

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Larissa Alves Trevilato

**Análise Comparativa de Modelos de
Representação de Texto e Métodos de
Aprendizado de Máquina na Classificação de
Notícias Falsas em Português**

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Larissa Alves Trevilato

**Análise Comparativa de Modelos de Representação de
Texto e Métodos de Aprendizado de Máquina na
Classificação de Notícias Falsas em Português**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, como parte dos requi-
sitos exigidos para a obtenção título de Ba-
charel em Ciência da Computação.

Orientadora: Maria Adriana Vidigal de Lima

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2025

Larissa Alves Trevilato

Análise Comparativa de Modelos de Representação de Texto e Métodos de Aprendizado de Máquina na Classificação de Notícias Falsas em Português

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Ciência da Computação.

Trabalho aprovado. Uberlândia, Brasil, 23 de maio de 2025:

Maria Adriana Vidigal de Lima
Orientadora

Claudiney Ramos Tinoco

Bruno Augusto Nassif Travençolo

Uberlândia, Brasil
2025

Para Luciane, Verônica e Leticia: minhas irmãs, as mulheres da minha vida.

Para Thales, meu esposo, o companheiro da minha caminhada.

Agradecimentos

Expresso minha mais profunda gratidão a todos os professores da Faculdade de Computação da Universidade Federal de Uberlândia com quem tive o privilégio de aprender ao longo da graduação. Agradeço por cada aula, palestra, orientação em laboratório, conversa nos corredores e em eventos acadêmicos.

A influência de cada professor deixou marcas em minha formação intelectual, ética e pessoal. Moldaram minha forma de pensar, de enfrentar desafios e de compreender a ciência com humildade e responsabilidade. Cada um deles tem um pouco de si que hoje faz parte de mim e que me compõe.

Me formaram como cientista da computação, mas também como um ser humano questionador, curioso, resiliente e consciente da vastidão do conhecimento que ainda temos por descobrir. Essa compreensão aconteceu somente após noites em claro estudando para as provas e trabalhos complexos, muitas vezes sem compreender qual a necessidade de passar por tudo aquilo.

Cheguei à universidade como uma adolescente oriunda da rede pública de ensino, com uma base educacional limitada pelas condições estruturais do nosso país. Antes mesmo de compreender os fundamentos teóricos da Ciência da Computação, precisei aprender a aprender — e meus professores garantiram que eu estudasse da maneira correta. Requisitaram de mim muito mais do que eu era acostumada e capaz de entregar, elevando o nível de exigência e profundidade em cada passo que eu dei na universidade. Mas, da mesma forma, me entregaram paciência, sabedoria, companheirismo e amor ao ensinar.

Agradeço, em especial, à minha professora orientadora, que me acompanhou neste trabalho com sensibilidade e generosidade. Sua postura inspira não apenas pelo domínio acadêmico, mas também pela humanidade e delicadeza com seus alunos. Foi sua orientação e incentivo que me ajudaram a dar os últimos passos para alcançar a conquista deste diploma.

A todos e a cada um, o meu mais sincero e respeitoso agradecimento. Fui forjada pelos meus professores e devo todo meu amadurecimento às mãos de cada um deles. Cada dificuldade, desafio ou obstáculo que me propuseram me fez alguém que não tem medo de enfrentar os problemas da ciência e também alguém que não teme os problemas da vida.

Resumo

Este trabalho investigou o impacto do aumento progressivo do volume de dados no desempenho de modelos de classificação de notícias falsas em língua portuguesa. Foram avaliadas três técnicas de representação textual — *Bag of Words* (BoW), TF-IDF e Word2Vec — combinadas com quatro classificadores supervisionados: *Support Vector Classifier* (SVC), Regressão Logística, *Multinomial Naive Bayes* e *Random Forest*. Os experimentos utilizaram quatro conjuntos de dados construídos de forma incremental, totalizando até 28 mil amostras. Os resultados mostraram que, embora os ganhos numéricos nas métricas (acurácia, precisão, revocação e *F1-score*) tenham sido suaves — geralmente na ordem de uma casa decimal —, o aumento da base contribuiu para maior estabilidade e robustez dos modelos frente à variabilidade textual. A representação TF-IDF, em especial, obteve os melhores desempenhos médios, com destaque para os classificadores SVC e *Random Forest*, cujos *F1-scores* superaram 0,96. Os resultados confirmam a importância de conjuntos de dados amplos e diversificados na construção de modelos de classificação mais confiáveis e generalizáveis.

Palavras-chave: Processamento de Linguagem Natural, Aprendizado de Máquina, *fake news*, Classificação de Textos, Representação Textual.

Lista de ilustrações

Figura 1 – Fluxograma do processo de classificação de textos, destacando as etapas de Processamento de Linguagem Natural (em azul) e Classificação (em laranja).	27
Figura 2 – Projeção 2D dos vetores das palavras após treinamento com Word2Vec.	44
Figura 3 – Exemplo ilustrativo do funcionamento do SVC em um espaço bidimensional, separando duas classes com o hiperplano e suas margens.	48
Figura 4 – Fronteira de decisão utilizando Regressão Logística.	51
Figura 5 – Fluxograma simplificado do processo de execução dos testes com bases de dados, métodos de representação e classificadores.	64
Figura 6 – Evolução do $F1$ -score dos modelos.	69
Figura 7 – Evolução da acurácia dos modelos.	70
Figura 8 – Evolução da precisão dos modelos.	71
Figura 9 – Evolução da revocação dos modelos.	72
Figura 10 – Radar com os valores de $F1$ -score obtidos pelos modelos de classificação em diferentes representações textuais para a Base 1.	73
Figura 11 – Radar com os valores de $F1$ -score obtidos pelos modelos de classificação em diferentes representações textuais para a Base 2.	74
Figura 12 – Radar com os valores de $F1$ -score obtidos pelos modelos de classificação em diferentes representações textuais para a Base 3.	75
Figura 13 – Radar com os valores de $F1$ -score obtidos pelos modelos de classificação em diferentes representações textuais para a Base 4.	76

Lista de tabelas

Tabela 1 – Composição de notícias nas bases utilizadas, considerando a junção progressiva dos 4 corpora.	32
Tabela 2 – Etapas do Pré-processamento de Texto.	32
Tabela 3 – Vetores iniciais atribuídos a cada palavra no início do treinamento (Word2Vec).	42
Tabela 4 – Vetores finais ajustados para cada palavra após o treinamento do Word2Vec.	43
Tabela 5 – Matriz final de entrada para o classificador com vetores médios (Word2Vec).	46
Tabela 6 – Base de dados de exemplo para treinamento do <i>Random Forest</i>	55
Tabela 7 – Resultados da Base 1 por representação e classificador.	65
Tabela 8 – Resultados da Base 2 por representação e classificador.	66
Tabela 9 – Resultados da Base 3 por representação e classificador.	66
Tabela 10 – Resultados da Base 4 por representação e classificador.	67

Lista de abreviaturas e siglas

PLN	Processamento de Linguagem Natural
AM	Aprendizado de Máquina
BoW	<i>Bag-of-Words</i>
TF-IDF	<i>Term Frequency-Inverse Document Frequency</i>
Word2Vec	<i>Word to Vector</i>
SVC	<i>Support Vector Classifier</i>
NB	<i>Naive Bayes</i>
RF	<i>Random Forest</i>
RL	Regressão Logística

Lista de símbolos

θ	Letra grega minúscula teta
σ	Letra grega minúscula sigma
μ	Letra grega minúscula mi

Sumário

1	INTRODUÇÃO	13
1.0.1	O Impacto das <i>fake news</i> no Brasil: Pandemia e Processo Eleitoral	14
1.0.2	Justificativa	15
1.0.3	Hipótese	16
1.0.4	Resultados esperados	18
1.0.5	Objetivo geral	18
1.0.6	Objetivos específicos	18
1.0.7	Organização do trabalho	19
2	REVISÃO BIBLIOGRÁFICA	20
3	FUNDAMENTAÇÃO TEÓRICA	25
3.1	Processamento de linguagem natural	25
3.2	Aprendizado de Máquina	26
3.3	Metodologia da Classificação	26
3.4	Base de dados	27
3.4.1	FAKE.BR	28
3.4.2	FakeRecogna	28
3.4.3	FakeTrueBR	29
3.4.4	BoatosBR	30
3.4.5	FullData	31
3.5	Pré-processamento	31
3.6	Modelos de Representação	34
3.6.1	<i>Bag of Words</i> - BOW	34
3.6.1.1	Exemplo Prático	34
3.6.2	TF-IDF	36
3.6.2.1	Exemplo Prático	38
3.6.3	Word2Vec	40
3.6.3.1	Exemplo Prático	41
3.7	Métodos de classificação	46
3.7.1	<i>Support Vector Classifier</i> (SVC)	47
3.7.2	Regressão Logística	50
3.7.3	<i>Multinomial Naive Bayes</i>	52
3.7.4	<i>Random Forest</i>	54
3.7.4.1	Exemplo Prático	54

3.8	Métricas de Avaliação	57
3.8.1	Acurácia	57
3.8.2	Precisão	57
3.8.3	Revocação	58
3.8.4	<i>F1-Score</i>	58
4	DESENVOLVIMENTO	59
4.1	Detalhes de Implementação	59
4.1.1	Problema da normalização	60
4.1.2	Convergência do método	60
4.1.3	Generalização da função	61
4.1.4	Divisão balanceada dos dados de treino e teste	62
4.1.5	Escolha do modelo Word2Vec: CBOW vs Skip-Gram	63
4.2	Testes	63
5	RESULTADOS	65
5.1	Resultados brutos	65
5.2	Análise em gráficos	67
5.2.1	Organização Visual dos Resultados	67
5.2.2	<i>Heatmap</i> por métrica	69
5.2.3	Radar de <i>F1-score</i> por base de dados	73
5.3	Análise dos Resultados	76
5.3.1	Gráficos de Radar	77
5.3.2	Gráficos de Calor	77
6	CONCLUSÃO	79
6.0.1	Desempenho do Word2Vec	80
7	TRABALHOS FUTUROS	82
	REFERÊNCIAS	85
	Appendices	89
	APÊNDICE A – ARQUIVOS DE RESULTADOS EM OBJETOS	
	JSON	90
A.1	Resultados para o BOW - Arquivo 1	90
A.2	Resultados para o TFIDF - Arquivo 2	93
A.3	Resultados para o Word2Vec - Arquivo 3	96

	APÊNDICE B – CÓDIGO FONTE DOS EXPERIMENTOS	100
B.1	Definição das bibliotecas utilizadas	100
B.2	Leitura dos repositórios	100
B.3	Método de pré processamento	101
B.4	Leitura da base FakeBR de seu diretório	102
B.5	Leitura da base FakeRecogna de seu diretório	102
B.6	Leitura da base FakeTrue de seu diretório	103
B.7	Leitura da base BoatosBR de seu diretório	104
B.8	Método BOW	104
B.9	Método TF-IDF	105
B.10	Método Word2Vec	105
B.11	Método SVC	106
B.12	Método Multinomial Naive Bayes	107
B.13	Método Random Forest	108
B.14	Método Regressão Logística	109
B.15	Leitura de todas as bases	109
B.16	Metodos auxiliares para execução dos métodos de representação e classificação	110
B.17	Criação das bases compostas	111
B.18	Execução dos testes para o método de representação BOW	112
B.19	Execução dos testes para o método de representação TF-IDF	113
B.20	Execução dos testes para o método de representação Word2Vec	114
B.21	Geração dos gráficos heatmap	115
B.22	Geração dos gráficos radar	116

1 Introdução

A disseminação de notícias falsas - as chamadas *fake news* - tem se tornado um dos principais desafios na era da informação digital. Por meio de redes sociais, aplicativos de mensagens, *blogs* e plataformas de conteúdo aberto, informações enganosas são frequentemente compartilhadas em larga escala, impactando negativamente a opinião pública, decisões políticas e até mesmo questões de saúde pública. Diante desse cenário, torna-se essencial o desenvolvimento de ferramentas computacionais que sejam capazes de identificar e mitigar a propagação de desinformação de maneira eficiente e automatizada.

Nesse contexto, o Processamento de Linguagem Natural (PLN) e o Aprendizado de Máquina (AM) têm se mostrado abordagens promissoras para a detecção de notícias falsas. Diversas técnicas de representação textual, como Bag-of-Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF) e Word to Vector (Word2Vec), possibilitam a conversão de textos em formatos estruturados que podem ser processados por algoritmos de classificação. Além disso, métodos como Support Vector Classifier (SVC), Naive Bayes (NB), Regressão Logística (RL) e Random Forest (RF) — além de vários outros — são amplamente utilizados na literatura para a categorização de textos binários, como é o caso da distinção entre notícias verdadeiras e falsas (JURAFSKY; MARTIN, 2019; SEBASTIANI, 2002).

Este trabalho propõe uma abordagem comparativa, na qual diferentes técnicas de representação textual e algoritmos de classificação são aplicados à tarefa de detecção de *fake news*. Para isso, são utilizados quatro conjuntos de dados distintos, com o objetivo de investigar a influência de diferentes domínios textuais e temáticos no desempenho dos modelos. Um diferencial metodológico deste estudo é a análise do comportamento dos modelos sob diferentes volumes e variedades de dados, por meio da combinação incremental dos corpora.

O termo *corpus* (plural: **corpora**) refere-se a um conjunto estruturado de textos utilizados para treinar, avaliar ou testar modelos de processamento de linguagem natural. No contexto deste trabalho, os corpora representam coleções de notícias (falsas e verdadeiras) oriundas de fontes diversas. O procedimento adotado consiste em iniciar os experimentos com um único corpus e, progressivamente, agregar os demais de forma acumulativa (ex.: *corpus A*; *corpus A + corpus B*; *corpus A + corpus B + corpus C*; *corpus A + corpus B + corpus C + corpus D*). Tal abordagem permite avaliar o impacto da escalabilidade, da heterogeneidade textual e da variação temática sobre a capacidade de generalização dos modelos, fornecendo evidências mais robustas sobre sua eficácia em contextos mais amplos e realistas.

Dessa forma, espera-se contribuir com uma análise sistemática e aprofundada sobre as relações entre a representação textual, os algoritmos de classificação e o volume de dados, oferecendo subsídios para o aprimoramento de sistemas automatizados de combate à desinformação.

1.0.1 O Impacto das *fake news* no Brasil: Pandemia e Processo Eleitoral

A desinformação tem se mostrado um dos maiores desafios da sociedade contemporânea, especialmente em contextos de crise como a pandemia de COVID-19 em 2020 e os processos eleitorais. No Brasil, o fenômeno das *fake news* atingiu proporções alarmantes, contribuindo para o agravamento de crises sanitárias e para a desestabilização do debate público. Dessa forma, vale analisar com mais profundidade como essa problemática se manifestou nos últimos anos e quais foram suas consequências, com base em evidências jornalísticas e acadêmicas.

Durante a pandemia, a circulação de informações falsas se espalhou de forma intensa nas redes sociais, aplicativos de mensagens e até mesmo em pronunciamentos públicos. Essa propagação teve impacto direto na forma como a população respondeu às medidas de prevenção e à vacinação. Segundo a reportagem [Senado \(2021\)](#) do Senado Federal, a disseminação de desinformação foi apontada por especialistas como um dos principais entraves no combate eficaz à COVID-19. Muitas pessoas passaram a desconfiar da eficácia das vacinas e a aderir a tratamentos sem comprovação científica, como a hidroxiclороquina, o que desviou o foco das autoridades de saúde e retardou a adoção de medidas realmente eficazes.

A [Brasil \(2021\)](#) destaca que o Twitter anunciou uma política de suspensão de usuários que divulgassem de forma recorrente informações falsas sobre a pandemia. Isso mostra como as redes sociais foram tanto veículos de propagação quanto tentaram se tornar agentes de contenção da desinformação.

A influência das *fake news* também foi marcante nas eleições presidenciais brasileiras, especialmente em 2018 e 2022. O processo eleitoral passou a conviver com campanhas de desinformação que usaram estratégias sofisticadas para manipular a opinião pública. No artigo de [Oliveira e Vieira \(2022\)](#), as autoras analisam como a proliferação de informações falsas durante a campanha de 2018 moldou percepções e influenciou diretamente o comportamento do eleitorado. A velocidade com que esses conteúdos circulam e a baixa alfabetização midiática da população brasileira agravaram o impacto das *fake news*.

Já o estudo de [Nascimento e Silva \(2023\)](#) investiga como os algoritmos de redes sociais amplificam o alcance de conteúdos falsos ao privilegiar o engajamento. Os autores apontam que os *trending topics* das eleições de 2022 muitas vezes foram dominados por desinformações, favorecendo discursos polarizadores e dificultando o debate racional.

Nesse sentido, a reportagem da [Brasil \(2020\)](#) alertava que o Brasil enfrentaria em 2020 os mesmos desafios observados em 2018, ressaltando a ausência de políticas públicas eficazes para combater a desinformação eleitoral.

O impacto das *fake news* vai além de eventos isolados. Entre os efeitos mais relevantes observam-se as consequências no âmbito social, político e cultural:

- **Desconfiança nas instituições:** A propagação de narrativas falsas mina a credibilidade de instituições democráticas e do sistema de saúde;
- **Polarização e radicalização política:** A desinformação alimenta discursos de ódio e extremismos, tornando o diálogo social mais hostil;
- **Desinformação em saúde pública:** As *fake news* dificultam a adesão às vacinas e outras medidas de proteção;
- **Erosão da democracia:** Ao manipular o eleitorado com informações falsas, compromete-se a integridade do voto e da representação política.

1.0.2 Justificativa

Conforme amplamente observado, a propagação de notícias falsas representa um fenômeno crescente e preocupante, com impactos diretos na sociedade, na política e na saúde pública. Em um cenário onde a informação é consumida rapidamente e compartilhada sem verificação, a capacidade de distinguir conteúdos verdadeiros de conteúdos manipulados torna-se essencial para a manutenção de uma sociedade bem-informada e crítica. Embora diversas iniciativas de checagem manual de fatos estejam em operação, o crescimento exponencial do volume de informações torna esse processo humanamente inviável em larga escala.

Embora haja um número expressivo de estudos sobre detecção e classificação de *fake news* em língua inglesa, observa-se uma lacuna significativa quando se trata da língua portuguesa. Segundo [Santos e Silva \(2023\)](#), a escassez de pesquisas específicas para o português compromete o desempenho de classificadores de notícias falsas, evidenciando a necessidade de desenvolver técnicas adaptadas às particularidades linguísticas do idioma. Além disso, [Monteiro et al. \(2020\)](#) destacam que a maioria das abordagens existentes não considera as nuances culturais e linguísticas do português, resultando em modelos menos precisos.

A maioria das bases de dados, modelos linguísticos e *benchmarks* disponíveis atualmente são voltados para o inglês, o que limita a aplicabilidade direta desses recursos em contextos socioculturais e linguísticos distintos, como o brasileiro. Considerando a diversidade semântica, morfosintática e pragmática do português, torna-se fundamental o

desenvolvimento de mais pesquisas voltadas para esse idioma, com a criação de corpora específicos, ajustes em técnicas de pré-processamento e validação de modelos de classificação textual. Investir em estudos voltados à realidade linguística do Brasil é essencial para garantir maior precisão na identificação de desinformação, bem como para fomentar soluções tecnológicas mais eficazes e culturalmente relevantes.

Nesse contexto, o uso de técnicas automatizadas de detecção de *fake news*, baseadas em Processamento de Linguagem Natural (PLN) e Aprendizado de Máquina, surge como uma solução promissora. No entanto, o desempenho desses modelos depende fortemente da forma como os textos são representados e das características dos algoritmos utilizados. Além disso, a escalabilidade e a generalização dos modelos frente a diferentes volumes e fontes de dados ainda são desafios relevantes.

Este trabalho justifica-se pela necessidade de investigar, de forma sistemática, a influência das técnicas de representação textual e dos classificadores na tarefa de detecção de notícias falsas, especialmente em contextos com diferentes tamanhos de conjunto de dados. A análise incremental, que avalia o desempenho dos modelos conforme o crescimento do volume de dados, representa uma contribuição importante para entender a robustez e a eficácia das abordagens propostas em cenários mais realistas e dinâmicos.

Além disso, ao empregar quatro bases de dados distintas, este estudo possibilita uma análise mais abrangente e comparativa dos métodos de representação e classificação aplicados. A abordagem proposta pode servir como base para futuras pesquisas e aplicações práticas voltadas ao combate à desinformação, especialmente diante da natureza dinâmica e crescente das informações disponíveis na web. Enquanto a maioria dos *corpus* existentes é estática, este trabalho reforça a importância de estratégias que viabilizem a expansão automatizada de conjuntos de dados, tornando o processo de criação e atualização de novos *corpus* mais eficiente e condizente com a realidade atual de produção contínua de conteúdo.

Isso permite que os modelos previamente treinados sejam continuamente alimentados com novos dados, aprimorando sua capacidade de classificação ao longo do tempo. Considerando que novas formas de *fake news* surgem constantemente, essa atualização contínua torna os modelos mais adaptáveis e menos restritos a padrões específicos, ampliando sua eficácia diante de diferentes estilos e estratégias de desinformação.

1.0.3 Hipótese

Modelos de aprendizado de máquina, especialmente aqueles aplicados à classificação de textos, tendem a apresentar melhor desempenho à medida que a quantidade de dados de treinamento aumenta. Essa hipótese parte do princípio de que, com mais exemplos, o modelo tem maiores chances de aprender padrões linguísticos relevantes e reduzir

erros de generalização, como falsos positivos e falsos negativos.

Diversos estudos na área de Processamento de Linguagem Natural (PLN) indicam que o aumento na base de treinamento pode trazer benefícios significativos, tais como:

1. **Aprimoramento da acurácia e de outras métricas de desempenho:** Com uma base de dados mais ampla e representativa, os modelos conseguem identificar com maior precisão os padrões linguísticos que diferenciam uma notícia falsa de uma verdadeira. [Hestness et al. \(2017\)](#) mostraram que, à medida que o número de dados de treinamento aumenta, o erro dos modelos de aprendizado profundo diminui de forma consistente, seguindo um padrão previsível, indicando melhorias consistentes com o aumento de dados;
2. **Redução do *overfitting*:** Modelos treinados com poucos dados tendem a memorizar exemplos específicos, o que compromete sua capacidade de generalização. O aumento do volume de dados contribui para uma aprendizagem mais robusta e eficaz. [Viering e Loog \(2021\)](#) observaram a forma das curvas de aprendizado, destacando que, embora muitas sigam uma lei de potência ou exponencial, há casos em que mais dados não levam necessariamente a melhor desempenho;
3. **Maior diversidade linguística e temática:** Com mais dados, cresce também a variedade de formatos, estilos de escrita e contextos textuais, o que favorece a construção de modelos mais generalistas e preparados para lidar com dados do mundo real. [Kaplan et al. \(2020\)](#) estudaram o desempenho de modelos de linguagem, observando que o desempenho melhora com o aumento do tamanho do modelo, do conjunto de dados e da quantidade de computação usada para o treinamento.

No entanto, é importante reconhecer que o ganho de desempenho não cresce indefinidamente com a adição de dados. A literatura aponta a existência de um ponto de saturação, a partir do qual os ganhos em acurácia tornam-se marginais. Além disso, a qualidade dos dados é um fator crítico: um conjunto de dados ruidoso, desbalanceado ou mal rotulado pode comprometer a eficácia do modelo, mesmo em grandes volumes ([HESTNESS et al., 2017](#)).

Segundo [Goodfellow, Bengio e Courville \(2016\)](#) modelos mais simples, como a Regressão Logística, podem não ser capazes de explorar todo o potencial de conjuntos de dados muito grandes, enquanto modelos mais sofisticados (como redes neurais profundas) podem se beneficiar mais desse crescimento — embora a um custo computacional maior. Portanto, a capacidade dos algoritmos utilizados é outro aspecto a ser considerado.

Sendo assim, a hipótese central deste trabalho é:

“O aumento progressivo do volume de dados de entrada resulta em uma melhoria no desempenho dos modelos de classificação de notícias falsas, até certo ponto de saturação, onde os ganhos se tornam reduzidos.”

Essa hipótese será testada por meio de experimentos que utilizam quatro corpora distintos, os quais serão combinados de forma incremental, permitindo avaliar o impacto direto do crescimento da base de dados sobre o desempenho dos modelos sob diferentes técnicas de representação textual e algoritmos de classificação.

1.0.4 Resultados esperados

Aumentar o número de notícias na base de dados de treinamento tende a melhorar a acurácia e a capacidade de generalização, desde que os dados sejam representativos e bem rotulados.

1.0.5 Objetivo geral

Avaliar o desempenho de diferentes técnicas de representação textual e algoritmos de classificação na detecção de notícias falsas, considerando a variação dos resultados à medida que o volume de dados aumenta com a inclusão progressiva de múltiplos *corpus* na base de dados de treinamento, contribuindo para o avanço de soluções na detecção automática de desinformação.

1.0.6 Objetivos específicos

1. Pré-processar e padronizar os quatro conjuntos de dados selecionados, garantindo a consistência dos textos e dos rótulos para aplicação dos métodos de análise;
2. Aplicar três técnicas de representação de texto — *Bag-of-Words* (BoW), TF-IDF e Word2Vec — sobre os *corpus*, transformando os dados textuais em vetores numéricos compatíveis com algoritmos de aprendizado de máquina;
3. Treinar e avaliar quatro algoritmos de classificação supervisionada — *Support Vector Classifier* (SVC), Naive Bayes (NB), Regressão Logística e *Random Forest* — utilizando as diferentes representações vetoriais;
4. Realizar experimentos de forma incremental, combinando as bases de dados gradualmente para analisar o impacto do aumento do volume de dados sobre o desempenho dos modelos;
5. Comparar os resultados obtidos com base em métricas de desempenho como acurácia, precisão, revocação e F1-score, identificando quais combinações de técnicas oferecem melhores resultados em diferentes cenários;

6. Analisar a escalabilidade e a robustez dos modelos frente à variação no tamanho e na diversidade do conjunto de dados.

1.0.7 Organização do trabalho

Este trabalho está organizado em sete capítulos, conforme descrito a seguir:

Capítulo 2 – Revisão Bibliográfica: reúne estudos e trabalhos relacionados à detecção de *fake news* por meio de técnicas de aprendizado de máquina e Processamento de Linguagem Natural, com o objetivo de contextualizar a pesquisa e identificar abordagens semelhantes na literatura. Foram realizadas pesquisas com foco em artigos que envolvem principalmente os métodos utilizados nesse trabalho;

Capítulo 3 – Fundamentação Teórica: apresenta os principais conceitos e técnicas utilizadas ao longo da pesquisa. São descritas as bases de dados adotadas, os processos de pré-processamento aplicados, os três modelos de representação textual empregados (*Bag-of-Words*, TF-IDF e Word2Vec), bem como os métodos de classificação testados (SVC, Naive Bayes e Regressão Logística). Em cada um dos tópicos, foi demonstrado um exemplo simples e didático para melhor entendimento do funcionamento de cada método;

Capítulo 4 – Desenvolvimento: Apresenta quais tecnologias foram adotadas para execução do trabalho. Além disso, esse capítulo é dividido em duas subseções: Detalhes de implementação, que trata dos problemas enfrentados durante a construção do código-fonte e observações importantes; Testes, que detalha quais e como foram realizados os experimentos; e Resultados, que apresenta as métricas obtidas, acompanhadas de gráficos e análises dos desempenhos dos modelos;

Capítulo 5 – Resultados: Apresenta os resultados adquiridos na execução dos testes, demonstrando os resultados brutos (todos os resultados) em forma de tabela, os gráficos gerados a partir destes resultados e a interpretação desses resultados correlacionando com a hipótese proposta;

Capítulo 6 - Conclusão: apresenta as considerações finais da pesquisa, destacando as principais observações e aprendizados obtidos a partir dos experimentos;

Capítulo 7 – Trabalhos Futuros: sugere possíveis melhorias, desdobramentos e continuação da pesquisa, considerando limitações enfrentadas e novas oportunidades de investigação na área de detecção automática de desinformação.

2 Revisão Bibliográfica

O trabalho de [Monteiro et al. \(2018\)](#) apresenta a criação do Fake.Br, o primeiro *corpus* de referência para detecção de notícias falsas em português brasileiro — uma contribuição inédita até então, conforme destacado pelos autores. Foram coletadas e rotuladas manualmente 7.200 notícias — sendo 3.600 falsas e 3.600 verdadeiras — escritas em texto puro e organizadas por categorias como política, celebridades, sociedade, ciência, economia e religião. As notícias falsas foram extraídas de quatro sites não confiáveis, enquanto as verdadeiras foram obtidas semiautomaticamente de grandes portais (G1, Folha, Estadão) com base em palavras-chave extraídas dos títulos das falsas, seguidas por uma etapa de verificação de similaridade e validação manual. O *corpus* também inclui metadados como autor, data e links, e busca manter homogeneidade de tamanho entre os textos. Foram extraídas métricas linguísticas como pausidade, emotividade, incerteza e não-imediaticidade, e conduzidos experimentos com técnicas de *machine learning* (como SVM, Naive Bayes e MLP), utilizando diferentes representações textuais (como BoW, POS tags e léxicos semânticos). Os melhores resultados foram alcançados com o uso de *bag of words* simples, atingindo até 90% de acurácia com MLP e 88% de F1 com SVM. O trabalho de [Monteiro et al. \(2018\)](#) também destaca que a filtragem de "meias-verdades" pode ter contribuído para o bom desempenho, mas limita a complexidade do *corpus* — um ponto a ser explorado em trabalhos futuros.

[Garcia, Afonso e Papa \(2022\)](#) propôs um novo *corpus* para detecção de *fake news* em português brasileiro, chamado FakeRecogna, que se destaca por conter notícias atualizadas (de 2019 a 2021), uma maior variedade de categorias, maior volume de textos, temas contemporâneos como a pandemia de Covid-19 e fontes reconhecidas internacionalmente. Foram realizados experimentos utilizando duas técnicas de representação textual (*Bag of Words* e *FastText*) e seis classificadores, com destaque para o uso de redes neurais convolucionais (CNN), que alcançaram acurácia superior a 94%. Os melhores resultados foram obtidos com *FastText* e *Multilayer Perceptron*, enquanto a remoção da lematização prejudicou o desempenho do BoW. Também foi avaliado o impacto da ampliação dos dados ao unir o FakeRecogna a outros corpora (Fake.Br e FACTCK.BR), resultando em melhorias na acurácia em alguns casos. O estudo conclui que o *corpus* FakeRecogna é promissor para fomentar pesquisas na área, especialmente pela escassez de dados em português, e propõe como trabalhos futuros a expansão da base e o uso de modelos mais avançados, como BERT e mecanismos de atenção.

[Chavarro et al. \(2023\)](#) apresenta a criação do FakeTrueBR, um novo *corpus* de notícias falsas e verdadeiras em português, desenvolvido com o objetivo de melhorar a

detecção de *fake news* por meio de um conjunto de dados bem alinhado, atualizado e limpo. Para isso, os autores coletaram 4.600 mensagens falsas do portal Boatos.org, selecionando 2.083 com mais de 300 caracteres e removendo as justificativas que explicam a falsidade, a fim de preservar a integridade da desinformação como aparece nas redes. Em seguida, utilizaram palavras-chave dos títulos dessas mensagens para buscar notícias verdadeiras correlatas em fontes confiáveis como G1 e Folha, aplicando *embeddings* do Sentence-BERT multilíngue para medir a similaridade semântica e garantir a correspondência temática entre os pares falso-verdadeiro. O resultado foi um *corpus* balanceado com 1.791 pares de notícias (falsas e verdadeiras), organizadas em cinco categorias e enriquecidas com metadados como data, link e autor. Nos testes realizados com representações BoW-TF e BoW-TF-IDF e modelos clássicos de *machine learning*, o *corpus* permitiu alcançar uma F1-score de até 0,945, destacando-se pela qualidade do alinhamento semântico e pela utilidade para futuras pesquisas e aplicações em PLN, embora os autores reconheçam limitações como o viés no comprimento dos textos e a necessidade de validação manual das correspondências.

O trabalho de CANTARINO (2024) teve como objetivo desenvolver uma base de dados atualizada em português brasileiro, chamada BoatosBr *corpus*, para auxiliar na detecção automática desse tipo de conteúdo. A coleta das notícias foi realizada por meio de *web crawlers* desenvolvidos em Python, com base em textos previamente verificados por sites de checagem, especialmente o Boatos.org. O *corpus* resultante contém mais de 3.000 textos rotulados como falsos ou verdadeiros. Após a coleta, foram aplicadas técnicas iniciais de Processamento de Linguagem Natural (PLN) e algoritmos clássicos de Aprendizado de Máquina (AM) — que alcançaram acurácia superior a 85% — para validar a viabilidade do *corpus*. Além disso, foram extraídas informações complementares dos textos, como número de *emojis*, verbos e palavras em maiúsculas, visando enriquecer futuros estudos. Apesar dos bons resultados, o *corpus* apresenta desequilíbrio entre classes e variação no tamanho dos textos, aspectos que devem ser aprimorados em pesquisas futuras.

Giordani et al. (2023) avaliou o impacto de diferentes técnicas de representação textual, como TF-IDF e Word2Vec, e de diversos modelos de aprendizado de máquina — incluindo Adaboost, Regressão Logística, *Random Forest*, SVM (com kernel RBF) e LightGBM — na detecção de *fake news* utilizando o *corpus* Fake.Br como base de dados. Foram testadas múltiplas configurações de hiperparâmetros por meio de validação cruzada com 5 folds, visando otimizar o desempenho dos modelos, mesmo com a limitação do número de amostras (7.200). Um dos principais achados foi que o treinamento do Word2Vec com vocabulário específico do *corpus* Fake.Br elevou a acurácia na classificação das notícias falsas. Além disso, foi desenvolvida uma plataforma *web* que permite a aplicação desses modelos para classificar novos textos e comparar seu desempenho, destacando

a viabilidade prática da solução. Os resultados indicaram que modelos como Regressão Logística oferecem maior interpretabilidade, enquanto modelos mais complexos, como LightGBM, obtêm desempenho superior, ainda que com menor transparência. A plataforma é voltada exclusivamente à verificação de autenticidade de textos jornalísticos e não se destina a frases soltas, perguntas ou conteúdos típicos de redes sociais.

O trabalho de DIAS (2019) propõe um novo conjunto de dados rotulado para detecção de notícias falsas em português, utilizando como fonte verificadores de fatos imparciais reconhecidos internacionalmente, como os vinculados à IFCN. Diferente de iniciativas anteriores, o *corpus* não foi rotulado manualmente pelos autores. O estudo avaliou diversos modelos de classificação, empregando diferentes algoritmos (como Naive Bayes e SVM) e representações textuais, incluindo *Bag of Words*, POS tags e Entidades Nomeadas — sendo uma das primeiras abordagens a utilizar Entidades Nomeadas como característica para classificação de *fake news*. Os resultados indicaram que essas entidades são promissoras na identificação de sinais de engano, especialmente quando combinadas com outras técnicas.

No estudo de Fischer et al. (2022) os autores avaliaram diferentes modelos de aprendizado de máquina e aprendizado profundo para a detecção de *fake news* em português brasileiro, utilizando o *corpus* Fake.Br. Inicialmente, foram testados modelos tradicionais com representações TF e TF-IDF, comparando textos completos e truncados, o que revelou que o comprimento dos textos influencia diretamente na acurácia. O melhor desempenho com textos completos foi obtido pelo modelo empilhado (*stacking*) com LinearSVC como meta-classificador, atingindo um F1-score de 97,49%. Modelos baseados em XGBoost apresentaram desempenho inferior, em parte devido à dependência de ajustes de hiperparâmetros. Em seguida, testaram redes neurais profundas com vetores de *embeddings* e observaram que a remoção de *stopwords* melhorava os resultados. Por fim, o modelo BERT multilingue superou todos os anteriores, alcançando F1-score de 98,40%, sendo estabelecido como o novo *benchmark* para a tarefa de detecção de notícias falsas em português. Os autores destacam o potencial da combinação de técnicas e o uso de modelos robustos como o BERT para futuras melhorias.

Moreira et al. (2023) avaliou a eficácia de diferentes algoritmos de aprendizado de máquina na detecção de *fake news* em português, utilizando o *corpus* Fake.Br como base de dados e aplicando as técnicas de vetorização *Bag of Words* e TF-IDF, além do modelo pré-treinado BERTimbau. Foram utilizados sete algoritmos, incluindo Naive Bayes, Regressão Logística, SVM e BERTimbau, com validação cruzada (10 folds) para garantir a robustez dos resultados. O BERTimbau se destacou com precisão média de 98,86%, superando todos os algoritmos tradicionais, inclusive sua versão original em inglês, devido à sua adaptação ao português e à capacidade de capturar contexto sem necessidade de reentrenamento. A superioridade estatística do BERTimbau foi comprovada por meio do

teste de Mann-Whitney U. Apesar dos resultados promissores, o estudo destaca limitações como o viés aprendido pelo modelo, custo computacional elevado e menor desempenho com palavras raras. Como contribuição adicional, foi desenvolvido um relatório completo de desempenho e apontadas futuras aplicações como o monitoramento de *fake news* sobre vacinas, em parceria com a Avico Brasil, visando mapear redes de colaboração na disseminação de desinformação.

Tigre (2023) propôs a criação de uma base de dados em português com notícias verdadeiras e falsas relacionadas à COVID-19, com o objetivo de treinar e comparar diferentes modelos de *Machine Learning* na tarefa de classificação automática de notícias falsas. Motivado pela rápida disseminação de desinformação durante a pandemia e pela escassez de *datasets* em língua portuguesa, o autor avaliou diversos algoritmos, como *Random Forest*, que apresentou o melhor desempenho geral entre os cenários testados. Além disso, foi observada uma melhora nos resultados quando técnicas de vetorização foram combinadas com métricas textuais. O trabalho contribui com uma ferramenta prática capaz de classificar automaticamente textos como reais ou falsos, além de explorar diferentes configurações de modelos e suas performances em contextos específicos.

Saavedra (2023) conduziu um estudo que investigou métodos de representação textual e modelos de aprendizado de máquina aplicados à classificação automática do tema de notícias em português. O trabalho desenvolveu um coletor personalizado que reuniu mais de 18 mil notícias atualizadas de portais brasileiros, rotuladas manualmente em 19 categorias temáticas. Foram testadas diversas técnicas de pré-processamento e representação textual, como Word2Vec, Doc2Vec, *Bag of Words* com TF-IDF e ajuste fino de um modelo BERT pré-treinado. Para o treinamento e avaliação, foram utilizados algoritmos como regressão logística e máquinas de vetor de suporte (SVM). O modelo BERT obteve os melhores resultados, com 93% de acurácia e F1-score ponderado de 0,98, superando significativamente as abordagens tradicionais. Ainda assim, combinações como *Bag of Words* com SVM e Doc2Vec com regressão logística também alcançaram resultados satisfatórios, apresentando-se como opções viáveis com menor custo computacional. Os achados oferecem contribuições relevantes para o desenvolvimento de sistemas eficazes de categorização automática de notícias em língua portuguesa.

Pires e Silva (2024) investiga a eficácia de modelos BERT, mBERT e BERTimbau na tarefa de detecção de notícias falsas em português, utilizando os corpora Fakepedia, FakeRecogna, FakeTrueBR e Fake.Br, separadamente. Os resultados mostram que o BERTimbau, pré-treinado especificamente para a língua portuguesa, supera tanto o BERT (em inglês) quanto o mBERT (multilíngue), apresentando ganhos médios de até 2,8% em acurácia, 2,64% em *recall* e 2,37% em *F1-score*. Esses resultados indicam que o uso de um modelo ajustado ao idioma-alvo proporciona classificações mais confiáveis e equilibradas. O estudo sugere ainda caminhos futuros, como testes com bases fora da distribuição,

análises estatísticas e inclusão de metadados e atributos linguísticos.

O estudo de Bastos et al. (2024) investiga o uso de algoritmos de aprendizado de máquina na detecção de *fake news*, considerando o impacto negativo que a disseminação de informações falsas pode gerar em áreas como política, educação e saúde. A pesquisa destaca o papel do aprendizado de máquina na verificação automatizada de notícias, especialmente diante do grande volume de conteúdo compartilhado em redes sociais. Foram aplicadas técnicas como Holdout e modelos de classificação (*Decision Tree*, SVM e KNN) para prever a veracidade de notícias, com destaque para o desempenho do SVM, que obteve a maior acurácia, e da *Decision Tree*, que apresentou bom equilíbrio entre sensibilidade e especificidade. Embora o modelo KNN tenha mostrado alta sensibilidade e precisão, sua baixa especificidade reduziu sua eficácia geral. Os resultados reforçam a importância de selecionar o algoritmo mais adequado às características do conjunto de dados e ao tipo de erro mais crítico para o contexto. O estudo recomenda o uso futuro de abordagens em conjunto (*ensemble*), ampliação de bases em português e aplicação em sistemas *web*, a fim de mitigar a propagação de *fake news* na sociedade.

3 Fundamentação Teórica

Este capítulo apresenta os principais conceitos teóricos que embasam o desenvolvimento deste trabalho. Primeiramente, é abordado o **Processamento de Linguagem Natural (PLN)**, área responsável por permitir que computadores compreendam e manipulem dados textuais em linguagem humana. Em seguida, discute-se o **Aprendizado de Máquina**, com ênfase nos métodos supervisionados voltados para tarefas de classificação. A seção sobre **Metodologia da Classificação** detalha o fluxo geral de construção de modelos, desde a preparação dos dados até a avaliação dos resultados. A **Base de Dados** utilizada nos experimentos é apresentada com suas principais características e particularidades. Na sequência, o processo de **Pré-processamento** textual é descrito, destacando as etapas fundamentais para a padronização e limpeza dos dados. Em seguida, são exploradas as técnicas de **Modelos de Representação**, como Bag-of-Words, TF-IDF e Word2Vec, que convertem textos em formatos numéricos compatíveis com algoritmos de aprendizado. A seção de **Métodos de Classificação** apresenta os principais algoritmos utilizados, como Regressão Logística, SVC, Naive Bayes e Random Forest. Por fim, são discutidas as **Métricas de Avaliação** adotadas na análise dos resultados, como acurácia, precisão, revocação e F1-score, fundamentais para medir o desempenho dos modelos na tarefa de detecção de notícias falsas.

3.1 Processamento de linguagem natural

O Processamento de Linguagem Natural (PLN) é uma subárea da inteligência artificial e da linguística computacional que estuda técnicas para que máquinas possam compreender, interpretar, gerar e manipular a linguagem humana em diferentes formatos (CAMBRIA; WHITE, 2014). Entre as aplicações mais comuns do PLN estão a classificação de textos, extração de informações, tradução automática, análise de sentimentos e detecção de entidades nomeadas (JURAFSKY; MARTIN, 2019).

No contexto deste trabalho, o PLN tem papel central, uma vez que a tarefa de detecção de notícias falsas envolve a análise de padrões linguísticos, semânticos e contextuais presentes nos textos. Para que um modelo de aprendizado de máquina consiga distinguir entre conteúdos verdadeiros e falsos, é necessário que ele seja capaz de reconhecer tais padrões de forma eficiente, utilizando representações vetoriais que traduzam as características linguísticas dos textos em dados computacionais processáveis (MANNING; SCHÜTZ, 1999).

Além disso, como o conteúdo analisado é textual, as etapas de pré-processamento,

vetorização e extração de características fazem parte essencial do fluxo de execução dos experimentos, sendo todas fundamentadas em técnicas e abordagens do PLN.

3.2 Aprendizado de Máquina

O Aprendizado de Máquina (AM), ou *Machine Learning*, é um ramo da inteligência artificial que se concentra no desenvolvimento de algoritmos capazes de aprender padrões a partir de dados e fazer previsões ou tomar decisões com base nesses aprendizados, sem serem explicitamente programados para isso (MITCHELL, 1997). Os algoritmos de AM são amplamente utilizados em tarefas de classificação, como detecção de fraudes, reconhecimento de imagens e, especialmente neste trabalho, na classificação de textos jornalísticos como verdadeiros ou falsos.

Em vez de depender de regras manuais ou codificações fixas, os modelos aprendem com exemplos. Para isso, eles são treinados em conjuntos de dados rotulados — isto é, nos quais se sabe previamente se uma notícia é falsa ou verdadeira — e, a partir dessas observações, constroem um modelo que busca generalizar essas informações para novos textos Bishop (2006).

No contexto da detecção de notícias falsas, o aprendizado de máquina permite automatizar a tarefa de identificação de padrões linguísticos, semânticos ou estilísticos associados a conteúdo enganoso. Modelos como a Regressão Logística, o Naive Bayes, Máquina de Vetores de Suporte (SVM) e *Random Forest* são utilizados nesta pesquisa por sua ampla aplicabilidade e eficiência em tarefas de classificação de texto (SEBASTIANI, 2002).

3.3 Metodologia da Classificação

O fluxo de execução de um modelo de classificação de textos segue uma sequência estruturada de etapas, fundamentais para garantir a eficácia do processo. Tudo começa com a **entrada dos dados brutos**, geralmente compostos por textos em linguagem natural. Esses dados passam por um processo de **pré-processamento**, que pode incluir a remoção de pontuação, *stopwords*, normalização de caixa (minúsculas), lematização ou *stemming*, com o objetivo de reduzir ruídos e padronizar o conteúdo textual. Em seguida, ocorre a **representação dos textos**, etapa em que o conteúdo textual é transformado em vetores numéricos que os algoritmos de classificação conseguem interpretar. Com os textos convertidos em vetores, esses dados vetorizados são então utilizados como entrada para um **algoritmo de classificação**, que pode ser supervisionado ou não supervisionado, dependendo da natureza do problema. O modelo é treinado com exemplos rotulados, aprendendo os padrões que distinguem as classes (por exemplo, “verdadeiro” e “falso”). Após

o treinamento, o modelo é capaz de classificar novos textos com base no que aprendeu, fornecendo como saída a **classe predita** e, em alguns casos, a **probabilidade associada a cada classe**. Esse fluxo é essencial para tarefas como a detecção de *fake news*, análise de sentimento, classificação de tópicos, entre outros desafios em Processamento de Linguagem Natural (MEDHAT; HASSAN; KORASHY, 2014). Todo esse fluxo é visualmente representado através da Figura 1.

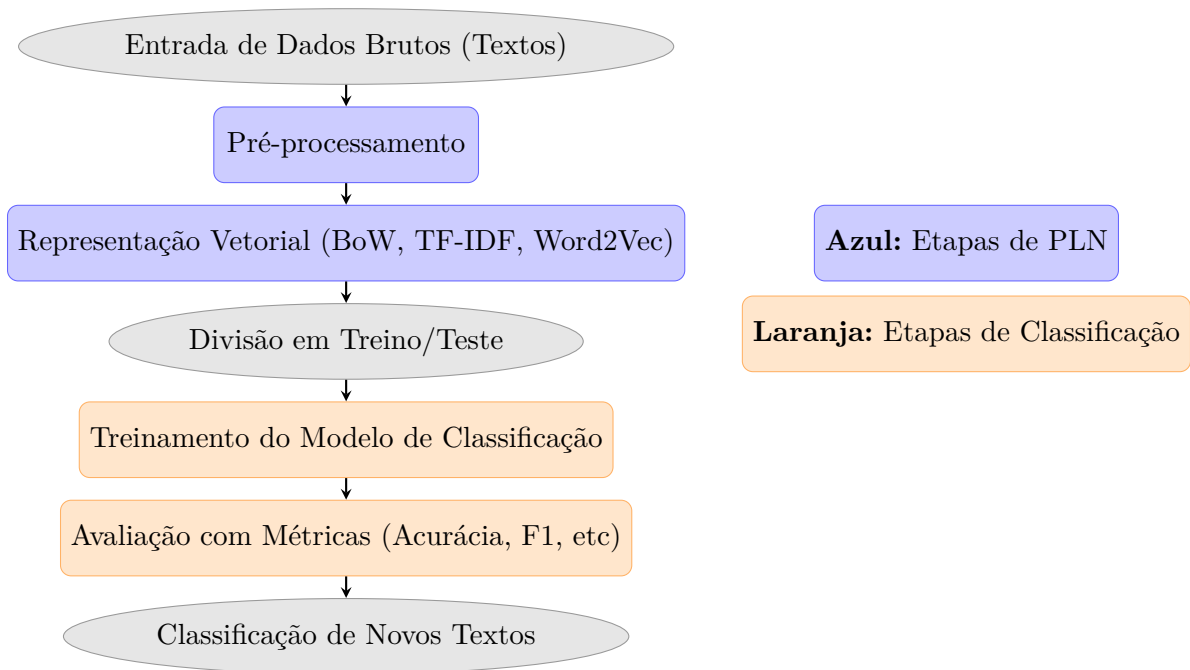


Figura 1 – Fluxograma do processo de classificação de textos, destacando as etapas de Processamento de Linguagem Natural (em azul) e Classificação (em laranja).

Fonte: Elaborado pela autora.

3.4 Base de dados

O presente trabalho tem como objetivo investigar a classificação de notícias falsas no contexto da língua portuguesa. Para isso, foi realizada uma pesquisa sobre os principais *corpus* em português amplamente utilizados em estudos científicos e acadêmicos, priorizando aqueles que apresentam respaldo metodológico, reconhecimento na literatura e que adotam procedimentos rigorosos de coleta, curadoria e validação das informações contidas. Além disso, também foram analisados *corpus* criados mais recentemente, com o intuito de identificar novas notícias, padrões de conteúdo e novos recursos disponíveis para a análise de desinformação na língua portuguesa.

3.4.1 FAKE.BR

O *corpus* FAKE.BR, criado em 2018, foi um dos primeiros conjuntos de dados em língua portuguesa voltados à detecção de notícias falsas. Conforme cita o autor, na época de sua publicação, não havia outra base de dados semelhante disponível em português, e por isso é um *corpus* comumente utilizado nos trabalhos publicados até agora no que se refere a classificação de notícias em língua portuguesa (MONTEIRO et al., 2018).

Essa base de dados contém 7.200 textos, igualmente divididos entre notícias verdadeiras e falsas - 3600 em cada categoria - abrangendo temas como política, televisão e celebridades. O repositório do *corpus* Fake.Br está disponível em <<https://github.com/roneysco/Fake.br-Corpus>>. Todos os textos estão em formato de texto simples, em arquivos .txt, em que cada unidade de texto se encontra em um arquivo diferente. No repositório, há duas pastas: /fake e /true, que contém em cada pasta seus respectivos textos: as 3.600 notícias verdadeiras estão no diretório /true e as 3.600 notícias falsas estão no diretório /fake.

As notícias falsas foram analisadas e coletadas manualmente de 4 sites: Diário do Brasil, A Folha do Brasil, The Journal Brasil e Top Five TV. Foi usado um *crawler* para coletar notícias das principais agências de notícias do Brasil: G1, Folha de São Paulo e Estadão. O rastreador procurou nas páginas da *web* correspondentes dessas agências por palavras-chave das notícias falsas. Cerca de 40.000 notícias verdadeiras foram coletadas desta forma.

Para cada notícia verdadeira, foi aplicada uma medida de similaridade lexical, escolhendo as mais semelhantes às notícias falsas, e por fim uma última verificação manual foi feita para garantir que as notícias falsas e verdadeiras estivessem relacionadas ao assunto. É interessante observar que houve casos em que as notícias verdadeiras negavam explicitamente as falsas correspondentes, mas outras eram meramente sobre o mesmo tópico.

3.4.2 FakeRecogna

O *corpus* FakeRecogna foi desenvolvido em 2022 e é composto por 11.902 textos jornalísticos, organizados em um único arquivo de extensão .xlsx e está disponível publicamente em um repositório no GitHub <<https://github.com/Gabriel-Lino-Garcia/FakeRecogna>>. O arquivo contém 8 colunas de metadados, onde cada linha representa uma amostra individual (seja uma notícia verdadeira ou falsa). Cada instância da base contém título, subtítulo e corpo da notícia onde cada uma dessas informações se encontra em uma coluna do arquivo (GARCIA; AFONSO; PAPA, 2022).

As notícias presentes no FakeRecogna abrangem uma variedade de temas, incluindo política, celebridades, ciência, saúde, entretenimento, notícias nacionais e internacionais.

Essas categorias foram definidas com base nas seções dos portais de onde os textos foram extraídos.

A coleta dos textos foi realizada por *crawlers* desenvolvidos especificamente para minerar páginas de agências de notícias brasileiras amplamente reconhecidas por sua relevância nacional. Esses *crawlers* foram ajustados de acordo com a estrutura de cada site analisado, categorizando as informações extraídas e agrupando-as por data.

A diversidade do *corpus* é favorecida pela pluralidade de fontes e estilos de escrita, o que o torna particularmente valioso para tarefas de Processamento de Linguagem Natural (PLN) e para a aplicação de algoritmos de aprendizado de máquina.

No caso das notícias falsas, a extração focou-se principalmente em sites identificados pelo *Duke Reporters' Lab*, que mantém uma lista mundial de agências de checagem de fatos. Em 2019, havia cerca de 160 agências ativas ao redor do mundo, e o Brasil se destacava como um ecossistema em expansão, com nove iniciativas identificadas. Destas, seis foram consideradas na construção do *corpus*, resultando em 5.951 amostras de notícias falsas, com variação significativa na quantidade extraída de cada fonte.

3.4.3 FakeTrueBR

O FakeTrueBR é um *corpus* brasileiro desenvolvido para auxiliar na detecção de notícias falsas em língua portuguesa. Criado por pesquisadores da Universidade Federal de Santa Catarina (UFSC) e da *London School of Economics and Political Science* (LSE), o *corpus* busca superar limitações de conjuntos de dados anteriores, como desatualização e desalinhamento entre notícias falsas e verdadeiras. O *corpus* está disponível publicamente no repositório GitHub: <<https://github.com/jpchav98/FakeTrue.Br>> (CHAVARRO et al., 2023).

A construção do *corpus* se deu pela criação de *crawler* em linguagem Python. As notícias falsas foram extraídas do site Boatos.org, conhecido por desmentir informações enganosas. Foram obtidas 4.600 mensagens falsas, das quais 2.083 atendiam ao requisito de ter mais de 300 caracteres.

Para a coleta de notícias verdadeiras, em cada uma das notícias falsas, foram extraídas palavras-chave dos títulos. Em seguida, um *crawler* semelhante foi adaptado para procurar nas fontes confiáveis G1 e Folha de S.Paulo notícias verdadeiras que contivessem essas palavras-chave. Esse processo resultou na recuperação de 3.032 notícias verdadeiras.

Após o processo de alinhamento e filtragem, o conjunto de dados final consiste em 3.582 notícias em português, publicadas entre 2017 e 2023, sendo 1.791 verdadeiras e 1.791 falsas. As notícias estão armazenadas em um arquivo de formato *.csv* e distribuídas em cinco categorias: Brasil, política, entretenimento, saúde e mundo. Além disso, foram

extraídos metadados como data de publicação, link e autor de cada notícia.

O *corpus* foi avaliado utilizando técnicas clássicas de processamento de linguagem natural para representação de textos, como *Bag of Words* (BoW) e TF-IDF, juntamente com vários métodos tradicionais de classificação. Os resultados demonstraram que o conjunto de dados é efetivo para a classificação de notícias, alcançando um *F1-score* de 0,945 usando um classificador de *Multi-layer Perceptron*.

3.4.4 BoatosBR

O BoatosBR é um *corpus* desenvolvido na Universidade Federal de Uberlândia (UFU) como parte de um Trabalho de Conclusão de Curso, com o objetivo de auxiliar na detecção de notícias falsas em língua portuguesa. O estudo visa enfrentar o desafio da rápida disseminação de informações falsas na internet, criando um conjunto de dados que possa ser utilizado para treinar modelos de aprendizado de máquina na identificação de *fake news* (CANTARINO, 2024).

Utilizou-se *web crawlers* desenvolvidos em Python para extrair textos de sites especializados em desmentir boatos, como o Boatos.org. Cada texto coletado foi classificado manualmente como “Verdadeiro” ou “Falso”, com base na veracidade da informação.

Para validar a eficácia do *corpus*, foram aplicadas técnicas de Processamento de Linguagem Natural (PLN) e algoritmos de classificação de Aprendizado de Máquina, incluindo Naive Bayes, *Support Vector Machine* (SVM) e *Random Forest*. Os resultados indicaram que o BoatosBR é um recurso viável para estudos e aplicações na detecção automatizada de notícias falsas.

Este novo *corpus* é composto no total por 3427 textos digitais, abrangendo boatos entre o período de 2022 a 2024. A proporção de boatos verdadeiros é maior do que a de textos falsos, sendo eles 1516 unidades de texto contendo notícias verdadeiras para 1911 unidades de texto com notícias falsas. Sendo assim, observasse que a base não está balanceada, assim como o tamanho dos textos, nos quais é possível encontrar diferença na quantidade de palavras em cada texto do *corpus*.

Esse *corpus* está armazenado em um único arquivo no formato `.json`, contendo todos os 3.427 textos, cada um representado como um objeto JSON dentro desse arquivo. Pode ser acessado publicamente através do GitHub em <https://github.com/Felipe-Harrison/boatos-br-protect/unhbox/voidb@x\bgroup\edef,{corpus}\let\futurelet\@let@token\let\itshapecorpus\egroup>. O conteúdo utilizado neste trabalho encontra-se no diretório `/base_simple`. Os textos digitais estão organizados em categorias conforme seus principais temas: política, sociedade, saúde, mundo, ciência e tecnologia, religião e economia.

A versão presente nesse diretório é a versão simples do *corpus*, com menos informações sobre cada texto. Em cada objeto foi encontrado dados tais como url, data-publicacao, origem, categorias, texto e rótulo - atributo que representa se o texto é 'verdade' ou 'falso'.

3.4.5 FullData

Apelidado de *FullData*, este *corpus* é composto pela união das quatro bases de dados mencionadas anteriormente: **Fake.br**, **FakeRecogna**, **FakeTrue** e **BoatosBR**.

Considerando que um dos objetivos deste trabalho é avaliar o comportamento dos classificadores à medida que aumenta a quantidade de dados de treinamento, o FullData é utilizado de forma incremental nos testes. Em alguns experimentos, é composto por apenas uma das bases; em outros, pela combinação de duas ou mais. A composição específica de cada conjunto de dados utilizado é explicitada nos resultados, de modo a possibilitar a compreensão do que está sendo empregado em cada experimento.

Base 1 - FullData = Fake.br

Base 2 - FullData = Fake.br + FakeRecogna

Base 3 - FullData = Fake.br + FakeRecogna + FakeTrue

Base 4 - FullData = Fake.br + FakeRecogna + FakeTrue + BoatosBR

Conforme mencionado, as quatro bases apresentam formatos diversos: algumas estão em *.txt*, outras em *.xlsx* ou *.json*. Por esse motivo, durante a etapa de leitura e pré-processamento, cada *corpus* foi lido em seu respectivo formato e posteriormente padronizado para compor um único objeto *DataFrame* da biblioteca pandas, possibilitando sua unificação.

Além disso, para equilibrar a proporção entre notícias verdadeiras e falsas na base **BoatosBR**, foi realizada uma amostragem aleatória com redução no número de amostras da classe com maior número de textos. Assim, o número de notícias falsas foi reduzido de 1.911 para 1.516, igualando-se à quantidade de notícias verdadeiras da mesma base.

3.5 Pré-processamento

O pré-processamento textual é uma etapa fundamental na área de Processamento de Linguagem Natural (PLN), especialmente em tarefas de classificação de textos, como a detecção de notícias falsas. O objetivo central dessa fase é transformar os textos brutos em uma forma estruturada e padronizada, que possa ser processada eficientemente por algoritmos de aprendizado de máquina. Dada a natureza não estruturada dos textos, especialmente aqueles provenientes da internet, esse processo contribui para a melhoria da

Corpus	Verdadeiras	Falsas	Total
FakeBR	3600	3600	7200
FakeRecogna	5951	5951	11902
FakeTrue	1791	1791	3582
BoatosBR	1516	1516	3032
Base 1 (Fake.br)	3600	3600	7200
Base 2 (Fake.br + FakeRecogna)	9551	9551	19102
Base 3 (+ FakeTrue)	11342	11342	22684
Base 4 (+ BoatosBR)	12858	12858	25716

Tabela 1 – Composição de notícias nas bases utilizadas, considerando a junção progressiva dos 4 corpora.

Fonte: Elaborado pela autora.

qualidade dos dados e, conseqüentemente, para o desempenho dos modelos (JURAFSKY; MARTIN, 2019).

Etapas	Descrição
Tokenização	Divide o texto em palavras (tokens).
Normalização	Converte todos os tokens para letras minúsculas.
Remoção de Ruídos	Elimina pontuação, símbolos e palavras irrelevantes (stopwords).
Lematização	Reduz as palavras à sua forma base (lema).

Tabela 2 – Etapas do Pré-processamento de Texto.

Fonte: Elaborado pela autora.

Neste trabalho, foi implementada uma função de pré-processamento que realiza as seguintes etapas principais:

1. **Tokenização:** A função inicia com a tokenização do texto, dividindo a sequência de caracteres em unidades menores chamadas *tokens*, geralmente palavras. Essa etapa é essencial para permitir a manipulação individual de cada palavra presente no texto.
2. **Normalização:** Após a tokenização, todos os *tokens* são convertidos para letras minúsculas (*lowercase*). Essa padronização evita que palavras como “Governo” e “governo” sejam tratadas como entidades distintas, contribuindo para a uniformidade do vocabulário.
3. **Remoção de *stopwords* e símbolos:** As *stopwords* são palavras comuns que geralmente não agregam valor semântico significativo ao contexto, como preposições, artigos e conjunções (por exemplo, “de”, “a”, “e”, “para”). Essas palavras são removidas, assim como *tokens* que não são alfanuméricos, reduzindo o ruído nos dados e melhorando a relevância das informações mantidas.

4. **Lematização:** A última etapa consiste na lematização dos *tokens*, um processo linguístico que transforma uma palavra em sua forma canônica ou de dicionário (lema). Por exemplo, verbos conjugados como “falando” ou “falei” são reduzidos à sua forma base “falar”. Isso reduz a dimensionalidade do vocabulário, agrupa variações morfológicas de uma mesma palavra e facilita a generalização dos modelos.

A importância do pré-processamento reside no fato de que modelos de aprendizado de máquina são sensíveis a variações superficiais nos dados. Um vocabulário mais coeso e relevante torna o espaço de representação mais eficiente e melhora a capacidade dos algoritmos de extrair padrões significativos. No contexto específico da classificação de textos em notícias falsas e verdadeiras, o pré-processamento ajuda a destacar os termos mais informativos, eliminar ruídos e diminuir a dispersão dos dados, contribuindo diretamente para um melhor desempenho dos classificadores (JURAFSKY; MARTIN, 2019).

- **Texto original:**

“Os cientistas estavam estudando os efeitos da vacina no combate à pandemia.”

- **Após tokenização:**

```
["Os", "cientistas", "estavam", "estudando", "os", "efeitos", "da",  
"vacina", "no", "combate", "à", "pandemia", "."]
```

- **Após normalização (lowercase):**

```
["os", "cientistas", "estavam", "estudando", "os", "efeitos", "da",  
"vacina", "no", "combate", "à", "pandemia", "."]
```

- **Após remoção de stopwords e símbolos:**

```
["cientistas", "estudando", "efeitos", "vacina", "combate", "pandemia"]
```

- **Após lematização:**

```
["cientista", "estudar", "efeito", "vacina", "combate", "pandemia"]
```

- **Resultado final do pré-processamento:**

```
cientista estudar efeito vacina combate pandemia
```

Em síntese, o pré-processamento não apenas organiza os dados, mas também atua como um filtro para garantir que os modelos recebam entradas de maior qualidade, promovendo resultados mais robustos, precisos e generalizáveis.

3.6 Modelos de Representação

O texto bruto “o presidente falou hoje” não pode ser fornecido diretamente a um classificador, ele precisa ser convertido em números significativos. Sem essa conversão, o modelo de aprendizado de máquina não saberia como lidar com os dados de texto.

Neste trabalho, foram empregadas três técnicas de representação vetorial para transformar os textos em entradas numéricas aptas ao uso por algoritmos de aprendizado de máquina: *Bag of Words* (BOW), *Term Frequency-Inverse Document Frequency* (TF-IDF) e Word2Vec.

3.6.1 *Bag of Words* - BOW

O *Bag of Words* (BOW) é uma das técnicas mais simples e populares de representação de texto para modelos de aprendizado de máquina. É um modelo baseado na frequência de palavras no *corpus*, sem considerar contexto ou ordem. Sua principal característica é que ele representa um texto como uma coleção de palavras (daí o termo “sacola”), ignorando a estrutura gramatical e a ordem das palavras. Cada texto é transformado em um vetor de características, onde cada dimensão do vetor corresponde à presença ou à frequência de uma palavra específica no conjunto de dados.

A técnica começa com a construção de um vocabulário, que é simplesmente um conjunto único de todas as palavras presentes no *corpus* (ou conjunto de textos). Em seguida, cada documento ou frase é representado como um vetor de contagens, onde cada posição do vetor corresponde a uma palavra do vocabulário, e o valor de cada posição é o número de vezes que a palavra aparece no documento.

3.6.1.1 Exemplo Prático

Para ilustrar o funcionamento do BOW, considere o seguinte exemplo de um pequeno *corpus* com três documentos:

Documento 1: “O presidente falou hoje.”

Documento 2: “Hoje o presidente falou sobre a política.”

Documento 3: “A política do governo foi discutida hoje.”

Pré-processamento

Foi realizado o pré-processamento dos textos, incluindo a tokenização, normalização (como letras minúsculas), remoção de *stopwords* e símbolos, além de lematização. O resultado é:

- **Documento 1:** [presidente, falar, hoje]
- **Documento 2:** [presidente, falar, hoje, política]
- **Documento 3:** [política, governo, discutir, hoje]

Vocabulário extraído (BoW)

Vocabulário = {presidente, falar, hoje, política, governo, discutir}

Matriz BoW (*Bag of Words*)

Cada documento é representado como um vetor que conta o número de vezes que cada palavra do vocabulário aparece no documento, sendo que a ordem das palavras no texto original não é considerada. Por exemplo, no Documento 1, a palavra “presidente” aparece uma vez, “falar” aparece uma vez, “hoje” aparece uma vez. Como a ordem das palavras não é relevante no BOW, considerando a ordem do vocabulário, isso resultaria no vetor [1, 1, 1, 0, 0, 0], que representa esse documento de forma numérica.

Os vetores para os documentos seriam os seguintes:

Documento	presidente	falar	hoje	política	governo	discutir
Documento 1	1	1	1	0	0	0
Documento 2	1	1	1	1	0	0
Documento 3	0	0	1	1	1	1

A técnica BOW é simples e eficaz, especialmente quando o texto é curto e a ordem das palavras não é tão importante. Entre as suas vantagens, podemos destacar:

- **Simplicidade:** A técnica é fácil de entender e implementar.
- **Eficiência:** Para muitos tipos de modelos de aprendizado de máquina, o BOW oferece uma maneira eficiente de representar texto, sem exigir modelos complexos.
- **Escalabilidade:** O BOW pode ser facilmente aplicado a grandes corpora.

No entanto, o BOW também apresenta algumas limitações:

- **Ignora a ordem das palavras:** O BOW não leva em consideração a ordem das palavras no texto, o que pode ser importante para compreender o significado de frases. Por exemplo, a frase “Maria elogiou João” tem um significado diferente de “João elogiou Maria”, mas o BOW representa ambas com vetores idênticos pois possuem as mesmas palavras. O agente da ação muda completamente. Na frase 1, Maria faz o elogio; na 2, João. Mas o BoW só enxerga as palavras: “Maria”, “elogiou”,

“João” — nas duas. Dessa forma, esse método não capta o significado profundo nem a estrutura da linguagem — apenas presença e frequência de palavras. Por isso, frases com significados distintos podem acabar com representações idênticas.

- **Dimensionalidade:** O número de características (dimensões do vetor) pode crescer significativamente com o aumento do tamanho do vocabulário, o que pode levar a problemas de alta dimensionalidade e aumento do custo computacional.
- **Dispersão:** A maioria dos vetores gerados pelo BOW tende a ser esparsa (ou seja, contém muitos zeros), já que nem todas as palavras do vocabulário aparecem em todos os documentos.

3.6.2 TF-IDF

A técnica **TF-IDF** (*Term Frequency – Inverse Document Frequency*) é amplamente utilizada na representação vetorial de textos e tem como objetivo medir a relevância de uma palavra em um documento dentro de um conjunto de documentos (*corpus*). Diferente do modelo *Bag of Words*, que considera apenas a frequência bruta dos termos, o TF-IDF leva em conta o quanto uma palavra é comum ou rara no *corpus* como um todo, penalizando termos muito frequentes e destacando os mais informativos.

A **frequência do termo** quantifica o número de vezes que uma palavra aparece em um documento, em relação ao número total de palavras naquele documento. A fórmula é dada por:

$$TF(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \quad (3.1)$$

Onde:

- $f_{t,d}$ é a frequência do termo t no documento d ;
- O denominador representa a soma das frequências de todos os termos no documento d .

A **frequência inversa de documentos** avalia o quão rara ou comum uma palavra é entre todos os documentos. O termo $IDF(t, D)$ não representa a frequência do termo t no *corpus* D , mas sim uma medida de raridade do termo t dentro do conjunto de documentos D . A fórmula é:

$$IDF(t, D) = \log \left(\frac{N}{1 + n_t} \right) + 1 \quad (3.2)$$

Onde:

- D é o conjunto de documentos (*corpus*);
- $N = |D|$ é o número total de documentos no *corpus*;
- n_t é o número de documentos nos quais o termo t aparece;
- O termo $1 + n_t$ evita divisão por zero e suaviza a medida.
- O $+1$ fora do log é uma forma adicional de suavização. Essa versão garante que o IDF nunca seja zero, mesmo para termos que aparecem em todos os documentos. Bibliotecas como o *scikit-learn* utilizam esse termo.

O valor final de TF-IDF é o produto das duas métricas anteriores:

$$TFIDF(t, d, D) = TF(t, d) \times IDF(t, D) \quad (3.3)$$

Interpretação:

- Se o termo aparece em quase todos os documentos, n_t é grande, o IDF será baixo e isso significa que o termo é comum e tem pouco valor discriminativo.
- Se o termo aparece em poucos documentos, n_t é pequeno, o IDF será alto e o termo é raro e pode ser mais importante para a classificação.

Para fins de demonstrar como o cálculo é realizado, considere os seguintes três documentos:

Documento 1: “o presidente falou hoje”

Documento 2: “hoje o presidente falou sobre a política”

Documento 3: “a política do governo foi discutida hoje”

Após o pré-processamento (remoção de *stopwords*, tokenização e lematização), suponha que o vocabulário gerado seja:

[presidente, falou, hoje, política, governo, discutida, sobre, foi, do]

A frequência de **presidente** no documento 1 é 1. Supondo que ela apareça em dois dos três documentos (Documento 1 e Documento 2), o cálculo de TF-IDF para **presidente** em Documento 1 seria:

- $TF(\text{“presidente”}, D1) = \frac{1}{4} = 0,25$

- $IDF(\text{"presidente"}) = \log\left(\frac{3}{1+2}\right) + 1 = \log(1) + 1 = 1$
- $TFIDF = 0,25 \times 1 = 0,25$

Ou seja, **presidente** tem peso 0,25 porque aparece em quase todos os documentos, sendo pouco informativa. Já termos como **governo**, que aparecem apenas em Documento 3, terão IDF maior e, portanto, maior importância relativa.

3.6.2.1 Exemplo Prático

Considere os seguintes documentos:

- **Documento 1:** “gato gosta de leite”
- **Documento 2:** “cachorro gosta de osso”
- **Documento 3:** “gato e cachorro não gostam um do outro”

Após pré-processamento:

Aplicou-se a tokenização, normalização, remoção de stopwords e símbolos, lematização. Resultado:

- Documento 1 $\rightarrow$ [gato, gostar, leite]
- Documento 2 $\rightarrow$ [cachorro, gostar, osso]
- Documento 3 $\rightarrow$ [gato, cachorro, gostar, outro]

Vocabulário final:

Vocabulário = {cachorro, gato, gostar, leite, osso, outro}

Frequência Termo (TF)

Termo	D1	D2	D3
cachorro	0	0,333	0,25
gato	0,333	0	0,25
gostar	0,333	0,333	0,25
leite	0,333	0	0
osso	0	0,333	0
outro	0	0	0,25

Frequência Inversa de Documento (IDF)

Com $N = 3$ documentos:

$$\text{IDF}(t) = \log\left(\frac{N}{1 + n_t}\right) + 1$$

Termo	IDF
cachorro	$\log(3/3) + 1 = 1$
gato	$\log(3/3) + 1 = 1$
gostar	$\log(3/4) + 1 \approx 0,887$
leite	$\log(3/2) + 1 \approx 1,176$
osso	$\log(3/2) + 1 \approx 1,176$
outro	$\log(3/2) + 1 \approx 1,176$

TF-IDF = TF × IDF

Termo	Documento 1	Documento 2	Documento 3
cachorro	0	$0,333 \times 1 = 0,333$	$0,25 \times 1 = 0,25$
gato	$0,333 \times 1 = 0,333$	0	$0,25 \times 1 = 0,25$
gostar	$0,333 \times 0,887 = 0,296$	0,296	$0,25 \times 0,887 = 0,222$
leite	$0,333 \times 1,176 = 0,392$	0	0
osso	0	$0,333 \times 1,176 = 0,392$	0
outro	0	0	$0,25 \times 1,176 = 0,294$

Vetores TF-IDF finais

Documento	Componentes do Vetor					
Documento 1	0	0,333	0,25	0	0	0
Documento 2	0,333	0	0,25	0	0	0
Documento 3	0,333	0,333	0,25	0	0	0

Matriz final TF-IDF (6 termos × 3 documentos)

$$\begin{bmatrix} \text{cachorro} & 0 & 0,333 & 0,25 \\ \text{gato} & 0,333 & 0 & 0,25 \\ \text{gostar} & 0,296 & 0,296 & 0,222 \\ \text{leite} & 0,392 & 0 & 0 \\ \text{osso} & 0 & 0,392 & 0 \\ \text{outro} & 0 & 0 & 0,294 \end{bmatrix}$$

3.6.3 Word2Vec

O **Word2Vec** é uma técnica de representação vetorial de palavras baseada em redes neurais artificiais. Desenvolvido por pesquisadores do Google em 2013, o Word2Vec é capaz de capturar relações semânticas e sintáticas entre palavras, representando-as como vetores densos em um espaço contínuo de alta dimensionalidade (MIKOLOV et al., 2013).

Diferentemente do BOW e do TF-IDF, que geram vetores esparsos e baseados em frequência, o Word2Vec aprende, a partir de grandes quantidades de texto, representações vetoriais que preservam relações de contexto. Por exemplo, os vetores das palavras *rei* e *rainha* estarão próximos no espaço vetorial e poderão até mesmo expressar relações de analogia, como:

$$\text{rei} - \text{homem} + \text{mulher} \approx \text{rainha}$$

Dado um *corpus* de texto, o *Skip-Gram* tenta prever as palavras de contexto a partir de uma palavra central (*target*). Por exemplo, dado o trecho:

O cachorro corre no parque

Com uma janela de contexto de tamanho 2, a palavra “corre” terá como pares de treinamento:

- (corre, cachorro)
- (corre, no)
- (corre, parque)

O Word2Vec pode ser treinado com dois modelos principais:

- **CBOW** (*Continuous Bag of Words*): prediz a palavra central a partir do contexto (as palavras vizinhas).
- **Skip-Gram**: prediz o contexto a partir da palavra central. Este modelo é mais eficiente para trabalhar com palavras menos frequentes.

Durante o treinamento, o Word2Vec aprende a posicionar palavras semelhantes em regiões próximas no espaço vetorial. Assim, palavras que aparecem em contextos parecidos (ex: “presidente”, “ministro”, “governador”) tendem a ter representações vetoriais próximas.

Suponha a frase: “O presidente falou sobre a política internacional”. Após o treinamento, palavras como *presidente* e *política* terão vetores similares a outras palavras do campo político. A similaridade entre palavras pode ser calculada por meio do cosseno do ângulo entre seus vetores:

$$\cos(\theta) = \frac{\vec{w}_1 \cdot \vec{w}_2}{\|\vec{w}_1\| \cdot \|\vec{w}_2\|}$$

Quanto mais próximo de 1 for esse valor, mais semelhantes são as palavras.

Essa técnica mapeia cada palavra do vocabulário para um vetor denso de dimensões fixas. É necessário, então, combinar os vetores das palavras de um documento para criar um vetor que represente esse documento. Isso pode ser feito por meio da média ou soma dos vetores de todas as palavras do texto, gerando um único vetor representativo do documento. Este vetor é então utilizado como entrada para os algoritmos de aprendizado de máquina.

3.6.3.1 Exemplo Prático

Neste exemplo, detalharemos o funcionamento do Word2Vec com *Skip-Gram*, desde a inicialização dos vetores até a obtenção da matriz final para ser utilizada como entrada em classificadores. Suponha que temos o seguinte *corpus* após o pré-processamento:

Documento 1: “O cachorro corre no parque.”

Documento 2: “O gato está dormindo no sofá.”

Documento 3: “O cachorro e o gato são amigos.”

Após o pré-processamento (tokenização, remoção de stopwords e lematização):

Documento 1: [“cachorro”, “corre”, “parque”]

Documento 2: [“gato”, “dormir”, “sofá”]

Documento 3: [“cachorro”, “gato”, “amigo”]

O objetivo é usar o modelo *Skip-Gram* para prever as palavras de contexto, ou seja, as palavras que estão ao redor da palavra central (alvo), usando palavras já normalizadas e sem *stopwords*. O modelo realiza isso da seguinte maneira:

Passo 1: Vetores Iniciais Aleatórios

O primeiro passo é atribuir vetores aleatórios às palavras no vocabulário (depois de pré-processadas). Esses vetores são aleatórios no início, mas à medida que o modelo treina, ele vai ajustando esses vetores. O tamanho desses vetores é um parâmetro que o

usuário define, e normalmente é algo entre 100 e 300 dimensões. No exemplo, para fins de simplificação usaremos 2 dimensões para facilitar a visualização. A Tabela 3 demonstra um exemplo de vetores iniciais (aleatórios):

Palavra	Dimensão 1	Dimensão 2
cachorro	0,1	-0,2
corre	0,3	0,4
parque	-0,4	0,2
gato	0,2	-0,3
dormir	0,5	-0,6
sofá	-0,1	0,7
amigo	0	0,1

Tabela 3 – Vetores iniciais atribuídos a cada palavra no início do treinamento (Word2Vec).

Fonte: Elaborado pela autora.

Passo 2: Previsão do Contexto

Para cada palavra alvo, o modelo tenta prever as palavras de contexto. Por exemplo, dado a palavra “cachorro”, o contexto (usando uma janela de contexto de tamanho 2) seria [“corre”, “parque”].

Palavra alvo: “cachorro”

Contexto: [“corre”, “parque”]

O modelo usa o vetor da palavra alvo “cachorro” (no caso, o vetor $[0.1, -0.2]$) e tenta prever os vetores das palavras de contexto (como “corre” e “parque”). O modelo faz uma previsão baseada nos vetores e depois compara o vetor de previsão com o vetor real da palavra contexto. Para fazer essa previsão, o modelo realiza um cálculo usando o produto escalar (ou similaridade) entre os vetores.

Passo 3: Cálculo do Erro (Função de Custo)

A previsão é comparada com as palavras reais (contexto) usando a função de custo. Normalmente a função de custo é a entropia cruzada ou a função de perda logarítmica. O erro é calculado com base no quanto o vetor previsto se afastou do vetor real da palavra contexto. Se a previsão estiver correta, o erro será pequeno. Se a previsão estiver errada, o erro será maior.

Passo 4: Ajuste dos Vetores (*Backpropagation*)

Com base no erro calculado, o modelo usa o algoritmo de otimização (como SGD ou Adam) para ajustar os vetores. Ele realiza o *backpropagation*, que é um algoritmo de ajuste de parâmetros em redes neurais, ajustando os vetores das palavras para reduzir o erro.

O vetor da palavra alvo e os vetores das palavras contexto são ajustados para que as palavras que aparecem com frequência juntas no texto fiquem com vetores próximos.

Esse processo se repete para todas as palavras e seus contextos em cada frase do *corpus*.

Passo 5: Vetores Ajustados após o Treinamento

Depois de várias iterações (épocas), os vetores das palavras são ajustados para refletir as relações semânticas entre elas. A Tabela 4 mostra os vetores ajustados após o treinamento do modelo.

Palavra	Dimensão 1	Dimensão 2
cachorro	0,9	0,7
corre	0,8	0,6
parque	0,7	0,8
gato	0,7	0,6
dormir	0,6	0,2
sofá	0,1	0,9
amigo	0,8	0,5

Tabela 4 – Vetores finais ajustados para cada palavra após o treinamento do Word2Vec.

Fonte: Elaborado pela autora.

Podemos perceber que as palavras relacionadas, como “cachorro” e “gato”, têm vetores próximos no espaço vetorial. O modelo aprendeu que essas palavras têm significados semelhantes, pois frequentemente aparecem em contextos semelhantes. A Figura 2 ilustra este cenário.

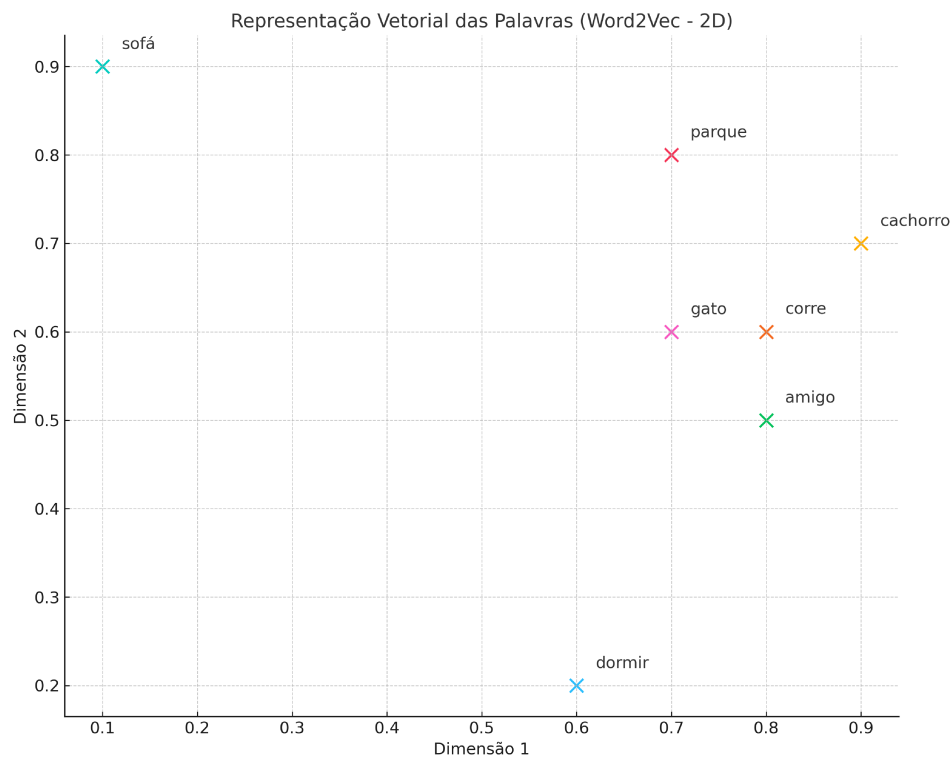


Figura 2 – Projeção 2D dos vetores das palavras após treinamento com Word2Vec.

Fonte: Elaborado pelo autora.

Passo 6: Matriz Final de Vetores

Após o treinamento, o modelo foi treinado e gerou vetores para cada palavra. Os vetores de palavras ajustados precisam ser representados em uma matriz que pode ser usada como entrada para o modelo de classificação. Como visto anteriormente, cada palavra do vocabulário é representada por um vetor denso de tamanho fixo (ex: 100 dimensões).

A matriz de Word2Vec em si é composta por:

- **Linhas:** correspondem às palavras do vocabulário do *corpus*;
- **Colunas:** representam as dimensões dos vetores de cada palavra.

Mas, para usar Word2Vec em um classificador, você precisa gerar um vetor por documento (como no BoW/TF-IDF). A forma mais comum é a média dos vetores das palavras do documento. O exemplo desses cálculos é demonstrado a seguir:

Documento 1

Palavras: cachorro, corre, parque

Vetores:

$$\text{cachorro} = [0,9|0,7] \quad \text{corre} = [0,8|0,6] \quad \text{parque} = [0,7|0,8]$$

Média dos vetores:

$$\frac{(0,9 + 0,8 + 0,7)}{3} = 0,8 \quad \frac{(0,7 + 0,6 + 0,8)}{3} = 0,7$$

$$\Rightarrow \text{Vetor do Documento 1} = [0,8|0,7]$$

Documento 2

Palavras: gato, dormir, sofá

Vetores:

$$\text{gato} = [0,7|0,6], \quad \text{dormir} = [0,6|0,2], \quad \text{sofá} = [0,1|0,9]$$

Média dos vetores:

$$\frac{(0,7 + 0,6 + 0,1)}{3} = 0,467 \quad \frac{(0,6 + 0,2 + 0,9)}{3} = 0,567$$

$$\Rightarrow \text{Vetor do Documento 2} \approx [0,467|0,567]$$

Documento 3

Palavras: cachorro, gato, amigo

Vetores:

$$\text{cachorro} = [0,9|0,7], \quad \text{gato} = [0,7|0,6], \quad \text{amigo} = [0,8|0,5]$$

Média dos vetores:

$$\frac{(0,9 + 0,7 + 0,8)}{3} = 0,8 \quad \frac{(0,7 + 0,6 + 0,5)}{3} = 0,6$$

$$\Rightarrow \text{Vetor do Documento 3} = [0,8|0,6]$$

Vantagens do Word2Vec:

- Captura relações semânticas e sintáticas entre palavras;
- Resulta em vetores densos, de dimensão fixa (ex.: 100 ou 300 dimensões);

Documento	Dimensão 1	Dimensão 2
Documento 1	0,800	0,700
Documento 2	0,467	0,567
Documento 3	0,800	0,600

Tabela 5 – Matriz final de entrada para o classificador com vetores médios (Word2Vec).

Fonte: Elaborado pela autora.

- Permite cálculos vetoriais úteis para inferência semântica;
- Mais informativo e compacto que BOW ou TF-IDF.

Limitações:

- Requer um volume considerável de dados para gerar boas representações;
- Palavras fora do vocabulário de treinamento (OOV – *Out Of Vocabulary*) não podem ser representadas;
- Não considera o significado completo de frases ou documentos, apenas das palavras individualmente.

3.7 Métodos de classificação

Para a etapa de classificação foram utilizados durante os testes deste trabalho quatro algoritmos amplamente reconhecidos na literatura de processamento de linguagem natural:

1. *Support Vector Classifier*
2. Regressão Logística
3. *Multinomial Naive Bayes*
4. *Random Forest*

A seleção desses métodos se justifica por serem algoritmos clássicos, bem consolidados academicamente, com vasta documentação e base teórica disponível. Além disso, são relativamente simples de implementar e interpretar, o que facilita tanto o processo experimental quanto a análise dos resultados obtidos. Ainda que existam técnicas mais sofisticadas — como redes neurais profundas, transformadores e modelos de aprendizado profundo como BERT e seus derivados —, a utilização de modelos mais complexos é

deixada como proposta para trabalhos futuros. Este trabalho opta por manter o foco em abordagens mais acessíveis e interpretáveis, sem prejuízo da qualidade analítica, buscando um equilíbrio entre viabilidade prática e fundamentação científica.

3.7.1 *Support Vector Classifier (SVC)*

O *Support Vector Classifier* (SVC) é uma implementação do algoritmo de *Support Vector Machine* (SVM) voltada para problemas de classificação supervisionada. Seu principal objetivo é encontrar um hiperplano ótimo que consiga separar os dados de classes diferentes com a maior margem possível. Essa margem é definida pela distância entre o hiperplano e os pontos de dados mais próximos de cada classe, conhecidos como vetores de suporte.

O SVC é eficaz em tarefas de classificação binária e também pode ser estendido para múltiplas classes. Ele é especialmente útil em situações onde os dados não são linearmente separáveis, utilizando para isso técnicas de transformação de espaço com os chamados kernels - funções que transformam os dados em um espaço de maior dimensão, onde a separação linear pode ser possível. Os kernels mais comuns são: linear, polinomial, RBF (*Radial Basis Function*) e sigmoide.

Além disso, o SVC possui um parâmetro de regularização (C) que controla o equilíbrio entre maximizar a margem e minimizar os erros de classificação. Um valor alto de C prioriza classificações corretas (menos tolerância ao erro), enquanto valores menores permitem margens maiores com mais tolerância a erros.

Em tarefas como detecção de *fake news*, o SVC tem se mostrado eficaz ao lidar com dados textuais vetorizados (como TF-IDF ou BoW), oferecendo bons resultados mesmo com conjuntos de dados limitados.

Imagine que temos dois grupos de notícias — verdadeiras e falsas — representados por pontos em um gráfico bidimensional, onde cada eixo representa uma característica do texto (por exemplo, número de palavras em maiúsculo e quantidade de pontuações). O SVC buscará traçar uma linha que separe esses grupos da melhor maneira, garantindo uma margem segura entre eles. Essa separação é representada graficamente pela Figura 3.

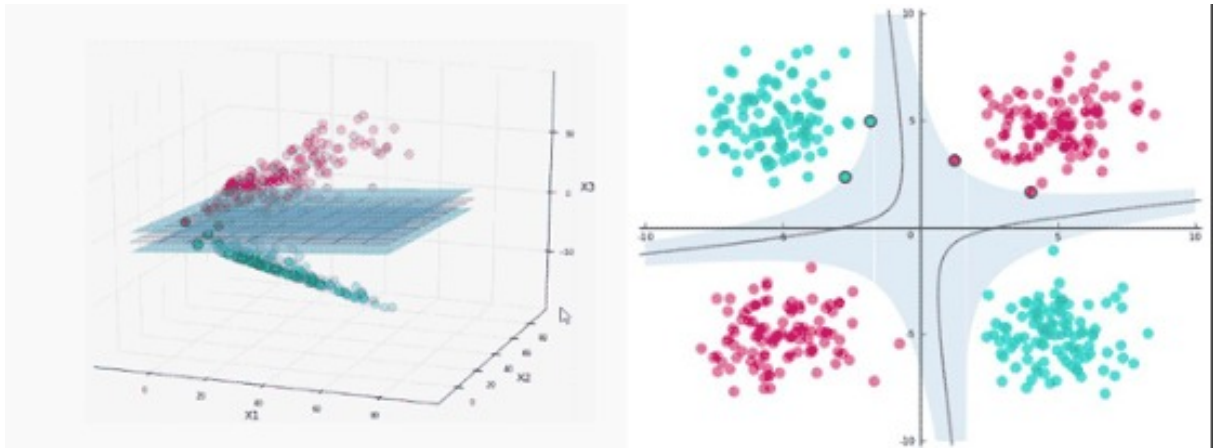


Figura 3 – Exemplo ilustrativo do funcionamento do SVC em um espaço bidimensional, separando duas classes com o hiperplano e suas margens.

Fonte: Soham Paul. *The Maths Behind Linear SVC Classifier*. Kaggle. Disponível em: <<https://www.kaggle.com/code/soham1024/the-maths-behind-linear-svc-classifier>>. Acesso em: maio de 2025.

Ao treinar o SVC com esse tipo de dado, o modelo aprende a distinguir padrões linguísticos ou estatísticos presentes nos textos de cada classe. Assim, ao receber um novo texto, ele pode prever com boa precisão se ele é verdadeiro ou falso com base nas características extraídas.

Exemplo Prático

Considere o seguinte conjunto de pontos em um plano bidimensional:

- **Classe A (label = 0):**

- Ponto 1: (1, 2)
- Ponto 2: (2, 3)

- **Classe B (label = 1):**

- Ponto 3: (4, 3)
- Ponto 4: (5, 2)

O SVC busca um **hiperplano de separação** com a maior margem possível entre as classes. Neste caso, o hiperplano pode ser:

$$y = -x + 5$$

Com as margens:

$$y = -x + 4 \quad \text{e} \quad y = -x + 6$$

Os pontos que tocam as margens são chamados de *vetores de suporte*.

Agora, suponha que queremos classificar o ponto $(3, 3)$. Substituindo na equação do hiperplano:

$$y = -x + 5 \Rightarrow y = -3 + 5 = 2$$

Como $3 > 2$, o ponto está acima do hiperplano e será classificado como **Classe A (0)**.

Conclusão: o SVC separa as classes por uma linha (ou plano) e usa apenas os vetores de suporte para definir sua fronteira de decisão.

Vantagens do SVC

- Funciona bem com alta dimensionalidade (muitos atributos).
- Possui bom desempenho com conjuntos de dados pequenos.
- Robusto contra *overfitting*, especialmente em problemas bem separáveis.

Limitações

- Pode ser mais lento com grandes volumes de dados.
- Requer normalização dos dados para melhor desempenho.
- A escolha do *kernel* pode impactar significativamente o resultado.

Overfitting (ou sobreajuste, em português) acontece quando um modelo de aprendizado de máquina aprende demais os dados de treino, incluindo ruídos e detalhes específicos que não são generalizáveis. Isso faz com que o modelo tenha um desempenho muito bom no conjunto de treinamento, mas não consiga generalizar bem para novos dados (dados de teste ou do mundo real).

Quando dizemos que o SVC é “robusto contra *overfitting*” significa que ele tende a encontrar uma fronteira de decisão que generaliza bem, ou seja, que separa as classes sem ficar dependente de detalhes específicos dos dados de treino. Isso acontece porque o SVC busca a margem máxima entre as classes, ou seja, ele tenta encontrar o espaço mais seguro possível entre os grupos, o que naturalmente ajuda a evitar o *overfitting*.

3.7.2 Regressão Logística

A Regressão Logística é uma técnica estatística que estima a probabilidade de uma instância pertencer a uma determinada classe com base em uma combinação linear das variáveis de entrada. É um método de classificação supervisionada amplamente utilizado em tarefas de categorização binária, ou seja, quando há duas classes possíveis. Apesar do nome “regressão”, ela é usada para resolver problemas de classificação, e não de regressão linear no sentido tradicional que prevê valores contínuos.

O objetivo da Regressão Logística é encontrar uma equação que estime a probabilidade de uma determinada entrada pertencer a uma classe específica. Essa equação se baseia na função logística (ou sigmoide), que transforma valores numéricos em um intervalo entre 0 e 1. Se o resultado for maior que 0,5, o modelo classifica como classe 1 (por exemplo, notícia falsa); caso contrário, classifica como classe 0 (notícia verdadeira). A Figura 4 demonstra visualmente, do lado direito como seria a construção da função sigmoide e, do lado esquerdo, uma projeção superior do plano com a separação entre as classes.

A fórmula da função sigmoide é:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

onde $z = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$, sendo x os atributos do texto, w os pesos aprendidos pelo modelo, e b o viés.

Exemplo Prático

Considere um conjunto de dados:

Horas de estudo (x)	Aprovado? (y)
1	0
2	0
3	0
4	1
5	1
6	1

O modelo da regressão logística usa a seguinte fórmula:

$$z = w \cdot x + b \quad \text{e} \quad \sigma(z) = \frac{1}{1 + e^{-z}}$$

Suponha que, após o treinamento, os parâmetros sejam:

$$w = 2, \quad b = -6$$

Assim, a função torna-se:

$$\sigma(x) = \frac{1}{1 + e^{-(2x-6)}}$$

Cálculo da probabilidade para alguns valores:

- Para $x = 3$: $\sigma(0) = 0,5$
- Para $x = 4$: $\sigma(2) \approx 0,88$
- Para $x = 5$: $\sigma(4) \approx 0,982$

Com um limiar de decisão $\theta = 0,5$ temos:

- $\sigma(x) < 0,5 \Rightarrow$ Classe 0 (reprovado)
- $\sigma(x) \geq 0,5 \Rightarrow$ Classe 1 (aprovado)

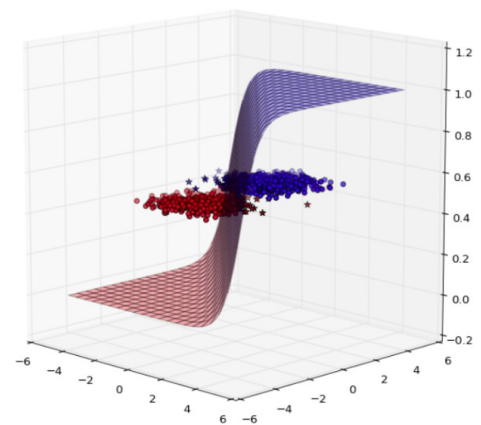
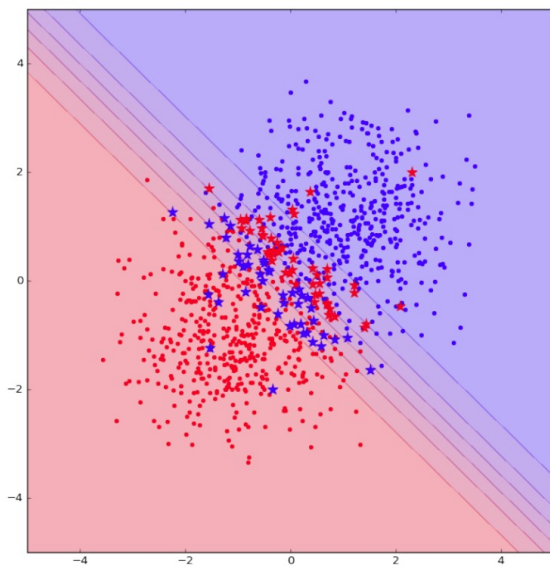


Figura 4 – Fronteira de decisão utilizando Regressão Logística.

Fonte: ([KANADE, 2016](#)).

Vantagens:

- Simples de entender e interpretar.
- Funciona bem quando as classes são linearmente separáveis.

Limitações:

- Pode não funcionar bem com classes que não são separáveis linearmente.
- Sensível a atributos irrelevantes ou correlacionados.

3.7.3 Multinomial Naive Bayes

O *Multinomial Naive Bayes* (NB) é um algoritmo de classificação supervisionada bastante utilizado em tarefas de categorização de texto. Ele é chamado de “Naive” (ingênuo) porque assume que os atributos são independentes entre si — uma simplificação que nem sempre é verdadeira, mas que costuma funcionar bem na prática.

O classificador *Naive Bayes* usa o Teorema de Bayes para calcular a probabilidade de um texto pertencer a uma determinada classe com base na frequência das palavras que ele contém. No caso do modelo multinomial, essa frequência é especialmente importante: ele considera quantas vezes cada palavra aparece no texto.

O Teorema de Bayes é dado por:

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)}$$

onde:

- $P(C|X)$ é a probabilidade da classe C dado o texto X ;
- $P(X|C)$ é a probabilidade de observar o texto X na classe C ;
- $P(C)$ é a probabilidade da classe C (frequência da classe no conjunto de treino);
- $P(X)$ é a probabilidade do texto (constante para todas as classes).

Suponha que a palavra “cura” aparece muitas vezes em textos classificados como *fake*. O modelo aprende essa associação e, ao encontrar essa palavra em um novo texto, aumenta a probabilidade de ele ser *fake* — mesmo que as demais palavras sejam neutras.

Exemplo Prático

Objetivo: Classificar se um aluno irá **passar** ou **reprovar**, com base na *Nota* e *Frequência*.

Conjunto de Treinamento:

Aluno	Nota	Frequência	Classe
A1	8,5	90	Passou
A2	9,0	95	Passou
A3	8,0	85	Passou
A4	5,0	70	Reprovou
A5	4,5	60	Reprovou
A6	4,0	65	Reprovou

Estatísticas por classe:

- **Passou:**
 - Nota: $\mu = 8,5$, $\sigma \approx 0,41$
 - Frequência: $\mu = 90$, $\sigma = 5$
- **Reprovou:**
 - Nota: $\mu = 4,5$, $\sigma \approx 0,41$
 - Frequência: $\mu = 65$, $\sigma = 5$

Novo aluno: Nota = 7,5, Frequência = 80

Fórmula da Gaussiana:

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

Probabilidades:

- **Classe Passou:**

$$P(\text{Nota} = 7,5 | \text{Passou}) \approx 0,087 \quad P(\text{Frequência} = 80 | \text{Passou}) \approx 0,011$$

$$P(\text{Dados} | \text{Passou}) = 0,087 \times 0,011 \approx 0,00096$$

- **Classe Reprovou:**

$$P(\text{Nota} = 7,5 | \text{Reprovou}) \approx 1,3 \times 10^{-12}, \quad P(\text{Frequência} = 80 | \text{Reprovou}) \approx 0,011$$

$$P(\text{Dados} | \text{Reprovou}) = (1,3 \times 10^{-12}) \times 0,011 \approx 1,43 \times 10^{-14}$$

Conclusão: Como $P(\text{Dados} | \text{Passou}) \gg P(\text{Dados} | \text{Reprovou})$, o aluno é classificado como **Passou**.

Vantagens:

- Rápido e eficiente, mesmo em conjuntos grandes de texto.
- Funciona bem com dados textuais esparsos.
- Fácil de implementar e interpretar.

Limitações:

- Assume que os atributos (palavras) são independentes, o que nem sempre é verdade.
- Pode ter desempenho inferior em textos onde a ordem das palavras ou o contexto são importantes.

3.7.4 *Random Forest*

O *Random Forest* (Floresta Aleatória) é um algoritmo de aprendizado supervisionado baseado em um conjunto de árvores de decisão. Ele combina os resultados de várias árvores para tomar uma decisão final mais precisa e robusta. Esse método é amplamente utilizado em tarefas de classificação e regressão, incluindo a detecção de notícias falsas.

A ideia principal do *Random Forest* é construir várias árvores de decisão durante o treinamento. Cada árvore recebe um subconjunto aleatório dos dados e das características (palavras, no caso de textos). No final, cada árvore “vota” sobre a classe de um novo texto, e a classe mais votada é a escolhida. Dessa forma, cada árvore aprende um padrão diferente dos dados e o modelo final é mais confiável, pois erros individuais das árvores são minimizados pelo conjunto.

Imagine que você tem 100 árvores analisando uma notícia. Se 70 delas indicarem que é *fake* e 30 disserem que é *verdadeira*, a floresta final classificará o texto como *fake*, com base na maioria dos votos.

3.7.4.1 Exemplo Prático

Objetivo: Classificar se uma pessoa irá **comprar** ou **não comprar** um produto com base em duas variáveis numéricas:

- Renda (em milhares de reais)
- Idade (em anos)

Conjunto de Treinamento

Pessoa	Renda (mil R\$)	Idade (anos)	Classe
P1	5	22	Não Comprar
P2	10	35	Comprar
P3	6	28	Não Comprar
P4	12	40	Comprar
P5	4	25	Não Comprar
P6	9	38	Comprar

Tabela 6 – Base de dados de exemplo para treinamento do *Random Forest*.*Fonte: Elaborado pela autora.***Etapas 1: Construção de Árvores Aleatórias**

Três árvores de decisão são criadas a partir de amostras aleatórias com reposição, e com seleção aleatória de atributos.

Árvore 1

- Dados utilizados: P1, P2, P5, P4
- Atributo selecionado: **Renda**
- Regras aprendidas:
 - Se Renda > 8 então **Comprar**
 - Caso contrário, **Não Comprar**

Árvore 2

- Dados utilizados: P2, P3, P4, P6
- Atributo selecionado: **Idade**
- Regras aprendidas:
 - Se Idade > 30 então **Comprar**
 - Caso contrário, **Não Comprar**

Árvore 3

- Dados utilizados: P2, P4, P6, P1
- Atributo selecionado: **Renda**
- Regras aprendidas:
 - Se Renda > 7 então **Comprar**

– Caso contrário, **Não Comprar**

Etapa 2: Classificação de Novo Exemplo

Novo exemplo a ser classificado:

- Renda = 9 (mil R\$)
- Idade = 32 (anos)

Previsões por árvore:

- **Árvore 1:** $\text{Renda} > 8 \Rightarrow \text{Comprar}$
- **Árvore 2:** $\text{Idade} > 30 \Rightarrow \text{Comprar}$
- **Árvore 3:** $\text{Renda} > 7 \Rightarrow \text{Comprar}$

Etapa 3: Votação Final

Votos: Comprar (3), Não Comprar (0)

Resultado final: Comprar

Conclusão:

O *Random Forest* combina várias árvores de decisão, onde:

- Cada árvore é treinada com subconjuntos aleatórios de dados e atributos;
- Cada árvore faz uma previsão individual;
- A classificação final é feita por maioria de votos.

Vantagens:

- Boa precisão mesmo com dados ruidosos ou incompletos.
- Reduz o risco de *overfitting* (excesso de aprendizado nos dados de treino).
- Funciona bem com dados de alta dimensionalidade (como textos vetorizados).

Limitações:

- Mais lento que classificadores simples, devido ao número de árvores.
- Difícil de interpretar: o modelo final é uma “caixa preta” com muitas árvores.

3.8 Métricas de Avaliação

Para avaliar o desempenho de modelos de classificação em tarefas de Processamento de Linguagem Natural (PLN), é essencial utilizar métricas que capturem diversos aspectos do comportamento do modelo, especialmente quando se trabalha com dados desbalanceados. As principais métricas utilizadas neste trabalho são: acurácia, precisão, revocação e *F1-score*.

3.8.1 Acurácia

A acurácia representa a proporção de previsões corretas em relação ao total de previsões realizadas. Ela é frequentemente utilizada como métrica principal quando há um equilíbrio entre as classes do problema.

A fórmula da acurácia é:

$$\text{Acurácia} = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.4)$$

Onde:

- *TP*: Verdadeiros Positivos
- *TN*: Verdadeiros Negativos
- *FP*: Falsos Positivos
- *FN*: Falsos Negativos

Exemplo: Se um modelo classifica corretamente 90 entre 100 amostras, a acurácia será $90/100 = 0,90$ (ou 90%).

Contudo, essa métrica pode ser enganosa em problemas com classes desbalanceadas, pois o modelo pode acertar apenas a classe majoritária e ainda assim apresentar alta acurácia.

3.8.2 Precisão

A precisão (ou *precision*) mede a proporção de exemplos que o modelo classificou como positivos e que realmente são positivos. Em outras palavras, ela responde à pergunta: “Dentre os casos que o modelo disse serem positivos, quantos realmente são?”. Essa métrica é especialmente útil quando o custo de um falso positivo é alto — por exemplo, ao acusar incorretamente uma notícia verdadeira como sendo falsa.

$$\text{Precisão} = \frac{TP}{TP + FP} \quad (3.5)$$

Exemplo: Se o modelo classificou 50 textos como falsos e apenas 40 estavam realmente incorretos, então houve 10 falsos positivos. A precisão será $40/(40 + 10) = 0,80$ (ou 80%).

3.8.3 Revocação

A revocação (ou *recall*), também conhecida como sensibilidade, mede a proporção de exemplos positivos corretamente identificados pelo modelo. Ela responde à pergunta: “Dentre todos os casos que realmente são positivos, quantos o modelo conseguiu encontrar?”. É uma métrica relevante quando se deseja minimizar os falsos negativos, como em diagnósticos médicos ou detecção de ameaças.

Enquanto a **precisão** foca na qualidade das previsões positivas (evitando falsos positivos), a **revocação** foca na cobertura dos positivos reais (evitando falsos negativos). Um modelo com alta precisão pode errar por deixar de identificar muitos casos reais (baixa revocação), e vice-versa.

$$\text{Revocação} = \frac{TP}{TP + FN} \quad (3.6)$$

Exemplo: Se existem 60 textos falsos no conjunto e o modelo consegue identificar 45, isso significa que 15 passaram despercebidos (falsos negativos). A revocação será $45/(45 + 15) = 0,75$ (ou 75%).

3.8.4 F1-Score

O *F1-score* é a média harmônica entre precisão e revocação. Ele busca um equilíbrio entre ambas as métricas e é útil quando se deseja uma medida única de desempenho em contextos com dados desbalanceados.

$$F1 = 2 \cdot \frac{\text{Precisão} \cdot \text{Revocação}}{\text{Precisão} + \text{Revocação}} \quad (3.7)$$

Exemplo: Supondo uma precisão de 0,80 e uma revocação de 0,75, o F1-score será:

$$F1 = 2 \cdot \frac{0,80 \cdot 0,75}{0,80 + 0,75} = 0,775$$

4 Desenvolvimento

Todo o desenvolvimento deste trabalho foi realizado em linguagem *Python*, versão (3.11.12), utilizando o ambiente computacional **Google Colab**, que oferece infraestrutura de execução em nuvem. A biblioteca que contém os métodos utilizados nesse trabalho, *scikit-learn*, está em sua versão (1.6.1). Apenas o método Word2Vec é oriundo da biblioteca *gensim* versão (4.3.3). Os métodos podem ser visualizados na Listagem 4.1.

```
1 from sklearn.feature_extraction.text import CountVectorizer
2 from sklearn.model_selection import train_test_split
3 from sklearn.svm import SVC
4 from sklearn.naive_bayes import MultinomialNB
5 from sklearn.linear_model import LogisticRegression
6 from sklearn.ensemble import RandomForestClassifier
7 from sklearn.feature_extraction.text import TfidfTransformer
8 from sklearn.metrics import accuracy_score, precision_score,
   recall_score, f1_score
9
10 from gensim.models import Word2Vec
```

Listagem 4.1 – Métodos utilizados da biblioteca scikit-learn

O código-fonte responsável pela execução dos testes descritos neste estudo encontra-se disponível publicamente no repositório: <https://github.com/laritretilato/fake_news_classification>. Além disso, o *script* completo utilizado para carregar as bases de dados, realizar as representações textuais, aplicar os classificadores e salvar os resultados foi incluído nos Apêndices deste trabalho, a fim de garantir a reprodutibilidade e transparência da metodologia adotada.

4.1 Detalhes de Implementação

Esta seção apresenta aspectos práticos e técnicos relacionados ao desenvolvimento do trabalho, com foco nas decisões tomadas ao longo da implementação dos experimentos. São discutidos impasses enfrentados durante o processo de codificação, alternativas consideradas, ajustes necessários e observações que se mostraram relevantes para a condução das análises. O objetivo é registrar os aprendizados e desafios encontrados na prática, complementando a fundamentação teórica com reflexões do processo de desenvolvimento.

4.1.1 Problema da normalização

Para a combinação do método de representação **Word2Vec** com o classificador **MultinomialNB**, foi necessária a **normalização dos dados**, uma vez que o **MultinomialNB** não aceita valores negativos como entrada. Caso contrário, ocorre o seguinte erro:

```
ValueError: Negative values in data passed to MultinomialNB (input X).
```

Para resolver esse problema, foi aplicado o **MinMaxScaler**, da biblioteca `<sklearn.preprocessing>`, que transforma os vetores de documentos para o intervalo $[0, 1]$, conforme ilustrado na Listagem 4.2.

```
1 # Apply MinMaxScaler to normalize document vectors
2 scaler = MinMaxScaler()
3 doc_vectors_normalized = scaler.fit_transform(doc_vectors)
```

Listagem 4.2 – Normalização dos vetores Word2Vec com MinMaxScaler.

4.1.2 Convergência do método

Durante a execução do classificador **LogisticRegression** com a configuração `max_iter=200`, foi gerado o seguinte aviso de convergência:

```
ConvergenceWarning: lbfgs failed to converge (status=1): STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.
```

Esse aviso indica que o otimizador **lbfgs** não conseguiu encontrar uma solução ótima dentro do número máximo de iterações especificado. Em outras palavras, o algoritmo não convergiu totalmente para um mínimo global, o que pode comprometer a qualidade do modelo aprendido.

Estudos recentes indicam que modelos de aprendizado de máquina, como a regressão logística, frequentemente alcançam acurácias entre 80% e 85% em tarefas de classificação binária de notícias falsas. Por exemplo, [Tembhurne e Almin \(2023\)](#) demonstraram que a regressão logística atingiu uma acurácia de 86,90% em experimentos de detecção de fake news, mostrando-se competitiva em relação a outros classificadores. Esses resultados sugerem que valores de acurácia nessa faixa são considerados satisfatórios e representam um bom desempenho para esse tipo de tarefa.

Apesar da mensagem de atenção em relação a convergência, o modelo foi treinado e apresentou métricas razoáveis de desempenho, com uma acurácia de 84,2%, o que sugere que ele atingiu uma solução parcial ou um mínimo local. Assim, o desempenho obtido, mesmo sem a convergência ideal, posiciona o modelo na fronteira entre o aceitável e o eficiente, demonstrando um bom potencial.

Para mitigar esse problema, foi ajustado o número máximo de iterações, de maneira empírica, elevando o valor de `max_iter` para 500, conforme ilustra a Listagem 4.3:

:

```
1 from sklearn.linear_model import LogisticRegression
2
3 # Aumenta o limite de iteracoes para permitir convergencia
4 model = LogisticRegression(max_iter=500)
```

Listagem 4.3 – Ajuste do número máximo de iterações.

Esse ajuste permitiu que o modelo tivesse mais tempo computacional para alcançar uma solução melhor ajustada aos dados, chegando a alcançar 97% de acurácia quando utilizado com o método de representação Bag of Words.

4.1.3 Generalização da função

Para simplificar e generalizar a chamada dos métodos de representação textual (BOW, TFIDF, Word2Vec), foi implementada uma função que seleciona dinamicamente o método a ser aplicado com base em uma *string* de entrada. Essa abordagem utiliza um dicionário que mapeia nomes de métodos para suas respectivas funções, permitindo que o código seja mais limpo, modular e facilmente extensível. Caso um método desconhecido seja informado, a função lança uma exceção.

O código básico dessa implementação é ilustrado a seguir:

```
def representation_method(method_name, dataframe):
    method_map = {
        'BOW': BOW,
        'TFIDF': TFIDF,
        'Word2Vec': Word2Vec
    }

    if method_name not in method_map:
        raise ValueError(f"Unknown method: {method_name}")

    return {
```

```
    'representation_method': method_name,
    'representation': method_map[method_name](dataframe)
}
```

4.1.4 Divisão balanceada dos dados de treino e teste

Para garantir que tanto o conjunto de treinamento quanto o de teste mantenham a proporção exata entre notícias verdadeiras e notícias falsas, foi utilizada a função `train_test_split` do *scikit-learn* com o parâmetro `stratify`.

Esse parâmetro assegura que a distribuição das classes (50% verdadeiras e 50% falsas) seja preservada em ambas as subdivisões, evitando vieses durante o treinamento e a avaliação dos modelos. Isso pois, caso não fosse utilizado esse parâmetro, não poderíamos garantir que haveria balanceamento durante a divisão dos dados entre a base de treino e a base de teste, colocando mais informações verdadeiras ou mais informações falsas.

O código utilizado foi:

```
bow_treino, bow_teste, rotulos_treino, rotulos_teste = train_test_split(
    bow_completa,
    rotulos_completos,
    test_size=0.2,
    random_state=52,
    stratify=rotulos_completos
)
```

Para validar o balanceamento, foi realizada a contagem simples das classes nos conjuntos gerados utilizando o método `value_counts()`, confirmando a manutenção da proporção original. Por exemplo:

```
rotulos_treino:
0    10286 (50.00\%)
1    10285 (50.00\%)
Name: label, dtype: int64
```

```
rotulos_teste:
0    2572 (50.00\%)
1    2573 (50.00\%)
Name: label, dtype: int64
```

Dessa forma, confirma-se que a proporção original das classes foi mantida, com aproximadamente 50% para cada classe em ambos os conjuntos.

Manter o balanceamento das classes é essencial para evitar vieses nos modelos de classificação binária, garantindo resultados mais confiáveis.

4.1.5 Escolha do modelo Word2Vec: CBOW vs Skip-Gram

No código utilizado para treinamento do modelo Word2Vec, a técnica empregada foi a *Continuous Bag of Words* (CBOW) por padrão. Isso ocorre porque o parâmetro `sg` não foi explicitamente definido na chamada da função `Word2Vec`, conforme o trecho:

```
w2v_model = Word2Vec(  
    sentences=tokenized_texts,  
    vector_size=vector_size,  
    window=window,  
    min_count=min_count,  
    workers=4  
)
```

Essa escolha impacta o treinamento das representações vetoriais das palavras, e a definição do parâmetro `sg` permite flexibilidade para se adaptar a diferentes tarefas e características dos dados.

4.2 Testes

Os testes realizados neste trabalho seguiram o fluxo ilustrado na Figura 5. Inicialmente, foram consideradas as quatro bases de dados distintas, compostas conforme descrito na legenda do respectivo nó do fluxograma: *Base 1* contendo apenas o `Fake.br`, *Base 2* composta por `Fake.br` + `FakeRecogna`, *Base 3* por `Fake.br` + `FakeRecogna` + `FakeTrue`, e *Base 4* que inclui `Fake.br` + `FakeRecogna` + `FakeTrue` + `BoatosBR`, seguindo a ideia de observar a base de dados sendo incrementada. Para cada base, foram aplicados os três métodos de representação textual: *Bag of Words* (BOW), TF-IDF e Word2Vec. Em seguida, para cada método de representação, foram avaliados os quatro classificadores diferentes: *Support Vector Classifier* (SVC), Regressão Logística (RL), *Multinomial Naive Bayes* (MNB) e *Random Forest* (RF). Todo o processo foi automatizado por meio de *scripts* de código em python desenvolvidos para este fim, cujo conteúdo completo está disponibilizado nos apêndices B.18, B.19 e B.20. Os resultados de cada execução foram salvos em arquivos no formato texto simples (`.txt`), contendo objetos JSON estruturados com informações detalhadas sobre o método de representação, classificador utilizado, métricas de desempenho (acurácia, precisão, revocação, F1-score) e a matriz de confusão. Os arquivos com os resultados estão demonstrados nos apêndices A.1, A.2 e A.3.

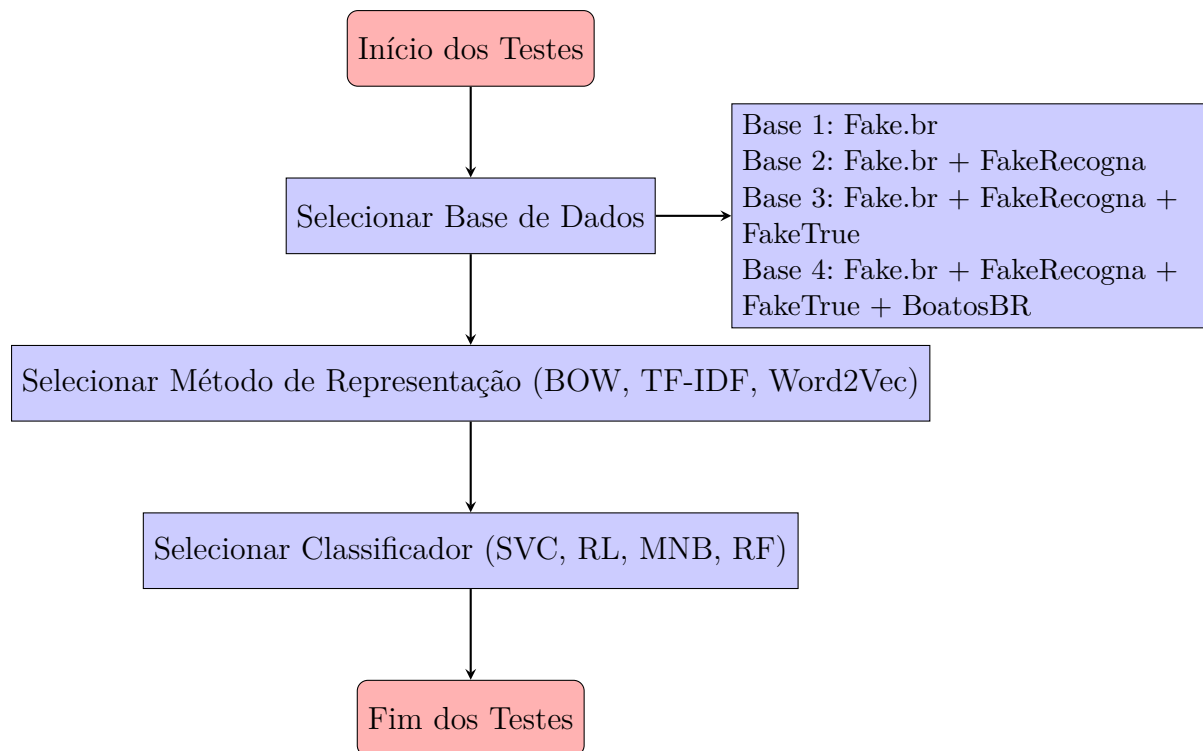


Figura 5 – Fluxograma simplificado do processo de execução dos testes com bases de dados, métodos de representação e classificadores.

Fonte: Elaborado pela autora.

A seguir, exemplifica-se o formato dos dados armazenados para a *Base 1*:

```

1 [
2   {
3     "method_representation": "BOW",
4     "method_classification": "SVC",
5     "result": {
6       "modelo": "SVC(kernel='linear')",
7       "acuracia": 0.9715,
8       "precisao": 0.9717,
9       "revocacao": 0.9715,
10      "f1_score": 0.9715,
11      "matriz_confusao": [[707, 13], [28, 692]]
12    }
13  }
14 ]

```

Listagem 4.4 – Exemplo de resultado salvo no arquivo .txt para a Base 1.

Esta organização facilita a análise comparativa dos métodos testados e contribui para a reproduzir os experimentos no mesmo formato, para então usar o padrão e criar os gráficos.

5 Resultados

Os resultados obtidos a partir dos experimentos realizados são apresentados e discutidos nesta seção. Inicialmente, são expostos os valores brutos de acurácia, precisão, revocação e F1-score para cada combinação de técnica de representação e método de classificação. Em seguida, esses resultados são analisados de forma comparativa por meio de visualizações gráficas, com o objetivo de destacar padrões de desempenho, identificar possíveis limitações e interpretar os achados com base na literatura. A análise visa responder à pergunta central do trabalho: quais abordagens são mais eficazes na tarefa de detecção de *fake news* em português.

5.1 Resultados brutos

Com base nos experimentos conduzidos, conforme detalhado na seção Testes, foram obtidos os resultados brutos obtidos considerando cada tipo de representação textual e cada classificador. Os resultados foram organizados em tabelas, onde se tem as métricas de avaliação na Tabela 7 para a Base 1, na Tabela 8 para a Base 2, na Tabela 9 para a Base 3 e na Tabela 10 para a Base 4.

Rep.	Classificador	Acurácia	Precisão	Revocação	F1-score
BOW	SVC	0.972	0.972	0.972	0.972
	Logistic Regression	0.972	0.972	0.972	0.972
	MultinomialNB	0.844	0.859	0.844	0.842
	Random Forest	0.948	0.950	0.948	0.948
TF-IDF	SVC	0.872	0.872	0.872	0.872
	Logistic Regression	0.838	0.839	0.838	0.838
	MultinomialNB	0.692	0.692	0.692	0.692
	Random Forest	0.872	0.872	0.872	0.872
Word2Vec	SVC	0.848	0.850	0.848	0.848
	Logistic Regression	0.838	0.839	0.838	0.838
	MultinomialNB	0.692	0.692	0.692	0.692
	Random Forest	0.872	0.872	0.872	0.872

Tabela 7 – Resultados da Base 1 por representação e classificador.

Fonte: Elaborado pela autora.

Rep.	Classificador	Acurácia	Precisão	Revocação	F1-score
BOW	SVC	0.949	0.949	0.949	0.949
	Logistic Regression	0.962	0.962	0.962	0.962
	MultinomialNB	0.733	0.735	0.733	0.733
	Random Forest	0.950	0.951	0.950	0.950
TF-IDF	SVC	0.893	0.894	0.894	0.894
	Logistic Regression	0.859	0.860	0.859	0.859
	MultinomialNB	0.732	0.732	0.732	0.732
	Random Forest	0.894	0.894	0.894	0.894
Word2Vec	SVC	0.862	0.865	0.862	0.862
	Logistic Regression	0.859	0.860	0.859	0.859
	MultinomialNB	0.732	0.732	0.732	0.732
	Random Forest	0.894	0.894	0.894	0.894

Tabela 8 – Resultados da Base 2 por representação e classificador.

Fonte: Elaborado pela autora.

Rep.	Classificador	Acurácia	Precisão	Revocação	F1-score
BOW	SVC	0.944	0.944	0.944	0.944
	Logistic Regression	0.954	0.954	0.954	0.954
	MultinomialNB	0.758	0.759	0.758	0.758
	Random Forest	0.937	0.938	0.937	0.937
TF-IDF	SVC	0.884	0.884	0.884	0.884
	Logistic Regression	0.834	0.834	0.834	0.834
	MultinomialNB	0.717	0.718	0.717	0.717
	Random Forest	0.884	0.884	0.884	0.884
Word2Vec	SVC	0.842	0.842	0.842	0.842
	Logistic Regression	0.834	0.834	0.834	0.834
	MultinomialNB	0.717	0.718	0.717	0.717
	Random Forest	0.884	0.884	0.884	0.884

Tabela 9 – Resultados da Base 3 por representação e classificador.

Fonte: Elaborado pela autora.

Rep.	Classificador	Acurácia	Precisão	Revocação	F1-score
BOW	SVC	0.948	0.949	0.948	0.948
	Logistic Regression	0.957	0.957	0.957	0.957
	MultinomialNB	0.774	0.775	0.774	0.774
	Random Forest	0.935	0.936	0.935	0.935
TF-IDF	SVC	0.893	0.893	0.893	0.893
	Logistic Regression	0.843	0.844	0.843	0.843
	MultinomialNB	0.722	0.722	0.722	0.722
	Random Forest	0.893	0.893	0.893	0.893
Word2Vec	SVC	0.846	0.847	0.846	0.846
	Logistic Regression	0.843	0.844	0.843	0.843
	MultinomialNB	0.722	0.722	0.722	0.722
	Random Forest	0.893	0.893	0.893	0.893

Tabela 10 – Resultados da Base 4 por representação e classificador.

Fonte: Elaborado pela autora.

5.2 Análise em gráficos

Com o objetivo de facilitar a interpretação dos dados experimentais e destacar padrões relevantes de desempenho entre os diferentes modelos testados, os resultados brutos foram organizados visualmente em gráficos. A representação gráfica permite uma análise comparativa mais clara entre os métodos de representação textual e os classificadores, evidenciando não apenas os melhores desempenhos, mas também a consistência e as variações entre as abordagens testadas. Esta subseção descreve a construção e a finalidade de cada tipo de visualização utilizada.

5.2.1 Organização Visual dos Resultados

Considerando o grande volume de informações geradas nesta pesquisa — envolvendo quatro classificadores, três técnicas de representação textual, quatro bases de dados e quatro métricas de avaliação (acurácia, precisão, revocação e *F1-score*) — optou-se por uma estratégia de visualização que entregue uma comparação eficiente entre os modelos. Foram aplicados dois tipos de gráficos para interpretação visual dos resultados:

- **Gráficos de calor (*heatmaps*):** permitem visualizar simultaneamente o desempenho de todos os modelos e representações ao longo das quatro bases, destacando padrões de evolução e estabilidade de forma condensada. Foram construídos por métrica de avaliação (1 gráfico para acurácia, 1 gráfico para precisão, 1 gráfico para revocação e 1 gráfico para *F1-score*), consolidando a performance de todos os modelos e representações ao longo das quatro bases.

- **Gráficos de radar (*radar plots*):** gerados separadamente para cada base - de 1 a 4 - e permitem comparar, em um único gráfico, o desempenho de todos os modelos sob diferentes representações textuais, facilitando a análise por base e entre bases (comparando-se um gráfico de radar com os demais). Os 4 gráficos de radar gerados representam os valores obtidos nos testes para *F1-score*. Os gráficos para as demais métricas também poderiam ser gerados, porém, o *F1-Score* foi a principal métrica observada já que é robusta, pois equilibra a precisão e a revocação, dois aspectos críticos na detecção confiável de notícias falsas. Ele penaliza modelos que tenham bom desempenho em apenas uma dessas dimensões, oferecendo uma visão mais justa e informativa da performance real dos classificadores.

A escolha por esses gráficos busca tornar a apresentação dos resultados mais clara e acessível, evitando excesso de informações visuais e facilitando a observação de tendências importantes, como o aumento do desempenho com o crescimento da base de dados e a possível estabilização dos resultados. Como reúnem várias informações em uma única imagem, esses gráficos ajudam a identificar, de forma rápida, quais combinações de representação e modelo se destacam e como se comportam ao longo das diferentes bases.

Para auxiliar a interpretação visual, é importante destacar que, nos gráficos de calor, quanto mais intensa a coloração azul de uma célula, melhor o desempenho obtido naquela combinação de representação, modelo e base, conforme o valor da métrica representada. Já nos gráficos de radar, quanto maior a amplitude das linhas (ou seja, quanto mais distante do centro do gráfico), melhor é o desempenho do modelo naquela combinação específica, permitindo comparações diretas entre as representações em uma mesma base de dados.

5.2.2 Heatmap por métrica

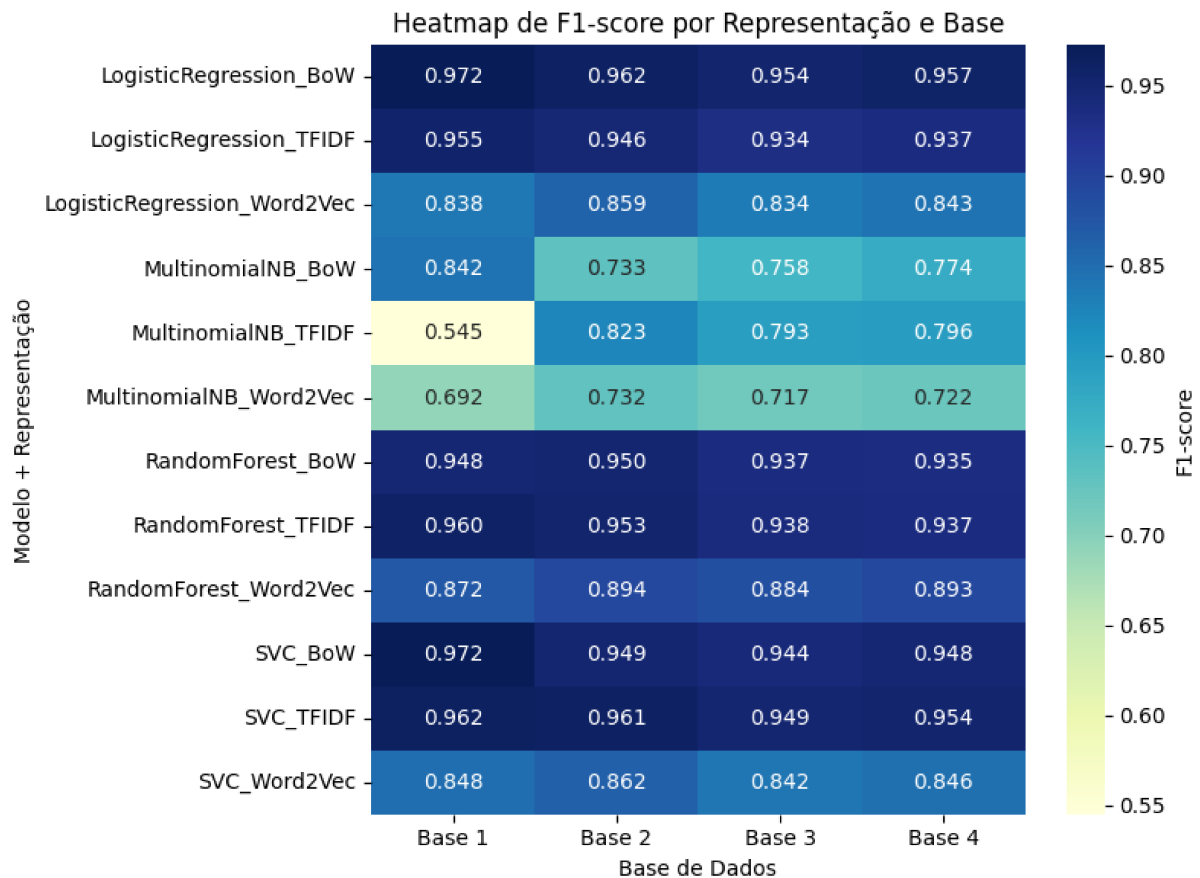


Figura 6 – Evolução do $F1$ -score dos modelos.

Fonte: Elaborado pela autora.

O *heatmap* da Figura 6 destaca a superioridade das representações BoW e TFIDF, especialmente quando utilizadas com os classificadores SVC e Regressão Logística. Em contraste, o Word2Vec obteve desempenho inferior, sobretudo com o classificador Naive Bayes, evidenciando sua menor eficácia nesse contexto. O que mais destaca-se nessa imagem é o classificador *Naive Bayes*, que no geral teve desempenho inferior aos outros métodos, mas podemos notar que, no caso do TFIDF teve um salto significativo na métrica entre as Bases 1 e 2 - o que reforça a hipótese.

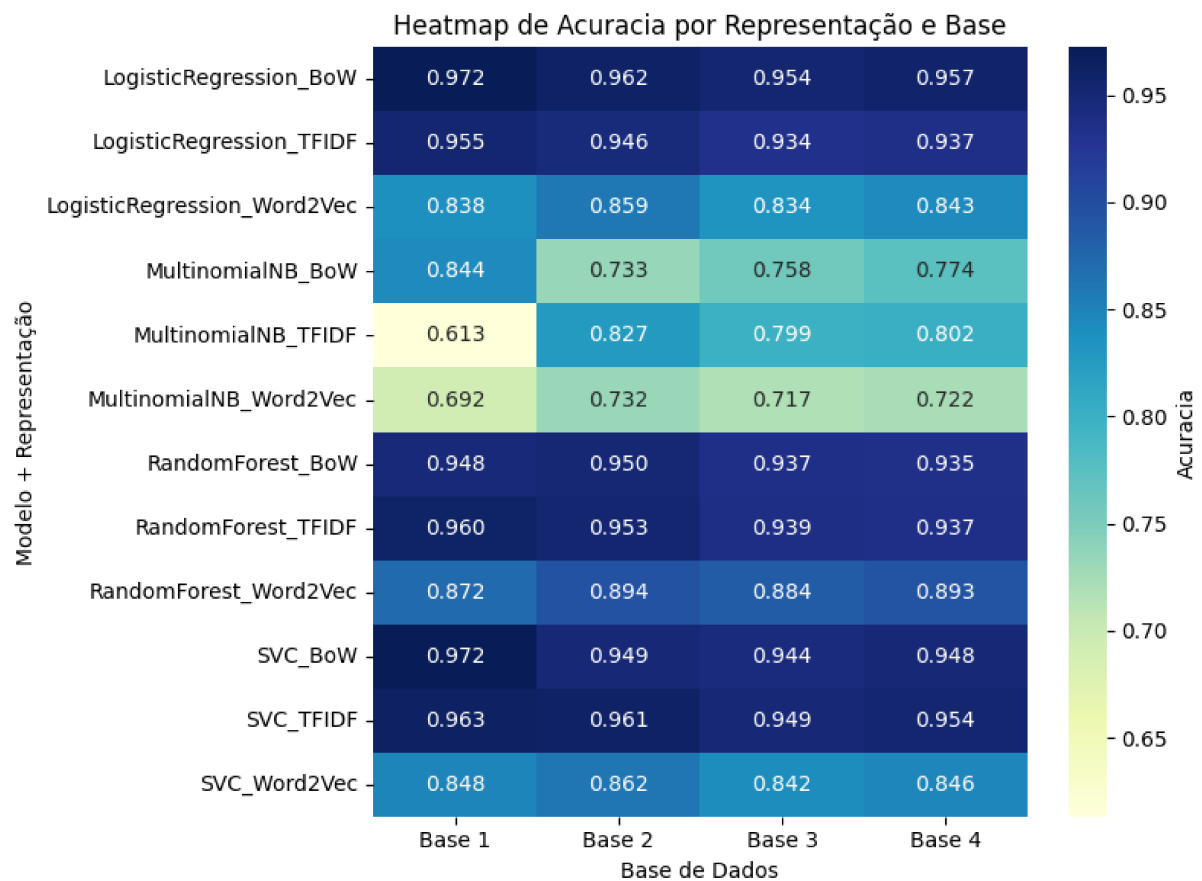


Figura 7 – Evolução da acurácia dos modelos.

Fonte: Elaborado pela autora.

Conforme mostra o *heatmap* de acurácia da Figura 7, os classificadores baseados em SVC e Regressão Logística mantêm uma performance elevada com BoW e TF-IDF. Os resultados mais baixos associados ao Word2Vec confirmam a tendência observada nas demais métricas.

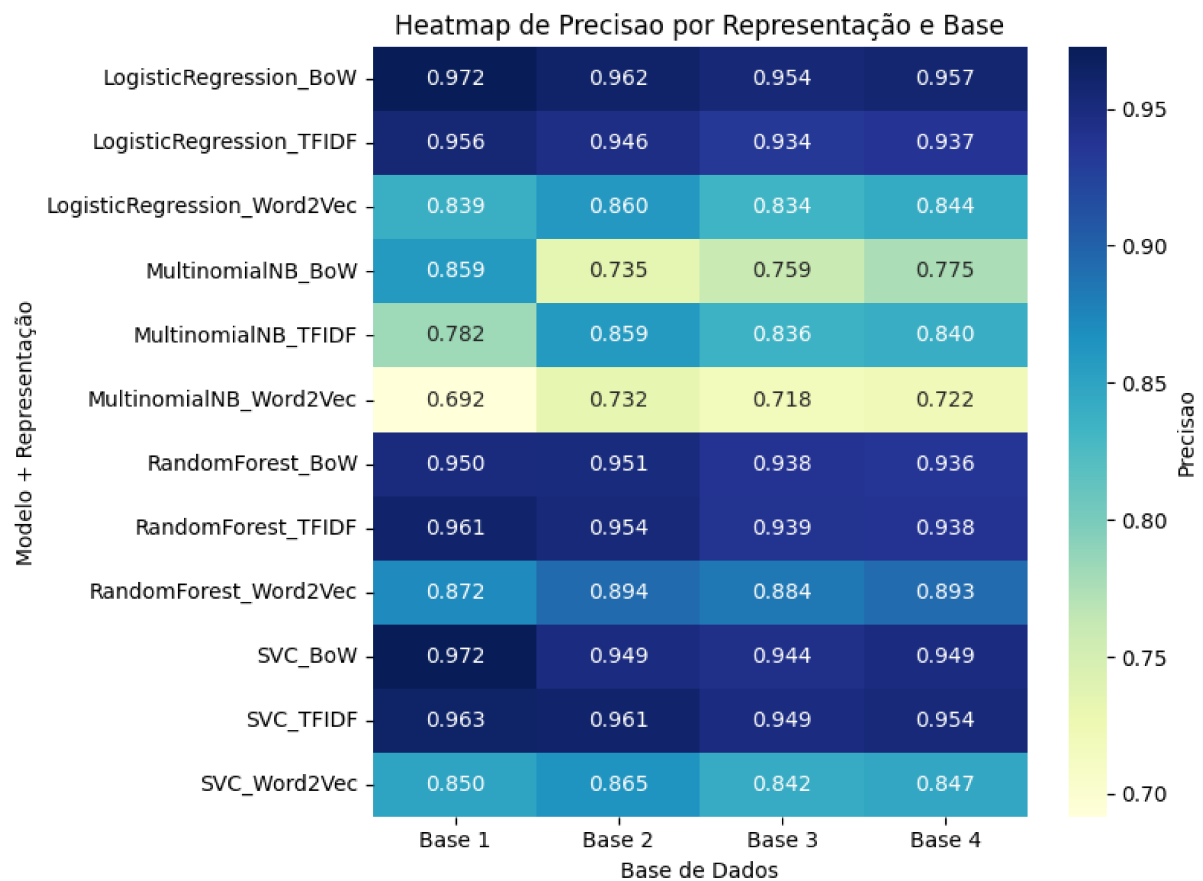


Figura 8 – Evolução da precisão dos modelos.

Fonte: Elaborado pela autora.

No *heatmap* de precisão da Figura 8, a vantagem dos métodos BoW e TF-IDF também é evidente. Nota-se que os classificadores lineares apresentam maior estabilidade, enquanto o Word2Vec demonstra variação e desempenho mais modesto, especialmente com *Naive Bayes*.

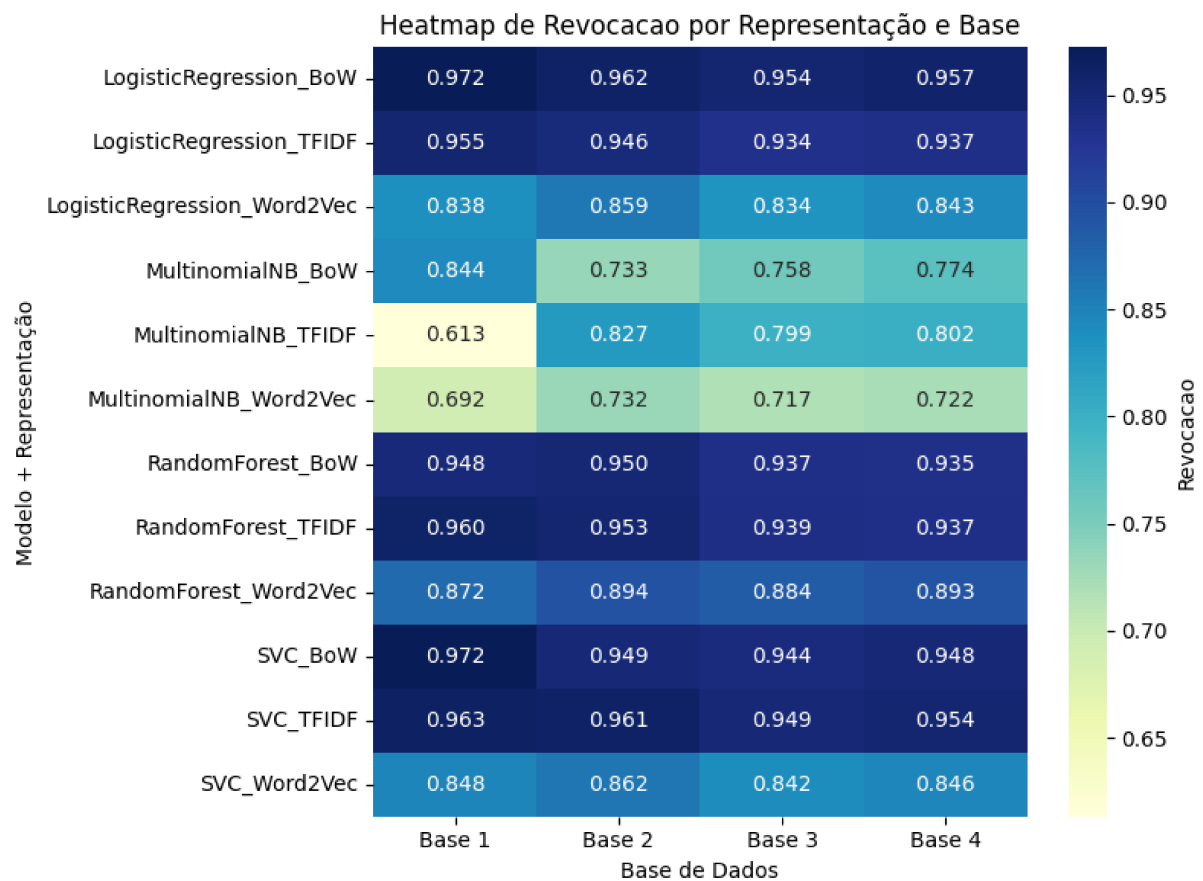


Figura 9 – Evolução da revocação dos modelos.

Fonte: Elaborado pela autora.

Para a Figura 9 de revocação, apresentada neste *heatmap*, acompanha o padrão geral das demais métricas, favorecendo BoW e TF-IDF. Os modelos *Random Forest* e SVC são os mais consistentes, enquanto o Word2Vec, mais uma vez, apresenta limitações no desempenho.

5.2.3 Radar de $F1$ -score por base de dados

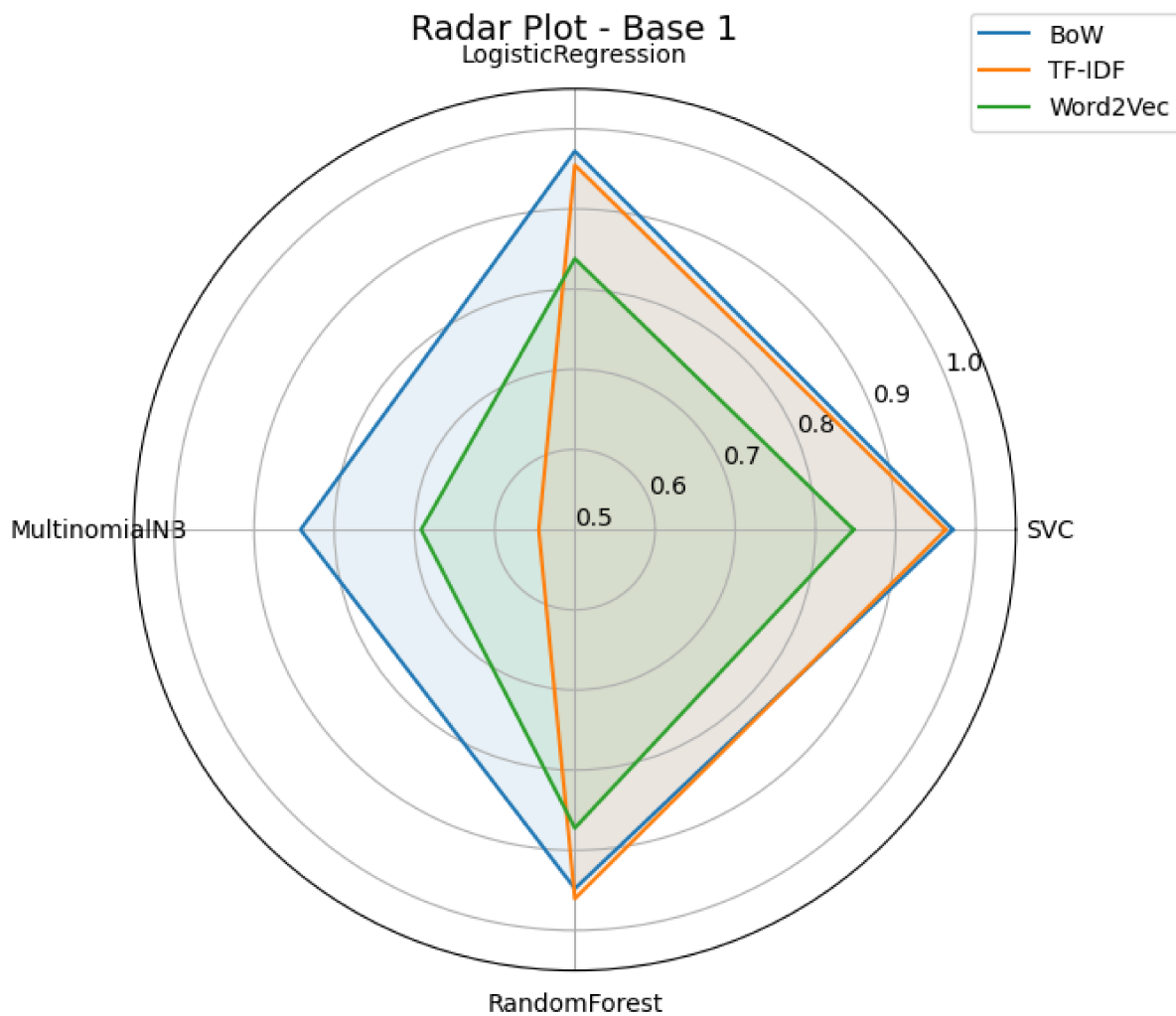


Figura 10 – Radar com os valores de $F1$ -score obtidos pelos modelos de classificação em diferentes representações textuais para a Base 1.

Fonte: Elaborado pela autora.

O gráfico de radar da Base 1 na Figura 10 revela que as representações BoW e TF-IDF proporcionaram os melhores resultados de F1-score, com destaque para os classificadores SVC e Regressão Logística. O Word2Vec, por sua vez, teve desempenho mais discreto em todas as combinações.

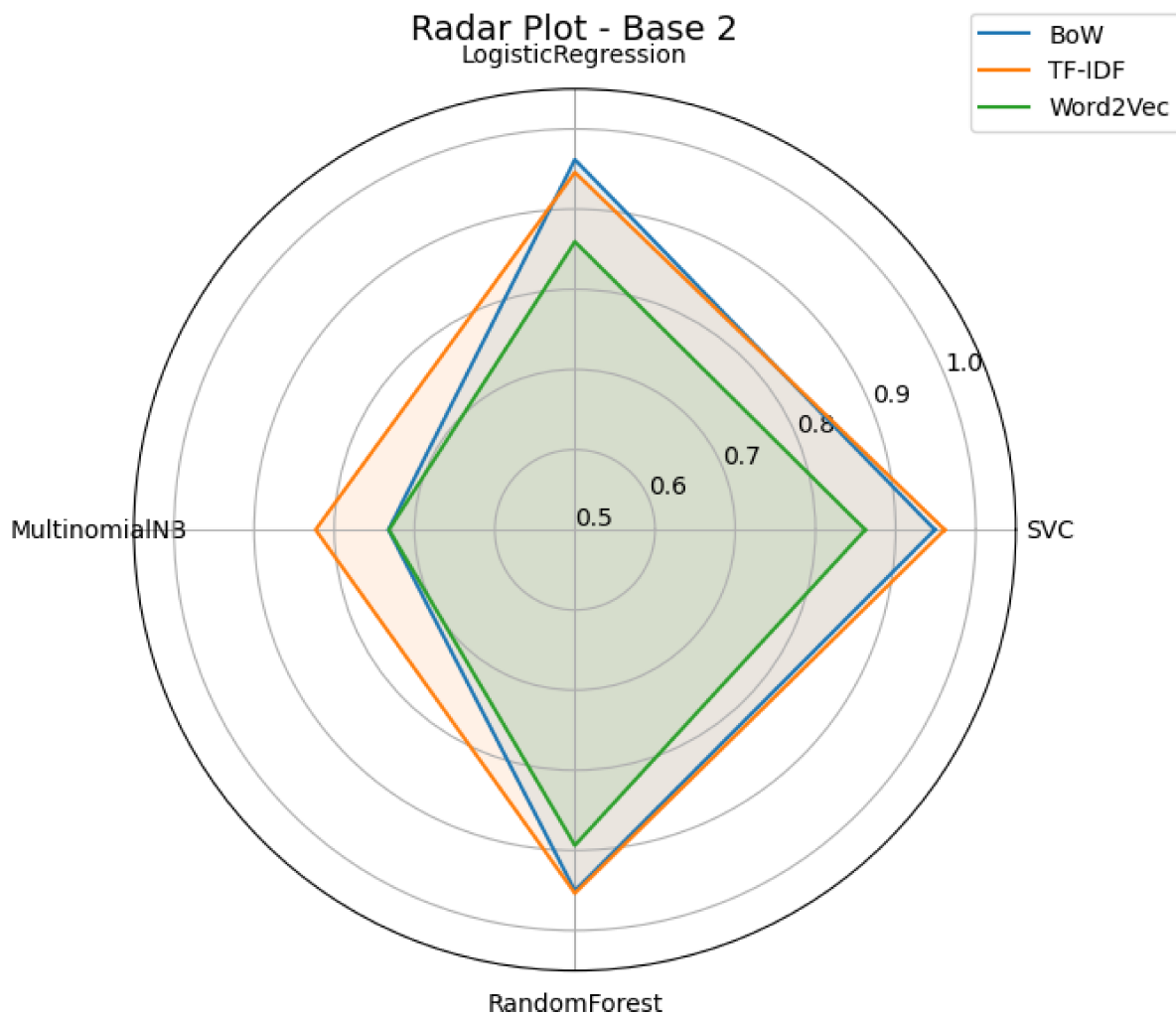


Figura 11 – Radar com os valores de $F1$ -score obtidos pelos modelos de classificação em diferentes representações textuais para a Base 2.

Fonte: Elaborado pela autora.

A Figura 11 para a Base 2 o radar mostra uma consistência entre BoW e TF-IDF, com resultados expressivos para Regressão Logística e Random Forest. O Word2Vec aparece com menor F1-score, reforçando a superioridade das abordagens tradicionais nesta base.

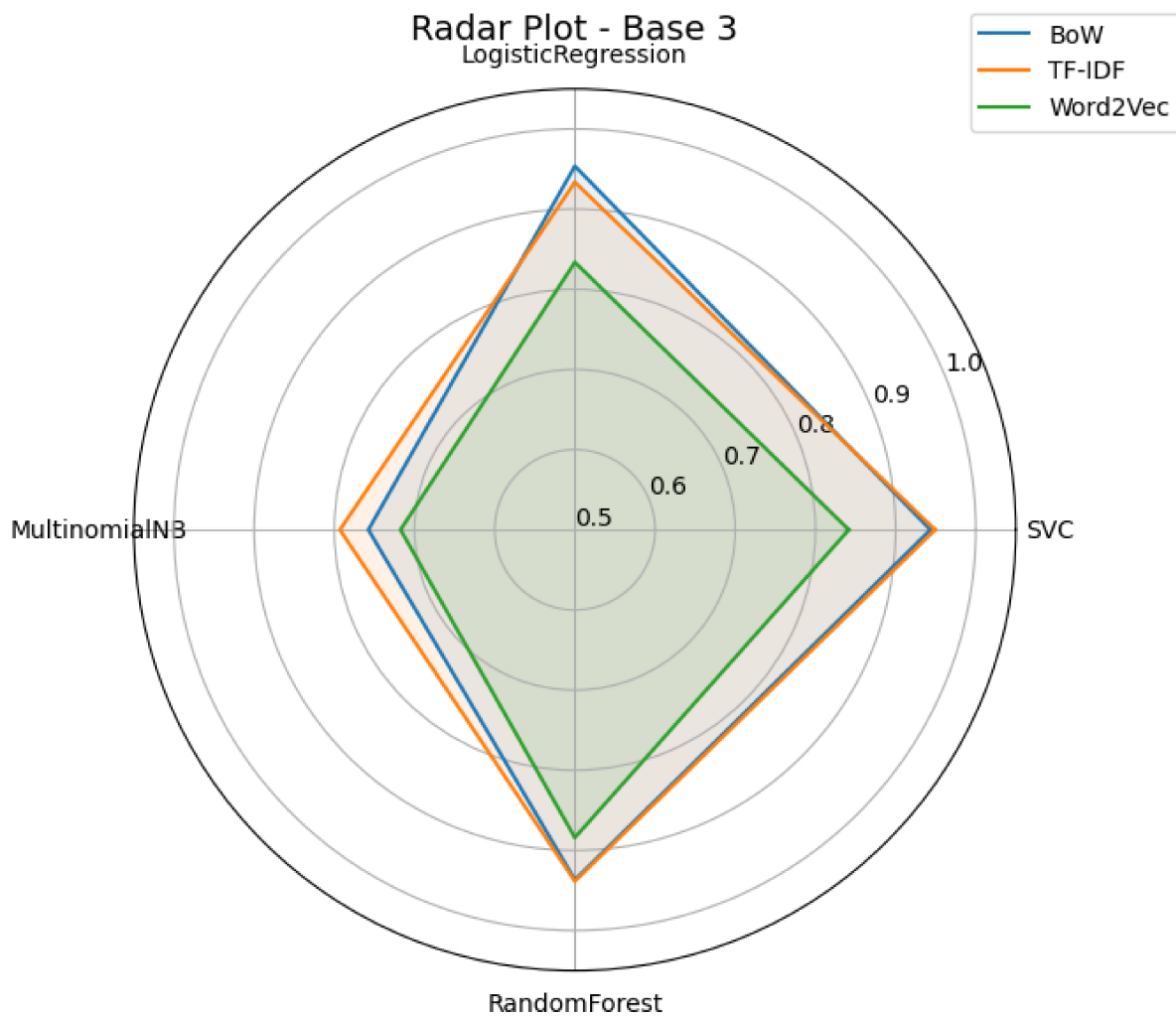


Figura 12 – Radar com os valores de $F1$ -score obtidos pelos modelos de classificação em diferentes representações textuais para a Base 3.

Fonte: Elaborado pela autora.

A Figura 12 para a Base 3 apresenta um cenário semelhante, com BoW e TF-IDF se sobressaindo novamente. O classificador *Naive Bayes* manteve desempenho inferior em todas as representações, especialmente com Word2Vec, que mais uma vez se mostrou limitado.

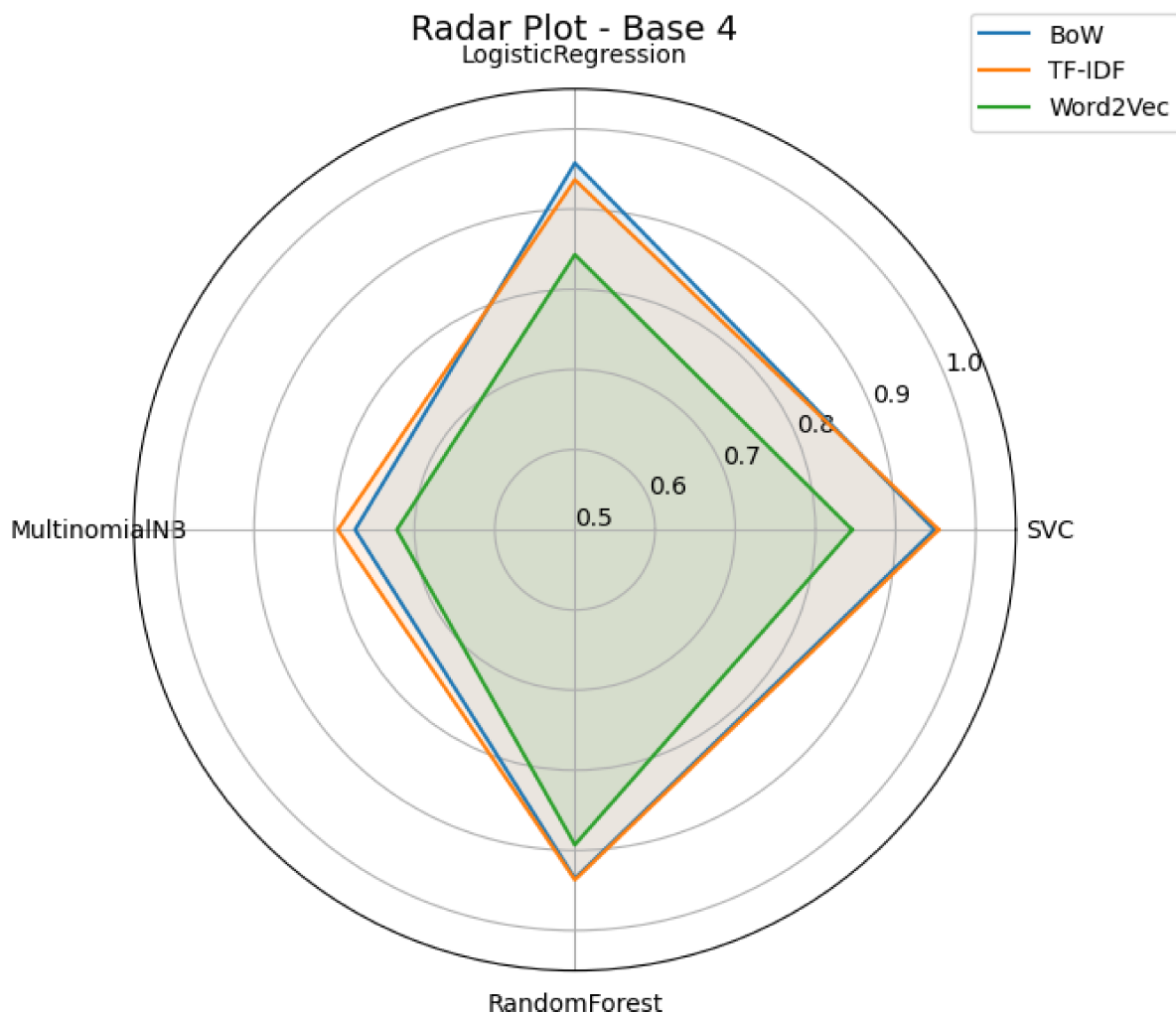


Figura 13 – Radar com os valores de $F1$ -score obtidos pelos modelos de classificação em diferentes representações textuais para a Base 4.

Fonte: Elaborado pela autora.

Na Base 4, Figura 13, observa-se um padrão consolidado de vantagem para BoW e TF-IDF, enquanto o Word2Vec aparece com menor $F1$ -score entre os classificadores - reforçando a teoria sobre estabilização e saturação. SVC e Regressão Logística continuam sendo os modelos com melhor desempenho geral.

5.3 Análise dos Resultados

A partir das visualizações apresentadas, esta subseção realiza uma análise crítica e interpretativa dos resultados obtidos. São considerados aspectos como o comportamento dos modelos em diferentes bases de dados, as variações de desempenho entre as técnicas de representação textual e os classificadores, além da influência das métricas utilizadas na avaliação. A análise busca, assim, contextualizar os achados com base nos objetivos do

trabalho e nas hipóteses levantadas inicialmente.

5.3.1 Gráficos de Radar

A análise dos gráficos de radar evidencia uma tendência clara: à medida que o volume de dados aumenta (da Base 1 para a Base 4), o desempenho dos modelos melhora, especialmente nas representações BoW e TF-IDF. Nas bases iniciais, é possível observar maior variação entre os classificadores e uma vantagem consistente dos modelos SVC e Regressão Logística. Já nas bases maiores (Base 3 e Base 4), os modelos tendem a se aproximar em desempenho, indicando um possível ponto de saturação, onde novos dados trazem ganhos marginais.

Em termos de representação textual, o TF-IDF mostra desempenho mais estável e competitivo em todas as bases, especialmente quando combinado com os classificadores SVC e *Random Forest*. O Word2Vec, por outro lado, apresenta desempenho inferior na maioria das combinações, sobretudo com o classificador *Multinomial Naive Bayes*, o que pode ser explicado pelo fato de que a representação por média vetorial tende a simplificar demais as informações do texto, reduzindo a capacidade do Multinomial NB de discriminar entre classes em modelos mais simples.

5.3.2 Gráficos de Calor

Os *heatmaps* permitem visualizar, para cada métrica, a evolução do desempenho dos modelos ao longo das diferentes bases. Observa-se que:

- **Acurácia e *F1-score*** apresentam um crescimento discreto, porém consistente, da Base 1 para a Base 2, seguido por uma estabilização nas Bases 3 e 4. Esse comportamento reforça a hipótese de que há um ponto de saturação a partir do qual a adição de novos dados resulta em ganhos sutis.
- **Precisão** mantém valores elevados principalmente nos classificadores SVC e Regressão Logística com as representações BoW e TF-IDF, o que sugere que esses modelos são mais confiáveis na identificação correta dos exemplos positivos.
- **Revocação** permanece equilibrada nesses mesmos modelos, indicando um bom comprometimento entre precisão e cobertura.

De modo geral, os modelos baseados em TF-IDF obtiveram os melhores desempenhos médios, tanto em acurácia quanto em *F1-score*. O classificador MultinomialNB, embora simples e eficiente computacionalmente, apresentou resultados inferiores com as

representações BoW e Word2Vec, mas demonstrou melhora perceptível quando combinado ao TF-IDF, refletindo maior compatibilidade com essa forma de representação de frequência ponderada.

6 Conclusão

Os resultados obtidos nesta pesquisa apoiam a hipótese central do estudo: o aumento progressivo do volume de dados tende a **melhorar** ou **estabilizar** o desempenho dos modelos de classificação de notícias falsas. Essa tendência é mais evidente na transição das Bases 1 e 2, onde os modelos apresentaram avanços suaves, porém consistentes, em diversas métricas de avaliação. A partir da Base 3, observa-se uma estabilização dos resultados, sugerindo um possível ponto de saturação, no qual a adição de novos dados contribui com ganhos marginais.

Por exemplo, ao observar a representação BoW com o classificador *Random Forest*, a acurácia evoluiu de aproximadamente 0,9479 na Base 1 para 0,9502 na Base 2, e manteve-se em níveis elevados nas bases seguintes. De forma mais expressiva, o modelo TF-IDF com *Multinomial Naive Bayes* apresentou crescimento na acurácia de 0,613 (Base 1) para 0,827 (Base 2), refletindo um ganho real de desempenho com a ampliação da base. Por outro lado, houve casos de variações discretas ou negativas, como no modelo TF-IDF com SVC, cuja acurácia passou de 0,9625 (Base 1) para 0,9613 (Base 2). Mesmo nesses cenários, as métricas mantiveram-se em patamares altos, evidenciando estabilidade e robustez do modelo diante de maior diversidade textual.

Esses resultados demonstram que, apesar dos ganhos numéricos serem suaves — geralmente limitados à segunda casa decimal —, o crescimento da base de dados proporcionou maior variabilidade linguística, contribuindo para o refinamento e generalização dos modelos de predição. O comportamento estável das métricas com o aumento da base sugere que os modelos conseguiram adaptar-se bem às variações sem perda de desempenho, o que é desejável em cenários de dados do mundo real.

Entre as representações textuais, TF-IDF destacou-se como a mais eficaz em conjunto com os classificadores SVC e *Random Forest*, enquanto BoW também apresentou bom desempenho, especialmente com Regressão Logística. Já o Word2Vec, embora com desempenho inferior nas médias, obteve bons resultados com *Random Forest*, revelando que esse método de classificação tem potencial em contextos mais complexos de representação de textos.

As visualizações gráficas — compostas por gráficos de radar (um por base) e gráficos de calor (um por métrica) — foram essenciais para evidenciar os padrões de estabilidade, evolução e saturação, permitindo comparações diretas entre modelos e representações de forma clara e compacta.

Conclui-se, portanto, que o aumento da base de dados, mesmo não elevando dras-

ticamente os valores das métricas, favoreceu a construção de modelos mais robustos, ajustados e generalizáveis. Essas observações reforçam a importância de se trabalhar com conjuntos de dados amplos e variados, ao mesmo tempo em que destacam o valor de representações textuais bem escolhidas e classificadores adequados ao problema.

Apesar dos resultados positivos, é importante reconhecer algumas limitações deste trabalho. O pré-processamento e a forma como os textos foram transformados em vetores podem ter removido detalhes importantes da linguagem que ajudariam na classificação. Além disso, os dados utilizados são compostos principalmente por textos jornalísticos, o que limita a análise a esse tipo de linguagem. Isso pode dificultar a aplicação dos modelos em outros contextos, como redes sociais ou mensagens instantâneas, que costumam ter textos mais curtos e informais. Por fim, os modelos foram treinados com bases fixas, sem considerar situações em que os dados chegam de forma contínua. Essas limitações indicam caminhos interessantes para trabalhos futuros, como testar os modelos em fontes mais variadas ou usar técnicas de aprendizado que permitam atualização contínua.

6.0.1 Desempenho do Word2Vec

Os resultados obtidos com a técnica Word2Vec apresentaram, de modo geral, desempenho inferior quando comparados com as abordagens tradicionais de representação textual, como BoW e TF-IDF. Esta diferença pode ser explicada por diversos fatores técnicos e conceituais.

Primeiramente, o Word2Vec depende fortemente da qualidade e do tamanho do corpus utilizado para seu treinamento. Como a técnica aprende vetores baseando-se no contexto das palavras, corpora pequenos ou muito específicos — como os utilizados neste trabalho, voltados para a detecção de *fake news* em português — podem não fornecer dados suficientes para a construção de representações significativas (MIKOLOV et al., 2013; CAMACHO-COLLADOS; PILEHVAR, 2018). Em contrapartida, BoW e TF-IDF exploram diretamente a frequência e a presença de termos, sendo altamente eficazes em tarefas que envolvem vocabulário discriminativo (JURAFSKY; MARTIN, 2019).

Além disso, é importante destacar que o Word2Vec, por gerar vetores densos e contínuos, normalmente precisa ser agregado (por média, soma etc.) para representar um documento completo. Esse tipo de agregação, embora simples e funcional, promove a perda de informação estrutural e sintática, além de suavizar as distinções entre documentos de classes diferentes (LAU; BALDWIN, 2016). Como a tarefa de detecção de *fake news* depende fortemente da presença de termos ou expressões específicas, essa redução da riqueza das informações do texto pode comprometer significativamente o desempenho do modelo.

Outro fator relevante é que classificadores lineares como SVC e Regressão Logística

tendem a obter melhor desempenho com vetores esparsos e de alta dimensionalidade, como os gerados por BoW e TF-IDF. Isso se deve à capacidade desses classificadores de aprender padrões baseados na presença e ausência de *tokens* informativos, o que não é diretamente viável com *embeddings* densos e contínuos (ZHANG; ZHAO; LECUN, 2015).

Conforme apontado por Wang et al. (2018), abordagens baseadas em *embeddings*, como o Word2Vec, podem não ser as mais eficazes para tarefas de classificação textual quando utilizadas isoladamente com modelos lineares, especialmente em cenários com bases de dados limitadas ou altamente específicas. Segundo os autores:

Tradução livre: “*Embeddings de palavras pré-treinadas nem sempre superam as representações tradicionais de características em tarefas de classificação de texto, especialmente quando os dados são pequenos ou específicos de domínio*”.(WANG; MANNING; SCHÜTZE, 2018).

Dessa forma, os resultados obtidos neste trabalho corroboram com achados da literatura, indicando que, para tarefas supervisionadas de detecção de *fake news* em português, técnicas tradicionais como BoW e TF-IDF continuam sendo mais eficazes, sobretudo quando combinadas com classificadores lineares robustos como SVC e Regressão Logística.

7 Trabalhos Futuros

Este trabalho utilizou 4 corporas distintos, unificando-os e realizando o treinamento dos modelos de classificação que não são adaptados a receber novos dados de treinamento se já estão treinados. Por isso, a cada novo teste feito, ou seja, aumentando a base com um novo corpora, o modelo foi completamente reiniciado. Se há um acréscimo nos dados de treinamento, os algoritmos precisaram ser treinados novamente com todos os dados (antigos + novos). Os algoritmos utilizados nesse trabalho - SVC, *Random Forest*, *Logistic Regression* - não possuem suporte a atualização incremental.

Considerando que, no mundo real, o volume de informações e a disseminação de notícias falsas ocorrem de forma contínua e acelerada — com ampla diversidade linguística, temáticas variadas e mudanças constantes na forma de comunicação —, torna-se importante avaliar abordagens que consigam lidar com esse caráter dinâmico. Em contextos assim, os dados não são estáticos e os modelos de classificação precisam se adaptar progressivamente às novas entradas. Nesse sentido, uma proposta relevante para trabalhos futuros seria a utilização de técnicas de aprendizado incremental (ou aprendizado online), que permitem atualizar o modelo à medida que novos dados se tornam disponíveis, sem a necessidade de reinicialização completa. Essa abordagem pode tornar os sistemas mais eficientes, responsivos e compatíveis com a realidade de ambientes em constante transformação, como o da desinformação digital.

Existem soluções em que é possível adicionar novos dados a um modelo já treinado. Essa capacidade de aprendizado contínuo permitem atualizar o modelo com novos dados sem precisar efetuar reprocessamento integral do conjunto de dados, ou seja, aprendem por lotes e conseguem adaptar-se com novos exemplos gradualmente. Exemplos de métodos de aprendizado incremental:

- SGDClassifier, SGDRegressor, MultinomialNB, BernoulliNB, PassiveAggressiveClassifier possuem a função `.partial_fit()`
- Alguns modelos de redes neurais, com arquitetura adequada, permitem reuso do modelo com novos batches de treinamento
- Modelos em bibliotecas como River ou Vowpal Wabbit foram desenhadas para uso contínuo em *streaming*

Além disso há diversas possibilidades de expansão e aprimoramento da pesquisa. Como sugestões para trabalhos futuros, destacam-se:

- **Explorar Representações Semânticas Mais Avançadas (BERT, RoBERTa, etc.):**

Utilizar modelos de linguagem baseados em Transformers, como BERT ou RoBERTa, que incorporam contexto e semântica profunda, pode melhorar significativamente o desempenho na classificação de textos. Esses modelos poderiam ser comparados com as abordagens clássicas aqui testadas.

- **Análise do Desempenho por Tipo de *fake news*:**

Realizar uma análise mais granular para verificar se o desempenho dos modelos varia conforme o tipo de notícia falsa (política, saúde, economia etc.). Isso pode exigir rotulagem mais detalhada ou agrupamento temático.

- **Testar Arquiteturas de Classificação Neurais:**

Modelos como LSTM, CNN ou redes neurais profundas poderiam ser testados em conjunto com Word2Vec ou *embeddings* mais complexos para verificar se oferecem ganho adicional de desempenho e se conseguem se adaptar de maneira estável ao longo do crescimento da base de treino.

- **Aplicar Técnicas de Balanceamento de Dados:**

Investigar o impacto de técnicas de *oversampling* (SMOTE) ou *undersampling*, caso haja desbalanceamento entre classes, e avaliar como isso afeta as métricas de desempenho.

- **Avaliação com Dados Reais de Redes Sociais:**

Testar os modelos com textos extraídos diretamente de redes sociais (como Twitter, Facebook ou WhatsApp), que tendem a ser mais curtos, informais e ruidosos, para avaliar a robustez do sistema em contextos mais desafiadores.

- **Estudo de Interpretabilidade e Explicabilidade dos Modelos:**

Investigar como explicar as decisões dos modelos (por meio de SHAP, LIME ou outras ferramentas) pode ser um diferencial em contextos jornalísticos ou judiciais, onde justificar a classificação é tão importante quanto acertar.

- **Fusão de Fontes Heterogêneas:**

Investigar os efeitos da fusão de bases de dados oriundas de diferentes fontes, como notícias jornalísticas, textos de redes sociais (por exemplo, Twitter e Facebook) e mensagens de aplicativos como WhatsApp. Essas fontes apresentam variações significativas na estrutura textual, grau de formalidade e presença de ruído linguístico. A integração desses estilos distintos pode oferecer um cenário mais desafiador, porém

mais realista, permitindo avaliar a capacidade de generalização dos modelos e sua robustez diante de textos com características heterogêneas.

Essas extensões podem contribuir significativamente para o avanço das pesquisas em detecção de desinformação e para o desenvolvimento de ferramentas mais robustas, interpretáveis e aplicáveis a cenários reais.

Referências

- BASTOS, L. M.; CRUZ, A. K. B. S. da; CRUZ, P. T. M. B. da; TEIXEIRA, M. M.; NETO, C. de S. S. **Fake News detection using decision tree, support vector machine and K-Nearest neighbors algorithms**. 2024. <<https://doi.org/10.5281/zenodo.14057842>>. Disponível em: <<https://zenodo.org/records/14057842>>. Acesso em: 7 mai. 2025. Citado na página 24.
- BISHOP, C. M. **Pattern Recognition and Machine Learning**. New York: Springer, 2006. Citado na página 26.
- BRASIL, B. N. **Eleição de 2020 terá mesmos problemas de fake news de 2018, dizem especialistas**. 2020. <<https://www.bbc.com/portuguese/brasil-53187041>>. Acesso em: 15 maio 2025. Citado na página 15.
- BRASIL, C. **Twitter vai banir quem postar de forma recorrente fake news sobre COVID-19**. 2021. Disponível em: <<https://www.cnnbrasil.com.br/tecnologia/twitter-vai-banir-quem-postar-de-forma-recorrente-fake-news-sobre-covid-19/>>. Acesso em: 15 mai. 2025. Citado na página 14.
- CAMACHO-COLLADOS, J.; PILEHVAR, M. T. Word embeddings: A survey. **Foundations and Trends in Information Retrieval**, Now Publishers, v. 11, n. 1, p. 1–89, 2018. Citado na página 80.
- CAMBRIA, E.; WHITE, B. Jumping nlp curves: A review of natural language processing research. **IEEE Computational Intelligence Magazine**, IEEE, v. 9, n. 2, p. 48–57, 2014. Citado na página 25.
- CANTARINO, F. H. S. **Criação de um corpus português para auxiliar a identificação de notícias verdadeiras e falsas**. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação), 2024. Disponível em: <<https://repositorio.ufu.br/handle/123456789/44215>>. Citado 2 vezes nas páginas 21 e 30.
- CHAVARRO, J.; CARVALHO, J.; PORTELA, T.; SILVA, J. Faketruebr: Um corpus brasileiro de notícias falsas. In: **Anais da XVIII Escola Regional de Banco de Dados**. Porto Alegre, RS, Brasil: SBC, 2023. p. 108–117. ISSN 2595-413X. Disponível em: <<https://sol.sbc.org.br/index.php/erbd/article/view/24352>>. Citado 2 vezes nas páginas 20 e 29.
- DIAS, C. R. M. **Towards fake news detection in Portuguese: New dataset and a claim-based approach for automated detection**. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação), 2019. Disponível em: <<http://hdl.handle.net/10183/198538>>. Citado na página 22.
- FISCHER, M.; HAQUE, R.; STYNES, P.; PATHAK, P. Identifying fake news in brazilian portuguese. In: **Natural Language Processing and Information Systems**. Springer, 2022. (Lecture Notes in Computer Science), p. 111–118. Disponível em: <https://www.researchgate.net/publication/361353998_Identifying_Fake_News_in_Brazilian_Portuguese>. Citado na página 22.

GARCIA, G. L.; AFONSO, L. C. S.; PAPA, J. P. FakeRecogna: A new brazilian corpus for fake news detection. In: **Proceedings of the 15th International Conference on Computational Processing of the Portuguese Language (PROPOR 2022)**. Springer, 2022. (Lecture Notes in Computer Science, v. 13207), p. 57–67. ISBN 978-3-031-08472-0. Disponível em: <https://www.researchgate.net/publication/359318957_FakeRecogna_A_New_Brazilian_Corpus_for_Fake_News_Detection>. Citado 2 vezes nas páginas 20 e 28.

GIORDANI, L.; DARÚ, G. H.; QUEIROZ, R. G. d. S.; BUZINARO, V.; NEIVA, D. K.; GUZMAN, D. C. F.; HENRIQUES, M. J.; JUNIOR, O. A. G.; LOUZADA, F. Automatic identification of fake news written in the portuguese language spoken in brazil. **Revista de la Facultad de Ingeniería**, 2023. Disponível em: <<https://repositorio.usp.br/item/003186882>>. Acesso em: 9 mai. 2025. Citado na página 21.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. 1st. ed. Cambridge, MA: MIT Press, 2016. Disponível em: <<https://www.deeplearningbook.org/>>. Citado na página 17.

HESTNESS, J.; NARANG, S.; ARDALANI, N.; DIAMOS, G.; JUN, H.; KIANINEJAD, H.; PATWARY, M. M. A.; YANG, Y.; ZHOU, Y. Deep learning scaling is predictable, empirically. **arXiv preprint arXiv:1712.00409**, 2017. Citado na página 17.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing**. 3. ed. Pearson, 2019. Disponível em: <<https://web.stanford.edu/~jurafsky/slp3/>>. Citado 5 vezes nas páginas 13, 25, 32, 33 e 80.

KANADE, V. **Machine Learning – MT 2016: Classification – Logistic Regression**. 2016. <<https://www.cs.ox.ac.uk/people/varun.kanade/teaching/ML-MT2016/slides/slides08.pdf>>. Acesso em: 19 maio 2025. Citado na página 51.

KAPLAN, J.; MCCANDLISH, S.; HENIGHAN, T.; BROWN, T. B.; CHESS, B.; CHILD, R.; GRAY, S.; RADFORD, A.; WU, J.; AMODEI, D. Scaling laws for neural language models. **arXiv preprint arXiv:2001.08361**, 2020. Citado na página 17.

LAU, J. H.; BALDWIN, T. An empirical evaluation of doc2vec with practical insights into document embedding generation. In: **Proceedings of the 1st Workshop on Representation Learning for Natural Language Processing (RepL4NLP)**. Berlin, Germany: Association for Computational Linguistics, 2016. (RepL4NLP@ACL), p. 78–86. Disponível em: <<https://aclanthology.org/W16-1609/>>. Acesso em: 26 maio 2025. Disponível em: <<https://aclanthology.org/W16-1609/>>. Citado na página 80.

MANNING, C. D.; SCHÜTZE, H. **Foundations of Statistical Natural Language Processing**. Cambridge, MA: MIT Press, 1999. ISBN 9780262133609. Disponível em: <<https://mitpress.mit.edu/9780262133609/foundations-of-statistical-natural-language-processing/>>. Citado na página 25.

MEDHAT, W.; HASSAN, A.; KORASHY, H. Sentiment analysis algorithms and applications: A survey. **Ain Shams Engineering Journal**, Elsevier, v. 5, n. 4, p. 1093–1113, 2014. Citado na página 27.

- MIKOLOV, T.; CHEN, K.; CORRADO, G.; DEAN, J. Efficient estimation of word representations in vector space. **arXiv preprint arXiv:1301.3781**, 2013. Disponível em: <<https://arxiv.org/pdf/1301.3781>>. Acesso em: 14 mai. 2025. Citado 2 vezes nas páginas 40 e 80.
- MITCHELL, T. M. **Machine Learning**. New York: McGraw-Hill, 1997. Citado na página 26.
- MONTEIRO, R. A.; SANTOS, R. L. S.; PARDO, T. A. S.; ALMEIDA, T. A. de; RUIZ, E. E. S.; VALE, O. A. **Contributions to the Study of Fake News in Portuguese: New Corpus and Automatic Detection Results**. Springer International Publishing, 2018. 324–334 p. Disponível em: <<https://sites.icmc.usp.br/taspardo/PROPOR2018-MonteiroEtAl.pdf>>. Citado 2 vezes nas páginas 20 e 28.
- _____. Entenda a detecção automática de fake news por similaridade difusa com machine learning & natural language processing (nlp). **Instituto Eldorado**, 2020. Disponível em: <<https://www.eldorado.org.br/blog/entenda-a-deteccao-automatica-de-fake-news-por-similaridade-difusa-com-machine-learning-natural-language-processing-nlp/>>. Citado na página 15.
- MOREIRA, L. S.; LUNARDI, G. M.; RIBEIRO, M. de O.; SILVA, W.; BASSO, F. P. A study of algorithm-based detection of fake news in brazilian election: Is bert the best? **IEEE Latin America Transactions**, v. 21, n. 8, p. 897–903, 2023. Disponível em: <<https://latamt.ieeer9.org/index.php/transactions/article/view/7900>>. Citado na página 22.
- NASCIMENTO, R.; SILVA, A. P. A mecânica da desinformação eleitoral: fake news e o paralelo com as trend topics das redes sociais em 2022. **Estudos em Jornalismo e Mídia**, v. 20, n. 1, p. 75–89, 2023. Acesso em: 15 maio 2025. Disponível em: <<https://www.scielo.br/j/eb/a/pQnFBqBYcKw9XRhPNp3MwSG/>>. Citado na página 14.
- OLIVEIRA, S. S.; VIEIRA, R. M. O impacto das fake news no processo eleitoral brasileiro. **Revista de Ciências Jurídicas e Sociais da UNAERP**, v. 4, n. 1, p. 67–82, 2022. Acesso em: 15 maio 2025. Disponível em: <<https://revistas.unaerp.br/red/article/download/2438/1960/8783>>. Citado na página 14.
- PIRES, V. B.; SILVA, D. G. e. Portuguese fake news classification with bert models. In: **Anais do Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)**. São Paulo, Brasil: Sociedade Brasileira de Computação, 2024. Disponível em: <https://www.researchgate.net/publication/390036739_Portuguese_Fake_News_Classification_with_BERT_models>. Citado na página 23.
- SAAVEDRA, J. D. O. **Comparação de métodos de aprendizado de máquina para classificação automática de notícias em português**. Dissertação (Monografia (MBA em Inteligência Artificial e Big Data)) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2023. Orientador: Rafael Geraldelli Rossi. Citado na página 23.

- SANTOS, L. de F.; SILVA, M. V. da. The effect of stemming and lemmatization on portuguese fake news text classification. **arXiv preprint arXiv:2310.11344**, 2023. Disponível em: <<https://arxiv.org/abs/2310.11344>>. Citado na página 15.
- SEBASTIANI, F. Machine learning in automated text categorization. **ACM Computing Surveys**, ACM, v. 34, n. 1, p. 1–47, 2002. Citado 2 vezes nas páginas 13 e 26.
- SENADO, A. **Desinformação e fake news são entrave no combate à pandemia, aponta debate**. 2021. Disponível em: <<https://www12.senado.leg.br/noticias/materias/2021/07/05/desinformacao-e-fake-news-sao-entrave-no-combate-a-pandemia-aponta-debate>>. Acesso em: 15 mai. 2025. Citado na página 14.
- TEMBHURNE, V.; ALMIN, S. Fake news detection using a logistic regression model and support vector machine. **Research Square**, 2023. Disponível em: <<https://assets-eu.researchsquare.com/files/rs-3156168/v1/f87824c6-4d9e-4f19-a27f-818a851b1de9.pdf>>. Citado na página 60.
- TIGRE, M. F. F. da S. L. **Utilizando Modelos de Machine Learning para Classificar Fake News de COVID-19**. Dissertação (Dissertação de Mestrado Profissional) — Universidade Federal de Pernambuco, Centro de Informática, Recife, 2023. Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação como requisito parcial para obtenção do grau de Mestre Profissional em Ciência da Computação. Orientador: Djamel Fawzi Hadj Sadok. Disponível em: <<https://repositorio.ufpe.br/handle/123456789/55324>>. Citado na página 23.
- VIERING, T.; LOOG, M. The shape of learning curves: a review. **arXiv preprint arXiv:2103.10948**, 2021. Citado na página 17.
- WANG, S.; MANNING, C. D.; SCHÜTZE, H. Comparison of traditional and neural information retrieval models for text classification. In: **Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)**. Melbourne, Australia: Association for Computational Linguistics, 2018. p. 12–19. Disponível em: <<https://aclanthology.org/P18-1002>>. Citado na página 81.
- ZHANG, X.; ZHAO, J.; LECUN, Y. Character-level convolutional networks for text classification. In: CORTES, C.; LAWRENCE, N. D.; LEE, D. D.; SUGIYAMA, M.; GARNETT, R. (Ed.). **Advances in Neural Information Processing Systems (NeurIPS)**. Curran Associates, Inc., 2015. v. 28, p. 649–657. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2015/file/250cf8b51c773f3f8dc8b4be867a9a02-Paper.pdf>. Citado na página 81.

Appendices

APÊNDICE A – Arquivos de Resultados em objetos JSON

A.1 Resultados para o BOW - Arquivo 1

```

1 {
2   "Base 1": {
3     "BOW": {
4       "SVC": {
5         "modelo": "SVC(kernel='linear')",
6         "acuracia": 0.9715277777777778,
7         "precisao": 0.9717325227963525,
8         "revocacao": 0.9715277777777778,
9         "f1_score": 0.9715246880086814,
10        "matriz_confusao": [[707, 13], [28, 692]]
11      },
12      "LogisticRegression": {
13        "modelo": "LogisticRegression(max_iter=200)",
14        "acuracia": 0.9722222222222222,
15        "precisao": 0.9724008305609374,
16        "revocacao": 0.9722222222222222,
17        "f1_score": 0.9722195963738857,
18        "matriz_confusao": [[707, 13], [27, 693]]
19      },
20      "MultinomialNB": {
21        "modelo": "MultinomialNB()",
22        "acuracia": 0.84375,
23        "precisao": 0.8587021504012755,
24        "revocacao": 0.84375,
25        "f1_score": 0.8421045706340818,
26        "matriz_confusao": [[534, 186], [39, 681]]
27      },
28      "RandomForest": {
29        "modelo": "RandomForestClassifier(n_jobs=-1, random_state=42)",
30        "acuracia": 0.9479166666666666,
31        "precisao": 0.9495199893137368,
32        "revocacao": 0.9479166666666666,
33        "f1_score": 0.9478701832411326,
34        "matriz_confusao": [[704, 16], [59, 661]]
35      }
36    }
37  },

```

```
38 "Base 2": {
39   "BOW": {
40     "SVC": {
41       "modelo": "SVC(kernel='linear')",
42       "acuracia": 0.9489662392043967,
43       "precisao": 0.9491349520658895,
44       "revocacao": 0.9489662392043967,
45       "f1_score": 0.9489615828473345,
46       "matriz_confusao": [[1831, 79], [116, 1795]]
47     },
48     "LogisticRegression": {
49       "modelo": "LogisticRegression(max_iter=200)",
50       "acuracia": 0.9617901073017535,
51       "precisao": 0.9619041572546088,
52       "revocacao": 0.9617901073017535,
53       "f1_score": 0.9617878302861192,
54       "matriz_confusao": [[1852, 58], [88, 1823]]
55     },
56     "MultinomialNB": {
57       "modelo": "MultinomialNB()",
58       "acuracia": 0.7330541742999215,
59       "precisao": 0.7350097965057732,
60       "revocacao": 0.7330541742999215,
61       "f1_score": 0.7325026615013086,
62       "matriz_confusao": [[1487, 423], [597, 1314]]
63     },
64     "RandomForest": {
65       "modelo": "RandomForestClassifier(n_jobs=-1, random_state=42)",
66       "acuracia": 0.9502747971735148,
67       "precisao": 0.9506623708504474,
68       "revocacao": 0.9502747971735148,
69       "f1_score": 0.9502643050548032,
70       "matriz_confusao": [[1843, 67], [123, 1788]]
71     }
72   }
73 },
74 "Base 3": {
75   "BOW": {
76     "SVC": {
77       "modelo": "SVC(kernel='linear')",
78       "acuracia": 0.9435750495922416,
79       "precisao": 0.9437739966646347,
80       "revocacao": 0.9435750495922416,
81       "f1_score": 0.9435688648823705,
82       "matriz_confusao": [[2164, 104], [152, 2117]]
83     },
84     "LogisticRegression": {
```

```

85     "modelo": "LogisticRegression(max_iter=200)",
86     "acuracia": 0.9537139078686356,
87     "precisao": 0.9538848593129149,
88     "revocacao": 0.9537139078686356,
89     "f1_score": 0.9537096531278988,
90     "matriz_confusao": [[2185, 83], [127, 2142]]
91 },
92 "MultinomialNB": {
93     "modelo": "MultinomialNB()",
94     "acuracia": 0.7577694511791933,
95     "precisao": 0.7587724630115033,
96     "revocacao": 0.7577694511791933,
97     "f1_score": 0.7575369355096166,
98     "matriz_confusao": [[1789, 479], [620, 1649]]
99 },
100 "RandomForest": {
101     "modelo": "RandomForestClassifier(n_jobs=-1, random_state=42)",
102     "acuracia": 0.936742340753802,
103     "precisao": 0.9375763200536334,
104     "revocacao": 0.936742340753802,
105     "f1_score": 0.93671251169816,
106     "matriz_confusao": [[2174, 94], [193, 2076]]
107 }
108 }
109 },
110 "Base 4": {
111     "BOW": {
112         "SVC": {
113             "modelo": "SVC(kernel='linear')",
114             "acuracia": 0.9484836702954899,
115             "precisao": 0.948576502366637,
116             "revocacao": 0.9484836702954899,
117             "f1_score": 0.9484810048554828,
118             "matriz_confusao": [[2458, 114], [151, 2421]]
119         },
120         "LogisticRegression": {
121             "modelo": "LogisticRegression(max_iter=200)",
122             "acuracia": 0.9574261275272161,
123             "precisao": 0.9574627096953658,
124             "revocacao": 0.9574261275272161,
125             "f1_score": 0.9574252763782233,
126             "matriz_confusao": [[2474, 98], [121, 2451]]
127         },
128         "MultinomialNB": {
129             "modelo": "MultinomialNB()",
130             "acuracia": 0.7743001555209953,
131             "precisao": 0.775103652092228,

```

```
132     "revocacao": 0.7743001555209953,
133     "f1_score": 0.7741352344418785,
134     "matriz_confusao": [[2061, 511], [650, 1922]]
135 },
136 "RandomForest": {
137     "modelo": "RandomForestClassifier(n_jobs=-1, random_state=42)",
138     "acuracia": 0.9350699844479005,
139     "precisao": 0.9359263928867219,
140     "revocacao": 0.9350699844479005,
141     "f1_score": 0.9350380788679865,
142     "matriz_confusao": [[2462, 110], [224, 2348]]
143 }
144 }
145 }
146 }
```

A.2 Resultados para o TFIDF - Arquivo 2

```
1 {
2   "Base 1": {
3     "TFIDF": {
4       "SVC": {
5         "modelo": "SVC(kernel='linear')",
6         "acuracia": 0.9625,
7         "precisao": 0.9628571428571429,
8         "revocacao": 0.9625,
9         "f1_score": 0.9624927648080263,
10        "matriz_confusao": [[703, 17], [37, 683]]
11      },
12      "LogisticRegression": {
13        "modelo": "LogisticRegression(max_iter=200)",
14        "acuracia": 0.9548611111111112,
15        "precisao": 0.9555016603209373,
16        "revocacao": 0.9548611111111112,
17        "f1_score": 0.9548452363895292,
18        "matriz_confusao": [[701, 19], [46, 674]]
19      },
20      "MultinomialNB": {
21        "modelo": "MultinomialNB()",
22        "acuracia": 0.6131944444444445,
23        "precisao": 0.7819107282693813,
24        "revocacao": 0.6131944444444445,
25        "f1_score": 0.545138772711729,
26        "matriz_confusao": [[163, 557], [0, 720]]
27      },
28      "RandomForest": {
```

```
29     "modelo": "RandomForestClassifier(n_jobs=-1, random_state=42)",
30     "acuracia": 0.9604166666666667,
31     "precisao": 0.9611648140598463,
32     "revocacao": 0.9604166666666667,
33     "f1_score": 0.9604006061486163,
34     "matriz_confusao": [[706, 14], [43, 677]]
35 }
36 },
37 "Base 2": {
38     "TFIDF": {
39         "SVC": {
40             "modelo": "SVC(kernel='linear')",
41             "acuracia": 0.9612666841141062,
42             "precisao": 0.9612671789994325,
43             "revocacao": 0.9612666841141062,
44             "f1_score": 0.96126666819634,
45             "matriz_confusao": [[1835, 75], [73, 1838]]
46         },
47         "LogisticRegression": {
48             "modelo": "LogisticRegression(max_iter=200)",
49             "acuracia": 0.9458257000785135,
50             "precisao": 0.9458268216369421,
51             "revocacao": 0.9458257000785135,
52             "f1_score": 0.9458256778151589,
53             "matriz_confusao": [[1808, 102], [105, 1806]]
54         },
55     },
56     "MultinomialNB": {
57         "modelo": "MultinomialNB()",
58         "acuracia": 0.8272703480764197,
59         "precisao": 0.858748680021047,
60         "revocacao": 0.8272703480764197,
61         "f1_score": 0.823388109502465,
62         "matriz_confusao": [[1297, 613], [47, 1864]]
63     },
64     "RandomForest": {
65         "modelo": "RandomForestClassifier(n_jobs=-1, random_state=42)",
66         "acuracia": 0.953415336299398,
67         "precisao": 0.9536786351197938,
68         "revocacao": 0.953415336299398,
69         "f1_score": 0.9534087305898372,
70         "matriz_confusao": [[1844, 66], [112, 1799]]
71     }
72 },
73 "Base 3": {
74     "TFIDF": {
```

```

76     "SVC": {
77         "modelo": "SVC(kernel='linear')",
78         "acuracia": 0.9493057086180295,
79         "precisao": 0.9493228909587484,
80         "revocacao": 0.9493057086180295,
81         "f1_score": 0.9493052603926415,
82         "matriz_confusao": [[2160, 108], [122, 2147]]
83     },
84     "LogisticRegression": {
85         "modelo": "LogisticRegression(max_iter=200)",
86         "acuracia": 0.9336566012783778,
87         "precisao": 0.9337094367833758,
88         "revocacao": 0.9336566012783778,
89         "f1_score": 0.9336546674249216,
90         "matriz_confusao": [[2130, 138], [163, 2106]]
91     },
92     "MultinomialNB": {
93         "modelo": "MultinomialNB()",
94         "acuracia": 0.7987657042098303,
95         "precisao": 0.8355297497646198,
96         "revocacao": 0.7987657042098303,
97         "f1_score": 0.7930886855069174,
98         "matriz_confusao": [[1436, 832], [81, 2188]]
99     },
100    "RandomForest": {
101        "modelo": "RandomForestClassifier(n_jobs=-1, random_state=42)",
102        "acuracia": 0.9385056204540445,
103        "precisao": 0.9393771758031221,
104        "revocacao": 0.9385056204540445,
105        "f1_score": 0.9384754327368215,
106        "matriz_confusao": [[2179, 89], [190, 2079]]
107    }
108 }
109 },
110 "Base 4": {
111     "TFIDF": {
112         "SVC": {
113             "modelo": "SVC(kernel='linear')",
114             "acuracia": 0.9537325038880249,
115             "precisao": 0.9537327782463969,
116             "revocacao": 0.9537325038880249,
117             "f1_score": 0.953732496893888,
118             "matriz_confusao": [[2452, 120], [118, 2454]]
119         },
120         "LogisticRegression": {
121             "modelo": "LogisticRegression(max_iter=200)",
122             "acuracia": 0.9366251944012441,

```

```

123     "precisao": 0.9366515973915399,
124     "revocacao": 0.9366251944012441,
125     "f1_score": 0.9366242363664619,
126     "matriz_confusao": [[2419, 153], [173, 2399]]
127 },
128 "MultinomialNB": {
129     "modelo": "MultinomialNB()",
130     "acuracia": 0.8015163297045101,
131     "precisao": 0.8403717016127753,
132     "revocacao": 0.8015163297045101,
133     "f1_score": 0.7956854106550817,
134     "matriz_confusao": [[1627, 945], [76, 2496]]
135 },
136 "RandomForest": {
137     "modelo": "RandomForestClassifier(n_jobs=-1, random_state=42)",
138     "acuracia": 0.9368195956454122,
139     "precisao": 0.9375212636695018,
140     "revocacao": 0.9368195956454122,
141     "f1_score": 0.9367942543338261,
142     "matriz_confusao": [[2461, 111], [214, 2358]]
143 }
144 }
145 }
146 }

```

A.3 Resultados para o Word2Vec - Arquivo 3

```

1 {
2   "Base 1": {
3     "Word2Vec": {
4       "SVC": {
5         "modelo": "SVC(kernel='linear')",
6         "acuracia": 0.8479166666666667,
7         "precisao": 0.8504320168493371,
8         "revocacao": 0.8479166666666667,
9         "f1_score": 0.8476432680364407,
10        "matriz_confusao": [[580, 140], [79, 641]]
11      },
12      "LogisticRegression": {
13        "modelo": "LogisticRegression(max_iter=500)",
14        "acuracia": 0.8381944444444445,
15        "precisao": 0.839089919173125,
16        "revocacao": 0.8381944444444445,
17        "f1_score": 0.8380875491197651,
18        "matriz_confusao": [[585, 135], [98, 622]]
19      },

```

```
20     "MultinomialNB": {
21         "modelo": "MultinomialNB()",
22         "acuracia": 0.6916666666666667,
23         "precisao": 0.6917613636363636,
24         "revocacao": 0.6916666666666667,
25         "f1_score": 0.6916285961229781,
26         "matriz_confusao": [[490, 230], [214, 506]]
27     },
28     "RandomForest": {
29         "modelo": "RandomForestClassifier(n_jobs=-1, random_state=42)",
30         "acuracia": 0.8722222222222222,
31         "precisao": 0.8723256460127813,
32         "revocacao": 0.8722222222222222,
33         "f1_score": 0.872213348149177,
34         "matriz_confusao": [[634, 86], [98, 622]]
35     }
36 }
37 },
38
39 "Base 2": {
40     "Word2Vec": {
41         "SVC": {
42             "modelo": "SVC(kernel='linear')",
43             "acuracia": 0.862339701648783,
44             "precisao": 0.8647065844832377,
45             "revocacao": 0.862339701648783,
46             "f1_score": 0.8621142661579299,
47             "matriz_confusao": [[1570, 340], [186, 1725]]
48         },
49         "LogisticRegression": {
50             "modelo": "LogisticRegression(max_iter=500)",
51             "acuracia": 0.8594608741167233,
52             "precisao": 0.8604663842156124,
53             "revocacao": 0.8594608741167233,
54             "f1_score": 0.8593616369612788,
55             "matriz_confusao": [[1591, 319], [218, 1693]]
56         },
57         "MultinomialNB": {
58             "modelo": "MultinomialNB()",
59             "acuracia": 0.7320073279246271,
60             "precisao": 0.7324385237227238,
61             "revocacao": 0.7320073279246271,
62             "f1_score": 0.7318853551444592,
63             "matriz_confusao": [[1439, 471], [553, 1358]]
64         },
65         "RandomForest": {
66             "modelo": "RandomForestClassifier(n_jobs=-1, random_state=42)",
```

```
67     "acuracia": 0.8937450929076158,
68     "precisao": 0.8937455535190114,
69     "revocacao": 0.8937450929076158,
70     "f1_score": 0.893745078352189,
71     "matriz_confusao": [[1708, 202], [204, 1707]]
72 }
73 }
74 },
75
76 "Base 3": {
77     "Word2Vec": {
78         "SVC": {
79             "modelo": "SVC(kernel='linear')",
80             "acuracia": 0.84174564690324,
81             "precisao": 0.8424638490765157,
82             "revocacao": 0.84174564690324,
83             "f1_score": 0.8416616480386988,
84             "matriz_confusao": [[1857, 411], [307, 1962]]
85         },
86         "LogisticRegression": {
87             "modelo": "LogisticRegression(max_iter=500)",
88             "acuracia": 0.833810888252149,
89             "precisao": 0.8341463381355291,
90             "revocacao": 0.833810888252149,
91             "f1_score": 0.8337684426389743,
92             "matriz_confusao": [[1855, 413], [341, 1928]]
93         },
94         "MultinomialNB": {
95             "modelo": "MultinomialNB()",
96             "acuracia": 0.7174344280361472,
97             "precisao": 0.7175101395825693,
98             "revocacao": 0.7174344280361472,
99             "f1_score": 0.717410787882105,
100             "matriz_confusao": [[1648, 620], [662, 1607]]
101         },
102         "RandomForest": {
103             "modelo": "RandomForestClassifier(n_jobs=-1, random_state=42)",
104             "acuracia": 0.8840643597090588,
105             "precisao": 0.884075242203606,
106             "revocacao": 0.8840643597090588,
107             "f1_score": 0.8840636162519979,
108             "matriz_confusao": [[2011, 257], [269, 2000]]
109         }
110     }
111 },
112
113 "Base 4": {
```

```
114 "Word2Vec": {
115   "SVC": {
116     "modelo": "SVC(kernel='linear')",
117     "acuracia": 0.8460342146189735,
118     "precisao": 0.8474124821291438,
119     "revocacao": 0.8460342146189735,
120     "f1_score": 0.8458813583404592,
121     "matriz_confusao": [[2095, 477], [315, 2257]]
122   },
123   "LogisticRegression": {
124     "modelo": "LogisticRegression(max_iter=500)",
125     "acuracia": 0.8433125972006221,
126     "precisao": 0.8438323595527393,
127     "revocacao": 0.8433125972006221,
128     "f1_score": 0.8432533597552825,
129     "matriz_confusao": [[2119, 453], [353, 2219]]
130   },
131   "MultinomialNB": {
132     "modelo": "MultinomialNB()",
133     "acuracia": 0.7220062208398134,
134     "precisao": 0.722028909784371,
135     "revocacao": 0.7220062208398134,
136     "f1_score": 0.7219991186716885,
137     "matriz_confusao": [[1870, 702], [728, 1844]]
138   },
139   "RandomForest": {
140     "modelo": "RandomForestClassifier(n_jobs=-1, random_state=42)",
141     "acuracia": 0.8930793157076206,
142     "precisao": 0.8931651380308266,
143     "revocacao": 0.8930793157076206,
144     "f1_score": 0.8930734805753684,
145     "matriz_confusao": [[2278, 294], [256, 2316]]
146   }
147 }
148 }
149 }
```

APÊNDICE B – Código Fonte dos Experimentos

B.1 Definição das bibliotecas utilizadas

```
1 !pip install --upgrade nltk
2 !pip install --upgrade numpy
3 !pip install --upgrade gensim
4 !pip install --upgrade pandas
5 !pip install --upgrade scipy
6 !pip install --upgrade scikit-learn
7 !pip install matplotlib
8 !pip install git-python
9
10 from nltk.tokenize import word_tokenize
11 import scipy
12 from sklearn.feature_extraction.text import CountVectorizer
13 from sklearn.model_selection import train_test_split
14 from sklearn.svm import SVC
15 from sklearn.naive_bayes import MultinomialNB
16 from sklearn.linear_model import LogisticRegression
17 from sklearn.metrics import accuracy_score
18 from sklearn.metrics import confusion_matrix
19 from nltk.corpus import stopwords
20 from nltk.stem import WordNetLemmatizer
21 import matplotlib.pyplot as plt
22 import pandas as pd
23 import seaborn as sns
24 import random
25 import os
26 import nltk
27 import git
28
29
30 nltk.download('punkt')
31 nltk.download('stopwords')
32 nltk.download('wordnet')
33 nltk.download('punkt_tab')
```

B.2 Leitura dos repositórios

```
1 repo_dir_Fakebr = "/content/repo/Fakebr"
2 repo_Fakebr = "https://github.com/roneysco/Fake.br-Corpus.git"
3 diretorio_verdadeiras_Fakebr = "/content/repo/Fakebr/full_texts/true"
4 diretorio_falsas_Fakebr = "/content/repo/Fakebr/full_texts/fake"
5
6 repo_dir_FakeRecogna = "/content/repo/FakeRecogna"
7 repo_FakeRecogna = "https://github.com/Gabriel-Lino-Garcia/FakeRecogna.
  git"
8 file_name_FakeRecogna = 'repo/FakeRecogna/dataset/FakeRecogna.xlsx'
9
10 repo_dir_FakeTrue = "/content/repo/FakeTrue"
11 repo_FakeTrue = "https://github.com/jpchav98/FakeTrue.Br.git"
12 file_name_FakeTrue = '/content/repo/FakeTrue/FakeTrueBr_corpus.csv'
13
14 repo_dir_BoatosBR = "/content/repo/BoatosBR"
15 repo_BoatosBR = "https://github.com/Felipe-Harrison/boatos-br-corpus.
  git"
16
17 if not os.path.exists(repo_dir_Fakebr):
18     git.Repo.clone_from(repo_Fakebr, repo_dir_Fakebr)
19
20 if not os.path.exists(repo_dir_FakeRecogna):
21     git.Repo.clone_from(repo_FakeRecogna, repo_dir_FakeRecogna)
22
23 if not os.path.exists(repo_dir_FakeTrue):
24     git.Repo.clone_from(repo_FakeTrue, repo_dir_FakeTrue)
25
26 if not os.path.exists(repo_dir_BoatosBR):
27     git.Repo.clone_from(repo_BoatosBR, repo_dir_BoatosBR)
```

B.3 Método de pré processamento

```
1 def pre_processamento(texto):
2
3     tokens = word_tokenize(texto)
4
5     tokens = [token.lower() for token in tokens]
6
7     stop_words = set(stopwords.words('portuguese'))
8     tokens = [token for token in tokens if token.isalnum() and token not
  in stop_words]
9
10    lemmatizer = WordNetLemmatizer()
11    tokens = [lemmatizer.lemmatize(token) for token in tokens]
12
```

```
13 return ' '.join(tokens)
```

B.4 Leitura da base FakeBR de seu diretório

```
1 def read_Fakebr():
2     noticias_verdadeiras = []
3     noticias_falsas = []
4
5     for arquivo in os.listdir(diretorio_verdadeiras_Fakebr):
6         with open(os.path.join(diretorio_verdadeiras_Fakebr, arquivo), 'r',
7             encoding='utf-8') as f:
8             texto = f.read()
9             texto_processado = pre_processamento(texto)
10            noticias_verdadeiras.append((texto_processado,1))
11
12    for arquivo in os.listdir(diretorio_falsas_Fakebr):
13        with open(os.path.join(diretorio_falsas_Fakebr, arquivo), 'r',
14            encoding='utf-8') as f:
15            texto = f.read()
16            texto_processado = pre_processamento(texto)
17            noticias_falsas.append((texto_processado,0))
18
19    df_Fakebr_true = pd.DataFrame(noticias_verdadeiras, columns=['FullText', 'Classe'])
20    df_Fakebr_fake = pd.DataFrame(noticias_falsas, columns=['FullText', 'Classe'])
21
22    return df_Fakebr_true, df_Fakebr_fake
```

B.5 Leitura da base FakeRecogna de seu diretório

```
1 def read_FakeRecogna():
2
3     df_FakeRecogna_raw = pd.read_excel(file_name_FakeRecogna)
4     df_FakeRecogna_raw = df_FakeRecogna_raw[['Titulo', 'Subtitulo', 'Noticia', 'Classe']]
5
6     # Convert the columns to strings before joining
7     df_FakeRecogna_raw[['Titulo', 'Subtitulo', 'Noticia']] =
8         df_FakeRecogna_raw[['Titulo', 'Subtitulo', 'Noticia']].astype(str)
9
10    # Replace non-finite values with a suitable integer (e.g., -1)
11    df_FakeRecogna_raw['Classe'] = df_FakeRecogna_raw['Classe'].fillna(-1)
12    .astype(int)
```

```
11
12 df_FakeRecogna = pd.DataFrame({
13     'FullText': df_FakeRecogna_raw[['Titulo', 'Subtitulo', 'Noticia'
14     ]].agg(' '.join, axis=1),
15     'Classe': df_FakeRecogna_raw['Classe']
16 })
17
18 df_FakeRecogna['FullText'] = df_FakeRecogna['FullText'].apply(
19     pre_processamento)
20
21 df_FakeRecogna_true = df_FakeRecogna[df_FakeRecogna['Classe'] == 1].
22     copy()
23 df_FakeRecogna_fake = df_FakeRecogna[df_FakeRecogna['Classe'] == 0].
24     copy()
25
26 return df_FakeRecogna_true, df_FakeRecogna_fake
```

B.6 Leitura da base FakeTrue de seu diretório

```
1 def read_FakeTrue():
2
3     df_FakeTrue_raw = pd.read_csv(file_name_FakeTrue)
4
5     # Convert the columns to strings before joining
6     df_FakeTrue_raw[['title_fake', 'fake', 'true']] = df_FakeTrue_raw[['
7     title_fake', 'fake', 'true']].astype(str)
8
9     df_FakeTrue = pd.DataFrame({
10         'Fake': df_FakeTrue_raw[['title_fake', 'fake']].agg(' '.join, axis
11         =1),
12         'True': df_FakeTrue_raw['true']
13     })
14
15     df_FakeTrue['Fake'] = df_FakeTrue['Fake'].apply(pre_processamento)
16     df_FakeTrue['True'] = df_FakeTrue['True'].apply(pre_processamento)
17
18     df_FakeTrue_true = pd.DataFrame({
19         'FullText': df_FakeTrue['True'],
20         'Classe': 1
21     })
22
23     df_FakeTrue_fake = pd.DataFrame({
24         'FullText': df_FakeTrue['Fake'],
25         'Classe': 0
26     })
27
28     return df_FakeTrue_true, df_FakeTrue_fake
```

B.7 Leitura da base BoatosBR de seu diretório

```
1 def read_BoatosBR():
2     file_name_BoatosBR = "/content/repo/BoatosBR/base_simples/
        boatos_br_corpus_simples.json"
3
4     df_BoatosBR_raw = pd.read_json(file_name_BoatosBR)
5     df_BoatosBR_raw = df_BoatosBR_raw[['texto', 'rotulo']]
6
7     df_BoatosBR_true = df_BoatosBR_raw[df_BoatosBR_raw['rotulo'] == '
        verdade'].copy()
8     df_BoatosBR_fake = df_BoatosBR_raw[df_BoatosBR_raw['rotulo'] == 'falso
        '].copy()
9
10    df_BoatosBR_true['texto'] = df_BoatosBR_true['texto'].apply(
        pre_processamento)
11    df_BoatosBR_fake['texto'] = df_BoatosBR_fake['texto'].apply(
        pre_processamento)
12
13    df_BoatosBR_true = pd.DataFrame({
14        'FullText': df_BoatosBR_true['texto'],
15        'Classe': 1
16    })
17
18    df_BoatosBR_fake = pd.DataFrame({
19        'FullText': df_BoatosBR_fake['texto'],
20        'Classe': 0
21    })
22
23    return df_BoatosBR_true, df_BoatosBR_fake
```

B.8 Método BOW

```
1 def BOW(noticias):
2     lista_noticias = noticias['FullText']
3     lista_rotulos = noticias['Classe']
4
5     vectorizer = CountVectorizer()
6     matriz_bow = vectorizer.fit_transform(lista_noticias)
7
8     return matriz_bow, lista_rotulos
```

B.9 Método TF-IDF

```
1 #TF-IDF (Term Frequency-Inverse Document Frequency)
2 from sklearn.feature_extraction.text import TfidfTransformer
3
4 def TFIDF(noticias):
5
6     # Step 1: Extract the text and labels
7     texts = noticias['FullText'].astype(str).tolist()
8     labels = noticias['Classe'].tolist()
9
10    # Step 2: Convert text to a count matrix
11    vectorizer = CountVectorizer()
12    count_matrix = vectorizer.fit_transform(texts)
13
14    # Step 3: Apply TF-IDF transformation
15    transformer = TfidfTransformer()
16    tfidf_matrix = transformer.fit_transform(count_matrix)
17
18    return tfidf_matrix, labels
```

B.10 Método Word2Vec

```
1 #Word2Vec
2
3 import pandas as pd
4 from gensim.models import Word2Vec
5 from nltk.tokenize import word_tokenize
6 import numpy as np
7 from sklearn.preprocessing import MinMaxScaler
8
9 def function_Word2Vec(noticias):
10
11     vector_size=300
12     window=10
13     min_count=5
14
15     # Step 1: Preprocess text
16     noticias['FullText'] = noticias['FullText'].astype(str).str.lower()
17     tokenized_texts = noticias['FullText'].apply(word_tokenize).tolist()
18
19     # Step 2: Train Word2Vec model
20     w2v_model = Word2Vec(
21
22         sentences=tokenized_texts,
23         vector_size=vector_size,
```

```

24     window=window,
25     min_count=min_count,
26     workers=4
27 )
28
29 # Step 5: Compute document vectors (average of word vectors)
30 def document_vector(doc):
31     vectors = [w2v_model.wv[word] for word in doc if word in
32                w2v_model.wv]
33     if vectors:
34         return np.mean(vectors, axis=0)
35     else:
36         return np.zeros(vector_size)
37
38 doc_vectors = np.array([document_vector(doc) for doc in
39                          tokenized_texts])
40
41 # Apply MinMaxScaler to normalize document vectors
42 scaler = MinMaxScaler()
43 doc_vectors_normalized = scaler.fit_transform(doc_vectors)
44
45 labels = noticias['Classe'].tolist()
46
47 # Return the normalized document vectors along with the labels
48 return doc_vectors_normalized, labels

```

B.11 Método SVC

```

1 from sklearn.svm import SVC
2 from sklearn.metrics import accuracy_score, precision_score,
3   recall_score, f1_score, confusion_matrix
4
5 def function_SVC(x_train, y_train, x_test, y_test):
6     # Cria o modelo SVC com kernel linear
7     model = SVC(kernel='linear', C=1.0)
8
9     # Treina o modelo
10    model.fit(x_train, y_train)
11
12    # Faz previsões
13    y_pred = model.predict(x_test)
14
15    # Calcula métricas de avaliação
16    accuracy = accuracy_score(y_test, y_pred)
17    precision = precision_score(y_test, y_pred, average='weighted',
18                               zero_division=0)

```

```
17     recall = recall_score(y_test, y_pred, average='weighted',
18                           zero_division=0)
19     f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)
20     conf_matrix = confusion_matrix(y_test, y_pred)
21
22     # Retorna os resultados
23     return {
24         'modelo': model,
25         'acuracia': accuracy,
26         'precisao': precision,
27         'revocacao': recall,
28         'f1_score': f1,
29         'matriz_confusao': conf_matrix
30     }
```

B.12 Método Multinomial Naive Bayes

```
1 from sklearn.naive_bayes import MultinomialNB
2 from sklearn.metrics import accuracy_score, precision_score,
3   recall_score, f1_score, confusion_matrix
4
5 def function_MultinomialNB(x_train, y_train, x_test, y_test):
6     # Cria o modelo
7     model = MultinomialNB(alpha=1.0)
8
9     # Treina o modelo
10    model.fit(x_train, y_train)
11
12    # Faz previsoes
13    y_pred = model.predict(x_test)
14
15    # Calcula metricas
16    accuracy = accuracy_score(y_test, y_pred)
17    precision = precision_score(y_test, y_pred, average='weighted',
18                              zero_division=0)
19    recall = recall_score(y_test, y_pred, average='weighted',
20                          zero_division=0)
21    f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)
22    conf_matrix = confusion_matrix(y_test, y_pred)
23
24    # Retorna resultados em um dicionario
25    return {
26        'modelo': model,
27        'acuracia': accuracy,
28        'precisao': precision,
29        'revocacao': recall,
```

```
27     'f1_score': f1,  
28     'matriz_confusao': conf_matrix  
29 }
```

B.13 Método Random Forest

```
1 from sklearn.ensemble import RandomForestClassifier  
2 from sklearn.metrics import accuracy_score, precision_score,  
   recall_score, f1_score, confusion_matrix  
3  
4 def function_RandomForest(x_train, y_train, x_test, y_test):  
5     # Inicializa o classificador Random Forest  
6     modelo = RandomForestClassifier(  
7         n_estimators=100,  
8         max_depth=None,  
9         random_state=42,  
10        n_jobs=-1  
11    )  
12  
13    # Treina o modelo  
14    modelo.fit(x_train, y_train)  
15  
16    # Realiza as previsoes  
17    y_pred = modelo.predict(x_test)  
18  
19    # Calcula metricas  
20    accuracy = accuracy_score(y_test, y_pred)  
21    precision = precision_score(y_test, y_pred, average='weighted',  
                               zero_division=0)  
22    recall = recall_score(y_test, y_pred, average='weighted',  
                          zero_division=0)  
23    f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)  
24    conf_matrix = confusion_matrix(y_test, y_pred)  
25  
26    # Retorna os resultados  
27    return {  
28        'modelo': modelo,  
29        'acuracia': accuracy,  
30        'precisao': precision,  
31        'revocacao': recall,  
32        'f1_score': f1,  
33        'matriz_confusao': conf_matrix  
34    }
```

B.14 Método Regressão Logística

```
1 from sklearn.linear_model import LogisticRegression
2 from sklearn.metrics import accuracy_score, precision_score,
  recall_score, f1_score, confusion_matrix
3
4 def function_LogisticRegression(x_train, y_train, x_test, y_test):
5     # Inicializa o modelo de Regressao Logistica
6     model = LogisticRegression(max_iter=500)
7
8     # Treina o modelo
9     model.fit(x_train, y_train)
10
11     # Realiza previsoes
12     y_pred = model.predict(x_test)
13
14     # Calcula metricas
15     accuracy = accuracy_score(y_test, y_pred)
16     precision = precision_score(y_test, y_pred, average='weighted',
17                                zero_division=0)
18     recall = recall_score(y_test, y_pred, average='weighted',
19                           zero_division=0)
20     f1 = f1_score(y_test, y_pred, average='weighted', zero_division=0)
21     conf_matrix = confusion_matrix(y_test, y_pred)
22
23     # Retorna os resultados
24     return {
25         'modelo': model,
26         'acuracia': accuracy,
27         'precisao': precision,
28         'revocacao': recall,
29         'f1_score': f1,
30         'matriz_confusao': conf_matrix
31     }
```

B.15 Leitura de todas as bases

```
1 df_FakeRecogna_true, df_FakeRecogna_fake = read_FakeRecogna()
2 df_Fakebr_true, df_Fakebr_fake = read_Fakebr()
3 df_FakeTrue_true, df_FakeTrue_fake = read_FakeTrue()
4 df_BoatosBR_true, df_BoatosBR_fake = read_BoatosBR()
5 df_BoatosBR_fake = df_BoatosBR_fake.sample(n=1516, random_state=42).
  reset_index(drop=True)
```

B.16 Metodos auxiliares para execução dos métodos de representação e classificação

```

1 def representation_method(method_name, dataframe):
2     method_map = {
3         'BOW': BOW,
4         'TFIDF': TFIDF,
5         'function_Word2Vec': function_Word2Vec
6     }
7
8     if method_name not in method_map:
9         raise ValueError(f"Unknown method: {method_name}")
10
11     # Call the appropriate function
12     return {
13         'representation_method': method_name,
14         'representation': method_map[method_name](dataframe)
15     }
16
17 from sklearn.model_selection import train_test_split
18
19 def process_classification_methods(representation, dataframe):
20     # Obtem a representacao dos dados
21     result = representation_method(representation, dataframe)
22     x_full, y_full = result["representation"]
23
24     # Separa em treino e teste
25     x_train, x_test, y_train, y_test = train_test_split(
26         x_full,
27         y_full,
28         test_size=0.2,
29         random_state=52,
30         stratify=y_full
31     )
32
33     # Aplica os classificadores e coleta os resultados
34     result_SVC = {
35         'method_representation': representation,
36         'method_classification': 'SVC',
37         'result': function_SVC(x_train, y_train, x_test, y_test)
38     }
39     result_LR = {
40         'method_representation': representation,
41         'method_classification': 'LogisticRegression',
42         'result': function_LogisticRegression(x_train, y_train, x_test,
43         y_test)

```

```
43     }
44     result_MNB = {
45         'method_representation': representation,
46         'method_classification': 'MultinomialNB',
47         'result': function_MultinomialNB(x_train, y_train, x_test,
48                                         y_test)
49     }
50     result_RF = {
51         'method_representation': representation,
52         'method_classification': 'RandomForest',
53         'result': function_RandomForest(x_train, y_train, x_test, y_test
54                                         )
55     }
56
57     return [result_SVC, result_LR, result_MNB, result_RF]
```

B.17 Criação das bases compostas

```
1 data1 = pd.concat([df_Fakebr_true, df_Fakebr_fake], ignore_index=True)
2 data2 = pd.concat([df_FakeRecognna_true, df_FakeRecognna_fake],
3                   ignore_index=True)
4 data3 = pd.concat([df_FakeTrue_true, df_FakeTrue_fake], ignore_index=
5                   True)
6 data4 = pd.concat([df_BoatosBR_true, df_BoatosBR_fake], ignore_index=
7                   True)
8
9 full_data1 = pd.concat([
10     df_Fakebr_true,
11     df_Fakebr_fake
12 ], ignore_index=True)
13
14 full_data2 = pd.concat([
15     df_Fakebr_true,
16     df_Fakebr_fake,
17     df_FakeRecognna_true,
18     df_FakeRecognna_fake
19 ], ignore_index=True)
20
21 full_data3 = pd.concat([
22     df_Fakebr_true,
23     df_Fakebr_fake,
24     df_FakeRecognna_true,
25     df_FakeRecognna_fake,
26     df_FakeTrue_true,
27     df_FakeTrue_fake
28 ], ignore_index=True)
```

```
26
27 full_data4 = pd.concat([
28     df_Fakebr_true,
29     df_Fakebr_fake,
30     df_FakeRecogna_true,
31     df_FakeRecogna_fake,
32     df_FakeTrue_true,
33     df_FakeTrue_fake,
34     df_BoatosBR_true,
35     df_BoatosBR_fake
36 ], ignore_index=True)
```

B.18 Execução dos testes para o método de representação BOW

```
1 #Testes com representacao BOW e Base 1
2
3 results = []
4 representation_name = 'BOW'
5 model_names = ['SVC', 'Logistic Regression', 'Multinomial NB', 'Random
6     Forest']
7 result_fulldata1 = process_classification_methods(representation_name,
8     full_data1)
9 results.append(result_fulldata1)
10 print(result_fulldata1)
11
12 #Testes com representacao BOW e Base 2
13
14 results = []
15 representation_name = 'BOW'
16 model_names = ['SVC', 'Logistic Regression', 'Multinomial NB', 'Random
17     Forest']
18 result_fulldata2 = process_classification_methods(representation_name,
19     full_data2)
20 results.append(result_fulldata2)
21 print(result_fulldata2)
22
23 #Testes com representacao BOW e Base 3
24
25 results = []
26 representation_name = 'BOW'
27 model_names = ['SVC', 'Logistic Regression', 'Multinomial NB', 'Random
28     Forest']
29 result_fulldata3 = process_classification_methods(representation_name,
30     full_data3)
31 results.append(result_fulldata3)
32 print(result_fulldata3)
```

```
27
28 #Testes com representacao BOW e Base 4
29
30 results = []
31 representation_name = 'BOW'
32 model_names = ['SVC', 'Logistic Regression', 'Multinomial NB', 'Random
    Forest']
33 result_fulldata4 = process_classification_methods(representation_name,
    full_data4)
34 results.append(result_fulldata4)
35 print(result_fulldata4)
```

B.19 Execução dos testes para o método de representação TF-IDF

```
1 #Testes com representacao TF-IDF e Base 1
2
3 results = []
4 representation_name = 'TFIDF'
5 model_names = ['SVC', 'Logistic Regression', 'Multinomial NB', 'Random
    Forest']
6 result_fulldata1 = process_classification_methods(representation_name,
    full_data1)
7 results.append(result_fulldata1)
8 print(result_fulldata1)
9
10 #Testes com representacao TFIDF e Base 2
11
12 results = []
13 representation_name = 'TFIDF'
14 model_names = ['SVC', 'Logistic Regression', 'Multinomial NB', 'Random
    Forest']
15 result_fulldata2 = process_classification_methods(representation_name,
    full_data2)
16 results.append(result_fulldata2)
17 print(result_fulldata2)
18
19 #Testes com representacao TFIDF e Base 3
20
21 results = []
22 representation_name = 'TFIDF'
23 model_names = ['SVC', 'Logistic Regression', 'Multinomial NB', 'Random
    Forest']
24 result_fulldata3 = process_classification_methods(representation_name,
    full_data3)
25 results.append(result_fulldata3)
26 print(result_fulldata3)
```

```
27
28 #Testes com representacao TFIDF e Base 4
29
30 results = []
31 representation_name = 'TFIDF'
32 model_names = ['SVC', 'Logistic Regression', 'Multinomial NB', 'Random
    Forest']
33 result_fulldata4 = process_classification_methods(representation_name,
    full_data4)
34 results.append(result_fulldata4)
35 print(result_fulldata4)
```

B.20 Execução dos testes para o método de representação Word2Vec

```
1 #Testes com representacao Word2Vec e Base 1
2
3 results = []
4 representation_name = 'function_Word2Vec'
5 model_names = ['SVC', 'Logistic Regression', 'Multinomial NB', 'Random
    Forest']
6 result_fulldata1 = process_classification_methods(representation_name,
    full_data1)
7 results.append(result_fulldata1)
8 print(result_fulldata1)
9
10 #Testes com representacao Word2Vec e Base 2
11
12 results = []
13 representation_name = 'function_Word2Vec'
14 model_names = ['SVC', 'Logistic Regression', 'Multinomial NB', 'Random
    Forest']
15 result_fulldata2 = process_classification_methods(representation_name,
    full_data2)
16 results.append(result_fulldata2)
17 print(result_fulldata2)
18
19 #Testes com representacao Word2Vec e Base 3
20
21 results = []
22 representation_name = 'function_Word2Vec'
23 model_names = ['SVC', 'Logistic Regression', 'Multinomial NB', 'Random
    Forest']
24 result_fulldata3 = process_classification_methods(representation_name,
    full_data3)
25 results.append(result_fulldata3)
26 print(result_fulldata3)
```

```

27
28 #Testes com representacao Word2Vec e Base 4
29
30 results = []
31 representation_name = 'function_Word2Vec'
32 model_names = ['SVC', 'Logistic Regression', 'Multinomial NB', 'Random
    Forest']
33 result_fulldata4 = process_classification_methods(representation_name,
    full_data4)
34 results.append(result_fulldata4)
35 print(result_fulldata4)

```

B.21 Geração dos gráficos heatmap

```

1 import pandas as pd
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
5 def plot_heatmap_vertical_por_metrica(results_bow, results_tfidf,
    results_word2vec, metrica="f1_score"):
6     modelos = ["SVC", "LogisticRegression", "MultinomialNB", "
    RandomForest"]
7     bases = ["Base 1", "Base 2", "Base 3", "Base 4"]
8
9     def extract(source, rep_key, rep_label):
10         rows = []
11         for base in bases:
12             base_data = source.get(base, {}).get(rep_key, {})
13             for model in modelos:
14                 valor = base_data.get(model, {}).get(metrica, 0)
15                 row_id = f"{model}_{rep_label}"
16                 rows.append({
17                     "Modelo_Representacao": row_id,
18                     "Base": base,
19                     "Valor": valor
20                 })
21         return pd.DataFrame(rows)
22
23     # Unir os dados
24     df_bow = extract(results_bow, "BOW", "BoW")
25     df_tfidf = extract(results_tfidf, "TFIDF", "TFIDF")
26     df_w2v = extract(results_word2vec, "Word2Vec", "Word2Vec")
27     df_all = pd.concat([df_bow, df_tfidf, df_w2v])
28
29     # Pivotar: linhas = modelos + representa o, colunas = base

```

```

30     df_pivot = df_all.pivot(index="Modelo_Representacao", columns="Base"
31                               , values="Valor")
32
33     # Plot
34     plt.figure(figsize=(8, 6))
35     sns.heatmap(df_pivot, annot=True, fmt=".3f", cmap="YlGnBu", cbar_kws
36                 ={"label": metrica.capitalize()})
37     plt.title(f"Heatmap de {metrica.capitalize()} por Representa o e
38               Base")
39     plt.xlabel("Base de Dados")
40     plt.ylabel("Modelo + Representa o")
41     plt.tight_layout()
42     plt.show()
43
44 plot_heatmap_vertical_por_metrice(results_bow, results_tfidf,
45                                   results_word2vec, metrica="f1_score")
46 plot_heatmap_vertical_por_metrice(results_bow, results_tfidf,
47                                   results_word2vec, metrica="acuracia")
48 plot_heatmap_vertical_por_metrice(results_bow, results_tfidf,
49                                   results_word2vec, metrica="precisao")
50 plot_heatmap_vertical_por_metrice(results_bow, results_tfidf,
51                                   results_word2vec, metrica="revocacao")

```

B.22 Geração dos gráficos radar

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 def plot_radar_f1score_per_base(results_bow, results_tfidf,
5                                results_word2vec):
6     modelos = ["SVC", "LogisticRegression", "MultinomialNB", "
7               RandomForest"]
8     representacoes = {
9         "BoW": (results_bow, "BOW"),
10        "TF-IDF": (results_tfidf, "TFIDF"),
11        "Word2Vec": (results_word2vec, "Word2Vec")
12    }
13    bases = ["Base 1", "Base 2", "Base 3", "Base 4"]
14
15    for base in bases:
16        f1_data = {rep_name: [] for rep_name in representacoes}
17
18        for rep_name, (resultados, rep_key) in representacoes.items():
19            base_data = resultados.get(base, {}).get(rep_key, {})
20            for modelo in modelos:
21                score = base_data.get(modelo, {}).get("f1_score", 0)

```

```
20         f1_data[rep_name].append(score)
21
22     # Prepara o dos dados
23     angles = np.linspace(0, 2 * np.pi, len(modelos), endpoint=False)
24         .tolist()
25     angles += angles[:1] # Fecha o círculo
26
27     fig, ax = plt.subplots(figsize=(7, 6), subplot_kw=dict(polar=
28         True))
29
30     for rep_name, valores in f1_data.items():
31         valores += valores[:1] # Fecha o polígono
32         ax.plot(angles, valores, label=rep_name)
33         ax.fill(angles, valores, alpha=0.1)
34
35     ax.set_xticks(angles[:-1])
36     ax.set_xticklabels(modelos)
37     ax.set_yticks([0.5, 0.6, 0.7, 0.8, 0.9, 1.0])
38     ax.set_yticklabels(["0.5", "0.6", "0.7", "0.8", "0.9", "1.0"])
39     ax.set_ylim(0.5, 1.05)
40     ax.set_title(f'Radial Plot - {base}', size=14, pad=20)
41     ax.legend(loc="upper right", bbox_to_anchor=(1.2, 1.1))
42     plt.tight_layout()
43     plt.show()
44
45 plot_radar_f1score_per_base(results_bow, results_tfidf, results_word2vec
46     )
```