implementados três funções de *kernel* distintas nas SVMs: linear, polinomial e RBF. Para cada um desses *kernels*, uma série de parâmetros foram testados a fim de encontrar o melhor modelo em cada um dos cenários. Os *kernels* empregados estão pontuados abaixo.

- SVM com *kernel* linear;
- SVM com *kernel* polinomial;
- SVM com *kernel* RBF.

3.2 Conjunto de Dados

O conjunto de dados - *The FAKE.BR CORPUS* - usado para o treinamento das SVMs, foi criado a partir de um trabalho realizado por Renato M.Silva, Roney L.S.Santos, Tiago A.Almeida e Thiago A.S.Pardo [36]. Os autores selecionaram 7200 (sete mil e duzentas) notícias, todas em português, e realizaram a rotulagem de cada uma individualmente, separando-as em 3600 (três mil e seiscentas) notícias verdadeiras e 3600 (três mil e seiscentas) falsas.

O *dataset* pode ser acessado através do *link* <https://github.com/roneysco/Fake.br-Corpus>. A tabela 3.1 apresenta dois exemplos de pares de notícias, uma falsa e uma verdadeira, a respeito de um mesmo acontecimento. É interessante notar como a versão falsa distorce a informação verdadeira, provavelmente com o intuito de chocar e de gerar injúria por parte do leitor.

Tabela 3.1 – Exemplo de notícias falsas e suas correspondentes verdadeiras

Notícias Falsas	Notícias Verdadeiras
Polos magnéticos da Terra podem se inverter e causar colapso mundial: "A Terra ficaria inabitável". Aos menos 100 mil pessoas morreriam por ano pela alta nos níveis de radiação espacial." Se o campo magnético continuar a diminuir e os polos magnéticos se inverterem, a Terra pode acabar como Marte – um local seco, árido e incapaz de preservar a vida.	Inversão dos polos magnéticos da Terra pode ocorrer mais rápido do que o previsto. Segundo afirmações, essas ocorrências são, a princípio, indistinguíveis das verdadeiras mudanças nos polos. Apesar dessas reversões não representarem qualquer ameaça á humanidade, os especialistas alertam que poderão gerar falhas nos satélites que orbitam a Terra.
Temer avisa que vai vetar a lei anti-Uber. Mesmo com a aprovação dos deputados federais a lei que dificulta o trabalho do UBER no Brasil poderá ser vetada pelo presidente Michel Temer. A equipe do presidente Michel Temer diz esperar que as emendas consideradas prejudiciais ao serviço de transporte Uber – empresa que conecta motoristas particulares a passageiros – e similares sejam alteradas ou derrubadas pela base aliada no Senado.	Prefeitura de SP flexibiliza futuras regras para motoristas de aplicativos. Ás vésperas do início da vigência das novas regras para aplicativos de transporte em São Paulo, a gestão João Doria (PSDB) decidiu flexibilizar nesta sexta-feira (5) alguns pontos da regulação e adiou o prazo para que motoristas e aplicativos se preparem antes de serem fiscalizados.

Para a criação do *dataset* os autores seguiram os seguintes passos:

1. Primeiro, procuraram na internet por notícias falsas e checaram cada uma das selecionadas manualmente para garantir que fossem realmente falsas;

2. Segundo, checaram se a notícia falsa é uma meia verdade. Caso seja, ela é excluída do banco de dados;

3. Terceiro, implementaram uma automatização da busca por versões verdadeiras correspondentes às notícias falsas selecionadas.

A priori, a premissa era procurar por tamanhos de notícia, em termos de quantidade de palavras, semelhantes para associar notícias falsas às suas correspondentes verdadeiras. Porém, como as notícias falsas, no geral, são escritas com menos palavras do que as verdadeiras, essa abordagem se mostrou falha.

Uma melhor visualização dos métodos usados para o preenchimento do banco de dados pode ser observado na Figura 3.2.

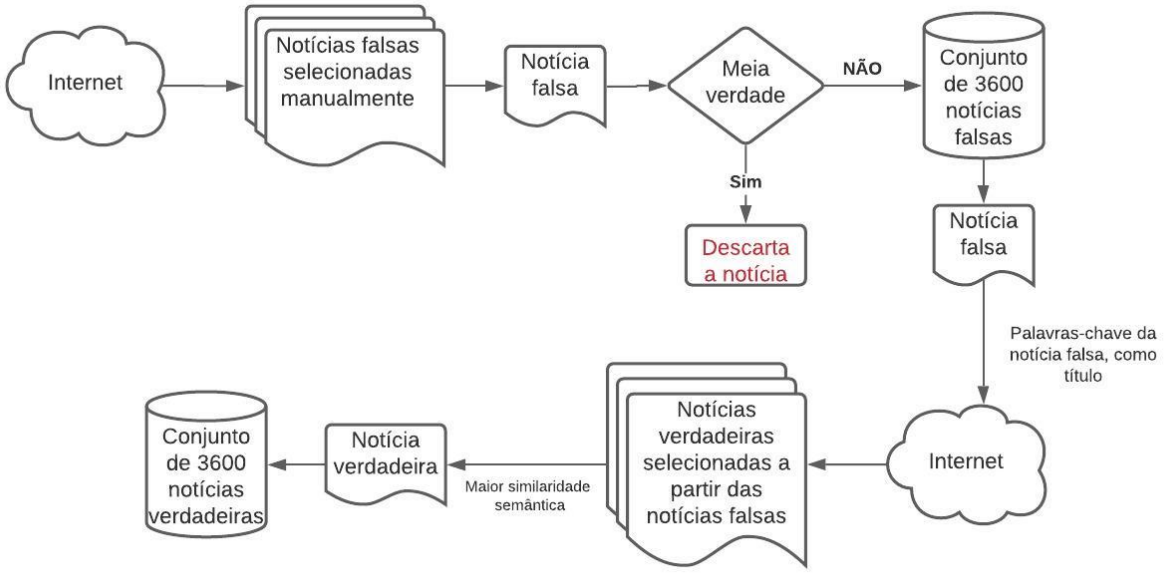

Figura 3.2 – Método utilizado pelo Corpus FAKE.BR para popular o banco de dados. Adaptado de [36]

Os autores ainda comentam que as seguintes observações, usadas por eles durante a busca por notícias falsas, podem auxiliar na identificação de *sites* que reproduzem tal tipo de conteúdo:

- O autor da notícia não é citado;
- Os títulos das notícias são sensacionalistas;
- As notícias contém erros gramaticais e de concordância, e usam de afirmações muito fortes, como "[Nome] é um ladrão";
- As notícias são escritas com uso exagerado de letras maiúsculas e uso desnecessário de pontos de exclamação;
- As notícias não contém as datas do ocorrido e, muitas vezes, não citam outras fontes;
- O *site* não identifica os jornalistas autores da notícia e as pessoas incumbidas de administrar o *site*;

- A visualização do *site* é poluída com muita informação com o intuito de parecer *sites* grandes de notícias.

Durante a criação do *dataset*, são criadas pastas contendo as notícias em sua forma original, a serem utilizadas pelo método de extração por BoW, e outras contendo suas respectivas bases linguísticas, a servirem de insumo para construção dos atributos linguísticos. Os valores dos argumentos salvos nas pastas referentes às *features* linguísticas foram escolhidos pelos autores do *Fake BR*.

Um exemplo de quais atributos são usados para cada notícia salva no *dataset* pode ser visto na Figura 3.3.

```
author
link
category
date of publication
number of tokens
number of words without punctuation
number of types
number of links inside the news
number of words in upper case
number of verbs
number of subjuntive and imperative verbs
number of nouns
number of adjectives
number of adverbs
number of modal verbs (mainly auxiliary verbs)
number of singular first and second personal pronouns
number of plural first personal pronouns
number of pronouns
pausality
number of characters
average sentence length
average word length
percentage of news with speeling errors
emotiveness
diversity
```

Figura 3.3 – Exemplo de uma notícia salva na pasta das *Linguistic-Based Features*

3.2.1 Separação do Conjunto de Dados

O conjunto de dados inicial completo, contendo as 7200 (sete mil e duzentas) notícias é subdividido para que se tenha condições distintas de validação e de teste, evitando o viés. Os dados foram separados conforme mostrado na figura 3.4. Primeiramente, distinguiu-se o conjunto de dados de treino completo do conjunto de dados de teste com uma relação de 80/20. Em seguida, o conjunto de treino completo é subdividido mais uma vez em conjunto de treino e conjunto de validação com uma relação de 53,4/26,6. Essa segunda subdivisão ocorre durante o processo de validação utilizando *K-fold*, técnica a ser detalhada mais a frente.

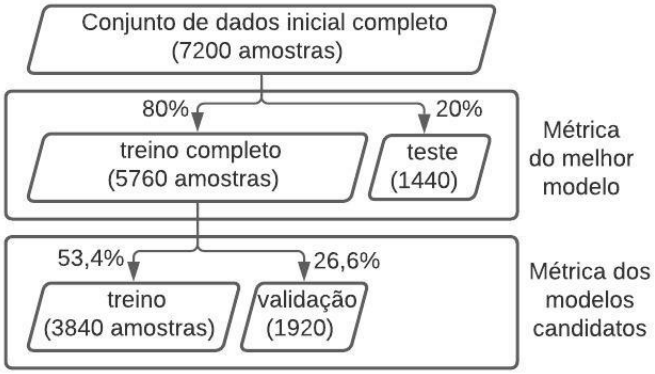

Figura 3.4 – Descrição de como os dados iniciais foram separados.

.

O conjunto de treino, como o nome já diz, é utilizado para treinar os modelos candidatos. Ou seja, serve como base para a construção do vocabulário e da matriz numérica a ser inserida nos diferentes modelos de SVM, cujo objetivo é ensinar as máquinas a classificarem textos. Por ser responsável pelo aprendizado, é o maior dentre os subconjuntos, já que os demais servem apenas para medir a qualidade do aprendizado.

Uma vez treinados, os modelos candidatos estão prontos para fazerem predições a respeito da veracidade das notícias. Então, os documentos do conjunto de validação são inseridos nos modelos treinados, que preveem a qual classe cada um deles pertence. Essa predição é comparada aos rótulos do conjunto de validação para que se saiba quantos acertos e quantos erros ocorreram. Utiliza-se uma ponderação dos erros e acertos de acordo com a métrica de *F1-score* e no fim, cada modelo recebe uma nota. Aquele que receber a maior nota é selecionado como melhor modelo.

Em seguida, o melhor modelo é treinado novamente, desta vez utilizando o conjunto de treino completo, já que não é mais necessário compará-lo a nenhum outro, o que possibilita o uso do conjunto de validação junto ao de treino. Depois, os documentos do conjunto de teste são inseridos no modelo final, que classifica cada uma das notícias. Tais predições são comparadas aos rótulos do conjunto de teste e uma nova nota é atribuída a ele, através do cálculo de *F1-score*. Essa nota refere-se à qualidade final do modelo.

É importante que, para calcular a nota de um modelo, utilize-se um conjunto de dados diferente do empregado no treinamento, pois caso contrário, é inserido viés à métrica de qualidade. Afinal, quando um modelo classifica uma notícia já vista, nada se sabe a respeito do seu aprendizado, apenas que foi capaz de memorizá-la. Por outro lado, quando um modelo classifica uma notícia nunca vista, o que se afere é o quão bem ele aprendeu a fazer essa distinção, em outras palavras, o quanto ele é capaz de generalizar. A segunda é a informação que interessa para medir a qualidade de um modelo de ML.

Vale ressaltar que, a cada subdivisão, é selecionada uma amostra aleatória dos dados, de maneira a manter o balanceamento dos rótulos em cada subconjunto. Como o conjunto de dados

inicial completo possui 50% de notícias falsas e 50% de notícias verdadeiras, essas porcentagens se mantêm, a diferir de uma pequena margem, nos subconjuntos provenientes dele.

3.3 Extração de *Features*

O bloco de extração de atributos pode ser subdividido em dois outros blocos, conforme mostrado na figura 3.5, e são eles: *Bag of Words* e *Linguistic Features*. Eles referem-se às técnicas empregadas para transformar os dados iniciais em formatos que, ao serem inseridos nas SVMs, produzem os resultados esperados. Como o conjunto de dados de entrada é textual, são implementadas técnicas de NLP para gerar matrizes preenchidas com informações numéricas.

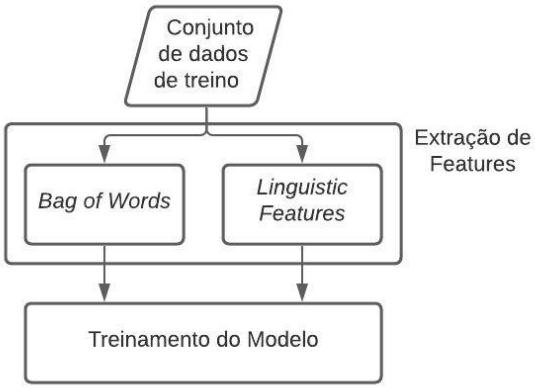

Figura 3.5 – Fluxograma das etapas da fase de extração de *features*.

3.3.1 *Bag Of Words*

O *Bag Of Words* (BoW) é uma técnica muito utilizada para extrair features quando se trata de dados textuais. O objetivo ao implementá-lo é transformar os dados de entrada, inicialmente em forma de texto, em uma matriz numérica. Caracteriza-se como uma boa tradução de escrita para uma linguagem que o modelo é capaz de compreender: a matemática. A figura 3.6 mostra um fluxograma dos processos envolvidos.

Os procedimentos iniciais, comuns a qualquer uma das variações de BoW implementadas no projeto, são a limpeza de dados e a tokenização. A limpeza é feita para que a escrita seja padronizada, de forma a fazer uma simplificação preliminar no texto, removendo elementos como acentos e símbolos, dentre outros procedimentos. O objetivo é que palavras semelhantes não sejam interpretadas como distintas por conta de diferenças na grafia. A tokenização é a etapa de quebra dos blocos de texto em unidades menores para que possam ser processadas, uma a uma.

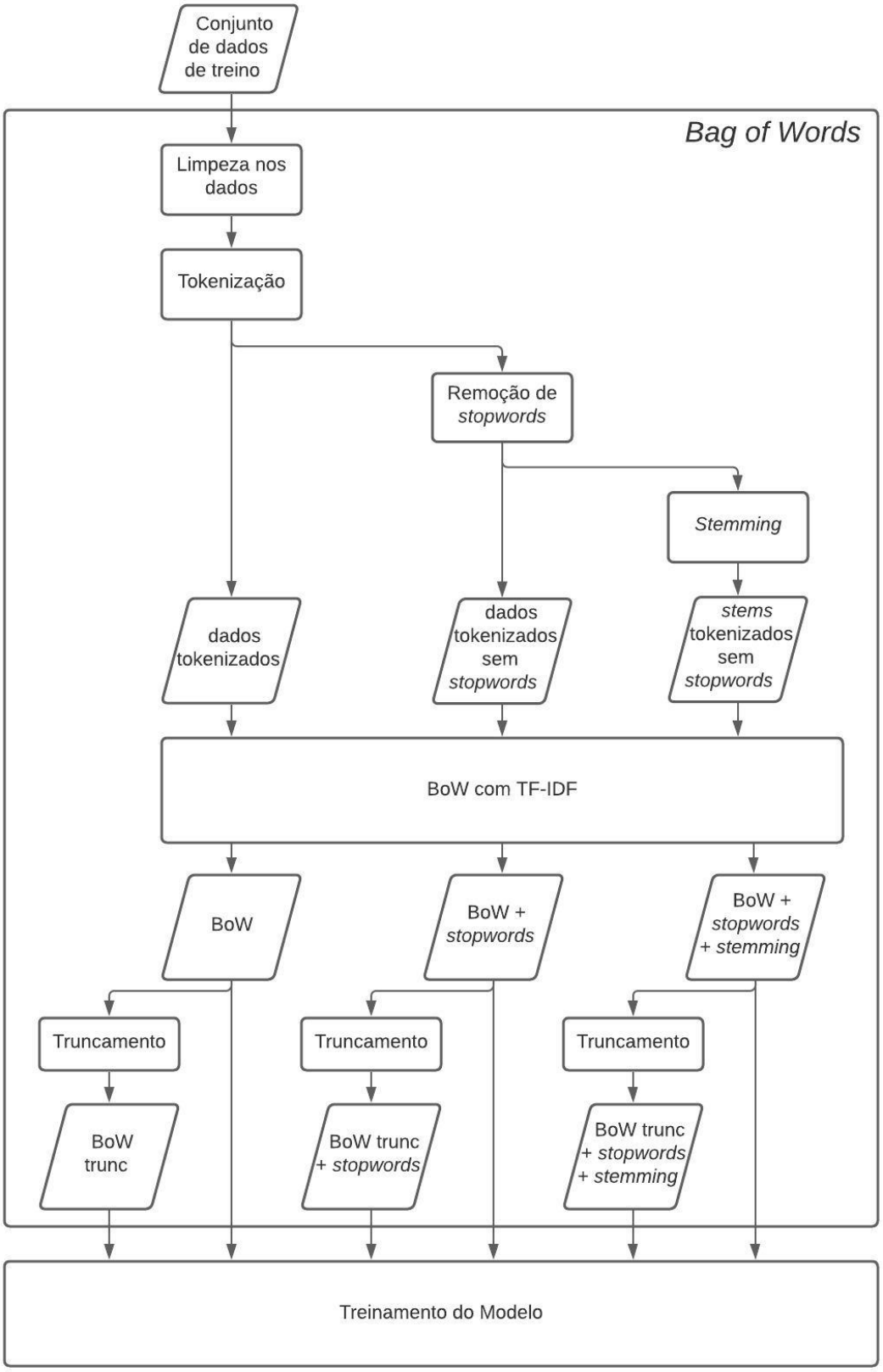

Figura 3.6 – Fluxograma dos processos internos do *Bag of Words*.

Uma vez limpos e tokenizados, os dados estão prontos para serem processados. A fim de comparar alguns dos métodos mais populares do NLP, são geradas seis matrizes numéricas de atributos, cada uma construída de uma maneira. Os documentos são representados numericamente através do mecanismo de Termo Frequência - Frequência Inversa de Documento, que leva em conta tanto a frequência direta quanto inversa dos *tokens* nos textos.

As especificidades de cada uma das saídas do *Bag of Words* estão pontuadas abaixo.

- BoW: a matriz é construída utilizando os dados tokenizados, sem qualquer pré-processamento adicional;
- BoW + *stopwords*: após a tokenização, as *stopwords* são removidas e o vocabulário reduzido é utilizado para construir a matriz;
- BoW + *stopwords* + *stemming*: além da remoção de *stopwords*, implementa-se um algoritmo de *stemming* para reduzir ainda mais o vocabulário usado para construir a matriz;
- BoW trunc: apenas os *tokens* mais relevantes do vocabulário do BoW são utilizados para construir a matriz;
- BoW trunc + *stopwords*: apenas os *tokens* mais relevantes do vocabulário do BoW + *stopwords* são utilizados para construir a matriz;
- BoW trunc + *stopwords* + *stemming*: apenas os tokens mais relevantes do BoW + *stopwords* + *stemming* são utilizados para construir a matriz.

3.3.1.1 Limpeza dos Dados

Antes de qualquer tratamento, as letras foram convertidas para a forma minúscula e os acentos foram retirados. Além disso, os dígitos foram substituídos por '0', as *URLs* por 'URL' e as páginas de redes sociais por 'REDESOCIAL'. Também removeu-se pontuação, caracteres especiais e espaços em branco extras.

A relevância de se fazer isso está no fato de que o computador entende cada caractere como um código binário. Assim, cada palavra é interpretada por um código binário diferente, caso seja escrita utilizando caracteres diferentes. Uma mesma letra, escrita com e sem acento, tal como 'a' e 'á', é vista como dois símbolos distintos. De maneira análoga, uma mesma palavra, escrita com letras maiúsculas e minúsculas, tal como 'exemplo' e 'EXEMPLO', é decifrada como dois termos díspares pelo processador.

Tendo isso em vista, palavras que estão no início de uma frase e que, consequentemente, iniciam com letra maiúscula, são diferenciadas de suas semelhantes localizadas no meio de uma frase, escritas em forma minúscula por completo, por exemplo. A padronização evita problemas como esse. Um dos mecanismos escolhidos para tal foi a criação de *tags*, conforme exemplificado na tabela 3.2.

Tabela 3.2 – Exemplo de transformações de *tags*.

Tag	**Trecho antes da criação da *tag***	**Trecho depois da criação da *tag***
Aspas duplas	O temor era de que cervero fosse "eliminado"como "queima de arquivo".	O temor era de que cervero fosse ASPAS eliminado ASPAS como ASPAS queima de arquivo ASPAS.
Número	o senador romero juca solicitou a retirada da tramitação da pec 3, de 2017, que altera o artigo 86.	o senador romero juca solicitou a retirada da tramitação da pec 0, de 0, que altera o artigo 0.
URL	um homem sem deus, ary lobo: https://www.youtube.com/watch?v=	um homem sem deus, ary lobo: URL.
Páginas de redes sociais	@rafatupy nem era nascido, mas pode perguntar pra ele sobre a trilha.	REDESOCIAL nem era nascido, mas pode perguntar pra ele sobre a trilha.

Uma *tag* é um marcador textual que faz referência a uma parte do discurso, de acordo com o seu respectivo contexto. No caso dos tipos específicos transformados em *tags* - dígitos, *URLs* e páginas de redes sociais - o que se busca é uma simplificação dos dados para criação de significado. Pensando em termos de funcionamento do classificador a ser construído, pouco importa quais são os números, *URLs* ou usuários específicos escritos em uma notícia. O relevante é simplesmente que ela traz informações numéricas, de endereços web e de redes sociais. No caso das aspas duplas, as *tags* foram criadas para que a informação não se perca ao serem removidos os caracteres especiais.

Assim, ao transformar palavras em *tags*, elas são interpretadas pelo processador pelos seus contextos, sem que o seus valores específicos sejam carregados também. Essa prática diminui a complexidade das *features*, pois um modelo que teria como *input* uma grande quantidade de códigos binários diferentes, com frequência baixa, passa a ter como *input* uma quantidade menor de códigos binários, com frequência mais alta. Um exemplo de notícia completa antes e depois do tratamento pode ser visto na tabela 3.3.

Para criar as *tags* referentes às aspas, basta procurar pelos caracteres que as representam e substituí-los pela palavra estipulada. No caso das demais *tags*, das pontuações e caracteres, optou-se por usar expressões regulares do Python. Elas permitem que seja discriminado um tipo de *string* a ser substituído por uma outra *string*. É preciso cuidado na hora de designar as regras de correspondência, pois caso contrário, pode ser que o resultado não fique da forma que se espera. Por fim, os espaços em branco duplicados foram removidos para fins de padronização do texto. As especificações empregadas em cada situação estão escritas a seguir.

- Para identificar dígitos: foram consideradas quaisquer palavras que contenham um ou mais números, seguidas ou não por qualquer *string* sem espaços em branco;
- Para identificar *URLs*: foram consideradas quaisquer palavras que se iniciem com 'http', seguidas de uma *string* sem espaços em branco;
- Para identificar páginas de redes sociais: foram consideradas quaisquer palavras iniciadas com um espaço em branco, seguido de um '@', seguido de uma *string* sem espaços em branco;

Tabela 3.3 – Exemplo de notícia antes e após a limpeza nos dados.

Notícia antes da limpeza	**Notícia após a limpeza**
Site MSN aponta o suposto novo affair de Fátima Bernardes: "Ele é mais jovem, mas sabe de tudo". O site MSN apontou o suposto novo affair da apresentadora Fátima Bernardes. Na noite de ontem (1), Fátima postou uma imagem em sua conta no Instagram ao lado do produtor Rafael Tupinambá. Rafael também é funcionário da Rede Globo e trabalha no programa "Encontro", ao ladinho de Fátima. A apresentadora destacou na descrição da imagem que o rapaz é bem mais jovem que ela e escreveu: "Ele sabe tudo de TV. @rafatupy nem era nascido, mas pode perguntar pra ele sobre a trilha, os atores, o desfecho das novelas. Hj cuida com carinho do elenco que diariamente visita "Encontro". Parceria de sucesso", escreveu. Seguidores começaram a questionar imediatamente: "Fátima, já arrumou outro?", questionou um seguidor. "Fátima está certa... bola pra frente outra", disse outro internauta. "Casal bonito, parabéns", escreveu um terceiro. "Linda, é vida que segue. Tchau Bonner"comentou outra fã. O casal Fátima e William anunciou a separação após 26 anos de casamento na última segunda-feira.	site msn aponta o suposto novo affair de fatima bernardes ASPAS ele e mais jovem mas sabe de tudo ASPAS o site msn apontou o suposto novo affair da apresentadora fatima bernardes na noite de ontem 0 fatima postou uma imagem em sua conta no instagram ao lado do produtor rafael tupinamba rafael tambem e funcionario da rede globo e trabalha no programa ASPAS encontro ASPAS ao ladinho de fatima a apresentadora destacou na descricao da imagem que o rapaz e bem mais jovem que ela e escreveu ASPAS ele sabe tudo de tv REDESOCIAL nem era nascido mas pode perguntar pra ele sobre a trilha os atores o desfecho das novelas hj cuida com carinho do elenco que diariamente visita ASPAS encontro ASPAS parceria de sucesso ASPAS escreveu seguidores comecaram a questionar imediatamente ASPAS fatima ja arrumou outro ASPAS questionou um seguidor ASPAS fatima esta certa bola pra frente outra ASPAS disse outro internauta ASPAS casal bonito parabens ASPAS escreveu um terceiro ASPAS linda e vida que segue tchau bonner ASPAS comentou outra fa o casal fatima e william anunciou a separacao apos 0 anos de casamento na ultima segundafeira

- Para identificar pontuações e caracteres especiais: foram consideradas somente conjuntos de caracteres não alfanuméricas seguidos por um espaço em branco.

3.3.1.2 Tokenização

A tokenização é o processo responsável pela quebra do texto em unidades menores, para que essas sejam interpretadas pelo modelo. Os *tokens* foram definidos como sendo as palavras do texto. Optou-se pelas palavras, em detrimento de frases ou blocos ainda maiores de texto, por conta do seu grau maior de especificidade. Deseja-se encontrar palavras que aumentem a probabilidade da notícia ser classificada como falsa ou verdadeira.

Não foram utilizados n-gramas, apesar de granularem ainda mais os dados de entrada, por questões de custo computacional. Os n-gramas aumentam demasiadamente a dimensão dos dados, tornando o treinamento lento e a demanda por memória alta. Existem aplicações em que esse custo pode ser necessário, como é o caso da filtragem de *spam* [19], pois trata-se de uma prática

conhecida e, por isso, muitos *spams* já são criados com o objetivo de burlá-las. Porém, esse não é o caso dos classificadores de notícias falsas, por serem uma solução relativamente recente.

Para identificar os *tokens*, o algoritmo utiliza espaços em branco e pontuações como critérios de separação. Como os dados foram previamente processados e limpos, apenas o espaço em branco será a indicação de onde as separações devem acontecer. Na tabela 3.4, observa-se um exemplo de notícia sendo tokenizada.

Tabela 3.4 – Exemplo de notícia antes e após a tokenização.

Notícia antes da tokenização	**Notícia após a tokenização**
tragedia anunciada reporter alertou sobre queda de camera na vila olimpica ninguem tomou atitude	['tragedia', 'anunciada', 'reporter', 'alertou', 'sobre', 'queda', 'de', 'camera', 'na', 'vila', 'olimpica', 'ninguem', 'tomou', 'atitude']

3.3.1.3 Remoção de *Stopwords*

O objetivo de remover as *stopwords* do vocabulário é simplificá-lo, sem que haja perdas significativas nos resultados do classificador. Com isso em mente, foram selecionadas palavras muito frequentes da língua portuguesa em um contexto geral, pois suas funções gramaticais as tornam necessárias em qualquer tipo de texto, independente de sua natureza. Dessa maneira, é pouco provável que, caso fossem consideradas *features*, teriam relevância na distinção das classes. As *stopwords* escolhidas para serem removidas do vocabulário foram disponibilizadas pela biblioteca NLTK [6] e estão descritas na tabela 3.5.

As principais classes gramaticais que atendem aos requisitos para serem boas *stopwords* estão descritas abaixo, bem como suas respectivas definições, conforme [3].

- Artigos: são colocados antes dos substantivos para determiná-los de modo particular (definido) ou geral (indefinido). Para o modelo, pode ser mais interessante conhecer os substantivos que precedem os artigos;
- Preposições: ligam duas outras palavras, estabelecendo entre elas determinadas relações de sentido e/ou dependência. Para o modelo, pode ser mais interessante conhecer as palavras ligadas pelas preposições;
- Pronomes: substituem nomes, partes de uma frase ou uma frase inteira e acompanham nomes, determinando os seus sentidos. Para o modelo, pode ser mais interessante conhecer os nomes aos quais os pronomes se referem;
- Advérbios: se relacionam aos verbos, para indicar diferentes circunstâncias. Para o modelo, pode ser mais interessante conhecer os verbos relacionados aos advérbios;
- Verbos auxiliares: juntam-se ao verbo principal para formar as estruturas verbais compostas. Para o modelo, pode ser mais interessante conhecer os verbos principais do que os auxiliares que os acompanham.

Tabela 3.5 – *Stopwords* removidas do vocabulário.

Lista de *stopwords*
['de', 'a', 'o', 'que', 'e', 'é', 'do', 'da', 'em', 'um', 'para', 'com', 'não', 'uma', 'os', 'no', 'se', 'na', 'por', 'mais', 'as', 'dos', 'como', 'mas', 'ao', 'ele', 'das', 'à', 'seu', 'sua', 'ou', 'quando', 'muito', 'nos', 'já', 'eu', 'também', 'só', 'pelo', 'pela', 'até', 'isso', 'ela', 'entre', 'depois', 'sem', 'mesmo', 'aos', 'seus', 'quem', 'nas', 'me', 'esse', 'eles', 'você', 'essa', 'num', 'nem', 'suas', 'meu', 'às', 'minha', 'numa', 'pelos', 'elas', 'qual', 'nós', 'lhe', 'deles', 'essas', 'esses', 'pelas', 'este', 'dele', 'tu', 'te', 'vocês', 'vos', 'lhes', 'meus', 'minhas', 'teu', 'tua', 'teus', 'tuas', 'nosso', 'nossa', 'nossos', 'nossas', 'dela', 'delas', 'esta', 'estes', 'estas', 'aquele', 'aquela', 'aqueles', 'aquelas', 'isto', 'aquilo', 'estou', 'está', 'estamos', 'estão', 'estive', 'esteve', 'estivemos', 'estiveram', 'estava', 'estávamos', 'estavam', 'estivera', 'estivéramos', 'esteja', 'estejamos', 'estejam', 'estivesse', 'estivéssemos', 'estivessem', 'estiver', 'estivermos', 'estiverem', 'hei', 'há', 'havemos', 'hão', 'houve', 'houvemos', 'houveram', 'houvera', 'houvéramos', 'haja', 'hajamos', 'hajam', 'houvesse', 'houvéssemos', 'houvessem', 'houver', 'houvermos', 'houverem', 'houverei', 'houverá', 'houveremos', 'houverão', 'houveria', 'houveríamos', 'houveriam', 'sou', 'somos', 'são', 'era', 'éramos', 'eram', 'fui', 'foi', 'fomos', 'foram', 'fora', 'fôramos', 'seja', 'sejamos', 'sejam', 'fosse', 'fôssemos', 'fossem', 'for', 'formos', 'forem', 'serei', 'será', 'seremos', 'serão', 'seria', 'seríamos', 'seriam', 'tenho', 'tem', 'temos', 'tém', 'tinha', 'tínhamos', 'tinham', 'tive', 'teve', 'tivemos', 'tiveram', 'tivera', 'tivéramos', 'tenha', 'tenhamos', 'tenham', 'tivesse', 'tivéssemos', 'tivessem', 'tiver', 'tivermos', 'tiverem', 'terei', 'terá', 'teremos', 'terão', 'teria', 'teríamos', 'teriam']

3.3.1.4 *Stemming*

O principal objetivo do *stemming* é obter uma forma base das palavras a partir da remoção de uma parte delas. O algoritmo escolhido para tal foi uma versão atualizada do Removedor de Sufixos da Língua Portuguesa (RSLP) [28], disponibilizado em forma de *toolkit* na biblioteca NLTK.

O RSLP, como o nome já indica, remove os sufixos das palavras para obter as suas formas base. Esse processo é feito por meio de uma série de regras de redução. Primeiro, identificamos se uma palavra está no plural, através da terminação em 's', e em caso afirmativo, realiza-se uma redução de plural. Depois, checamos se trata-se ela está em forma feminina, com terminação em 'a', e realiza-se a redução de feminino, caso essa checagem se confirme.

Em seguida, o RSLP realiza reduções de aumentativo, advérbio e substantivo para, então, checar se o sufixo foi, de fato, removido. Se não, implementa a redução de verbo e checa novamente. Se ainda houver sufixo, uma vogal é removida do final. Por fim, os acentos são removidos e o programa retorna o stem da palavra. Um exemplo de notícia após ser processada pelo RSLP é mostrado na tabela 3.6.

Observa-se que a técnica é bem sucedida em algumas situações específicas, tal qual na redução de 'reporter' para 'report'. Essa transformação permite que as variações do verbo 'reportar' sejam associadas ao substantivo 'repórter', já que ambos dizem respeito a uma mesma ação / ideia. Em outras, no entanto, mostra-se ineficaz, como quando reduz nomes próprios. Além de nomes próprios, por definição, não possuírem sufixos, ainda podem ser feitas associações incorretas a eles.

Tabela 3.6 – Exemplos de trechos antes e após o *stemming*.

Exemplos	**Trecho tokenizado**	**Trecho após o *stemming***
Exemplo 1	['tragedia', 'anunciada', 'reporter', 'alertou', 'sobre', 'queda', 'de', 'camera', 'na', 'vila', 'olimpica', 'ninguem', 'tomou', 'atitude']	['traged', 'anunci', 'report', 'alert', 'sobr', 'qued', 'de', 'cam', 'na', 'vil', 'olimp', 'ning', 'tom', 'atitud']
Exemplo 2	['katia', 'abreu', 'diz', 'que', 'vai', 'colocar', 'sua', 'expulsao', 'em', 'uma', 'moldura', 'mas', 'nao', 'para', 'de', 'reclamar']	['kat', 'abr', 'diz', 'que', 'vai', 'coloc', 'sua', 'expulsa', 'em', 'uma', 'mold', 'mas', 'nao', 'par', 'de', 'reclam']

É o caso a redução do sobrenome 'Abreu' para 'abr', já que o *token* gerado acaba sendo associado às variações do verbo 'abrir'.

3.3.2 Frequência de Termo e Frequência Inversa de Documento (TF-IDF)

A motivação para a ponderação dos termos é semelhante à motivação para a remoção das *stopwords*. As palavras que são muito frequentes em todos os documentos não são úteis para discriminá-los em suas respectivas classes. O que realmente interessa é encontrar termos capazes de diferenciar documentos de classes distintas. Porém, palavras muito raras também não são candidatas ideais para tal função, pois fiar-se nelas pode ocasionar uma perda de generalidade do modelo. São tão pontuais que a probabilidade de aparecerem em um novo documento é pequena, o que diminui sua importância perante outras, mais frequentes.

Então, o ideal é fazer um balanceamento para que palavras frequentes, mas não tão frequentes, sejam consideradas mais relevantes que as demais. Nesse sentido, o TF-IDF surge como uma implementação dessa ideia. Trata-se de uma ponderação entre a frequência de cada *token* ao longo de um documento, $tf(t)$, e a frequência inversa dos documentos em que ele aparece, $idf(t)$,conforme evidenciado pela equação 3.1 [31].

$$tfidf(t) = tf(t) \cdot idf(t) = tf(t) \cdot \left[log \frac{1+n}{1+df(t)} + 1 \right] \tag{3.1}$$

Tem-se que n é o número total de documentos no *corpus*, $if(t)$ é o número de vezes que o termo t aparece em um documento e $df(t)$ é o número de documentos no *corpus* que contêm o termo t. Cada *token* t_j, $j = 0, 1, ..., D-1$ do vocabulário, de tamanho D, ocupa uma posição do vetor unidimensional mostrado na equação 3.2.

$$vocabulario = [t_0, t_1, ..., t_j, ..., t_{D-1}] \tag{3.2}$$

A matriz de *features* a ser inserida na SVM, expressa pela equação 3.3, é preenchida de acordo

com o TF-IDF de cada *token* do vocabulário, segundo a equação 3.4.

$$Feature = \begin{bmatrix} f_1^1 & f_2^1 & ... & ... & ... & f_D^1 \\ f_1^2 & f_2^2 & ... & ... & ... & f_D^2 \\ ... & ... & ... & f_j^i & ... & ... \\ f_1^N & f_2^N & ... & ... & ... & f_D^N \end{bmatrix} \tag{3.3}$$

$$Feature = \begin{bmatrix} tfidf(t_1^1) & tfidf(t_2^1) & ... & ... & ... & tfidf(t_D^1) \\ tfidf(t_1^2) & tfidf(t_2^2) & ... & ... & ... & tfidf(t_D^2) \\ ... & ... & ... & tfidf(t_j^i) & ... & ... \\ tfidf(t_1^N) & tfidf(t_2^N) & ... & ... & ... & tfidf(t_D^N) \end{bmatrix} \tag{3.4}$$

Cada linha i refere-se a um documento f^i, $i = 0, 1, ..., N-1$, tal que N é o número total de documentos no conjunto de treino. Cada coluna f_j, $j = 0, 1, ..., D-1$ representa um *token* do vocabulário e as colunas de cada documento indicam se o *token* relativo à posição j é relevante ou não, de acordo com o TF-IDF calculado para ele. Assim, o que se tem é que $f_j^i = tfidf(t_j^i)$, tal que i indica o documento e j indica a posição do *token*.

3.3.2.1 Truncamento

Talvez não seja interessante utilizar todo o vocabulário construído e sim, somente as palavras mais relevantes presentes nele. A motivação para cogitar essa abordagem está no fato de que palavras pouco relevantes em pouco acrescentam na discriminação das classes. Nesse sentido, os três vocabulários gerados até então - Bow, BoW + *stopwords* e BoW + *stopwords* + *stemming* - são duplicados e deles, retiram-se apenas as *features* mais frequentes.

Esse processo é feito através da ordenação do vocabulário em ordem decrescente de frequência, $tf(t)$, e apenas as 200 (duzentas) palavras mais relevantes são selecionadas para compor o novo vocabulário. Fala-se, então, em vocabulário truncado. Esse valor foi escolhido, pois os 200 (duzentos) primeiros *tokens* representam aproximadamente 50% da frequência total de palavras do vocabulário gerado a partir apenas da limpeza dos dados e tokenização, conforme mostrado na figura 3.7. Em outras palavras, ao ordenar os *tokens* por frequência, tem-se que os 200 (duzentos) mais frequentes representam 50% da soma de todas as frequências aferidas nos mais de cinco mil *tokens* do vocabulário de BoW.

Caso esses *tokens* sejam suficientemente bons para discriminar os dados perante as classes verdadeira e falsa, ganha-se em velocidade de processamento, pois um vocabulário menor representa menos *features* para serem computadas pela SVM e, consequentemente, ganha-se também em velocidade. O problema que o truncamento do vocabulário pode acarretar é que, caso palavras demais estejam sendo desconsideradas, o modelo pode ter dificuldade em discernir veracidade de falsidade, causando assim, um erro maior no final.

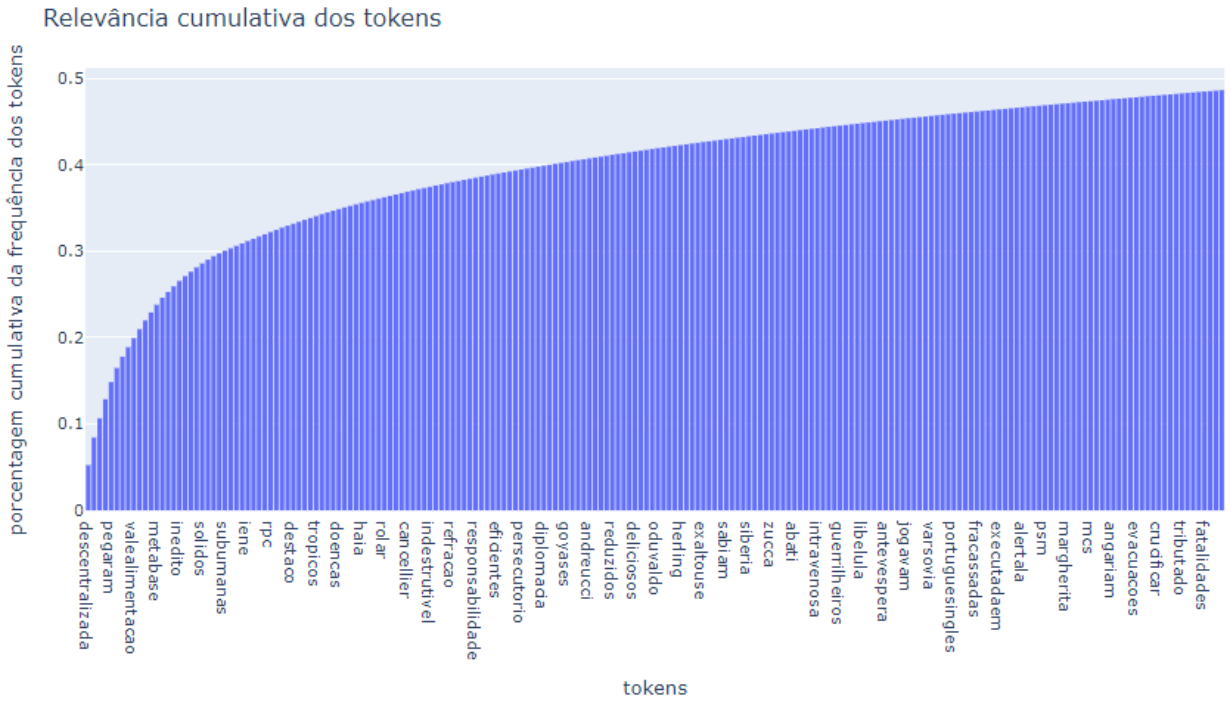

Figura 3.7 – Porcentagem acumulada da frequência dos tokens de BoW.

.

3.3.3 Saída do *Bag of Words*

A quantidade de exemplos em todos os conjuntos de dados tratados será a mesma, $N = 5760$, já que esse é o número de documentos presentes nos conjuntos de dados de treino. O que muda, de um para outro, dentre os seis gerados, é o vocabulário. Ao serem aplicadas diferentes técnicas de NLP para construí-los, o tamanho e/ou a forma dos *tokens* varia. As especificidades de cada um dos vocabulários estão descritos na tabela 3.7.

O vetor de rótulos, expresso na equação 3.5, não sofre alterações, afinal, sua função é de servir de gabarito para o modelo.

$$Label = \begin{bmatrix} l^1 \\ l^2 \\ \dots \\ l^i \\ \dots \\ l^N \end{bmatrix} \tag{3.5}$$

Cada linha i diz respeito à *label* de um documento e o seu valor, l^i, indica a qual classe ele

Tabela 3.7 – Diferenças entre os vocabulários dos conjuntos de dados a serem inseridos na SVM.

Conjunto de dados	**Tamanho do vocabulário**	**10 primeiros *tokens* do vocabulário**
BoW	$D = 78672$	['ASPAS', 'EUR', 'PS', 'PS0', 'REDESOCIAL', 'SS', 'SS0', 'URL', 'a0', 'aa', 'aabb', 'aabertura', 'aachen', 'aapesar', 'aarhus', 'aaron', 'aashish', 'ab', 'abacaxi', 'abadia', 'abaete', 'abafa', 'abafada', 'abafadas', 'abafado']
BoW + *stopwords*	$D = 78516$	['ASPAS', 'EUR', 'PS', 'PS0', 'REDESOCIAL', 'SS', 'SS0', 'URL', 'a0', 'aa', 'aabb', 'aabertura', 'aachen', 'aapesar', 'aarhus', 'aaron', 'aashish', 'ab', 'abacaxi', 'abadia', 'abaete', 'abafa', 'abafada', 'abafadas', 'abafado']
BoW + *stopwords* + *stemming*	$D = 38432$	['a0', 'aa', 'aabb', 'aabert', 'aachen', 'aapes', 'aarhu', 'aaron', 'aashish', 'ab', 'aba', 'abacax', 'abad', 'abaet', 'abaf', 'abag', 'abair', 'abaix', 'abaixoassin', 'abajur', 'abal', 'abaladiss', 'abalarams', 'abalro', 'aban']
BoW trunc	$D = 200$	['ASPAS', 'acao', 'acordo', 'afirma', 'afirmou', 'agora', 'ainda', 'alem', 'ano', 'anos', 'antes', 'ao', 'aos', 'apenas', 'apos', 'as', 'assim', 'ate', 'bem', 'brasil', 'camara', 'casa', 'caso', 'com', 'como']
BoW trunc + *stopwords*	$D = 200$	['ASPAS', 'acao', 'acordo', 'advogado', 'aecio', 'afirma', 'afirmou', 'agora', 'ainda', 'alem', 'alguns', 'ano', 'anos', 'antes', 'apenas', 'apos', 'aqui', 'assim', 'ate', 'bem', 'brasil', 'cada', 'camara', 'campanha', 'casa']
BoW trunc + *stopwords* + *stemming*	$D = 200$	['acontec', 'acord', 'advog', 'afirm', 'agor', 'aind', 'alem', 'algum', 'ano', 'ant', 'ao', 'apen', 'apo', 'apresent', 'as', 'asp', 'ate', 'bem', 'brasil', 'cam', 'campanh', 'cas', 'cham', 'cheg', 'cidad']

pertence, de acordo com a equação 3.6.

$$l^i = \begin{cases} 1 & \text{se } f^i \text{ é uma notícia falsa} \\ 0 & \text{se } f^i \text{ é uma notícia verdadeira} \end{cases} \tag{3.6}$$

As predições dos modelos para cada um dos conjuntos de dados supracitados são comparadas à matriz de *labels* e é feita uma aferição de qualidade do aprendizado.

3.3.4 *Linguistic Features*

Como citado anteriormente na seção 2.6.3, Zhou, Burgoon, Twitchell, Qin e Nunamaker [43] propuseram usar os seguintes atributos de um texto para realizar o treinamento:

- Número de verbos, advérbios e adjetivos;
- Complexidade do Texto;

- Quantidade de aparições de sinais de pontuações em uma sentença;
- Número de verbos modais e voz passiva;
- Frequência de uso de pronomes pessoais;
- Cálculo da soma do número de adjetivos e advérbios divida pela soma do número de substantivos e verbos;
- Diversidade;
- Quantidade de erros ortográficos;

Para iniciar a parte prática do treinamento das SVMs, foi preciso, primeiro, extrair cada atributo descrito acima. Felizmente, estes já estão pré-calculados na base do Fake.BR. Assim a matriz 3.7 exemplifica como a matriz de exemplos de entrada fica, tal que cada linha é uma notícia individual e cada coluna é um atributo.

$$Feature = \begin{bmatrix} \begin{pmatrix} f_1^1 & f_2^1 & f_3^1 & ... & f_{10}^1 \end{pmatrix} \\ \begin{pmatrix} f_1^2 & f_2^2 & f_3^2 & ... & f_{10}^2 \end{pmatrix} \\ \begin{pmatrix} ... & ... & ... & ... & ... \end{pmatrix} \\ \begin{pmatrix} f_1^n & f_2^n & f_3^n & ... & f_{10}^n \end{pmatrix} \end{bmatrix} \tag{3.7}$$

Ao final tem-se uma matriz de $Feature$ de dimensões n por 8.

Feito isso, há a necessidade de criar outra matriz contendo os rótulos para prosseguir com o treinamento das SVMs. Os arquivos provenientes do banco de dados já estão separados em duas pastas, tal que uma delas contém as 'Notícias Falsas' e a outra contém as 'Notícias Verdadeiras'. Então, aproveita-se desta separação para criar um novo *array* chamado *Label* e nele é armazenado o rótulo de cada notícia, dependendo da pasta em que ela se encontra igual ao feito na parte do *Bag of Words* que pode ser visto na equação 3.5.

3.4 Treinamento do Modelo

Após tratar os dados iniciais e gerar as matrizes de dados de entrada, a próxima etapa é servir-se delas para treinar um modelo de ML, de forma a torná-lo capaz de classificar notícias em falsas ou verdadeiras. Um fluxograma representando os processos envolvidos nessa tarefa é apresentado na figura 3.8. A saída final do treinamento, bem como do projeto como um todo, é o melhor modelo treinado, dentre os verificados, e uma métrica de qualidade, indicando o quão bom ele é.

A abordagem de ML escolhida foi a de SVM, que realiza a classificação através da procura por hiperplanos em um espaço multidimensional que distingam bem as classes. As SVMs podem ser configuradas de maneiras diferentes, que determinam a forma com que essa busca se dá. Assim, o primeiro passo é executar diversos testes com parâmetros de configuração distintos, a fim de encontrar os melhores, neste caso, os que produzem um modelo com maior *F1-score*. A esse

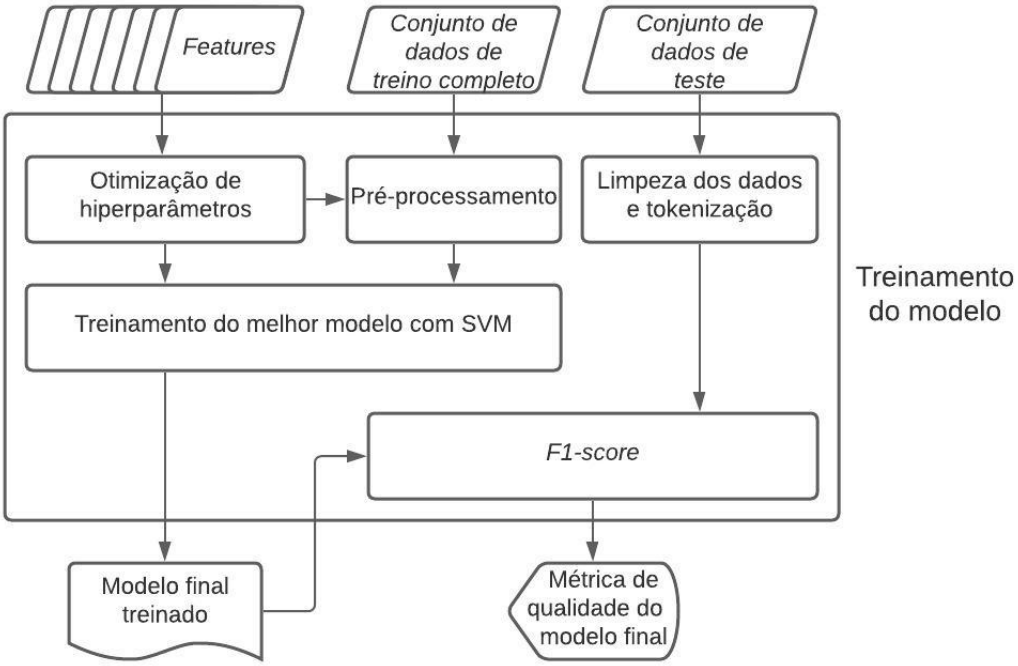

Figura 3.8 – Fluxograma dos processos internos do treinamento do modelo.

processo, dá-se o nome de seleção de hiperparâmetros e a estratégia para comparar os resultados é a validação cruzada.

Uma vez estabelecida a melhor técnica de extração de *features* e os melhores hiperparâmetros dentre os candidatos, eles são empregados no pré-processamento do conjunto de dados de treino completo e no treinamento da SVM "final", respectivamente. A métrica de qualidade, então, é calculada a partir do conjunto de dados de teste, até então não utilizado.

3.4.1 Otimização de Hiperparâmetros

O principal hiperparâmetro a ser definido na configuração das SVMs é a função de *kernel*, cujo papel é o de calcular a similaridade entre cada par de instâncias para encontrar os vetores de suporte. Os vetores de suporte se referem às instâncias mais próximas de um hiperplano e são as distâncias entre elas e o hiperplano que definem a margem, que se deseja maximizar. Optou-se por testar três *kernels* diferentes ao longo dos experimentos. São eles: linear, polinomial e RBF. Uma representação do funcionamento da SVM utilizando cada um deles é ilustrado na figura 3.9.

O parâmetro de regularização C é comum a todos os *kernels* e sua função é de controlar a complexidade dos modelos. Ele serve como um termo que penaliza a margem das funções de fronteira. Ou seja, Quanto maior é o C, menor é a distância entre os vetores de suporte e o hiperplano. Quanto menor é essa distância, maior é a variância do modelo e, consequentemente, maior é a sua complexidade. Um C grande demais pode ocasionar em *overfitting*, bem como um C pequeno demais pode causar *underfitting*.

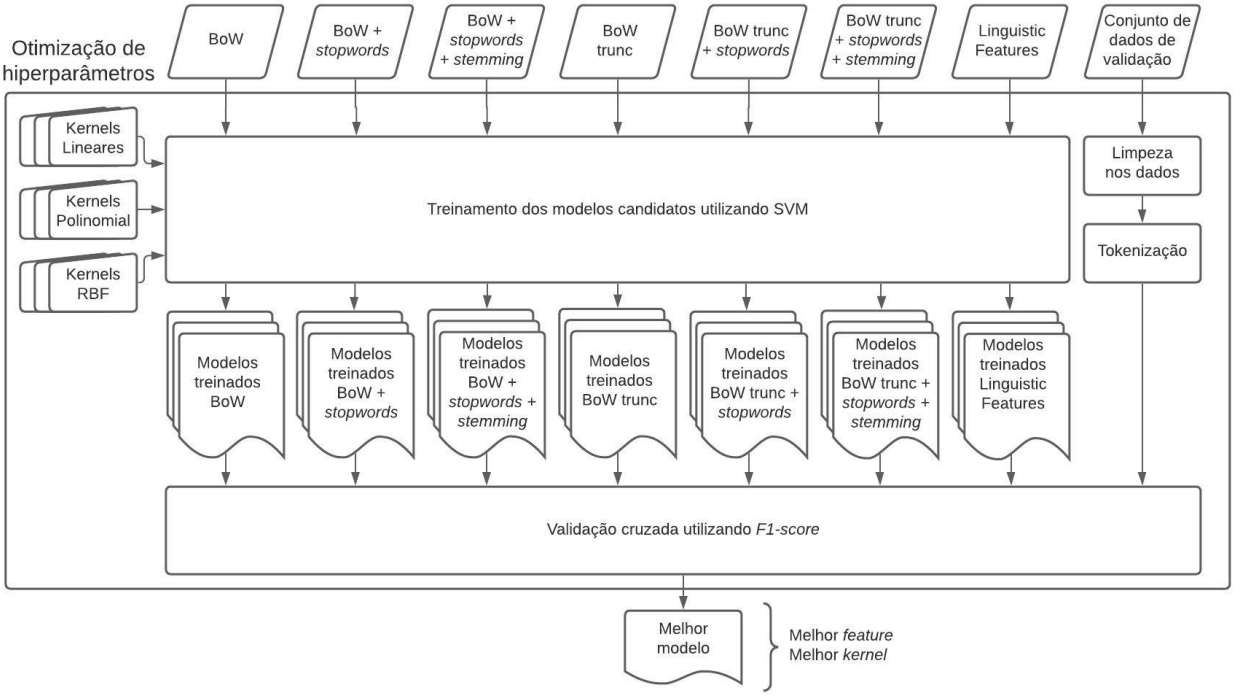

Figura 3.9 – Exemplos da maneira com que os *kernels* linear, polinomial de grau três e RBF atuam nas SVMs.

O *kernel* linear mede a similaridade através da computação do produto interno entre as amostras, conforme mostrado na equação 3.8 [31].

$$K(x, x') =< x, x' > \tag{3.8}$$

Como o *kernel* linear não utiliza-se do truque do *kernel* para calcular o produto interno, pode ser muito custoso para o computador. Ele requer que os cálculos sejam desenvolvidos em um espaço dimensional maior do que o original e quanto maior é o número de atributos, maior é a demanda por recursos computacionais.

O *kernel* de base radial (RBF) procura pela similaridade através do cálculo da exponencial da distância entre duas instâncias ponderadas por uma variável *gamma* (γ), conforme mostrado na equação 3.9 [31].

$$K(x, x') = exp(-\gamma||x - x'||^2) \tag{3.9}$$

O termo γ do *kernel* RBF é inversamente proporcional ao raio de influência de uma única amostra. Ou seja, quanto menor é o γ, maior é o raio de influência. O RBF retorna o quão próximas duas instâncias estão, em termos de distância euclidiana, e caso estejam distantes demais, as considera descorrelacionadas. O γ define o quão distante elas precisam estar para que uma não interfira na outra.

Se γ é grande demais, o raio é pequeno demais e cada vez menos instâncias são agrupadas pela sua similaridade. O modelo vai se tornando cada vez mais específico e, com isso, sua variância e sua complexidade aumentam, podendo causar *overfitting*. Se o γ é pequeno demais, as instâncias passam a influências muitas outras instâncias e vai ficando cada vez mais difícil de encontrar uma distinção clara entre as classes. O modelo vai se tornando simples demais e, com isso, o viés aumenta, podendo causar *underfitting*.

O *kernel* polinomial encontra a similaridade por meio do cálculo de um polinômio, conforme expresso pela equação 3.10 [31]:

$$K(x, x') = (\gamma < x, x' > +r)^d \tag{3.10}$$

O que define o grau do polinômio em questão é o parâmetro *degree* (d). Quanto maior é o grau do polinômio, mais a função de fronteira se adapta aos dados de treino. Logo, a variância e a complexidade também aumentam. Quanto menor é o grau, mais simples é a função que separa os dados e, assim, maior é o viés. Se o d for grande demais, pode causar *overfitting* e, se for pequeno demais, pode causar *underfitting*.

O parâmetro *coef0* (r), muitas vezes chamado de viés, e sua função é a de compensar vetores que não estão centralizados em zero. Em outras palavras, ele normaliza os dados. Ele serve como *offset* de similaridade, pois mesmo quando as duas instâncias são ortogonais, o resultado da similaridade é igual ao coeficiente estipulado, e não zero. O parâmetro *gamma*, bem como no *kernel* RBF, funciona como um regulador to raio de influência das instâncias.

A figura 3.10 detalha os processos que ocorrem durante a otimização dos hiperparâmetros, cujo objetivo é encontrar o melhor modelo. Para cada um dos três *kernels* a serem comparados, se estabelece um conjunto de parâmetros a serem testados, a fim de encontrar a melhor combinação deles. Os parâmetros a serem combinados em cada treinamento estão descritos na tabela 3.8.

Tabela 3.8 – Conjuntos de hiperparâmetros testados para cada uma das funções *kernel* utilizadas.

Kernel	C	***Gamma***	***Degree***	***Coef0***
Linear	$[2^{-1}, 2^0, 2^1, 2^2, 2^3, 2^4]$			
Polinomial	$[2^{-1}, 2^0, 2^1, 2^2, 2^3, 2^4]$	$[1^{-3}, 1^{-2}, 1^{-1}, 1^0]$	$[2, 3, 4]$	$[0, 1^1, 1^2]$
RBF	$[2^{-1}, 2^0, 2^1, 2^2, 2^3, 2^4]$	$[1^{-3}, 1^{-2}, 1^{-1}, 1^0]$		

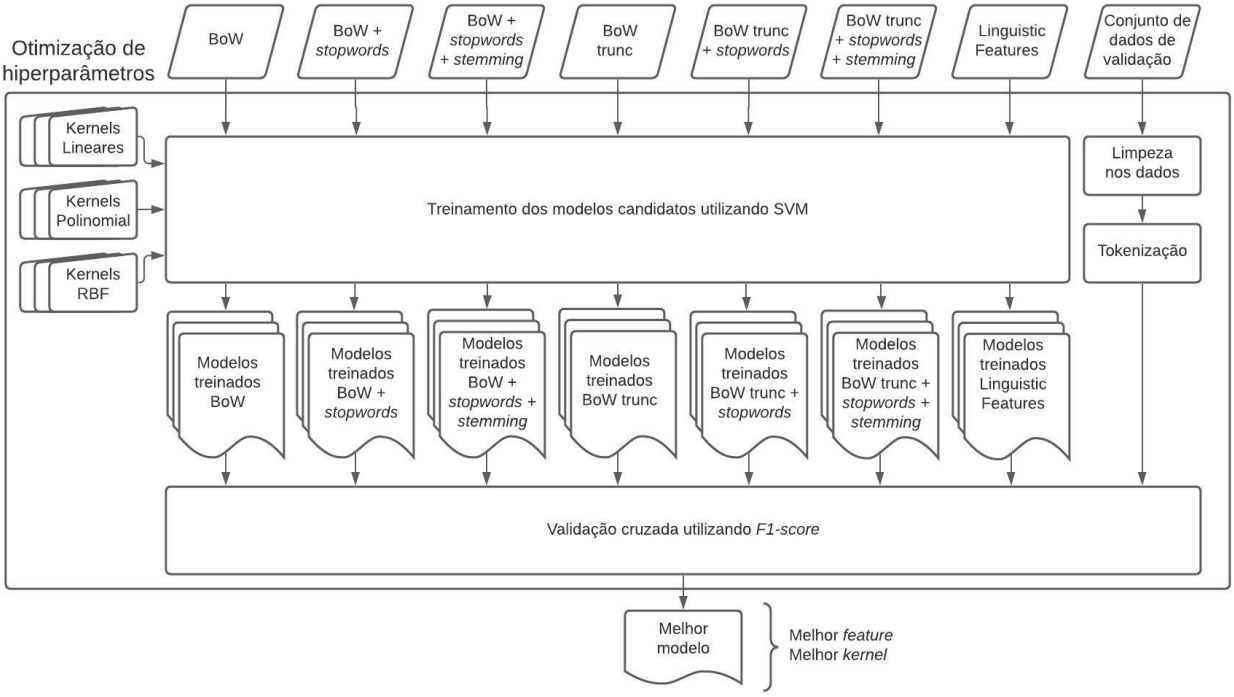

Figura 3.10 – Fluxograma dos processos internos da otimização de hiperparâmetros.

.

A partir dos valores especificados, foi criada uma variável chamada *param_grid*

$$param\_grid = \begin{bmatrix} \begin{bmatrix} 'kernel' : ['linear'] \\ 'C' : [0.2, 2^0, 2^1, 2^2, 2^3, 2^4] \end{bmatrix} \\ \begin{bmatrix} 'kernel' : ['rbf'] \\ 'C' : [0.2, 2^0, 2^1, 2^2, 2^3, 2^4] \\ 'gamma' : [1, 0.1, 0.01, 0.001] \end{bmatrix} \\ \begin{bmatrix} 'kernel' : ['poly'] \\ 'C' : [0.2, 2^0, 2^1, 2^2, 2^3, 2^4] \\ 'gamma' : [1, 0.1, 0.01, 0.001] \\ 'degree' : [2, 3, 4] \\ 'coef0' : [0, 1^1, 1^2] \end{bmatrix} \end{bmatrix} \tag{3.11}$$

A equação 3.11 consiste em um *array* com os valores de cada argumento para diferentes tipos de *kernel*. A matriz é útil, pois simplifica o processo de validação cruzada, que passa a ser executado através de um *pipeline*, e não de uma extensa cadeia de códigos de treinamento. O *pipeline* é inserido na função *GridSearch*, do *Scikit Learn* [31], e a saída são as métricas para cada um dos modelos treinados.

A seguir, é descrito, em forma de pseudocódigo, como a função de *GridSearch* executa o processo

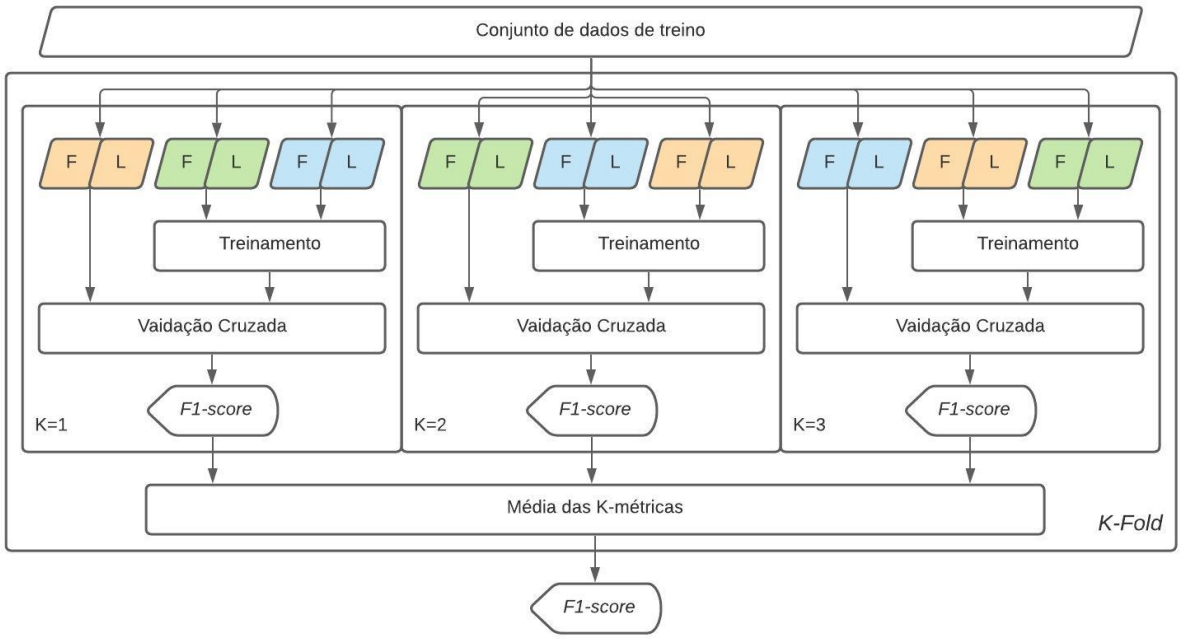

Figura 3.11 – Representação de processo de *K-fold* com K=3.

.

de treinamento do modelo de SVM.

Algoritmo 1: Treinamento

Entrada: $param\_grid$, $features$, $labels$
Saída: modelo_treinado e suas métricas de qualidade
início
  para cada *tipo de kernel* **faça**
    GridSearch($param\_grid[kernel]$, $features$, $labels$);
  fim
fim

Além disso, para cada um dos modelos distintos, foi aplicada a técnica de *K-fold* para garantir uma validação cruzada mais eficaz nos dados de treino, conforme mostrado na figura 3.11. O conjunto de dados é dividido em K subgrupos, a serem combinados de maneiras distintas para gerarem K métricas de qualidade.

O objetivo desse método é que se tenha resultados menos tendenciosos, já que se tenta reduzir o viés gerado pela própria divisão dos dados. Quanto maior é o K, mais assertivos se tornam os resultados, porém maior é a demanda por recursos computacionais, dado que o número de modelos treinados aumenta. Assim, optou-se por um $K = 3$, através do qual obtém-se resultados razoáveis sem demandar demais do computador. Afinal, apenas as combinações de parâmetros já origina mais de mil e setecentos modelos. Ao acrescentar o *K-fold*, esse número triplica.

O treinamento de todas as combinações de extrações de atributos, parâmetros e *k-folds* levou,

aproximadamente, 90 horas para ser concluído. Inicialmente, utilizar-se-ia as mesmas combinações de parâmetros para o treinamento de todas as matrizes de atributos. Porém, durante o treinamento das variáveis linguísticas, à medida em que o *gamma* aumentava, o tempo de treinamento dos modelos crescia exponencialmente, o que se mostrou como um impedimento em termos de recurso para que todos os experimentos pretendidos fossem realizados.

Ao aumentar o parâmetro *gamma*, reduz-se o raio de influência das instâncias e o modelo se torna mais complexo e menos generalista. Enquanto os modelos de atributos linguísticos utilizando $\gamma \leqslant 1^{-2}$ levavam entre 1 minuto e 2 horas para serem treinados, a depender do *kernel*, os modelos com $\gamma \geqslant 1^{-1}$ levavam em torno de dias. Por esse motivo, optou-se por removê-los.

Além disso, à princípio, os treinos do *kernel* polinomial seriam feitos com graus somente de 2 a 4. Porém, ao analisar os resultados dos modelos de BoW, cujo melhor parâmetro de grau encontrado foi 4, surgiu a curiosidade a respeito dos graus maiores e o que resultaria deles. Então, utilizou-se a combinação prévia dos parâmetros C e *gamma*, dessa vez variando o grau do polinômio entre 5 e 6, para treinar mais modelos.

No final das contas, foram treinados 1758 (mil, setecentos e cinquenta e oito) modelos candidatos distintos. Como cada um foi treinado três vezes, por conta do *K-fold*, o número total de treinamentos realizados foi de 5274 (cinco mil, duzentos e setenta e quatro). O resultado da média das métricas dos *folds* de cada modelo foi armazenado e, assim, obteve-se uma tabela de resultados contendo 1758 aferições de qualidade.

Para que se pudesse comparar os resultados de cada métrica, foram inseridas quatro delas na validação cruzada, por meio de um vetor, especificado pela equação 3.12.

$$scoring = \Big[acuracia \quad f1 \quad precisao \quad sensibilidade\Big] \tag{3.12}$$

A métrica escolhida para determinar qual é o melhor modelo foi o *F1-score*, pois destaca-se por ser mais robusta que a acurácia, afinal, leva em conta tanto a precisão quanto a sensibilidade (*recall*) dos resultados. Espera-se, no entanto, que os resultados não diferenciem-se tanto da acurácia em razão dos dados iniciais estarem balanceados, ou seja, possuírem 50% de rótulos verdadeiros, e 50% de rótulos falsos.

A partir dos resultados obtidos, foram selecionados os melhores modelos para cada um dos métodos de extração de atributos, a fim de compará-los, e o melhor dentre esses foi designado como modelo final. Ao estabelecer qual é o melhor método de extração e os melhores parâmetros de configuração da SVM, o modelo final foi treinado com base no conjunto de dados de treino completo, ou seja, no conjunto de treino utilizado na validação cruzada, acrescido do conjunto de validação. Por fim, são medidas as notas de *F1-score* do modelo final e com isso, conclui-se a etapa de experimentação do projeto.

4 Resultados

Com o intuito de analisar os resultados e fazer uma comparação entre os diferentes modelos treinados, foram estudados os melhores hiperparâmetros para cada combinação de *kernel* e *feature*. Na primeira parte se discute a relevância de cada um desses parâmetros e como eles influenciam na complexidade, variância e viés dos modelos.

A métrica utilizada como referência para identificar quais são as melhores configurações é o *F1-score*. Nos entanto, são apresentadas outras métricas além dele: precisão, sensibilidade e acurácia. Através delas, se faz uma análise a respeito do desempenho de cada modelo na segunda parte da análise dos resultados. Por fim, discute-se a nota do modelo final do projeto.

4.1 Análise de hiperparâmetros

Na etapa de otimização de hiperparâmetros, o objetivo foi encontrar as melhores associações para cada combinação de *feature* e *kernel*. Essa escolha representa um impacto na qualidade final dos modelos. A fim de demonstrar a influência da configuração da SVM sobre os modelos treinados, serão abordados nessa seção os hiperparâmetros descritos abaixo. Além disso, são retomados conceitos relacionados a cada um deles.

- C: trata-se do parâmetro de regularização. Quanto maior é o C, maior é a variância e complexidade. Quanto menor é o C, maior é o viés.
- *gamma* (γ): trata-se do tamanho do raio de influência das instâncias. Quanto maior é o , maior é a variância e a complexidade. Quanto menor é o γ, maior é o viés.
- *degree* (d): trata-se do grau do polinômio a ser utilizado para calcular similaridade. Quanto maior é ele maior é a variância e a complexidade. Quanto menor é o d, maior é o viés.
- *coef0*: trata-se do viés do *kernel* polinomial. Consequentemente, quanto maior é ele, menor é a variância e a complexidade.

Para entender como os modelos se comportaram, são feitas duas análises acerca dos hiperparâmetros treinados. A primeira diz respeito à distribuição de desempenho resultante dos treinamentos. Ela possibilita uma visão mais ampla em relação à influência de cada hiperparâmetro nos modelos, especialmente em termos de incerteza. A segunda é voltada para a forma com que esses hiperparâmetros relacionam-se uns com os outros em cada modelo, visão que se tem nos gráficos de dispersão.

4.1.1 Hiperparâmetro C

A figura 4.1 expressa o desempenho do hiperparâmetro C ao longo dos modelos. Os gráficos são do tipo *box plot* e mostram a maneira com que o desempenho dos diferentes valores testados

de C está distribuído. Esses valores foram gerados a partir do treinamento de SVMs com os três *kernels* - linear, polinomial e RBF - pois o mesmo é um hiperparâmetro comum a todos.

Figura 4.1 – Distribuição do *F1-score* médio do hiperparâmetro C.

Nos gráficos, o tamanho da caixa diz respeito ao desvio padrão, a linha horizontal dentro da caixa representa a mediana e as linhas verticais acima e abaixo das caixas representam os *outliers*. Nesse contexto, *outliers* são modelos que apresentaram um resultado muito divergente dos demais. Foram gerados seis gráficos, um para cada técnica de extração de *feature*. O eixo *F1-score* consiste em uma média dos três *F1-scores* calculados para cada modelo distinto, por conta do *K-fold* implementado.

Os gráficos referentes às técnicas de *Bag of Words*, em geral, mostram que para $C = 1$, pouco se pode dizer a respeito da influência desse parâmetro nos resultados do modelo. O desvio padrão grande indica que houve uma gama de resultados diferentes para os modelos treinados com esse valor de C. Porém, à medida em que o valor de C aumenta, a variabilidade do desempenho diminui e a nota tende a ser mais alta. Assim, pode-se inferir que valores mais altos de C, em especial da ordem das centenas para cima, em geral, reproduzem modelos melhores. A mediana de desempenho dos modelos treinados com valores mais altos de C está em torno de 95%.

No gráfico da distribuição de desempenho dos modelos que utilizaram atributos linguísticos, percebe-se que a tendência observada no *Bag of Words* não se mantém. Na verdade, além do desvio padrão apresentar a tendência a crescer à medida em que se aumenta o valor de C, a nota média diminui. Porém, como não há uma nítida relação entre os valores de C e o desempenho dos modelos, conclui-se que, no caso das *features* linguísticas, não há um valor ótimo de C que garanta um bom resultado.

4.1.2 Hiperparâmetro *Gamma*

A figura 4.2 ilustra a distribuição do desempenho dos modelos de acordo com os valores de *gamma* utilizados para treiná-los. Nessa representação, além dos elementos já descritos anteriormente, aparecem pontos distantes das caixas. Esses pontos representam modelos *outliers* que apresentaram resultados completamente divergentes dos demais.

Os modelos treinados com atributos de *Bag of Words* apresentam uma tendência a fazerem boas predições para valores mais altos de *gamma*. Esse comportamento é similar ao que acontece no caso do parâmetro de regularização C e pode ser explicado através da mesma ideia. O *gamma*, bem como o C, influencia de forma proporcional sobre a variância dos modelos. Dessa forma, *gamma* não pode ser pequeno demais, pois caso contrário, torna o modelo mais simples e se torna difícil dizer o quão bem um modelo será capaz de predizer dados complexos, como é o caso de *Bag of Words*.

Figura 4.2 – Distribuição do *F1-score* médio do parâmetro *gamma*.

Observa-se que há indicativos de que modelos de BoW treinados com *gamma* configurado da ordem dos decimais para cima reproduzem bons resultados. Mesmo para modelos treinados com valor igual à '1', que se caracteriza como o maior valor, ocorrem casos de modelos que apresentam desempenhos *outliers*.

Verifica-se que os modelos treinados com *features* linguísticas seguem o padrão contrário ao dos modelos com BoW. Com o aumento do valor de γ, a qualidade tende a diminuir e, quanto maior é o parâmetro, mais certeza se tem de que o modelo não terá um bom desempenho. Para valores pequenos de γ, modelos desse tipo possuem uma tendência a fazerem melhores predições. Apesar de existir uma certa variabilidade nos resultados, a distribuição se encontra acima de 90%, o que é considerado um bom desempenho.

A explicação por trás desse comportamento pode estar no fato de que a matriz de atributos linguísticos seja, por definição, muito menor do que a do *Bag of Words*. Ela leva em consideração poucas variáveis, enquanto o BoW tem mais de sete mil no caso sem truncamento, e duzentas, no conjunto truncado.

4.1.3 Hiperparâmetros *Degree* e *Coef0*

A figura 4.3 mostra a distribuição de desempenho dos modelos de acordo com cada valor de grau polinomial utilizado para treinar as SVMs com *kernel* polinomial. Observa-se que, tanto para *features* linguísticas, quanto para BoW truncado, os resultados variam muito pouco e localizam-se acima de 90%. Já para BoW não truncado, o desvio padrão aumenta conforme o grau polinomial aumenta.

Espera-se mesmo que modelos treinados com BoW truncado apresentem comportamento semelhante entre sim, pois as matrizes, nos três casos, possuem a mesma quantidade de variáveis. Isso aproxima o grau de complexidade das três abordagens, ao menos em termos de entrada do modelo de ML. O que é interessante nesse cenário de perceber é que os resultados são semelhantes para todos os graus testados e a mediana fica em torno dos 93% para todos eles. O que mais destoa é o grau 4 para o BoW truncado sem *stopwords*, que apresenta uma variabilidade maior, mas ainda assim, mostra-se como uma boa escolha de parâmetro.

De maneira semelhante, os dois graus testados para modelos com atributos linguísticos se mostraram boas escolhas e expressaram resultados próximos. A diferença está no fato de que os treinos com polinômio de grau dois parecem ter ocasionado no surgimento de mais *outliers*. Como as distinções mais significativas se passam nos modelos com BoW sem truncamento, pode-se inferir que o tamanho das matrizes inseridas na SVM influenciam sobre a certeza que se tem a respeito dos resultados esperados em modelos treinados com *kernel* polinomial.

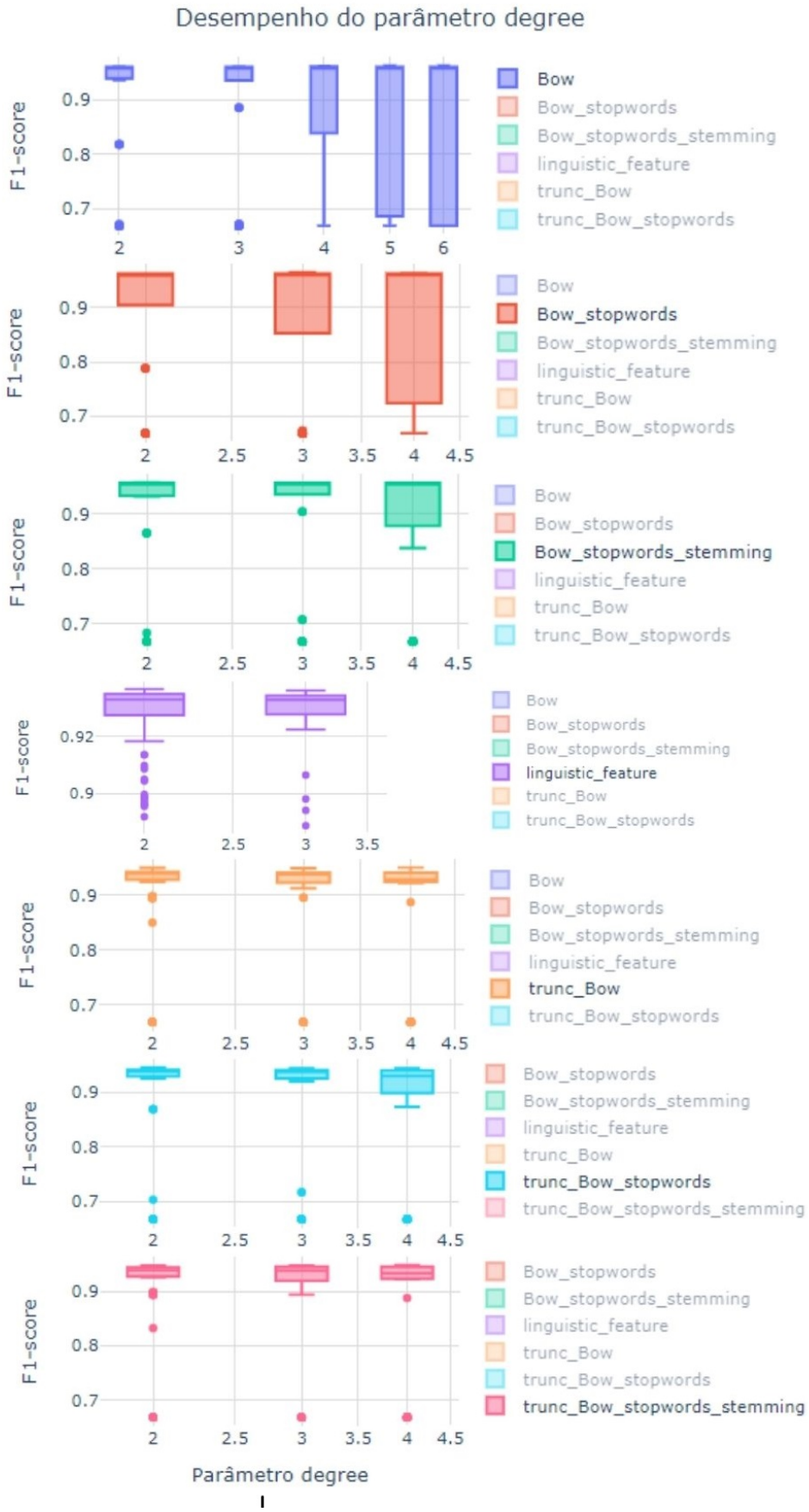

Figura 4.3 – Distribuição do *F1-score* médio do parâmetro *degree*.

Os resultados mais destoantes aparecem nos modelos treinados com BoW sem truncamento. Identifica-se, em todos eles, uma tendência ao aumento da variabilidade à medida em que o grau aumenta. E essa diferença aparece nesses modelos e não nos anteriores justamente por eles serem os mais complexos em termos de tamanho de matriz de *features*. Assim, ao aumentar o grau e, com ele, a qualidade das predições para o conjunto de dados de treino, os modelos se tornam muito mais específicos do que o treinados com menos atributos. Consequentemente, a perda na generalização é muito mais significativa.

Uma hipótese que possivelmente explica a diferença entre o comportamento dos modelos com BoW sem truncamento está na quantidade e na característica de seus atributos. Os vocabulários da BoW completa e da BoW sem *stopwords* possuem comprimentos semelhantes e são muito maiores do que o da BoW sem *stopwords* com *stemming*. Por serem maiores, são mais complexos e por isso, sofrem mais com o aumento da complexidade nos graus polinomiais elevados.

Já entre os dois, apesar de ter um vocabulário menor, o BoW sem *stopwords* mostra-se mais inclinado à incerteza do que o BoW completo. Isso pode ser fruto da natureza dos seus *tokens*, pois ao retirar as palavras muito frequentes em todos os documentos, remove-se viés dos modelos. Sendo mais específicos do que os modelos treinados com BoW completo, tornam-se mais suscetíveis a incertezas no conjunto de validação.

4.1.4 Melhores hiperparâmetros

Nas tabelas 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 e 4.7, são descritos quais foram os parâmetros mais otimizados para cada uma das combinações de *kernel* e *features* testadas.

Tabela 4.1 – Tabela de melhores parâmetros do modelo *Bag Of Words*

BoW				
	C	*gamma*	*coef0*	*degree*
Linear	2.0			
RBF	20.0	0.1		
Poly	0.2	1.0	10.0	4

Tabela 4.2 – Tabela de melhores parâmetros do modelo *Bag Of Words*, com remoção de *stopwords*

BoW + *stopwords*				
	C	*gamma*	*coef0*	*degree*
Linear	2.0			
RBF	2000.0	0.01		
Poly	0.2	0.01	10.0	3

Um resultado que pode ser retirado das tabelas é sobre o parâmetro C nos *kernels* polinomial e RBF. Geralmente o valor desse parâmetro é maior nos *kernels* RBF do que nos polinomiais, o que faz sentido já que normalmente o *poly* é mais complexo, então o parâmetro C seria menor para contrabalancear essa complexidade.

Uma caso curioso observado na tabela 4.3 é o fato do parâmetro C ter sido maior no *kernel* polinomial do que no linear. Isso se dá ao fato do *gamma* ter sido tão pequeno, que faz com que

Tabela 4.3 – Tabela de melhores parâmetros do modelo *Bag Of Words*, com remoção de *stopwords* e *stemming*

BoW + *stopwords* + *stremming*				
	C	*gamma*	*coef0*	*degree*
Linear	20.0			
RBF	200.0	0.1		
Poly	200.0	0.001	100.0	2

Tabela 4.4 – Tabela de melhores parâmetros do modelo *Bag Of Words* truncado

BoW trunc				
	C	*gamma*	*coef0*	*degree*
Linear	2.0			
RBF	20.0	1.0		
Poly	2.0	1.0	0	4

Tabela 4.5 – Tabela de melhores parâmetros do modelo *Bag Of Words* truncado com remoção de *stopwords*

BoW trunc + *stopwords*				
	C	*gamma*	*coef0*	*degree*
Linear	0.2			
RBF	20.0	1.0		
Poly	2.0	1.0	0	2

Tabela 4.6 – Tabela de melhores parâmetros do modelo *Bag Of Words* truncado, com remoção de *stopwords* e *stemming*

BoW trunc + *stopwords* + *stremming*				
	C	*gamma*	*coef0*	*degree*
Linear	20.0			
RBF	200.0	1.0		
Poly	2.0	1.0	0	4

Tabela 4.7 – Tabela de melhores parâmetros do modelo utilizando as *Liguistic-Based Features*

Linguistic-Based Features				
	C	*gamma*	*coef0*	*degree*
Linear	200.0			
RBF	2000.0	0.001		
Poly	20000.0	0.01	100.0	2

o raio seja tão pequeno que o 'C serve para balancear essa diminuição da complexidade mesmo o grau do polinômio ser 2.

Infere-se da tabela dos atributos linguísticos que o parâmetro 'C', dentre todos os treinamentos realizados foi maior, ou igual. O que já é esperado, levando em conta que dentre todos os modelos, ela é a que tem o menor *array* de atributos, consequentemente, é um modelo mais simples do que os com *Bag of Words*.

Com o auxílio das tabelas de resultados, conseguiu-se observar que os *kernels* polinomiais se saem melhor com os modelos de *Bag of Words* sem o truncamento e *Linguistic Features*, enquanto que o *kernel* RBF é melhor utilizando o *BoW* com truncamento.

4.1.5 Análise comparativa das métricas entre métodos de extração de *feature*

Tabela 4.8 – Tabela de resultados dos melhores modelos para BoW

BoW				
	F1	Precisão	Sensibilidade	Acurácia
Linear	0.9616	0.9641	0.9591	0.9616
RBF	0.9620	0.9638	0.9601	0.9619
Poly	0.9623	0.9638	0.9608	0.9623

Tabela 4.9 – Tabela de resultados dos melhores modelos para BoW, com remoção de *stopwords*

BoW + *stopwords*				
	F1	Precisao	Sensibilidade	Acuracia
Linear	0.9632	0.9640	0.9626	0.9631
RBF	0.9624	0.9636	0.9612	0.9623
Poly	0.9648	0.9599	0.9698	0.9645

Tabela 4.10 – Tabela de resultados dos melhores modelos para BoW, com remoção de *stopwords* e *stemming*

BoW + *stopword* + *stemming*				
	F1	Precisao	Sensibilidade	Acuracia
Linear	0.9569	0.9606	0.9532	0.9569
RBF	0.9569	0.9597	0.9543	0.9569
Poly	0.9576	0.9597	0.9556	0.9576

Tabela 4.11 – Tabela de resultados dos melhores modelos para BoW truncado

BoW trunc				
	F1	Precisao	Sensibilidade	Acuracia
Linear	0.9464	0.9536	0.9394	0.9467
RBF	0.9517	0.9582	0.9453	0.9519
Poly	0.9508	0.9509	0.9508	0.9506

Tabela 4.12 – Tabela de resultados dos melhores modelos para BoW truncado, com remoção de *stopwords*

BoW trunc + *stopwords*				
	F1	Precisao	Sensibilidade	Acuracia
Linear	0.9424	0.94412	0.9408	0.9423
RBF	0.9480	0.9491	0.9470	0.9479
Poly	0.9458	0.9427	0.9491	0.9454

Tabela 4.13 – Tabela de resultados dos melhores modelos para BoW truncado, com remoção de *stopwords* e *stemming*

BoW trunc + *stopwords* + *stremming*				
	F1	Precisao	Sensibilidade	Acuracia
Linear	0.9401	0.9482	0.9321	0.9404
RBF	0.9500	0.9562	0.9439	0.9501
Poly	0.9479	0.9479	0.9480	0.9477

Tabela 4.14 – Tabela de resultados dos melhores modelos para atributos linguísticos

Linguistic-Based Features				
	F1	Precisao	Sensibilidade	Acuracia
Linear	0.9352	0.9285	0.9420	0.9347
RBF	0.9343	0.9319	0.9368	0.9342
Poly	0.9365	0.9330	0.9401	0.9361

Pode-se observar entre as tabelas que dentre todos os modelos treinados o que se saiu melhor foi o *Bag of Words* com a utilização das *stopwords*, pois o BoW inteiro é complexo demais por conta da quantidade de palavras usadas, e em contra partida, a utilização do *stemming* torna o BoW mais simples, um dos motivos disso é por não haver muitos algoritmos nesta área utilizando da língua portuguesa como base, fazendo com que as vezes, palavras sejam alteradas de maneira errônea. Acredita-se que caso fossem usados de algoritmos melhores de *stemming* para realização do treinamento, o resultado dele se sairia bem melhor.

É interessante observar também, que dentre os modelos que utilizam do *Bag of Words*, os treinamentos com o truncamento são os que se saem pior, devido ao fato de que se pode estar removendo palavras que são relevantes para a discriminação entre as classes quando se utiliza das 200 palavras mais frequentes. Julga-se que possíveis trabalhos futuros que utilizam de uma quantidade maior do número de palavras mais frequentes se deem com resultados melhores dos que foram encontrados neste trabalho.

Outro detalhe importante de se notar é de que dentre todos os modelos treinados, o que se saiu pior foram os treinamentos utilizando as *Linguitic-Based Features*. Isso se da pelo fato desse modelo ser mais simples que os outros pelo detalhe dele possuir menos *features* que os demais, o que faz com que ele se saia pior nos resultados. Vale ressaltar que mesmo tendo obtido os piores resultados entre todos os treinamentos, ainda sim foram resultados satisfatórios, já que o modelo consegue predizer com 93,61% de acurácia notícias falsas.

Um fato curioso que conseguiu retirar dos resultados obtidos, foi o fato dos valores da acurácia terem sido muito próximos dos valores de f1, o que mostra que os rótulos dos conjuntos de dados estão bem distribuídos entre eles.

4.1.6 Modelo Final

Após todos os testes e análises feitas sobre os resultados dos modelos, o vencedor e escolhido para ser treinado utilizando o conjunto de dados de treino inteiro foi a SVM com *kernel* polinomial

construído a partir de uma matriz de *Bag of Words* sem *stopwords*. Ele apresentou um *F1-score* de aproximadamente 95,63%, o que é considerado um bom desempenho de predição. A figura 4.4 mostra como ficou a matriz de confusão do modelo final.

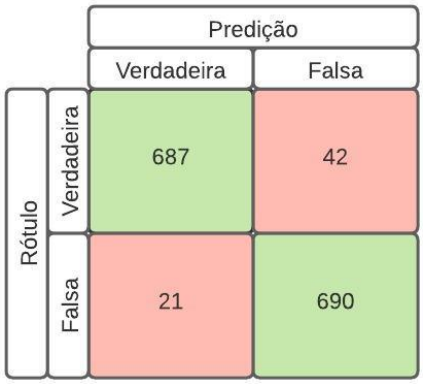

Figura 4.4 – Matriz de confusão do melhor modelo.

Nota-se, na matriz de resultados apresentada, que o classificador foi capaz de identificar 687 (seiscentos e oitenta e sete) notícias verdadeiras como sendo verdadeiras, dentre as 729 (setecentos e vinte e nove) rotuladas como verdadeiras, ou seja, acertou que se tratava de uma informação verdadeira em 94,24% das vezes. Além disso, classificou 690 (seiscentos e noventa) notícias falsas como sendo falsas, dentre as 711 (setecentos e onze) rotuladas como falsas, o que representa cerca de 97,05% de qualidade nas predições de notícias *fake*.

Constata-se que o melhor modelo se sai melhor na tarefa de detectar notícias falsas do que verdadeiras, o que era um resultado esperado.

5 Conclusões e Trabalhos Futuros.

5.1 Conclusões.

A distribuição de *fake-news* é um problema muito difiícil de se combater na sociedade moderna por conta de todos os motivos comentados no decorrer do trabalho, porém um dos principais artifícios usados contra ela é a informação. Acredita-se que com este trabalho contribuiu-se para a comunidade respondendo as seguintes perguntas:

- *Pergunta 1: Quais são as melhores features para se utilizar como entrada das SVM?*

  Foram testadas diversas *features* como entrada para as SVM, dentre elas o BoW simples, BoW truncado, utilizando as *stopwords* e o *stemming*, e as *Linguistic-Based features*, como mostrado a partir dos resultados, a melhor escolha de *feature* foi a do *Bag of Words* usando as *stopwords*, atingindo valores de até 96,5% de acurácia para detecção de *fake-news*;

- *Pergunta 2: Qual é o melhor tipo de kernel para se treinar uma SVM?*

  Os *kernels* são pontos cruciais para o treinamento das SVMs, no trabalho foi usado o *kernel* linear, o RBF e o polinomial e chegou-se no resultado de que o *kernel* RBF e polinomial se saíram melhores em todos os casos quando comparado com o linear. Recomenda-se usar o linear apenas em casos de recursos computacionais escaços pelo fato dele ser mais rápido de treinar.

- *Pergunta 3: O tamanho do texto influencia no resultado da classificação?*

  Como constado por [36] as notícias rotuladas como verdadeira costumam ser maiores que as falsas, logo, utilizou do método de truncamento das notícias para avaliar se isso iria ter algum efeito no resultado. Com isso, foi avaliado que os modelos utilizando os textos sem o truncamento se saíram melhores do que os que foram truncados, logo constata que o tamanho dos textos influenciam sim os resultados da classificação.

5.2 Trabalhos Futuros.

Uma contribuição muito importante que um trabalho futuro poderia ter, é a criação de um novo banco de dados com a língua portuguesa tida como base, já que o FAKE.BR CORPUS é um marco importante para o crescimento de pesquisas na área, porém, como dito anteriormente, o fato das notícias serem escritas dentre os períodos de Janeiro de 2016 e Janeiro de 2018 impacta diretamente nos modelos.

Como citado anteriormente, um possível trabalho futuro que possa levar a uma provável melhora dos resultados dos treinamentos é o desenvolvimento de um algoritmo de *stemming*, pois como constado pelo trabalho, o atual algoritmo ainda possui algumas falhas que podem causar uma queda de desempenho.

O presente projeto também pode ter um desdobramento interessante no tocante a utilizar uma variedade maior de números para realizar o truncamento dos *Bag of Words*, já que para este trabalho apresentado, foi usado apenas um único valor de 200.

Bibliografia

[1] Latifa Abdin. “Bots and fake news: the role of WhatsApp in the 2018 Brazilian Presidential election”. Em: *Casey Robertson* 41.1 (2019).

[2] Ethem Alpaydin. *Introduction to Machine Learning*. Cambridge, MA, 2014.

[3] E BECHARA. *Moderna gramática portuguesa*. 37ª ed. Rio de Janeiro: Nova Fronteira, 2009.

[4] Dylan de Beer e Machdel Matthee. “Approaches to Identify Fake News: A Systematic Literature Review”. Em: *Integrated Science in Digital Age 2020*. Ed. por Tatiana Antipova. Cham: Springer International Publishing, 2021, pp. 13–22. ISBN: 978-3-030-49264-9.

[5] Steven Bird, Ewan Klein e Edward Loper. *Natural Language Processing with Python*. Jan. de 2009. ISBN: 978-0-596-51649-9.

[6] Steven Bird, Ewan Klein e Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. "O'Reilly Media, Inc.", 2009.

[7] Anshika Choudhary e Anuja Arora. “Linguistic feature based learning model for fake news detection and classification”. Em: *Expert Systems with Applications* 169 (2021), p. 114171. ISSN: 0957-4174. DOI: <https://doi.org/10.1016/j.eswa.2020.114171>. URL: <https://www.sciencedirect.com/science/article/pii/S095741742030909X>.

[8] Mariano Felice e Lucia Specia. “Linguistic Features for Quality Estimation”. Em: *Proceedings of the Seventh Workshop on Statistical Machine Translation*. Montreal, Canada: Association for Computational Linguistics, jun. de 2012, pp. 96–103. URL: <https://www.aclweb.org/anthology/W12-3110>.

[9] C. French. *Data Processing and Information Technology*. THOMSON LEARNING, 1996. ISBN: 9781844801008. URL: <https://books.google.com.br/books?id=zVCdg7Tg6-AC>.

[10] Agnela da Silva Giusta. “ConcepÃ§Ãde aprendizagem e prÃ¡ticas pedagÃ”. pt. Em: *EducaÃ§Ãem Revista* 29 (mar. de 2013), pp. 20–36. ISSN: 0102-4698. URL: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0102-46982013000100003&nrm=iso>.

[11] Yoav Goldberg. “Neural Network Methods for Natural Language Processing”. Em: *Synthesis Lectures on Human Language Technologies* 10 (abr. de 2017), pp. 1–309. DOI: <10.2200/S00762ED1V01Y201703HLT037>.

[12] Rebecca L Gómez e LouAnn Gerken. “Infant artificial language learning and language acquisition”. Em: *Trends in cognitive sciences* 4.5 (2000), pp. 178–186.

[13] Georgios Gravanis et al. “Behind the cues: A benchmarking study for fake news detection”. Em: *Expert Systems with Applications* 128 (2019), pp. 201–213. ISSN: 0957-4174. DOI: <https://doi.org/10.1016/j.eswa.2019.03.036>. URL: <https://www.sciencedirect.com/science/article/pii/S0957417419301988>.

[14] Nir Grinberg et al. “Fake news on Twitter during the 2016 U.S. presidential election”. Em: *Science* 363.6425 (2019), pp. 374–378. ISSN: 0036-8075. DOI: <10.1126/science.aau2706>. eprint: <https://science.sciencemag.org/content/363/6425/374.full.pdf>. URL: <https://science.sciencemag.org/content/363/6425/374>.

[15] Sparks Hannah e Hannah Frishberg. “Facebook gives step-by-step instructions on how to spot fake news”. Em: *New York Post* 13.2 COVID-19 (2020), p. 331. URL: <https://nypost.com/2020/03/26/facebook-gives-step-by-step-instructions-on-how-to-spot-fake-news/>.

[16] Yuval N. Harari. *Sapiens : a brief history of humankind*. 13ª ed. New York, NY: Harper, 2015. ISBN: 9780062316097.

[17] Thomas Hofmann, Bernhard Schölkopf e Alexander J Smola. “Kernel methods in machine learning”. Em: *The annals of statistics* (2008), pp. 1171–1220.

[18] Thorsten Joachims. “Text categorization with support vector machines: Learning with many relevant features”. Em: *European conference on machine learning*. Springer. 1998, pp. 137–142.

[19] Ioannis Kanaris et al. “Words versus character n-grams for anti-spam filtering”. Em: *International Journal on Artificial Intelligence Tools* 16.06 (2007), pp. 1047–1067.

[20] Man Lan et al. “A comprehensive comparative study on term weighting schemes for text categorization with support vector machines”. Em: *Special interest tracks and posters of the 14th international conference on World Wide Web*. 2005, pp. 1032–1033.

[21] Oxford Languages. *Dicionário de Português*. 2021. URL: <https://languages.oup.com/google-dictionary-pt/>.

[22] Instagram LTA. *Combatting Misinformation on Instagram*. 2021. URL: <https://about.instagram.com/blog/announcements/combatting-misinformation-on-instagram> (acesso em 19/05/2021).

[23] Christopher D. Manning e Hinrich Schütze. *Foundations of statistical natural language processing*. Cambridge, MA: MIT Press, 1999.

[24] Rafael Christian de Matos. “Fake news frente a pandemia de COVID-19”. Em: *Vigilância Sanitária em Debate: Sociedade, Ciência amp; Tecnologia (Health Surveillance under Debate: Society, Science amp; Technology) – Visa em Debate* 8.3 (maio de 2020), pp. 78–85. DOI: <10.22239/2317-269x.01595>. URL: <https://visaemdebate.incqs.fiocruz.br/index.php/visaemdebate/article/view/1595>.

[25] J.P. Meneses. “Sobre a necessidade de conceptualizar o fenômeno das fake news”. pt. Em: *Revista FAMECOS* (jan. de 2018). DOI: <10.15847/obsOBS12520181376>. URL: <https://doi.org/10.15847/obsOBS12520181376> (acesso em 27/01/2021).

[26] Tom Michael Mitchell. *The discipline of machine learning*. Vol. 9. Carnegie Mellon University, School of Computer Science, Machine Learning . . ., 2006.

[27] Ann T. Musgrove et al. “Real or fake? Resources for teaching college students how to identify fake news”. Em: *College & Undergraduate Libraries* 25.3 (2018), pp. 243–260. DOI: <10.1080/10691316.2018.1480444>. eprint: <https://doi.org/10.1080/10691316.2018.1480444>. URL: <https://doi.org/10.1080/10691316.2018.1480444>.

[28] Viviane Moreira Orengo e Christian R Huyck. “A Stemming Algorithmm for the Portuguese Language.” Em: *spire*. Vol. 8. 2001, pp. 186–193.

[29] F. Yergeau P. Hoffman. *UTF-16, an encoding of ISO 10646*. RFC. RFC Editor, fev. de 2000. DOI: <10.17487/RFC2781>. URL: <https://www.rfc-editor.org/info/rfc2781>.

[30] Danny Paskin. “Real or Fake News: Who Knows?” Em: *The Journal of Social Media in Society* 7.2 (2018), pp. 252–273. ISSN: 2325-503x. URL: <https://thejsms.org/index.php/TSMRI/article/view/386>.

[31] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. Em: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[32] Julie Posetti e Alice Matthews. “A short guide to the history of’fake news’ and disinformation”. Em: *International Center for Journalists* 7 (2018), pp. 2018–07.

[33] Victor Raskin. “Linguistics and natural language processing”. Em: *Machine translation: Theoretical and methodological issues* (1987), pp. 42–58.

[34] Marco Antonio Roxo e Seane Melo. “Hiperjornalismo: Uma visada sobre fake news a partir da autoridade jornalística”. Em: *Revista FAMECOS* 25.3 (ago. de 2018), p. ID30572. DOI: <10.15448/1980-3729.2018.3.30572>. URL: <https://revistaseletronicas.pucrs.br/ojs/index.php/revistafamecos/article/view/30572>.

[35] Stuart Russell e Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3ª ed. Prentice Hall, 2010.

[36] Renato M. Silva et al. “Towards automatically filtering fake news in Portuguese”. Em: *Expert Systems with Applications* 146 (2020), p. 113199. ISSN: 0957-4174. DOI: <https://doi.org/10.1016/j.eswa.2020.113199>. URL: <https://www.sciencedirect.com/science/article/pii/S0957417420300257>.

[37] Herbert A Simon. “Information-processing theory of human problem solving”. Em: *Handbook of learning and cognitive processes* 5 (1978), pp. 271–295.

[38] João Henriques de Sousa Júnior et al. “Da Desinformação ao Caos: uma análise das Fake News frente à pandemia do Coronavírus (COVID-19) no Brasil”. Em: *Cadernos de Prospecção* 13.2 COVID-19 (2020), p. 331.

[39] Mercedes Neto e Tatiana Gomes e Fernando Porto e Ricardo Rafael e Mary Hellem Fonseca e Julia Nascimento. “FAKE NEWS NO CENÁRIO DA PANDEMIA DE COVID-19”. Em: *Cogitare Enfermagem* 25.0 (2020). ISSN: 2176-9133. DOI: <10.5380/ce.v25i0.72627>. URL: <https://revistas.ufpr.br/cogitare/article/view/72627>.

[40] Vladimir N Vapnik. “Constructing learning algorithms”. Em: *The nature of statistical learning theory*. Springer, 1995, pp. 119–166.

[41] F. Yergeau. *UTF-8, a transformation format of ISO 10646*. RFC. RFC Editor, nov. de 2003. DOI: <10.17487/RFC3629>. URL: <https://www.rfc-editor.org/info/rfc3629>.

[42] L. Zhou et al. "An exploratory study into deception detection in text-based computer-mediated communication". Em: *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*. 2003, 10 pp.-. DOI: <10.1109/HICSS.2003.1173793>.

[43] LINA ZHOU et al. "A Comparison of Classification Methods for Predicting Deception in Computer-Mediated Communication". Em: *Journal of Management Information Systems* 20.4 (2004), pp. 139–166. DOI: <10.1080/07421222.2004.11045779>. eprint: <https://doi.org/10.1080/07421222.2004.11045779>. URL: <https://doi.org/10.1080/07421222.2004.11045779>.