Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Classificação de petições iniciais no Conselho Nacional do Ministério Público

Paulo Célio Soares da Silva Júnior

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Orientador
Prof. Dr. Thiago de Paulo Faleiros

Brasília
2024

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

d111c

da Silva Júnior, Paulo Célio Soares
Classificação de petições iniciais no Conselho Nacional do Ministério Público / Paulo Célio Soares da Silva Júnior; orientador Thiago de Paulo Faleiros. -- Brasília, 2024.
98 p.

Dissertação(Mestrado Profissional em Computação Aplicada) -- Universidade de Brasília, 2024.

1. Classificação de Texto. 2. Petição Inicial. 3. Processamento de Linguagem Natural. 4. BERT. 5. Inteligência Artificial. I. Faleiros, Thiago de Paulo, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Classificação de petições iniciais no Conselho Nacional do Ministério Público

Paulo Célio Soares da Silva Júnior

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof. Dr. Thiago de Paulo Faleiros (Orientador)
CIC/UnB

Prof. Dr. Guilherme Souza Rodrigues
EST/UnB

Prof.a Dr.a Lilian Berton
ICT/Unifesp

Prof.a Dr.a Edna Dias Canedo
Coordenadora do Programa de Pós-graduação em Computação Aplicada

Brasília, 20 de dezembro de 2024

Dedicatória

Dedico esta dissertação, em primeiro lugar, aos meus filhos, que são tudo para mim e a razão de cada esforço que empreendi nesta jornada. Vocês trouxeram luz e esperança para os dias mais difíceis, com os gestos de carinho mais puros, que ajudaram, por inúmeras vezes, a dissipar qualquer sombra de cansaço. Saibam que o amor e o orgulho que sinto por vocês foram e sempre serão o combustível que me motiva a buscar o melhor.

À minha amada namorada, a quem devo uma gratidão eterna. Você não apenas caminhou ao meu lado, mas segurou minhas mãos com firmeza quando eu sentia que poderia fraquejar. Sua paciência, compreensão e incentivo foram fundamentais para que eu seguisse adiante, mesmo quando as noites se estendiam em intermináveis horas de estudo e trabalho. O seu apoio incondicional me deu forças para enfrentar cada desafio, e o seu amor foi o abrigo seguro que me acolheu nos momentos de maior vulnerabilidade. Obrigado por acreditar em mim, mesmo quando eu duvidava de mim mesmo.

À minha família, que sempre esteve comigo, mesmo quando a distância física parecia imensa. Queridos mãe e irmãos, saibam que sou profundamente grato por toda a paciência e cada oração que vocês dedicaram ao meu sucesso. Mesmo durante os dias em que estive ausente, a lembrança de tudo o que vocês significam para mim me impulsionou a continuar. Este trabalho é um testemunho da força que me deram, da resiliência que aprendemos a ter diante de todos os desafios que enfrentamos juntos e da esperança de que todos os sacrifícios valem a pena.

Sei que o tempo que passei distante deixou lacunas em muitos momentos importantes, e é com o coração pesado que expresso meu sincero pedido de desculpas. As longas horas de ausência, os finais de semana dedicados às leituras e à escrita, as noites sem fim... tudo isso foi um caminho que trilhamos juntos, mesmo quando eu não estava fisicamente ao lado de vocês. Peço que aceitem esta conquista como um tributo ao amor e à paciência que vocês me deram.

A todos vocês, dedico cada palavra, cada página e cada momento desta realização. Esta vitória é nossa!

Agradecimentos

A realização deste trabalho foi possível graças à colaboração, apoio e dedicação de muitas pessoas e instituições que, de diferentes maneiras, contribuíram para que esta jornada pudesse ser concluída com sucesso.

Em primeiro lugar, agradeço ao Conselho Nacional do Ministério Público, minha casa, pela oportunidade e pelo suporte que tornaram possível a condução desta pesquisa. A confiança e os recursos investidos foram fundamentais para a concretização deste projeto, permitindo que eu pudesse me aprofundar nas pesquisas que aqui apresento.

Aos meus colegas de mestrado, com quem compartilhei incontáveis desafios e vitórias, expresso um agradecimento especial. A caminhada foi árdua, marcada por noites sem dormir, dúvidas incessantes e uma carga de trabalho que muitas vezes parecia insuperável. No entanto, juntos, encontramos forças para seguir adiante. O apoio mútuo, as discussões acadêmicas, o companheirismo e, acima de tudo, as amizades que se formaram ao longo deste caminho foram essenciais para tornar cada obstáculo mais leve e cada conquista ainda mais significativa. Foi uma honra trilhar este caminho ao lado de vocês.

Ao meu orientador, Professor Thiago, expresso minha mais profunda gratidão. Sua paciência e compromisso foram imprescindíveis ao longo de todo o processo. Suas críticas construtivas e conselhos muito bem-vindos me desafiaram a pensar de forma crítica e sempre buscar o melhor possível. Obrigado por acreditar no meu potencial e por dedicar seu tempo e conhecimento para me guiar até este ponto.

À Universidade de Brasília (UnB), agradeço por me ensinar o que é o ambiente acadêmico. A UnB não só me acolheu, mas também me ofereceu as condições para o desenvolvimento do meu trabalho. Sinto-me privilegiado por fazer parte de uma instituição tão respeitada, que fomenta a excelência e promove o crescimento intelectual e profissional de seus alunos.

Por fim, agradeço a todos aqueles que, de alguma forma, fizeram parte desta jornada, seja com uma palavra de incentivo, um gesto de carinho ou uma presença silenciosa e constante. Este trabalho é fruto de um esforço coletivo, e sou profundamente grato a cada um que, direta ou indiretamente, contribuiu para esta realização.

Resumo

Este trabalho propõe a aplicação de técnicas de Processamento de Linguagem Natural, utilizando modelos de linguagem baseados em BERT, para melhorar a classificação de petições iniciais do Conselho Nacional do Ministério Público (CNMP). Os modelos BERTimbau e Albertina PT-BR demonstraram desempenho superior aos algoritmos tradicionais de aprendizado de máquina, evidenciando a eficácia do ajuste fino desses modelos para a tarefa de classificação conforme as classes processuais definidas no artigo 37 do Regimento Interno do CNMP. A pesquisa integrou estratégias de pré-processamento, como digitalização e limpeza textual, além de técnicas de sumarização abstrativa com RAG e LLM, que contribuíram significativamente para o desempenho dos classificadores. Também foi explorada a técnica de *Data Augmentation* para balanceamento de dados, a qual mostrou impacto positivo nos classificadores tradicionais, especialmente no SVM combinado com vetorização por *embeddings*, e aprimorou o desempenho do modelo Albertina PT-BR. Os resultados indicam que o ajuste fino de modelos BERT é uma alternativa eficaz para a classificação de petições iniciais, superando abordagens tradicionais de aprendizado de máquina. A pesquisa demonstrou eficácia e inovação na classificação de textos no contexto do Ministério Público brasileiro.

Palavras-chave: petição inicial, Processamento de Linguagem Natural, classificação de texto, aprendizado de máquina, aprendizado profundo, BERT, RAG, LLM, *Data Augmentation*

Abstract

This work proposes the application of Natural Language Processing techniques, using BERT-based language models, to improve the classification of initial petitions by the National Council of the Public Ministry (CNMP). The BERTimbau and Albertina-PTBR models demonstrated superior performance compared to traditional machine learning algorithms, highlighting the effectiveness of fine-tuning these models for the classification task according to the procedural classes defined in Article 37 of the CNMP's Internal Regulations. The research integrated preprocessing strategies, such as digitization and text cleaning, as well as abstractive summarization techniques with RAG and LLM, which significantly contributed to the classifiers' performance. The Data Augmentation technique for data balancing was also explored, showing a positive impact on traditional classifiers, especially SVM combined with embeddings vectorization, and improved the performance of the Albertina PT-BR model. The results indicate that fine-tuning BERT models is an effective alternative for classifying initial petitions, surpassing traditional machine learning approaches. The research demonstrated effectiveness and innovation in text classification within the context of the Brazilian Public Ministry.

Keywords: initial petition, Natural Language Processing, text classification, machine learning, deep learning, BERT, RAG, LLM, Data Augmentation

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API *Application Programming Interface.*

BERT *Bidirectional Encoder Representations for Transformers.*

BiLSTM *Bidirecional Long Short-Term Memory.*

BOW *Bag-of-Words.*

CADE Conselho de Administrativo de Defesa Econômica.

CNJ Conselho Nacional de Justiça.

CNMP Conselho Nacional do Ministério Público.

CNN Rede Neural Convolucional.

COPAD Coordenadoria de Protocolo, Autuação e Distribuição.

CPC Código de Processo Civil.

GPT *Generative Pre-trained Transformer.*

k-NN *k-Nearest Neighbors.*

LLM *Large Language Model.*

LSTM *Long Short-Term Memory.*

MLP *Multilayer Perceptron.*

MP Ministério Público.

MPDFT Ministério Público Distrito Federal e Territórios.

MPE Ministério Público dos Estados.

MPF Ministério Público Federal.

MPM Ministério Público Militar.

MPT Ministério Público do Trabalho.

MPU Ministério Público da União.

NB *Naïve Bayes*.

OCR *Optical Character Recognition*.

PART *Recursive Partitioning and Regression Trees*.

PDF *Portable Document Format*.

PLN Processamento de Linguagem Natural.

POS *Part of Speech*.

RAG *Retrieval-Augmented Generation*.

RICNMP Regimento Interno do CNMP.

RNN Rede Neural Recorrente.

SG Secretaria-Geral.

SHA *Secure Hash Algorithm*.

SPR Secretaria Processual.

SVC *Support Vector Classification*.

SVM *Support Vector Machine*.

TF-IDF *Term Frequency-Inverse Document Frequency*.

ULMFiT *Universal Language Model Fine-tuning*.

Capítulo 1

Introdução

1.1 Contextualização

O Conselho Nacional do Ministério Público (CNMP) atua em prol do cidadão, executando a fiscalização administrativa, financeira e disciplinar de ramos e unidades do Ministério Público (MP) do Brasil, bem como de seus membros, respeitando a autonomia da instituição. Criado em 30 de dezembro de 2.004 pela Emenda Constitucional nº 45, teve sua instalação concluída em 21 de junho de 2.005. A sede fica em Brasília-DF.

O órgão tem como alguns de seus principais objetivos fiscalizar e imprimir uma visão nacional ao MP, cabendo-lhe orientar todos os ramos e unidades do MP brasileiro, que possui a seguinte formação (Figura 1.1): Ministério Público da União (MPU), composto por Ministério Público Federal (MPF), Ministério Público Militar (MPM), Ministério Público do Trabalho (MPT) e Ministério Público Distrito Federal e Territórios (MPDFT); e Ministério Público dos Estados (MPE).

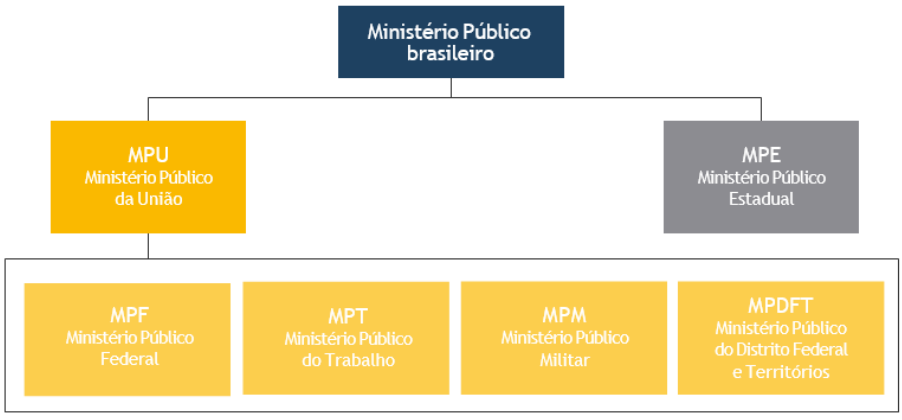

Figura 1.1: Estrutura do Ministério Público brasileiro (Fonte: [1]).

Composto por 14 membros, que representam setores diversos da sociedade, ele é presidido pelo Procurador-Geral da República. Somados a ele, também compõem o CNMP: quatro integrantes do MPU; três membros do MPE; dois juízes indicados, um pelo Supremo Tribunal Federal e outro pelo Superior Tribunal de Justiça; dois advogados indicados pelo Conselho Federal da Ordem dos Advogados do Brasil; e dois cidadãos de notável saber jurídico e reputação ilibada indicados, um pela Câmara dos Deputados e outro pelo Senado Federal, nos termos do artigo 130-A da Constituição Federal [2].

Para que seja possível o pleno desempenho de suas atividades finalísticas, o CNMP conta com a organização administrativa ilustrada na Figura 1.2, na forma estabelecida pelo artigo 3º do Regimento Interno do CNMP (RICNMP) [3] e no artigo 1º da Resolução CNMP 146/2.016 [4]:

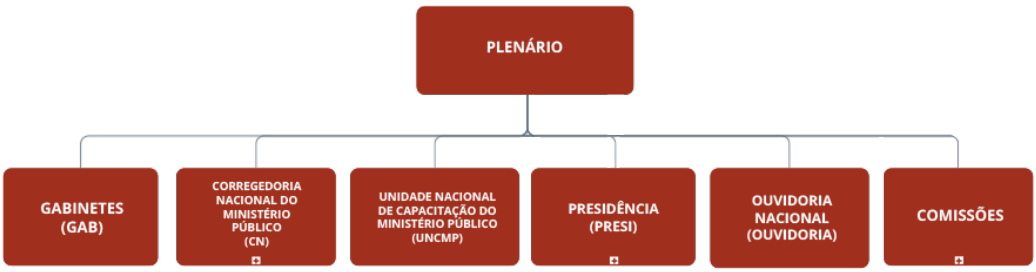

Figura 1.2: Organograma do CNMP (Fonte: [5]).

Essa estrutura está organizada a fim de efetivar o cumprimento da fiscalização e da orientação do exercício administrativo e financeiro do Ministério Público no Brasil, além de promover sua integração e seu desenvolvimento.

Para realização de boa parte de sua atividade finalística, o CNMP dá origem a processos que funcionam como instrumentos formais onde, no decorrer de sua instrução, são realizadas e registradas as atividades necessárias a fim de dar efeito àquilo em que caiba a atuação do Conselho.

Comumente, de acordo com o RICNMP [3], os processos são iniciados com petições (artigo 36). Elas são recebidas e protocolizadas e, caso cumpram todos os requisitos regimentais estabelecidos, registradas e autuadas em sistema eletrônico de acompanhamento processual, tornando-se então processos e ganhando numeração contínua e seriada, obedecendo às classes processuais[1] descritas na Tabela 1.1 (artigo 37).

Após sua autuação, cada processo é distribuído imediatamente pela Secretaria-Geral (SG) do CNMP, de acordo com a ordem de autuação, por meio de sorteio eletrônico em

[1]A classe processual Ordem do Mérito não consta no rol de classes do artigo 37 do RICNMP, tendo sido definida no artigo 31 da Resolução CNMP 252/2.022 [6].

Tabela 1.1: Classes Processuais do CNMP (Fonte: [3]).

Inciso	**Sigla**	**Nome**
I	INSP	Inspeção
II	COR	Correição
III	RD	Reclamação Disciplinar
IV	SIND	Sindicância
V	RIEP	Representação por Inércia ou Excesso de Prazo
VI	PAD	Processo Administrativo Disciplinar
VII	AVOC	Avocação
VIII	RPD	Revisão de Processo Disciplinar
IX	RPA	Reclamação para Preservação da Autonomia do Ministério Público
X	RCA	Reclamação para Preservação da Competência e da Autoridade das Decisões do Conselho
XI	PCA	Procedimento de Controle Administrativo
XII	ASI	Arguição de Impedimento ou Suspeição
XIII	RA	Restauração de Autos
XIV	PP	Pedido de Providências
XV	RIP	Remoção por Interesse Público
XVI	PROP	Proposição
XVII	RDC	Revisão de Decisão do Conselho
XVIII	PAVOC	Procedimento Avocado
XIX	CONS	Consulta
XXI	PIC	Procedimento Interno de Comissão
XXII	NT	Nota Técnica
XXIII	AL	Anteprojeto de Lei
XXIV	NF	Notícia de Fato
XXV	CA	Conflito de Atribuições
-	OM	Ordem do Mérito

sessão pública, em cada classe de processo, entre todos os Conselheiros, com exclusão do Presidente do Conselho e do Corregedor Nacional (artigo 38 do RICNMP [3]). Embora essa seja a forma padrão, há ainda outras formas de distribuição previstas regimentalmente.

De acordo com o artigo 43 do RICNMP [3], uma vez distribuído a um Conselheiro, este se torna Relator do processo, sendo agora o responsável pela instrução processual (inciso I), pela concessão de medidas liminares (inciso VIII), pela decisão quanto a pedido de sigilo (inciso XI), além de realizar outras atividades previstas regimentalmente a fim de dar bom andamento ao feito.

1.1.1 Petição Inicial

A petição inicial é o ato processual praticado pelo autor, o qual implica a quebra da inércia do Juiz, com a finalidade de dar início ao processo. Conforme diz o artigo 2º do Código de Processo Civil (CPC) [7], o processo somente começa por iniciativa da parte e, então, se desenvolve por impulso oficial.

O documento da petição inicial deve ser puramente técnico e redigido de acordo com os ditames legais do artigo 319 do CPC [8]. Este contém os requisitos exigidos para que ela seja considerada válida e apta a iniciar o processo judicial. Aplica-se subsidiariamente esse entendimento aos processos iniciados no CNMP.

A petição deve conter a menção aos elementos da ação, que são as partes envolvidas, o pedido e a fundamentação do pedido. Esses elementos são essenciais para delimitar o objeto da demanda e fornecer as informações necessárias para o Juízo entender o que está sendo pleiteado.

Diante disso, a petição inicial é o ato processual, oriundo dos princípios e direitos constitucionais, que movimenta o direito de ação, instaurando o processo civil. Para que seja deferida, é exigido ao procurador que redija sua petição inicial cumprindo as exigências requeridas pelo direito processualista civil [9].

1.2 Descrição do Problema

Dentro da SG, incumbe à Secretaria Processual (SPR) a tarefa de autuação e distribuição de processos, por meio de sua Coordenadoria de Protocolo, Autuação e Distribuição (COPAD).

Atualmente, com dois servidores encarregados pelo protocolo e três pela autuação e distribuição, o setor recebe as petições iniciais em documento texto e encarrega-se de providenciar os trâmites iniciais para que aquele documento possa se transformar em um processo caso atenda aos requisitos do RICNMP [3] e esteja dentro das competências de atuação do CNMP.

A Tabela 1.2 ilustra o número de processos autuados ano a ano, para todas as classes processuais definidas, desde o início do órgão, com data de corte em 30/06/2.023.

Considerando-se que nem todas as petições iniciais necessariamente são atuadas e viram processos, o número de documentos a serem analisados é ainda maior do que o ilustrado. Para cada um desses documentos, a equipe da COPAD/SPR necessita fazer sua leitura para que consiga avaliar sua aceitabilidade e, em caso positivo, extrair os atributos necessários do texto para a autuação do processo, como classe processual, assunto, objeto, pedido de liminar, pedido de sigilo e partes interessadas.

Tabela 1.2: Autuações de processos no CNMP.

Ano	**Quantidade**
2.005	144
2.006	736
2.007	1.058
2.008	1.115
2.009	1.536
2.010	1.988
2.011	1.790
2.012	1.569
2.013	1.816
2.014	1.807
2.015	1.353
2.016	1.593
2.017	1.369
2.018	1.260
2.019	1.121
2.020	1.085
2.021	1.472
2.022	1.314
2.023	1.160
2.024[2]	1.307

Além disso, cada um desses documentos pode conter mais de uma página e, em alguns casos, estar em formato físico, necessitando ser digitalizados, o que amplia ainda mais o esforço necessário para a realização do trabalho face à exiguidade de recursos humanos destinados a esse propósito, no caso, os três servidores públicos responsáveis pela autuação e distribuição.

Apesar das melhorias realizadas ao longo dos anos, o fluxo de trabalho ainda é lento e exige a alocação de recursos humanos escassos para ser viabilizado. Assim, a leitura e extração de informação de grandes volumes de documentos por um número limitado de pessoas torna-se uma tarefa bastante onerosa no contexto de atuação do setor responsável, especialmente em atividades como a classificação processual.

Ao problema negocial, soma-se o problema da qualidade dos dados processuais da instituição. Embora o processo eletrônico tenha evoluído ao longo dos anos, muitos dos documentos, incluindo petições iniciais, não possuem formato definido, nem critério de avaliação de qualidade. E, durante o processo de digitalização, ainda se perde muita da informação textual.

Por fim, quando as petições são recebidas e autuadas em processos CNMP, há ainda o problema de desbalanceamento dos dados. Desde a instituição do processo eletrônico, a

grande maioria dos processos se concentra em cerca de cinco classes, embora o órgão tenha regulamentado vinte e cinco delas. Assim sendo, qualquer tipo de problema a ser resolvido utilizando-se os dados provenientes precisa considerar o balanceamento inadequado.

1.3 Hipótese

Com o intuito de abordar o problema descrito, a principal hipótese de pesquisa é que, a partir de um modelo de linguagem baseado na técnica de Processamento de Linguagem Natural (PLN) chamada *Bidirectional Encoder Representations for Transformers* (BERT), realizando-se ajuste fino para a tarefa de classificação de texto das petições iniciais conforme as classes processuais definidas no artigo 37 do RICNMP [3], combinando estratégias de pré-processamento, seleção de características e sumarização de texto, seja possível obter acurácia superior em relação a algoritmos tradicionais de aprendizado de máquina.

1.4 Objetivos

O objetivo geral desse trabalho é desenvolver um sistema de classificação supervisionada de texto que, a partir de documentos de texto de petições iniciais, seja capaz de atribuir classes processuais CNMP com base em algoritmos de aprendizado de máquina e técnicas de Processamento de Linguagem Natural, sendo possível comparar seus desempenhos.

Como objetivos específicos, podem-se citar:

1. implementar processo de pré-processamento de texto, incluindo digitalização adequada e limpeza das informações textuais obtidas;

2. comparar técnicas para redução do tamanho máximo de sequência para uso nos modelos de linguagem, incluindo técnica de sumarização prévia de texto;

3. tratar o desbalanceamento dos dados, propondo técnica que permita utilização de algoritmo classificador de forma adequada;

4. implementar e avaliar diferentes algoritmos de aprendizado de máquina para a tarefa de classificação de texto, explorando algoritmos tradicionais e *fine-tuning* de modelos de linguagem em língua portuguesa baseados em BERT para a tarefa.

Capítulo 2

Fundamentação Teórica

Esta seção propõe-se a apresentar os conceitos necessários para que o problema estudado seja adequadamente compreendido e que os resultados obtidos possam ser corretamente avaliados. As informações aqui apresentadas fornecem uma fundação para que, além da compreensão do assunto, as contribuições apresentadas possam ser criticamente avaliadas.

2.1 Processamento de Linguagem Natural

Processamento de Linguagem Natural (PLN) [10] é um campo de pesquisa que tem como objetivo investigar e propor métodos e sistemas de processamento computacional da linguagem humana. Ela permite que a linguagem seja interpretada e transformada em uma forma que seja possível a utilização pelos computadores.

Diversas são as aplicações de PLN encontradas no mundo cotidiano: classificação de spam ou outras categorias em caixas de e-mails, interação com assistentes virtuais, mecanismos de busca, sugestão de próxima palavra ou análise ortográfica durante a redação de textos, tradução ou sumarização de textos, análise de sentimentos em redes sociais, atendimento automatizado com robôs de conversação (*chatbots*), etc.

Considerando-se a diversidade de usos possíveis, o PLN atualmente é uma das áreas mais estudadas dentro da ciência de dados, dado que ela possibilita a compreensão e a extração de dados não estruturados contidos em textos.

2.1.1 Tarefas em PLN

Muitas são as aplicações de Processamento de Linguagem Natural no dia a dia, como já visto. A partir desses casos de uso concretos, podem ser abstraídas algumas das tarefas mais comumente realizadas nessa área. Como exemplos mais comuns [11, 12, 13], podem-se citar:

- Classificação de Texto: como será visto mais à frente, o objetivo desta tarefa é classificar ou categorizar textos a partir de conjunto de rótulos ou categorias pré-definidos;
- Modelagem de Linguagem: esta tarefa utiliza-se de técnicas estatísticas para a predição de palavras em sentenças, baseada na sequência ou contexto em que essas sentenças estão estruturadas;
- Tradução de Texto: tarefa tipicamente baseada em converter um texto, ou parte dele, em uma outra linguagem, diferente da original;
- Sumarização de Texto: tem como objetivo criar resumos de textos longos, preservando as principais ideias desses textos;
- Extração de Informações: tarefa direcionada a extrair informações relevantes de textos, como, por exemplo, nome, países, organizações, entre outras;
- Obtenção de Informações: utilizada em grande escala em mecanismos de busca na internet, esta tarefa tem o objetivo de, a partir de em uma entrada de usuário fornecida em linguagem natural, gerar resultados de busca relevantes em documentos, sites, etc., sem que o usuário necessite saber exatamente quais termos de pesquisa utilizar.

2.1.2 *Pipeline* PLN

Para cada tarefa realizada em Processamento de Linguagem Natural é necessária uma série de passos ou etapas de processamento para que seja possível a utilização, por máquinas, de um determinado texto escrito em linguagem natural [14]. Esse conjunto de passos ou etapas é comumente chamado de *pipeline* PLN.

Cabe dizer que ele não apresenta grandes diferenças em relação ao utilizado em aprendizado de máquina ou aprendizado profundo para outros tipos de dados. Contudo, em função da natureza do dado textual, e considerando-se que o objetivo final do PLN é converter dados em formato texto para representações numéricas para que possam servir de insumo para o processamento por um computador, os passos do *pipeline* PLN diferem em alguns pontos.

Os principais pontos de diferença giram em torno do fato de que dados textuais demandam tratamento diferenciado em relação a dados comuns, já que a linguagem humana carrega uma série de regras e ambiguidades, levando-se em consideração também o idioma, trazendo consigo uma série de desafios. De qualquer forma, é durante a realização das etapas do *pipeline* que o texto é transformado e, posteriormente, sua representação

numérica utilizada para a execução e monitoramento de uma tarefa, de forma semelhante ao dado não-textual.

Cumpre destacar que o *pipeline* PLN, para melhor adequação ao contexto específico do problema, pode variar em complexidade, de acordo com a tarefa a ser realizada e com os requisitos do negócio. Para isso, algumas etapas podem ser adaptadas conforme a necessidade.

Assim sendo, de maneira genérica, os passos mais comuns dentro de um *pipeline* PLN [15, 16] compreendem, como ilustra a Figura 2.1: a aquisição de dados textuais, sua limpeza, seu pré-processamento, a engenharia, a partir do texto, de características, também chamadas de *features*, a construção do modelo de aprendizado de máquina ou aprendizado profundo, a avaliação, a implantação, e o monitoramento do modelo.

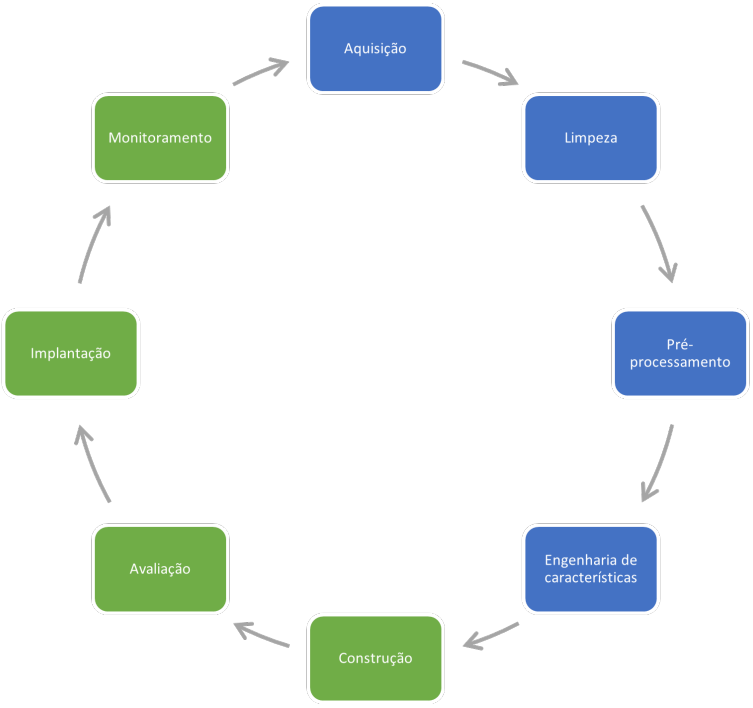

Figura 2.1: *Pipeline* PLN.

2.1.2.1 Aquisição do dado textual

Processo de coletar e filtrar os dados textuais para que eles possam ser armazenados e posteriormente utilizados de forma eficiente nos passos seguintes dentro do *pipeline*. Podem ser provenientes de uma fonte interna, como arquivos de petições iniciais no caso da pesquisa aqui desenvolvida, ou uma fonte externa, como o texto de página web obtido por um *crawler*.

2.1.2.2 Limpeza do dado textual

Uma vez coletado, o dado textual bruto precisa ser preparado, eliminando-se caracteres desnecessários ou elementos não textuais, como metadados, *tags* e marcações de documentos, como os encontrados em xml, html, dentre outros, preservando-se o texto puro para as etapas subsequentes.

2.1.2.3 Pré-processamento do dado textual

Este passo, em síntese, consiste em estruturar o texto limpo de forma que ele possa ser analisado nos passos seguintes da tarefa em execução. Entre as técnicas mais comuns de pré-processamento de texto, podem-se citar:

- *lowercasing*: transformação dos caracteres das palavras em sua forma minúscula;
- remoção de *stopwords*: remoção de palavras de ocorrência mais comum dentro de um texto por serem de alta frequência na língua, como artigos e preposições, e que, por isso, não possuem relevância para as etapas seguintes do *pipeline*; na língua portuguesa podem-se citar palavras como "em", "do", "da", "para", "um", "entre"e muitas outras;
- remoção de pontuação e espaços em branco extras: remove toda a pontuação de um texto, bem como caracteres adicionais não necessários para o texto que representam espaços em branco;
- tokenização: separação de uma sentença em trechos menores, chamados *tokens*, podendo esses trechos, de forma geral, serem caracteres, palavras ou parte de palavras;
- segmentação de sentença: de forma similar à tokenização, a segmentação de sentença consiste na separação de parágrafos inteiros em sentenças individuais;
- stemização: contração de uma palavra em sua forma raiz ou *stem*; nesta técnica, a palavra reduzida obtida não necessariamente será uma palavra existente e gramaticalmente correta;
- lematização: assim como a stemização, consiste na redução de uma palavra à sua forma raiz, ou, neste caso, lema; diferentemente da stemização, a palavra é reduzida removendo-se todas as suas flexões, permanecendo gramaticalmente correta após a transformação;
- POS *tagging*: *Part of Speech tagging* é a técnica que representa a associação (*tagging*) das palavras de uma sentença às suas respectivas classes gramaticais (*Part of Speech*).

2.1.2.4 Engenharia de características

Também chamada de extração de características, ou vetorização, esta etapa tem o objetivo de, a partir do texto pré-processado, estruturar características do texto em representações numéricas para que possam ser utilizadas na construção de modelos baseados em algoritmos de aprendizado de máquina ou de aprendizado profundo.

Corresponde à criação de um conjunto de vetores contendo a representação numérica de palavras contidas em um texto. Dentre as técnicas mais comuns, podem-se citar *Bag-of-Words* (BOW), *Term Frequency-Inverse Document Frequency* (TF-IDF) e *Embeddings*.

2.1.2.5 Construção do modelo

Extraídos os vetores numéricos de características a partir do texto, os dados estão prontos para a aplicação dentro de um modelo de aprendizado de máquina ou aprendizado profundo. Nesta etapa, os algoritmos de aprendizado de máquina ou de aprendizado profundo serão aplicados, ou treinados, para a descoberta de padrões presentes dentro de um determinado conjunto de dados, obtendo-se, ao fim do processo, um modelo ajustado.

2.1.2.6 Avaliação do modelo

Após o ajuste do modelo a partir da etapa anterior, procede-se à etapa de avaliação, que corresponde, tão somente, ao processo de se confirmar que o modelo ajustado consegue atingir seu propósito, ou seja, o quão efetivo ele é à luz dos requisitos estabelecidos.

Aqui procura-se avaliar o desempenho e capacidade de generalização do modelo ajustado, com o intuito de evitar a ocorrência de anomalias na captura dos padrões do conjunto de dados: *overfitting*, quando os dados se ajustam demais aos dados de treinamento, e *underfitting* quando os padrões dos dados não são detectados de forma adequada.

Nesta etapa também são realizados os ajustes dos parâmetros dos algoritmos de aprendizado de máquina ou de aprendizado profundo (chamados de hiperparâmetros) com base nos resultados obtidos.

Técnicas de validação diferentes podem ser utilizadas em contextos distintos, a variar, entre outras coisas, conforme as propriedades do conjunto de dados (tamanho por exemplo). Podem-se mencionar como algumas das técnicas mais comuns de validação:

- *hold-out*: o conjunto de dados é divido em duas ou três partes, sendo uma delas, na maior proporção, destinada ao ajuste do modelo, outra destinada à validação propriamente dita, e, uma terceira, caso necessária, destinada a testes do modelo final ajustado, utilizando-se dados desconhecidos;

- validação cruzada (*cross-validation*): o conjunto de dados é dividido em amostras chamadas de partições ou *folds*, mutuamente exclusivos, e o processo de validação é realizado em várias iterações, onde, para cada uma delas, uma partição é usada para validação do modelo e as demais para seu o treinamento; o modelo é avaliado pelo conjunto dos resultados das iterações; algumas das variações dessa técnica incluem *k-fold cross-validation*, *leave-one-out cross-validation*, *nested cross-validation*, *stratified cross-validation* entre outras;

- *bootstrap*: nessa técnica são retiradas amostras aleatórias (com substituição) do conjunto de dados e, para cada iteração do processo de validação, a amostra retirada é usada como conjunto de treinamento, sendo o restante dos dados utilizado para validação do modelo; como na validação cruzada, o modelo é avaliado pelo conjunto dos resultados das iterações.

Comumente são utilizadas algumas métricas de desempenho para avaliação do modelo durante o processo de validação, incluindo:

- acurácia: representa a proporção de acertos do modelo e é dada pela equação

$$\frac{VP + VN}{VP + VN + FP + FN} \tag{2.1}$$
$$(VP = \text{Verdadeiro Positivo}, VN = \text{Verdadeiro Negativo},$$
$$FP = \text{Falso Positivo e } FN = \text{Falso Negativo})$$

- precisão: indica qual a proporção verdadeiramente positiva de todos os dados classificados como positivos e é dada pela equação

$$\frac{VP}{VP + FP} \tag{2.2}$$

- *recall*, sensibilidade ou revocação: proporção dos dados classificados como positivos dentre todos os dados realmente positivos que existem na amostra, dado pela equação

$$\frac{VP}{VP + FN} \tag{2.3}$$

- *F1-Score*: é a média harmônica entre precisão e *recall*, bem útil para avaliar modelos com desbalanceamento, e é calculada pela equação

$$2 \times \frac{\text{Precisão} \times \text{Recall}}{\text{Precisão} + \text{Recall}} \tag{2.4}$$

2.1.2.7 Implantação do modelo

Se o desempenho do modelo é considerado satisfatório sob o prisma dos requisitos do negócio, o modelo é então implantado para utilização em situações reais do dia a dia.

2.1.2.8 Monitoramento do modelo

Depois de implantado, o modelo passa a ser monitorado. Caso sejam detectadas quedas de desempenho ou outras anomalias, o *pipeline* é reiniciado, buscando um aprimoramento do modelo.

2.2 Classificação de Texto

O problema a ser estudado no escopo da pesquisa aqui desenvolvida, dentre os tipos já mencionados, diz respeito à tarefa de classificação. Classificação é uma tarefa de mineração de dados baseada em aprendizado de máquina supervisionado, usada para categorizar cada um dos itens de um conjunto de dados em variáveis de interesse ou categóricas, que correspondem a classes ou categorias pré-definidas, com base em determinadas características ou atributos (*features*) extraídos do referido conjunto de dados.

Por aprendizado supervisionado, entende-se como a técnica de aprendizado de máquina que aprende a partir de dados pré-existentes destinados ao treinamento de um modelo. É a técnica pela qual o algoritmo é apresentado a dados de treinamento que, por sua vez, são exemplos que incluem as entradas e as saídas correspondentes desejadas, habilitando esse algoritmo a aprender uma determinada função [17], no caso presente, a função de classificação.

Assim sendo, classificação é uma forma de análise de dados que extrai modelos, descrevendo as classes de dados importantes [18]. Tais modelos, chamados classificadores, preveem rótulos de classes categóricas (discretas e não ordenadas) [19].

Essa tarefa segue, de forma bem próxima, o *pipeline* anteriormente descrito, destacando-se por dois momentos distintos: o primeiro consistindo no aprendizado ou treinamento no qual o modelo é construído, utilizando-se dados previamente categorizados (aprendizado supervisionado), e o segundo correspondendo à etapa de classificação, onde o modelo é usado para prever as classes (rótulos) para o conjunto de dados fornecido.

O modo como essa classificação é feita pode variar, a depender do número de classes existentes ou do número de classes que cada item do conjunto de dados pode ser rotulado. Caso o modelo de classificação seja treinado para identificar e atribuir apenas uma classe dentre múltiplas, ele será chamado de classificador multiclasses. Caso precise atribuir mais de uma classe dentre várias possíveis, é chamado de classificador multirótulos.

Também existem tipos de classificação em que apenas uma entre duas classes pode ser atribuída, chamado de classificação binária e ainda outra, utilizada em situações de desbalanceamento severo do conjunto de dados [20], em que apenas uma classe, dentre as várias existentes, é utilizada para o treinamento do modelo, sendo, dessa forma, especializado em reconhecer apenas a classe em que foi treinado. Nesse caso, tem-se um classificador de classe única ou *one-class classifier*.

Além das abordagens, existem vários tipos de algoritmos de classificação em aprendizado de máquina e aprendizado profundo, cada um com suas próprias características e aplicabilidades. O melhor classificador dependerá do problema específico e do conjunto de dados que está sendo analisado. Portanto, não é possível determinar um único melhor classificador [18].

A classificação de texto, parte relacionada a Processamento de Linguagem Natural, é uma tarefa de aprendizado supervisionado na qual um modelo é treinado para prever rótulos de documentos ainda não vistos, a partir da observação de documentos previamente rotulados [21].

É uma tarefa cujo objetivo é identificar a classe de um texto pré-processado, a partir de um modelo de aprendizado de máquina ou de aprendizado profundo construído a partir de um *corpus* de documentos previamente categorizados. Como exemplos de utilização dessa tarefa, citam-se a filtragem de leis, de notícias e de outros tipos de texto filtráveis, classificação de documentos, filtragem de spam, análise de sentimentos, além de várias outras possibilidades.

2.3 Algoritmos de Classificação de Texto

A escolha do algoritmo classificador mais adequado ao caso de uso é um passo muito importante dentro do *pipeline* PLN. Como dito anteriormente, considerando-se que não existe um classificador considerado melhor para qualquer tipo de situação, é muito importante que o algoritmo classificador escolhido leve em consideração os aspectos específicos do problema a ser resolvido, além, é claro, dos aspectos relacionados ao conjunto de dados em análise.

Vários são os algoritmos de classificação estudados na comunidade acadêmica ao longo do tempo. À luz das abordagens de classificação de texto pesquisadas, podem-se definir dois grandes grupos de algoritmos: algoritmos tradicionais de aprendizado de máquina e algoritmos de aprendizado profundo [22].

2.3.1 Algoritmos de Aprendizado de Máquina

O primeiro grande grupo refere-se aos algoritmos tradicionais de aprendizado de máquina. Dentro do aprendizado supervisionado, essa categoria de algoritmos tem sido objeto de pesquisas em diversas aplicações e aspectos, incluindo a tarefa de classificação de texto.

Ainda considerados na pesquisa científica como estado da arte, podem-se citar como alguns dos principais algoritmos tradicionais constantemente comparados em *surveys* mais recentes [22, 23, 24] sobre métodos de classificação de texto: *Support Vector Machine* (SVM), *Naïve Bayes* (NB), Algoritmos baseados em Árvores de Decisão, *k-Nearest Neighbors* (k-NN) e Regressão Logística.

2.3.1.1 *Support Vector Machine*

Support Vector Machine é uma técnica de aprendizado de máquina supervisionado, comumente usada para classificação de texto, originalmente pensada para problemas de classificação de dois grupos (classificação binária) [25], podendo classificar mais grupos, utilizando múltiplas tarefas de classificação binárias [26]. Como ilustrado na Figura 2.2, ele funciona criando uma fronteira, ou hiperplano, entre dois grupos de dados, onde se busca maximizar a distância entre o hiperplano e os grupos.

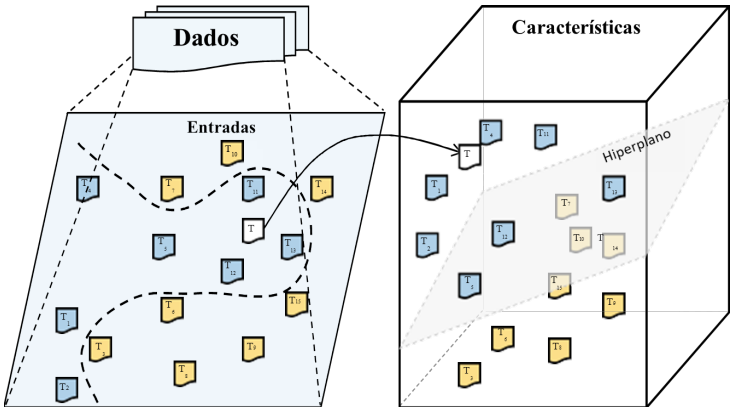

Figura 2.2: Estrutura do SVM (Fonte: [22]).

O SVM demonstrou ser eficaz nas tarefas de classificação de texto e tem vantagens como alta precisão e capacidade de lidar com dados de alta dimensionalidade [18, 27], sendo robusto no que diz respeito a *overfitting*, além de poder lidar com problemas de classificação linear e não linear [28].

No entanto, o algoritmo também tem algumas desvantagens. A primeira grande desvantagem é o fato de ser um classificador nativamente binário, tendo problema para tarefas de classificação multirrótulos [29] por exemplo. Além disso, para que possua desempenho

satisfatório, requer uma seleção cuidadosa de parâmetros e pode ser sensível à escolha da função de kernel [22, 28].

2.3.1.2 *Naïve Bayes*

Método de classificação de texto amplamente usado, o *Naïve Bayes* (Figura 2.3) aplica a teoria estatística baseada no Teorema de Bayes (Equação 2.5 e Equação 2.6) para a tarefa de classificação de texto. Esse classificador assume que os valores das características de uma determinada classe são condicionalmente independentes, sendo, por este motivo, considerado ingênuo (*naive*). Isso tem o efeito de simplificar a computação envolvida [19].

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.5}$$

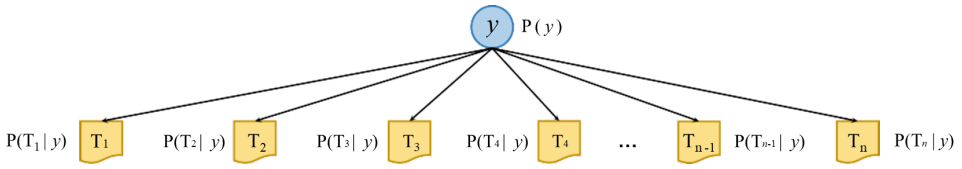

Figura 2.3: Estrutura do *Naïve Bayes* (Fonte: [22]).

Como mencionado, o algoritmo [30] assume a independência condicional entre as variáveis preditoras, dado o valor da variável-alvo. Seja um conjunto de variáveis $T = [T_1, T_2, \ldots, T_n]$ representando características extraídas de um texto e uma variável de classe y. A probabilidade posterior pode ser expressa pelo Teorema de Bayes:

$$P(y|T_1, T_2, \ldots, T_n) = \frac{P(y)P(T_1, T_2, \ldots, T_n|y)}{P(T_1, T_2, \ldots, T_n)} \tag{2.6}$$

No entanto, o denominador $P(T_1, T_2, \ldots, T_n)$ é constante para todas as classes e pode ser ignorado para fins de classificação. Aplicando a suposição de independência condicional, tem-se:

$$P(y|T_1, T_2, \ldots, T_n) \propto P(y) \prod_{j=1}^{n} P(T_j|y), \tag{2.7}$$

onde $P(y)$ é a probabilidade *a priori* da classe y, e $P(T_j|y)$ representa a probabilidade condicional de cada termo T_j dado y.

Sua principal força é sua eficiência, executando de forma rápida ambos os processos de treinamento e classificação. Por aliar eficiência e boa precisão, é frequentemente utilizado como linha de base em pesquisas de classificação de textos [31]. Outra vantagem é o bom desempenho com dados textuais [28], sendo simples de implementar e requerendo uma

quantidade pequena de dados de treinamento para realizar a estimativa dos parâmetros necessários à classificação [32], lidando bem com múltiplas classes [33].

A simplificação trazida por se assumir a independência condicional também pode ser problema caso as características extraídas sejam altamente correlacionadas. O algoritmo também não considera a frequência com que palavras ocorrem em um texto [32]. Pela violação da assunção de independência condicional, acaba sendo muito sensível a como os dados de entrada são extraídos e preparados [33].

2.3.1.3 Algoritmos baseados em Árvores de Decisão

Um algoritmo de Árvore de Decisão [34] tem como princípio de funcionamento a construção de um modelo com base em estimativas estatísticas e em uma série de verificações de condições lógicas. Esse modelo consiste em um grafo de árvore, descrito na teoria de grafos como um grafo orientado acíclico, possuindo nós e arestas entre eles, sendo que cada nó tem uma entrada e duas ou mais arestas de saída para nós adjacentes. Dois dos mais conhecidos algoritmos são ID3 (*Iterative Dichotomiser 3*) e C4.5, expansão do ID3 [29].

A raiz da árvore corresponde ao estado inicial do processo de classificação. O estado inicial constitui-se na primeira verificação lógica a ser realizada. Cada nó interno tem uma premissa de divisão associada, ou seja, uma função lógica relacionada ao cálculo da condição de ramificação, procedendo às verificações lógicas seguintes. Os nós finais, ou folhas, representam as classes resultantes. Ao percorrer a árvore, começando pela raiz, cada nó divide o conjunto de soluções até alcançar o nó folha que define a classe prevista pelo algoritmo.

Ou seja, pela ótica do fluxo dos dados, o algoritmo começa no nó raiz e testa as amostras de dados (compostas por conjuntos de instâncias, que têm várias características), dividindo o conjunto de dados em subconjuntos diversos de acordo com diferentes resultados. Cada subconjunto constitui um nó filho e cada nó folha na árvore de decisão representa a classe prevista. A construção da Árvore de Decisão, dessa maneira, tem o objetivo de determinar a correlação entre classes e características, explorada posteriormente para prever as classes para dados ainda desconhecidos [22].

Além de facilidade de compreensão e a vantagem visual que possuem, as Árvores de Decisão possuem bom desempenho em classificação de texto quando há número menor de características [35], sendo muito rápidas para treinamento e predição [28]. Também fazem bem o trabalho de transformar os dados em conhecimento, pegando um conjunto de dados não familiares e extraindo um conjunto de regras, e são baratas computacionalmente [33]. Como desvantagens, elas são propensas a *overfitting* [28], possuem tendência a basear as classificações em um menor número possível de testes, levando a um desempenho fraco em

classificação de texto, e também podem levar a uma grande e excessivamente complexa estrutura de árvore quando o conjunto de dados possui grande número de entradas [32].

Funcionando como uma extensão das Árvores de Decisão, a Floresta Aleatória é um método de aprendizado em conjunto (*ensemble learning*) para classificação de texto. A ideia principal do método é construir e agrupar várias Árvores de Decisão aleatórias durante o treinamento [28] e por isso é chamada de "Floresta". Os resultados da classificação de suas saídas são determinados pelo número de votos de todos os resultados de saída das árvores agrupadas. Essas estruturas são ilustradas na Figura 2.4.

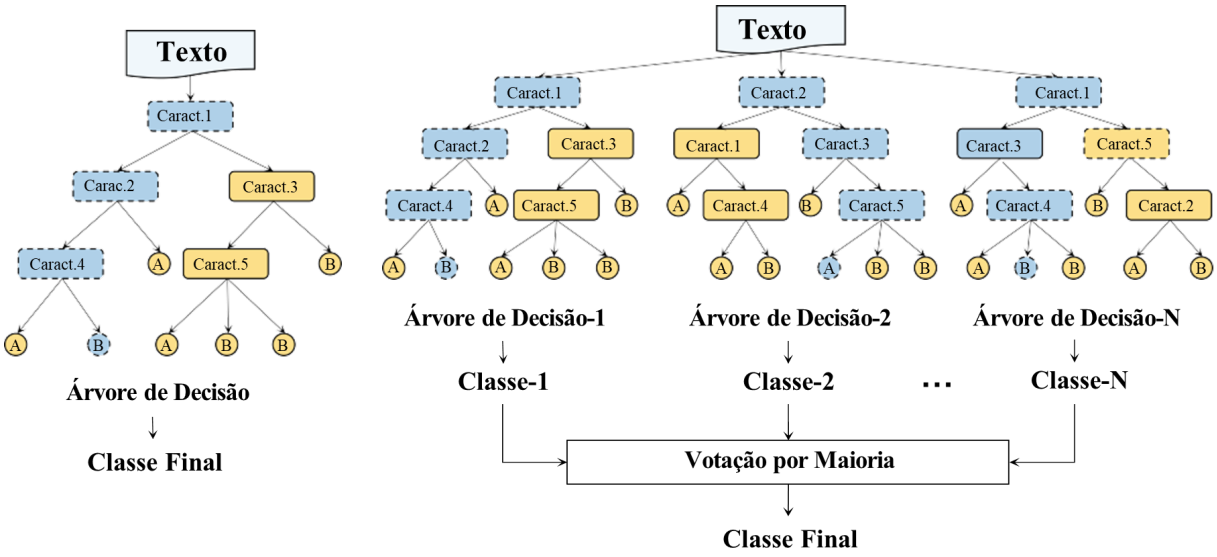

Figura 2.4: Um exemplo de Árvore de Decisão (esquerda) e a estrutura de uma Floresta Aleatória (direita). Os nós tracejados representam a rota de decisão (Fonte: [22]).

A Árvore de Decisão neste caso é a unidade básica e, cada uma que compõe uma Floresta é um classificador. Dessa forma, para uma determinada amostra de entrada, N Árvores terão N resultados de classificação. A Floresta Aleatória integra todos os resultados de votação de classificação e especifica a classe com o maior número de votos como sua saída final [36].

Florestas Aleatórias são mais robustas a erros e *outliers*, embora ainda possam ocorrer a depender da força dos classificadores individuais e da medida da correlação entre eles (quanto menor, melhor). Além disso, são insensíveis ao número de características selecionadas em cada divisão e eficientes em conjuntos de dados muito grandes [19].

Elas também são muito rápidas para se treinar em comparação com outras técnicas, tais como as de aprendizado profundo e demais técnicas de aprendizado em conjunto, mas muito lentas para predições uma vez treinadas, o que piora quanto maior o número de Árvores de Decisão no sistema. Por fim, além de serem também propensas a *overfitting*,

perdem a facilidade de interpretação visual, considerada uma vantagem no contexto de uma única Árvore de Decisão [28].

2.3.1.4 *k-Nearest Neighbors*

O algoritmo *k-Nearest Neighbors* tem como funcionamento atribuir a uma amostra não classificada a classe com maior incidência entre o conjunto de k vizinhos mais próximos previamente classificados [37]. O número k de vizinhos mais próximos a serem inspecionados é um hiperparâmetro especificado quando da criação do modelo. O funcionamento da técnica é ilustrado na Figura 2.5.

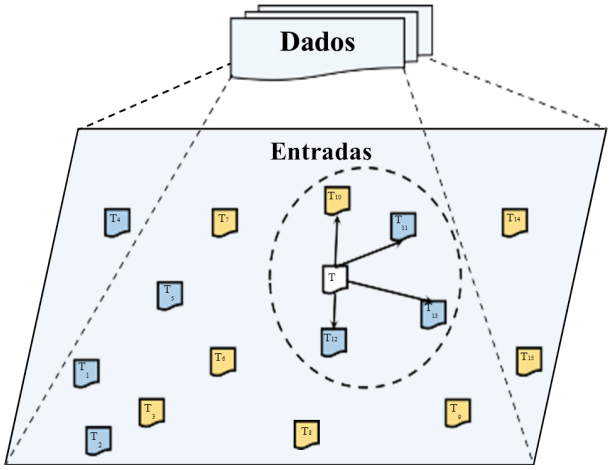

Figura 2.5: Estrutura do k-NN com $k = 4$ (Fonte: [22]).

Esse algoritmo é comumente utilizado e um dos melhores algoritmos de classificação de texto [27, 38], sendo muito efetivo para conjuntos de dados textuais [28]. A técnica é bem conhecida para reconhecimento de padrões, sendo simples de implementar e entender, além de ser particularmente adequada para a tarefa de classificação multiclasses [28, 39] e quando o conjunto de dados é grande [27]

Contudo, existem desvantagens no uso do k-NN para classificação de texto. Suas versões tradicionais utilizam distâncias (euclidiana, similaridade por cosseno, etc.) para o cálculo de similaridade entre os vetores de características das amostras, não conseguindo capturar adequadamente o acoplamento das características entre os dados [38]. Isso pode levar a uma menor precisão da classificação. Além disso, o k-NN pode ser computacionalmente caro [27, 28, 36, 39], especialmente ao lidar com grandes conjuntos de dados [28, 40].

2.3.1.5 Regressão Logística

Um dos métodos de classificação mais antigos e simples, a Regressão Logística é um tipo de algoritmo de aprendizado supervisionado que funciona calculando probabilidade em vez de classes [28], utilizando esse mecanismo para medir a relação entre variáveis dependentes e independentes [41]. Utiliza a função sigmoide, dada pela Equação 2.8, para cálculo do valor da probabilidade. O gráfico dos valores gerados por essa função é semelhante ao da Figura 2.6.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

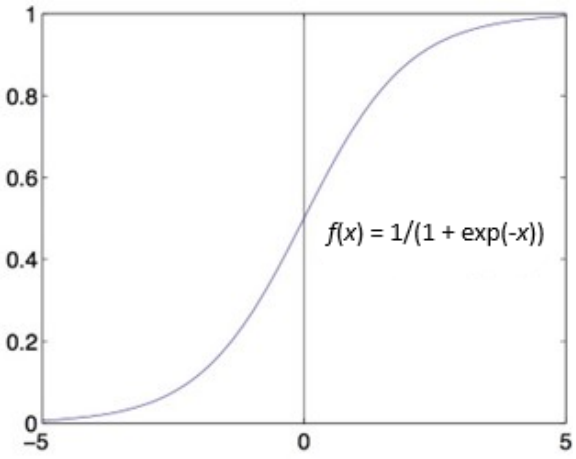

Figura 2.6: Gráfico da função sigmoide.

No classificador por Regressão Logística, cada uma das características (c) é multiplicada por um peso (coeficiente de regressão, representado por β) e depois somadas entre si (Equação 2.9). O resultado é então utilizado na função sigmoide, obtendo-se um número entre 0 e 1, como uma estimativa de probabilidade. Os melhores coeficientes da regressão são obtidos com a utilização de algoritmos de otimização [33].

$$x = \beta_0 + \beta_1 c_1 + \beta_2 c_2 + ... + \beta_n c_n \tag{2.9}$$

Algoritmo essencialmente simples, que pode lidar bem com dados de alta dimensionalidade (dados textuais, por exemplo) [27] e lineares [42]. É computacionalmente barato, fácil de se implementar e fácil de se interpretar quanto à representação do conhecimento [33]. Em compensação, é propenso *underfitting* e pode ter baixa acurácia [33], além de ter a possibilidade de que o processo de treinamento seja caro em função da busca da otimização desejada dos coeficientes [27] e não ser melhor com dados não lineares [42].

2.3.2 Algoritmos de Aprendizado Profundo

Mesmo com a pesquisa científica realizada em décadas sobre os algoritmos tradicionais de aprendizado de máquina, essas técnicas [43] são limitadas em sua habilidade de processar dados naturais em sua forma bruta.

Construir um sistema de reconhecimento de padrões ou de aprendizado de máquina tradicional ainda demanda engenharia cuidadosa e expertise considerável do domínio para modelagem de um extrator de características que transforme o dado bruto em um vetor de características adequado pelo qual o subsistema de aprendizado, por exemplo, um classificador, possa detectar ou classificar os padrões de uma determinada entrada de dados.

A área de Aprendizado Profundo tem constantemente evoluído no intuito de ampliar as possibilidades dentro da inteligência artificial, superando barreiras como as enfrentadas pelas técnicas tradicionais de aprendizado de máquina. É dentro dessa área que estão acontecendo os maiores avanços em resolver problemas que resistiram, por muitos anos, às melhores tentativas da comunidade acadêmica da área de inteligência artificial.

Em estudos diversos, esse campo da pesquisa tem se mostrado muito bom em descobrir estruturas intricadas em dados com alta dimensionalidade, sendo, desta forma, aplicável a vários domínios diferentes do conhecimento, dado que suas técnicas possuem a capacidade de modelar relações complexas e não-lineares entre os dados.

Conceituando-se, pode-se dizer que Aprendizado Profundo constitui um conjunto de técnicas de aprendizado de máquina baseadas no aprendizado de representações, pelo qual um sistema automaticamente aprende e descobre as características necessárias para classificação a partir do processamento de múltiplas camadas de entradas de dados.

Ele compreende técnicas baseadas em redes neurais profundas, versões de redes neurais artificiais com várias camadas escondidas entre a camada de entrada e a de saída, que simulam o funcionamento do cérebro humano para automaticamente aprender características de alto nível a partir de dados, obtendo resultados melhores do que modelos tradicionais para tarefas de aprendizado de máquina, como reconhecimento de fala, processamento de imagens e compreensão de texto [22].

Assim como seu correspondente biológico [44], essas redes são construídas por cadeias densamente interconectadas de unidades simples (neurônios) que recebem um número de entradas de valores reais, possivelmente saídas de outras unidades, e produzem uma única saída de valor real, que, por sua vez, pode ser entrada para várias outras unidades.

Em que pesem os muitos avanços dessas técnicas nos últimos anos, o estudo de redes neurais artificiais não é necessariamente algo novo e, desde o início, uma das motivações dessas redes é justamente capturar o tipo de computação altamente paralelizada baseada em representações distribuídas do cérebro humano.

Hoje em dia, diversas são as arquiteturas de rede neurais existentes e estudadas, assim como suas aplicações. Nesta seção, serão descritas algumas das mais estudadas e utilizadas dentro do Aprendizado Profundo e que são relevantes para o objeto desta pesquisa envolvendo a tarefa de classificação de texto.

Os algoritmos aqui vistos são: Rede Neural Convolucional (CNN), Rede Neural Recorrente (RNN), *Long Short-Term Memory* (LSTM), *Transformers* e uma variante deste chamada *Bidirectional Encoder Representations for Transformers* (BERT).

2.3.2.1 Rede Neural Convolucional

Uma das mais populares redes neurais profundas, a Rede Neural Convolucional [45], ou *Convolutional Neural Network* em inglês. Ela tem seu nome derivado da operação matemática linear entre matrizes chamada convolução.

A CNN tem sido bastante utilizada, com excelente desempenho, em problemas de aprendizado de máquina, sendo especialmente utilizada em aplicações que trabalham com dados de imagens. Pelos bons resultados apresentados, também tem sido bastante utilizada em tarefas de PLN.

Sua arquitetura para tarefas de classificação de texto, como a proposta para classificação de sentenças [46] ou para classificação de textos de alta dimensionalidade [47], de forma geral, começa com a entrada na rede de uma matriz contendo os vetores de características, resultado da transformação da sentença em uma representação numérica na etapa do pipeline de engenharia de características. Essa arquitetura é ilustrada na Figura 2.7

O primeiro estágio do funcionamento do algoritmo envolve a realização do produto escalar entre a matriz de características e as matrizes de pesos pré-definidos, também chamadas de filtros. Logo após, é utilizada uma função de ativação[1] para a transformação da matriz na saída da camada de convolução da rede [50].

Cada um desses filtros possui a mesma dimensionalidade dos vetores de características que compõem a matriz, cobrindo uma janela de altura h de linhas da matriz, também chamada de região do filtro [48]. São feitas várias multiplicações, percorrendo-se todas as linhas da matriz, cada uma gerando um valor único. Ao fim, após a aplicação da função de ativação, serão gerados vetores, chamados de mapas de características, na quantidade equivalente a de filtros da camada de convolução.

Já na camada de *pooling* ou agrupamento, uma função de agrupamento é então aplicada para cada mapa de características. Juntas, as saídas geradas a partir de cada mapa

[1]Função de ativação é uma função matemática para a aplicação de uma transformação não-linear na saída de uma camada de uma rede neural, tornando essa rede capaz de aprender e executar tarefas mais complexas. Elas basicamente decidem se um neurônio deve ser ativado ou não. Ou seja, se a informação que o próximo neurônio está recebendo é relevante para a informação fornecida ou deve ser ignorada [49].

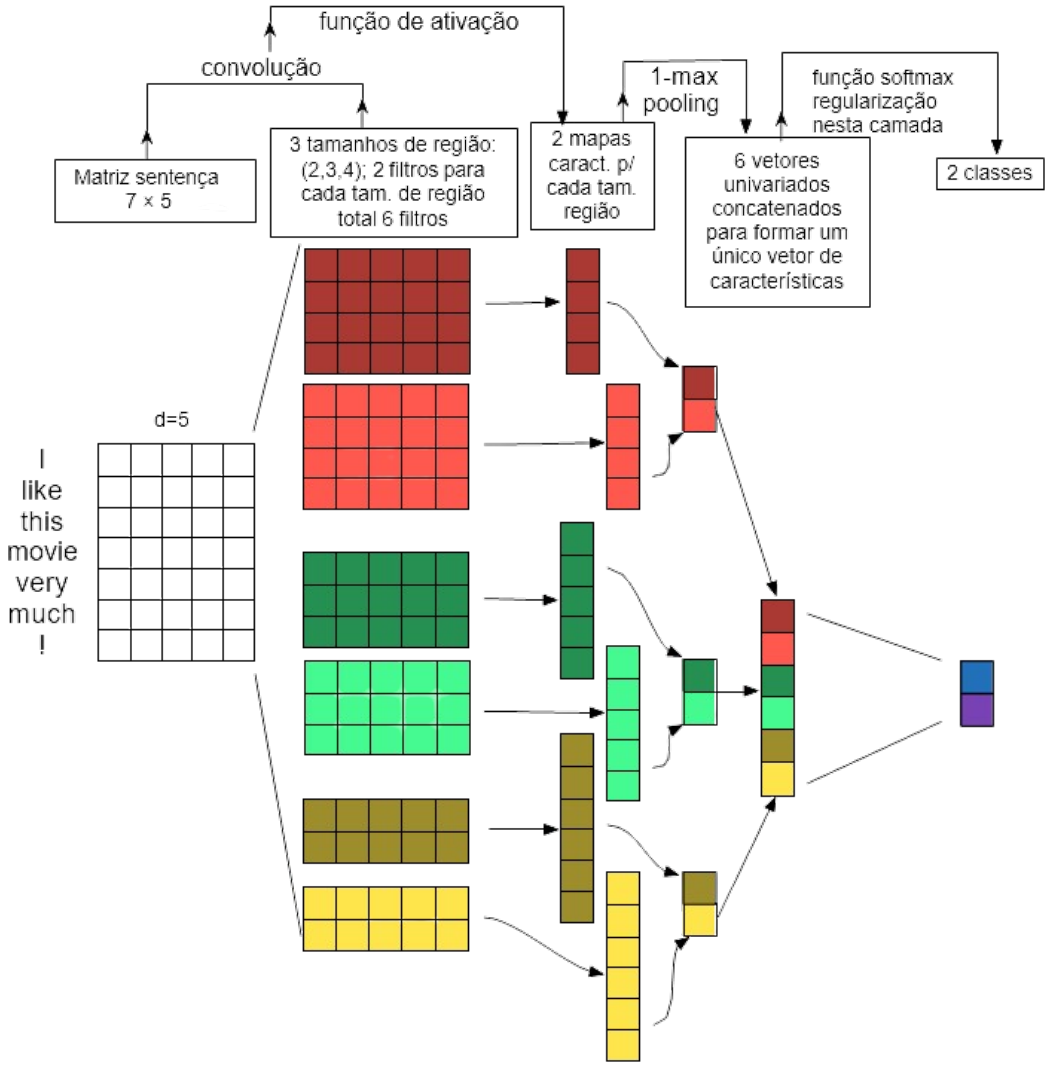

Figura 2.7: Ilustração de uma arquitetura CNN para classificação de sentenças (Fonte: [48]).

são concatenadas em um vetor de características de tamanho fixo que será enviado à camada densa. Uma estratégia comum de agrupamento utilizada com CNN é chamada de *1-max pooling*, pela qual o número de maior valor é extraído do mapa de características.

Por fim, após o vetor ser devidamente processado na camada densa, é feita a aplicação de uma função, no caso da classificação, a função softmax, obtendo-se as previsões de classes. Na camada densa, onde é implementada a rede neural totalmente conectada, pode ser inserida uma camada de *dropout* a título de regularização [45, 46, 48]. Esta técnica zera aleatoriamente valores do vetor de pesos.

2.3.2.2 Rede Neural Recorrente

A Rede Neural Recorrente ou *Recurrent Neural Network* em seu nome original na língua inglesa, é uma das arquiteturas de rede neurais profundas preferidas para tarefas de PLN [50], por serem melhores para tarefas que envolvem entradas sequenciais [51], como fala e linguagem [43].

A principal diferença dessa arquitetura de rede neural se dá pela existência de uma ligação recorrente de retroalimentação das camadas ocultas. Ela permite que, após a aplicada a função de ativação, o resultado retroalimente a próxima camada oculta, além de ser passado para a camada de saída. Esse mecanismo permite a criação de uma memória de curto prazo, que vai se modificando ao longo do tempo. Uma das consequências para tarefas de PLN é lidar melhor com dados sequenciais.

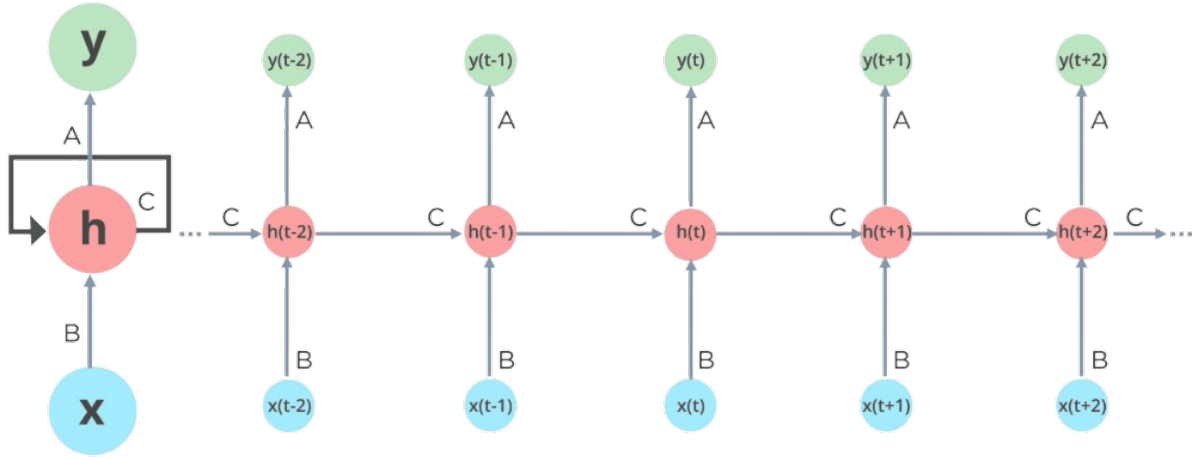

Figura 2.8: Arquitetura de uma Rede Neural Recorrente desdobrada no tempo. As variáveis x e y representam, respectivamente, as entradas e saídas da rede em função do tempo (t). A variável h representa as camadas ocultas da rede e A, B e C representam parâmetros (pesos) padrão da rede usados em todos os passos (Fonte: [52]).

Em termos práticos, como ilustrado na Figura 2.8, para uma tarefa de classificação de texto, a RNN recebe como entrada os vetores de características, *Word Embeddings* por exemplo, provenientes da transformação da sentença em uma representação numérica na etapa do *pipeline* de engenharia de características.

Logo após a aplicação dos pesos, do somatório e da função de ativação, a saída da camada oculta, que possui a mesma dimensão do vetor de entrada, é também repassada para a próxima camada oculta da rede RNN e assim sucessivamente.

Por fim, após a passagem por todos os vetores de características, a previsão da classe é realizada pela última saída de camada escondida, aplicando-se uma função para classifi-

cação. Ressalte-se que a RNN compartilha os mesmos parâmetros pelas diferentes partes do modelo, possuindo os mesmos pesos para cada palavra inserida [22].

Problemas de Desaparecimento e de Explosão do Gradiente Uma RNN, quando desdobrada no tempo, pode ser vista como uma rede neural *feedfoward* muito profunda, na qual todas as camadas compartilham os mesmos pesos [53]. Isso significa que, nessa arquitetura de rede, cada camada se conecta à próxima, sempre em uma única direção, rumo à camada de saída, sem caminho volta.

Descrevendo de forma superficial e genérica a dinâmica do treinamento, os dados de entrada são transformados camada por camada da seguinte forma: multiplica-se os valores dos neurônios de cada camada pelos pesos associados, realiza-se o somatório, aplica-se uma função de ativação e obtém-se uma saída que alimentará as camadas subsequentes, até alcançar a camada de saída da rede.

O resultado final é então comparado aos valores esperados e o cálculo do erro é realizado. Com base nesse cálculo, é aplicado o algoritmo de retropropagação do erro (*backpropagation*), que ajusta os pesos dos neurônios em todas as camadas, partindo da camada de saída até a de entrada.

Esse ciclo de apresentação dos dados de treinamento e ajuste dos pesos é chamado de época (*epoch*). Cada época constitui um passo do processo de aprendizado supervisionado da rede neural. O objetivo do treinamento de uma rede neural *feedforward* é encontrar os valores de pesos que minimizem o erro, definido por uma função de perda, também chamada de função de custo.

Especificamente durante o treinamento de uma RNN, podem ocorrer dois tipos de problemas [53, 54]: problema de desaparecimento (*vanishing*) ou de explosão (*exploding*) do gradiente[2].

A função de perda possui gradientes calculados que servem como guia no ajuste dos pesos. A influência dos gradientes, que apontam para direção onde os erros são menores, se dará ao serem utilizados nas multiplicações para a otimização desses pesos. No caso das ligações recorrentes entre as camadas ocultas, os pesos sofrerão várias multiplicações ao longo do tempo em função do algoritmo de *backpropagation* (*backpropagation through time* - BPTT).

Em situações específicas, como no processamento de longas sequências, eles poderão encolher demasiadamente ou explodir, dadas as sucessivas multiplicações. Nestas situações, a RNN não consegue atualizar seus pesos de maneira eficaz, produzindo resultados indesejados.

[2]No cálculo vetorial, o gradiente é um vetor que indica o sentido e a direção na qual, por deslocamento a partir do ponto especificado, obtém-se o maior incremento possível no valor de uma grandeza a partir da qual se define um campo escalar para o espaço em consideração [55].

2.3.2.3 *Long Short-Term Memory*

Várias são as estratégias para lidar com os problemas de desaparecimento ou explosão do gradiente em redes RNN. Uma das soluções propostas para lidar especificamente com o problema da desaparecimento baseia-se na introdução de um conjunto especial de unidades escondidas chamadas de *Long Short-Term Memory* (LSTM) [56].

Recapitulando o conceito da arquitetura de uma rede RNN tradicional (*vanilla*), a saída de uma camada oculta alimenta a próxima após a aplicação de uma função de ativação, uma tangente hiperbólica no caso da Figura 2.9.

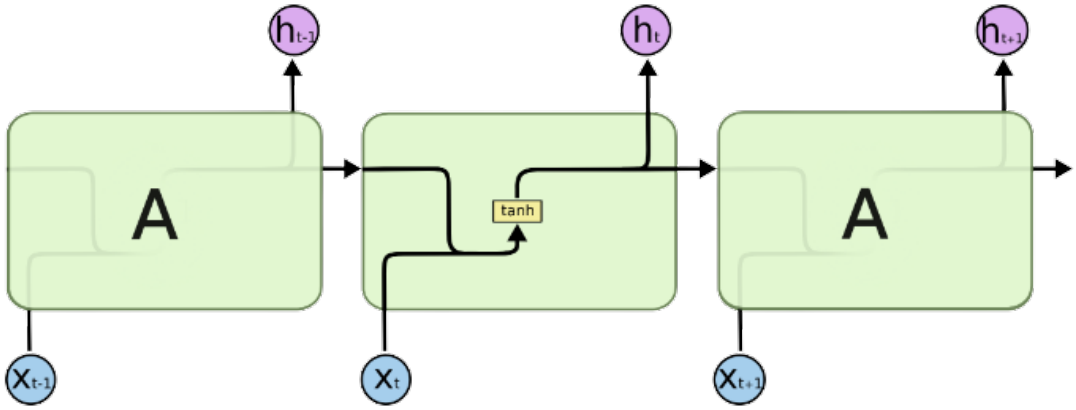

Figura 2.9: Arquitetura de uma Rede Neural Recorrente. Neste exemplo, x e h representam, respectivamente, entradas e saídas em função do tempo (t) (Fonte: [57]).

As redes LSTM (Figura 2.10) possuem esse mesmo funcionamento recorrente entre as camadas ocultas. Porém, essas redes dividem o problema de gerenciamento do contexto em dois subproblemas: remoção da informação não mais necessária do contexto e adição de informação possivelmente necessária para tomada de decisão posterior. A chave para resolver esses subproblemas está em aprender como gerenciar o contexto [13].

O mecanismo necessário para o gerenciamento do contexto reside na própria camada oculta que, nesta variante, é composta por uma célula de memória para lembrar valores através do tempo e três estruturas que se assemelham a válvulas, chamadas de portões, que controlam o fluxo da informação da célula.

A célula de memória, representada pela linha horizontal no topo do diagrama da Figura 2.10, atua como uma espécie de cano, por onde a informação da memória da rede flui. Ela é conectada diretamente nas células de memória das camadas ocultas anterior e posterior [43].

Portões, representados por uma camada de rede neural com função de ativação sigmoide e a operação de multiplicação de vetores, são a forma de remover ou adicionar informação ao estado atual da célula de memória. O quanto de informação deverá trafe-

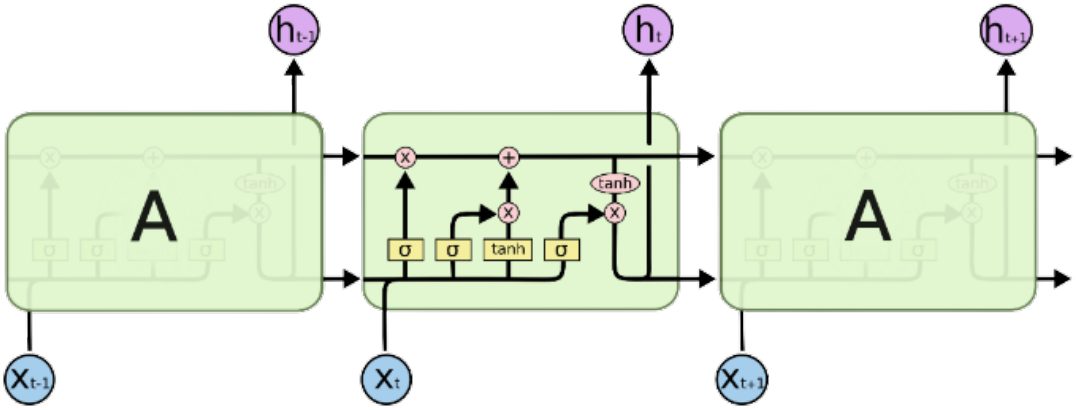

Figura 2.10: Arquitetura de uma RNN com LSTM (Fonte: [57]).

gar por cada componente é regulado pelo resultado da função sigmoide. O resultado de sua saída combinado com a operação de multiplicação tem um efeito de máscara binária, quanto mais ou menos próximo de zero, mais ou menos informação inalterada passará.

Os três portões [58], na ordem em que aparecem na Figura 2.10 da esquerda pra direita, são assim chamados:

- portão de esquecimento (*forget gate*): tem a função de esquecer ou deixar passar a informação do estado anterior da célula de memória;
- portão de entrada (*input gate*): responsável por definir a nova informação a ser adicionada e que modificará o estado atual da célula de memória; a nova informação é um vetor gerado por uma camada de rede neural com função de ativação tangente hiperbólica, a partir da entrada de informações da camada oculta anterior (h_{t-1}) e da camada de entrada atual (x_t);
- portão de saída (*output gate*): recebe o estado atual da célula filtrado por uma função tangente hiperbólica e deixa passar o que deverá sair da rede e que deverá ser encaminhado para a próxima camada oculta (h_t).

Dados esses aspectos, a LSTM tornou-se, ao longo dos últimos anos, o estado da arte para uma variedade de problemas de aprendizado de máquina, principalmente por ser um modelo efetivo e escalável para aqueles problemas de aprendizado relacionados a dados sequenciais, conseguindo ser ampla e efetiva em capturar dependências temporais de longo prazo [59].

Quanto à tarefa de classificação de texto, seu método pode capturar melhor a conexão entre o contexto das palavras e usar a estrutura de portão de esquecimento para filtrar as informações inúteis, o que é propício à melhoria da capacidade total de captura do classificador [22].

2.3.3 *Transformers*

Desde sua introdução em 2.017 [60], a arquitetura *Transformer* tornou-se o padrão para lidar com uma variedade de tarefas de Processamento de Linguagem Natural, tanto do meio acadêmico quanto na indústria [13].

Essa arquitetura [61] foi a síntese de várias ideias como atenção e transferência de aprendizado, além de ter impulsionado arquiteturas de redes neurais que estavam crescendo na comunidade acadêmica. Estes avanços foram os catalisadores de dois dos mais conhecidos *transformers* da atualidade: GPT (*Generative Pre-trained Transformer*) e BERT.

Combinando a arquitetura *Transformer* com pré-treinamento de modelo de linguagem, estes modelos removeram a necessidade do treinamento, a partir do zero, de arquiteturas para realização de tarefas específicas e superaram quase todos os indicadores de desempenho em tarefas de PLN por uma margem significativa.

Nesta seção serão demonstrados os três principais conceitos que estão por trás de todo o impacto que a arquitetura trouxe na pesquisa em Processamento de Linguagem Natural: estrutura codificador-decodificador, mecanismo de atenção e transferência de aprendizado. Por fim, será descrita a própria arquitetura do modelo.

2.3.3.1 Estrutura Codificador-Decodificador

Antes dos *transformers* [13], arquiteturas recorrentes, tal como LSTM, constituíam o estado da arte em PLN. Como já visto, essas arquiteturas possuem estruturas de retroalimentação nas conexões da rede que permitem que a informação seja propagada ao longo do tempo, tornando-as ideais para modelagem de dados sequenciais.

Em uma tarefa que envolve classificação de sequências, (POS *tagging* por exemplo, onde cada *token* recebe um rótulo), existem duas sequências de mesmo tamanho, uma na camada de entrada e outra na camada de saída. Isso significa que, para cada entrada na rede, existe uma saída associada. O processo de classificação que produz essa saída utiliza, em sua maior parte, informação da palavra de entrada e de palavras vizinhas próximas.

No entanto, na configuração tradicional, essas redes não são apropriadas para tipos de tarefas como tradução, por exemplo. Isso se dá porque, nessas situações, uma sequência de entrada de tamanho arbitrário pode resultar em uma sequência de saída de tamanho diferente, além de não estarem necessariamente alinhadas palavra a palavra.

Foi justamente na área de tradução em que as redes RNN desempenharam um papel importante. Para lidar adequadamente com essa tarefa, uma solução proposta foi a adaptação das arquiteturas dessas redes, até então seguindo um modelo palavra a palavra, para possibilitar o aprendizado sequência a sequência.

A estrutura sequência a sequência [62] proposta (Figura 2.11), também chamada de codificador-decodificador (*encoder-decoder*), utiliza duas redes LSTM: uma para codificar a sequência de entrada e outra para decodificar a sequência de saída. A partir da sequência de entrada, a solução proposta especificou a geração de um vetor de dimensão fixa contendo a representação numérica do contexto, dado pelo último estado oculto do codificador. O vetor então serviria como a entrada do decodificador, que geraria a sequência de saída.

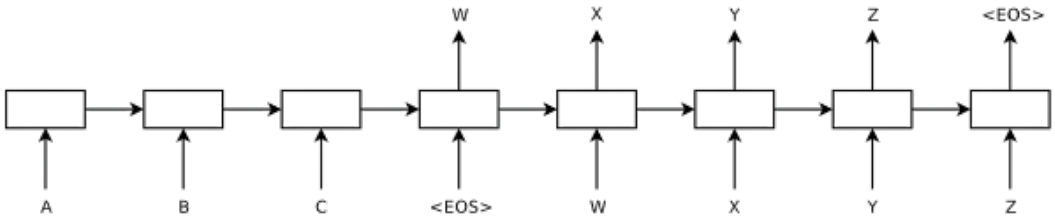

Figura 2.11: Estrutura codificador-decodificador baseada em redes LSTM. A rede interrompe as previsões quando sua última saída é o *token* de fim de sentença (<EOS>) (Fonte: [62]).

Contudo, o principal ponto fraco dessa estrutura é que o último estado oculto do codificador cria um gargalo de informação: ele precisa representar o significado de toda a sequência de entrada, porque o decodificador precisa disso para gerar a sequência de saída. Isso é um problema para longas sequências, já que a informação do início pode se perder no processo de compressão em um único vetor de dimensão fixa que contém a representação numérica.

Para superar essa limitação, foi proposto um conceito chamado de mecanismo de atenção [63], que permite que o decodificador tenha acesso, não só ao estado oculto final, mas a todos os estados ocultos do codificador ao longo do tempo. Esse é um componente chave em muitas das arquiteturas atuais modernas de redes neurais [61].

2.3.3.2 Mecanismo de Atenção

A ideia principal por trás do mecanismo de atenção [61] é que, em vez de produzir um único estado oculto para a sequência de entrada, o codificador disponibilize ao decodificador o estado oculto de todos os passos do processo de codificação. Contudo, usar todos eles ao mesmo tempo criaria uma entrada muito grande para o decodificador.

Assim sendo, algum mecanismo é necessário para priorizar quais estados utilizar. Esse é justamente o conceito por trás de atenção. Ela permite que o decodificador atribua um peso (atenção) diferente para cada um dos estados do codificador, conforme o andamento da etapa de decodificação ao longo do tempo.

Da mesma maneira do modelo anterior, o mecanismo de atenção ainda trabalha com um único vetor de contexto de tamanho fixo utilizado pelo decodificador em um momento específico do tempo. Contudo, como ilustrado no esquema da Figura 2.12, em vez de representar o último estado final do codificador, esse vetor de contexto é uma função de todos os estados ocultos, que podem variar em quantidade conforme o tamanho da entrada.

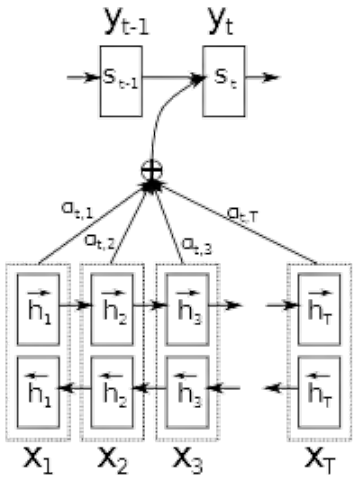

Figura 2.12: Mecanismo de Atenção (Fonte: [63]).

A forma de cálculo desse vetor de contexto de tamanho fixo passado ao decodificador é dada pelo somatório dos produtos de todos os pesos (α_{ij}) pelos valores de anotações, que são o mapeamento da sequência de entrada realizado por uma rede RNN bidirecional.

Cada anotação (h_i) contém a informação sobre toda a sequência de entrada, com um forte foco nas partes que cercam a i-ésima palavra da sequência [63]. Desta maneira, os pesos são focados a uma parte específica do texto que é relevante para o saída que o decodificador está produzindo no momento (y_t).

É desta forma que o mecanismo de atenção [13] substitui o vetor estático de contexto da abordagem anterior por outro que é dinamicamente derivado dos estados ocultos do codificador, diferentes para cada palavra que está sendo prevista no decodificador.

A cada passo da decodificação, o contexto é gerado novamente, levando todos os estados ocultos do codificador em consideração. Esses contextos dinâmicos são disponibilizados durante o processo de decodificação, condicionando o processamento do estado oculto atual do decodificador, que também utiliza seu estado oculto (s_{t-1}) e saída (y_{t-1}) anteriores.

Ao priorizar quais palavras de entrada são mais relevantes em cada momento do tempo, esses modelos baseados em mecanismos de atenção são capazes de aprender alinhamentos não triviais entre palavras das sentenças de entrada e saída. A Figura 2.13 ilustra como

o decodificador é capaz de corretamente alinhar, por exemplo, as palavras *zone* e *Area*, que estão ordenadas diferentemente nos dois idiomas utilizados no processo de tradução (francês e inglês) [61].

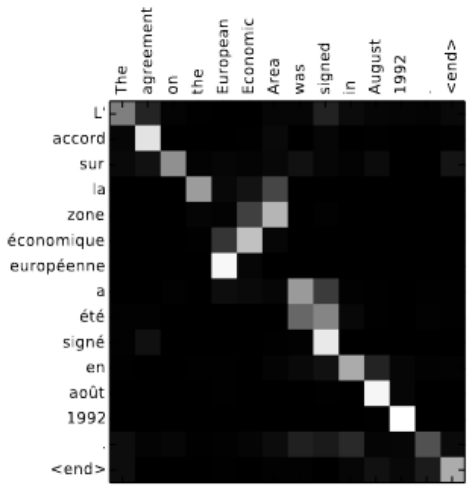

Figura 2.13: Sequência arbitrária traduzida do inglês para o francês utilizando o mecanismo de atenção. Os eixos x e y do gráfico correspondem, respectivamente, às palavras da sentença no idioma de origem e no idioma de destino. Os pesos, graduados de zero a um, são representados por pixels em escala de cinza (0 = preto, 1 = branco) (Fonte: [63]).

2.3.3.3 Transferência de Aprendizado

Transferência de aprendizado (*transfer learning*) refere-se a uma situação onde o que foi aprendido em um ajuste é explorado para melhorar a generalização em outro ajuste [64]. Em outras palavras, é o método de adquirir conhecimento, a partir de uma tarefa ou domínio, aplicando-o (transferindo-o) para resolver uma nova tarefa [13].

Nesta técnica, uma rede de base [65] é treinada sobre um conjunto de dados de base, para realização de uma determinada tarefa de base. Após, as características aprendidas são reaproveitadas, ou transferidas, para uma segunda rede alvo para ser treinada com um conjunto de dados alvo, para realização de uma tarefa alvo.

Este processo tenderá a funcionar adequadamente se as características usadas para treinamento da rede base forem genéricas o suficiente para servirem para ambas as tarefas, base e alvo, em vez de serem específicas apenas para a tarefa base. Isso permite que a rede alvo faça uso do conhecimento aprendido da tarefa original.

Arquiteturalmente [61], isso envolve dividir o modelo em um corpo (*body*) e uma cabeça (*head*). O corpo pode ser reaproveitado em vários domínios diferentes, mudando-se apenas a cabeça, a depender da tarefa que se realizará utilizando-se aquele corpo. Comparada ao

aprendizado supervisionado tradicional, esta abordagem tipicamente produz modelos de alta qualidade que podem ser treinados muito mais eficientemente em uma variedade de tarefas-alvo, com muito menos dados rotulados. A comparação entre as duas abordagens é ilustrada na Figura 2.14.

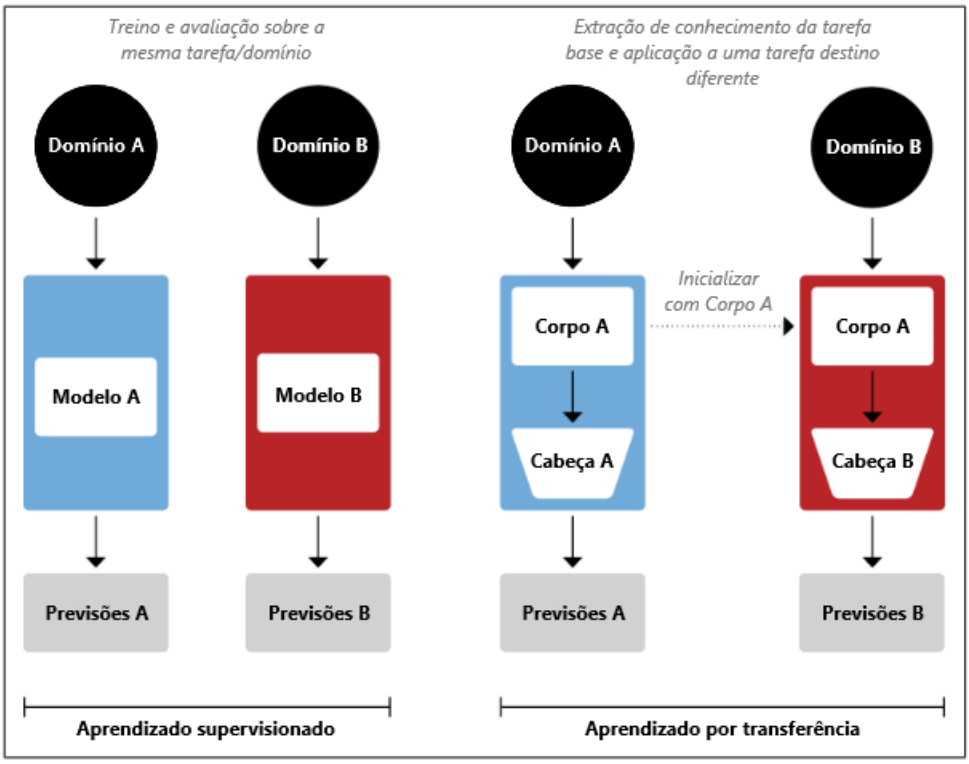

Figura 2.14: Comparação entre os aprendizados supervisionado tradicional (esquerda) e por transferência (direita) (Fonte: [61]).

Durante o treinamento, os pesos do corpo aprendem sobre um amplo conjunto de características do domínio base e esses pesos são usados para inicializar o novo modelo para a tarefa alvo. Quando o conjunto de dados alvo é significativamente menor do que o conjunto de dados de base, a transferência de aprendizado pode ser uma ferramenta poderosa para permitir o treinamento de uma rede alvo grande (com muitos parâmetros[3]), sem o problema de *overfitting* [65].

Em PLN, o processo de treinar esse conjunto de dados base para o aprendizado de algum tipo de representação de significado de palavras ou sentenças sobre quantidades muito grandes de texto é chamado de pré-treinamento. Os modelos resultantes são chamados de modelos de linguagem pré-treinados [13].

Na transferência de aprendizado, estes modelos pré-treinados podem sofrer um ajuste fino (*fine-tuning*) para a realização da tarefa alvo, classificação de texto por exemplo, com

[3]O mesmo que pesos.

um número muito menor de dados rotulados, tipicamente atingindo maior acurácia do que modelos treinados do zero (aprendizado supervisionado), com o mesmo volume de dados rotulados [61].

O ajuste fino consiste justamente em aproveitar o corpo (rede pré-treinada) e colocar a rede neural adicional (cabeça) para a realização da tarefa específica, como classificação de texto, por exemplo.

Opcionalmente, como proposto no método para transferência de aprendizado para tarefas de PLN chamado *Universal Language Model Fine-tuning* (ULMFiT) [66], originalmente aplicado a redes LSTM, pode-se realizar o passo intermediário de adaptação de domínio, onde o modelo de linguagem pré-treinado genérico passa por uma nova tarefa de modelagem de linguagem, desta vez sobre o domínio alvo específico.

2.3.3.4 Arquitetura do Modelo

Um dos gargalos computacionais sofridos por redes RNN é o processamento sequencial do texto. Embora redes CNN sejam menos sequenciais, o custo computacional de capturar relacionamentos entre palavras também cresce dependendo do tamanho da sentença em que estão inseridas [67].

Essa natureza sequencial [60] inviabiliza a paralelização dentro dos exemplos de treinamento, o que se torna crítico em tamanhos maiores de sequências. A ideia por trás da arquitetura do modelo *Transformer* é evitar a recorrência e, em vez disso, apoiar-se inteiramente sobre um mecanismo de atenção para extrair dependências globais entre entrada e saída. Com isso em mente, os pesquisadores propuseram a arquitetura vista na Figura 2.15, brevemente descrita como aparece no trabalho de referência, chamado *Attention Is All You Need*.

Pilhas de Codificadores e Decodificadores A arquitetura do modelo *Transformer*, assim como outros modelos de transdução[4] de sequências, possui uma estrutura codificador-decodificador para realização de tarefas de Processamento de Linguagem Natural.

O componente de codificação [68] é uma pilha de seis codificadores. O mesmo acontece com o componente de decodificação. Os codificadores são idênticos e suas estruturas se subdividem em duas camadas, uma com um mecanismo de autoatenção e outra com uma rede neural *feedfoward* totalmente conectada.

O decodificador também possui ambas as camadas. Contudo, entre as duas, também possui uma camada com um mecanismo de atenção que auxilia o decodificador em cap-

[4]De forma ampla, um modelo de transdução produz uma saída para cada entrada lida. No contexto aqui exposto, o termo se refere à tarefa de modelagem de linguagem sequência a sequência, onde, para cada sequência de entrada, é predita uma sequência de saída [12].

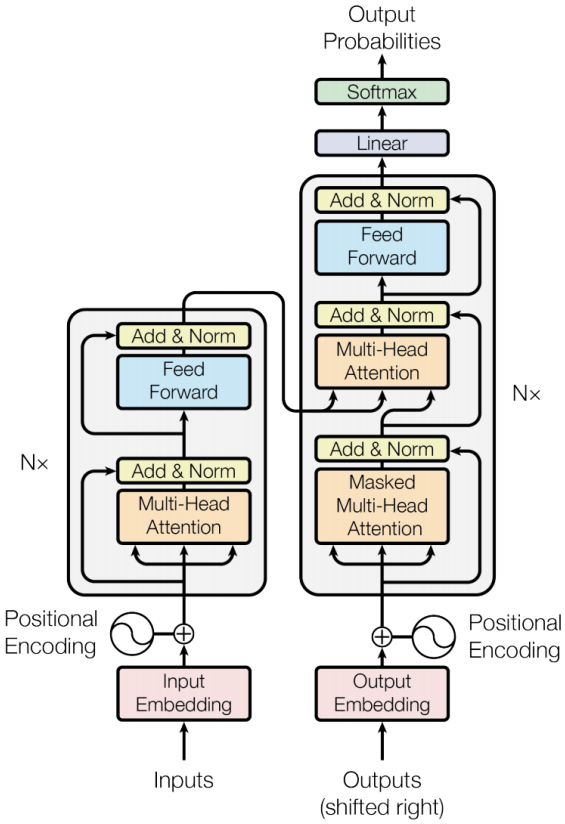

Figura 2.15: Arquitetura *Transformer* proposta no artigo *Attention Is All You Need* (Fonte: [60]).

tar as partes relevantes da sentença de entrada, de forma similar ao funcionamento do mecanismo de atenção original descrito na seção 2.3.3.2.

Por fim, um detalhe na arquitetura é que existe uma conexão residual em volta das camadas de autoatenção e de *feedfoward*, somando as saídas das camadas com suas entradas, seguida por um passo de normalização, onde cada entrada é normalizada para ter média zero e variância unitária [61].

Em redes neurais profundas [13], conexões residuais são conexões que passam informação de uma camada inferior para uma superior, sem passar por uma camada intermediária, o que auxilia na melhoria do aprendizado da rede.

Já a normalização de camada tem como objetivo melhorar o desempenho do treinamento nessas redes, mantendo os valores de uma camada oculta em uma faixa que facilite o treinamento baseado em gradiente.

Atenção Da forma proposta, a arquitetura permite uma paralelização significativamente maior, sendo o primeiro modelo de transdução que depende inteiramente do meca-

nismo de atenção chamado de autoatenção (*self-attention*) para calcular representações de suas entradas e saídas, sem usar convoluções ou RNNs alinhados a sequências.

Às vezes chamada de intra-atenção, a autoatenção é um mecanismo de atenção que relaciona diferentes posições de uma única sequência a fim de calcular uma representação da sequência [60].

Na implementação desse mecanismo, é calculado uma espécie de índice de atenção para cada palavra em uma sentença ou documento, a fim de modelar a influência que cada palavra tem sobre outra. Em função disso, a arquitetura *Transformer* permite muito mais paralelização do que redes CNN ou RNN, o que torna possível treinar eficientemente modelos muito grandes sobre grandes volumes de dados [67].

Como pode ser visualizado na Figura 2.16, ela captura o relacionamento entre diferentes elementos em uma mesma sequência, enfatizando aqueles que são mais relevantes entre si. Na arquitetura proposta, é chamada de *Scaled Dot-Product Attention*.

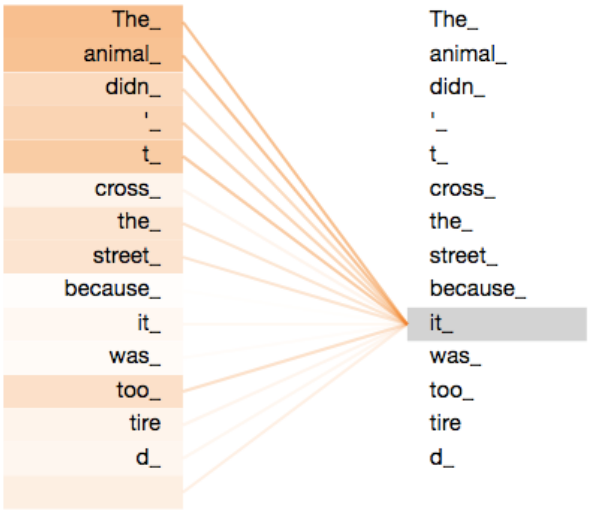

Figura 2.16: Visualização do funcionamento da autoatenção. Ela é o método que a arquitetura *Transformer* usa para construir o entendimento de outras palavras relevantes relacionadas à palavra sendo processada no momento (Fonte: [68]).

O modelo proposto [60] também define mais um mecanismo de atenção chamado de *Multi-Head Attention*, composto por várias cabeças de atenção (*attention heads*), cada uma sendo uma *Scaled Dot-Product Attention*. A arquitetura utiliza oito delas para cada codificador/decodificador. O objetivo desse mecanismo é permitir que o modelo trate de forma conjunta informações de diferentes subespaços de representação em posições distintas.

A Figura 2.17 ilustra o funcionamento do mecanismo *Multi-Head Attention*, mostrando a representação em dois subespaços diferentes. Na codificação da palavra *it*, a primeira

cabeça de atenção (laranja) dá mais ênfase em *The animal*, enquanto a outra (verde) está enfatizando a palavra *tired*.

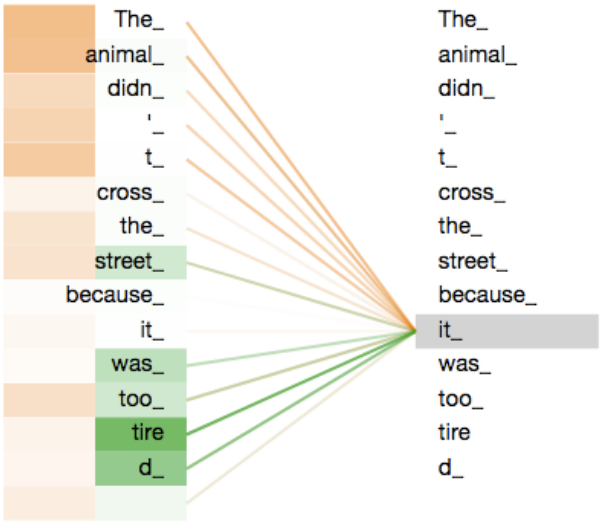

Figura 2.17: Visualização do funcionamento do mecanismo de autoatenção com várias cabeças (*Multi-Head Attention*) (Fonte: [68]).

Ao fim do processo, os resultados das cabeças de atenção são concatenados e multiplicados por uma matriz adicional de pesos com o propósito de gerar uma matriz a ser passada para a camada *feedfoward*.

No decodificador, a camada de autoatenção funciona de forma um pouco diferente do codificador. A ela é apenas permitido ter acesso a posições anteriores da sequência de saída. Isso é feito mascarando-se futuras posições na sequência de saída [68].

Redes *Feedfoward* por Posição Em adição às camadas de atenção, cada codificador e decodificador contém uma rede neural *feedfoward* totalmente conectada, aplicada a cada posição de entrada separadamente e de forma idêntica [60]. Resumidamente, o principal papel da pilha de codificadores [61] é atualizar as representações das entradas (*embeddings*) para produzir representações que codifiquem informação contextual à sequência.

***Embeddings* e Softmax** De forma similar a outros modelos de transdução de sequências, a arquitetura utiliza, como já visto, *embeddings* aprendidos para converter os *tokens* de entrada e de saída em vetores de dimensão pré-definida, no caso $d_{model} = 512$.

A arquitetura também prevê, após a pilha de decodificadores, uma camada de transformação linear e uma camada de aplicação da função softmax, utilizada para converter a saída do decodificador em probabilidades previstas para o próximo *token* [60].

A camada Linear é uma rede neural totalmente conectada simples que projeta o vetor produzido pela pilha de decodificadores em um vetor muito maior chamado vetor de logitos (*logit*).

A camada Softmax então transforma esses valores em probabilidades (todas positivas, somando 1). A posição do vetor com a maior probabilidade é escolhida e a palavra associada a ela é produzida como a saída corrente [68].

Codificação Posicional Dado que a arquitetura [60] do modelo não se baseia em recorrência, como em uma rede RNN, ou convolução, como em uma rede CNN, além do fato de que a ordem das palavras em uma sequência importa, para que o modelo possa saber essa ordem, alguma informação sobre a posição relativa ou absoluta dos *tokens* na sequência deve ser introduzida.

Para esse propósito, no início das pilhas de codificadores e decodificadores, vetores contendo as codificações posicionais (*positional encodings*) são adicionados a cada *embedding* de entrada.

Esses vetores seguem um padrão específico que o modelo aprende, o qual ajuda a determinar a posição de cada palavra ou a distância entre palavras na sequência [68] e possuem a mesma dimensão dos *embeddings* para que eles possam ser somados.

2.3.4 BERT

Os últimos anos foram marcados pela proliferação de múltiplos modelos de linguagem pré-treinados destinados ao ajuste fino para tarefa de classificação de texto. Entre esses tantos, *Bidirectional Encoder Representations for Transformers* (BERT) permanece o mais popular e amplamente utilizado modelo [24].

Sua principal característica é a bidirecionalidade. Diferentemente da arquitetura *Transformer* original, ele [69] é projetado para pré-treinar representações profundas bidirecionais a partir de texto não rotulado, condicionando, simultaneamente, o contexto à esquerda e à direita em todas as camadas.

Como resultado, como ilustrado na Figura 2.18, um modelo BERT pré-treinado pode sofrer ajuste fino de forma mais simples, propiciando a criação de modelos para uma variedade ampla de tarefas em PLN, sem a necessidade de mudanças substanciais na arquitetura para execução dessas tarefas, inclusive classificação de texto [70, 71, 72].

Durante o pré-treinamento, o modelo [69] é treinado em dados não rotulados e diferentes tarefas de pré-treinamento. Além disso, ele tem dois objetivos: prever *tokens* mascarados em textos e determinar se um trecho de texto é provável vir após outro. A primeira tarefa é chamada de modelagem de linguagem mascarada (*masked language mo-*

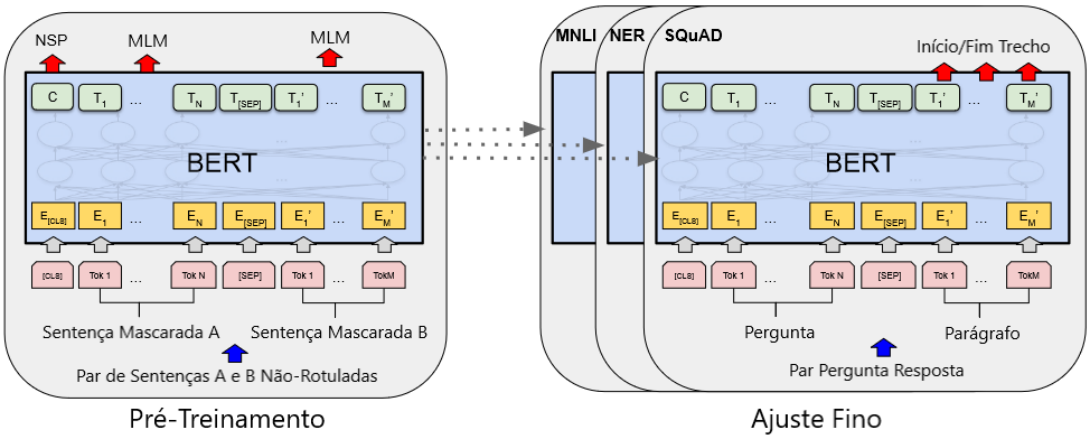

Figura 2.18: Procedimento geral de pré-treinamento e ajuste fino para BERT (Fonte: [69]).

deling - MLM), e a segunda é a previsão da próxima sentença (*next sentence prediction* - NSP).

Para o ajuste fino, o modelo é primeiro inicializado com os parâmetros pré-treinados. A seguir, os parâmetros sofrem ajuste fino utilizando-se dados rotulados das tarefas alvo. Cada tarefa alvo possui modelos com ajustes finos diferentes, mesmo que sejam inicializados com os mesmos parâmetros pré-treinados. Uma característica distinta do BERT é sua arquitetura unificada em diferentes tarefas. No fim do processo, diferença é mínima entre a arquitetura pré-treinada e a arquitetura final para as tarefas subsequentes.

2.3.4.1 Arquitetura do Modelo

A arquitetura do modelo BERT [69] é composta por uma pilha de codificadores *Transformer* [60] bidirecionais, com uma implementação quase idêntica à original. As diferenças estão basicamente no número de codificadores empilhados, no tamanho dos *embeddings* e no número de cabeças de atenção em cada mecanismo *Multi-Head Attention*.

Além disso, a arquitetura proposta possui duas versões com tamanhos diferentes: BERT_{BASE} e BERT_{LARGE}. A Tabela 2.1 contém a diferença entre a arquitetura Transformer original e as duas versões da arquitetura BERT propostas.

Tabela 2.1: Comparação entre as arquiteturas *Transformer* e BERT (Fonte: [60, 69])

	Codificadores	**Tam. *Embeddings***	**Cab. de Atenção**
Transformer	6	512	8
$\textbf{BERT}_{BASE}$	12	768	12
$\textbf{BERT}_{LARGE}$	24	1.024	16

Por fim, uma questão fundamental com os *transformers* é que o tamanho da camada de entrada dita a complexidade do modelo. Ambos os requisitos de tempo e memória apresentam crescimento quadrático em função do tamanho da entrada. Desta forma, é necessário definir tamanho de entrada fixa, suficiente para prover contexto suficiente para que o modelo funcione e ainda continue computacionalmente tratável. Para o *BERT*, o tamanho máximo de entrada utilizado foi de 512 *tokens* [13].

Capítulo 3

Revisão da Literatura

O problema de classificação de textos já foi abordado de várias formas na literatura. Contudo, a pesquisa realizada foi direcionada para trabalhos científicos que tivessem similaridade com a contextualização feita para este estudo ou com a hipótese de pesquisa apresentada. Ambos os critérios aplicados reduziram significativamente o volume de estudos encontrados sobre o tema.

Assim sendo, entende-se que o presente trabalho tenha a capacidade de colaborar com a comunidade científica e também, especialmente, com a disseminação do conhecimento entre os ramos e unidades do MP Ministério Público brasileiro, sem prejuízo de que atinja outros órgãos da administração pública que possuam problemática similar. Neste contexto, esta seção apresenta os trabalhos relacionados à presente dissertação.

A problemática de leitura e classificação de petições vivida pelo CNMP não é algo único dentro do sistema de justiça brasileiro, sendo ainda mais intensamente percebida dentro do Poder Judiciário. Em termos absolutos, por se tratar do poder responsável pela resolução de conflitos dentro da sociedade, o número de petições atinge volumes muito maiores em cada um dos tribunais de justiça espalhados pelo país.

Segundo o sumário executivo do anuário Justiça em Números, publicado pelo Conselho Nacional de Justiça (CNJ), em relação ao levantamento 2.024, ano-base 2.023, registrou-se cerca de 35,3 milhões de novos casos [73], que provavelmente incluem petições iniciais na grande parte dos processos, ou mesmo em sua totalidade.

Debruçando-se sobre o problema no contexto do Poder Judiciário brasileiro, face à escassez de recursos, Marinato et al. [74] propõem, como forma de classificação das petições iniciais, a utilização de classificador combinado do tipo *stacking*, onde um ou mais classificadores são combinados em níveis distintos, a fim de melhorar, no fim da pilha, o resultado da predição. No estudo, o classificador combinado foi aplicado sobre uma base de dados de treinamento com 1.787 modelos de petições, construída a partir de diversos portais com conteúdo jurídico, com modelos distintos, todas em formato *Portable Document*

Format (PDF). Durante o experimento realizado pelos pesquisadores, o primeiro passo constituiu-se na realização do pré-processamento do texto das petições iniciais, com subsequente vetorização de palavras, utilizando-se TF-IDF como técnica. Após isso, os dados foram treinados e validados com validação cruzada, com método de amostragem *k-fold* com $k = 10$. Os classificadores utilizados foram Regressão Logística, Floresta Aleatória, SVM e XGBoost. O processo foi repetido com o classificador combinado, utilizando-se todos esses classificadores, como ilustra a Figura 3.1.

Figura 3.1: Classificador combinado (Fonte: [74]).

Para avaliação do desempenho dos classificadores, tanto isoladamente, como combinados, os pesquisadores utilizaram as métricas: acurácia, precisão, *recall* e *F1-score*. Os resultados foram reproduzidos na Tabela 3.1.

Tabela 3.1: Resultados para as medidas de desempenho adotadas (Fonte: [74]).

Algoritmo	**Acurácia**	**Precisão**	***Recall***	***F1-Score***
Regressão Logística	83%	89%	74%	78%
Floresta Aleatória	86%	90%	82%	84%
SVM	88%	86%	74%	77%
XGBoost	83%	81%	76%	77%
Classificador Combinado	90%	91%	87%	89%

Embora o resultado apresentado tenha sido bastante satisfatório, não foi abordado como a questão do desbalanceamento dos dados foi tratada e se foi realmente um problema no decorrer do experimento.

Sousa [75] também aborda a problemática de classificação de petições iniciais no contexto do processo judicial eletrônico do Poder Judiciário do Estado de Tocantins, dada a escassez de recursos e aumento da demanda judicial. O estudo propôs uma ferramenta, denominada MINERJUS, utilizando técnicas de PLN e aprendizado de máquina, por meio de algoritmo de classificação supervisionada. O objetivo principal da ferramenta é o apoio

aos operadores do direito que efetuam a autuação de processos judiciais, sugerindo um assunto ao processo.

Sobre a base de dados dos experimentos do trabalho de Sousa [75], a partir de um *corpus* de treinamento devidamente preparado e rotulado por equipe do Tribunal de Justiça do Estado de Tocantins, contendo 897 petições iniciais em formato PDF, realizou-se o pré-processamento e a extração de características, utilizando-se TF-IDF como técnica. Ato contínuo, a matriz contendo cada vetor de texto do *corpus* foi então submetida a treinamento e, após, a validação de 78 petições, para avaliação do desempenho dos classificadores. Os quatro algoritmos de classificação diferentes utilizados, para comparação de resultados, foram: Árvore de Decisão, *Naïve Bayes*, SVM e k-NN. A Tabela 3.2 reproduz a avaliação realizada sobre o desempenho dos algoritmos.

Tabela 3.2: Avaliação dos algoritmos de aprendizagem supervisionada (Fonte: [75]).

Classificador	**Precisão**	**Acurácia**
Árvore de Decisão	57,89%	79,48%
NB	53,78%	57,69%
SVM	**72,72%**	**93,58%**
k-NN	51%	82,05%

Concluindo sua pesquisa, o autor aponta a utilização de SVM como algoritmo adotado na classificação de petições da solução final, que ainda possui interface web para inserção de documentos e exibição da predição do algoritmo treinado. Embora o resultado do trabalho tenha apontado na direção da capacidade da solução em mitigar problemas no ato da classificação processual, não fica claro se o volume de dados treinado e, em especial, o validado, é suficiente para que se considere a solução adequada. Há indícios de *overfit* do modelo ajustado escolhido.

Também não foram mencionadas quais medidas técnicas, se foram aplicadas, para tratamento do desbalanceamento entre as classes. Além disso, fica subentendido que a questão tempo tenha sido fator influenciador no volume de documentos do *corpus*. Isso posto, pode-se dizer que essas dificuldades percebidas também são fatores importantes a ser considerados na pesquisa aqui desenvolvida.

Ainda sobre a tarefa de classificação de texto no Poder Judiciário, motivados pela dificuldade de acesso a informações sobre dados jurídicos de modo a permitir uma abordagem estatística que possibilite entender a relação entre os processos e as variáveis que os levam a classificação de sentenças, Almeida Neto et al. [76], em trabalho realizado anos antes, abordaram a inferência de resultados processuais pela procedência, procedência parcial ou improcedência em julgamentos de magistrados, utilizando-se classificadores baseados em

árvore de decisão e redes neurais artificiais sobre os processos em tramitação na justiça estadual brasileira, a partir de informações extraídas de causas jurídicas.

Como ponto de partida, os pesquisadores concentraram-se na montagem da base de dados utilizada nos algoritmos classificadores. Para isso, os dados foram extraídos dos portais dos Tribunais de Justiça a partir de técnicas de PLN como *web scraping* e expressões regulares.

Em seguida, já nas etapas de pré-processamento e engenharia de características, os dados foram organizados, enriquecidos e rotulados de acordo com o indicador de êxito (procedente, parcialmente procedente, improcedente), tendo sido aplicada a técnica de *one hot encoding* para transformação das variáveis categóricas.

Para realização do experimento, foram utilizadas duas etapas para os classificadores baseados em árvore de decisão, C4.5 e *Recursive Partitioning and Regression Trees* (PART), e rede neural artificial, *Multilayer Perceptron* (MLP). Uma etapa compreendeu o uso de base de dados com atributos reduzidos e, outra, utilizando base com mais atributos. Em cada uma delas, executou-se 30 vezes o processo de treinamento, teste e validação, utilizando-se validação cruzada, com método de amostragem *k-fold* com $k = 3$.

Os autores do trabalho não deixam claro em sua conclusão os resultados de qual etapa foram considerados melhores em sua avaliação final, embora presuma-se que sejam aqueles correspondentes à segunda etapa, com médias na casa de 74% para as árvores de decisão e de 76% para a rede neural artificial. Contudo, indicam a dificuldade da extração de texto e transformação em uma base jurídica que convergisse em resultados confiáveis e aceitáveis para padrões técnicos e necessidades corporativas e acadêmicas.

Acrescentam também suas considerações sobre a redução de eficácia na aplicação de técnicas de aprendizado de máquina em informações de natureza jurídica, em especial aquelas com alto grau de subjetividade. Isso leva a crer que as técnicas utilizadas talvez não sejam as melhores para o tipo de problema que se deseja resolver no presente estudo.

Trazendo a problemática de classificação de petições iniciais para o âmbito do MP brasileiro, Noguti, Vellasques e Oliveira [77], citando como desafio a falta de treinamento jurídico adequado pelo corpo de servidores, além da falta de protocolo unificado para inserção de dados no sistema de controle processual, propõem técnicas de PLN e de aprendizado de máquina para classificação automática da classe das petições, com o intuito de possibilitar melhor alocação de recursos, redução de tempo e geração de estatísticas mais confiáveis.

Após a realização de pré-processamento do texto, incluindo o processo de lematização de palavras, realizou-se a engenharia de características a partir do *corpus* de 17.740 petições iniciais de 18 classes processuais distintas, recebidas entre 2.016 e 2.019. Para este processo, a fim de comparação futura dos resultados, utilizaram-se as técnicas de PLN

baseadas em TF-IDF e *Word Embeddings*.

O passo final foi realizar o processo de treinamento, validação e teste com classificadores supervisionados, no caso, Regressão Logística, SVM, Floresta Aleatória, *Gradient Boosting* e redes neurais, mais especificamente CNN e RNN. Durante o experimento, foi detectado desbalanceamento entre as classes dos dados rotulados, o que levou os autores a aplicar a técnica de subamostragem aleatória para comparação com o conjunto de dados original.

A avaliação foi realizada utilizando-se as métricas de acurácia e *F1-Score*. Pela análise, os melhores resultados foram obtidos com uma combinação de *Word Embeddings* e RNN, mais especificamente, LSTM, levando a uma acurácia de 90% e *F1-Score* de 85%, com dados provenientes do conjunto original de documentos rotulados, ou seja, em detrimento da técnica de subamostragem, que se mostrou menos eficiente.

O artigo mostra muita relação com o estudo aqui desenvolvido, dado que pode ser utilizado como base de comparação direta de resultados, como fonte de modelos pré-treinados para transferência de aprendizado ou como ponto de partida para o estudo aqui realizado, dado que os pesquisadores apontam no sentido de evoluir sua pesquisa aplicando técnicas que façam uso de modelos de linguagem e arquiteturas de *Transformers*, como BERT, que é a proposta da presente dissertação.

Ainda sobre BERT, este *Transformer* foi uma das técnicas de PLN utilizada por Leme [78] em sua dissertação. A problemática, também relacionada à classificação de texto, envolve o grande esforço humano necessário para a classificação do rito processual de processos de atos de concentração abertos no Conselho de Administrativo de Defesa Econômica (CADE), que pode ser sumário ou ordinário, feita manualmente. Assim sendo, a automatização desta tarefa é desejável em função da economia de recursos gerada.

O conjunto de dados usado no estudo consistiu em 1.275 documentos coletados de processos, sendo 217 (17%) de rito ordinário e 1.058 (83%) de rito sumário, portanto, bastante desbalanceado. Além da extração, a título de pré-processamento, foi aplicada a conversão de documentos digitalizados em documentos de texto, com o uso de ferramenta de *Optical Character Recognition* (OCR), ou Reconhecimento Óptico de Caracteres em português.

A partir da base original, foram selecionados, aleatoriamente, 30% dos documentos para validação dos modelos treinados (técnica de validação chamada *hold-out*). Sobre o restante, foi aplicada validação cruzada com método de amostragem *k-fold* com $k = 5$.

Várias técnicas foram usadas no estudo. As de PLN incluíram BOW, *Word Embeddings* (*Skipgram*) e BERT. Como algoritmos de aprendizado de máquina/aprendizado profundo, utilizaram-se Regressão Logística, CNN, LSTM e BiLSTM. A pesquisa também avaliou a combinação dessas técnicas para melhorar o desempenho e utilizou as técnicas de BOW

e Regressão Logística como linha de base.

A avaliação foi feita usando métricas como acurácia, precisão, *recall* e *F1-Score*. Os resultados mostraram que os modelos combinados, baseados em BERT, foram superiores ao modelo adotado como linha de base, melhorando o desempenho dos classificadores binários.

O problema do desbalanceamento encontrado, e não resolvido, pode ter levado a mudanças nos resultados, embora tenha sido aplicada, sem sucesso, técnica de subamostragem. O pesquisador também relata a dificuldade quanto ao tamanho dos documentos, o que também pode ter levado a mudanças nos resultados. Isso posto, o trabalho mostra-se de grande relevância para a pesquisa desenvolvida nesta dissertação.

Capítulo 4

Metodologia

Esta seção apresenta a metodologia utilizada para a realização do experimento objeto da pesquisa desenvolvida nesta dissertação de mestrado. Este experimento foi baseado no estudo de caso do CNMP que versa sobre classificação automatizada de petições iniciais. Cumpre ressaltar que esta é uma iniciativa inovadora no âmbito do Ministério Público brasileiro, dado que, durante a revisão da literatura, não foram encontradas publicações específicas sobre esse tipo de tarefa. No campo prático, nas diversas interações entre o CNMP e os órgãos do MP, também se observa um campo de estudo e aplicação ainda incipiente.

Em termos das técnicas, pretendeu-se, com o experimento realizado, adotar práticas consideradas como estado da arte na tarefa de classificação de texto, comparando-se o desempenho entre elas, obtendo-se, no fim, aquele conjunto que trouxesse o melhor resultado em termos negociais e eficiência computacional.

Assim sendo, sobre a metodologia adotada nesta pesquisa, de forma genérica, uma das primeiras etapas envolveu a aquisição dos dados para utilização no estudo, incluindo uma análise exploratória prévia, a conversão de documentos digitalizados em textos que pudessem ser processados por técnicas de PLN e a realização do pré-processamento dos textos, com o objetivo de padronizá-los e melhorar o desempenho dos modelos a serem desenvolvidos. Os dados foram então separados em conjuntos de dados, com a realização de balanceamento e conversão textual em representações vetoriais numéricas e densas. Por fim, foram realizadas as atividades de modelagem de aprendizado de máquina e aprendizado profundo, bem como a validação dos modelos, testando-se diferentes técnicas para inferir as classes das petições com base nas características dos documentos analisados.

4.1 Aquisição de Dados

Os dados textuais utilizados ao longo de toda a pesquisa são provenientes de petições iniciais em formato PDF autuadas em processos no CNMP. Esses arquivos estavam armazenados inicialmente no sistema de arquivos do órgão e foram obtidas cópias para utilização na pesquisa.

Essas petições estão referenciadas na base de dados do sistema processual eletrônico do Conselho Nacional do Ministério Público, chamado sistema ELO, de onde foram extraídas também as informações dos rótulos, neste caso, as classes processuais correspondentes. Cabe ressaltar que, em conformidade com os normativos existentes, só foram selecionados processos que não possuíam nenhum tipo de sigilo ou continham petições criptografadas.

4.1.1 Análise Exploratória

Em posse dos dados descritos brevemente acima, realizou-se primeiramente uma análise exploratória com o objetivo de avaliar a distribuição de frequência entre as classes processuais, bem como para avaliar a qualidade e a estrutura dos dados textuais contidos nos arquivos das petições obtidas.

Em termos quantitativos, a base de dados totalizou 9.390 processos autuados em 24 das 25 classes processuais existentes, no período compreendido entre 1º de junho de 2015 e 15 de março de 2024. A classe processual Restituição de Autos não possuía nenhum processo autuado em todo o período mencionado, tendo sido desconsiderada para o propósito da pesquisa. A Figura 4.1 ilustra a distribuição de frequência de petições entre as classes processuais.

A avaliação qualitativa constituiu-se basicamente pela contagem do número de palavras por documento e pela verificação da estrutura textual das petições, e foi conduzida com auxílio da biblioteca *PyPDF2*[1], ferramenta escrita em *Python* para a extração de texto de documentos no formato PDF.

Como um dos pontos mais relevantes identificados, constatou-se que um total de 2.210 petições (23,54% do total) não continham dados textuais. Ou seja, eram documentos cuja extração do conteúdo não foi possível pela ausência de dado textual processável, possivelmente em virtude do processo de digitalização adotado originalmente ter produzido imagens em vez de texto.

Adicionalmente, em alguns casos, foi observada a existência de documentos que apresentavam montagens de recortes de imagens contendo textos, além de outros elementos não textuais, como, por exemplo, gráficos e outros tipos de imagens. Isso dificultou ainda mais a obtenção adequada dos dados textuais. Outro obstáculo encontrado foi a difi-

[1]Disponível em `https://pypdf2.readthedocs.io/en/3.x/`.

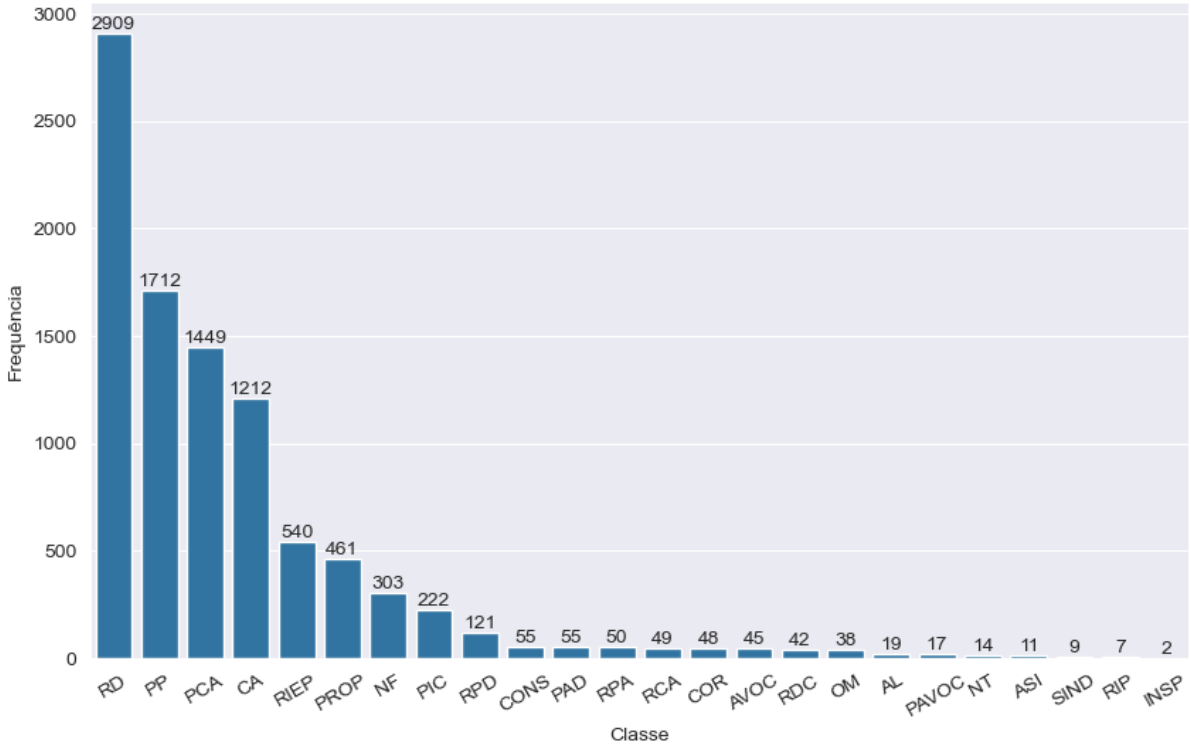

Figura 4.1: Distribuição de petições iniciais entre as classes.

culdade de leitura de documentos com estruturas mais elaboradas, como tabelas, o que limitou a capacidade de extração automatizada de texto desses arquivos, muitas vezes causando truncamento dos dados.

O extrato da avaliação qualitativa está apresentado na Tabela 4.1. Nela é possível observar as classes processuais com suas respectivas frequências de petições, além de algumas estatísticas obtidas com base na contagem de palavras dessas petições, cujos textos puderam ser extraídos sem erros pela ferramenta de extração.

Dessa forma, como uma das primeiras conclusões da análise exploratória, verificou-se a necessidade de refinamento das técnicas de extração de texto, demandando a utilização de métodos complementares, como OCR, para melhorar a qualidade do conteúdo textual obtido.

4.1.2 Reconhecimento Óptico de Documentos

Para otimizar a extração do texto das petições, realizou-se o processo de reconhecimento óptico (OCR) de arquivos PDF utilizados no estudo. Optou-se, para fins de padronização, que todas as petições, mesmo que não possuíssem problemas de extração de texto, fossem submetidas à técnica, dado que o processo original de criação do arquivo era desconhecido.

Tabela 4.1: Número de palavras por Classe Processual.

Sigla	Classe	Quant.	Número de Palavras		
			Média	Máx.	Des. Pad.
INSP	Inspeção	2	882	0	0
COR	Correição	48	2.323,19	71.826	10.255,73
RD	Reclamação Disciplinar	2.909	3.685,26	657.804	20.615,32
SIND	Sindicância	9	2.651,44	11.560	4.695,68
RIEP	Representação por Inércia ou Excesso de Prazo	540	2.244,38	247.008	11.663,55
PAD	Processo Administrativo Disciplinar	55	3.480,87	25.396	4.941,79
AVOC	Avocação	45	5.149,16	39.547	8.170,81
RPD	Revisão de Processo Disciplinar	121	9.878,56	239.224	27.025,34
RPA	Reclamação para Preservação da Autonomia do Ministério Público	50	8.469,76	92.114	16.709,90
RCA	Reclamação para Preservação da Competência e da Autoridade das Decisões do Conselho	49	11.313,24	312.769	44.744,33
PCA	Procedimento de Controle Administrativo	1.449	6.269,39	596.526	24.818,54
ASI	Arguição de Impedimento ou Suspeição	11	5.057,36	16.452	5.969,79
RA	Restauração de Autos	0	0	0	0
PP	Pedido de Providências	1.712	3.233,45	214.259	11.509,77
RIP	Remoção por Interesse Público	7	8.494,86	30.726	11.051,92
PROP	Proposição	461	4.573,34	355.385	20.208,44
RDC	Revisão de Decisão do Conselho	42	4.287,74	38.529	8.070,18
PAVOC	Procedimento Avocado	17	30.322,53	496.703	120.202,76
CONS	Consulta	55	773,22	5.873	1.313,31
PIC	Procedimento Interno de Comissão	222	9.389,71	243.181	25.220,84
NT	Nota Técnica	14	2.371,57	9.906	3.835,02
AL	Anteprojeto de Lei	19	5.687,79	20.941	7.962,39
NF	Notícia de Fato	303	1.982,41	63.673	5.594,54
CA	Conflito de Atribuições	1.212	16.111,01	2.380.149	79.225,37
OM	Ordem do Mérito	38	5.420,55	42.487	8.301,80

Como abordagem inicial, optou-se pelo uso da biblioteca *Python Tesseract*[2], que encapsula o motor da ferramenta de código aberto *Tesseract-OCR*[3]. As páginas dos documentos foram todas convertidas em arquivos de imagem e submetidas à referida biblioteca de OCR para conversão em texto. Contudo, ao fim do processo, não foram obtidos bons resultados no geral.

Diante disso, testou-se a solução baseada no serviço *Azure AI Document Intelligence*[4] da *Microsoft Azure*. Seu uso compreendia o envio do arquivo PDF por chamada a API,

[2]Disponível em `https://github.com/madmaze/pytesseract`.

[3]Disponível em `https://github.com/tesseract-ocr/tesseract`.

[4]Disponível em `https://azure.microsoft.com/en-us/products/ai-services/ai-document-intelligence`.

que, a seu turno, devolvia objetos contendo as informações textuais enriquecidas com metadados sobre a estrutura do documento convertido.

Comparando-se com os resultados da biblioteca *Python Tesseract*, o serviço em nuvem apresentou desempenho significativamente melhor, com maior precisão na extração de texto, mesmo em documentos de baixa qualidade visual. A ferramenta foi então utilizada para a totalidade das petições, garantindo-se extração de texto com muito melhor qualidade, adequado para utilização na continuidade da pesquisa.

A fim de otimizar o uso de recursos financeiros e computacionais, os resultados da conversão dos arquivos das petições em texto, retornados pelo serviço como objetos em memória, foram serializados em disco como representações binárias. O objetivo do procedimento descrito foi possibilitar a leitura e manipulação rápida do conteúdo textual nas etapas subsequentes do trabalho, sem a necessidade de reproduzir o processo de OCR, evitando-se, dessa forma, custos adicionais com novas chamadas à API e garantindo-se a reprodutibilidade dos resultados obtidos.

4.1.3 Pré-Processamento de Texto

Como já mencionado, o serviço de OCR da *Microsoft Azure* devolve o texto com informações relativas à estrutura do documento submetido ao reconhecimento óptico. Essa estrutura basicamente contempla: parágrafos, tabelas, figuras, etc. Alguns desses itens, embora relevantes no contexto original dos documentos, representavam ruído no processo de análise e modelagem textual.

Dessa forma, apenas parágrafos foram mantidos na obtenção dos textos, facilitando o processamento posterior. Adicionalmente, dentro dos parágrafos, textos marcados pela ferramenta como cabeçalho de página, rodapé, número de página e nota de rodapé também foram ignorados.

Além das remoções realizadas na estrutura do documento representada no objeto retornado pelo serviço *Azure AI Document Intelligence*, foi aplicada uma camada adicional de filtragem, utilizando-se expressões regulares para eliminar elementos indesejados como *tags* HTML, endereços de e-mail, URLs, *emojis* e outros componentes que não agregavam valor ao texto analisado.

Ao final do processo de limpeza, o texto foi gravado em arquivos com extensão .txt, totalizando 9.387 petições processadas com sucesso. Embora o volume inicial de documentos fosse de 9.390, dois arquivos não possuíam texto e um apresentou erro e não pôde ser processado pelo serviço de conversão por OCR.

Com o texto devidamente limpo, foi necessária preparação adicional para possibilitar a separação dos conjuntos de dados para o desenvolvimento dos passos seguintes da pesquisa. Essa preparação envolveu a criação de duas versões dos dados textuais: a primeira baseada

no texto original das petições e a segunda produzida a partir de sumarização dos textos originais, caracterizando-se pela utilização de dados mais sucintos e concentrados (texto reduzido).

A intenção dessa abordagem dupla foi possibilitar comparar o desempenho dos modelos de classificação sobre todo o *corpus* de documentos, já que algumas técnicas não permitiriam o uso dos dados mais densos, o que levaria a uma análise incompleta. Com isso, foi possível avaliar como as duas representações do mesmo conteúdo poderiam impactar os resultados dos diferentes métodos de classificação das petições.

4.1.3.1 Preparação do Texto Original

Para a versão com o texto original das petições, a preparação envolveu a tokenização do texto, removendo-se *stopwords* (palavras que não carregam significado semântico relevante) e aplicando-se a lematização (redução das palavras em suas formas canônicas) das palavras. Essas técnicas também foram aplicadas para garantir que os modelos de classificação pudessem focar nos aspectos mais importantes dos textos, eliminando-se ruído e redundância. Essas tarefas foram realizadas com o auxílio da ferramenta de PLN *spaCy* para *Python*. Ao final do processo, todos os 9.387 textos originais haviam sido pré-processados.

4.1.3.2 Preparação do Texto Reduzido

Como já evidenciado pelos dados da Tabela 4.1, observou-se um grande volume textual nos documentos obtidos, que possuíam, em média, 5.836,67 palavras. Para as fases seguintes da pesquisa, pensando-se no uso de métodos tradicionais de vetorização, como TF-IDF por exemplo, esse número não constituiria grande barreira para o processo de conversão do texto em sua representação numérica.

Contudo, no caso da utilização dos modelos baseados em BERT para tarefa de classificação, tais textos precisariam ser truncados quando excedessem o tamanho máximo da janela de contexto da arquitetura de 512 *tokens*, o que acarretaria perda de informação textual relevante para o processo de classificação. Ressalta-se ainda que estes modelos utilizam dois *tokens* especiais, [CLS] e [SEP], para delimitar o início e fim de uma sentença respectivamente, o que resultaria em um tamanho líquido de 510 *tokens* a ser trabalhada na tarefa de classificação de texto.

Embora uma conversão direta do número de palavras para o número de *tokens* seja algo difícil de se obter, já que depende do tipo de tokenizador e da língua do vocabulário do modelo, adotou-se como estimativa que 100 *tokens* equivaleriam a aproximadamente 75 palavras [79]. Isso significa dizer que o conjunto de petições da pesquisa possuiria,

obedecendo-se a essa taxa de conversão, uma média aproximada de 7.782,23 *tokens*, ou seja, mais de 15 vezes maior do que a supracitada janela de contexto.

Algumas estratégias foram cogitadas em relação ao tratamento da restrição do número máximo de *tokens* [72] para utilização em classificadores BERT. Uma delas consistia na utilização dos primeiros 510 *tokens*, outra utilizaria os 510 *tokens* finais e uma terceira possível consistia em uma estratégia híbrida, utilizando os 128 *tokens* iniciais e os 382 finais.

Ainda que pudessem funcionar em alguns cenários, levando-se em consideração os números de palavras/*tokens* observados no *corpus* de documentos, essas estratégias não se aplicariam para a maioria dos textos obtidos. Nenhuma delas seria plenamente capaz de captar a ideia central do texto em documentos significativamente maiores do que a janela de contexto, que, ainda por cima, não possuía nenhum padrão de disposição de conteúdo pré-definida pela área negocial do CNMP. Isso, sem sobra de dúvida, prejudicaria a análise de desempenho dos classificadores baseados em BERT em relação aos modelos tradicionais.

Contudo, de alguma forma seria necessário capturar a essência do documento. Assim sendo, fez-se necessária a preparação de uma versão reduzida dos textos das petições, utilizando-se a tarefa de sumarização abstrativa, um processo em que é gerada uma nova versão condensada do texto original, que transmite as ideias mais importantes, mas com estrutura textual própria. Essa técnica difere da sumarização extrativa, onde trechos do conteúdo original considerados mais importantes são selecionados e agrupados para compor a versão resumida. Cabe ressaltar que, após a análise de algumas amostras de textos sumarizados, o conteúdo foi considerado satisfatório para a continuidade da pesquisa.

Seleção dos documentos a serem sumarizados Dada a limitação da janela de contexto dos modelos *BERT*, todos os textos originais de petições que excederam o limite de 512 (incluindo os *tokens* especiais) foram submetidos ao processo de sumarização abstrativa. Para que o critério pudesse ser aplicado, foi necessário tokenizar o texto e contar o número *tokens* dos textos originais já limpos das petições. Para isso, aplicaram-se os tokenizadores dos modelos BERT utilizados como referência para a tarefa de classificação de texto que seriam utilizados nas etapas posteriores da pesquisa.

Os dois modelos em língua portuguesa (português brasileiro) utilizados foram: BERTimbau[5] [80] (versão com 350 milhões de parâmetros) e Albertina PT-BR[6] [81] (versão com 100 milhões de parâmetros). Ressalta-se que ambos utilizam o tokenizador *SentencePiece* [82], embora o BERTimbau converta o vocabulário obtido para o formato *WordPiece*

[5]Disponível em `https://huggingface.co/neuralmind/bert-base-portuguese-cased`
[6]Disponível em `https://huggingface.co/PORTULAN/albertina-100m-portuguese-ptbr-encoder`

[83] para manutenção da compatibilidade com o código BERT original. Alguns dos hiperparâmetros desses modelos estão listados na Tabela 4.2.

Tabela 4.2: Comparação entre hiperparâmetros dos modelos BERTimbau e Albertina PT-BR (Fonte: [80, 81])

	Codificadores	**Tam. *Embeddings***
BERTimbau	24	1.024
Albertina PT-BR	24	1.536

Como dito anteriormente, o critério adotado para a sumarização consistiu na seleção dos textos cujo número de *tokens* excedesse 512. Todos os textos foram tokenizados com ambos os modelos BERT e, por fim, adotou-se como padrão para seleção de textos a contagem de *tokens* produzida pelo modelo Albertina PT-BR, uma vez que seu tokenizador produziu maior número de *tokens* em relação ao BERTimbau.

Retrieval-Augmented Generation Embora existam modelos de PLN especializados na tarefa de sumarização de texto, optou-se em utilizar a técnica *Retrieval-Augmented Generation* (RAG) [84] para esse propósito. Ela combina a capacidade de geração de texto com a obtenção de conhecimento externo para aprimorar a relevância das respostas geradas por um *Large Language Model* (LLM).

Essa vantagem foi justamente o motivo de sua escolha, já que permitiu, no âmbito da pesquisa, o processamento de textos longos, obtendo o máximo de informação textual semanticamente relevante. De forma ampla, pode-se dizer que a técnica possui duas grandes etapas, a primeira responsável pela indexação da informação e a segunda por sua recuperação e geração do texto com o LLM. Na pesquisa, essas etapas foram executadas com o auxílio da biblioteca *LangChain*[7] para *Python*.

Na primeira etapa (Figura 4.2), o texto das petições selecionadas foi dividido em pedaços menores, também chamados de *chunks*, de até 2.048 caracteres. Após, esses *chunks* de texto foram submetidos, por chamada a API, ao serviço *Azure OpenAI*[8] da *Microsoft* para geração de *embeddings* que, por sua vez, foram gravados e indexados no banco de dados vetorial, no caso o *PostgreSQL* com a extensão *pgvector*.

Para a geração dessas representações vetoriais, utilizou-se modelo específico denominado *text-embeddings-large-3*, na versão 1. Esse modelo possui a capacidade de gerar vetores de dimensões variáveis, até o tamanho máximo de 3.072 dimensões. Contudo, pelas características do banco de dados vetorial utilizado, configurou-se o serviço para geração de vetores com o tamanho de 2.000 dimensões, tamanho máximo suportado pela supracitada extensão.

[7]Disponível em `https://www.langchain.com/langchain`.
[8]Disponível em `https://azure.microsoft.com/en-us/products/ai-services/openai-service`.

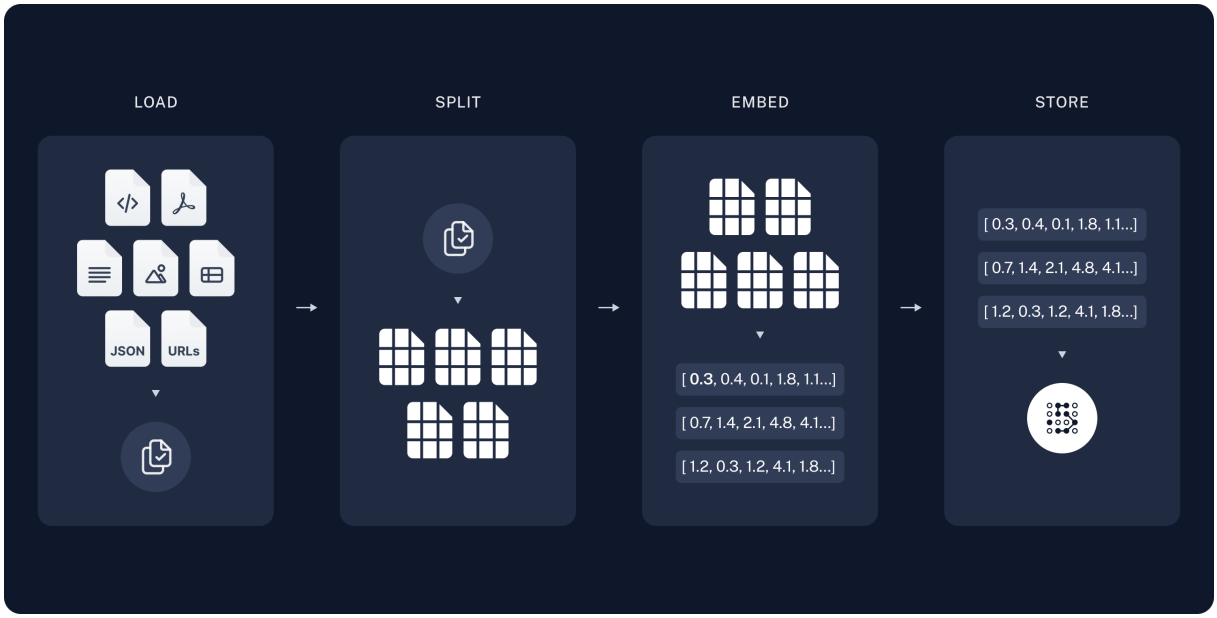

Figura 4.2: Etapa de indexação da informação da técnica RAG (Fonte: [85]).

Os *chunks* indexados tiveram ainda adicionada a informação de metadado do *checksum*[9] do arquivo *PDF* correspondente à petição inicial. O principal objetivo foi garantir que petições com nomes de arquivos diferentes, mas que eventualmente fossem o mesmo arquivo, não fossem indexadas em multiplicidade.

Para a tarefa de sumarização, parte da segunda etapa da técnica RAG (Figura 4.3), foi previamente configurado na biblioteca o *template* de *prompt* a ser passado ao final do processo para o LLM, cuja instrução era produzir respostas completas, com textos entre 200 e 300 palavras, utilizando como base o contexto fornecido. A seguir, a transcrição do *template* como usado na pesquisa:

```
"""Responda a pergunta usando o contexto informado.
   Responda da forma mais completa entre 200 e 300 palavras caso seja
   possível. \n\n
   Contexto: \n {context} \n
   Pergunta: \n {question} \n
   Resposta:
"""
```

[9]Código frequentemente usado para verificar a integridade de dados armazenados em algum meio [86]. Nesta pesquisa, esse código foi calculado utilizando-se a função de *hash* [87] SHA-256 da família SHA-2 (*Secure Hash Algorithm*) [88].

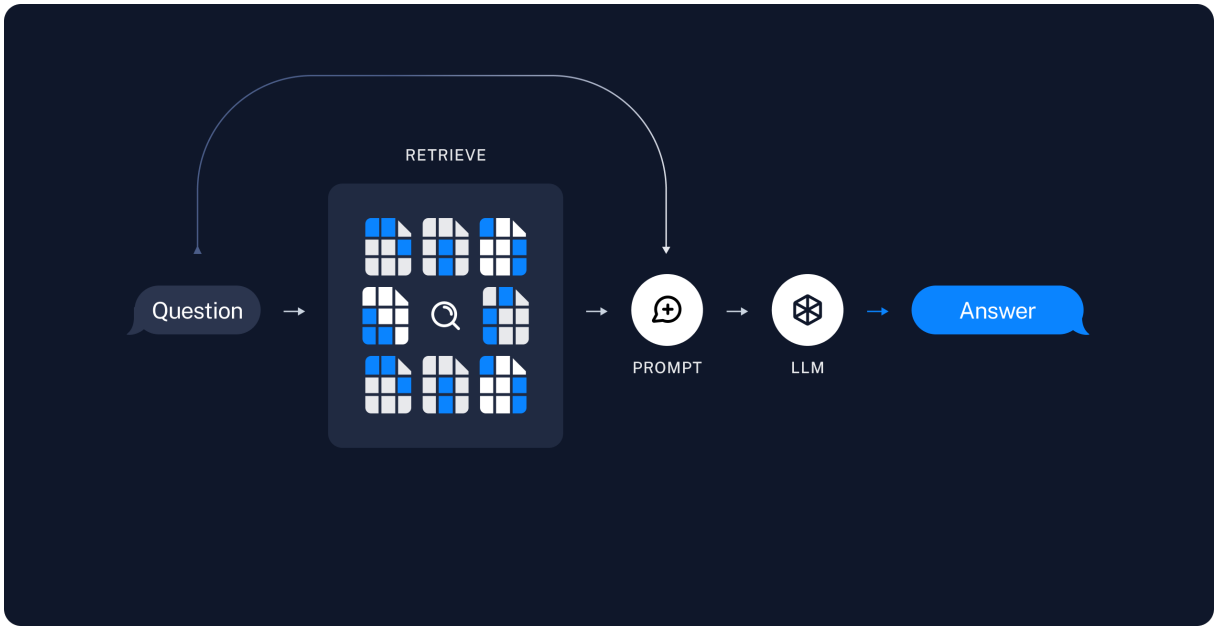

Figura 4.3: Etapa de recuperação da informação e geração de texto da técnica RAG (Fonte: [85]).

A cada petição que seria reduzida foram selecionados, a partir do banco de dados vetorial, os cinco *chunks* de texto mais relevantes por meio da busca por similaridade por cosseno[10] com o *embedding* retornado pelo modelo *text-embeddings-large-3* da pergunta:

```
"Qual o objeto do documento e a ação que se pede ao Conselho Nacional do
Ministério Público (CNMP)?"
```

Além disso, apenas os *chunks* relacionados à petição sendo resumida foram buscados no banco de dados vetorial. Para isso, previamente à busca por similaridade, a biblioteca recebeu como filtro de pesquisa o valor relativo ao *checksum* da petição, armazenado como metadado no procedimento de indexação dos *chunks*.

Os *chunks* de texto retornados foram então passados como parâmetro (*context*) para o *prompt* anteriormente configurado, assim como a pergunta utilizada na busca por similaridade no banco de dados vetorial (parâmetro *question*). Esse *prompt*, contendo todos os elementos necessários, foi finalmente fornecido ao LLM para a realização da sumarização. Como modelo de linguagem utilizou-se o GPT-4, também do serviço *Azure OpenAI*, cuja versão do modelo era a 0613. Para a realização do experimento, utilizaram-se os hiperparâmetros com os valores padrão para o modo de conversa (*chat*) do modelo, conforme demonstra a Tabela 4.3

[10] A similaridade por cosseno é uma medida da similaridade entre dois vetores em um espaço vetorial que avalia o valor do cosseno do ângulo compreendido entre eles [89].

Tabela 4.3: Hiperparâmetros dos modelos GPT-4 e GPT-3.5-Turbo

Hiperparâmetro	**Valor**
Mensagens anteriores incluídas	10
Resposta máxima	800
Temperatura	0,7
Top P	0,95
Penalidade por frequência	0
Penalidade por presença	0

Essa abordagem garantiu que o modelo focasse nas informações mais pertinentes ao conteúdo da petição e ao seu objetivo, reduzindo o texto de forma eficiente e preservando o contexto necessário para a tarefa de classificação que seria realizada nas etapas posteriores do estudo. O texto retornado foi então salvo em um novo arquivo com extensão .txt para garantir a reprodutibilidade dos resultados e evitar custos adicionais com o serviço em nuvem no caso de necessidade de nova execução do código. Dos 9.387 textos originais, 7 deles não puderam ser reduzidos por erros relacionados a filtros de conteúdo disparados pelo LLM.

Para que pudessem ser utilizados com modelos tradicionais de aprendizado de máquina, os textos reduzidos também passaram por processo de tokenização, com remoção de *stopwords* e aplicação de lematização das palavras, seguindo os mesmos procedimentos realizados na versão dos textos originais.

No caso dos dois modelos *BERT*, os textos reduzidos foram tokenizados para posterior utilização durante o ajuste fino para a tarefa de classificação de texto, utilizando-se os próprios tokenizadores dos modelos. Esses tokenizadores foram previamente ajustados para que, durante a tokenização, truncassem os textos que excedessem o comprimento máximo do respectivo modelo de 512 *tokens* (hiperparâmetro *truncation=True*) ou, caso abaixo desse comprimento, para que preenchessem a janela com o *token* especial [PAD] (hiperparâmetro *padding="max_length"*) até o limite superior.

4.2 Conjuntos de Dados

Os conjuntos de dados que foram utilizados para o ajuste dos modelos levaram em consideração as diferentes combinações de tamanho de textos e técnicas. Cada combinação visava explorar aspectos variados da estrutura textual e a sua influência nos resultados dos classificadores. Assim sendo, para a continuidade da pesquisa, como cenários para a comparação dos classificadores, os conjuntos de dados foram divididos da seguinte maneira:

1. Texto original desbalanceado / TF-IDF: conjunto de dados não balanceado, contendo as representações vetoriais de textos originais, geradas pelo método TF-IDF, e os rótulos das respectivas classes;

2. Texto reduzido desbalanceado / TF-IDF: conjunto de dados não balanceado, contendo as representações vetoriais de textos reduzidos, geradas pelo método TF-IDF, e os rótulos das respectivas classes;

3. Texto reduzido desbalanceado / *Embeddings*: conjunto de dados não balanceado, contendo *embeddings* de textos reduzidos e os rótulos das respectivas classes;

4. Texto reduzido desbalanceado / BERT: conjunto de dados não balanceado, contendo textos reduzidos e os rótulos das respectivas classes;

5. Texto reduzido balanceado / TF-IDF: conjunto de dados balanceado, contendo as representações vetoriais de textos reduzidos, geradas pelo método TF-IDF, e os rótulos das respectivas classes;

6. Texto reduzido balanceado / *Embeddings*: conjunto de dados balanceado, contendo *embeddings* de textos reduzidos e os rótulos das respectivas classes;

7. Texto reduzido balanceado / BERT: conjunto de dados balanceado, contendo textos reduzidos e os rótulos das respectivas classes.

4.2.1 Balanceamento das Classes

Uma característica observada nos dados obtidos para esta dissertação foi o desbalanceamento significativo entre as 25 classes presentes no conjunto de petições analisadas. Constatou-se que as cinco com maior frequência de petições iniciais (Reclamação Disciplinar, Pedido de Providências, Procedimento de Controle Administrativo, Conflito de Atribuições e Representação por Inércia ou Excesso de Prazo) representavam cerca de 79,5% do total de petições, sendo responsáveis por 7.819 dos 9.837 documentos coletados.

Esse desbalanceamento poderia afetar negativamente o desempenho dos modelos, especialmente aqueles sensíveis a desequilíbrios, gerando vieses e reduzindo a acurácia para classes menos representadas. Assim, foi necessário adotar técnicas de balanceamento de dados antes de prosseguir com a vetorização dos textos. Ressalta-se, contudo, que, dadas as limitações computacionais e do tamanho dos textos originais já mencionados anteriormente, optou-se em realizar o balanceamento apenas dos conjuntos de dados contendo os textos reduzidos.

4.2.1.1 Agrupamento de classes

Em função a proporção observada, visando simplificar a tarefa de classificação e aumentar a eficiência dos modelos gerados, ficou estabelecido como requisito pela área negocial que as classes minoritárias fossem agrupadas. Assim sendo, como ilustrado na Figura 4.4, as cinco classes majoritárias foram mantidas como categorias individuais, enquanto as demais 20 classes foram consolidadas sob o rótulo "OUTRAS", totalizando agora seis classes ao todo. Essa estratégia permitiu uma redução na complexidade do problema, além de garantir que as classes mais relevantes permanecessem detalhadas, enquanto aquelas com menos representatividade fossem tratadas de forma agrupada.

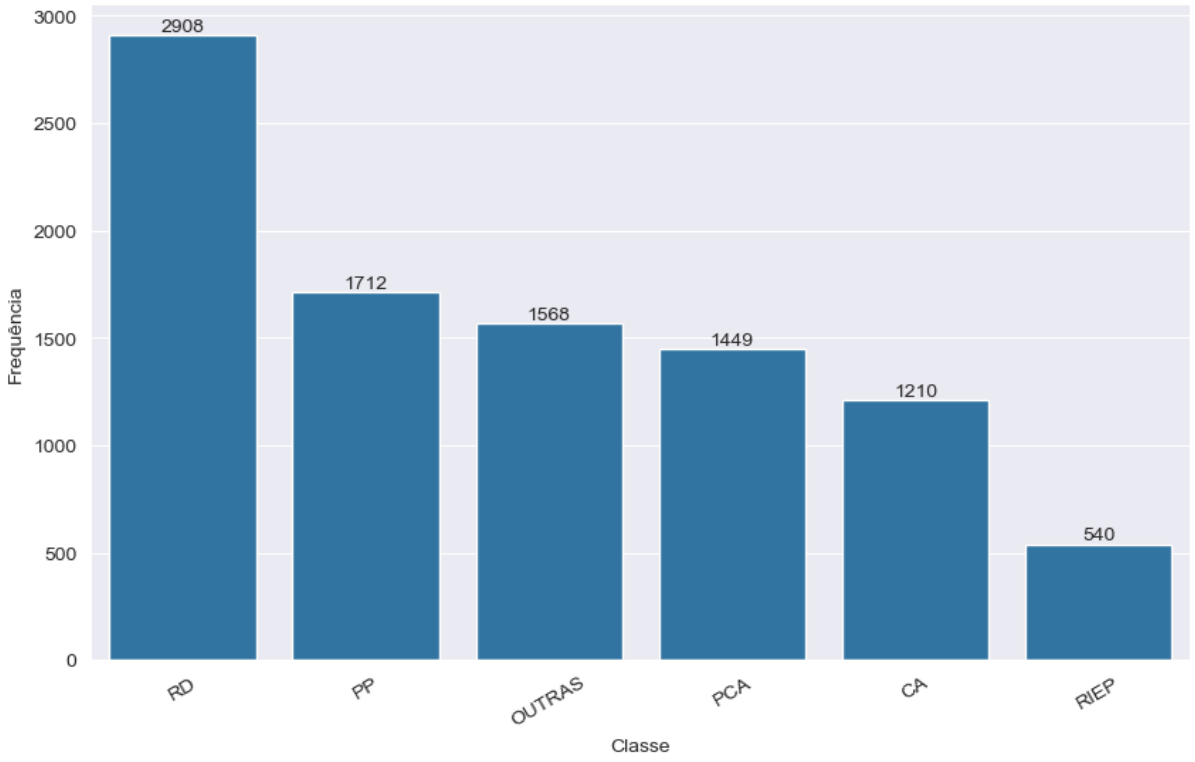

Figura 4.4: Distribuição de petições iniciais entre as classes após agrupamento das classes minoritárias.

4.2.1.2 Geração de dados sintéticos

No escopo da pesquisa, optou-se pelo uso da técnica de *Data Augmentation* ou, em uma tradução literal, Aumento de Dados, utilizando o poder de um LLM para a geração de dados sintéticos [90], a fim de propiciar o balanceamento das classes [91]. Ao contrário de técnicas de sobreamostragem existentes, que geram dados sintéticos a partir de vetores

de dados, sua diferença reside no fato de que a geração de dados sintéticos é feita a partir de dados brutos, antes mesmo do processo de vetorização.

Para o presente estudo, a geração de dados sintéticos se deu basicamente pela geração de textos resumidos sintéticos, produzidos por um LLM a partir de petições resumidas, contendo dados reais, das classes que necessitavam de balanceamento. Mais uma vez, todas as etapas foram executadas com o auxílio da biblioteca *LangChain* para *Python*.

Dessa forma, após os passos preparatórios, o conjunto de dados de treino separado anteriormente foi submetido ao balanceamento utilizando a técnica de aumento de dados para geração de dados sintéticos, com o objetivo de igualar a quantidade de petições entre as classes, permitindo que as classes menos frequentes tivessem uma maior quantidade de amostras.

Com base no número de petições presentes na classe majoritária para a divisão de treino (Reclamação Disciplinar), foram criados dados sintéticos para as demais cinco classes até que todas atingissem o mesmo número de amostras da primeira. Descrito de forma genérica, o processo consistiu em embaralhar o conjunto de dados, classe por classe, e, dentro de um laço, para cada texto resumido contendo dados reais, reescrevê-lo com dados modificados para se obter uma nova versão sintética, utilizando, para isso, um modelo de linguagem.

O modelo utilizado foi o GPT-3.5-Turbo versão 0301 (Azure OpenAI), configurado no modo de conversa na biblioteca *LangChain*, com os mesmos valores padrão de hiperparâmetros aplicados para a tarefa de sumarização abstrativa de textos pelo GPT-4 (Tabela 4.3) da fase anterior. Também foi pré-configurado na biblioteca o mesmo *template* de *prompt* utilizado para a tarefa de sumarização, como reproduzido novamente a seguir:

```
"""Responda a pergunta usando o contexto informado.
   Responda da forma mais completa entre 200 e 300 palavras caso seja
   possível. \n\n
   Contexto: \n {context} \n
   Pergunta: \n {question} \n
   Resposta:
"""
```

Recapitulando, no *template*, o modelo foi instruído a produzir respostas completas, com textos entre 200 e 300 palavras, utilizando como base o contexto fornecido. Esse contexto foi constituído pelo texto reduzido contendo dados reais de petições resumidas nas etapas anteriores e foi passado como parâmetro (*context*) para o LLM no *prompt* pré-configurado.

Além do texto reduzido real, o *template* de *prompt* também previa que fosse passada uma instrução (parâmetro *question*) para a geração da nova versão modificada da petição inicial. A instrução foi escrita no código como se segue:

```
"""
    Por favor reescreva o texto de forma que sua estrutura seja diferente
    da atual.
    Mude os nomes das pessoas, os Ministérios Públicos (MP), os estados e
    as cidades, com exceção do Conselho Nacional do Ministério Público
    (CNMP).
"""
```

O cuidado tomado na instrução foi que o contexto representado no texto reduzido real fosse modificado sem que sua essência fosse afetada, diminuindo a possibilidade de que a classe fosse descaracterizada na nova versão, que também foi rotulada com a mesma classe. Além da redação original, apenas informações de partes e localidades foram modificadas. Essa abordagem teve como propósito possibilitar que os textos gerados fossem variados o suficiente para enriquecer o conjunto de dados sem comprometer a integridade das informações contextuais.

4.2.2 Vetorização

A tarefa constituiu-se na estruturação dos textos em representações numéricas, ou vetores, utilizando as técnicas TF-IDF e *embeddings*. Cada uma das técnicas foi aplicada nos conjuntos de dados contendo os textos originais e textos reduzidos.

O trabalho se iniciou com a aplicação da técnica de vetorização TF-IDF, utilizando a biblioteca *scikit-learn* para *Python*. O primeiro passo foi vetorizar o texto original a partir dos *tokens* obtidos nas etapas anteriores. Depois, foi a vez dos textos reduzidos, cujo conjunto de dados houvera sido balanceado previamente. Para ambos os casos, a ferramenta foi configurada para produzir vetores com 3.000 dimensões que foram processados e utilizados na continuidade da pesquisa.

A seguir, foi a vez dos textos serem vetorizados como *embeddings*. Como na técnica de *RAG*, para essa tarefa, utilizou-se o modelo *text-embeddings-large-3*, na versão 1. Embora possua a capacidade de gerar vetores de dimensões variáveis, dessa vez foram produzidos pelo modelo vetores com o tamanho máximo suportado de 3.072 dimensões. Além disso, cumpre ressaltar que só foram gerados *embeddings* para os textos reduzidos, uma vez que o modelo aceita um tamanho máximo de entrada de 8.191 *tokens*[11], o que inviabilizaria sua utilização com os textos originais.

[11] Disponível em `https://platform.openai.com/docs/guides/embeddings/what-are-embeddings`

No caso dos modelos BERT, o processo de vetorização também se baseia em *embeddings*, que, na arquitetura *Transformer*, correspondem aos estados ocultos da rede neural. Contudo, todo esse processo é encapsulado dentro *pipeline* de treinamento automatizado da rede para o ajuste fino da tarefa de classificação de texto executado pela biblioteca *Hugging Face*. Dessa maneira, não houve necessidade de conduzir explicitamente essa etapa no conjunto de dados de textos reduzidos que seriam utilizados com os modelos BERT avaliados no estudo.

4.3 Construção dos Modelos

Nesta etapa ocorreu o processo de construção dos modelos de classificação de texto, que envolveu tanto algoritmos tradicionais de aprendizado de máquina quanto os baseados em aprendizado profundo, no caso, com modelos baseados em BERT. O objetivo foi explorar as diferentes combinações de conjuntos de dados e de algoritmos para comparar os desempenhos e tentar encontrar as melhores soluções para a tarefa de classificação de petições iniciais do CNMP.

4.3.1 Algoritmos de Aprendizado de Máquina Tradicionais

Para os conjuntos de dados de número 1, 2, 3, 5 e 6, foram ajustados classificadores utilizando a biblioteca *scikit-learn* para *Python*. Os algoritmos empregados foram: *Naïve Bayes*, k-NN, Árvore de Decisão, SVM, Floresta Aleatória e Regressão Logística. Além disso, foi ajustado um classificador *Dummy*[12], que não usa nenhuma técnica de aprendizado real, mas uma estratégia muito básica que envolve prever a classe mais frequente. Seu propósito era oferecer um ponto de referência para comparação com o desempenho dos outros modelos.

Algumas considerações são necessárias sobre os ajustes realizados. Começando pelos classificadores baseados em *Naïve Bayes*, testaram-se as variações Bernoulli, Multinomial e Gaussiana para avaliar qual delas melhor se adequava aos dados. Em sua variação Multinomial, foi necessário realizar a normalização prévia dos dados para valores entre zero e um, dado que o algoritmo não aceita números negativos. Isso não foi problema para os vetores TF-IDF, mas vetores de *embeddings* continham números menores do que zero. Essa transformação foi realizada utilizando-se a classe *MinMaxScaler* também da biblioteca *scikit-learn*.

Já para o algoritmo k-NN, foram ajustados classificadores para valores de k iguais a 1, 3, 5 e 11 (hiperparâmetro *n_neighbors*), a fim de analisar o impacto do número de

[12]Disponível em `https://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyClassifier.html`.

vizinhos no desempenho do modelo. Também se optou por avaliar a aplicação de pelo menos um método de aprendizado em conjunto, no caso, Floresta Aleatória, que utiliza várias árvores de decisão em conjunto, 100 por padrão da biblioteca (hiperparâmetro *n_estimators*).

Também foi configurado o hiperparâmetro *class_weight* = "*balanced*" para os classificadores baseados em Árvore de Decisão, Floresta Aleatória, SVM e Regressão Logística, com o intuito de melhorar o desempenho nos conjuntos de dados com distribuições de classes desbalanceadas. Com essa configuração, o algoritmo ajusta automaticamente o peso de cada classe de forma inversamente proporcional à sua frequência no conjunto de dados. Isso significou que classes minoritárias receberam um peso maior, enquanto classes mais frequentes receberam um peso menor, em uma tentativa de fazer com que o classificador não ficasse tendencioso em favor das classes com maior número de petições.

Por fim, registra-se que foram configurados os hiperparâmetros *max_iter* = 6.000 e *multi_class* = "*ovr*" para classificador baseado em Regressão Logística. O primeiro significa que o algoritmo tinha até 6.000 iterações para convergir durante o treinamento do classificador. Ou seja, ele estabelece o número máximo de iterações que o algoritmo de otimização terá à sua disposição até encontrar uma solução ótima para os coeficientes da regressão.

O segundo hiperparâmetro, "*multi_class*", determina como o modelo lida com problemas de classificação quando existem mais de duas classes. Relembrando, o problema de classificação de petições tratado nesta pesquisa é chamado de multiclasse. Isso significa dizer que o classificador sendo ajustado precisa atribuir uma classe dentre várias a cada petição. O valor configurado "*ovr*" significa "*one-vs-rest*" ou "um contra todos". Com essa configuração, o algoritmo cria um classificador binário para cada classe e cada um deles aprende a distinguir uma classe específica contra todas as outras combinadas, simplificando o problema multiclasse em múltiplos problemas binários mais simples.

Ambas as configurações foram atribuídas de forma empírica, baseadas nas observações das tentativas de ajuste do classificador sobre os conjuntos de dados da pesquisa, visando melhor desempenho e adequação ao contexto tratado no estudo.

4.3.2 Algoritmos de Aprendizado Profundo

Além dos algoritmos tradicionais, foram realizados experimentos com os modelos baseados em BERT, representando a categoria dos algoritmos de aprendizado profundo. A abordagem com esses modelos complementou os experimentos com os classificadores tradicionais, permitindo a comparação entre técnicas clássicas de aprendizado de máquina e técnicas mais modernas de PLN.

Com os dados dos conjuntos de número 4 e 7, passou-se à realização do ajuste fino dos modelos BERT (BERTimbau e Albertina PT-BR), cada um a seu turno, utilizando-se para isso a biblioteca *Hugging Face*. Descrito de forma bastante genérica, o processo realizado contempla cinco passos principais: carga do modelo para a tarefa de classificação, configuração dos hiperparâmetros e da execução do treinamento, execução do treinamento e persistência do modelo ajustado.

4.3.2.1 Carga do modelo para a tarefa de classificação

Os modelos foram carregados com uma cabeça de classificação de sequência de texto. Essa camada adicional é responsável por fazer a previsão final da classe de uma petição a partir das representações vetoriais geradas pelo modelo a partir dos textos processados. Como já visto, a cabeça representa a última camada de uma rede neural que recebe essas representações dos estados ocultos e as transforma em um vetor cuja dimensionalidade corresponde ao número de classes da tarefa de classificação, permitindo assim a previsão final.

4.3.2.2 Configuração dos hiperparâmetros de treinamento

Uma instância da classe *TrainingArguments* foi configurada com os hiperparâmetros desejados para o novo processo de treinamento da rede neural com vistas ao ajuste fino. A Tabela 4.4 detalha os hiperparâmetros configurados e seus valores.

É importante frisar que, embora não tenha sido explicitamente configurada, a função de otimização *AdamW* [92] é utilizada, por padrão, no rol dos hiperparâmetros do objeto *TrainingArguments*. Em relação aos hiperparâmetros *learning_rate* e *weight_decay*, eles adotaram como referência os valores propostos por Tunstall *et al* [61] no capítulo sobre classificação de texto de sua obra intitulada *Natural Language Processing with Transformers*.

A diminuição da sobrecarga de memória foi o objetivo ao se ativar o hiperparâmetro *gradient_checkpointing*. De forma resumida, ao ser ativado (valor *True*) ele modifica a forma de armazenamento das ativações intermediárias da rede durante a propagação entre as camadas da rede, armazenando apenas um subconjunto delas em pontos chamados de *checkpoints*. Quando as ativações intermediárias não calculadas são necessárias durante a retropropagação, elas são então recalculadas executando-se novamente a propagação para as partes necessárias da rede neural, utilizando-se como pontos de partida o subconjunto de ativações intermediárias previamente armazenado.

Por fim, é importante ressaltar que esse hiperparâmetro foi necessário dadas as limitações computacionais encontradas com a GPU utilizada, em especial aquelas relacionadas

ao uso de memória. Contudo, como ponto negativo, ele trouxe aumento significativo no tempo de treinamento em função de suas formas de funcionamento.

Tabela 4.4: Hiperparâmetros de ajuste fino dos modelos BERT

Hiperparâmetro	**Descrição**	**Valor**
num_train_epochs	Número total de épocas para treinar o modelo.	4
per_device_train_batch_size	Tamanho do lote de treinamento por dispositivo.	32
per_device_eval_batch_size	Tamanho do lote de avaliação por dispositivo.	32
learning_rate	Taxa de aprendizado inicial.	0,00002
weight_decay	Fator de decaimento dos pesos.	0,01
evaluation_strategy	Estratégia para avaliação durante o treinamento.	"*epoch*"
logging_strategy	Estratégia para registro de *logs* durante o treinamento.	"*epoch*"
save_strategy	Estratégia para salvar *checkpoints* do modelo durante o treinamento.	"*epoch*"
gradient_checkpointing	Ativa o *checkpointing* de gradientes.	*True*

4.3.2.3 Configuração da execução do treinamento

A instância da classe *Trainer* foi configurada com os parâmetros de entrada para a execução do ajuste fino no respectivo modelo. Como parâmetros recebidos, incluíram-se: o objeto contendo os hiperparâmetros configurados no passo anterior, a função de cálculo das métricas de avaliação, os conjuntos de dados de treino e teste (já previamente separados na etapa de engenharia de características) e o tokenizador usado para a tokenização dos conjuntos de dados.

4.3.2.4 Execução do treinamento

Após a realização das etapas de configuração, o treinamento do respectivo modelo foi executado pela instância configurada da classe *Trainer* sobre os conjuntos de dados da pesquisa para a tarefa de classificação de texto. O processo de ajuste fino foi executado por 4 épocas (hiperparâmetro *num_train_epochs*) e as avaliações realizadas ao final de cada época (hiperparâmetro *evaluation_strategy*) com as métricas configuradas para a execução do treinamento.

4.3.2.5 Persistência do modelo ajustado

O modelo ajustado foi salvo após o ajuste fino para uso posterior, permitindo que novas inferências fossem feitas com base nesse treinamento.

4.4 Avaliação dos modelos

Na etapa de avaliação dos modelos desenvolvidos para a tarefa de classificação de texto, foram empregadas duas técnicas de validação distintas: *hold-out* e validação cruzada. Essas técnicas foram escolhidas com o intuito de avaliar a robustez e a generalização dos resultados obtidos.

Para ambas as técnicas de validação, foram utilizadas métricas amplamente reconhecidas na literatura de aprendizado de máquina para a avaliação de modelos de classificação. Especificamente, foram calculadas a acurácia e o *F1-Score* nas suas variantes micro, macro e ponderado. Essas métricas permitem uma avaliação abrangente do desempenho dos modelos, considerando tanto o equilíbrio entre as classes quanto a relevância de cada uma no conjunto de dados. Além dessas métricas, calculou-se também o tempo decorrido para ajuste dos modelos.

4.4.1 Validação Hold-out

Pela técnica de *hold-out* os conjuntos de dados foram divididos em 70% para treinamento e 30% para teste. Essa proporção é comumente utilizada e oferece um equilíbrio adequado entre a quantidade de dados para treinar o modelo e a quantidade para avaliá-lo.

Todos os algoritmos tradicionais ajustados foram avaliados utilizando a técnica de *hold-out* em todos os conjuntos de dados disponíveis. Isso incluiu tanto os conjuntos de dados contendo os textos originais, quanto os textos resumidos por sumarização abstrativa, sem balanceamento e com balanceamento. Além disso, foram avaliados todos os textos com os dois tipos de vetorização, TF-IDF e *embeddings*.

Para os modelos baseados em BERT ajustados, a técnica de *hold-out* foi aplicada apenas aos conjuntos de dados contendo os textos reduzidos, tanto para o conjunto desbalanceado quanto para o balanceado. Como já colocado, a validação limitou-se aos conjuntos de dados contendo textos reduzidos dada a janela de contexto de 512 *tokens* da arquitetura, o que tornou inviável a utilização dos textos originais completos.

4.4.2 Validação Cruzada

A validação cruzada, especificamente com método de amostragem estratificada *k-fold* com $k = 10$, foi utilizada para avaliar os modelos de forma mais robusta, garantindo que os

resultados sejam menos suscetíveis a variações decorrentes de uma única divisão dos dados. Para cada uma das métricas calculadas, também foi calculado seu intervalo de confiança de 95%.

Os algoritmos tradicionais ajustados foram submetidos à validação cruzada em todos os conjuntos de dados, empregando ambos os tipos de vetorização. Contudo, é importante ressaltar que a técnica de validação cruzada não foi aplicada aos modelos BERT. Essa decisão baseou-se na baixa capacidade computacional disponível, uma vez que a execução da validação cruzada com modelos tão complexos e pesados em termos de processamento demandaria um tempo computacional impraticável no contexto deste trabalho.

Capítulo 5

Resultados

Na presente seção são apresentadas as análises dos resultados obtidos nos experimentos para a tarefa de classificação de texto sobre os conjuntos de dados preparados a partir das diferentes abordagens de pré-processamento de texto, balanceamento de dados e tipo de vetorização. Esses experimentos tiveram como objetivo comparar o desempenho dos diversos classificadores baseados nos algoritmos tradicionais de aprendizado de máquina e nos de aprendizado profundo selecionados para a pesquisa.

Como detalhado na seção de metodologia da pesquisa, as métricas comparadas foram acurácia e *F1-Score* (micro, macro e ponderado). Já os classificadores ajustados incluíram aqueles baseados em aprendizado de máquina e em modelos baseados em BERT (BERTimbau e Albertina PT-BR), representando os algoritmos de aprendizado profundo.

As métricas mencionadas estão ilustradas em gráficos de mapas de calor, oferecendo uma visão consolidada do desempenho dos classificadores. Pelo fato de os resultados das métricas de acurácia e F1-Score micro terem sido idênticos, os gráficos correspondentes ilustram apenas a métrica de acurácia.

Os primeiros resultados a serem abordados são os de algoritmos tradicionais de aprendizado de máquina, obtidos com a técnica de validação *hold-out*. Como representado na Figura 5.1, de maneira geral, observa-se que o classificador baseado em SVM (*Support Vector Classification* (SVC)) apresentou o melhor desempenho entre os diversos conjuntos de dados da pesquisa. No respectivo mapa de calor, observa-se que o classificador alcançou o valor máximo de 76,03% no conjunto de dados configurado com a versão reduzida do texto, o balanceamento das classes e a vetorização baseada em *embeddings* (conjunto de dados número 6) , destacando-se dos demais classificadores.

Da mesma forma, nas métricas de *F1-Score*, tanto na métrica macro (Figura 5.2) quanto ponderada (Figura 5.3), o SVC obteve os melhores índices, com *F1-Score* macro de 74,17% e ponderado de 75,70%, reforçando a robustez do modelo frente às diferentes abordagens adotadas nos conjuntos de dados.

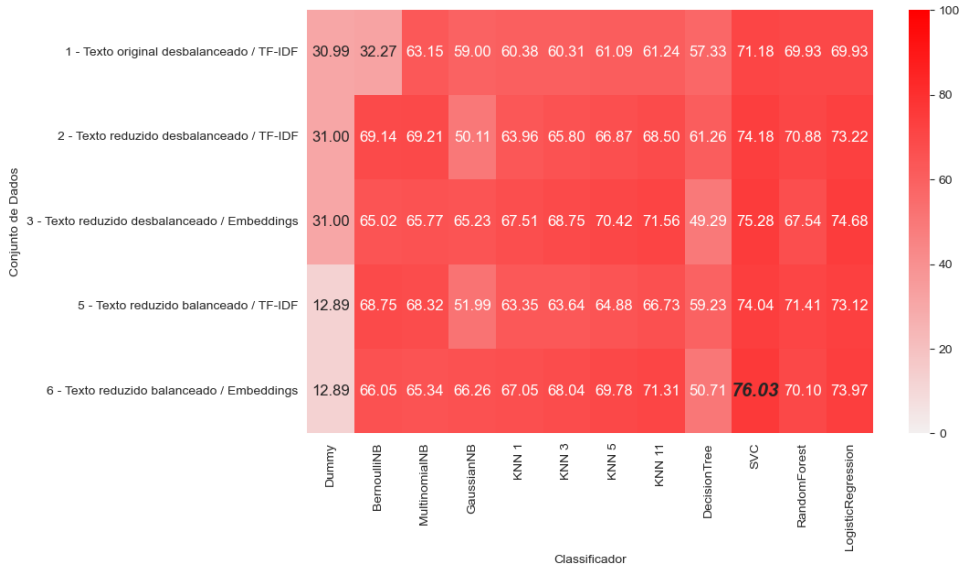

Figura 5.1: Mapa de calor da acurácia dos classificadores tradicionais de aprendizado de máquina (validação *hold-out*).

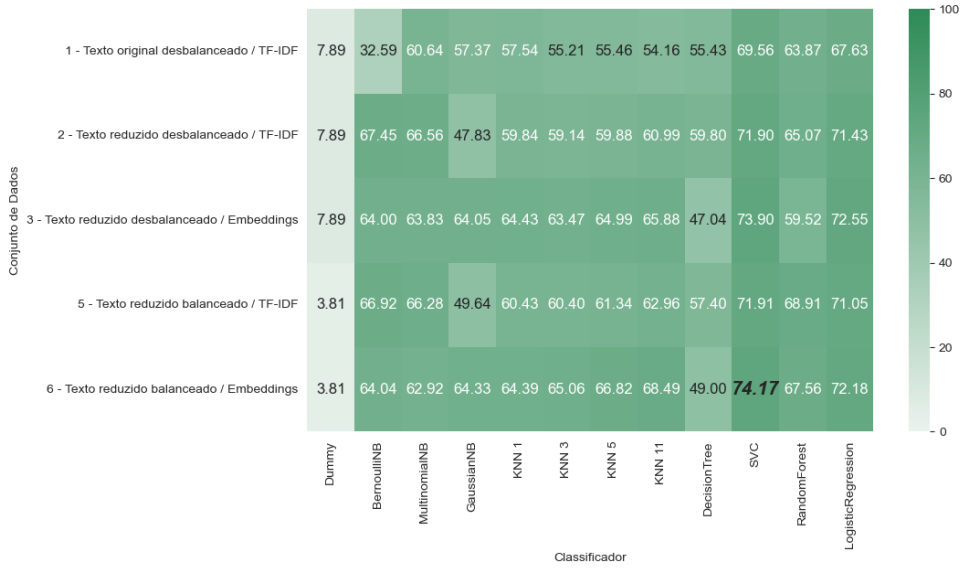

Figura 5.2: Mapa de calor do F1-Score macro dos classificadores tradicionais de aprendizado de máquina (validação *hold-out*).

Comparando-se os resultados entre os diferentes conjuntos de dados, é evidente que os conjuntos que utilizam *embeddings* como representação vetorial (conjuntos 3 e 6) obtiveram melhores desempenhos em relação aos demais conjuntos. Isso sugere que a representação vetorial na forma de *embeddings* se mostrou vantajosa em contraste com a técnica TF-IDF para a tarefa de vetorização de dado textual, em especial para o classificador

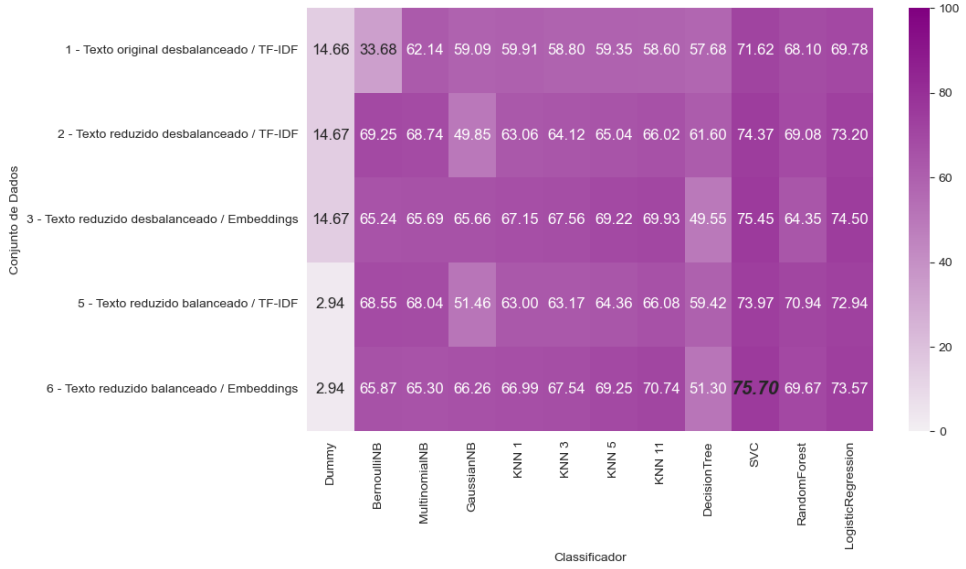

Figura 5.3: Mapa de calor do F1-Score ponderado dos classificadores tradicionais de aprendizado de máquina (validação *hold-out*).

baseado em SVC.

Outro ponto relevante está relacionado à comparação entre os diferentes algoritmos de classificação. O classificador baseado em Regressão Logística (*Logistic Regression*) também obteve um bom desempenho em termos de *acurácia e F1-Score* nos conjuntos com a vetorização por *embeddings* (conjuntos 3 e 6). O modelo *Dummy*, utilizado como linha de base, apresentou os piores resultados em todas as configurações, conforme esperado.

Passando à análise dos resultados da validação para os algoritmos de aprendizado profundo baseados em BERT, BERTimbau e Albertina PT-BR, ainda utilizando-se como referência a técnica de validação por *hold-out*, observa-se um desempenho superior em comparação aos classificadores tradicionais. Como representado no mapa de calor de acurácia da Figura 5.4, o modelo BERTimbau aplicado ao conjunto de dados desbalanceado (conjunto 4) obteve uma acurácia de 78,87%, superando todos os classificadores tradicionais. Além disso, o modelo Albertina PT-BR apresentou acurácia de 76,60% no conjunto balanceado (conjunto 7), o que também é superior aos valores observados para o SVC.

Nas métricas de *F1-Score* (macro e ponderado), os modelos BERT também mostraram resultados expressivos. No *F1-Score* macro (Figura 5.5), o BERTimbau atingiu 77,22% no conjunto desbalanceado (conjunto 4), enquanto Albertina PT-BR obteve 74,95% no conjunto balanceado (conjunto 7). Esses valores são maiores do que os obtidos pelos classificadores tradicionais, indicando uma maior capacidade de generalização dos modelos de linguagem baseados em BERT. Já no *F1-Score* ponderado (Figura 5.6), o BERTimbau alcançou 78,58% (conjunto 4), enquanto o Albertina PT-BR atingiu 76,51% no conjunto

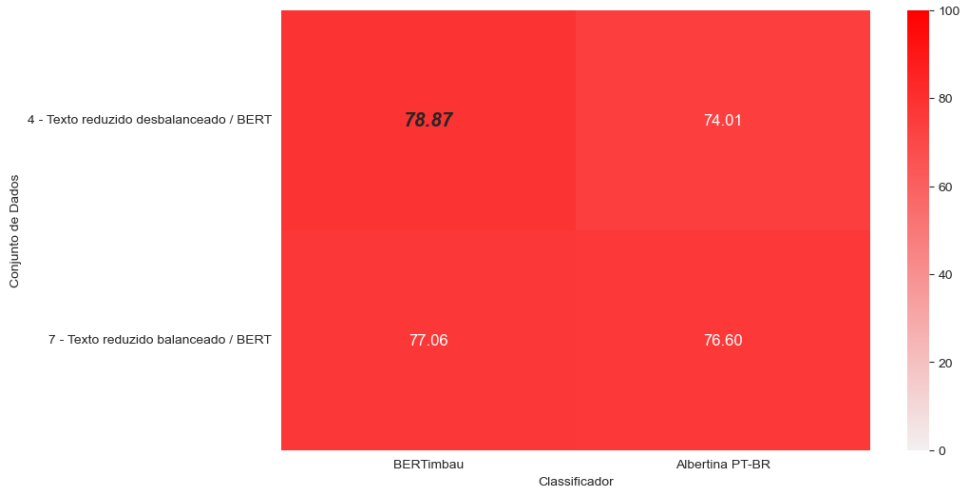

Figura 5.4: Mapa de calor da acurácia dos classificadores baseados em aprendizado profundo (validação *hold-out*).

balanceado (conjunto 7), novamente superando os melhores resultados dos classificadores tradicionais.

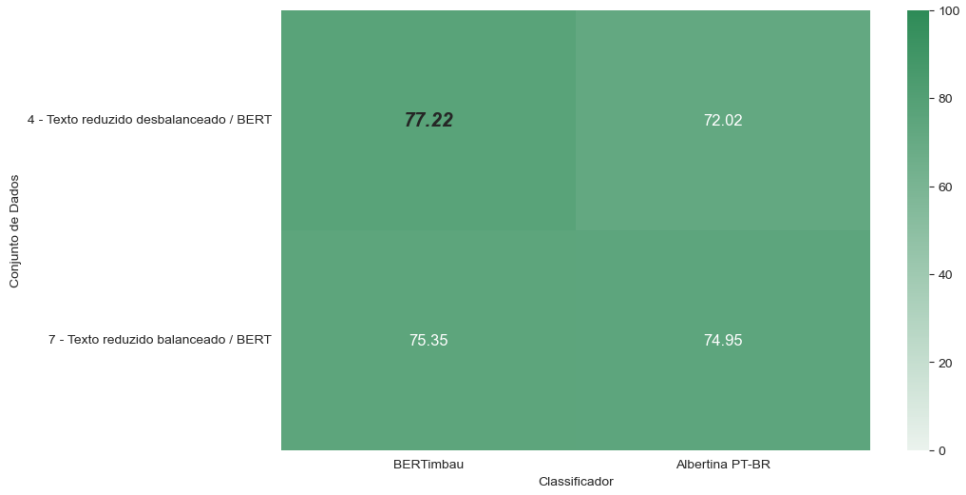

Figura 5.5: Mapa de calor do F1-Score macro dos classificadores baseados em aprendizado profundo (validação *hold-out*).

Embora o SVC tenha se destacado entre os classificadores tradicionais, os modelos baseados em BERT apresentaram desempenho ainda superior. Isso reforça a eficácia dos modelos de linguagem pré-treinados, que conseguem capturar nuances semânticas mais profundas dos textos em comparação com técnicas tradicionais de representação vetorial como TF-IDF e *embeddings*. Comparando-se os modelos BERTimbau e Albertina PT-BR, o primeiro leva ligeira vantagem sobre o segundo, possivelmente pelo número de

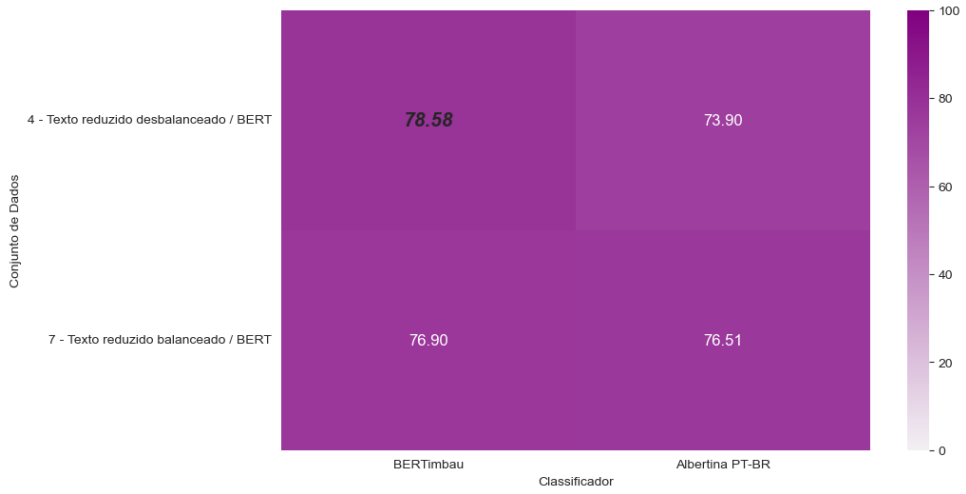

Figura 5.6: Mapa de calor do F1-Score ponderado dos classificadores baseados em aprendizado profundo (validação *hold-out*).

parâmetros de modelo ser mais do que o dobro (350 milhões contra 100 milhões), muito embora a técnica de balanceamento utilizada tenha ajudado a melhorar significativamente o desempenho do menor modelo em que pese o tamanho.

Com o intuito de complementar a avaliação dos classificadores, também foi realizada a validação cruzada com método de amostragem estratificada *k-fold* com $k = 10$ a fim de garantir uma avaliação menos suscetível a variações eventualmente causadas na amostragem utilizada pelo método *hold-out*. Ressalta-se que, pelas limitações computacionais já mencionadas, os modelos BERT não foram validados utilizando-se a técnica de validação cruzada.

Nos mapas de calor, é possível observar que, no geral, o desempenho nas métricas confirmou os resultados obtidos com a técnica de validação por *hold-out* para os conjuntos de dados com distribuição de classes desbalanceadas (conjuntos 1, 2 e 3). Contudo, para os conjuntos de dados balanceados (conjuntos 5 e 6), possivelmente ocorreu sobreajuste, dado que os valores da acurácia (Figura 5.7) das métricas da amostragem do método *hold-out* estão fora do intervalo de confiança de 95% calculado na validação cruzada, o que leva a crer que a estratégia de balanceamento das classes como realizada na pesquisa pode ter inserido viés no ajuste dos modelos. O mesmo foi observado nas métricas de *F1-Score* macro (Figura 5.8) e ponderado (Figura 5.9).

Isso posto, caso sejam considerados apenas os conjuntos de dados desbalanceados para a análise (conjuntos 1, 2 e 3), ainda é possível ver a vantagem do classificador baseado em SVM, confirmando o resultado da validação *hold-out*, com destaque para a vetorização por *embeddings* em relação à técnica TF-IDF.

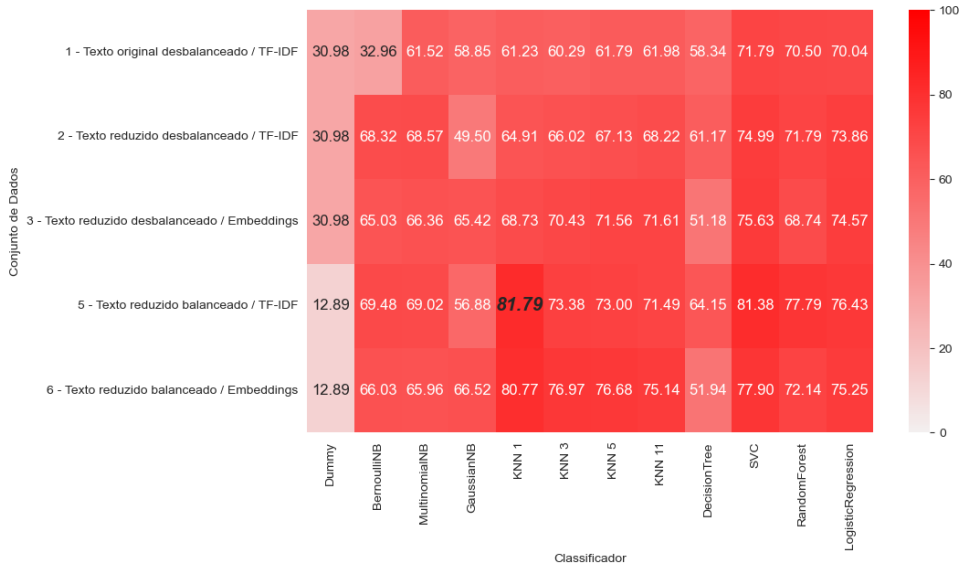

Figura 5.7: Mapa de calor da acurácia dos classificadores tradicionais de aprendizado de máquina (validação cruzada).

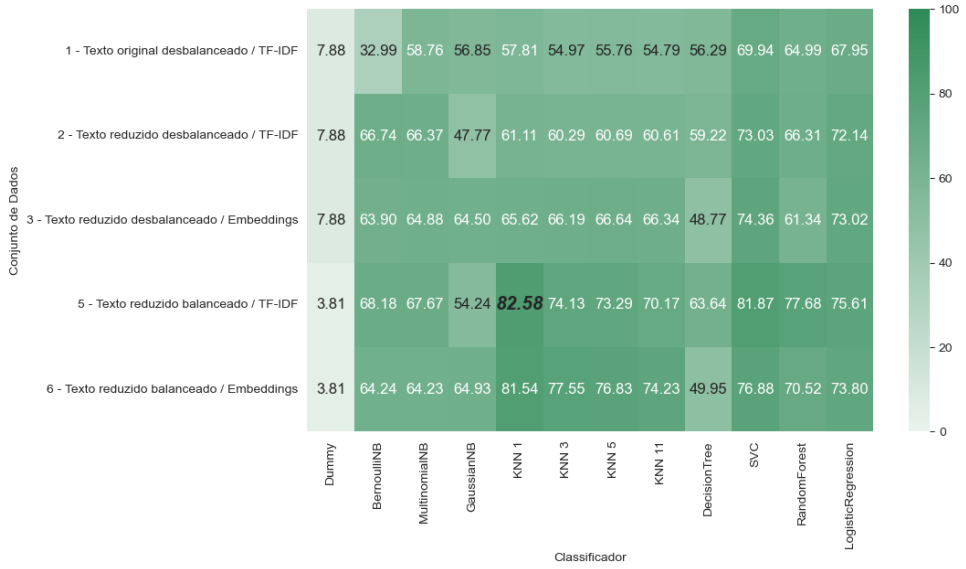

Figura 5.8: Mapa de calor do F1-Score macro dos classificadores tradicionais de aprendizado de máquina (validação cruzada).

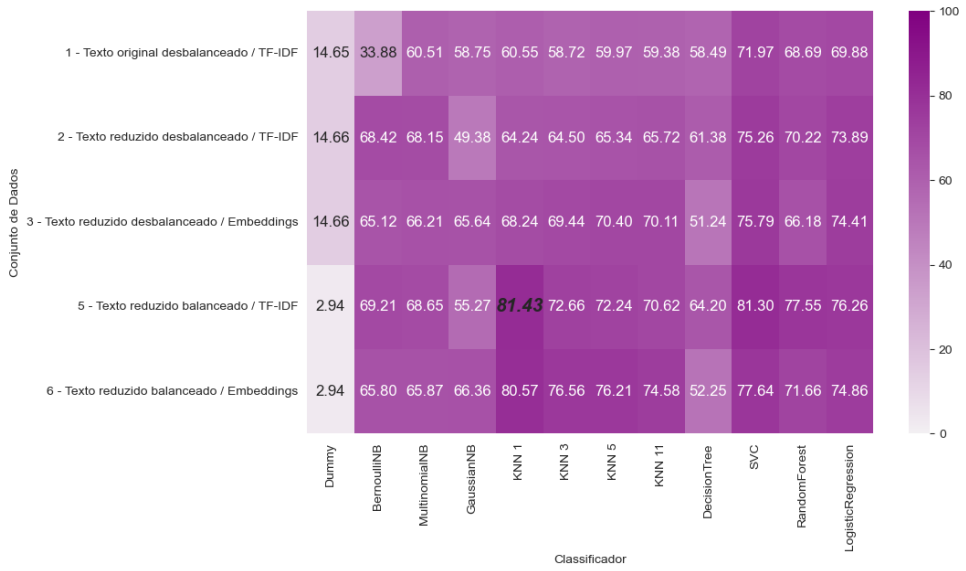

Figura 5.9: Mapa de calor do F1-Score ponderado dos classificadores tradicionais de aprendizado de máquina (validação cruzada).

Capítulo 6

Conclusão

Com base nos experimentos realizados e na análise dos resultados obtidos, a hipótese deste trabalho foi sustentada. Os modelos de linguagem baseados em BERT, ajustados especificamente para a tarefa de classificação das petições iniciais conforme as classes processuais definidas no artigo 37 do Regimento Interno do CNMP, demonstraram desempenho superior em comparação aos algoritmos tradicionais de aprendizado de máquina.

A análise dos resultados mostra que os modelos BERTimbau e Albertina-PTBR superaram os classificadores tradicionais em todas as métricas, destacando-se como as melhores opções para a tarefa de classificação de texto, especialmente quando se busca maior acurácia e capacidade de generalização. Essa superioridade evidenciou a eficácia da técnica de ajuste fino de modelos de linguagem para classificação de texto no contexto desta pesquisa.

A integração de estratégias de pré-processamento, incluindo a digitalização e a limpeza de informações textuais, e a implementação de técnicas para redução do tamanho textual, com sumarização abstrativa utilizando *Retrieval-Augmented Generation* e LLM, contribuíram significativamente para o desempenho dos classificadores. Esses aspectos foram fundamentais para lidar com a complexidade das petições iniciais, garantindo a preservação das informações mais relevantes para a tarefa de classificação, em que pese a média elevada de palavras e tamanho dos documentos encontrados.

Ademais, o trabalho focado no balanceamento dos dados por meio da técnica de *Data Augmentation*, implementada com o suporte de LLM, embora não tenha melhorado o desempenho do melhor classificador ajustado, o BERTimbau (treinado com dados desbalanceados), demonstrou impacto positivo no estudo. Especificamente, a técnica contribuiu para aprimorar os resultados obtidos com algoritmos tradicionais, especialmente no classificador baseado em SVM em combinação com a vetorização por *embeddings*. Além disso, o uso da técnica foi crucial para o aprimoramento do modelo BERT com menor número de parâmetros, o Albertina PT-BR.

Portanto, o objetivo geral deste trabalho foi atingido, demonstrando que o ajuste fino de modelos BERT é uma alternativa eficaz para a classificação de petições iniciais do Conselho Nacional do Ministério Público, superando abordagens mais tradicionais de aprendizado de máquina. Os objetivos específicos também foram cumpridos, com destaque para a implementação de técnicas inovadoras de pré-processamento e balanceamento de dados, além da comparação bem-sucedida entre diferentes abordagens de classificação.

Como trabalhos futuros, pretende-se repetir o estudo com outros modelos BERT, bem como com outros modelos baseados na arquitetura *Transformers*. Além disso, pretende-se ampliar a pesquisa, utilizando as técnicas aqui desenvolvidas para realizar a classificação de outros tipos de documentos além de petições iniciais. De forma geral, os resultados demonstraram a eficácia e o ineditismo, dentro do Ministério Público brasileiro, das técnicas empregadas durante a pesquisa para classificação de petições iniciais.

Referências

[1] CNMP: *Roteiro prático de atuação no Conselho Nacional do Ministério Público*, 2020. https://www.cnmp.mp.br/portal/publicacoes/245-cartilhas-e-manuais/13681-roteiro-pratico-de-atuacao-no-conselho-nacional-do-ministerio-publico, acesso em 2023-04-29. 1

[2] Brasil: *Constituição da República Federativa do Brasil de 1988*, maio 1988. https://www.planalto.gov.br/ccivil_03/constituicao/constituicaocompilado.htm, acesso em 2023-04-29. 2

[3] CNMP: *Regimento Interno do Conselho Nacional do Ministério Público*, março 2013. https://www.cnmp.mp.br/portal/atos-e-normas/norma/46, acesso em 2023-04-27. 2, 3, 4, 6

[4] CNMP: *Resolução CNMP nº 146/2016*, junho 2016. https://www.cnmp.mp.br/portal/atos-e-normas/norma/4182/, acesso em 2023-04-27. 2

[5] CNMP: *Estrutura Organizacional do Conselho Nacional do Ministério Público*, agosto 2021. https://www.cnmp.mp.br/portal/institucional/estrutura-organizacional, acesso em 2023-04-27. 2

[6] CNMP: *Resolução CNMP nº 252/2022*, novembro 2022. https://www.cnmp.mp.br/portal/atos-e-normas-busca/norma/9352, acesso em 2023-08-18. 2

[7] Brasil: *Lei nº 13.105/2015*, março 2015. https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2015/lei/l13105.htm, acesso em 2023-08-18. 4

[8] Pinheiro, Guilherme César e Lorena Ribeiro De Carvalho Sousa: *A PETIÇÃO INICIAL E OS SEUS REQUISITOS NO NOVO CÓDIGO DE PROCESSO CIVIL*. REVISTA ESMAT, 10(15):75–104, agosto 2018, ISSN 2447-9896, 2177-0360. http://esmat.tjto.jus.br/publicacoes/index.php/revista_esmat/article/view/234, acesso em 2023-08-17. 4

[9] Moura, Alexia Maria Barbosa Medina de, Nathália Ferreira Araújo e Ana Flávia Sales: *Técnica processual de elaboração da petição inicial*. junho 2021. https://repositorio.animaeducacao.com.br/handle/ANIMA/13288, acesso em 2023-08-17. 4

[10] Caseli, Helena de Medeiros e Maria das Graças Volpe Nunes: *Processamento de linguagem natural: conceitos, técnicas e aplicações em português*. BPLN, 2023. 7

[11] Manning, Christopher D. e Hinrich Schütze: *Foundations of statistical natural language processing*. MIT Press, Cambridge, Mass, 2e éd. avec des corrections edição, 1999. 7

[12] Goldberg, Yoav: *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers, abril 2017, ISBN 978-1-62705-298-6. 7, 33

[13] Jurafsky, Daniel e James H. Martin: *Speech and Language Processing - An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 3rd edition draft edição, janeiro 2023. https://web.stanford.edu/~jurafsky/slp3/ed3book_jan72023.pdf, acesso em 2023-07-31. 7, 26, 28, 30, 31, 32, 34, 39

[14] Cole Stryker e Jim Holdsworth: *O que é PLN (processamento de linguagem natural)? | IBM*, setembro 2021. https://www.ibm.com/br-pt/think/topics/natural-language-processing, acesso em 2025-02-27. 8

[15] *Natural Language Processing (NLP) Pipeline*. https://www.geeksforgeeks.org/natural-language-processing-nlp-pipeline/, acesso em 2025-02-27, Section: Machine Learning. 9

[16] Ali, Asjad: *Understanding the NLP Pipeline: A Comprehensive Guide*, janeiro 2024. https://medium.com/@asjad_ali/understanding-the-nlp-pipeline-a-comprehensive-guide-828b2b3cd4e2, acesso em 2025-02-27. 9

[17] Ayodele, Taiwo Oladipupo: *Machine Learning Overview*. Em *New Advances in Machine Learning*. IntechOpen, fevereiro 2010, ISBN 978-953-307-034-6. https://www.intechopen.com/chapters/10683, acesso em 2023-07-17. 13

[18] S.Archana e Dr K.Elangovan: *Survey of Classification Techniques in Data Mining*. International Journal of Computer Science and Mobile Applications, 2(2):65–71, 2014, ISSN 2321-8363. https://www.ijcsma.com/abstract/survey-of-classification-techniques-in-data-mining-95467.html, acesso em 2023-07-17, Publisher: International Journal of Computer Science and Mobile Applications. 13, 14, 15

[19] Han, Jiawei, Micheline Kamber e Jian Pei: *8 - Classification: Basic Concepts*. Em Han, Jiawei, Micheline Kamber e Jian Pei (editores): *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, páginas 327–391. Morgan Kaufmann, Boston, janeiro 2012, ISBN 978-0-12-381479-1. https://www.sciencedirect.com/science/article/pii/B9780123814791000083, acesso em 2023-07-17. 13, 16, 18

[20] Bellinger, Colin, Shiven Sharma e Nathalie Japkowicz: *One-Class versus Binary Classification: Which and When?* Em *2012 11th International Conference on Machine Learning and Applications*, volume 2, páginas 102–106, 2012. 14

[21] Moreo, Alejandro, Andrea Esuli e Fabrizio Sebastiani: *Word-class embeddings for multiclass text classification*. Data Mining and Knowledge Discovery, 35(3):911–963, maio 2021, ISSN 1384-5810. 14

[22] Li, Qian, Hao Peng, Jianxin Li, Congying Xia, Renyu Yang, Lichao Sun, Philip S. Yu e Lifang He: *A Survey on Text Classification: From Traditional to Deep Learning*. ACM Transactions on Intelligent Systems and Technology, 13(2):31:1–31:41, 2022, ISSN 2157-6904. https://dl.acm.org/doi/10.1145/3495162, acesso em 2023-07-31. 14, 15, 16, 17, 18, 19, 21, 25, 27

[23] Zhou, Xujuan, Raj Gururajan, Yuefeng Li, Revathi Venkataraman, Xiaohui Tao, Ghazal Bargshady, Prabal D. Barua e Srinivas Kondalsamy-Chennakesavan: *A survey on text classification and its applications*. Web Intelligence, 18(3):205–216, janeiro 2020, ISSN 2405-6456. https://content.iospress.com/articles/web-intelligence/web200442, acesso em 2023-07-29, Publisher: IOS Press. 15

[24] Akpatsa, Samuel K., Xiaoyu Li e Hang Lei: *A Survey and Future Perspectives of Hybrid Deep Learning Models for Text Classification*. Em Sun, Xingming, Xiaorui Zhang, Zhihua Xia e Elisa Bertino (editores): *Artificial Intelligence and Security*, Lecture Notes in Computer Science, páginas 358–369, Cham, 2021. Springer International Publishing, ISBN 978-3-030-78609-0. 15, 37

[25] Cortes, Corinna e Vladimir Vapnik: *Support-vector networks*. Machine Learning, 20(3):273–297, setembro 1995, ISSN 1573-0565. https://doi.org/10.1007/BF00994018, acesso em 2023-07-27. 15

[26] Noble, William S: *What is a support vector machine?* Nature Biotechnology, 24(12):1565–1567, dezembro 2006, ISSN 1087-0156. 15

[27] Kumar, Akshi, Vikrant Dabas e Parul Hooda: *Text classification algorithms for mining unstructured data: a SWOT analysis*. International Journal of Information Technology, 12(4):1159–1169, dezembro 2020, ISSN 2511-2112. https://doi.org/10.1007/s41870-017-0072-1, acesso em 2023-07-27. 15, 19, 20

[28] Kowsari, Kamran, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes e Donald Brown: *Text Classification Algorithms: A Survey*. Information, 10(4):150, abril 2019, ISSN 2078-2489. https://www.mdpi.com/2078-2489/10/4/150, acesso em 2023-07-25, Number: 4 Publisher: Multidisciplinary Digital Publishing Institute. 15, 16, 17, 18, 19, 20

[29] Soofi, Aized Amin e Arshad Awan: *Classification Techniques in Machine Learning: Applications and Issues*. Journal of Basic & Applied Sciences, 13:459–465, janeiro 2017, ISSN 1927-5129. https://setpublisher.com/index.php/jbas/article/view/1715, acesso em 2023-07-17. 15, 17

[30] Maron, M. E.: *Automatic Indexing: An Experimental Inquiry*. Journal of the ACM, 8(3):404–417, julho 1961, ISSN 0004-5411, 1557-735X. https://dl.acm.org/doi/10.1145/321075.321084, acesso em 2025-02-26. 16

[31] Manning, Christopher D., Prabhakar Raghavan e Hinrich Schütze: *Introduction to Information Retrieval*. Cambridge University Press, New York, illustrated edition edição, julho 2008, ISBN 978-0-521-86571-5. 16

[32] Khan, Aurangzeb, Baharum Baharudin, Lam Hong Lee e Khairullah Khan: *A Review of Machine Learning Algorithms for Text-Documents Classification*. Journal of Advances in Information Technology, 1(1):4–20, fevereiro 2010, ISSN 1798-2340. `http://www.jait.us/index.php?m=content&c=index&a=show&catid=160&id=859`, acesso em 2023-07-28. 17, 18

[33] Harrington, Peter: *Machine Learning in Action*. Manning Publications Co., USA, março 2012, ISBN 978-1-61729-018-3. 17, 20

[34] Kalcheva, Neli, Maya Todorova e Ginka Marinova: *NAIVE BAYES CLASSIFIER, DECISION TREE AND ADABOOST ENSEMBLE ALGORITHM – ADVANTAGES AND DISADVANTAGES*. Em *International Scientific Conference ERAZ – Knowledge Based Sustainable Development*, páginas 153–157, Online-virtual, maio 2020. ISBN 978-86-80194-33-2. `https://eraz-conference.com/eraz-2020-153/`, acesso em 2023-07-28. 17

[35] Agarwal, Basant e Namita Mittal: *Text Classification Using Machine Learning Methods-A Survey*. Em Babu, B. V., Atulya Nagar, Kusum Deep, Millie Pant, Jagdish Chand Bansal, Kanad Ray e Umesh Gupta (editores): *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012), December 28-30, 2012*, Advances in Intelligent Systems and Computing, páginas 701–709, New Delhi, 2014. Springer India, ISBN 978-81-322-1602-5. 17

[36] Li, Ruiguang, Ming Liu, Dawei Xu, Jiaqi Gao, Fudong Wu e Liehuang Zhu: *A Review of Machine Learning Algorithms for Text Classification*. Em Lu, Wei, Yuqing Zhang, Weiping Wen, Hanbing Yan e Chao Li (editores): *Cyber Security*, Communications in Computer and Information Science, páginas 226–234, Singapore, 2022. Springer Nature, ISBN 9789811692291. 18, 19

[37] Cover, T. e P. Hart: *Nearest neighbor pattern classification*. IEEE Transactions on Information Theory, 13(1):21–27, janeiro 1967, ISSN 1557-9654. Conference Name: IEEE Transactions on Information Theory. 19

[38] Li, Huijuan, He Jiang, Dongyuan Wang e Bing Han: *An Improved KNN Algorithm for Text Classification*. Em *2018 Eighth International Conference on Instrumentation & Measurement, Computer, Communication and Control (IMCCC)*, páginas 1081–1085, julho 2018. ISSN: 2373-6844. 19

[39] Sun, Jingwen, Weixing Du, Niancai Shi, Jingwen Sun, Weixing Du e Niancai Shi: *A Survey of kNN Algorithm*. Information Engineering and Applied Computing, 1(1), maio 2018, ISSN 2630-4619, 2630-4619. `https://ojs.whioce.com/index.php/ieac/article/view/770`, acesso em 2023-07-27. 19

[40] Zhao, Yan, Yun Qian e Cuixia Li: *Improved KNN text classification algorithm with MapReduce implementation*. Em *2017 4th International Conference on Systems and Informatics (ICSAI)*, páginas 1417–1422, novembro 2017. 19

[41] Arshad, Waqas, Muhammad Ali, Muhammad Mumtaz Ali, Arifa Javed e Saddam Hussain: *Multi-Class Text Classification: Model Comparison and Selection*. Em *2021 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, páginas 1–5, junho 2021. 20

[42] Hassan, Sayar Ul, Jameel Ahamed e Khaleel Ahmad: *Analytics of machine learning-based algorithms for text classification*. Sustainable Operations and Computers, 3:238–248, 2022, ISSN 26664127. Publisher: Elsevier BV. 20

[43] LeCun, Yann, Yoshua Bengio e Geoffrey Hinton: *Deep learning*. Nature, 521(7553):436–444, maio 2015, ISSN 1476-4687. `https://www.nature.com/articles/nature14539`, acesso em 2023-07-30, Number: 7553 Publisher: Nature Publishing Group. 21, 24, 26

[44] Mitchell, Tom M.: *Machine Learning*. McGraw-Hill Science/Engineering/Math, New York, 1ª edição edição, março 1997, ISBN 978-0-07-042807-2. 21

[45] Albawi, Saad, Tareq Abed Mohammed e Saad Al-Zawi: *Understanding of a convolutional neural network*. Em *2017 International Conference on Engineering and Technology (ICET)*, páginas 1–6, agosto 2017. 22, 23

[46] Kim, Yoon: *Convolutional Neural Networks for Sentence Classification*. Em *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, páginas 1746–1751, Doha, Qatar, outubro 2014. Association for Computational Linguistics. `https://aclanthology.org/D14-1181`, acesso em 2023-07-31. 22, 23

[47] Johnson, Rie e Tong Zhang: *Effective Use of Word Order for Text Categorization with Convolutional Neural Networks*. CoRR, abs/1412.1058, 2014. `http://arxiv.org/abs/1412.1058`, arXiv: 1412.1058. 22

[48] Zhang, Ye e Byron C. Wallace: *A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification*. CoRR, abs/1510.03820, 2015. `http://arxiv.org/abs/1510.03820`, arXiv: 1510.03820. 22, 23

[49] DSA, Equipe: *Capítulo 8 - Função de Ativação*, dezembro 2022. `https://www.deeplearningbook.com.br/funcao-de-ativacao/`, acesso em 2023-08-01. 22

[50] Pouyanfar, Samira, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei Ling Shyu, Shu Ching Chen e S. S. Iyengar: *A Survey on Deep Learning: Algorithms, Techniques, and Applications*. ACM Computing Surveys, 51(5):92:1–92:36, 2018, ISSN 0360-0300. `https://doi.org/10.1145/3234150`, acesso em 2023-08-01. 22, 24

[51] Graves, Alex, Abdel rahman Mohamed e Geoffrey Hinton: *Speech recognition with deep recurrent neural networks*. Em *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, páginas 6645–6649, maio 2013. ISSN: 2379-190X. 24

[52] Avijeet Biswal: *Recurrent Neural Network (RNN) Tutorial: Types and Examples [Updated] | Simplilearn.* https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn, acesso em 2023-08-01. 24

[53] Bengio, Y., P. Simard e P. Frasconi: *Learning long-term dependencies with gradient descent is difficult.* IEEE Transactions on Neural Networks, 5(2):157–166, março 1994, ISSN 1941-0093. Conference Name: IEEE Transactions on Neural Networks. 25

[54] Pascanu, Razvan, Tomas Mikolov e Yoshua Bengio: *On the difficulty of training Recurrent Neural Networks*, 2013. https://arxiv.org/abs/1211.5063, acesso em 2023-05-21, _eprint: 1211.5063. 25

[55] *Gradiente*, outubro 2022. https://pt.wikipedia.org/w/index.php?title=Gradiente&oldid=64610822, acesso em 2023-08-02, Page Version ID: 64610822. 25

[56] Hochreiter, Sepp e Jürgen Schmidhuber: *Long Short-Term Memory.* Neural Computation, 9(8):1735–1780, novembro 1997, ISSN 0899-7667. https://doi.org/10.1162/neco.1997.9.8.1735, acesso em 2023-05-21. 26

[57] Christopher Olah: *Understanding LSTM Networks*, agosto 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/, acesso em 2023-05-21. 26, 27

[58] Yang, JinXiong, Liang Bai e Yanming Guo: *A survey of text classification models.* Em *Proceedings of the 2020 2nd International Conference on Robotics, Intelligent Control and Artificial Intelligence*, RICAI '20, páginas 327–334, New York, NY, USA, 2020. Association for Computing Machinery, ISBN 978-1-4503-8830-6. https://doi.org/10.1145/3438872.3439101, acesso em 2023-08-04. 27

[59] Greff, Klaus, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink e Jürgen Schmidhuber: *LSTM: A Search Space Odyssey.* IEEE Transactions on Neural Networks and Learning Systems, 28(10):2222–2232, outubro 2017, ISSN 2162-237X, 2162-2388. http://arxiv.org/abs/1503.04069, acesso em 2023-05-23, arXiv:1503.04069 [cs]. 27

[60] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser e Illia Polosukhin: *Attention Is All You Need.* 2017. http://arxiv.org/abs/1706.03762, acesso em 2023-08-04, arXiv:1706.03762 [cs]. 28, 33, 34, 35, 36, 37, 38

[61] Tunstall, Lewis, Leandro von Werra e Thomas Wolf: *Natural Language Processing with Transformers, Revised Edition.* O'Reilly Media, Sebastopol, CA, 1st edition edição, julho 2022, ISBN 978-1-09-813679-6. 28, 29, 31, 32, 33, 34, 36, 63

[62] Sutskever, Ilya, Oriol Vinyals e Quoc V. Le: *Sequence to sequence learning with neural networks.* Em *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, páginas 3104–3112, Cambridge, MA, USA, 2014. MIT Press. 29

[63] Bahdanau, Dzmitry, Kyung Hyun Cho e Yoshua Bengio: *Neural machine translation by jointly learning to align and translate*. San Diego, United States, janeiro 2015. http://www.scopus.com/inward/record.url?scp=85062889504&partnerID=8YFLogxK, acesso em 2023-08-05. 29, 30, 31

[64] Goodfellow, Ian, Yoshua Bengio e Aaron Courville: *Deep Learning*. Adaptive Computation and Machine Learning. MIT Press, 2016, ISBN 978-0-262-03561-3. 31

[65] Yosinski, Jason, Jeff Clune, Yoshua Bengio e Hod Lipson: *How transferable are features in deep neural networks?* 2014. https://arxiv.org/abs/1411.1792, acesso em 2023-08-07. 31, 32

[66] Howard, Jeremy e Sebastian Ruder: *Universal Language Model Fine-tuning for Text Classification*. 2018. https://arxiv.org/abs/1801.06146, acesso em 2023-08-10. 33

[67] Minaee, Shervin, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu e Jianfeng Gao: *Deep Learning–based Text Classification: A Comprehensive Review*. ACM Computing Surveys, 54(3):62:1–62:40, 2021, ISSN 0360-0300. https://doi.org/10.1145/3439726, acesso em 2023-08-04. 33, 35

[68] Alammar, Jay: *The Illustrated Transformer*, junho 2018. https://jalammar.github.io/illustrated-transformer/, acesso em 2023-08-11. 33, 35, 36, 37

[69] Devlin, Jacob, Ming Wei Chang, Kenton Lee e Kristina Toutanova: *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019. https://arxiv.org/abs/1810.04805, acesso em 2023-05-13, _eprint: 1810.04805. 37, 38

[70] Adhikari, Ashutosh, Achyudh Ram, Raphael Tang e Jimmy Lin: *DocBERT: BERT for Document Classification*, agosto 2019. http://arxiv.org/abs/1904.08398, acesso em 2023-08-14, arXiv:1904.08398 [cs]. 37

[71] González-Carvajal, Santiago e Eduardo C. Garrido-Merchán: *Comparing BERT against traditional machine learning text classification*. Journal of Computational and Cognitive Engineering, maio 2020, ISSN 28109503. http://arxiv.org/abs/2005.13012, acesso em 2023-08-14, arXiv:2005.13012 [cs, stat]. 37

[72] Sun, Chi, Xipeng Qiu, Yige Xu e Xuanjing Huang: *How to Fine-Tune BERT for Text Classification?*, maio 2019. http://arxiv.org/abs/1905.05583, acesso em 2023-08-14, arXiv:1905.05583 [cs]. 37, 52

[73] CNJ: *Justiça em Números*, 2024. https://www.cnj.jus.br/pesquisas-judiciarias/justica-em-numeros/, acesso em 2024-12-14. 40

[74] Marinato, Matheus Serrão, Antonio F. L. Jacob Junior, Fábio M. F. Lobato e Omar A. C. Cortes: *Classificação Automática de Petições Iniciais Usando Classificadores Combinados*. Em *Anais do XVI Brazilian e-Science Workshop (BRESCI 2022)*, páginas 89–96. Sociedade Brasileira de Computação - SBC, julho 2022. 40, 41

_BR_BR

[75] Sousa, Rogério Nogueira de: *MINERJUS: solução de apoio à classificação processual com uso de Inteligência Artificial*. Tese de Doutoramento, Universidade Federal do Tocantins, 2019. http://repositorio.uft.edu.br/handle/11612/1446, acesso em 2023-05-06, Accepted: 2019-10-25T14:13:39Z. 41, 42

[76] Almeida Neto, Manoel Alves de, Vinícius Malloni Moura, Jonathan Da Silva Bandeira, Pedro Rudá Freitas e Roberta Andrade de Araújo Fagundes: *Um modelo de inferência para a classificação de resultados processuais da Justiça Estadual*. Revista de Engenharia e Pesquisa Aplicada, 3(3), agosto 2018, ISSN 2525-4251. 42

[77] Noguti, Mariana Y., Eduardo Vellasques e Luiz S. Oliveira: *Legal Document Classification: An Application to Law Area Prediction of Petitions to Public Prosecution Service*. Proceedings of the International Joint Conference on Neural Networks, julho 2020. arXiv: 2010.12533 Publisher: Institute of Electrical and Electronics Engineers Inc. ISBN: 9781728169262. 43

[78] Leme, Bruno: *Classificação automática de documentos de características econômicas para defesa jurídica*, maio 2021. https://www.teses.usp.br/teses/disponiveis/45/45134/tde-05082021-152340/pt-br.php, acesso em 2023-04-25. 44

[79] *What are tokens and how to count them? | OpenAI Help Center*. https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them, acesso em 2024-09-27. 51

[80] Souza, Fábio, Rodrigo Nogueira e Roberto Lotufo: *BERTimbau: pretrained BERT models for Brazilian Portuguese*. Em *9th Brazilian Conference on Intelligent Systems, BRACIS, Rio Grande do Sul, Brazil, October 20-23 (to appear)*, 2020, ISBN 978-3-030-61377-8. 52, 53

[81] Rodrigues, João, Luís Gomes, João Silva, António Branco, Rodrigo Santos, Henrique Lopes Cardoso e Tomás Osório: *Advancing Neural Encoding of Portuguese with Transformer Albertina PT-**, maio 2023. http://arxiv.org/abs/2305.06721, acesso em 2023-08-15, arXiv:2305.06721 [cs]. 52, 53

[82] Kudo, Taku e John Richardson: *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. CoRR, abs/1808.06226, 2018. http://arxiv.org/abs/1808.06226, arXiv: 1808.06226. 52

[83] Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes e Jeffrey Dean: *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*, outubro 2016. http://arxiv.org/abs/1609.08144, acesso em 2023-08-20, arXiv:1609.08144 [cs]. 53

[84] Lewis, Patrick, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel e Douwe Kiela: *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. 2020. https://arxiv.org/abs/2005.11401, acesso em 2024-09-05. 53

[85] *Build a Retrieval Augmented Generation (RAG) App: Part 1 | LangChain*. https://python.langchain.com/docs/tutorials/rag/, acesso em 2025-02-27. 54, 55

[86] *Checksum*, setembro 2024. https://en.wikipedia.org/w/index.php?title=Checksum&oldid=1247103467, acesso em 2024-09-28, Page Version ID: 1247103467. 54

[87] *Cryptographic hash function*, setembro 2024. https://en.wikipedia.org/w/index.php?title=Cryptographic_hash_function&oldid=1248067098, acesso em 2024-09-28, Page Version ID: 1248067098. 54

[88] *SHA-2*, setembro 2024. https://en.wikipedia.org/w/index.php?title=SHA-2&oldid=1248158630, acesso em 2024-09-28, Page Version ID: 1248158630. 54

[89] *Cosine similarity*, agosto 2024. https://en.wikipedia.org/w/index.php?title=Cosine_similarity&oldid=1238947356, acesso em 2024-09-28, Page Version ID: 1238947356. 55

[90] Dai, Haixing, Zhengliang Liu, Wenxiong Liao, Xiaoke Huang, Yihan Cao, Zihao Wu, Lin Zhao, Shaochen Xu, Wei Liu, Ninghao Liu, Sheng Li, Dajiang Zhu, Hongmin Cai, Lichao Sun, Quanzheng Li, Dinggang Shen, Tianming Liu e Xiang Li: *AugGPT: Leveraging ChatGPT for Text Data Augmentation*, março 2023. http://arxiv.org/abs/2302.13007, acesso em 2024-03-04, arXiv:2302.13007 [cs]. 58

[91] Luyang Fang, Gyeong-Geon Lee e Xiaoming Zhai: *Using GPT-4 to Augment Unbalanced Data for Automatic Scoring*. 2023. https://rgdoi.net/10.13140/RG.2.2.20161.74087, acesso em 2024-03-04. 58

[92] Loshchilov, Ilya e Frank Hutter: *Decoupled Weight Decay Regularization*, janeiro 2019. http://arxiv.org/abs/1711.05101, acesso em 2024-10-19, arXiv:1711.05101. 63