UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

Gabriel Andreazi Bertho

Arquiteturas híbridas explicáveis com modelos lineares e LLMs para classificação de peças processuais: estudo de caso no Supremo Tribunal Federal

São Carlos - SP

2025

Gabriel Andreazi Bertho

Arquiteturas híbridas explicáveis com modelos lineares e LLMs para classificação de peças processuais: estudo de caso no Supremo Tribunal Federal

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia de Computação da Universidade Federal de São Carlos, como requisito para a obtenção do título de Bacharel em Engenharia de Computação.

Orientação Prof. Dr. Alexandre Magalhães Levada

São Carlos - SP

2025

Dedico este trabalho, com todo o meu carinho e gratidão, aos meus pais, Fabiana e Fernando, à minha namorada Maria Fernanda, à minha madrasta Maria Zilda, aos meus avós Eunice, Maria Clara (in memoriam) e Guilherme, e aos meus familiares e amigos.

Agradecimentos

A realização deste trabalho marca o encerramento de um ciclo acadêmico marcado por muito crescimento pessoal, inúmeras mudanças de trajetórias, correção de rotas e superações. Nenhum caminho é trilhado sozinho, e chegar até aqui só foi possível graças à presença e apoio de pessoas muito especiais, às quais expresso aqui a minha mais sincera gratidão.

Agradeço, em primeiro lugar, aos meus pais, Fabiana e Fernado, pelo amor incondicional, pelos sacrifícios silenciosos e pela base que sempre me proporcionaram. Sem dúvidas, tudo o que conquistei até hoje tem raízes no que recebi de vocês, que nunca mediram esforços para que eu pudesse ter as melhores oportunidades na vida. Agradeço, também, à minha madrasta, Maria Zilda, pela presença constante e apoio ao longo da caminhada, e principalmente por sempre me tratar como um filho.

Aos meus avós, Eunice, Guilherme e Maria Clara (in memoriam) que deixaram em mim valores preciosos e memórias inesquecíveis. Sinto muita gratidão pelo legado que construíram e pela presença que moldou a minha infância e juventude.

À minha namorada, Maria Fernanda, minha profunda gratidão pelo amor, pelo companheirismo, pela paciência e por estar ao meu lado mesmo nos períodos mais desafiadores. A sua presença ao longo de todos esses anos foi essencial para que eu conseguisse seguir em frente. Estendo este agradecimento à sua família, em especial ao meu cunhado João Vitor, à minha sogra Gabriela, que sempre me recebeu como um filho, à vó Cida (Maria Aparecida), que sempre me acolheu como neto, e à tia Ivone, que desde o início me recebeu com tanto carinho. Todos vocês se tornaram parte importante da minha trajetória.

Aos meus irmãos, José Eduardo, Luísa e Miguel, e a toda minha família - tios, primos e demais entes queridos - agradeço por serem fonte constante de alegria, apoio e inspiração.

Aos amigos que fiz durante a graduação, vocês se tornaram parte da minha família e possuem a minha mais sincera gratidão por tornarem essa jornada mais leve, divertida e memorável.

Ao professor Stefan Johansson, agradeço por tornar a minha experiência universitária ainda mais enriquecedora, com aulas de sueco que foram também momentos de acolhimento, cultura e entusiasmo que marcaram muito positivamente a minha graduação.

Agradeço profundamente ao professor Roberto Santos Inoue e aos membros do Laboratory of Autonomous Robots and Intelligent Systems (LARIS) da UFSCar, pela confiança, ensinamentos e pela oportunidade de participar de projetos que mudaram a

minha visão sobre o mundo.

Agradeço também à Visagio e à VLS, empresas que tiveram papel fundamental na minha formação profissional neste período final da graduação. Em especial, deixo meu agradecimento a Renato Miyaji e Renato Moulin, pelo apoio, pela confiança, pela receptividade e por abrirem espaço para que eu aprendesse com os temas desafiadores desenvolvidos neste trabalho.

Ao meu orientador, professor Alexandre Levada, minha gratidão pela paciência, pela orientação e pelo incentivo constante. Foi um privilégio poder contar com sua experiência e apoio durante este percurso.

"No man should escape our universities without knowing how little he knows."
J.Robert Oppenheimer

Resumo

O crescimento do volume de processos nos tribunais superiores brasileiros tem impulsionado o uso de aprendizado de máquina para apoiar tarefas de triagem e organização de documentos jurídicos. Neste trabalho, investigamos a classificação automática de peças processuais do Supremo Tribunal Federal (STF) em seis tipos canônicos (recurso extraordinário, agravo em recurso extraordinário, sentença, acórdão, despacho e outros) utilizando o conjunto de dados Victor, bem como a geração de explicações em linguagem natural para as decisões do modelo. Comparamos baselines clássicos baseados em TF–IDF com classificadores lineares (SVM e Naive Bayes) a dois modelos de linguagem Gemma 3 ajustados para *sequence classification* (270M parâmetros com *full fine-tuning* e 4B parâmetros com QLoRA). Os resultados mostram que o Naive Bayes, embora apresente acurácia alta, falha nas classes minoritárias, resultando em F1-macro muito baixa, enquanto a SVM com pesos balanceados atinge desempenho robusto (F1-macro em torno de 0,82 no teste) e já competitivo para o problema estudado; os modelos Gemma 3 270M e 4B QLoRA obtêm ganhos apenas modestos em F1-macro, reforçando que, para tipificação de peças, sinais lexicais simples capturam grande parte da estrutura da tarefa. Em seguida, propomos arquiteturas híbridas em que um motor de decisão (SVM ou Gemma 3 270M) é combinado a uma LLM Gemma 3 4B em modo conversa, encarregada de gerar explicações em português jurídico condicionadas a *n-grams* relevantes extraídos pelo classificador linear. A qualidade das explicações é avaliada por uma métrica simples de fidelidade lexical (*feature coverage*), com coberturas médias em torno de 0,79 no cenário SVM + LLM e valores muito próximos (aproximadamente 0,76 vs. 0,77) na comparação entre SVM e Gemma 3 270M como motores de decisão. Os resultados sugerem que a combinação TF–IDF + SVM constitui uma base leve e eficaz para classificação de tipo de peça no Victor, enquanto arquiteturas híbridas com LLM explicadora oferecem um compromisso pragmático entre desempenho, custo computacional e transparência mínima das decisões.

Palavras-chave: Classificação de documentos jurídicos; Modelos de Linguagem de Grande Escala (LLMs); Inteligência Artificial Explicável; Dataset Victor

Abstract

The growing volume of cases in Brazil's higher courts has fostered the use of machine learning to support the triage and organization of legal documents. This work investigates automatic classification of Brazilian Supreme Court (STF) documents into six canonical types (Extraordinary Appeal, Interlocutory Appeal in Extraordinary Appeal, Judgment, Appellate Decision, Admissibility Order, and Others) using the Victor dataset, as well as the generation of natural-language explanations for model decisions. We compare classical baselines based on TF–IDF and linear classifiers (SVM and Naive Bayes) with two Gemma 3 language models fine-tuned for sequence classification (270M parameters with full fine-tuning and a 4B-parameter model trained via QLoRA). Results show that Naive Bayes, despite high accuracy, collapses on minority classes and yields very low macro-F1, whereas a class-weighted SVM achieves robust performance (test macro-F1 around 0.82) and is already competitive for this task; Gemma 3 270M and 4B QLoRA deliver only modest macro-F1 gains, indicating that simple lexical cues capture most of the structure of document type classification. We then propose hybrid architectures in which a decision engine (SVM or Gemma 3 270M) is combined with a Gemma 3 4B chat LLM that generates legal Portuguese explanations conditioned on relevant *n-grams* extracted by the linear classifier. Explanation quality is assessed through a simple lexical fidelity metric (*feature coverage*), with average coverage around 0.79 in the SVM + LLM scenario and very similar values (approximately 0.76 vs. 0.77) when comparing SVM and Gemma 3 270M as decision engines. Overall, the results suggest that TF–IDF + SVM provides a lightweight and effective baseline for document type classification on Victor, while hybrid architectures with an LLM explainer offer a pragmatic trade-off between performance, computational cost, and minimal transparency of automatic decisions.

Keywords: Legal Document Classification; Large Language Models; Explainable Artificial Intelligence; Victor Dataset

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ARE	Agravo em Recurso Extraordinário
BERT	*Bidirectional Encoder Representations from Transformers*
CNJ	Conselho Nacional de Justiça
IA	Inteligência Artificial
LIME	*Local Interpretable Model-agnostic Explanations*
LLM	*Large Language Model* (Modelo de Linguagem de Grande Escala)
LoRA	*Low-Rank Adaptation*
NB	*Naive Bayes*
PJe	Processo Judicial Eletrônico
PLN	Processamento de Linguagem Natural
PEFT	*Parameter-Efficient Fine-Tuning*
QLoRA	*Quantized Low-Rank Adaptation*
RE	Recurso Extraordinário
SHAP	*SHapley Additive exPlanations*
STF	Supremo Tribunal Federal
SVM	*Support Vector Machine* (Máquina de Vetores de Suporte)
TF–IDF	*Term Frequency–Inverse Document Frequency*
XAI	*Explainable Artificial Intelligence* (Inteligência Artificial Explicável)

Sumário

1 Introdução

O Poder Judiciário brasileiro convive, há anos, com um grande acúmulo de processos. O relatório *Justiça em Números 2024* indica cerca de 83,8 milhões de processos em tramitação em 2023 e 35,3 milhões de casos novos apenas nesse ano (Conselho Nacional de Justiça, 2024). Esse volume pressiona toda a estrutura da Justiça e afeta, em especial, os tribunais superiores, responsáveis por uniformizar a interpretação da Constituição. No Supremo Tribunal Federal (STF), isso aparece em filas de recursos extraordinários e agravos que exigem análise prévia de admissibilidade, exame da repercussão geral e identificação de precedentes, tarefas que consomem muito tempo de servidores e ministros.

Diante desse cenário, o Judiciário intensificou a digitalização dos processos e passou a investir em automação para ganhar agilidade. Ferramentas como o Processo Judicial Eletrônico (PJe) e programas de inovação tecnológica passaram a integrar a rotina dos tribunais. Nesse movimento, o Projeto Victor, desenvolvido pelo STF em parceria com a Universidade de Brasília, tornou-se uma das principais referências: o sistema utiliza modelos de aprendizado de máquina para separar peças processuais relevantes e classificar recursos por temas de repercussão geral, apoiando o juízo de admissibilidade dos recursos extraordinários (FILHO; JUNQUILHO, 2018).

Como grande parte da atividade judicial é baseada em textos — petições, decisões, acórdãos e pareceres —, técnicas de Processamento de Linguagem Natural (PLN) passaram a ter papel central nessas iniciativas. Revisões sistemáticas mostram que métodos de aprendizado de máquina, como máquinas de vetores de suporte (SVM), redes neurais e modelos baseados em *embeddings*, vêm sendo aplicados com bons resultados em tarefas de classificação de documentos jurídicos, por exemplo na identificação de tipos de ações, áreas do direito e resultados de processos (MARTINS; SILVA, 2022). No contexto brasileiro, o próprio Projeto Victor deu origem a bases de dados padronizadas com peças de recursos extraordinários e agravos, o que estimulou estudos focados em classificar automaticamente recursos por temas de repercussão geral e em apoiar a filtragem de processos que chegam ao STF (FILHO; JUNQUILHO, 2018). Esses trabalhos indicam que modelos de PLN podem reduzir o tempo gasto em tarefas repetitivas de leitura e triagem de processos.

Por outro lado, apenas obter boas métricas de desempenho não é suficiente quando se trata de decisões que afetam direitos fundamentais. Parte da literatura em Direito e tecnologia chama a atenção para o risco de que sistemas algorítmicos se tornem verdadeiras "caixas-pretas", em que nem mesmo os operadores do direito compreendem de forma clara como a decisão foi produzida (MARANHÃO; FLORÊNCIO; ALMADA, 2021). No caso específico do Projeto Victor, estudos destacam que, embora o sistema seja eficiente

para triagem, ainda existem desafios ligados à transparência do seu funcionamento e à forma como os resultados são apresentados aos magistrados (VALLE, 2023). A Resolução nº 332/2020 reforça que ferramentas de IA devem atuar apenas como apoio e que a responsabilidade final permanece com o julgador humano, o que aumenta a importância de mecanismos de explicabilidade e de controle (Conselho Nacional de Justiça, 2020).

É nesse ponto que se insere o problema central deste trabalho. De um lado, modelos de classificação de textos jurídicos já mostram bom desempenho em tarefas como a triagem de recursos que chegam ao STF. De outro, ainda é pouco explorado como tornar essas classificações mais transparentes e compreensíveis para juízes, servidores e demais atores do sistema de justiça. Em linha com a discussão sobre os impactos da inteligência artificial no Direito (MARANHÃO; FLORÊNCIO; ALMADA, 2021), este Trabalho de Conclusão de Curso investiga uma abordagem que combina modelos de aprendizado de máquina tradicionais, responsáveis pela classificação automática de peças processuais, com grandes modelos de linguagem capazes de gerar explicações textuais em linguagem natural para cada decisão automatizada. A ideia é aproximar o desempenho desses modelos das exigências de justificativa próprias do ambiente jurídico, contribuindo para um uso de IA que seja, ao mesmo tempo, eficiente e mais alinhado aos princípios de transparência e legitimidade do processo decisório no Supremo Tribunal Federal.

1.1 Objetivo

Este trabalho tem como objetivo investigar uma abordagem de apoio à triagem de recursos constitucionais que chegam ao Supremo Tribunal Federal, combinando modelos de aprendizado de máquina tradicionais e grandes modelos de linguagem, empregados tanto na classificação automática de textos jurídicos quanto na geração de explicações em linguagem natural sobre essas classificações.

1.2 Objetivos Específicos

Para atingir o objetivo geral, este trabalho foi dividido nos seguintes objetivos específicos:

1. Treinar e avaliar modelos de classificação de textos jurídicos, incluindo abordagens de aprendizado de máquina tradicionais e grandes modelos de linguagem, utilizando a base de dados adotada neste trabalho.

2. Desenvolver um fluxo de geração de explicações em linguagem natural, a partir das saídas dos modelos de classificação e dos textos originais dos recursos, utilizando um grande modelo de linguagem.

3. Explorar diferentes estratégias de *prompting* e de fornecimento de contexto ao modelo gerador de explicações, avaliando a coerência das justificativas produzidas em relação às decisões de classificação.

4. Analisar os resultados obtidos, discutindo o potencial de uso da abordagem proposta como apoio à triagem de recursos constitucionais no Supremo Tribunal Federal, bem como suas limitações e riscos.

2 Fundamentação Teórica

Este capítulo apresenta os conceitos teóricos que embasam o problema de classificação automática de documentos jurídicos e as abordagens adotadas neste trabalho. Inicialmente, descreve-se o sistema de justiça brasileiro e os recursos constitucionais no Supremo Tribunal Federal, bem como o Projeto Victor e o conjunto de dados VICTOR utilizados nos experimentos, situando o problema em seu contexto jurídico e computacional.

Em seguida, a Seção 2.2 revisa noções básicas de aprendizado de máquina supervisionado e discute o impacto do desbalanceamento de classes na função de perda. A partir daí, são apresentadas diferentes formas de representação vetorial de textos, incluindo o modelo de saco de palavras e o esquema TF–IDF na Seção 2.3.2, e discutidos modelos clássicos para classificação de texto na Seção 2.4, com destaque para o Naive Bayes Multinomial (2.4.1) e para SVM lineares (2.4.2). Na sequência, introduzem-se conceitos de Processamento de Linguagem Natural e de modelos de linguagem baseados na arquitetura Transformer, detalhando o mecanismo de atenção e autoatenção (2.6.1), a estrutura de decoders (2.6.2) e a adaptação de Transformers para tarefas de classificação de sequência (2.6.3), bem como o uso de modelos de conversa para geração condicional (2.6.4).

Por fim, o capítulo discute técnicas de *fine-tuning* de modelos de linguagem na Seção 2.7, com ênfase em abordagens eficientes em parâmetros como PEFT (2.7.1) e LoRA/QLoRA (2.7.2), e apresenta noções de explicabilidade em modelos de classificação de texto na Seção 2.8, incluindo a métrica de fidelidade *feature coverage* (2.8.4). As principais métricas de avaliação usadas ao longo do trabalho, como acurácia e F1-macro, são revisadas na Seção 2.9.

2.1 Sistema de Justiça e Projeto Victor

2.1.1 Recursos constitucionais no Supremo Tribunal Federal

No sistema de justiça brasileiro, os processos normalmente começam em um juízo de primeira instância e podem ser revistos por tribunais de segunda instância. Em alguns casos, ainda é possível levar a discussão ao Supremo Tribunal Federal (STF), por meio de recursos chamados constitucionais. O principal deles é o *recurso extraordinário*, utilizado quando a parte alega que a decisão de um tribunal violou a Constituição Federal (Tribunal de Justiça do Distrito Federal e dos Territórios, 2023). De forma simplificada, o objetivo desse recurso não é reavaliar provas, mas verificar se houve algum problema na aplicação ou interpretação da Constituição.

Antes de chegar ao STF, o recurso extraordinário passa por uma análise preliminar

no tribunal de origem, que pode aceitar ou recusar o envio do processo à Corte. Se o tribunal decidir não encaminhar o recurso, a parte pode apresentar um *agravo em recurso extraordinário* (ARE), que é um pedido para que essa decisão seja revista e o caso seja efetivamente analisado pelo STF (Tribunal de Justiça do Distrito Federal e dos Territórios, 2023). Além dessas peças, os processos que chegam ao STF incluem outros documentos, como acórdãos (decisões de tribunais), sentenças (decisões de primeira instância), despachos (atos mais simples de andamento do processo) e diferentes tipos de petições e anexos.

Do ponto de vista deste trabalho, cada um desses documentos pode ser visto como um texto a ser classificado em uma entre várias categorias possíveis. Esse é o problema de classificação de documentos que será explorado nos capítulos seguintes.

2.1.2 O Projeto Victor e a base de dados

O Projeto Victor é uma iniciativa do Supremo Tribunal Federal em parceria com a Universidade de Brasília que tem como objetivo usar técnicas de inteligência artificial para apoiar a triagem de recursos constitucionais que chegam ao STF (DIAS et al., 2023). A ideia central é utilizar modelos de aprendizado de máquina para ler automaticamente partes dos processos e ajudar a identificar quais casos tratam de determinados temas e quais merecem atenção especial dos ministros.

A partir dos processos digitais usados no Projeto Victor, foi construído o conjunto de dados conhecido como *VICTOR*, voltado para pesquisas em processamento de linguagem natural no domínio jurídico (ARAUJO et al., 2020). Essa base reúne dezenas de milhares de processos e centenas de milhares de documentos individuais, cada um associado a um tipo específico de peça processual. Na tarefa de classificação de tipo de documento, amplamente estudada na literatura, cada texto é rotulado em uma entre seis classes principais (SILVA et al., 2018; BRAZ et al., 2018; ARAUJO et al., 2023):

- **Acórdão**: decisão de um tribunal, geralmente de segunda instância;
- **Recurso Extraordinário (RE)**: peça em que a parte leva uma questão constitucional ao STF;
- **Agravo em Recurso Extraordinário (ARE)**: peça usada para contestar a decisão que recusou o envio de um recurso extraordinário ao STF;
- **Despacho**: ato simples do juiz ou ministro para dar andamento ao processo (por exemplo, determinar uma intimação ou juntar um documento);
- **Sentença**: decisão de mérito proferida por juiz de primeira instância;
- **Others (Outros)**: documentos que não se encaixam nas categorias anteriores, como petições intermediárias, certidões e anexos.

Na descrição original do conjunto de dados, Luz de Araujo et al. (ARAUJO et al., 2020) apresentam três versões do VICTOR (BVic, MVic e SVic). As versões Medium (MVic) e Small (SVic), mais utilizadas em estudos de classificação de tipo de documento, reúnem dezenas de milhares de processos e centenas de milhares de documentos, com forte desbalanceamento em favor da classe Others, como resumido na Tabela 1.

Tabela 1 – Métricas gerais das versões Medium (MVic) e Small (SVic) do dataset VICTOR, conforme descrito em (ARAUJO et al., 2020).

Versão	Processos	Documentos	Páginas	Nº de classes	% Docs. (Others)
MVic	44 855	628 820	2 086 899	6	96,5%
SVic	6 510	94 267	339 478	6	95,7%

Trabalhos anteriores já utilizaram o VICTOR para treinar modelos de classificação de textos jurídicos, com diferentes técnicas, desde representações simples baseadas em *bag-of-words* e máquinas de vetores de suporte até arquiteturas neurais mais complexas, como redes convolucionais e redes recorrentes (SILVA et al., 2018; BRAZ et al., 2018; ARAUJO et al., 2020; ARAUJO et al., 2023). Neste trabalho, o VICTOR é adotado como base para um problema de classificação multiclasse de documentos jurídicos, sobre o qual são aplicados os modelos e métodos de explicabilidade apresentados nos capítulos seguintes.

2.2 Aprendizado de Máquina Supervisionado

O aprendizado de máquina supervisionado é um ramo da inteligência artificial em que um modelo é treinado a partir de exemplos rotulados, isto é, pares de entrada e saída esperada. A ideia central é aprender uma função que, ao receber uma nova entrada, seja capaz de prever o rótulo correto com boa precisão (BISHOP, 2006; GOODFELLOW; BENGIO; COURVILLE, 2016). Em problemas de classificação, como neste trabalho, as saídas são categorias discretas (por exemplo, tipos de documentos jurídicos), enquanto em problemas de regressão as saídas são valores numéricos contínuos.

2.2.1 Conceitos básicos de classificação

Em termos gerais, um problema de classificação supervisionada é composto por um conjunto de treinamento $\{(x_i, y_i)\}_{i=1}^{N}$, em que cada x_i representa uma instância de entrada (por exemplo, o texto de um documento) e y_i é o rótulo associado (por exemplo, a classe do documento). O modelo recebe esses pares como exemplo e ajusta seus parâmetros para minimizar uma função de perda que mede o erro entre as previsões do modelo e os rótulos reais (GOODFELLOW; BENGIO; COURVILLE, 2016).

Em modelos de classificação baseados em redes neurais, a saída para cada exemplo x_i é um vetor de pontuações (ou *logits*) $z_\theta(x_i) \in \mathbb{R}^K$, em que K é o número de classes e θ representa os parâmetros do modelo. Essas pontuações são convertidas em probabilidades por meio da função *softmax*:

$$p_\theta(y = k \mid x_i) = \frac{\exp\Big(z_k(x_i)\Big)}{\sum_{j=1}^{K} \exp\Big(z_j(x_i)\Big)}, \quad k = 1, \ldots, K. \tag{2.1}$$

A função de perda mais usada em classificação multiclasse é a entropia cruzada. Para um exemplo rotulado (x_i, y_i), a perda é dada por:

$$\ell(x_i, y_i) = -\log p_\theta(y_i \mid x_i), \tag{2.2}$$

e a perda média no conjunto de treinamento é

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, y_i). \tag{2.3}$$

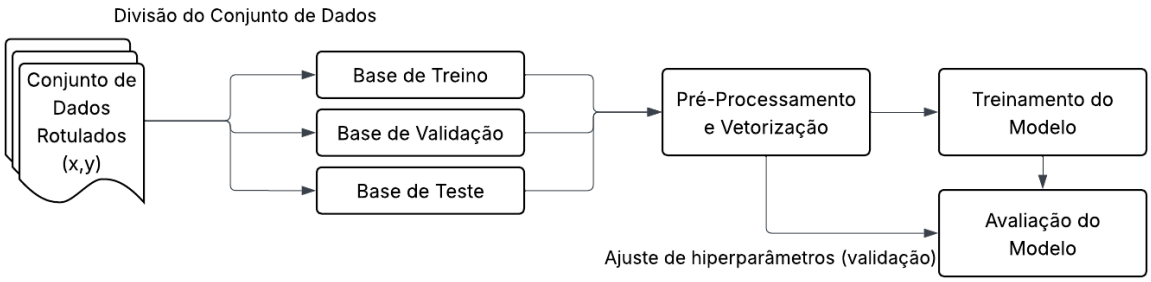
Na prática, o conjunto de dados disponível é dividido em subconjuntos com papéis distintos. O conjunto de *treinamento* é usado para ajustar os parâmetros do modelo minimizando a função de perda. O conjunto de *validação* é utilizado durante o desenvolvimento para escolher hiperparâmetros (como taxa de aprendizado, número de épocas ou tamanho do modelo) e para decidir quando interromper o treinamento, evitando sobreajuste (*overfitting*). Por fim, o conjunto de *teste* é reservado para a avaliação final, sendo composto por exemplos que não foram utilizados nem para o treinamento nem para a validação (BISHOP, 2006; GOODFELLOW; BENGIO; COURVILLE, 2016). Essa separação permite estimar de forma mais confiável a capacidade de generalização do modelo para novos dados.

De forma resumida, o fluxo de um problema de classificação supervisionada de textos, como o adotado neste trabalho, está esquematizado na Figura 1. Nele, um conjunto inicial de dados rotulados (x, y) é dividido em três subconjuntos: base de treino, base de validação e base de teste. Os exemplos de treino e validação passam por uma etapa comum de pré-processamento e vetorização dos textos e são utilizados no ajuste dos parâmetros do modelo e na escolha de hiperparâmetros. Por fim, o modelo treinado é avaliado em dados de teste, que também passam pelo mesmo pré-processamento, de modo a estimar seu desempenho em exemplos não vistos durante o treinamento.

2.2.2 Desbalanceamento de classes e ponderação da função de perda

Em muitos problemas reais de classificação, algumas classes aparecem com muito mais frequência do que outras. Esse fenômeno é conhecido como *desbalanceamento de classes*. Quando isso ocorre, um modelo pode obter uma alta acurácia apenas por acertar

Figura 1 – Esquema geral de um problema de classificação supervisionada de textos.



Fonte: elaborado pelo autor.

bem as classes majoritárias, mesmo tendo um desempenho ruim nas classes minoritárias (HE; GARCIA, 2009). Em contextos sensíveis, como o jurídico, isso pode ser problemático, pois classes com poucos exemplos podem ser justamente as mais importantes do ponto de vista prático.

Uma estratégia simples e bastante utilizada para mitigar esse problema é atribuir pesos diferentes às classes na função de perda. De forma geral, classes menos frequentes recebem pesos maiores, de modo que erros nessas classes "custem" mais durante o treinamento do modelo. Em modelos lineares e redes neurais, isso é feito por meio de uma versão ponderada da entropia cruzada (*weighted cross-entropy*), em que o termo de cada exemplo é multiplicado pelo peso da sua classe (BISHOP, 2006; GOODFELLOW; BENGIO; COURVILLE, 2016; HE; GARCIA, 2009).

Seja $w_k > 0$ o peso associado à classe k. A perda ponderada para um exemplo (x_i, y_i) pode ser escrita como:

$$\ell_{\mathrm{w}}(x_i, y_i) = -w_{y_i} \log p_\theta(y_i \mid x_i), \tag{2.4}$$

de modo que a contribuição de cada exemplo para a perda total depende tanto da probabilidade prevista quanto da importância relativa da sua classe. Os valores de w_k são tipicamente escolhidos em função da frequência de cada classe, como será detalhado no Capítulo 4.

2.3 Representação Vetorial de Textos

Modelos de aprendizado de máquina operam sobre vetores numéricos, não sobre texto bruto. Por isso, em tarefas de classificação de textos é necessário transformar cada documento em uma representação vetorial que capture, de forma compacta, o conteúdo relevante daquele texto (JURAFSKY; MARTIN, 2023; MANNING; RAGHAVAN; SCHÜTZE, 2008). Existem diversas estratégias para isso, desde modelos mais simples

baseados em contagem de palavras até representações densas aprendidas por redes neurais (*embeddings*). Nesta seção são apresentados os conceitos básicos de tokenização e n-grams, que servem de base para as representações utilizadas neste trabalho.

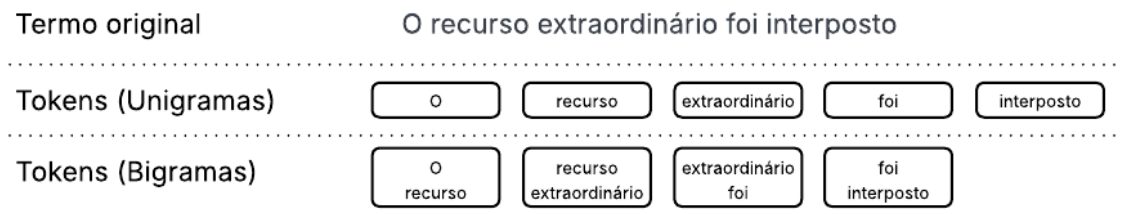
2.3.1 Tokens, vocabulário e n-grams

O primeiro passo para representar um texto é dividi-lo em unidades menores chamadas *tokens*. Em muitos modelos clássicos de processamento de linguagem natural, um token corresponde aproximadamente a uma palavra, obtida a partir da divisão do texto em espaços em branco e sinais de pontuação. Em modelos mais recentes, como os grandes modelos de linguagem, a tokenização costuma ser feita em nível de subpalavras, de modo que palavras raras possam ser decompostas em partes menores, mas o princípio permanece o mesmo: um texto é visto como uma sequência ordenada de tokens (JURAFSKY; MARTIN, 2023).

Dado um conjunto de documentos, o *vocabulário* é o conjunto de todos os tokens distintos observados nesse conjunto. Em representações tradicionais como *bag-of-words*, cada token do vocabulário passa a ocupar uma posição fixa em um vetor, e a presença ou frequência desse token em cada documento é usada como característica para o modelo de classificação (MANNING; RAGHAVAN; SCHÜTZE, 2008).

Uma forma simples de enriquecer essa representação é considerar não apenas tokens isolados, mas também sequências de tokens consecutivos, chamadas de n-grams. Um *unigrama* (1-gram) corresponde a um único token, um *bigrama* (2-gram) corresponde a uma sequência de dois tokens adjacentes, e assim por diante. Por exemplo, na expressão "recurso extraordinário", os unigramas seriam "recurso" e "extraordinário", enquanto o bigrama "recurso extraordinário" captura a expressão composta inteira. Em textos jurídicos, em que muitas expressões possuem significado próprio (como "agravo interno", "repercussão geral" ou "recurso extraordinário"), o uso de n-grams é especialmente útil para representar melhor essas combinações de termos. Essa ideia é ilustrada na Figura 2, que mostra a tokenização da frase e a formação de unigramas e bigramas a partir dos mesmos tokens.

Figura 2 – Exemplo de tokenização e formação de unigramas e bigramas.



Fonte: elaborado pelo autor.

2.3.2 Modelo de saco de palavras e TF–IDF

Uma forma clássica de representar textos para uso em modelos de aprendizado de máquina é o modelo de *saco de palavras* (*bag-of-words*). Nessa abordagem, a ordem dos tokens no texto é ignorada e cada documento é descrito apenas pela informação de quais termos do vocabulário aparecem e com que frequência (MANNING; RAGHAVAN; SCHÜTZE, 2008). A partir de um conjunto de documentos, constrói-se uma matriz termo–documento em que cada linha corresponde a um termo do vocabulário, cada coluna corresponde a um documento e cada célula armazena a contagem de ocorrências desse termo naquele documento.

Para ilustrar a representação vetorial, considere dois documentos curtos no domínio jurídico, denotados por D_1 e D_2, com o seguinte conteúdo textual:

- D_1: “O recurso extraordinário foi interposto”;
- D_2: “O agravo em recurso extraordinário foi provido”.

Considerando um vocabulário reduzido aos termos mais relevantes desses textos, a matriz termo–documento no modelo de saco de palavras pode ser escrita como na Tabela 2. Cada célula contém a contagem de ocorrências do termo na linha em cada um dos documentos D_1 e D_2.

Tabela 2 – Exemplo simplificado de matriz termo–documento com contagens de termos.

Termo	D_1	D_2
recurso	1	1
extraordinário	1	1
agravo	0	1
interposto	1	0
provido	0	1

Embora simples, essa representação se mostrou bastante eficaz em tarefas de classificação de textos, especialmente quando combinada com modelos lineares como Naive Bayes e máquinas de vetores de suporte (MANNING; RAGHAVAN; SCHÜTZE, 2008). Na prática, o vocabulário pode incluir não apenas unigramas, mas também n-grams, como discutido na subseção anterior, o que permite capturar expressões compostas relevantes.

Em vez de utilizar diretamente as contagens brutas de termos, é comum empregar um esquema de ponderação chamado TF–IDF (*term frequency–inverse document frequency*). A ideia é atribuir peso maior aos termos que são frequentes em um documento específico, mas raros na coleção como um todo (JURAFSKY; MARTIN, 2023). De forma simplificada,

a frequência do termo t no documento d pode ser representada por $\mathrm{tf}_{t,d}$, enquanto a frequência inversa do documento é dada por:

$$\mathrm{idf}_t = \log\left(\frac{N}{\mathrm{df}_t}\right), \tag{2.5}$$

em que N é o número total de documentos e df_t é o número de documentos em que o termo t aparece. O peso final TF–IDF é obtido pelo produto:

$$w_{t,d} = \mathrm{tf}_{t,d} \cdot \mathrm{idf}_t. \tag{2.6}$$

Cada documento passa, assim, a ser representado por um vetor w_d de dimensão igual ao tamanho do vocabulário. Esses vetores são tipicamente esparsos, isto é, a maior parte de suas posições é igual a zero, já que cada documento contém apenas uma fração dos termos possíveis. Essa esparsidade é explorada por implementações eficientes de modelos lineares, o que torna a combinação entre TF–IDF e classificadores como SVM e Naive Bayes bastante adequada para grandes coleções de texto (MANNING; RAGHAVAN; SCHÜTZE, 2008).

2.3.3 Embeddings e representações distribuídas

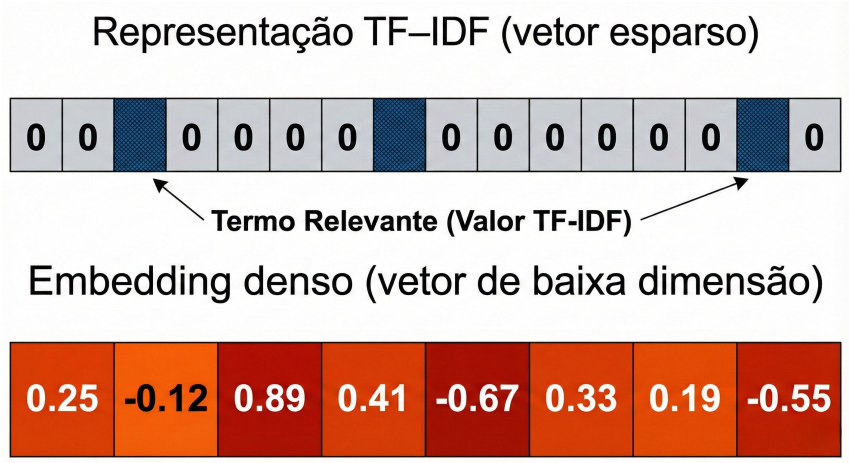
As representações baseadas em saco de palavras e TF–IDF tratam cada termo do vocabulário como uma dimensão independente em um vetor esparso. Nessa visão, palavras diferentes ocupam posições distintas e não há, na própria representação, informação direta sobre similaridade semântica entre termos. Por exemplo, os tokens “recurso” e “agravo” são representados por dimensões totalmente separadas, mesmo que estejam relacionados no contexto jurídico. Na parte superior da Figura 3, essa ideia é ilustrada por um vetor comprido em que a maior parte das posições assume valor zero, enquanto apenas algumas dimensões correspondentes a termos relevantes apresentam valores diferentes de zero.

A ideia de *embeddings* de palavras surgiu como uma alternativa mais expressiva a esse tipo de representação. Em vez de associar cada termo a uma dimensão de um vetor esparso, cada palavra (ou subpalavra) é mapeada para um vetor denso de dimensão fixa, tipicamente com dezenas ou centenas de componentes. Essas representações são chamadas de *distribuídas* porque a informação sobre o significado de cada termo é distribuída ao longo de várias dimensões, e não concentrada em uma única posição (BENGIO et al., 2003). Na parte inferior da Figura 3, esse conceito aparece como um vetor curto em que todas as componentes assumem valores reais distintos, mostrando que cada dimensão contribui um pouco para codificar o significado do termo ou documento. Modelos como Word2Vec aprendem esses vetores automaticamente a partir de grandes coleções de texto, ajustando os embeddings para que palavras que aparecem em contextos semelhantes tenham vetores próximos no espaço vetorial (MIKOLOV et al., 2013).

Em representações desse tipo, relações semânticas e sintáticas podem ser aproximadas por operações geométricas no espaço de embeddings. Termos que desempenham papéis parecidos em frases (por exemplo, "recurso extraordinário" e "recurso especial" em certos contextos jurídicos) tendem a ser mapeados para pontos próximos, enquanto termos menos relacionados, como "juiz" e "advogado", podem formar outro agrupamento. A Figura 4 mostra um exemplo bidimensional simplificado desse espaço: cada ponto representa o embedding projetado de um termo, e observa-se que "recurso extraordinário" e "recurso especial" aparecem em uma mesma região, indicando alta similaridade, enquanto "juiz" e "advogado" formam um grupo separado, mas também próximo entre si. Esse tipo de organização geométrica é o que permite que modelos de aprendizado de máquina utilizem não apenas a presença de um termo, mas também sua posição relativa em relação a outros termos.

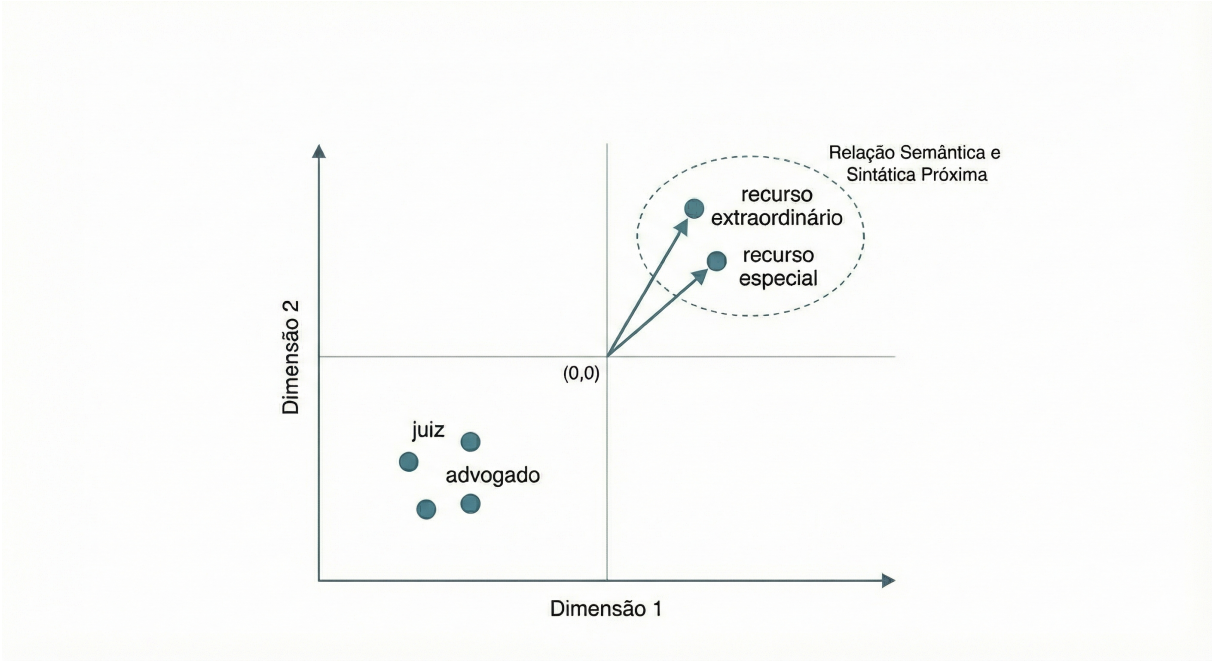
Modelos baseados em transformadores e grandes modelos de linguagem (LLMs) generalizam essa ideia ao produzir não apenas um único embedding por palavra, mas representações *contextuais*. Em vez de um vetor fixo para cada termo, a representação de um token passa a depender da frase em que ele aparece. Assim, a palavra "recurso" em "recurso extraordinário" pode receber um embedding diferente do que em "recurso administrativo", pois o modelo leva em conta o contexto completo da sentença (DEVLIN et al., 2019; JURAFSKY; MARTIN, 2023). Essas representações densas contextuais são geradas internamente pelo modelo e servem como base para camadas de classificação ou outras tarefas de processamento de linguagem natural.

Figura 3 – Comparação entre uma representação TF–IDF esparsa e um embedding denso de baixa dimensão.



Fonte: elaborado pelo autor.

Figura 4 – Exemplo ilustrativo de um espaço de embeddings em duas dimensões.



Fonte: elaborado pelo autor.

2.4 Modelos Clássicos para Classificação de Texto

Antes do surgimento dos modelos baseados em redes neurais profundas e em transformadores, a maior parte das aplicações de classificação de textos era baseada em modelos chamados aqui de *clássicos*, isto é, métodos de aprendizado de máquina supervisionado que operam sobre representações como saco de palavras e TF–IDF (SEBASTIANI, 2002). Nessa linha, classificadores lineares como Naive Bayes, regressão logística e máquinas de vetores de suporte (SVM) tornaram-se padrões de fato em tarefas como categorização de documentos, filtragem de spam e classificação de notícias (MANNING; RAGHAVAN; SCHÜTZE, 2008; JOACHIMS, 1998).

Esses modelos apresentam algumas vantagens importantes: são relativamente simples de treinar e de interpretar, funcionam bem com vetores esparsos de alta dimensão e têm custo computacional baixo, o que os torna adequados como *baselines* em estudos experimentais. Além disso, em muitos cenários de classificação de textos, o desempenho obtido por esses métodos ainda é competitivo em relação a abordagens mais recentes, quando os recursos computacionais ou o volume de dados anotados são limitados (SEBASTIANI, 2002).

Neste trabalho, dois desses modelos são utilizados como referência para comparar o desempenho de abordagens mais recentes baseadas em grandes modelos de linguagem: o Naive Bayes Multinomial, descrito na Seção 2.4.1, e as máquinas de vetores de suporte, descritas na Seção 2.4.2.

2.4.1 Naive Bayes Multinomial

O classificador Naive Bayes Multinomial é um modelo probabilístico amplamente utilizado em tarefas de classificação de textos. A ideia central é estimar, para cada classe c, a probabilidade de um documento x pertencer a essa classe, isto é, $P(c \mid x)$. Pela regra de Bayes, tem-se

$$P(c \mid x) \propto P(c)\, P(x \mid c), \tag{2.7}$$

em que $P(c)$ é a probabilidade *a priori* da classe e $P(x \mid c)$ é a probabilidade do documento dado a classe (MANNING; RAGHAVAN; SCHÜTZE, 2008).

Para textos representados no modelo de saco de palavras, cada documento é descrito por um vetor de contagens $x = (x_1, x_2, \ldots, x_V)$, em que x_j indica quantas vezes o termo j do vocabulário aparece no documento. O modelo multinomial assume que, dada a classe c, as ocorrências dos termos seguem uma distribuição multinomial parametrizada por $P(w_j \mid c)$, a probabilidade de observar o termo w_j em documentos da classe c. Sob a hipótese de independência condicional entre os termos, tem-se (MCCALLUM; NIGAM, 1998)

$$P(x \mid c) \propto \prod_{j=1}^{V} P(w_j \mid c)^{x_j}. \tag{2.8}$$

Na prática, trabalha-se no espaço logarítmico, de modo que a decisão do classificador consiste em escolher a classe que maximiza

$$\log P(c) + \sum_{j=1}^{V} x_j \log P(w_j \mid c). \tag{2.9}$$

Os parâmetros $P(w_j \mid c)$ são estimados a partir das contagens de termos nos documentos de treinamento de cada classe, usualmente com suavização de Laplace para evitar probabilidades nulas. Se N_{cj} denota o número total de ocorrências do termo w_j em documentos da classe c, então uma estimativa com suavização aditiva $\alpha > 0$ é dada por

$$\hat{P}(w_j \mid c) = \frac{N_{cj} + \alpha}{\sum_{k=1}^{V} N_{ck} + \alpha V}. \tag{2.10}$$

Esse modelo é particularmente adequado para classificação de textos porque lida naturalmente com vetores de contagens ou frequências de termos, assume uma distribuição multinomial coerente com o modelo de saco de palavras e pode ser treinado de forma muito eficiente mesmo em coleções grandes (MANNING; RAGHAVAN; SCHÜTZE, 2008; NIGAM et al., 2000). Apesar da hipótese de independência condicional entre termos ser bastante simplificadora, o Naive Bayes Multinomial costuma fornecer um *baseline* forte para classificação de documentos, com baixo custo computacional de treinamento e de predição.

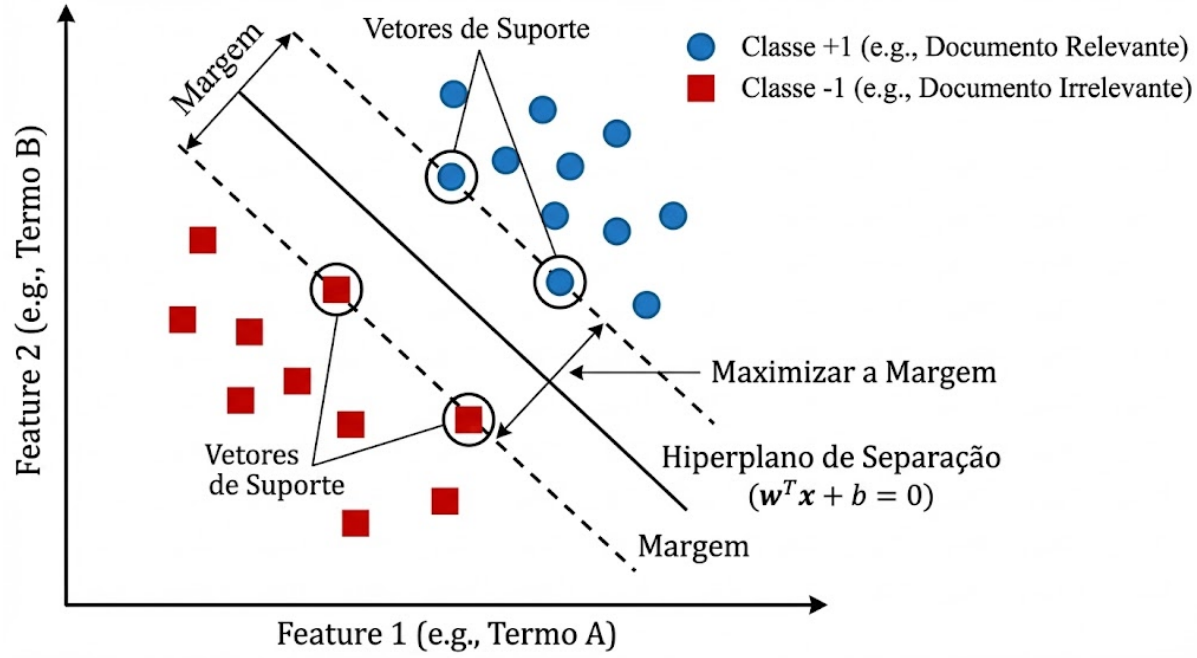
2.4.2 Máquinas de Vetores de Suporte (SVM) lineares

Máquinas de vetores de suporte (SVM, do inglês *Support Vector Machines*) são modelos de classificação que buscam encontrar um hiperplano que separe, com a maior margem possível, os exemplos de classes diferentes em um espaço vetorial (CORTES; VAPNIK, 1995; BISHOP, 2006). No caso binário, com rótulos $y_i \in \{-1, +1\}$ e vetores de características $x_i \in \mathbb{R}^d$, um classificador SVM linear procura um vetor de pesos w e um viés b tais que a fronteira de decisão seja dada por

$$f(x) = w^\top x + b = 0. \tag{2.11}$$

A Figura 5 ilustra esse conceito em um cenário bidimensional simplificado: exemplos de duas classes são representados como pontos em um plano, o hiperplano de separação (linha contínua) divide, idealmente, os exemplos da Classe +1 (círculos azuis) dos exemplos da Classe −1 (quadrados vermelhos), e as linhas tracejadas correspondem às margens. Os pontos circulados, que tocam essas margens, são os *vetores de suporte* e são eles que determinam a largura da margem.

Figura 5 – Exemplo bidimensional de uma máquina de vetores de suporte linear



Fonte: elaborado pelo autor.

Na formulação de *margem suave* (*soft-margin*), o problema de treinamento pode ser escrito como

$$\min_{w,b} \ \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \max\Big(0,\ 1 - y_i(w^\top x_i + b)\Big), \tag{2.12}$$

em que o primeiro termo incentiva hiperplanos com grande margem (norma de w pequena), e o segundo termo, baseado na *hinge loss*, penaliza exemplos classificados incorretamente

ou próximos demais da fronteira de decisão (BISHOP, 2006). O parâmetro $C > 0$ controla o compromisso entre maximizar a margem e permitir violações das restrições de separação.

Em tarefas de classificação de textos, as SVM lineares se tornaram particularmente populares porque lidam bem com vetores de alta dimensão e esparsos, como aqueles obtidos a partir de representações TF–IDF (JOACHIMS, 1998; SEBASTIANI, 2002). Nesses cenários, o número de atributos (termos do vocabulário) costuma ser muito maior do que o número de documentos, e o uso de um kernel não linear raramente é necessário. O modelo adotado neste trabalho utiliza o *kernel linear*, definido por

$$K(x, x') = x^\top x', \tag{2.13}$$

de modo que a função de decisão pode ser escrita como

$$f(x) = w^\top x + b = \sum_{i \in \mathcal{S}} \alpha_i y_i \, K(x_i, x) + b, \tag{2.14}$$

em que $\mathcal{S}$ denota o conjunto de vetores de suporte e $\alpha_i \geq 0$ são coeficientes aprendidos na etapa de treinamento (CORTES; VAPNIK, 1995).

Uma vantagem adicional das SVM lineares em relação a modelos mais complexos é a possibilidade de inspecionar diretamente o vetor de pesos w. Cada componente de w está associado a um termo do vocabulário, de modo que pesos de maior magnitude indicam atributos mais relevantes para a decisão do modelo. Termos com pesos positivos tendem a favorecer a atribuição de um documento a determinada classe, enquanto pesos negativos indicam evidências em sentido contrário. Essa propriedade torna as SVM lineares um ponto de partida interessante para técnicas de explicação baseadas em atributos, que serão retomadas em seções posteriores sobre explicação de modelos.

2.5 Processamento de Linguagem Natural e Modelos de Linguagem

O Processamento de Linguagem Natural (PLN) estuda métodos computacionais para analisar, representar e gerar texto em línguas naturais (JURAFSKY; MARTIN, 2023). Dentro desse campo, os modelos de linguagem têm papel central: eles procuram estimar a probabilidade de sequências de tokens, isto é, $P(w_1, w_2, \ldots, w_T)$, e são a base para tarefas como predição de próxima palavra, geração de texto e *fine-tuning* para classificação (JURAFSKY; MARTIN, 2023).

Modelos de linguagem modernos, como os baseados na arquitetura de transformadores, são treinados em grandes coleções de textos e aprendem representações densas e contextuais para cada token (VASWANI et al., 2017). Quando esses modelos possuem bilhões de parâmetros e são treinados em dados de larga escala, passam a ser chamados de grandes modelos de linguagem (*Large Language Models – LLMs*), que podem ser adaptados

para diversas tarefas, incluindo classificação de textos jurídicos, por meio de técnicas de *fine-tuning* ou de *prompting* (BROWN; MANN; RYDER, 2020).

Um componente fundamental desses modelos é a forma como o texto bruto é quebrado em unidades menores, chamadas *tokens*, que são os verdadeiros elementos manipulados pelo modelo. A próxima subseção descreve o processo de tokenização adotado em LLMs modernos.

2.5.1 Tokenização em grandes modelos de linguagem

Modelos de linguagem não operam diretamente sobre caracteres ou palavras, mas sobre sequências de *tokens*. Em línguas como o português, uma palavra pode ser composta por vários morfemas (prefixos, sufixos, flexões), e o vocabulário possível de palavras distintas é muito grande. Por isso, LLMs costumam usar esquemas de *subword tokenization*, em que palavras são quebradas em unidades menores, chamadas *subtokens* (SENNRICH; HADDOW; BIRCH, 2016). Nessa abordagem, o texto é segmentado em um conjunto finito de subpalavras que pode ser combinado para formar tanto palavras comuns quanto termos raros.

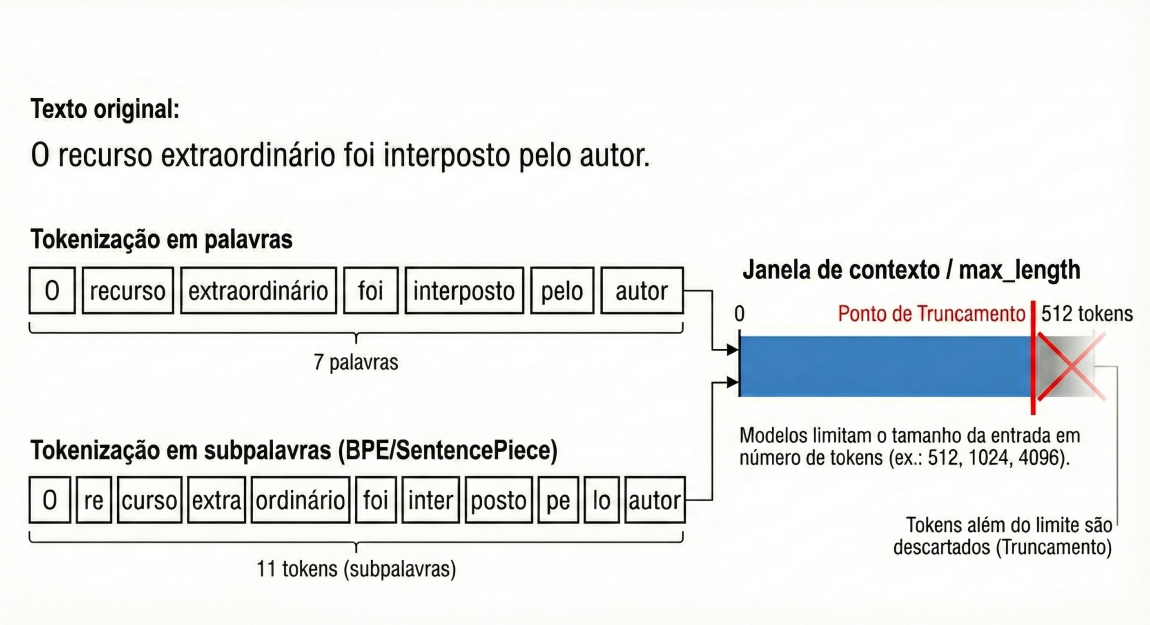
Algoritmos como *Byte Pair Encoding* (BPE) e SentencePiece são exemplos populares de tokenização em subpalavras. O BPE aprende iterativamente um vocabulário de subunidades a partir de pares de caracteres ou subpalavras mais frequentes, permitindo representar palavras raras como combinações de subtokens frequentes (SENNRICH; HADDOW; BIRCH, 2016). O SentencePiece, por sua vez, implementa BPE e modelos baseados em unigramas de forma independente de língua e pode treinar diretamente a partir de texto bruto, sem exigir tokenização prévia em palavras (KUDO; RICHARDSON, 2018). Na prática, isso significa que uma mesma palavra pode ser decomposta em um ou mais tokens, e que termos jurídicos menos comuns (por exemplo, expressões latinas) ainda podem ser representados por combinações de subpalavras presentes no vocabulário.

Uma frase relativamente curta em português pode ter poucas palavras, mas, ao ser decomposta em subtokens, o número total de tokens pode ser bem maior. A Figura 6 ilustra esse efeito em um exemplo jurídico simples. Na parte superior, o texto original (“O recurso extraordinário foi interposto pelo autor.”) é segmentado em sete palavras. Na parte inferior, a mesma frase é decomposta em subpalavras (BPE/SentencePiece), resultando em onze tokens. É essa contagem em tokens, e não em palavras, que de fato importa para o modelo.

Modelos de linguagem também possuem um comprimento máximo de sequência (*context window*), expresso em número de tokens. Na Figura 6, o painel à direita representa uma janela de contexto hipotética de 512 tokens, com um ponto de truncamento marcado em vermelho. Qualquer token além desse limite é descartado (truncamento), o que significa

que partes do documento podem ficar invisíveis para o modelo. Em textos jurídicos longos, como as peças presentes na base utilizada neste trabalho, essa diferença entre número de palavras e número de tokens, bem como a escolha do `max_length` e da estratégia de truncamento, tem impacto direto nas informações disponíveis para a decisão de classificação.

Figura 6 – Exemplo de tokenização em palavras e em subpalavras para uma frase jurídica, e relação com o limite de comprimento da janela de contexto.



Fonte: elaborado pelo autor.

2.5.2 Modelos de linguagem

De forma geral, um modelo de linguagem procura atribuir uma probabilidade a sequências de tokens $(t_1, t_2, \ldots, t_T)$. Um modo clássico de formular esse problema é decompor a probabilidade conjunta como um produto de probabilidades condicionais:

$$P_\theta(t_1, t_2, \ldots, t_T) = \prod_{i=1}^{T} P_\theta(t_i \mid t_1, \ldots, t_{i-1}), \tag{2.15}$$

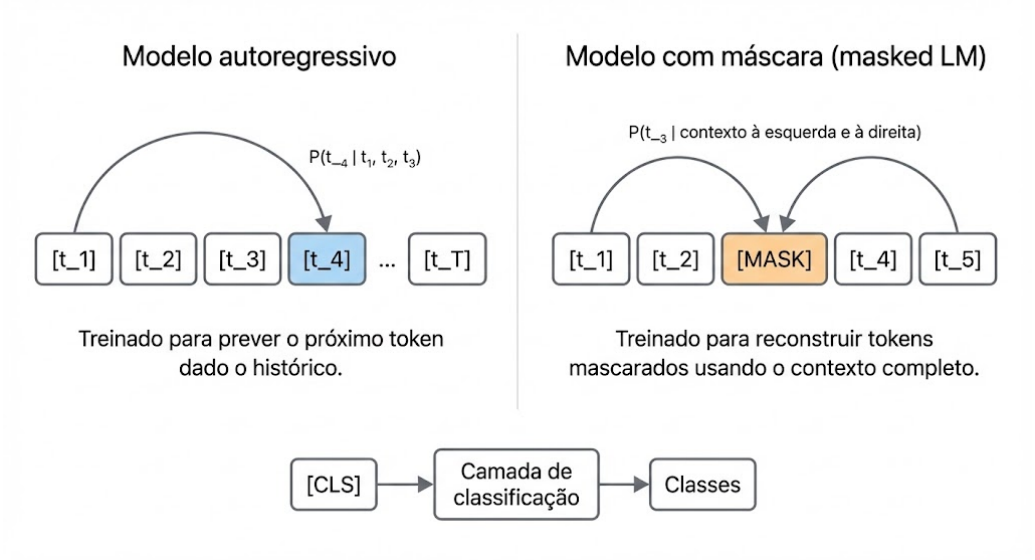
em que $P_\theta(t_i \mid t_1, \ldots, t_{i-1})$ representa a probabilidade do próximo token dado o histórico anterior, parametrizada por um modelo com parâmetros θ (JURAFSKY; MARTIN, 2023; GOLDBERG, 2017). Modelos treinados com esse objetivo são chamados de *autoregressivos* e aprendem a prever o próximo token em uma sequência.

Outro tipo importante de objetivo é o de *masked language modeling*, popularizado por modelos como o BERT (DEVLIN et al., 2019). Nesse caso, alguns tokens da sequência são substituídos por um símbolo especial (`[MASK]`), e o modelo é treinado para prever os tokens originais com base no restante do contexto. Em vez de prever sempre o próximo

token, o modelo aprende a reconstruir partes ausentes da frase, explorando informação de ambos os lados do token mascarado.

A Figura 7 resume esses dois padrões de treinamento. No painel esquerdo, o modelo autoregressivo recebe uma sequência de tokens $[t_1, t_2, t_3, \ldots, t_T]$ e é treinado para atribuir alta probabilidade ao próximo token, como indicado pela seta que liga o histórico (t_1, t_2, t_3) ao token t_4 destacado em azul ($P(t_4 \mid t_1, t_2, t_3)$). No painel direito, o modelo com máscara recebe uma sequência em que um dos tokens foi substituído por `[MASK]` e aprende a reconstruir o token oculto a partir do contexto à esquerda e à direita, como indicado pelas setas que convergem para o token mascarado.

Figura 7 – Esquema ilustrando os objetivos de treinamento autoregressivo e com máscara, e o uso do token `[CLS]` para classificação.



Fonte: elaborado pelo autor.

Tanto em modelos autoregressivos quanto em modelos com máscara, o treinamento é feito minimizando uma função de perda baseada na entropia cruzada entre as probabilidades previstas e os tokens observados. Após o treinamento, modelos autoregressivos podem ser usados para geração de texto: dado um *prompt* inicial, o próximo token é escolhido com base na distribuição $P_\theta(t_i \mid t_1, \ldots, t_{i-1})$ (por exemplo, via amostragem ou escolha do token mais provável), e o processo se repete até atingir um comprimento máximo ou um token de fim de sequência (BROWN; MANN; RYDER, 2020).

Modelos do tipo BERT, treinados com máscara, costumam incluir um token especial de início de sequência (como `[CLS]`), cuja representação contextual é usada como resumo do texto e alimentada a uma camada linear de classificação, como ilustrado na parte inferior da Figura 7 (DEVLIN et al., 2019). Em modelos autoregressivos, é possível adicionar uma *classification head* que recebe a representação de um token específico (por exemplo, o último token) ou usar o próprio modelo para produzir, em linguagem natural, a resposta desejada

a partir de um *prompt* adequado. Neste trabalho, exploramos essas duas capacidades: modelos de linguagem são utilizados tanto como classificadores de textos jurídicos quanto como geradores de explicações em linguagem natural para as decisões de classificação.

2.6 Arquitetura Transformer e grandes modelos de linguagem

Grande parte dos modelos de linguagem modernos, incluindo os LLMs utilizados neste trabalho, baseia-se na arquitetura Transformer proposta por Vaswani et al. (2017). Diferentemente de redes recorrentes, os Transformers utilizam camadas de *autoatenção* para processar todos os tokens de uma sequência em paralelo, permitindo que cada posição "enxergue" diretamente o restante do contexto. Essa ideia, combinada com camadas feed-forward e mecanismos de normalização, deu origem a modelos profundos capazes de capturar dependências de longo alcance em texto, tornando-se a base de LLMs como BERT, GPT e seus sucessores (DEVLIN et al., 2019; BROWN; MANN; RYDER, 2020; JURAFSKY; MARTIN, 2023).

A seguir, apresenta-se de forma resumida o mecanismo de atenção e autoatenção, que constitui o núcleo da arquitetura Transformer.

2.6.1 Mecanismo de atenção e autoatenção

A ideia central do mecanismo de atenção é permitir que uma representação vetorial *consulta* (*query*) foque em outras representações relevantes (*keys*) e combine suas informações (*values*). Em um Transformer, cada token de entrada é projetado em três vetores: consulta q_i, chave k_i e valor v_i. Agrupando essas projeções em matrizes Q, K e V, a atenção *scaled dot-product* é definida por

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V, \tag{2.16}$$

em que d_k é a dimensão das chaves e a função softmax produz uma distribuição de pesos sobre as posições da sequência (VASWANI et al., 2017; JURAFSKY; MARTIN, 2023).

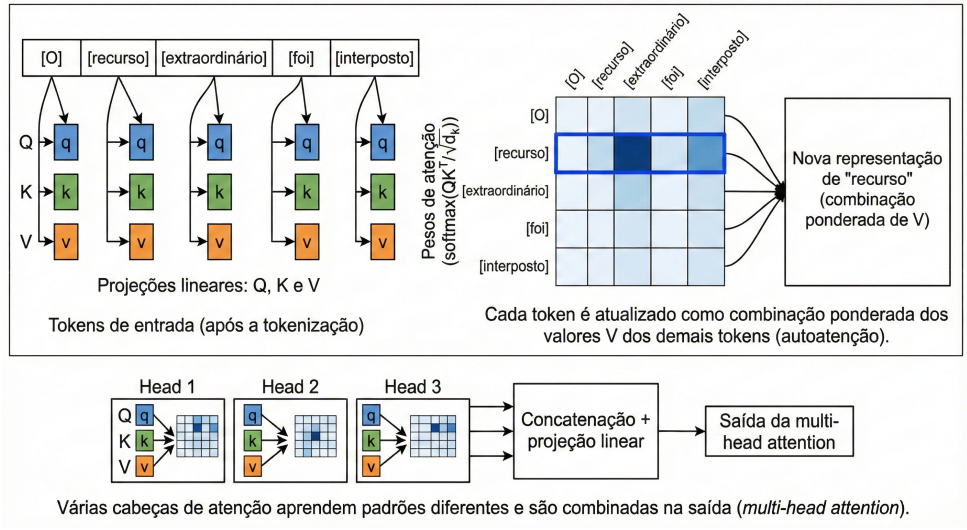
Quando consultas, chaves e valores são obtidos da mesma sequência de entrada, fala-se em *autoatenção*. Cada token passa a "olhar" para os demais tokens do texto e gera uma nova representação como combinação ponderada dos valores V, levando em conta o contexto inteiro. Em uma frase jurídica, por exemplo, o token associado a "recurso" pode dar mais peso a termos como "extraordinário" ou "especial", que ajudam a precisar o seu sentido no contexto.

A Figura 8 ilustra esse processo de forma esquemática. Na parte superior, uma frase jurídica curta é decomposta em tokens, e cada um deles gera vetores Q, K e V por projeções lineares. A matriz central representa os pesos de atenção $\text{softmax}(QK^\top/\sqrt{d_k})$;

a linha destacada para o token “recurso” mostra que ele atribui maior peso a tokens semanticamente relacionados, como “extraordinário”. À direita, esses pesos são usados para combinar os valores V e produzir uma nova representação de “recurso”, agora enriquecida pelas informações dos demais tokens.

Na parte inferior da Figura 8, apresenta-se a ideia de *multi-head attention*. Várias cabeças de atenção processam a mesma sequência em subespaços diferentes, cada uma com suas próprias projeções Q, K e V e sua própria matriz de atenção. As saídas dessas cabeças são concatenadas e projetadas de volta para o espaço original, permitindo que o modelo capture, ao mesmo tempo, diferentes tipos de relações entre os tokens (VASWANI et al., 2017).

Figura 8 – Esquema simplificado do mecanismo de autoatenção em uma camada Transformer.



Fonte: elaborado pelo autor om base em Vaswani et al. (2017).

2.6.2 Estrutura básica de Transformers *decoders*

Modelos de linguagem autoregressivos modernos, como os do tipo GPT, são baseados em pilhas de camadas *decoder* da arquitetura Transformer (VASWANI et al., 2017; BROWN; MANN; RYDER, 2020). Em alto nível, cada camada decoder recebe as representações dos tokens de entrada e as transforma por meio de três componentes principais: uma camada de autoatenção multi-head mascarada, uma rede totalmente conectada (*feed-forward* ou MLP) e conexões residuais com normalização.

Sejam $h^{(l)}$ as representações na camada l. Primeiro aplica-se a autoatenção multi-head, seguida de conexão residual e *LayerNorm*:

$$\tilde{h}^{(l)} = \mathrm{LayerNorm}\Big(h^{(l)} + \mathrm{MHA}(h^{(l)})\Big). \tag{2.17}$$

Em seguida, aplica-se uma MLP posicional (mesma MLP em cada posição), também com conexão residual e normalização:

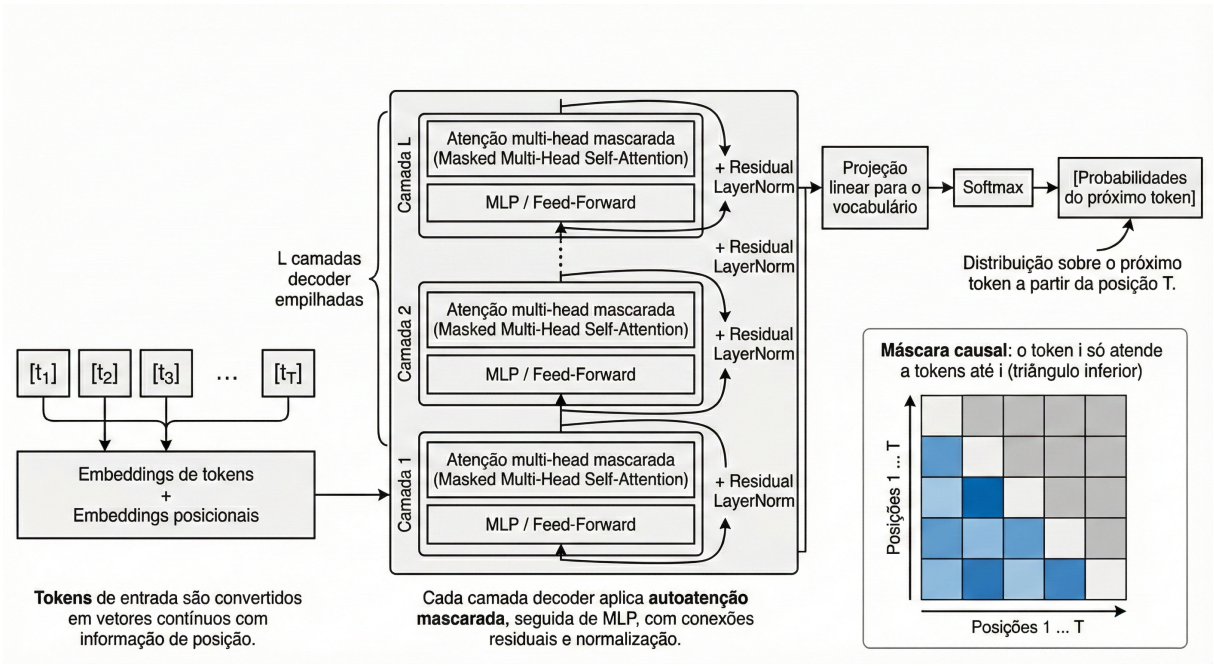
$$h^{(l+1)} = \text{LayerNorm}\Big(\tilde{h}^{(l)} + \text{MLP}(\tilde{h}^{(l)})\Big). \tag{2.18}$$

Empilhando várias camadas desse tipo, o modelo refina as representações dos tokens, combinando informações de todo o contexto via autoatenção e aplicações não lineares da MLP (VASWANI et al., 2017; JURAFSKY; MARTIN, 2023).

Para que o modelo seja autoregressivo, a matriz de atenção recebe uma *máscara causal*. Essa máscara zera os pesos correspondentes a posições futuras ($j > i$) antes da aplicação do softmax, de modo que cada token na posição i só possa atender a si mesmo e aos tokens anteriores. Assim, durante o treinamento e a geração, o modelo respeita o objetivo de prever o próximo token apenas com base no histórico.

A Figura 9 resume esse funcionamento. À esquerda, os tokens de entrada são convertidos em embeddings de token e de posição. No centro, esses vetores atravessam uma pilha com L camadas decoder: em cada camada, aplica-se autoatenção multi-head mascarada, seguida de MLP, com conexões residuais e normalização. À direita, a representação resultante (por exemplo, na posição T) passa por uma projeção linear para o vocabulário e por uma função *softmax*, gerando uma distribuição de probabilidade sobre o próximo token. O painel inferior mostra a máscara causal na matriz de atenção: o triângulo inferior representa as posições permitidas (tokens até i), enquanto o triângulo superior, correspondente a posições futuras, é bloqueado.

Figura 9 – Estrutura simplificada de um Transformer do tipo decoder para modelos de linguagem autoregressivos.



Fonte: elaborado pelo autor om base em Vaswani et al. (2017).

2.6.3 Transformers para classificação de sequência

Embora Transformers tenham sido propostos originalmente como modelos de tradução e linguagem, eles podem ser adaptados de forma direta para tarefas de classificação de sequência. A ideia básica é resumir toda a sequência de entrada em um vetor de representação e, em seguida, aplicar uma camada linear para produzir os escores de cada classe.

Em modelos do tipo BERT, isso costuma ser feito introduzindo um token especial `[CLS]` no início da sequência. Após o processamento pelas camadas Transformer, o vetor oculto associado a esse token, $h_{\text{CLS}} \in \mathbb{R}^d$, funciona como uma representação compacta da sequência inteira. Um classificador linear é então aplicado:

$$z = W h_{\text{CLS}} + b, \tag{2.19}$$

em que $W \in \mathbb{R}^{K \times d}$ e $b \in \mathbb{R}^K$ são os parâmetros da *classification head* e K é o número de classes. Os escores z são transformados em probabilidades por meio de uma função *softmax*, e o modelo é treinado com entropia cruzada sobre os rótulos de classe (DEVLIN et al., 2019; JURAFSKY; MARTIN, 2023).

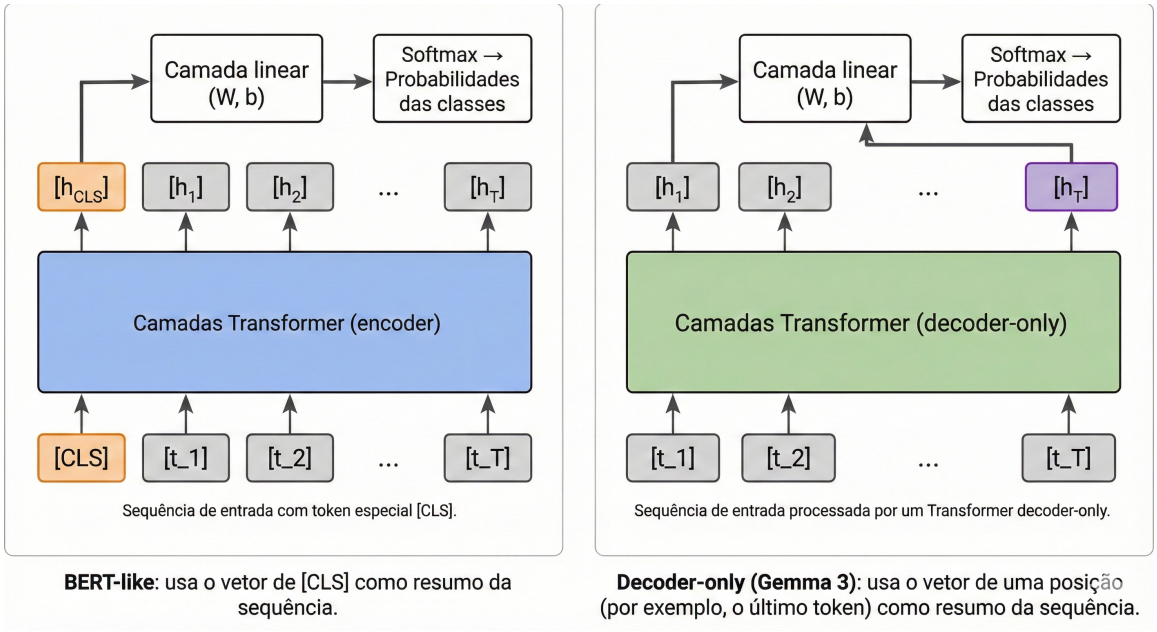
Em modelos *decoder-only*, como grandes modelos de linguagem autoregressivos, a ideia é semelhante, mas a representação da sequência pode ser obtida, por exemplo, a partir do vetor oculto de um token especial no início da entrada ou do último token processado

(BROWN; MANN; RYDER, 2020). Seja h_{seq} essa representação sequencial; aplica-se novamente uma camada linear $z = Wh_{\text{seq}} + b$ e o modelo é ajustado para minimizar a entropia cruzada entre as probabilidades previstas e as classes verdadeiras, em linha com a formulação da Seção 2.2.

Na prática, essa mesma estratégia pode ser usada com diferentes arquiteturas Transformer. Modelos encoder, como o BERT, utilizam o vetor do token `[CLS]`; modelos *decoder-only* recentes, como os da família Gemma 3 (por exemplo, Gemma 3 270M-IT e Gemma 3 4B-IT), podem adotar o vetor de uma posição específica (tipicamente o último token) como resumo da sequência, sobre o qual se adiciona uma *classification head* linear para adaptação a tarefas de classificação de texto (Gemma Team, 2025).

A Figura 10 ilustra essas duas variantes. No painel (A), de estilo BERT, a sequência de entrada recebe um token `[CLS]`, e seu vetor oculto h_{CLS} alimenta uma camada linear seguida de *softmax*, produzindo as probabilidades das classes. No painel (B), de estilo *decoder-only*, um Transformer autoregressivo gera vetores ocultos $h_1, h_2, \ldots, h_T$ para cada posição, e a representação de uma posição escolhida (como o último token h_T) é usada da mesma forma para obter os escores de classe.

Figura 10 – Adaptação de modelos Transformer para classificação de sequência.



Fonte: elaborado pelo autor com base em Devlin et al. (2019), Brown, Mann e Ryder (2020)

2.6.4 Modelos de conversa e geração condicional

Modelos de linguagem podem ser usados de duas formas principais em tarefas de processamento de texto. Na forma *classificadora*, o objetivo é receber uma sequência de entrada x e produzir uma distribuição de probabilidade sobre um conjunto finito de classes

$y \in \{1, \ldots, K\}$, tipicamente por meio de uma *classification head* linear seguida de *softmax*, como discutido na Seção 2.6.3. Já na forma *geradora*, o modelo estima uma distribuição sobre sequências de tokens condicionada a um contexto:

$$p_\theta(y_1, \ldots, y_T \mid x) = \prod_{t=1}^{T} p_\theta(y_t \mid x, y_{<t}), \tag{2.20}$$

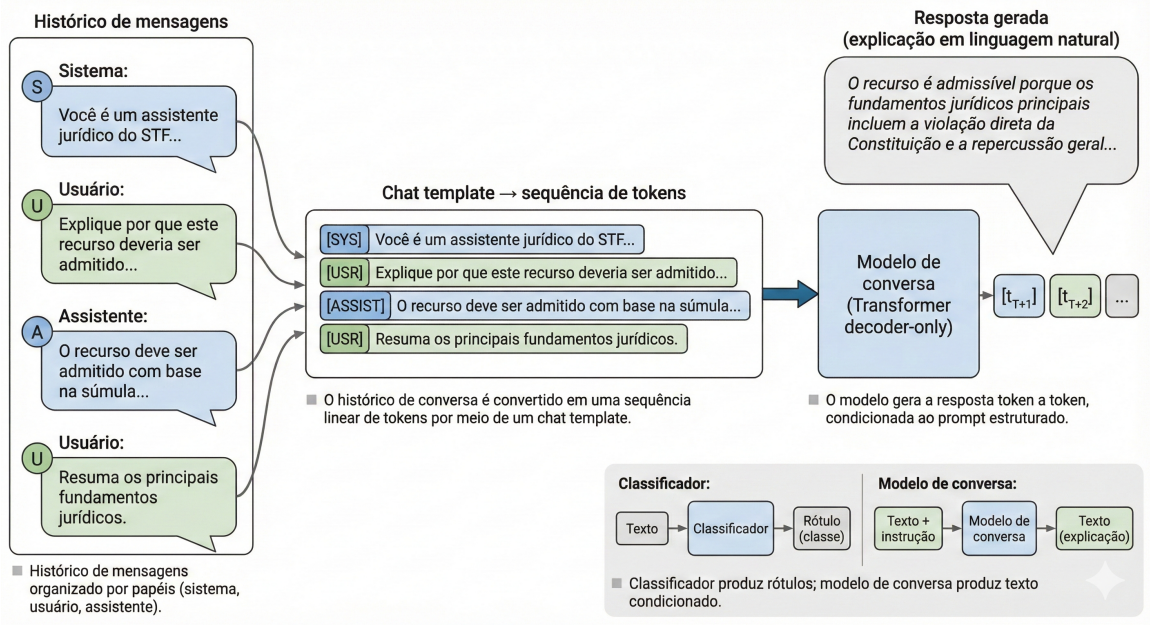
o que permite gerar textos completos token a token a partir de um prompt inicial (BROWN; MANN; RYDER, 2020; JURAFSKY; MARTIN, 2023).

Modelos de conversa (*chat models*) são modelos geradores ajustados para seguir instruções e interagir em diálogo, geralmente por meio de afinamento com feedback humano (*instruction tuning*, RLHF) (OUYANG et al., 2022). Em vez de receber apenas uma frase isolada, o modelo passa a condicionar suas predições ao histórico da conversa, organizado em mensagens com papéis como *sistema*, *usuário* e *assistente*. A mensagem de sistema define o comportamento esperado (do tipo "aja como um auxiliar jurídico"), enquanto as mensagens de usuário e assistente representam, respectivamente, perguntas e respostas ao longo dos turnos do diálogo.

Na prática, esse histórico de mensagens é convertido em uma sequência linear de tokens por meio de um *chat template*, que intercala marcadores especiais para cada papel (por exemplo, `[SYS]`, `[USR]`, `[ASSIST]`) e define explicitamente o ponto a partir do qual o modelo deve continuar gerando texto. Modelos de conversa do tipo Gemma 3 4B-IT Chat seguem essa abordagem: o prompt é formado por uma combinação de instruções de sistema e conteúdo do usuário, e a saída do modelo corresponde à geração condicional de uma resposta coerente com esse contexto (Gemma Team, 2025).

De forma esquemática, a Figura 11 ilustra esse processo. À esquerda, o histórico de mensagens é exibido com seus papéis: uma instrução de sistema define o comportamento do modelo, mensagens de usuário formulam perguntas sobre um recurso e uma resposta anterior do assistente aparece como parte do contexto. No painel central, esse histórico é linearizado em uma sequência `[SYS]`, `[USR]` e `[ASSIST]` que codifica explicitamente quem disse o quê e em qual ordem, funcionando como o *chat template*. À direita, um modelo de conversa baseado em Transformer *decoder-only* recebe essa sequência como entrada e gera, token a token, uma resposta em linguagem natural, aqui interpretada como uma explicação jurídica condicionada ao prompt. O painel inferior compara esse comportamento com o de um classificador tradicional: enquanto o classificador recebe texto e devolve um rótulo de classe, o modelo de conversa recebe texto mais instrução e devolve uma sequência textual explicativa.

Figura 11 – Esquema simplificado de uso de um modelo de conversa para geração condicional.



Fonte: elaborado pelo autor com base em Brown, Mann e Ryder (2020), Ouyang et al. (2022)

2.7 Fine-tuning de Modelos de Linguagem

Modelos de linguagem de grande porte são, em geral, pré-treinados em grandes coleções de textos de domínio amplo, aprendendo padrões estatísticos de linguagem de forma genérica (HOWARD; RUDER, 2018; DEVLIN et al., 2019; BROWN; MANN; RYDER, 2020). Para aplicar esses modelos a tarefas específicas — como classificação de documentos, resumo ou resposta a perguntas — normalmente é necessário um processo de adaptação, conhecido como *fine-tuning*. Nesse processo, o modelo pré-treinado passa a ser treinado em um conjunto menor de exemplos rotulados da tarefa-alvo, ajustando seus parâmetros para produzir saídas coerentes com os rótulos desejados.

2.7.1 Fine-tuning eficiente em parâmetros (PEFT)

Antes de discutir abordagens eficientes, vale lembrar a ideia do *fine-tuning* tradicional. No *full fine-tuning*, parte-se de um modelo já pré-treinado com parâmetros iniciais θ_0 e atualizam-se *todos* esses parâmetros com base em um conjunto de treinamento rotulado $\{(x_i, y_i)\}_{i=1}^{N}$ (DEVLIN et al., 2019; HOWARD; RUDER, 2018). O objetivo pode ser escrito como:

$$L_{\text{tarefa}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell\Big(f_\theta(x_i), y_i\Big), \tag{2.21}$$

em que f_θ representa o modelo de linguagem acrescido da *head* da tarefa (por exemplo, uma camada linear para classificação) e ℓ é a perda adequada, como a entropia cruzada.

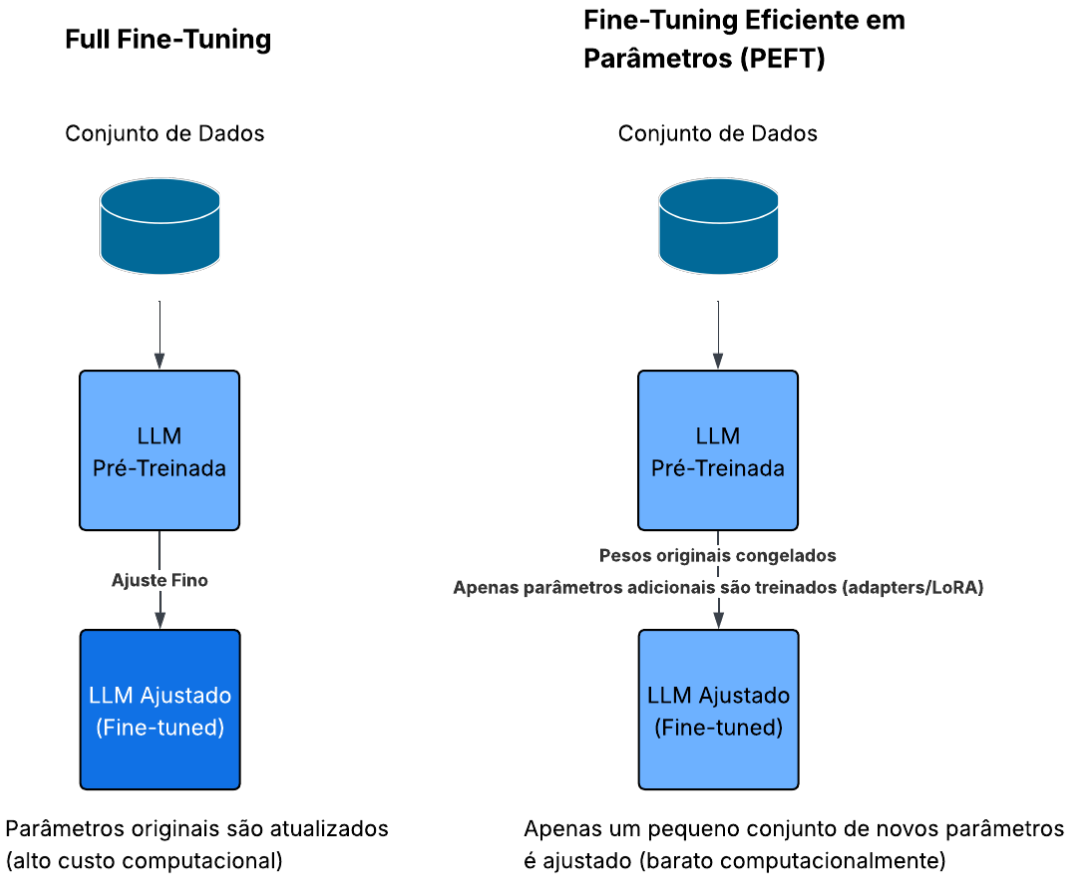
Durante o treinamento, todos os parâmetros θ são atualizados por métodos como SGD ou Adam.

Essa estratégia oferece grande flexibilidade, pois o modelo pode se adaptar de forma fina às características do novo domínio (por exemplo, texto jurídico). Por outro lado, o custo de memória e computação cresce com o tamanho do modelo: além dos pesos, é preciso armazenar gradientes e estados do otimizador, o que pode multiplicar por três ou mais o uso de memória em relação à inferência. Em modelos com centenas de milhões ou bilhões de parâmetros, isso torna o *full fine-tuning* pouco prático em GPUs limitadas.

Com esse contexto, surgem as técnicas de *fine-tuning eficiente em parâmetros* (*parameter-efficient fine-tuning*, PEFT). A idéia central é adaptar o modelo a uma nova tarefa atualizando apenas uma pequena fração de parâmetros, mantendo o restante do modelo pré-treinado congelado (HOULSBY et al., 2019; HU et al., 2022). Em métodos baseados em *adapters*, pequenas camadas adicionais são inseridas dentro das camadas Transformer, e somente esses blocos extras são treinados (HOULSBY et al., 2019). Em abordagens como o LoRA (*Low-Rank Adaptation*), algumas matrizes de projeção do modelo são decompostas em uma parte congelada e uma correção de baixa dimensão; apenas essa correção de baixo posto é atualizada durante o treinamento (HU et al., 2022).

De forma esquemática, a Figura 12 compara o *full fine-tuning* com uma abordagem de PEFT. No lado esquerdo, o conjunto de dados rotulados é usado para ajustar todos os pesos da LLM pré-treinada, produzindo um novo modelo ajustado. No lado direito, o mesmo conjunto de dados alimenta a LLM pré-treinada, mas seus pesos originais permanecem congelados: apenas parâmetros adicionais são aprendidos (adapters ou camadas de correção), que são "acoplados" ao modelo base para especializá-lo na nova tarefa.

Figura 12 – Comparação conceitual entre *full fine-tuning* e fine-tuning eficiente em parâmetros (PEFT).



Fonte: elaborado pelo autor.

Na prática, o número de parâmetros atualizáveis passa a ser uma fração pequena do total, reduzindo significativamente o consumo de memória e o custo de treinamento, sem modificar a arquitetura principal do modelo. Esse tipo de abordagem permite adaptar modelos grandes mesmo em ambientes com recursos computacionais mais restritos e serve de base para variações ainda mais econômicas, como técnicas de *fine-tuning* com pesos quantizados, que podem explorar simultaneamente PEFT e quantização dos parâmetros.

2.7.2 Low-Rank Adaptation (LoRA) e QLoRA

Dentro do contexto de PEFT, um dos métodos mais utilizados para adaptar modelos de linguagem grandes é o LoRA (*Low-Rank Adaptation*) (HU et al., 2022). A idéia básica é evitar atualizar diretamente todas as entradas de uma matriz de pesos $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. Em vez disso, essa matriz é mantida congelada e introduz-se uma correção de baixo posto

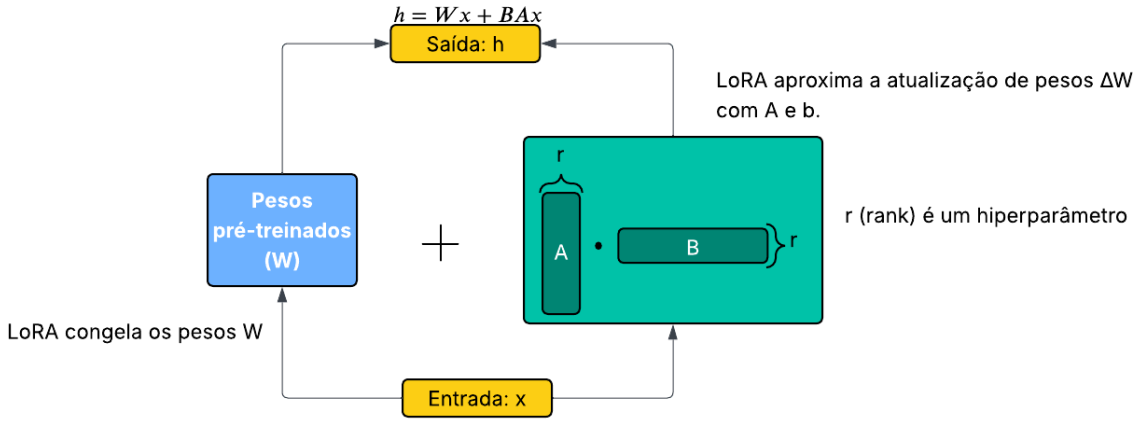
ΔW, parametrizada como o produto de duas matrizes de baixa dimensionalidade:

$$W' = W + \Delta W = W + BA, \tag{2.22}$$

em que $A \in \mathbb{R}^{r \times d_{\text{in}}}$ e $B \in \mathbb{R}^{d_{\text{out}} \times r}$, com $r \ll \min(d_{\text{out}}, d_{\text{in}})$. Durante o *fine-tuning*, apenas A e B são treinados, enquanto W permanece fixo. Na prática, isso reduz o número de parâmetros atualizáveis em ordens de grandeza, mantendo praticamente inalterado o custo de inferência e permitindo salvar e carregar apenas os *adapters* LoRA em vez do modelo completo.

De forma esquemática, a Figura 13 ilustra esse mecanismo em uma camada linear. A entrada x é aplicada tanto aos pesos pré-treinados W quanto aos adaptadores de baixo posto A e B. A contribuição de LoRA corresponde ao termo BAx, que é somado à saída original Wx para produzir o vetor final h. Assim, o modelo passa a se comportar como se utilizasse $W' = W + BA$, sem que os pesos pré-treinados precisem ser modificados.

Figura 13 – Esquema conceitual da adaptação de baixo posto (LoRA) sobre uma matriz de pesos congelada.



Fonte: elaborado pelo autor.

O LoRA costuma ser aplicado em matrizes de projeção das camadas de atenção ou MLP dentro do Transformer, seguindo o mesmo princípio: as matrizes originais do modelo pré-treinado são congeladas, e termos adicionais de baixo posto são acoplados para capturar a adaptação à nova tarefa (HU et al., 2022). Dessa forma, o modelo base funciona como um "backbone" fixo, enquanto a tarefa específica é representada pelos parâmetros extras de baixa dimensão.

Uma extensão importante dessa ideia é a QLoRA, que combina LoRA com quantização de baixa precisão para reduzir ainda mais o consumo de memória em modelos grandes (DETTMERS et al., 2023). Nessa abordagem, os pesos do modelo base são quantizados, tipicamente para formatos de 4 bits como o NF4 (*Normalized Float 4*), usando bibliotecas

especializadas (como `bitsandbytes`). O modelo quantizado é utilizado nas passagens para frente, enquanto os adapters LoRA são mantidos em maior precisão (por exemplo, 16 bits) e atualizados durante o treinamento. Assim, a maior parte da memória é ocupada por pesos quantizados, e apenas uma pequena fração de parâmetros em alta precisão precisa ser armazenada e atualizada.

De forma conceitual, a QLoRA pode ser vista como a composição de duas ideias: (i) quantizar o modelo base para reduzir o custo de armazenamento e cálculo; e (ii) utilizar LoRA para adaptar o modelo por meio de correções de baixo posto treinadas em alta precisão (DETTMERS et al., 2023). Essa combinação permite realizar *fine-tuning* de modelos com vários bilhões de parâmetros em GPUs com VRAM limitada, com degradação relativamente pequena de desempenho em comparação a abordagens de *fine-tuning* mais pesadas. Por outro lado, a estabilidade do treinamento e a qualidade final dependem da escolha adequada do formato de quantização, da precisão dos adapters e de hiperparâmetros como o posto r, o que exige algum cuidado na configuração do procedimento.

2.8 Explicabilidade em modelos de classificação de texto

Em aplicações sensíveis, como saúde ou Direito, não basta que um modelo de aprendizado de máquina apresente boas métricas de desempenho: muitas vezes, é necessário entender por que uma certa decisão foi tomada (MOLNAR, 2022). Essa demanda levou ao desenvolvimento de métodos de *interpretabilidade* e *explicabilidade*, que buscam tornar modelos mais transparentes ou fornecer explicações sobre seus resultados, especialmente em decisões de alto impacto (DOSHI-VELEZ; KIM, 2017; RUDIN, 2019).

No contexto de classificação de textos, como documentos jurídicos, a discussão sobre explicabilidade envolve tanto o tipo de modelo utilizado (por exemplo, modelos lineares ou redes neurais profundas) quanto a forma de apresentar ao usuário (juízes, servidores, advogados) os motivos da classificação atribuída. Nesta seção, são introduzidos conceitos básicos de interpretabilidade e diferentes tipos de explicações, que servirão de base para a abordagem proposta nos capítulos seguintes.

2.8.1 Conceitos básicos de interpretabilidade e explicabilidade

De forma geral, fala-se em modelo *caixa-preta* quando a relação entre entradas e saídas não é facilmente compreensível por um humano, ainda que o modelo possa ser avaliado numericamente (DOSHI-VELEZ; KIM, 2017). Grandes redes neurais e modelos de linguagem são frequentemente vistos dessa forma: sabe-se que aprendem padrões complexos, mas não é trivial identificar quais aspectos do texto levaram a uma decisão específica. Em contraste, fala-se em modelo *explicável* ou *interpretável* quando a estrutura do modelo ou

as informações derivadas dele permitem entender, ao menos qualitativamente, como as decisões estão sendo tomadas (MOLNAR, 2022; RUDIN, 2019).

As explicações podem ser vistas em dois níveis principais. No nível *global*, busca-se descrever o comportamento do modelo como um todo, por exemplo, identificando quais variáveis (ou termos do vocabulário) tendem a ser mais importantes para cada classe ou como certas faixas de valores afetam a decisão (MOLNAR, 2022). Em classificação de textos, isso pode incluir a análise de pesos de um classificador linear ou de padrões recorrentes em documentos de determinada categoria.

Já no nível *local*, o foco está em explicar a decisão do modelo para um exemplo específico. Nesse caso, uma explicação pode indicar quais palavras ou trechos de um documento jurídico mais contribuíram para a classificação em certa classe, ou como pequenas modificações no texto poderiam alterar o rótulo previsto. Esse tipo de explicação local é particularmente relevante em cenários em que cada decisão individual precisa ser justificada, como na triagem de recursos em tribunais.

Na prática, tanto explicações globais quanto locais podem ser construídas a partir de diferentes famílias de métodos, que vão desde modelos intrinsecamente mais simples até técnicas pós-hoc aplicadas sobre modelos complexos. Nos próximos tópicos, são apresentados alguns desses métodos, com foco em sua aplicação em classificação de textos.

2.8.2 Modelos lineares como modelos intrinsecamente explicáveis

Modelos lineares, como regressões logísticas e SVM lineares, são frequentemente citados como exemplos de modelos intrinsecamente explicáveis (RUDIN, 2019; MOLNAR, 2022). Em lugar de depender de um procedimento pós-hoc para interpretar uma decisão, a própria estrutura do modelo fornece informações sobre a importância relativa de cada atributo.

No caso de um classificador linear para uma classe c, a pontuação atribuída a um documento representado por um vetor $x \in \mathbb{R}^d$ pode ser escrita como:

$$f_c(x) = w_c^\top x + b_c = \sum_{j=1}^{d} w_{c,j}\, x_j + b_c, \tag{2.23}$$

em que $w_c \in \mathbb{R}^d$ é o vetor de pesos associado à classe c, b_c é o viés e cada componente x_j corresponde a uma característica do documento, como um termo ou *n-grama*. Quando se utiliza uma representação TF–IDF, como discutido na Seção 2.3.2, os valores x_j são não negativos e indicam a relevância de cada termo para o documento.

Nessa perspectiva, o sinal e a magnitude de $w_{c,j}$ têm interpretações diretas. Um peso $w_{c,j} > 0$ indica que a presenção (ou maior peso TF–IDF) do termo j contribui para aumentar a pontuação da classe c, enquanto um peso $w_{c,j} < 0$ indica uma contribuição contrária. Pesos de maior magnitude, em valor absoluto, correspondem a atributos mais

influentes para a decisão do modelo (SEBASTIANI, 2002). Em classificação de textos, isso permite identificar, por exemplo, quais palavras ou *n-gramas* são mais característicos de cada classe.

A partir dessa formulação, é possível construir explicações tanto globais quanto locais. Em nível global, a análise do vetor de pesos w_c permite listar os termos mais relevantes para cada classe, fornecendo uma visão geral do comportamento do modelo. Em nível local, para um documento específico, as contribuições individuais $w_{c,j}x_j$ indicam o quanto cada termo presente no texto influenciou a pontuação de $f_c(x)$. Isso torna os modelos lineares uma base natural para explicações baseadas em atributos em tarefas de classificação de documentos.

2.8.3 Uso de LLMs para explicações em linguagem natural

Com o avanço dos grandes modelos de linguagem (LLMs), surgiu a possibilidade de utilizá-los não apenas como classificadores, mas também como ferramentas de explicação em linguagem natural para decisões tomadas por outros modelos. Em vez de o próprio LLM decidir a classe final, ele pode ser acionado para gerar um texto que esclareça por que uma certa decisão foi tomada (BILAL; EBERT; LIN, 2025; CAMBRIA et al., 2024).

De forma conceitual, esse uso de LLMs para explicabilidade segue um fluxo em duas etapas. Primeiro, um modelo de classificação (por exemplo, um modelo linear com TF–IDF ou um Transformer ajustado para *sequence classification*) recebe o documento e produz um rótulo previsto e, se disponível, alguma medida de importância de atributos, como os termos ou *n-gramas* com maior contribuição para a decisão (cf. Seção 2.8.2). Em seguida, essas informações são usadas para construir um *prompt* de entrada para o LLM, que pode incluir:

- um trecho representativo do documento (por exemplo, parágrafos mais relevantes);
- o rótulo predito pelo modelo de classificação;
- uma lista dos termos ou *n-gramas* mais importantes para essa decisão.

A partir desse *prompt*, o LLM é instruído a gerar uma explicação em linguagem natural que descreva, em termos acessíveis ao usuário final, por que o documento foi associado àquela classe. Em contextos jurídicos, por exemplo, o modelo pode ser orientado a produzir uma justificativa em estilo próximo ao da linguagem jurídica, mencionando os trechos do texto que indicam determinada categoria de recurso ou tema.

Abordagens desse tipo se enquadram em uma linha recente de pesquisa que explora LLMs como "camada de explicação" para outros modelos, traduzindo saídas numéricas ou listas de atributos em narrativas compreensíveis (BILAL; EBERT; LIN, 2025; CAMBRIA

et al., 2024). Embora ofereçam um meio flexível de gerar explicações mais ricas, essas técnicas também trazem desafios, como o risco de alucinações e a necessidade de garantir que a explicação se mantenha fiel à decisão original do modelo de classificação, aspectos discutidos na literatura recente sobre LLMs para XAI.

2.8.4 Métrica de fidelidade das explicações via *feature coverage*

Ao avaliar explicações geradas por modelos de aprendizado de máquina, não basta verificar apenas se o texto produzido é gramatical ou "convincente". Em muitos trabalhos de interpretabilidade, diferencia-se entre explicações que são apenas plausíveis para um humano e explicações que são, de fato, fiéis ao comportamento do modelo subjacente (DOSHI-VELEZ; KIM, 2017). Nesse contexto, métricas de *fidelidade* procuram medir o quanto a explicação está alinhada com as características que realmente influenciaram a decisão do modelo.

Uma forma simples de aproximar essa ideia são métricas de *cobertura* de características, que verificam quantas das pistas relevantes para o modelo aparecem, de fato, na explicação em linguagem natural. Abordagens desse tipo já foram usadas em sistemas de recomendação explicável, em que se mede a proporção de atributos importantes (por exemplo, características de produtos) efetivamente mencionados no texto gerado como explicação (LI; ZHANG; CHEN, 2020).

No caso de classificadores lineares de texto, é comum dispor de um conjunto de atributos (por exemplo, *n-grams*) com relevância estimada para a decisão do modelo, a partir de pesos ou escores de contribuição. Seja F_d o conjunto formado pelos m *n-grams* mais importantes para a classificação de um documento d, e seja E_d o texto da explicação gerada para esse mesmo documento. A métrica de cobertura de características (*feature coverage*) para esse exemplo pode ser definida como:

$$\mathrm{FC}(d) = \frac{\left|\{\, g \in F_d \;:\; g \text{ aparece em } E_d \,\}\right|}{|F_d|}, \tag{2.24}$$

isto é, a proporção de *n-grams* considerados relevantes pelo modelo que também são explicitamente mencionados na explicação. Valores de $\mathrm{FC}(d)$ próximos de 1 indicam que a explicação faz referência à maior parte das pistas usadas pelo modelo; valores próximos de 0 sugerem que o texto gerado está desconectado das características que realmente motivaram a predição.

A partir dessa definição por documento, é possível calcular estatísticas resumidas (como média e desvio-padrão) em um conjunto de exemplos, obtendo uma medida global de fidelidade das explicações. Ainda assim, trata-se de uma métrica relativamente simples e com limitações claras. Em particular, a cobertura é estritamente lexical: sinônimos, paráfrases ou expressões mais gerais que capturem a mesma ideia que um *n-gram* relevante

não são reconhecidos como coincidência. Por outro lado, uma explicação pode repetir vários termos importantes sem necessariamente organizar um raciocínio jurídico coerente. Por isso, métricas de cobertura tendem a ser vistas como um ponto de partida quantitativo para avaliar fidelidade, mas não substituem análises qualitativas ou avaliações humanas mais detalhadas.

2.9 Métricas de avaliação de modelos de classificação

A avaliação de modelos de classificação não depende apenas de uma única métrica. Em problemas multiclasse e com classes desbalanceadas, diferentes medidas podem destacar aspectos distintos do desempenho do modelo (BISHOP, 2006; SOKOLOVA; LAPALME, 2009). Nesta seção, são revisadas algumas métricas clássicas usadas em classificação de textos: matriz de confusão, acurácia, precisão, revocação e F1-score, com ênfase na variante F1 macro.

2.9.1 Matriz de confusão

A matriz de confusão resume, em forma de tabela, as contagens de acertos e erros de um classificador (MANNING; RAGHAVAN; SCHÜTZE, 2008). No caso binário, com uma classe positiva e uma negativa, ela pode ser organizada como:

	Rótulo real	
Predição	Positivo	Negativo
Positivo	VP (verdadeiro positivo)	FP (falso positivo)
Negativo	FN (falso negativo)	VN (verdadeiro negativo)

Os verdadeiros positivos (VP) são exemplos positivos que o modelo acertou, enquanto os falsos positivos (FP) correspondem a exemplos negativos classificados como positivos. De forma análoga, falsos negativos (FN) são positivos que o modelo errou, e verdadeiros negativos (VN) são negativos classificados corretamente. Em problemas multiclasse, a matriz de confusão pode ser generalizada para uma tabela $K \times K$, em que cada linha representa a classe predita e cada coluna representa a classe real.

2.9.2 Acurácia

A acurácia mede a proporção de exemplos corretamente classificados em relação ao total de exemplos avaliados. No caso geral, em termos da matriz de confusão, ela pode ser definida como:

$$\text{Acc} = \frac{\text{soma dos elementos da diagonal}}{\text{número total de exemplos}}. \tag{2.25}$$

Em problemas binários, isso equivale a:

$$\text{Acc} = \frac{VP + VN}{VP + FP + FN + VN}. \tag{2.26}$$

Apesar de simples e intuitiva, a acurácia pode ser enganosa em cenários com classes desbalanceadas (SOKOLOVA; LAPALME, 2009). Por exemplo, se 95% dos exemplos pertencem a uma mesma classe, um classificador que sempre prediz essa classe terá acurácia de 95%, mesmo sem aprender nada sobre a classe minoritária. Por isso, costumam-se usar métricas que levem em conta separadamente erros e acertos por classe.

2.9.3 Precisão, revocação e F1-score

Em classificação, especialmente em problemas desbalanceados, é comum analisar *precisão* e *revocação* para cada classe (SOKOLOVA; LAPALME, 2009). Considerando uma classe c, definem-se:

- TP_c: número de exemplos da classe c corretamente classificados como c;
- FP_c: número de exemplos de outras classes classificados incorretamente como c;
- FN_c: número de exemplos da classe c classificados incorretamente como pertencentes a outras classes.

A *precisão* para a classe c mede a fração de predições da classe c que estão corretas:

$$\text{Prec}_c = \frac{TP_c}{TP_c + FP_c}, \tag{2.27}$$

enquanto a *revocação* (ou sensibilidade) mede a capacidade do modelo de recuperar exemplos da classe c:

$$\text{Rec}_c = \frac{TP_c}{TP_c + FN_c}. \tag{2.28}$$

O F1-score para a classe c é definido como a média harmônica entre precisão e revocação:

$$\text{F1}_c = 2 \cdot \frac{\text{Prec}_c \cdot \text{Rec}_c}{\text{Prec}_c + \text{Rec}_c}, \tag{2.29}$$

e assume valores entre 0 e 1, sendo maior quando tanto precisão quanto revocação são altas.

Em problemas multiclasse, é comum combinar esses valores por classe em uma única métrica agregada. Duas formas usuais são:

- **Métricas micro (micro-averaged)**: somam-se TP_c, FP_c e FN_c sobre todas as classes e, em seguida, calcula-se a precisão, revocação e F1 como se fosse um problema binário agregado. Nessa abordagem, classes com mais exemplos têm maior peso.

- **Métricas macro (macro-averaged)**: calcula-se a métrica por classe e, depois, tira-se a média aritmética entre as classes. Assim, cada classe contribui igualmente para o valor final, independentemente do seu tamanho (SOKOLOVA; LAPALME, 2009).

2.9.4 F1 macro como métrica principal

Em cenários em que as classes têm tamanhos bastante diferentes, mas cada classe é relevante do ponto de vista prático, é interessante usar uma métrica que não seja dominada pelas classes majoritárias. A F1 macro se enquadra nesse caso: ao calcular primeiro o F1 por classe e depois fazer a média entre as classes, essa medida trata igualmente classes frequentes e classes raras (SOKOLOVA; LAPALME, 2009). Em termos formais, se K é o número de classes, tem-se:

$$\mathrm{F1}_{\mathrm{macro}} = \frac{1}{K} \sum_{c=1}^{K} \mathrm{F1}_c. \tag{2.30}$$

Dessa forma, um modelo que performa muito bem apenas em classes grandes, mas mal em classes menores, tende a obter um F1 macro mais baixo do que um modelo que apresenta desempenho razoavelmente equilibrado entre as classes. Por isso, a F1 macro é frequentemente adotada como métrica principal em tarefas de classificação de textos multiclasse com distribuições desbalanceadas, sendo também amplamente suportada por bibliotecas de aprendizado de máquina que implementam o cálculo de F1 com opções de *macro averaging*.

3 Revisão da Literatura

Este capítulo apresenta um panorama dos trabalhos relacionados ao uso de técnicas de Processamento de Linguagem Natural (PLN) e aprendizado de máquina no domínio jurídico, com ênfase no contexto brasileiro e no Supremo Tribunal Federal (STF). Em seguida, discute contribuições recentes sobre explicabilidade de modelos de classificação de texto e o uso de grandes modelos de linguagem (LLMs) como ferramentas de apoio à decisão.

3.1 Aplicações de PLN e aprendizado de máquina no Direito

Nas últimas décadas, o uso de técnicas de PLN em textos jurídicos cresceu de forma significativa em diferentes jurisdições. Revisões recentes mostram que tarefas como classificação de documentos, previsão de julgamentos, sumarização de decisões e recuperação de precedentes vêm sendo tratadas com modelos baseados em redes neurais e *transformers* (CUI et al., 2022; ARIAI et al., 2024). Em geral, esses estudos buscam automatizar partes do fluxo de trabalho de tribunais e escritórios de advocacia, reduzindo o tempo gasto em tarefas repetitivas de leitura e organização de documentos.

No contexto de *legal judgment prediction*, por exemplo, Chalkidis, Androutsopoulos e Aletras (2019) introduzem um conjunto de casos da Corte Europeia de Direitos Humanos e avaliam diferentes arquiteturas neurais para prever o resultado do julgamento a partir da descrição dos fatos. Outros trabalhos vêm explorando modelos pré-treinados específicos para o domínio jurídico, como a família LEGAL-BERT, que adapta a arquitetura BERT a grandes coleções de textos legais e obtém ganhos consistentes em tarefas de classificação e recuperação de documentos (CHALKIDIS et al., 2020). Uma revisão recente destaca que esses modelos especializados tendem a superar abordagens baseadas apenas em *bag-of-words* ou *embeddings* genéricos em diversas tarefas jurídicas (ARIAI et al., 2024).

No Brasil, Martins e Silva (2022) realizam uma revisão sistemática sobre classificação de textos na área jurídica e mostram que há um interesse crescente em aplicar técnicas de PLN a documentos judiciais em português. Os autores identificam aplicações em classificação de sentenças, identificação de áreas do Direito, separação de tipos de peças processuais e análise de decisões, com o uso de modelos clássicos (como SVM e Naive Bayes) e, mais recentemente, de modelos baseados em *transformers*. Trabalhos como Gomes (2020) ilustram o uso de mineração de texto para classificar sentenças judiciais brasileiras, mostrando que mesmo modelos relativamente simples já podem auxiliar na organização de grandes acervos de decisões.

Especificamente em relação ao Supremo Tribunal Federal, o Projeto Victor foi desenvolvido para apoiar a triagem de recursos que chegam ao tribunal, especialmente recursos extraordinários, classificando peças por temas de repercussão geral e identificando processos repetitivos (FILHO; JUNQUILHO, 2018). A partir dessa iniciativa, Araujo et al. (2020) apresentaram o *VICTOR dataset*, composto por dezenas de milhares de recursos extraordinários do STF anotados com classes relacionadas à repercussão geral. Os autores avaliam diferentes modelos de classificação de texto, incluindo SVM, redes neurais recorrentes e modelos baseados em *embeddings*, mostrando que é possível obter bons resultados na identificação automática dos temas a partir do texto das peças. Esse conjunto de dados tornou-se uma referência para estudos posteriores sobre classificação de documentos jurídicos em língua portuguesa.

Em paralelo à evolução técnica, a literatura jurídica tem discutido os impactos do uso de inteligência artificial no Judiciário brasileiro. Autores como Maranhão, Florêncio e Almada (2021) e Valle (2023) chamam atenção para a necessidade de compatibilizar ganhos de eficiência com princípios como transparência, controle humano e respeito ao devido processo legal. Relatórios e pesquisas ligados ao Conselho Nacional de Justiça (CNJ) apontam que, apesar do aumento de projetos de IA nos tribunais, ainda há desafios relacionados à governança, à qualidade dos dados e à compreensão das decisões algorítmicas por parte de magistrados e servidores (Conselho Nacional de Justiça, 2020). Nesse cenário, há espaço para propostas que não apenas busquem bons resultados de classificação, mas também forneçam justificativas mais claras para as decisões automatizadas.

3.2 Explicabilidade e uso de LLMs em sistemas de apoio à decisão jurídica

A discussão sobre transparência e explicabilidade em modelos de aprendizado de máquina é ampla e não se limita ao Direito. De forma geral, a literatura distingue modelos considerados "caixa-preta" de modelos intrinsicamente interpretáveis e propõe diferentes critérios para avaliar a qualidade de explicações geradas para decisões algorítmicas (DOSHI-VELEZ; KIM, 2017; RUDIN, 2019). Em tarefas de classificação de texto, abordagens como visualização de pesos em modelos lineares, explicações baseadas em *saliency* e métodos pós-hoc como LIME e SHAP são comuns para identificar quais termos ou expressões mais contribuíram para uma predição (MOLNAR, 2022; JURAFSKY; MARTIN, 2023).

No domínio jurídico, esse debate ganha força pelo fato de decisões automatizadas afetarem direitos fundamentais. Trabalhos que discutem o uso de IA no Judiciário brasileiro ressaltam que a falta de explicabilidade pode comprometer a legitimidade das decisões e gerar desconfiança por parte de magistrados, advogados e cidadãos (MARANHÃO; FLORÊNCIO; ALMADA, 2021; VALLE, 2023). Em muitos casos, não basta que um

modelo acerte a classificação de um recurso ou de uma sentença; é necessário compreender quais elementos do texto foram considerados relevantes e se isso é compatível com critérios jurídicos aceitos.

Em diferentes jurisdições, parte da literatura em *legal judgment prediction* aponta justamente a tension entre desempenho e explicabilidade. Revisões como Cui et al. (2022) e Feng, Li e Ng (2022) destacam que muitos modelos de alto desempenho baseados em redes profundas oferecem pouca transparência, enquanto abordagens mais simples, como modelos lineares, são mais fáceis de interpretar, mas nem sempre atingem o mesmo nível de acerto. Alguns trabalhos exploram arquiteturas híbridas ou técnicas de atenção para tentar conciliar esses dois objetivos, mas ainda há um campo aberto para propostas que gerem explicações em linguagem natural mais próximas da forma como juristas justificam suas decisões.

Com o avanço dos grandes modelos de linguagem, surgiram novas possibilidades de uso de LLMs em tarefas jurídicas, incluindo responder perguntas sobre legislação, resumir decisões, sugerir rascunhos de peças e auxiliar na análise de casos (ARIAI et al., 2024; MASRI et al., 2025). Revisões recentes sobre o uso de LLMs na área jurídica mostram que esses modelos têm sido aplicados tanto de forma direta, como classificadores ou geradores de texto, quanto de forma complementar, por exemplo para explicar o resultado de outros modelos ou para reescrever decisões em linguagem mais acessível. Ainda assim, esses estudos também apontam desafios importantes, como o risco de alucinações (*hallucinations*), vieses herdados de dados de treinamento e a dificuldade de avaliar, de forma sistemática, a fidelidade das explicações geradas em relação aos sinais efetivamente utilizados pelo modelo subjacente.

Dentro desse cenário, ganha relevância a ideia de usar LLMs não apenas como classificadores, mas como *explicadores* em linguagem natural de decisões produzidas por outros modelos. Em uma abordagem típica, um classificador de texto (por exemplo, um modelo linear com representação TF–IDF ou um *transformer* ajustado para classificação) produz o rótulo para um documento e também um conjunto de atributos mais relevantes para aquela decisão (como os n-gramas de maior peso). Essas informações podem então ser fornecidas como contexto a um LLM, que gera uma explicação textual para a classificação, em estilo mais próximo da argumentação jurídica. A literatura recente sugere que esse tipo de combinação pode aumentar a utilidade prática das ferramentas de IA para operadores do Direito, mas ressalta a necessidade de métricas que avaliem a *fidelidade* das explicações, isto é, o quanto elas de fato se apoiam nas mesmas evidências que guiaram a classificação (ARIAI et al., 2024).

Uma linha de pesquisa relacionada consiste em propor medidas quantitativas de fidelidade entre explicações e modelos subjacentes, por exemplo comparando quais atributos são destacados em uma explicação e quais tiveram maior peso na decisão do

modelo. Métricas simples, como a cobertura de atributos importantes (*feature coverage*), podem servir como ponto de partida para avaliar se o texto explicativo realmente faz referência às mesmas pistas que o modelo utilizou para classificar o documento. Embora tais métricas não capturem todos os aspectos semânticos envolvidos, elas ajudam a aproximar a avaliação de explicações de critérios mais objetivos e reproduzíveis, algo particularmente desejável em contextos sensíveis como o sistema de justiça.

4 Metodologia

Este capítulo descreve a metodologia adotada para avaliar diferentes abordagens de classificação automática de documentos jurídicos do Supremo Tribunal Federal (STF) a partir da base Victor. São apresentados, de forma sistemática, o processo de preparação dos dados, o ambiente computacional utilizado, os modelos de classificação considerados (tanto métodos clássicos quanto modelos de linguagem baseados em Transformer) e as arquiteturas híbridas propostas para geração de explicações em linguagem natural. Por fim, detalha-se o protocolo de avaliação empregado, incluindo as métricas utilizadas tanto para desempenho de classificação quanto para a fidelidade das explicações.

Ao longo do capítulo, procura-se separar claramente os aspectos de *como* os experimentos foram conduzidos (metodologia) dos resultados numéricos obtidos, que são apresentados e discutidos no Capítulo 5. Do ponto de vista organizacional, a Seção 4.1 descreve, em alto nível, a arquitetura da abordagem proposta. A Seção 4.2 descreve o ambiente computacional utilizado. A Seção 4.3 detalha o conjunto de dados e o pré-processamento aplicado. As Seções 4.4 e 4.5 apresentam, respectivamente, os modelos de classificação considerados e as arquiteturas híbridas para geração de explicações. Por fim, a Seção 4.6 formaliza o protocolo de avaliação e as métricas adotadas.

4.1 Visão geral da abordagem

De maneira resumida, a abordagem proposta neste trabalho pode ser vista como um pipeline com quatro componentes principais, ilustrado na Figura 14. A figura organiza a metodologia em: (1) preparação dos dados; (2) treinamento dos modelos de classificação; (3) arquitetura híbrida de geração de explicações; e (4) etapa de avaliação do desempenho e da fidelidade das explicações.

1. **Preparação dos dados**: a partir dos arquivos CSV fornecidos pela base Victor, realiza-se a limpeza dos textos, a agregação de páginas por documento e o mapeamento dos tipos documentais originais para um conjunto fixo de seis classes canônicas. São utilizados diretamente os *splits* de treino, validação e teste disponibilizados pelo dataset, preservando a separação original entre esses conjuntos.

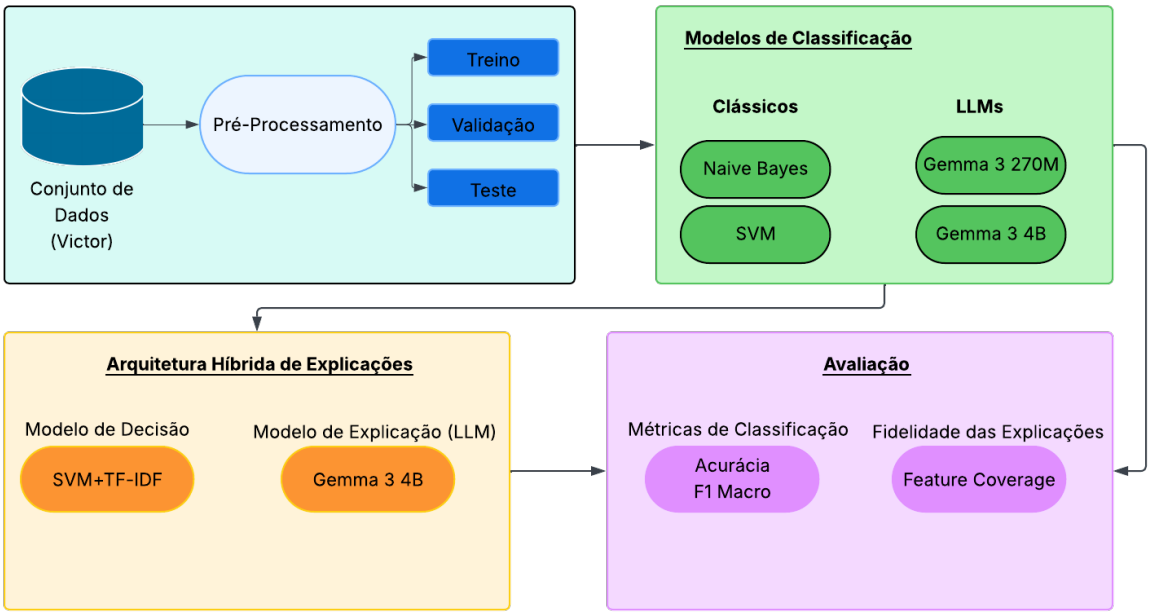
2. **Modelos de classificação**: sobre os textos consolidados são treinados e avaliados dois grupos de modelos: (i) baselines clássicos baseados em representações TF–IDF combinadas com classificadores lineares (SVM) e Naive Bayes; e (ii) modelos de linguagem do tipo Transformer (Gemma 3), ajustados para a tarefa de

sequence classification por meio de *fine-tuning* completo ou de técnicas eficientes em parâmetros (como QLoRA).

3. **Arquiteturas híbridas de explicação**: a partir de um modelo de decisão principal (em especial, o classificador SVM com TF–IDF), são extraídas evidências lexicais relevantes (uni- e bigramas com maior contribuição para a decisão), que são então fornecidas, juntamente com o texto do documento e o rótulo predito, a uma LLM em modo de conversa (Gemma 3 4B-*chat*) encarregada de gerar explicações em linguagem jurídica natural. Adicionalmente, compara-se uma variante em que o rótulo utilizado na explicação é obtido de um classificador neural (Gemma 3 270M-IT) em vez do SVM, embora essa variação não esteja explicitada em detalhe na figura.

4. **Protocolo de avaliação**: por fim, são definidas as métricas de desempenho de classificação (com ênfase no F1-macro) e uma métrica simples de fidelidade das explicações, baseada na proporção de *n-grams* relevantes que aparecem explicitamente no texto gerado pela LLM. Essa avaliação é conduzida tanto de forma global quanto estratificada por classe, utilizando as predições dos modelos e as explicações produzidas na etapa anterior.

Figura 14 – Esquema conceitual da abordagem proposta.



Fonte: elaborado pelo autor.

4.2 Ambiente computacional e ferramentas

Os experimentos foram conduzidos em ambiente *Python* (versão 3.10) utilizando o Google Colab[1] como plataforma principal de execução. Para os ajustes finos dos modelos de linguagem Gemma 3, foi empregada uma GPU NVIDIA A100[2] com 40 GB de memória, em configuração de GPU única.

Os modelos Gemma 3 foram obtidos a partir do Hugging Face Hub[3], tanto nas versões voltadas a *sequence classification* (Gemma 3 270M-IT[4] e Gemma 3 4B-IT[5]) quanto na versão de conversação usada como LLM explicadora. Tamanhos de *batch*, comprimentos máximos de sequência e demais hiperparâmetros de treinamento foram escolhidos de forma a respeitar a limitação de memória dessa GPU, garantindo que todos os experimentos fossem executados integralmente em uma única placa.

4.3 Conjunto de dados Victor e pré-processamento

4.3.1 Conjunto Victor e recorte utilizado

Os experimentos utilizam o conjunto de dados Victor, composto por peças processuais do Supremo Tribunal Federal (STF) anotadas com o tipo de documento (ARAUJO et al., 2020).

Foi adotado o recorte *small* disponibilizado pela base, que já traz os dados particionados em três arquivos CSV: `train_small.csv`, `validation_small.csv` e `test_small.csv`. Cada linha desses arquivos corresponde a uma página, contendo, entre outros campos, o texto extraído (`body`), o nome do arquivo original (`file_name`), o tipo documental original (`document_type`) e, quando disponível, o número da página (`pages`).

No recorte utilizado, a volumetria é a seguinte:

- **Treino**: aproximadamente 149,2 mil páginas, que, após agregação, resultam em 38 817 documentos distintos;
- **Validação**: cerca de 94,7 mil páginas, consolidadas em 25 455 documentos;
- **Teste**: aproximadamente 95,5 mil páginas, agregadas em 25 303 documentos.

Importante notar que os arquivos já chegam separados em treino, validação e teste pelo próprio Victor. Em todos os experimentos deste trabalho, essa partição original é preservada, sem reembaralhamento ou criação de novos *splits*.

[1] https://colab.research.google.com/
[2] https://www.nvidia.com/pt-br/data-center/a100/
[3] https://huggingface.co/
[4] https://huggingface.co/google/gemma-3-270m-it
[5] https://huggingface.co/google/gemma-3-4b-it

4.3.2 Limpeza textual e agregação por documento

O campo `body` do Victor nem sempre vem como texto puro: em diversos casos, o conteúdo está envolto por estruturas como `{ "..."}` ou por chaves simples. Para uniformizar o texto, foi definida a função `clean_body_text`, que executa três passos principais:

1. conversão do valor para `string` e remoção de espaços em branco nas extremidades;
2. remoção de invólucros externos, tratando casos do tipo `{"texto"}` e `{texto}`;
3. correção de aspas duplicadas, substituindo ocorrências de por ".

Entradas nulas são convertidas para a cadeia vazia, garantindo que não haja valores faltantes no texto de entrada dos modelos.

O dataset original está em nível de página, mas o problema de interesse é a classificação do tipo de peça como um todo. Por isso, em cada um dos três arquivos (treino, validação e teste) é realizado um pré-processamento com os seguintes passos:

1. aplicação da função `clean_body_text` em todas as linhas;
2. criação de uma coluna `label` com o tipo documental já mapeado (Seção 4.3.3);
3. ordenação das linhas por `file_name` e, quando disponível, por `pages`;
4. agregação das páginas de um mesmo documento por meio de um `groupby` em `(file_name, label)`, concatenando os campos `body` em um único texto.

O resultado são três *dataframes* em nível de documento, cada um com uma coluna de texto (`body`) e uma coluna de rótulo (`label`). Essa opção de trabalhar com o documento inteiro, e não com páginas isoladas, é motivada pelo fato de que o tipo de peça costuma ser determinado por elementos que podem aparecer em diferentes trechos do texto (preâmbulo, fundamentação, dispositivo), exigindo uma visão mais ampla do contexto.

4.3.3 Mapeamento de classes e distribuição

Os tipos documentais originais do Victor são mais finos e incluem códigos como `peticao_do_re` ou `acordao_de_2_instancia`. Para tornar o problema mais estável e compatível com a tarefa de interesse, foi definida a função `map_label`, que mapeia o campo

`document_type` para seis classes canônicas:

$$\begin{aligned} \texttt{peticao\_do\_re} &\to \texttt{RE}, \\ \texttt{agravo\_em\_recurso\_extraordinario} &\to \texttt{ARE}, \\ \texttt{sentenca} &\to \texttt{SENTENCA}, \\ \texttt{acordao\_de\_2\_instancia} &\to \texttt{ACORDAO}, \\ \texttt{despacho\_de\_admissibilidade} &\to \texttt{DESPACHO}, \\ \text{qualquer outro valor} &\to \texttt{OUTROS}. \end{aligned}$$

As classes são fixadas na seguinte ordem:

```
FIXED_LABELS = ["ACORDAO", "ARE", "DESPACHO", "OUTROS", "RE", "SENTENCA"]
```

e são construídos dois dicionários auxiliares: `label2id`, que associa cada rótulo a um inteiro de 0 a 5, e `id2label`, que faz o mapeamento inverso. Esses mapeamentos são usados em todo o notebook, tanto pelos modelos clássicos quanto pelos modelos baseados em Transformers.

A Tabela 3 resume a distribuição de documentos por classe em cada *split* após o pré-processamento:

Tabela 3 – Distribuição de documentos por classe nos conjuntos de treino, validação e teste.

Classe	**Treino**	**Validação**	**Teste**
ACORDAO	301	198	197
ARE	266	227	203
DESPACHO	265	143	146
OUTROS	37 115	24 293	24 194
RE	450	317	301
SENTENCA	420	277	262

Observa-se um grande desbalanceamento: a classe `OUTROS` concentra a grande maioria dos documentos em todos os conjuntos, enquanto as demais classes têm poucas centenas de exemplos cada. Todas as experiências de treinamento e avaliação descritas nos capítulos seguintes utilizam *exatamente* esse mapeamento de classes e esses mesmos conjuntos *train*/*validation*/*test*, garantindo comparabilidade entre modelos clássicos, modelos baseados em LLMs e arquiteturas híbridas de explicação.

4.4 Modelos de classificação de documentos

Esta seção descreve os modelos utilizados para atribuir rótulos às peças jurídicas, isto é, para resolver a tarefa de classificação multiclasse nas seis categorias definidas na Seção 4.3.3. São considerados dois grupos principais: (i) baselines clássicos baseados em TF–IDF com classificadores lineares; e (ii) modelos de linguagem do tipo Transformer Gemma 3 (Gemma Team, 2025), ajustados por *fine-tuning* para *sequence classification*. As explicações em linguagem natural, produzidas posteriormente por uma LLM em modo de conversa, são tratadas separadamente na Seção 4.5.

4.4.1 Representação TF–IDF

Para os modelos clássicos, o texto de cada documento consolidado (Seção 4.3.2) é representado por meio de um *vetor de características* construído com a classe `TfidfVectorizer` da biblioteca `scikit-learn`. O vetor de TF–IDF é obtido a partir da contagem de unigramas e bigramas, normalizada pela frequência de documentos em que cada termo aparece. No código, o `vectorizer` é configurado da seguinte forma:

- `max_features = 200000`, limitando o vocabulário aos 200 000 *n-grams* mais frequentes no conjunto de treino;
- `ngram_range = (1, 2)`, incluindo unigramas e bigramas;
- `lowercase = True`, convertendo todo o texto para minúsculas antes da tokenização.

O vocabulário (isto é, o mapeamento de *n-grams* para índices) é aprendido exclusivamente a partir do conjunto de treino. Uma vez ajustado, o mesmo `TfidfVectorizer` é utilizado para transformar os documentos dos conjuntos de validação e teste, garantindo que todas as etapas de avaliação utilizem exatamente o mesmo espaço vetorial.

4.4.2 Baselines clássicos: SVM e Naive Bayes

Como ponto de partida, são considerados dois baselines clássicos, ambos implementados como *pipelines* em `scikit-learn`, combinando a representação TF–IDF descrita na Seção 4.4.1 com um classificador supervisionado:

- **SVM linear com TF–IDF** (`svm_tfidf`): pipeline composto por um `TfidfVectorizer` seguido de um classificador `LinearSVC` com `class_weight="balanced"`. O uso de pesos balanceados faz com que a SVM penalize mais fortemente erros nas classes minoritárias, mitigando parcialmente o desbalanceamento apresentado na Tabela 3.

- **Naive Bayes Multinomial com TF–IDF** (`nb_tfidf`): pipeline que utiliza o mesmo `TfidfVectorizer`, seguido de um classificador `MultinomialNB`, apropriado para dados de contagem ou ponderados de forma não-negativa como TF–IDF.

Em ambos os casos, o protocolo é o mesmo: os modelos são treinados sobre o conjunto de treino em nível de documento e avaliados, sem ajuste adicional, nos conjuntos de validação e teste. Para cada modelo, são computadas a acurácia e o F1-macro, além de um relatório de classificação (`classification_report`) com métricas por classe, que será retomado no Capítulo 5.

Entre os dois baselines, o classificador SVM com TF–IDF apresentou desempenho superior em F1-macro no conjunto de teste e, por isso, foi adotado como modelo clássico principal. O pipeline completo (`TfidfVectorizer` + `LinearSVC`) é serializado em disco com `joblib` e reutilizado posteriormente tanto para comparação com as LLMs quanto como componente de decisão na arquitetura híbrida de explicações (Seção 4.5).

4.4.3 Classificador Gemma 3 270M-IT (fine-tuning completo)

Para explorar modelos de linguagem pré-treinados, foi utilizado o modelo Gemma 3 270M-IT (Gemma Team, 2025), uma LLM de porte moderado ($\approx$ 270 milhões de parâmetros) ajustada originalmente para diálogo. Neste trabalho, esse modelo é adaptado para classificação de sequência por meio de *full fine-tuning*.

Conversão para `datasets.DatasetDict`. Os três *dataframes* pré-processados são convertidos para objetos da biblioteca `datasets`, resultando em um `DatasetDict` com `train`, `validation` e `test`. Cada *split* possui duas colunas: `text` (conteúdo do documento) e `label` (rótulo canônico), obtidas pela simples renomeação das colunas `body` e `label`, sem alteração do conteúdo.

Tokenização. A tokenização é realizada com o `AutoTokenizer` associado ao modelo `google/gemma-3-270m-it`, configurado com `trust_remote_code=True`. Caso o token de preenchimento (*pad*) não esteja definido, ele é alinhado ao token de fim de sequência (*eos*). Para cada exemplo, aplica-se truncamento com `max_length = 768`, `truncation=True` e `padding=False`, sendo o preenchimento feito dinamicamente pelo `DataCollatorWithPadding`. O rótulo textual é mapeado para um inteiro em $\{0, \ldots, 5\}$ via `label2id`, armazenado no campo `labels`. O resultado é um conjunto tokenizado com tensores de entrada (`input_ids`, `attention_mask`) e o rótulo numérico.

Pesos de classe. Para lidar com o forte desbalanceamento entre classes, são calculados pesos de classe a partir da distribuição de rótulos no conjunto de treino. Seja n_c o número de documentos da classe c, N o total de documentos de treino e C o número

de classes. O peso associado à classe c é dado por

$$w_c = \frac{N}{C \cdot n_c}. \tag{4.1}$$

Esses pesos são armazenados em um vetor `class_weights_tensor` e posteriormente usados na função de perda ponderada.

Arquitetura e fine-tuning. O modelo de base é carregado, especificando `num_labels = 6`, `id2label` e `label2id`. O *checkpoint* `google/gemma-3-270m-it` é carregado em precisão `float32` (`torch_dtype=torch.float32`) em uma única GPU, com `use_cache=False`. Nenhuma técnica de PEFT é utilizada aqui: todos os parâmetros do modelo são atualizados (*full fine-tuning*).

O treinamento é conduzido por um *trainer* customizado (`WeightedTrainer270`), que sobrescreve o método `compute_loss` para aplicar uma `CrossEntropyLoss` ponderada pelos pesos $\{w_c\}$. Os principais hiperparâmetros de `TrainingArguments` são:

- `num_train_epochs = 5`;
- `per_device_train_batch_size = 8` e `gradient_accumulation_steps = 2`, resultando em um *batch* efetivo de aproximadamente 16 documentos;
- `learning_rate = 2e-5`;
- `weight_decay = 0.01` e `warmup_ratio = 0.1`;
- `eval_strategy = "steps"`, com `eval_steps = 2000` e `save_steps = 2000`;
- `load_best_model_at_end = True`, `metric_for_best_model = "f1_macro"`.

Durante o treinamento, são computadas acurácia e F1-macro em cada etapa de avaliação, utilizando a biblioteca `evaluate`. Ao final, o melhor *checkpoint* em validação (segundo o F1-macro) é mantido, e o modelo ajustado, juntamente com o *tokenizer*, é salvo no diretório `gemma-270m-victor-finalv2`, sendo reutilizado posteriormente na comparação de modelos e na arquitetura híbrida de explicações (Seção 4.5).

4.4.4 Classificador Gemma 3 4B-IT com QLoRA

Além do modelo de 270M parâmetros, foi treinado um modelo maior, Gemma 3 4B-IT (Gemma Team, 2025), adaptado para classificação de sequência por meio de uma combinação de quantização em 4 bits e LoRA (QLoRA). O objetivo é investigar se um modelo de maior capacidade, ajustado de forma eficiente em parâmetros, traz ganhos adicionais em relação ao *full fine-tuning* tradicional.

Tokenização. O procedimento segue o mesmo padrão da Seção 4.4.3, com a diferença de que se utiliza o *checkpoint* `google/gemma-3-4b-it` e um limite de contexto maior:

- `max_length = 1024`, permitindo que cada exemplo inclua um trecho mais longo do documento;
- truncamento e ausência de *padding* estático, com preenchimento dinâmico via `DataCollatorWithPadding`;
- mapeamento de rótulos via os mesmos dicionários `label2id` e `id2label`.

Os pesos de classe são os mesmos $\{w_c\}$ calculados a partir do conjunto de treino.

Quantização em 4 bits. Para reduzir o consumo de memória, o modelo base é carregado em formato quantizado de 4 bits utilizando a configuração `BitsAndBytesConfig`:

- `load_in_4bit = True`;
- `bnb_4bit_quant_type = "nf4"` (formato NF4);
- `bnb_4bit_compute_dtype = torch.float16`;
- `bnb_4bit_use_double_quant = True`.

O modelo é instanciado com `AutoModelForSequenceClassification`, `num_labels = 6`, `id2label`, `label2id` e `device_map = "auto"`, permitindo que a biblioteca `accelerate` faça o particionamento automático dos pesos entre GPU e CPU. Em seguida, o modelo é preparado para treino em baixa precisão com `prepare_model_for_kbit_training`.

Configuração LoRA (QLoRA). Sobre o modelo quantizado, é aplicada uma camada de PEFT com LoRA, conforme a Seção 2.7.2. A configuração utilizada é:

- `task_type = TaskType.SEQ_CLS`;
- `r = 16` (posto baixo das matrizes LoRA);
- `lora_alpha = 32` e `lora_dropout = 0.1`;
- `target_modules = ["q_proj","k_proj","v_proj","o_proj"]`, isto é, adaptações nas projeções do bloco de atenção;
- `bias = "none"`;
- `modules_to_save = ["score"]`, garantindo que a cabeça de classificação também seja preservada em alta precisão.

A função `get_peft_model` envolve o modelo base quantizado com os adapters LoRA, de modo que apenas esses parâmetros adicionais, juntamente com a camada de classificação, sejam atualizados durante o treinamento. Os pesos originais do modelo Gemma 4B em 4 bits permanecem congelados.

Hiperparâmetros de treinamento e perda ponderada. O treinamento é conduzido por um *trainer* customizado (`WeightedTrainer4B`), análogo ao caso de 270M, que aplica uma `CrossEntropyLoss` ponderada pelos mesmos pesos de classe $\{w_c\}$, garantindo que o vetor de pesos seja movido para o mesmo *device* e tipo dos logits antes do cálculo da perda.

Os principais hiperparâmetros de `TrainingArguments` são:

- `num_train_epochs = 3`;
- `per_device_train_batch_size = 4` e `gradient_accumulation_steps = 4`, mantendo um *batch* efetivo de cerca de 16 documentos, em linha com a limitação de memória;
- `learning_rate = 2e-4`;
- `weight_decay = 0.01`, `warmup_ratio = 0.03` e `max_grad_norm = 0.3`;
- `optim = "paged_adamw_8bit"`, adequado para cenários de quantização em baixa precisão;
- `fp16 = True` (cálculos em meia precisão), `eval_strategy = "steps"` com `eval_steps = 200`, `save_steps = 200`, `load_best_model_at_end = True` e `metric_for_best_model = "f1_macro"`.

Assim como no modelo de 270M parâmetros, o melhor *checkpoint* em validação (segundo o F1-macro) é mantido ao final do treino. Os adapters LoRA, juntamente com o *tokenizer*, são salvos no diretório `gemma-4b-victor-final`, permitindo a reutilização do classificador Gemma 3 4B-IT QLoRA nos experimentos de comparação de desempenho e na arquitetura de explicação híbrida descrita na Seção 4.5.

4.5 Arquiteturas híbridas de explicação

Além da avaliação puramente preditiva dos modelos de classificação (Seção 4.4), este trabalho investiga arquiteturas híbridas que combinam um *modelo de decisão* com uma LLM explicadora em modo de conversa. A ideia central é utilizar um classificador automático para gerar rótulos e evidências lexicais relevantes, e delegar a uma LLM de maior porte a tarefa de converter essas evidências em explicações em linguagem jurídica

natural. Esta seção descreve os componentes dessa arquitetura e os dois experimentos híbridos conduzidos.

4.5.1 Extração de evidências léxicas via SVM + TF–IDF

O ponto de partida das arquiteturas híbridas é o classificador SVM com TF–IDF apresentado na Seção 4.4.2, que é reutilizado aqui como *motor de decisão* e gerador de evidências léxicas. A partir do *pipeline* salvo em disco (`TfidfVectorizer` + `LinearSVC`), é possível acessar tanto o vocabulário TF–IDF (`feature_names`) quanto os coeficientes lineares associados a cada classe no modelo SVM.

A função `get_top_ngrams_for_document` implementa o procedimento de extração de evidências para um documento arbitrário. Dado um texto de entrada e o *pipeline* treinado, o processo segue os seguintes passos:

- o documento é classificado pela SVM, produzindo um rótulo predito `pred_label`, que é utilizado para selecionar a linha correspondente em `clf.coef_`, isto é, o vetor de coeficientes lineares associado à classe predita;
- em seguida, o texto é transformado pelo `TfidfVectorizer`, resultando em um vetor esparso de TF–IDF, que é convertido para uma representação densa `X_dense` de dimensão V, onde V é o tamanho do vocabulário;
- para cada termo do vocabulário, computa-se uma medida de contribuição linear dada por `contributions` = `X_dense` × `coef_class`, isto é, o produto elemento a elemento entre o peso TF–IDF do termo no documento e o coeficiente aprendido pela SVM para a classe predita;
- são mantidos apenas os termos com contribuição positiva (ou acima de um limiar mínimo `min_contribution`), descartando *n-grams* que contribuem negativamente para a decisão;
- as contribuições positivas são ordenadas em ordem decrescente, e a função retorna os `top_k` *n-grams* mais importantes, na forma de uma lista `top_features` contendo pares $(\text{termo}, \text{contribuição})$, juntamente com o rótulo predito.

Do ponto de vista conceitual, essa decomposição aproveita a natureza linear da SVM sobre o espaço TF–IDF para aproximar, localmente, quanta "responsabilidade" cada *n-gram* teve na decisão final. Esses termos de maior contribuição são então utilizados como evidências lexicais na fase de geração de explicações pela LLM.

4.5.2 LLM explicadora Gemma 3 4B-IT em modo conversa

A LLM explicadora utilizada nas arquiteturas híbridas é o modelo Gemma 3 4B-IT (Gemma Team, 2025) em sua variante de diálogo, instanciado. Diferentemente dos classificadores da Seção 4.4, esse modelo não é novamente ajustado: é utilizado *como está*, apenas condicionado por *prompts* construídos programaticamente.

Carregamento do modelo e processamento. O modelo é carregado a partir do *checkpoint* `google/gemma-3-4b-it`, com `device_map = "auto"` e `torch_dtype = torch.bfloat16`, deixando a alocação entre GPU e CPU a cargo da infraestrutura de `accelerate`. O modo de avaliação (`model.eval()`) é ativado e o processador correspondente (`AutoProcessor`) é carregado para cuidar tanto da tokenização quanto da aplicação do *chat template*.

Construção do prompt. Para controlar o tamanho do contexto textual enviado à LLM, é utilizada a função `truncate_text_for_prompt`, que limita o corpo do documento a cerca de 3000 caracteres. Caso o texto seja truncado, uma mensagem de aviso é anexada ao final (*"[Texto truncado para fins de explicação.]"*).

A função `build_explainer_prompt_text` recebe como entrada: (i) um trecho do documento (`document_text`); (ii) o rótulo predito para esse documento (`predicted_label`); e (iii) a lista de *n-grams* mais importantes (`top_features`) retornada pela SVM. A partir dessas informações, ela constrói um texto instrucional em português, que:

- apresenta um trecho do documento entre aspas;
- explicita a classe predita pelo modelo de decisão;
- lista as palavras e expressões mais relevantes para a classificação;
- orienta a LLM a explicar, em linguagem jurídica clara, por que o documento é compatível com a classe indicada, fazendo referência aos termos destacados sempre que pertinente;
- impõe restrições para evitar alucinações (*"não invente fatos que não estejam no texto"*) e limita a explicação a um tamanho de 1 a 3 parágrafos;
- encerra o *prompt* com o marcador *"Explanação:"*, que será utilizado posteriormente para extrair apenas a parte relevante da resposta.

A função `build_chat_messages` empacota esse texto em uma estrutura de mensagens no formato esperado pelo `AutoProcessor`. O *turno* de `system` define o papel da LLM como "assistente jurídico especializado em explicar decisões de um classificador automático de documentos do STF", enquanto o *turno* de `user` contém o *prompt* detalhado gerado anteriormente.

Geração determinística. A função `generate_explanation_with_llm` aplica o *chat template* por meio de `apply_chat_template` com `add_generation_prompt = True`, tokeniza as mensagens e envia os tensores para o dispositivo do modelo. Em seguida, chama-se `generate` com um limite de `max_new_tokens` entre 256 e 400 (dependendo do experimento) e `do_sample = False`, de modo a produzir respostas determinísticas para um mesmo *prompt*.

O texto completo (incluindo o *prompt*) é então decodificado, e uma heurística simples é aplicada: toma-se apenas o trecho posterior ao marcador *"Explanação:"* como a explicação final a ser utilizada nos experimentos.

4.5.3 Experimento híbrido 1: SVM + LLM explicadora

O primeiro experimento híbrido avalia a arquitetura composta por:

1. SVM com TF–IDF (Seção 4.4.2) como modelo de decisão principal; e
2. Gemma 3 4B-IT em modo conversa (Seção 4.5.2) como LLM explicadora.

Pipeline por documento. A função `hybrid_explain_document_svm` encapsula o fluxo completo para um único documento. Dados o texto do documento e o rótulo verdadeiro (`true_label`), o procedimento é:

- a SVM é aplicada ao texto, por meio de `get_top_ngrams_for_document`, produzindo o rótulo predito `pred_label` e a lista de termos mais relevantes `top_features`;
- o texto, o rótulo predito e as evidências lexicais são então fornecidos à LLM explicadora por meio de `generate_explanation_with_llm`, com um limite típico `max_new_tokens = 400`, resultando em uma explicação em linguagem natural;
- por fim, é calculada a métrica de *feature coverage* definida na Seção 4.6, utilizando a função `compute_feature_coverage`: o texto da explicação é convertido para minúsculas, e conta-se a proporção de *n-grams* de `top_features` que aparecem literalmente na explicação.

A função retorna um dicionário contendo o rótulo verdadeiro (`true_label`), o rótulo predito pela SVM (`pred_label`), a lista de termos mais importantes (`top_features`), o texto da explicação (`explanation`) e a cobertura lexical (`feature_coverage`).

Protocolo experimental. Para comparar o comportamento da arquitetura entre classes, é realizada uma amostragem estratificada no conjunto de teste: para cada classe em `FIXED_LABELS`, são selecionados até $N\_PER\_CLASS = 5$ documentos, via `sample`

com `random_state = 42` para garantir reprodutibilidade. Os subconjuntos por classe são concatenados em um único *dataframe* `samples_df`.

Para cada linha de `samples_df`, a função `hybrid_explain_document_svm` é chamada com `top_k = 10` e `max_new_tokens = 400`. Em seguida, os resultados são organizados em um *dataframe* `exp_df` com as colunas:

- `true_label` (rótulo verdadeiro do documento);
- `pred_label` (rótulo predito pela SVM);
- `top_terms` (lista de termos relevantes concatenados em uma string);
- `feature_coverage` (cobertura lexical para o documento);
- `explanation` (texto da explicação gerada pela LLM);
- `document_text` (trecho do documento, truncado para cerca de 2000 caracteres via `truncate_text_for_prompt`).

A partir de `exp_df`, calcula-se a cobertura média global e a cobertura média por classe verdadeira (via `groupby` em `true_label`). Tanto as explicações detalhadas quanto as métricas agregadas são persistidas em arquivos CSV na pasta `BASE_SAVE_DIR`, que serão utilizados no Capítulo 5 para análise qualitativa e quantitativa das explicações.

4.5.4 Experimento híbrido 2: comparação SVM vs Gemma 270M como motor de decisão

O segundo experimento híbrido tem como objetivo comparar duas variantes da arquitetura de explicação, mantendo a mesma LLM explicadora, mas trocando o modelo de decisão que fornece o rótulo de entrada:

1. **Variante (a):** SVM + LLM explicadora, tal como em Seção 4.5.3;
2. **Variante (b):** classificador Gemma 3 270M-IT (Seção 4.4.3) + LLM explicadora.

Em ambos os casos, as evidências lexicais (*n-grams* importantes) são extraídas do SVM com TF–IDF, de modo que a comparação foque na influência do rótulo de entrada (SVM vs Gemma 270M) sobre o comportamento da LLM explicadora, mantendo fixas as "pistas" lexicais.

Predição com o classificador Gemma 3 270M-IT. A função `predict_with_gemma270` realiza a inferência com o modelo Gemma 3 270M-IT fine-tunado e salvo no Drive. O *checkpoint* é carregado com `AutoTokenizer` e `AutoModelForSequenceClassification`,

usando `device_map = "auto"` e `dtype = torch.float16` para inferência eficiente. Dado um texto, a função:

- aplica a tokenização com truncamento (`max_length = 1024`) e `padding=False`;
- envia os tensores para o dispositivo do modelo;
- calcula os logits, toma o `argmax` e converte o índice em rótulo textual por meio de `model_270.config.id2label`.

Geração de explicações duplas. A função `explain_document_svm_and_llm` recebe o texto de um documento, o rótulo verdadeiro e o *pipeline* SVM. A lógica é a seguinte:

- primeiro, são extraídas o rótulo predito pela SVM (`svm_pred`) e as evidências lexicais `top_features` via `get_top_ngrams_for_document`;
- utilizando o rótulo `svm_pred` e as mesmas evidências lexicais, gera-se uma explicação com a LLM por meio de `generate_explanation_with_llm`, e calcula-se a cobertura lexical `svm_feature_coverage`;
- em seguida, o mesmo texto é classificado pelo Gemma 3 270M-IT, obtendo-se um segundo rótulo `llm_pred`;
- com o rótulo `llm_pred` e as mesmas `top_features`, gera-se uma segunda explicação via `generate_explanation_with_llm` e calcula-se a cobertura lexical `llm_feature_coverage`.

O resultado para cada documento inclui o rótulo verdadeiro (`true_label`), os rótulos preditos (`svm_pred` e `llm_pred`), a lista de termos mais importantes, as duas explicações em linguagem natural e as duas métricas de cobertura lexical.

Protocolo experimental. Para este experimento, é utilizada uma amostragem maior do conjunto de teste: para cada classe em `FIXED_LABELS`, são selecionados até $N\_\text{PER}\_\text{CLASS} = 15$ documentos, via `sample` com `random_state = 123`. Esses subconjuntos são concatenados em um *dataframe* `samples_df` contendo aproximadamente 90 documentos no total.

Para cada exemplo em `samples_df`, a função `explain_document_svm_and_llm` é executada com `top_k = 10` e `max_new_tokens = 256`. Os resultados são agregados em um *dataframe* `exp_compare_df` com as colunas:

- `true_label`, `svm_pred`, `llm_pred`;

- `top_terms`;
- `svm_feature_coverage` e `llm_feature_coverage`;
- `svm_explanation` e `llm_explanation`;
- `document_text` (trecho truncado do documento).

A partir de `exp_compare_df`, são calculadas as coberturas médias globais para cada variante e as coberturas médias por classe verdadeira (`true_label`), por meio de uma agregação `groupby`. Tanto o *dataframe* completo de explicações quanto as estatísticas agregadas são salvos em arquivos CSV na pasta `BASE_SAVE_DIR`, e serão utilizados no Capítulo 5 para discutir, em detalhe, a fidelidade lexical das explicações e o impacto da escolha do modelo de decisão.

4.6 Protocolo de avaliação

Esta seção descreve as métricas e procedimentos utilizados para avaliar tanto o desempenho preditivo dos modelos de classificação quanto a fidelidade das explicações geradas pelas arquiteturas híbridas apresentadas na Seção 4.5.

4.6.1 Métricas de desempenho de classificação

As métricas de desempenho empregadas seguem diretamente a discussão do Capítulo 2 (Seção 2.8), com foco em duas medidas principais: *acurácia* e F1-macro. A acurácia corresponde à proporção de exemplos corretamente classificados no conjunto de teste. Já o F1-macro é calculado como a média aritmética do F1-score por classe, atribuindo o mesmo peso a cada classe, independentemente de sua frequência, o que é particularmente importante no cenário desbalanceado da base Victor.

Na implementação, essas métricas são computadas com as funções `accuracy_score` e `f1_score` (`average="macro"`) da biblioteca `scikit-learn`. Em todos os experimentos de classificação (SVM/Naive Bayes, Gemma 3 270M-IT e Gemma 3 4B-IT QLoRA), os modelos são comparados tendo o F1-macro no conjunto de teste como métrica principal, sendo a acurácia utilizada como medida complementar. Além disso, são gerados relatórios de classificação detalhados (`classification_report`), com precisão, revocação e F1-score por classe, que serão analisados no Capítulo 5.

Nos treinamentos baseados em `Trainer` da biblioteca `transformers` (Gemma 3 270M-IT e Gemma 3 4B-IT QLoRA), o F1-macro em validação é utilizado como critério para seleção de *checkpoint*, via parâmetro `metric_for_best_model = "f1_macro"` e `load_best_model_at_end = True`. Assim, o modelo final avaliado no conjunto de teste é sempre aquele que obteve o melhor F1-macro no conjunto de validação.

4.6.2 Métrica de fidelidade das explicações

Para avaliar a qualidade das explicações geradas pelas arquiteturas híbridas, é adotada a métrica de *feature coverage* discutida conceitualmente na Seção 2.7.4. Essa métrica procura quantificar, de forma simples e objetiva, a fidelidade lexical da explicação em relação às evidências geradas pelo modelo de decisão.

Na implementação (`compute_feature_coverage`), utiliza-se, para cada documento, a lista dos k *n-grams* mais importantes retornados pela função `get_top_ngrams_for_document`, com `top_k = 10`. Dada uma explicação em linguagem natural produzida pela LLM, procede-se da seguinte forma:

- a explicação é convertida para letras minúsculas (*case-folding* simples), produzindo `exp_lower`;
- para cada termo na lista de evidências, realiza-se um pré-processamento simples (`strip()` e conversão para minúsculas); termos vazios após esse processo são ignorados;
- verifica-se se o termo aparece literalmente como *substring* em `exp_lower` (teste `term_clean in exp_lower`);
- conta-se o número de termos que aparecem pelo menos uma vez na explicação, e a cobertura é definida como a fração de termos presentes em relação ao total de termos considerados.

Formalmente, se $\mathcal{T} = \{t_1, \ldots, t_k\}$ é o conjunto de termos relevantes e $\mathcal{T}' \subseteq \mathcal{T}$ o subconjunto de termos que aparecem na explicação, a *feature coverage* de um documento é dada por $|\mathcal{T}'|/|\mathcal{T}|$, assumindo valores no intervalo $[0, 1]$.

Essa métrica é calculada:

- **por documento**, associando a cada explicação uma cobertura individual;
- **como média global**, tomando a média da cobertura sobre todos os documentos amostrados em cada experimento híbrido;
- **como média por classe**, agrupando os documentos por rótulo verdadeiro (`true_label`) e calculando a média da cobertura em cada grupo.

Os valores de cobertura por documento e as estatísticas agregadas (média global e médias por classe) são exportados para arquivos CSV gerados nas Células 7 e 8, sendo retomados no Capítulo 5 para a análise comparativa das arquiteturas de explicação.

Além das médias de *feature coverage*, foi calculado um intervalo de confiança aproximado de 95% para a cobertura global em cada experimento. Interpretando a cobertura como uma proporção de "acertos" sobre os pares (documento, *n-gram*) avaliados, seja $N = |D| \cdot k$ o número total de *n-grams* considerados (com $|D|$ documentos e k *n-grams* por documento) e seja $\hat{p}$ a cobertura média observada. Sob a aproximação binomial, o erro-padrão da estimativa é dado por

$$\mathrm{EP}(\hat{p}) \approx \sqrt{\frac{\hat{p}\,(1-\hat{p})}{N}},$$

e um intervalo de confiança de 95% para a cobertura verdadeira p é obtido por

$$\hat{p} \pm 1{,}96 \cdot \mathrm{EP}(\hat{p}).$$

Essa aproximação assume independência entre os pares (documento, *n-gram*), mas é suficiente para quantificar, de forma simples, a incerteza em torno das médias de cobertura reportadas.

4.6.3 Amostragem e reprodutibilidade

Os experimentos de explicabilidade descritos na Seção 4.5 operam sobre subconjuntos amostrados do conjunto de teste, de forma a tornar o custo de geração de explicações computacionalmente manejável, sem perder a diversidade de tipos de peça.

No **Experimento híbrido 1** (Seção 4.5.3), são selecionados até $N_{\text{PER\_CLASS}} = 5$ documentos por classe em `FIXED_LABELS`, resultando em aproximadamente 30 documentos no total. A amostragem é realizada separadamente para cada classe, por meio do método `sample` do `pandas`, com `random_state = 42`, garantindo reprodutibilidade exata do subconjunto utilizado.

No **Experimento híbrido 2** (Seção 4.5.4), a amostragem é ampliada para $N_{\text{PER\_CLASS}} = 15$ documentos por classe, o que leva a um total de cerca de 90 documentos. O procedimento é análogo, mas com `random_state = 123`, permitindo replicar exatamente o mesmo conjunto de exemplos na comparação entre as variantes SVM + LLM explicadora e Gemma 3 270M-IT + LLM explicadora.

Em todos os casos, as amostras são retiradas exclusivamente do conjunto de teste, que é mantido completamente separado dos dados utilizados no treino e na validação dos modelos de classificação. Dessa forma, preserva-se a independência entre os dados usados para ajustar os modelos e aqueles usados para avaliar tanto o desempenho preditivo quanto a qualidade das explicações.

5 Análise e discussão dos resultados

Neste capítulo são apresentados e discutidos os resultados obtidos com os diferentes modelos de classificação de documentos jurídicos e com as arquiteturas híbridas de explicação propostas. São analisados tanto os desempenhos de classificação (em termos de acurácia e F1-macro) quanto a fidelidade lexical das explicações geradas pela LLM explicadora, medida pela métrica de *feature coverage* definida na Seção 2.8.4 e operacionalizada no protocolo da Seção 4.6.

Ao longo do capítulo, são comparados: (i) os baselines clássicos baseados em TF–IDF com SVM e Naive Bayes; (ii) os modelos Gemma 3 270M-IT (com *full fine-tuning*) e Gemma 3 4B-IT ajustado via QLoRA; e (iii) duas arquiteturas híbridas de explicação, que combinam um modelo de decisão (SVM ou Gemma 3 270M-IT) com uma LLM em modo de conversa (Gemma 3 4B-IT *chat*) responsável por gerar justificativas em linguagem jurídica natural. As métricas principais seguem as definições do Capítulo 2, em especial a Seção 2.9 para desempenho de classificação e a Seção 2.8.4 para fidelidade das explicações.

Do ponto de vista de organização, a Seção 5.2 apresenta e discute os resultados dos modelos de classificação. A Seção 5.3 concentra-se nas arquiteturas híbridas de explicação e na análise da métrica de fidelidade. Por fim, a Seção 5.4 sintetiza os principais achados, discute implicações práticas e limitações, e prepara o terreno para as conclusões finais do trabalho.

5.1 Visão geral dos experimentos

Conforme detalhado no Capítulo 4, os experimentos deste trabalho dividem-se em dois eixos principais. No eixo de classificação, foram treinados: (i) dois baselines clássicos baseados em TF–IDF, combinados com um classificador SVM linear e com um Naive Bayes Multinomial (Seção 4.4.2); e (ii) dois modelos de linguagem da família Gemma 3, sendo o Gemma 3 270M-IT ajustado por *full fine-tuning* (Seção 4.4.3) e o Gemma 3 4B-IT ajustado por QLoRA (Seção 4.4.4), ambos formulados como modelos de *sequence classification* com seis rótulos.

No eixo de explicabilidade, foram conduzidos dois experimentos híbridos que combinam um modelo de decisão com uma LLM explicadora em modo de conversa (Gemma 3 4B-IT *chat*). No primeiro experimento, o classificador SVM com TF–IDF atua como *motor de decisão* e como gerador de evidências lexicais, fornecendo os *n-grams* mais relevantes da decisão para a LLM (Seção 4.5.3). No segundo experimento, essa arquitetura é estendida para comparar duas variantes: (a) SVM + LLM explicadora e (b) Gemma 3

270M-IT + LLM explicadora, mantendo fixo o conjunto de evidências léxicas extraídas pelo SVM (Seção 4.5.4).

Em todos os modelos de classificação, o desempenho é avaliado com base na acurácia e, principalmente, no F1-macro, conforme definido na Seção 2.9. Para as arquiteturas híbridas de explicação, a qualidade das justificativas é medida pela métrica de *feature coverage* (Seção 2.8.4), calculada por documento e agregada globalmente e por classe, seguindo o protocolo descrito na Seção 4.6. As próximas seções detalham e discutem os resultados obtidos nesses diferentes cenários.

5.2 Desempenho dos modelos de classificação de documentos

Nesta seção são apresentados e discutidos os resultados quantitativos dos modelos que atribuem rótulos às peças jurídicas, isto é, os baselines clássicos baseados em TF–IDF (SVM e Naive Bayes) e os modelos Gemma 3 ajustados para classificação de sequência. As métricas consideradas são a acurácia e o F1-macro no conjunto de teste, bem como o F1 por classe, em linha com as definições da Seção 2.9.

5.2.1 Baselines clássicos: SVM vs Naive Bayes

A Tabela 4 resume os resultados globais dos dois baselines clássicos, ambos treinados sobre o conjunto de treino em nível de documento (Seção 4.4.2) e avaliados nos conjuntos de validação e teste.

Tabela 4 – Resultados globais dos baselines clássicos com TF–IDF.

Modelo	**Acurácia(val)**	**F1-macro(val)**	**Acurácia(teste)**	**F1-macro(teste)**
SVM + TF–IDF	0,9793	0,8212	0,9802	0,8223
Naive Bayes + TF–IDF	0,9598	0,3513	0,9614	0,3447

A Tabela 5 detalha o F1 por classe no conjunto de teste, evidenciando o comportamento muito distinto dos dois modelos nas classes minoritárias.

Tabela 5 – F1-score por classe no conjunto de teste para os baselines clássicos.

Classe	SVM + TF–IDF	Naive Bayes + TF–IDF
ACÓRDÃO	0,9468	0,4943
ARE	0,6111	0,0000
DESPACHO	0,7622	0,3006
OUTROS	0,9897	0,9802
RE	0,7455	0,0455
SENTENÇA	0,8786	0,2475
F1-macro	0,8223	0,3447

Observa-se que o Naive Bayes atinge uma acurácia relativamente alta ($\approx 0{,}96$), mas um F1-macro muito baixo ($\approx 0{,}34$). Isso ocorre porque o modelo praticamente colapsa para a classe majoritária `OUTROS`: nessa classe, o F1 é elevado (0,9802), mas o desempenho nas demais classes é muito ruim, chegando a F1 igual a zero para `ARE` e valores próximos de zero para `RE` e `SENTENÇA`. Como o F1-macro atribui o mesmo peso a todas as classes, esse colapso nas classes minoritárias puxa drasticamente a média.

Já o SVM linear com `class_weight="balanced"` apresenta um comportamento muito mais estável entre as classes. Embora a acurácia também seja alta ($\approx 0{,}98$), o F1-macro sobe para cerca de 0,82, refletindo um equilíbrio melhor entre classes majoritárias e minoritárias. Mesmo nas classes com menos exemplos (`ARE`, `DESPACHO`, `RE`), o F1 situa-se na faixa de 0,61 a 0,76, o que é compatível com a forte assimetria da Tabela 3.

Esses resultados são coerentes com a intuição teórica discutida na Seção 2.4: modelos lineares com TF–IDF tendem a ser muito competitivos em classificação de texto, especialmente quando os textos são relativamente longos e as classes diferem por padrões léxicos claros. O uso de pesos de classe na SVM permite tratar explicitamente o desbalanceamento, enquanto o Naive Bayes, apesar de simples e eficiente, mostra-se pouco robusto nesse cenário.

Diante disso, o SVM com TF–IDF é adotado como baseline clássico principal e como *motor de decisão* nos experimentos híbridos de explicação (Seção 4.5.3), ao passo que o Naive Bayes é descartado como candidato competitivo para a tarefa.

5.2.2 Gemma 3 270M-IT (full fine-tuning) vs baseline clássico

A Tabela 6 compara diretamente o desempenho global do SVM com TF–IDF com o do modelo Gemma 3 270M-IT ajustado por *full fine-tuning*, ambos avaliados no conjunto de teste.

Tabela 6 – Comparação global entre SVM + TF–IDF e Gemma 3 270M-IT no conjunto de teste.

Modelo	**Acurácia (teste)**	**F1-macro (teste)**
SVM + TF–IDF	0,9802	0,8223
Gemma 3 270M-IT (full FT)	0,9825	0,8255

Em termos globais, o Gemma 3 270M-IT apresenta uma acurácia ligeiramente superior e um F1-macro marginalmente maior que o SVM. A Tabela 7 detalha o F1 por classe, evidenciando onde esses ganhos se concentram.

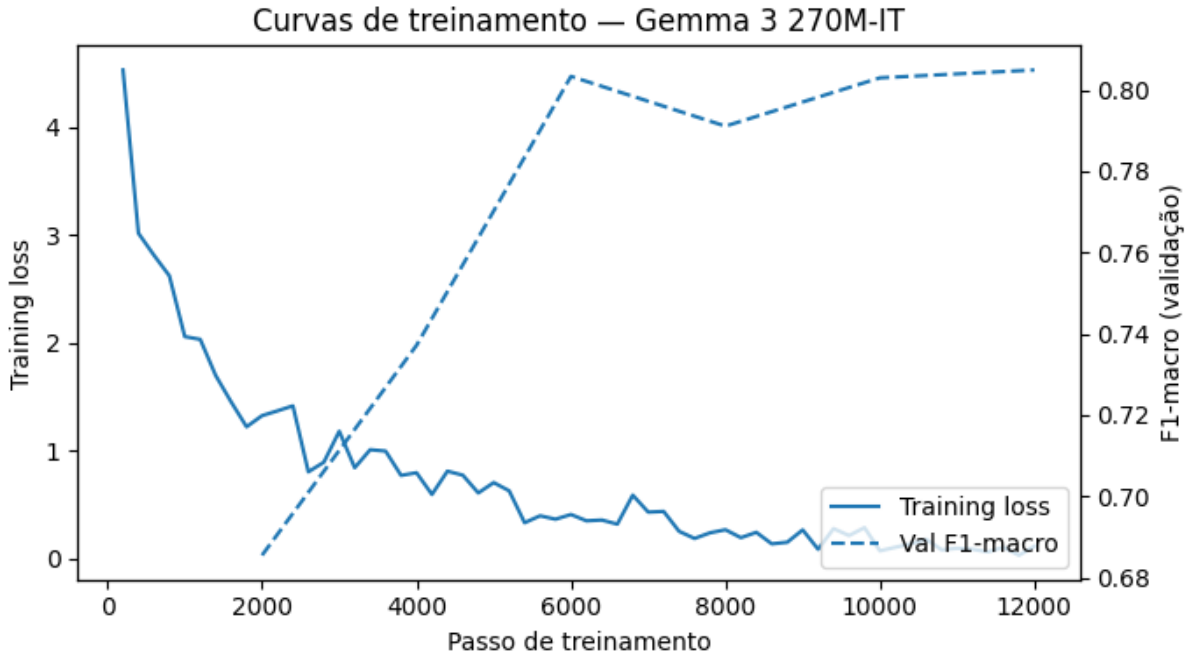
Tabela 7 – F1-score por classe no conjunto de teste: SVM + TF–IDF vs Gemma 3 270M-IT.

Classe	**SVM + TF–IDF**	**Gemma 3 270M-IT**
ACÓRDÃO	0,9468	0,9337
ARE	0,6111	0,5829
DESPACHO	0,7622	0,7832
OUTROS	0,9897	0,9909
RE	0,7455	0,7765
SENTENÇA	0,8786	0,8858
F1-macro	0,8223	0,8255

Os resultados indicam que o Gemma 3 270M-IT melhora principalmente em `DESPACHO` e `RE`, com ganhos de F1 em torno de alguns pontos percentuais, além de pequenos avanços em `SENTENÇA` e `OUTROS`. Por outro lado, em `ACÓRDÃO` e `ARE` o desempenho é ligeiramente inferior ao do SVM, embora ainda alto.

A Figura 15 mostra que o *training loss* decresce de forma monotônica enquanto o F1-macro em validação cresce rapidamente nas primeiras iterações e depois se estabiliza em torno de 0,80, em linha com os valores de teste das Tabelas 6 e 7. Isso sugere um processo de *fine-tuning* estável, sem sinais fortes de sobreajuste nas cinco épocas consideradas.

Figura 15 – Curvas de treinamento do classificador Gemma 3 270M-IT.

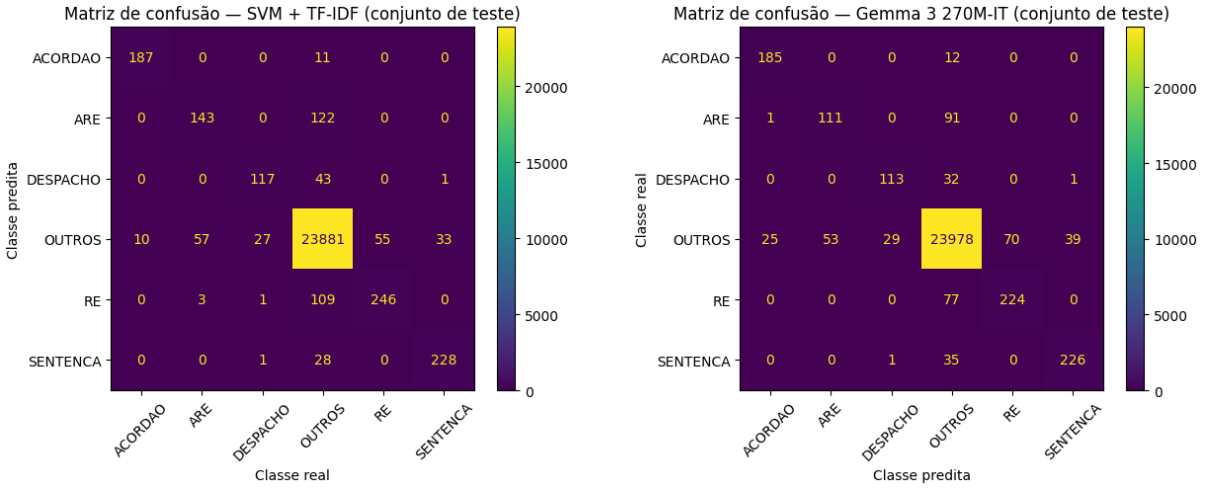


Fonte: elaborado pelo autor.

No conjunto, o ganho em F1-macro do Gemma 3 270M-IT em relação ao SVM é modesto. Isso sugere que, para a tarefa específica de classificação de tipo de peça no dataset Victor, um modelo linear bem configurado com TF–IDF já é capaz de capturar grande parte da estrutura do problema. Trata-se de uma tarefa fortemente lexical, em que rótulos como `PETIÇÃO`, `ACÓRDÃO` ou `DESPACHO` estão associados a padrões de forma e expressões jurídicas relativamente estáveis, o que favorece modelos lineares com bons recursos léxicos.

A distribuição desses acertos e erros fica mais clara nas matrizes de confusão das Figuras 16a e 16b. Em ambos os casos, a diagonal é dominante, indicando que a maior parte dos documentos é corretamente classificada. No entanto, nota-se que o SVM com TF–IDF comete menos erros ao confundir peças de classes minoritárias (`ARE`, `DESPACHO`) com `OUTROS`, enquanto o Gemma 3 270M-IT reduz ligeiramente a confusão de `RE` e `SENTENÇA` com `OUTROS`. Esse padrão é coerente com os F1 por classe apresentados na Tabela 7.

Figura 16 – Matrizes de confusão dos modelos principais no conjunto de teste.



(a) SVM + TF–IDF **(b)** Gemma 3 270M-IT (full fine-tuning)

Fonte: elaborado pelo autor.

5.2.3 Gemma 3 4B-IT QLoRA: impacto de um modelo maior com PEFT

A Tabela 8 resume os resultados globais dos três modelos principais de classificação: o SVM com TF–IDF, o Gemma 3 270M-IT com *full fine-tuning* e o Gemma 3 4B-IT ajustado via QLoRA.

Tabela 8 – Resultados globais no conjunto de teste: SVM, Gemma 3 270M-IT e Gemma 3 4B-IT QLoRA.

Modelo	Acurácia (teste)	F1-macro (teste)
SVM + TF–IDF	0,9802	0,8223
Gemma 3 270M-IT (full FT)	0,9825	0,8255
Gemma 3 4B-IT (QLoRA)	0,9835	0,8239

O modelo Gemma 3 4B-IT QLoRA atinge uma acurácia ligeiramente superior e um F1-macro muito próximo ao do Gemma 3 270M-IT. A Tabela 9 detalha o F1 por classe para os dois modelos Gemma, evidenciando que as diferenças são pequenas e distribuem-se de forma heterogênea entre as classes.

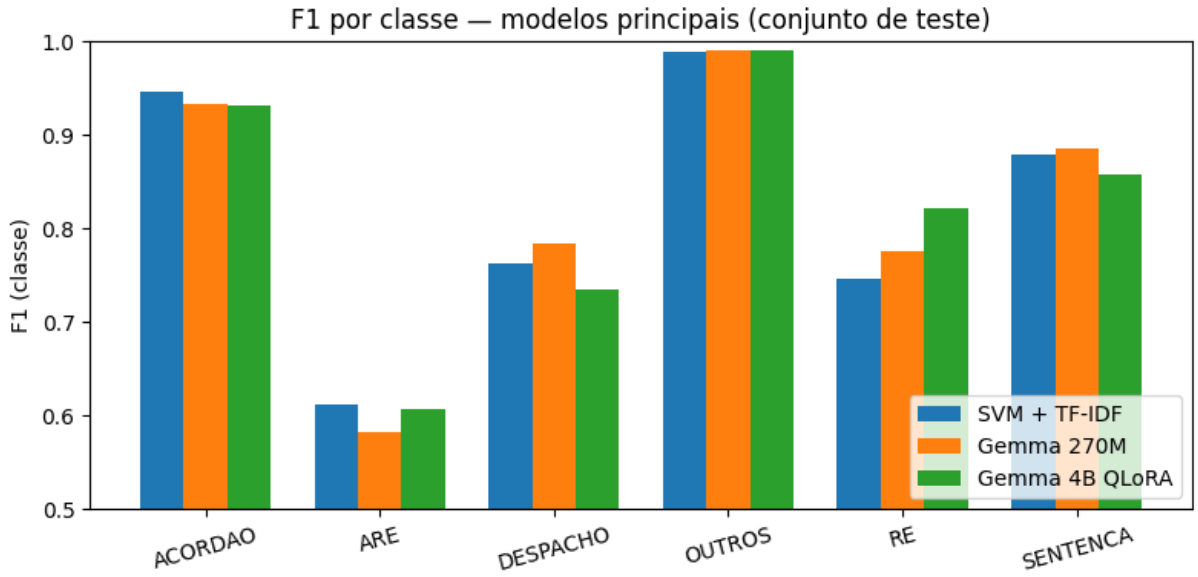
Tabela 9 – F1-score por classe no conjunto de teste: Gemma 3 270M-IT vs Gemma 3 4B-IT QLoRA.

Classe	Gemma 3 270M-IT	Gemma 3 4B-IT QLoRA
ACÓRDÃO	0,9337	0,9313
ARE	0,5829	0,6067
DESPACHO	0,7832	0,7341
OUTROS	0,9909	0,9915
RE	0,7765	0,8224
SENTENÇA	0,8858	0,8577
F1-macro	0,8255	0,8239

Verifica-se que o modelo 4B QLoRA melhora o F1 em classes como `ARE` e `RE`, além de um ligeiro aumento em `OUTROS`, mas perde um pouco em `DESPACHO` e `SENTENÇA`, mantendo desempenho muito próximo em `ACÓRDÃO`. Em média, o F1-macro dos dois modelos é praticamente equivalente, com variações dentro da ordem de centésimos.

A Figura 17 organiza esses resultados por classe para os três modelos principais. O gráfico reforça que o SVM já apresenta F1 elevado em quase todas as classes e que os modelos Gemma 3 apenas redistribuem pequenos ganhos e perdas: o 270M melhora levemente `DESPACHO`, enquanto o 4B QLoRA tende a favorecer `ARE` e `RE`, sem alterar de forma substantiva o patamar global de F1-macro.

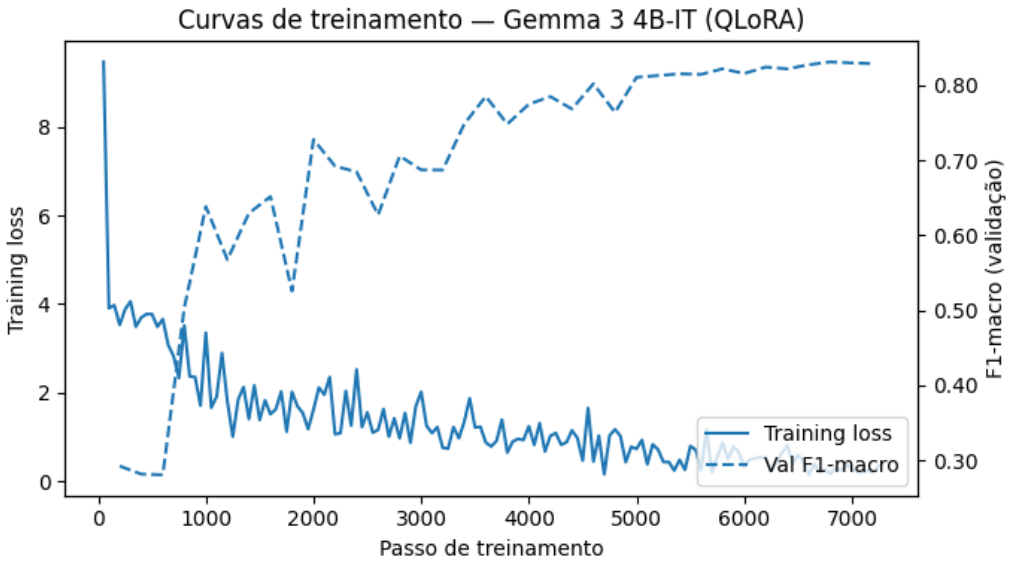
Figura 17 – F1 por classe no conjunto de teste para os três modelos principais de classificação.



Fonte: elaborado pelo autor.

A Figura 18 mostra que, apesar de o *training loss* apresentar mais ruído — em parte devido à quantização em 4 bits e ao *batch* efetivo menor — o F1-macro em validação cresce de forma consistente até valores próximos de 0,83, compatíveis com os resultados da Tabela 8. Assim como no caso do modelo de 270M parâmetros, o treino converge para um patamar estável de desempenho.

Figura 18 – Curvas de treinamento do classificador Gemma 3 4B-IT com QLoRA.



Fonte: elaborado pelo autor.

Alguns fatores podem explicar por que o aumento no número de parâmetros não se traduz em ganhos expressivos nesta tarefa: (i) o modelo 4B é treinado em formato quantizado (4 bits) com LoRA, o que restringe a capacidade efetiva de ajuste fino dos pesos; (ii) o número de épocas de treinamento é menor (3 épocas para o 4B, contra 5 para o 270M); e (iii) a própria natureza fortemente lexical da tarefa, já bem capturada tanto pelo SVM quanto pelo Gemma 3 270M-IT, reduz o espaço para melhorias substanciais de um modelo ainda maior.

O resultado é um achado interessante: para a classificação de tipo de peça na base Victor, um modelo Gemma 3 de porte moderado (270M parâmetros) e mesmo o SVM com TF–IDF já atingem um patamar de desempenho elevado, de modo que a passagem para um modelo de 4 bilhões de parâmetros ajustado via QLoRA não gera ganhos dramáticos em F1-macro.

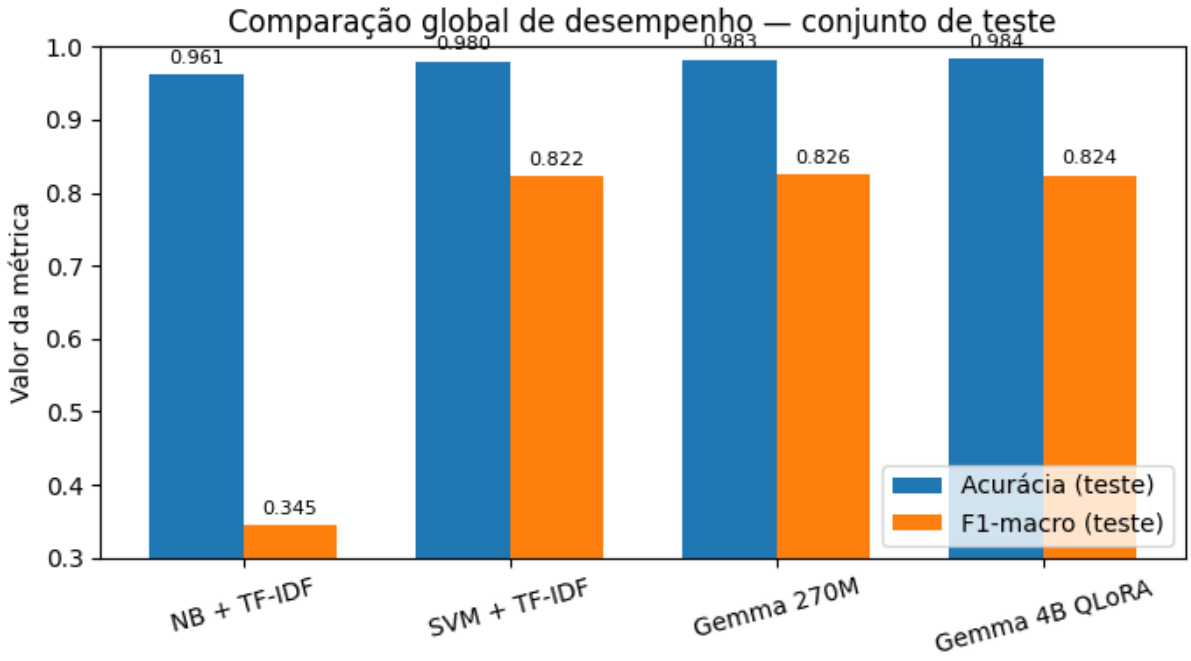
5.2.4 Síntese dos resultados de classificação

De forma sintética, os resultados de classificação indicam que:

- o Naive Bayes com TF–IDF, embora apresente acurácia alta, sofre de um colapso nas classes minoritárias e é descartado como baseline competitivo, dada a F1-macro muito baixa;
- o SVM linear com TF–IDF, com tratamento explícito do desbalanceamento via `class_weight="balanced"`, já atinge um desempenho elevado (F1-macro $\approx 0{,}82$), com boa performance em todas as classes, considerando a simplicidade do modelo e o baixo custo computacional;
- os modelos Gemma 3 270M-IT (full *fine-tuning*) e Gemma 3 4B-IT QLoRA oferecem ganhos modestos ou desempenho muito semelhante em F1-macro, com pequenas oscilações por classe, ao custo de um treinamento e inferência significativamente mais caros.

De forma agregada, a Figura 19 resume o desempenho global de todos os modelos avaliados. Observa-se que o Naive Bayes apresenta acurácia relativamente alta, mas F1-macro muito baixa, confirmando seu colapso nas classes minoritárias. Já o SVM com TF–IDF e os dois modelos Gemma mantêm F1-macro na faixa de 0,82–0,83, com ganhos apenas marginais em relação ao baseline linear.

Figura 19 – Comparação global de acurácia e F1-macro no conjunto de teste para todos os modelos avaliados.



Fonte: elaborado pelo autor.

Em termos de trade-offs, o SVM com TF–IDF surge como uma opção muito atrativa em cenários com infraestrutura limitada ou quando se deseja um classificador

simples, interpretável e barato para produção. Por outro lado, modelos Gemma 3 ajustados tornam-se mais interessantes quando se busca uma LLM capaz de acumular múltiplas funcionalidades (por exemplo, classificação, resumo, geração de texto e explicações) em um único modelo, ou quando se deseja explorar formas de explicabilidade *semétricas*, em que o mesmo modelo que decide também participa da geração de justificativas — temas que são explorados nas próximas seções ao analisar as arquiteturas híbridas de explicação.

5.3 Resultados das arquiteturas híbridas de explicação

Nesta seção são analisados os experimentos que combinam um modelo de decisão (SVM ou Gemma 3 270M-IT) com uma LLM explicadora (Gemma 3 4B-IT em modo *chat*), conforme descrito na Seção 4.5. A qualidade das explicações é avaliada com a métrica de *feature coverage* definida na Seção 2.8.4, que mede a proporção de *n-grams* relevantes (extraídos pelo SVM + TF–IDF) que aparecem explicitamente no texto gerado pela LLM.

5.3.1 SVM + LLM explicadora: qualidade e fidelidade lexical

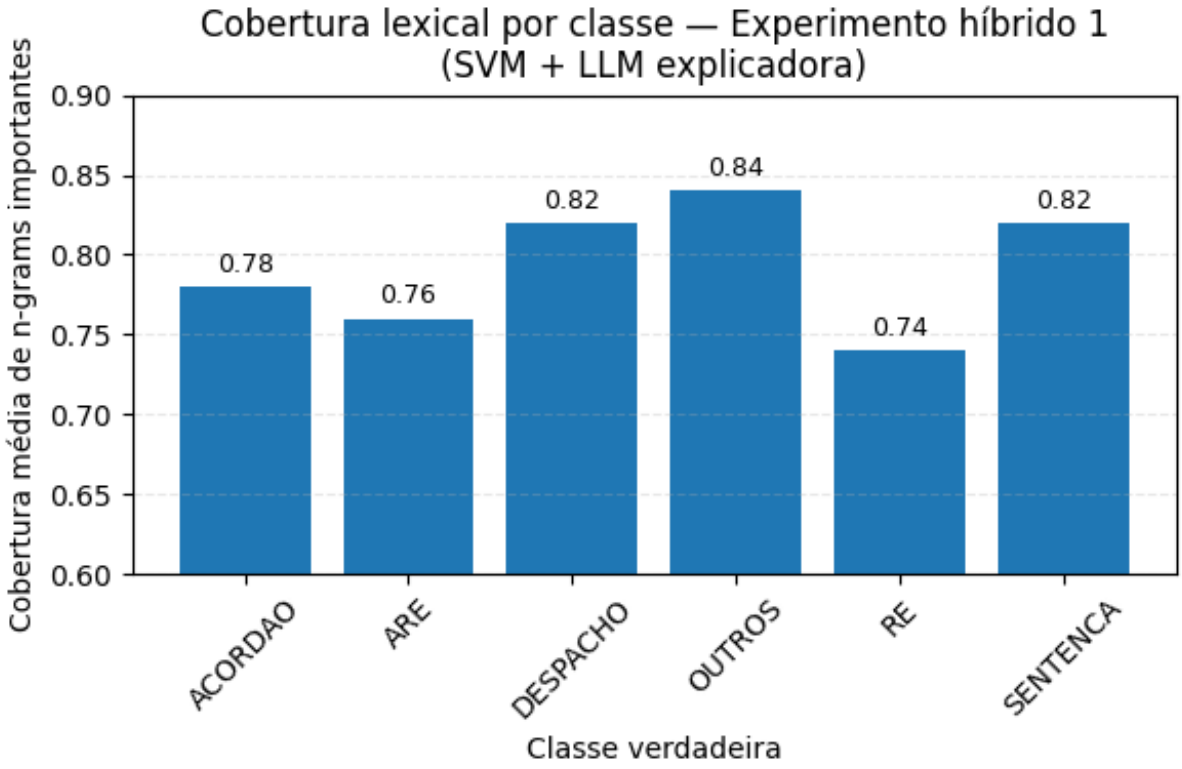
No primeiro experimento híbrido (Seção 4.5.3), o SVM com TF–IDF é utilizado como modelo de decisão e como fonte de evidências léxicas: para cada documento de teste, são extraídos os 10 *n-grams* com maior contribuição positiva para a classe predita, e essas expressões são fornecidas à LLM explicadora Gemma 3 4B-IT juntamente com o texto do documento e o rótulo previsto. A Tabela 10 resume a cobertura média global dessa métrica, calculada sobre uma amostra de $N_{\text{por classe}} = 5$ documentos de teste por classe.

Tabela 10 – Cobertura média global de *features* no experimento híbrido SVM + LLM.

Configuração	**Cobertura média global**
SVM (decisão) + Gemma 3 4B-IT (explicadora)	0,793

A cobertura média global do Experimento híbrido 1 é de aproximadamente 0,79, com valores por classe variando de cerca de 0,74 (`RE`) a 0,84 (`OUTROS`). A Figura 20 resume essa distribuição, mostrando que classes como `DESPACHO`, `SENTENÇA` e `OUTROS` tendem a apresentar maior reaproveitamento lexical das pistas fornecidas ao modelo, enquanto `ARE` e `RE` exibem coberturas ligeiramente inferiores. Uma cobertura média em torno de 0,79 indica que, em média, cerca de 80% dos *n-grams* destacados pelo SVM como mais relevantes para a decisão aparecem explicitamente na explicação produzida pela LLM. Ou seja, a LLM tende a reutilizar, em linguagem natural, a maior parte das pistas lexicais fornecidas, o que é um sinal de boa fidelidade *superficial* entre explicação e mecanismo de decisão.

Figura 20 – Cobertura lexical média por classe verdadeira no Experimento híbrido 1.



Fonte: elaborado pelo autor.

Considerando $|D| = 30$ documentos e $k = 10$ *n-grams* por documento ($N = 300$ pares no total), a cobertura média global de aproximadamente $\hat{p} = 0{,}79$ resulta em um intervalo de confiança de 95% em torno de [0,75, 0,84], indicando uma incerteza da ordem de poucos pontos percentuais.

A Tabela 11 detalha a cobertura média por classe verdadeira, conforme o arquivo gerado na Célula 7.

Tabela 11 – Cobertura média de *features* por classe verdadeira (SVM + LLM).

Classe verdadeira	Cobertura média
ACÓRDÃO	0,78
ARE	0,76
DESPACHO	0,82
OUTROS	0,84
RE	0,74
SENTENÇA	0,82
Global	0,79

As diferenças entre classes são relativamente pequenas, mas reveladoras. As classes

`OUTROS`, `DESPACHO` e `SENTENÇA` apresentam as maiores coberturas (entre 0,82 e 0,84), enquanto `ARE` e `RE` ficam ligeiramente abaixo da média global. Uma hipótese é que, em classes como `ACÓRDÃO`, `SENTENÇA` e `DESPACHO`, há padrões de redação mais *formulaicos* (por exemplo, expressões ligadas a relatoria, voto, dispositivo e despacho), de modo que os *n-grams* extraídos pelo SVM coincidem naturalmente com trechos que a LLM tende a reutilizar ao justificar a decisão.

Os exemplos qualitativos corroboram essa interpretação. Em um caso típico de documento rotulado como `OUTROS`, a LLM explicadora recebe, em modo *chat*, uma mensagem de sistema e uma mensagem de usuário estruturadas conforme a Célula 6 do código. De forma esquemática, o *input* enviado ao modelo pode ser resumido da seguinte forma:

> **System:** "Você é um assistente jurídico especializado em explicar decisões de um classificador automático de documentos do STF."
>
> **User:** "Sua tarefa é explicar, em linguagem natural e de forma didática, por que o modelo de classificação atribuiu o rótulo `OUTROS` para o documento abaixo.
>
> Documento (trecho relevante - confidencial): `"... texto do documento, truncado para até 3000 caracteres ..."`
>
> Classe predita pelo modelo: OUTROS
>
> Palavras e expressões mais importantes usadas pelo modelo para tomar essa decisão: "dje", "de julho", "autos origem", "portaria", "origem", "termo", "judiciária"
>
> Instruções: - Explique, em português jurídico claro, por que esse documento é compatível com a classe OUTROS. - Faça referência às palavras e expressões destacadas quando forem relevantes. - Não invente fatos que não estejam no texto; baseie-se no conteúdo apresentado. - A explicação deve ter de 1 a 3 parágrafos.
>
> Explanação:"

Dado esse *prompt*, a Gemma 3 4B-IT gera uma resposta em português jurídico. No exemplo em questão, a saída (após o marcador `"Explanação:"` e removidos detalhes irrelevantes) é do gênero:

> "A classificação do documento como `"OUTROS"` pelo modelo automático do STF reflete a sua natureza, que se distancia dos temas mais específicos e previamente definidos no treinamento do sistema. O texto em questão apresenta uma série de termos e expressões que, embora relevantes para o contexto jurídico, não se encaixam em categorias predefinidas, como processos judiciais com temas específicos.
>
> As palavras-chave destacadas, como "dje", "de julho", "autos origem", "portaria", "origem", "termo"e "judiciária", indicam que o documento é um termo de remessa, provavelmente relacionado a um processo judicial. No entanto, a presença de termos como "dje"(abreviação de Diário da Justiça Eletrônico) e a menção à "secretaria

> judiciária"sugerem que se trata de um documento interno de registro e lavratura, com foco na formalização de um ato processual, e não de um tema jurídico complexo a ser classificado em uma categoria específica.
>
> Em suma, o modelo, ao identificar a combinação de termos e expressões, concluiu que o documento não se enquadra em nenhuma das classes previamente definidas, sendo mais apropriadamente classificado como `"OUTROS"`."

Nesse exemplo, observa-se claramente que a LLM justifica o rótulo usando explicitamente a maior parte dos *n-grams* passados pelo SVM (*feature coverage* alta), ao mesmo tempo em que produz uma narrativa jurídica plausível sobre a natureza administrativa ou de registro do documento.

Por outro lado, mesmo em classes minoritárias como `ARE` ou `RE`, a LLM ainda recupera boa parte das *features* passadas, embora a cobertura seja um pouco menor. Nessas classes, o vocabulário é mais técnico e menos redundante, o que pode dificultar que todas as expressões críticas sejam reaproveitadas literalmente na explicação, apesar de o modelo ainda produzir justificativas juridicamente plausíveis.

5.3.2 Comparação SVM vs Gemma 270M como motor de decisão

No segundo experimento híbrido (Seção 4.5.4), são comparadas duas variantes da arquitetura de explicação:

(a) SVM + LLM explicadora (como na Seção 5.3.1);

(b) Gemma 3 270M-IT (classificador) + LLM explicadora.

Em ambos os casos, as evidências léxicas (*top* 10 *n-grams*) continuam sendo extraídas pelo SVM + TF–IDF, de modo que a única diferença é o rótulo fornecido à LLM (previsão do SVM ou do Gemma 3 270M-IT). A amostra contém $N_{\text{por classe}} = 15$ documentos de teste por classe, e a Tabela 12 apresenta as coberturas médias globais.

Tabela 12 – Cobertura média global de *features*: SVM + LLM vs Gemma 3 270M-IT + LLM.

Configuração	**Cobertura média global**
SVM (decisão) + LLM explicadora	0,760
Gemma 3 270M-IT (decisão) + LLM explicadora	0,773

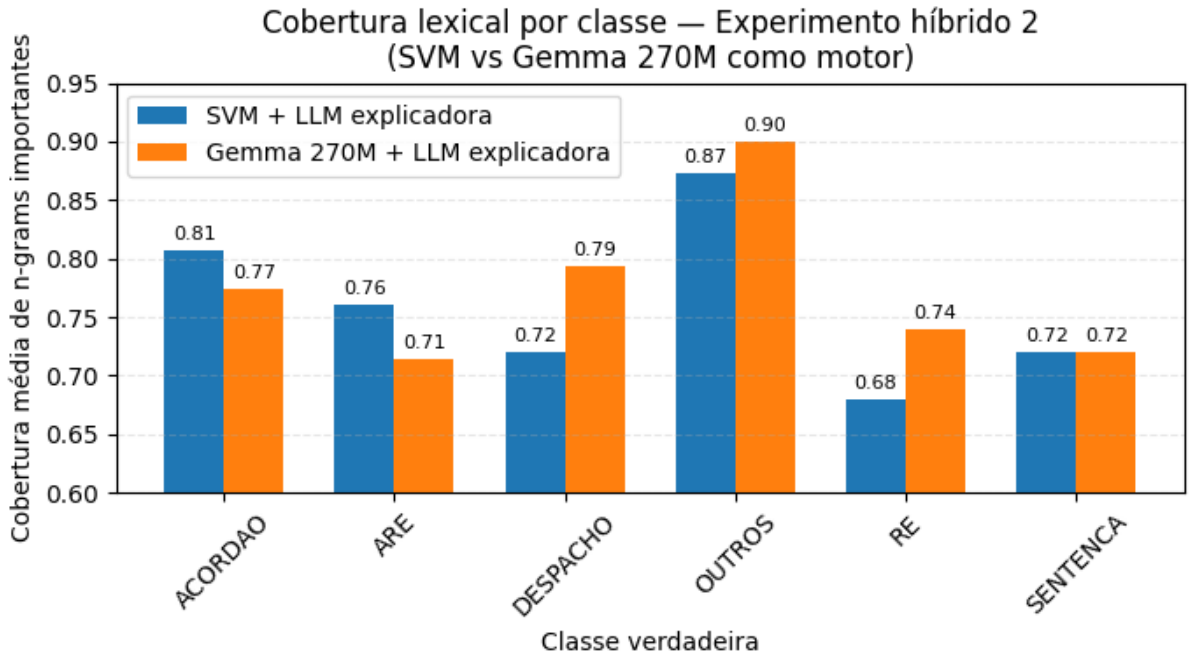
As coberturas globais são muito próximas, com uma diferença de apenas 0,013 a favor da variante que usa o Gemma 3 270M-IT como gerador do rótulo. A Tabela 13 detalha a cobertura média por classe verdadeira para ambas as configurações.

Tabela 13 – Cobertura média de *features* por classe verdadeira: SVM + LLM vs Gemma 3 270M-IT + LLM.

Classe verdadeira	SVM + LLM	Gemma 3 270M-IT + LLM
ACÓRDÃO	0,8067	0,7733
ARE	0,7600	0,7133
DESPACHO	0,7200	0,7933
OUTROS	0,8733	0,9000
RE	0,6800	0,7400
SENTENÇA	0,7200	0,7200
Global	0,7600	0,7730

As coberturas médias globais são muito próximas (cerca de 0,76 para SVM + LLM e 0,77 para Gemma 270M + LLM), indicando que, em nível agregado, o motor de decisão tem pouco impacto na fidelidade lexical. A Figura 21 detalha esses valores por classe verdadeira: observa-se que o cenário Gemma 270M + LLM apresenta coberturas um pouco maiores em `DESPACHO`, `OUTROS` e `RE`, enquanto o SVM + LLM é ligeiramente superior em `ACÓRDÃO` e `ARE`, com desempenho praticamente idêntico em `SENTENÇA`. Isso sugere que, do ponto de vista da *fidelidade lexical* da explicação, o fator dominante é o conjunto de *n-grams* fornecidos como evidência (sempre proveniente do SVM), e não o modelo utilizado para produzir o rótulo final.

Figura 21 – Cobertura lexical média por classe verdadeira no Experimento híbrido 2.



Fonte: elaborado pelo autor.

Em outras palavras, uma vez fixadas as pistas lexicais relevantes, a LLM explicadora consegue "verbalizar" essas evidências de maneira consistente, seja o rótulo de entrada oriundo do SVM ou do Gemma 3 270M-IT. Há casos pontuais em que SVM e Gemma 3 divergem no rótulo previsto para um mesmo documento, mas as explicações continuam cobrindo boa parte dos mesmos termos, o que se traduz em valores de *feature coverage* elevados em ambas as configurações. Nesses cenários, a explicação pode parecer plausível do ponto de vista textual, mesmo quando a classe prevista é incorreta, o que ilustra um limite importante dessa métrica. Por fim, neste segundo experimento híbrido, com $|D| = 90$ documentos e $k = 10$ *n-grams* por documento ($N = 900$), as coberturas globais observadas de $\hat{p}_{\text{SVM}} \approx 0{,}76$ e $\hat{p}_{\text{Gemma270M}} \approx 0{,}77$ levam a intervalos de confiança de 95% de aproximadamente [0,73, 0,79] e [0,75, 0,80], respectivamente. Esses intervalos se sobrepõem amplamente, sugerindo que a diferença de fidelidade lexical entre usar o SVM ou o Gemma 3 270M como motor de decisão é pequena e compatível com a própria margem de erro da estimativa.

5.3.3 Exemplos qualitativos de explicações

Para além das métricas agregadas, alguns exemplos qualitativos ajudam a entender melhor como as explicações são produzidas e quais são seus limites.

Em um caso típico de `OUTROS` corretamente classificado pelo SVM, a LLM explicadora recebe como *features* termos como `"dje"`, `"de julho"`, `"autos origem"`, `"portaria"` e `"judiciária"`. A explicação resultante destaca que se trata de um documento de natureza administrativa ou de registro (por exemplo, um termo de remessa ou ato interno da secretaria judiciária), argumentando que a presença de referências ao Diário de Justiça, à secretaria e a formalidades de registro afasta a caracterização como petição, acórdão ou sentença. Nesse tipo de exemplo, a correspondência entre as pistas fornecidas e o texto explicativo é clara, e a explicação tende a reforçar a confiança do usuário na predição do modelo.

Em contraste, há situações em que o modelo de decisão erra o rótulo, mas a explicação permanece textualmente convincente. Por exemplo, em alguns documentos de classe verdadeira `ACÓRDÃO` que o SVM confunde com `OUTROS`, a LLM constrói uma narrativa coerente com os termos destacados (por exemplo, menções a publicação ou movimentação processual), sem necessariamente perceber que o teor do documento contém elementos típicos de decisão colegiada. Do ponto de vista da *feature coverage*, a explicação ainda é considerada "fiel", pois recupera a maioria dos *n-grams* passados, mas isso não implica que o rótulo esteja correto.

Por fim, em classes minoritárias como `ARE` ou `DESPACHO`, as explicações tendem a enfatizar termos técnicos específicos (por exemplo, referências a recurso extraordinário, admissibilidade, decisão monocrática, etc.), frequentemente alinhados às *features* extraídas

pelo SVM. Esses casos ilustram o potencial da abordagem híbrida para produzir justificativas em linguagem jurídica relativamente natural, mesmo em cenários com menos dados, embora ainda sujeitas a eventuais omissões ou interpretações excessivamente genéricas.

5.4 Discussão geral e implicações

Os resultados obtidos permitem articular dois eixos principais de análise: desempenho de classificação (Seção 5.2) e qualidade das explicações (Seção 5.3). Do ponto de vista puramente preditivo, modelos clássicos baseados em TF–IDF e SVM já alcançam um desempenho muito elevado na tarefa de classificação de tipo de peça na base Victor, com F1-macro em torno de 0,82. Os modelos Gemma 3 ajustados (270M-IT com *full fine-tuning* e 4B-IT com QLoRA) oferecem ganhos modestos ou desempenho muito semelhante, ao custo de maior complexidade computacional.

Isso sugere que, para tarefas fortemente lexicais como a classificação de tipos documentais, heurísticas robustas de representação (TF–IDF com *n-grams*) combinadas com modelos lineares e tratamento explícito de desbalanceamento já são altamente competitivas. Por outro lado, LLMs ajustadas trazem uma vantagem qualitativa importante: a capacidade de serem reaproveitadas para múltiplas tarefas (classificação, sumarização, resposta a perguntas, etc.) e, sobretudo, de gerar explicações em linguagem natural a partir de prompts estruturados.

Nesse contexto, a arquitetura híbrida proposta — SVM como motor de decisão principal, aliado a uma LLM explicadora que recebe as evidências léxicas extraídas — se mostra um *meio termo* interessante. Ela preserva um classificador simples, barato, estável e de fácil implantação, ao mesmo tempo em que delega a uma LLM a tarefa de produzir justificativas em linguagem jurídica natural, com boa fidelidade lexical medida pela métrica de *feature coverage*. Os experimentos indicam que essa fidelidade permanece alta mesmo quando o motor de decisão é trocado por um modelo neural (Gemma 3 270M-IT), desde que as mesmas pistas lexicais sejam utilizadas, reforçando o papel central do componente SVM + TF–IDF na extração de evidências.

Por outro lado, a própria métrica de *feature coverage* tem limitações claras. Ela mede coincidência lexical literal entre *features* e explicação, sem capturar sinônimos, paráfrases ou aspectos semânticos mais profundos. Uma explicação pode parecer juridicamente plausível e ainda assim não mencionar explicitamente todas as *features* importantes; do mesmo modo, pode obter alta cobertura lexical mesmo quando o rótulo previsto está errado. Ainda assim, trata-se de um primeiro passo simples, objetivo e reprodutível para avaliar o grau de alinhamento entre o que o modelo usa para decidir (no nível de *n-grams*) e o que a LLM explicadora efetivamente expressa em linguagem natural.

Em síntese, os experimentos deste trabalho mostram que é possível combinar, de

forma relativamente simples, um motor de decisão eficiente (SVM ou LLM ajustada) com uma LLM explicadora para produzir classificações de documentos jurídicos com alto desempenho e explicações com boa fidelidade lexical. Essa combinação abre caminho para recomendações práticas sobre quando usar modelos clássicos, quando investir em LLMs ajustadas e como integrar mecanismos de explicabilidade em fluxos de trabalho reais de classificação de documentos jurídicos — questões que são retomadas e consolidadas no Capítulo de conclusões.

6 Conclusão

Este trabalho investigou a classificação de documentos jurídicos do Supremo Tribunal Federal (STF) utilizando o conjunto de dados Victor, com foco em seis tipos canônicos de peças (RE, ARE, SENTENÇA, ACÓRDÃO, DESPACHO e OUTROS) e em cenários de forte desbalanceamento de classes. Foram comparados modelos clássicos baseados em TF–IDF e classificadores lineares, modelos de linguagem Gemma 3 ajustados via *fine-tuning* e arquiteturas híbridas combinando um motor de decisão (SVM ou LLM) com uma LLM explicadora em português jurídico. A avaliação contemplou tanto o desempenho de classificação (acurácia e F1-macro) quanto a fidelidade lexical de explicações em linguagem natural (*feature coverage*).

Os resultados mostraram que o Naive Bayes com TF–IDF, apesar de atingir acurácias elevadas, colapsa nas classes minoritárias e produz F1-macro muito baixo, sendo inadequado como baseline competitivo. Em contraste, o classificador linear SVM com TF–IDF e `class_weight="balanced"` alcançou desempenho robusto e equilibrado entre as classes, com F1-macro em torno de 0,82 no conjunto de teste, demonstrando que modelos lineares bem ajustados continuam muito eficazes em tarefas de classificação de tipo de peça jurídica, nas quais padrões lexicais e de forma textual são altamente informativos.

Os modelos Gemma 3 270M-IT (full *fine-tuning*) e Gemma 3 4B-IT com QLoRA obtiveram acurácias ligeiramente superiores e F1-macro muito próximas às da SVM, com ganhos modestos e diferenças por classe relativamente pequenas. Esse comportamento indica que, para o problema estudado — classificação de tipo de peça no Victor *small* — grande parte da estrutura do sinal já é capturada por uma combinação simples de TF–IDF e modelo linear, e que o aumento de capacidade e custo computacional das LLMs não se traduz em ganhos dramáticos de desempenho global.

No eixo de explicabilidade, as arquiteturas híbridas propostas mostraram que é possível combinar um motor de decisão simples com uma LLM explicadora em português jurídico, mantendo boa fidelidade lexical às evidências usadas na decisão. O experimento híbrido com SVM + Gemma 3 4B-IT chat apresentou *feature coverage* média global em torno de 0,79, com coberturas consistentes entre classes e explicações que, em geral, reutilizam explicitamente os *n-grams* mais relevantes extraídos pelo modelo linear. Na comparação entre SVM e Gemma 3 270M-IT como motores de decisão, as coberturas médias permaneceram muito próximas ($\approx$0,76 vs. $\approx$0,77), sugerindo que, sob a métrica adotada, a fidelidade lexical das explicações depende mais das pistas lexicais fornecidas ao *prompt* do que do modelo específico que produziu o rótulo.

Como limitações, destacam-se o uso exclusivo do recorte Victor *small* (com distri-

buição altamente desbalanceada) e a adoção de uma métrica de fidelidade estritamente lexical, que não captura sinônimos, paráfrases ou aspectos mais profundos de compreensão semântica. Além disso, a avaliação das explicações foi feita apenas por métricas automáticas, sem estudos de usuário com profissionais do Direito. Ainda assim, os resultados apontam que a arquitetura híbrida baseada em SVM + LLM explicadora é uma solução pragmática: mantém um motor de decisão leve, estável e barato, ao mesmo tempo em que explora a flexibilidade das LLMs para produzir justificativas em linguagem jurídica natural.

6.1 Trabalhos futuros

Como desdobramentos naturais deste trabalho, destacam-se algumas frentes de pesquisa. A primeira é a ampliação do escopo experimental, avaliando os mesmos métodos em recortes maiores do Victor (por exemplo, *medium* ou completo), em outros tribunais ou com esquemas de rotulagem mais finos (subtipos de peças, etapas processuais, tarefas multilabel ou hierárquicas). Isso permitiria testar a robustez dos resultados e verificar se os achados sobre a competitividade da SVM com TF–IDF e o ganho limitado das LLMs se mantêm em cenários mais complexos.

Uma segunda direção é o refinamento das métricas de explicabilidade. A *feature coverage* poderia ser estendida para considerar variações morfossintáticas, sinônimos e expressões semanticamente equivalentes, por exemplo, usando representações em espaço vetorial ou medidas de similaridade lexical mais ricas. Avaliações com especialistas jurídicos também são fundamentais para confrontar métricas automáticas com julgamentos humanos, identificando em que medida as explicações são não apenas fiéis às evidências, mas também úteis e confiáveis na prática.

Por fim, abre-se espaço para investigar arquiteturas em que o modelo produza, de forma conjunta, o rótulo e uma explicação estruturada (por exemplo, sob a forma de *rationales* ou trechos destacados do texto), bem como para explorar o uso de LLMs especializadas em domínio jurídico brasileiro. Comparações sistemáticas entre modelos puramente neurais, modelos clássicos e abordagens híbridas em cenários reais de apoio à triagem e análise de processos podem consolidar recomendações práticas para o uso responsável de modelos explicáveis em classificação de documentos jurídicos no país.

Referências

ARAUJO, P. H. Luz de et al. Sequence-aware multimodal page classification of brazilian legal documents. *International Journal on Document Analysis and Recognition*, v. 26, n. 1, p. 33–49, 2023. Disponível em: <https://doi.org/10.1007/s10032-022-00406-7>. Citado 2 vezes nas páginas 28 e 29.

ARAUJO, P. H. Luz de et al. Victor: A dataset for brazilian legal documents classification. In: *Proceedings of the 12th International Conference on Language Resources and Evaluation (LREC 2020)*. Marseille, France: European Language Resources Association (ELRA), 2020. p. 1440–1447. Disponível em: <https://aclanthology.org/2020.lrec-1.181>. Citado 5 vezes nas páginas 15, 28, 29, 62 e 67.

ARIAI, F. et al. Natural language processing for the legal domain: A survey of tasks, datasets, models, and challenges. *ACM Computing Surveys*, 2024. Preprint. Disponível em: <https://arxiv.org/abs/2410.21306>. Citado 2 vezes nas páginas 61 e 63.

BENGIO, Y. et al. A neural probabilistic language model. *Journal of Machine Learning Research*, v. 3, p. 1137–1155, 2003. Citado na página 34.

BILAL, A.; EBERT, D.; LIN, B. LLMs for Explainable AI: A comprehensive survey. *arXiv preprint*, arXiv:2504.00125, 2025. Citado 2 vezes nas páginas 55 e 56.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. New York: Springer, 2006. Citado 6 vezes nas páginas 29, 30, 31, 38, 39 e 57.

BRAZ, F. A. et al. Document classification using a bi-lstm to unclog brazil's supreme court. *arXiv preprint arXiv:1811.11569*, 2018. Disponível em: <https://arxiv.org/abs/1811.11569>. Citado 2 vezes nas páginas 28 e 29.

BROWN, T. B.; MANN, B.; RYDER, N. e. a. Language models are few-shot learners. *Advances in Neural Information Processing Systems (NeurIPS)*, v. 33, p. 1877–1901, 2020. Citado 7 vezes nas páginas 40, 42, 43, 44, 47, 48 e 49.

CAMBRIA, E. et al. XAI meets LLMs: A survey of the relation between explainable ai and large language models. *arXiv preprint*, arXiv:2407.15248, 2024. Citado 2 vezes nas páginas 55 e 56.

CHALKIDIS, I.; ANDROUTSOPOULOS, I.; ALETRAS, N. Neural legal judgment prediction in english. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, 2019. p. 4317–4323. Citado na página 61.

CHALKIDIS, I. et al. Legal-bert: The muppets straight out of law school. *Findings of the Association for Computational Linguistics: EMNLP 2020*, p. 2898–2904, 2020. Disponível em: <https://aclanthology.org/2020.findings-emnlp.261>. Citado na página 61.

Conselho Nacional de Justiça. *Resolução nº 332, de 21 de agosto de 2020: estabelece diretrizes para o uso de inteligência artificial no Poder Judiciário*. Brasília: [s.n.], 2020. Conselho Nacional de Justiça. Norma que dispõe sobre princípios como direitos

fundamentais, não discriminação, publicidade, transparência e supervisão humana no uso de IA no Judiciário. Citado 2 vezes nas páginas 24 e 62.

Conselho Nacional de Justiça. *Justiça em Números 2024: ano-base 2023*. Brasília: Conselho Nacional de Justiça, 2024. Relatório estatístico do Poder Judiciário brasileiro. Citado na página 23.

CORTES, C.; VAPNIK, V. Support-vector networks. *Machine Learning*, v. 20, n. 3, p. 273–297, 1995. Citado 2 vezes nas páginas 38 e 39.

CUI, J. et al. A survey on legal judgment prediction: Datasets, metrics, models and challenges. *arXiv preprint*, 2022. Disponível em: <https://arxiv.org/abs/2204.04859>. Citado 2 vezes nas páginas 61 e 63.

DETTMERS, T. et al. QLoRA: Efficient finetuning of quantized LLMs. *arXiv preprint*, arXiv:2305.14314, 2023. Citado 2 vezes nas páginas 52 e 53.

DEVLIN, J. et al. Bert: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. [S.l.]: Association for Computational Linguistics, 2019. p. 4171–4186. Citado 7 vezes nas páginas 35, 41, 42, 43, 46, 47 e 49.

DIAS, S. A. d. J. et al. Inteligência artificial e redes de colaboração: o caso victor, ia do supremo tribunal federal. *Contemporânea – Revista de Ética e Filosofia Política*, v. 3, n. 7, p. 7608–7635, 2023. Acesso em: 9 dez. 2025. Disponível em: <https://ojs.revistacontemporanea.com/ojs/index.php/home/article/view/846>. Citado na página 28.

DOSHI-VELEZ, F.; KIM, B. Towards a rigorous science of interpretable machine learning. *arXiv preprint*, arXiv:1702.08608, 2017. Citado 3 vezes nas páginas 53, 56 e 62.

FENG, Y.; LI, C.; NG, V. Legal judgment prediction: A survey of the state of the art. In: *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI 2022)*. [S.l.: s.n.], 2022. p. 5532–5539. Citado na página 63.

FILHO, M. S. M.; JUNQUILHO, T. A. Projeto victor: perspectivas de aplicação da inteligência artificial ao direito. *Revista de Direitos e Garantias Fundamentais*, v. 19, n. 3, p. 219–238, 2018. Citado 2 vezes nas páginas 23 e 62.

Gemma Team. Gemma 3 technical report. *arXiv preprint*, arXiv:2503.19786, 2025. Google DeepMind. Citado 6 vezes nas páginas 47, 48, 70, 71, 72 e 76.

GOLDBERG, Y. *Neural Network Methods for Natural Language Processing*. [S.l.]: Morgan & Claypool, 2017. Citado na página 41.

GOMES, L. M. *Línguas Naturais e Máquinas Artificiais: aplicação de técnicas de mineração de texto para a classificação de sentenças judiciais brasileiras*. [S.l.], 2020. Disponível em: <https://repositorio.ipea.gov.br/handle/11058/10246>. Citado na página 61.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. Citado 3 vezes nas páginas 29, 30 e 31.

HE, H.; GARCIA, E. A. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, v. 21, n. 9, p. 1263–1284, 2009. Citado na página 31.

HOULSBY, N. et al. Parameter-efficient transfer learning for nlp. In: *Proceedings of the 36th International Conference on Machine Learning*. Long Beach, USA: PMLR, 2019. p. 2790–2799. Also available as arXiv:1902.00751. Citado na página 50.

HOWARD, J.; RUDER, S. Universal language model fine-tuning for text classification. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, 2018. p. 328–339. Citado na página 49.

HU, E. J. et al. Lora: Low-rank adaptation of large language models. In: *Proceedings of the 10th International Conference on Learning Representations (ICLR)*. [S.l.: s.n.], 2022. ArXiv:2106.09685. Citado 3 vezes nas páginas 50, 51 e 52.

JOACHIMS, T. Text categorization with support vector machines: Learning with many relevant features. In: *Proceedings of ECML-98, 10th European Conference on Machine Learning*. [S.l.]: Springer, 1998. p. 137–142. Citado 2 vezes nas páginas 36 e 39.

JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing*. [S.l.: s.n.], 2023. Manuscrito da 3. edição, disponível em: <https://web.stanford.edu/~jurafsky/slp3/>. Acesso em: 9 dez. 2025. Citado 11 vezes nas páginas 31, 32, 33, 35, 39, 41, 43, 45, 46, 48 e 62.

KUDO, T.; RICHARDSON, J. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Brussels, Belgium: Association for Computational Linguistics, 2018. p. 66–71. Citado na página 40.

LI, L.; ZHANG, Y.; CHEN, L. Generate neural template explanations for recommendation. In: *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM 2020)*. [S.l.]: Association for Computing Machinery, 2020. p. 755–764. Citado na página 56.

MANNING, C. D.; RAGHAVAN, P.; SCHÜTZE, H. *Introduction to Information Retrieval*. Cambridge: Cambridge University Press, 2008. Citado 7 vezes nas páginas 31, 32, 33, 34, 36, 37 e 57.

MARANHÃO, J. S. d. A.; FLORÊNCIO, J. A.; ALMADA, M. Inteligência artificial aplicada ao direito e o direito da inteligência artificial. *Suprema: Revista de Estudos Constitucionais*, v. 1, p. 154–180, 2021. Citado 3 vezes nas páginas 23, 24 e 62.

MARTINS, V. S.; SILVA, C. D. Text classification in law area: a systematic review. *Journal of Information and Data Management*, v. 13, n. 6, p. 594–609, 2022. Disponível em: <https://journals-sol.sbc.org.br/index.php/jidm/article/view/2547>. Citado 2 vezes nas páginas 23 e 61.

MASRI, Y. et al. Comparative analysis of bert and gpt for classifying legal documents. *Algorithms*, v. 18, n. 7, p. 420, 2025. Citado na página 63.

MCCALLUM, A.; NIGAM, K. A comparison of event models for naive bayes text classification. In: *AAAI-98 Workshop on Learning for Text Categorization*. Madison, WI: [s.n.], 1998. p. 41–48. Citado na página 37.

MIKOLOV, T. et al. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. Acesso em: 9 dez. 2025. Disponível em: <https://arxiv.org/abs/1301.3781>. Citado na página 34.

MOLNAR, C. *Interpretable Machine Learning*. 2. ed. [S.l.]: Lulu.com, 2022. Versão online disponível em interpretable-ml.net. Citado 3 vezes nas páginas 53, 54 e 62.

NIGAM, K. et al. Text classification from labeled and unlabeled documents using em. *Machine Learning*, v. 39, n. 2–3, p. 103–134, 2000. Citado na página 37.

OUYANG, L. et al. Training language models to follow instructions with human feedback. *arXiv preprint*, arXiv:2203.02155, 2022. Citado 2 vezes nas páginas 48 e 49.

RUDIN, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, v. 1, n. 5, p. 206–215, 2019. Citado 3 vezes nas páginas 53, 54 e 62.

SEBASTIANI, F. Machine learning in automated text categorization. *ACM Computing Surveys*, v. 34, n. 1, p. 1–47, 2002. Citado 3 vezes nas páginas 36, 39 e 55.

SENNRICH, R.; HADDOW, B.; BIRCH, A. Neural machine translation of rare words with subword units. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, 2016. p. 1715–1725. Citado na página 40.

SILVA, N. C. d. et al. Document type classification for brazil's supreme court using a convolutional neural network. In: *Tenth International Conference on Forensic Computer Science and Cyber Law (ICoFCS 2018)*. [s.n.], 2018. p. 7–11. Disponível em: <https://teodecampos.github.io/ViP/correiaDaSilva_etal_icofcs2018.pdf>. Citado 2 vezes nas páginas 28 e 29.

SOKOLOVA, M.; LAPALME, G. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, Elsevier, v. 45, n. 4, p. 427–437, 2009. Citado 3 vezes nas páginas 57, 58 e 59.

Tribunal de Justiça do Distrito Federal e dos Territórios. *Recurso Especial x Recurso Extraordinário*. 2023. Direito Fácil. Acesso em: 9 dez. 2025. Disponível em: <https://www.tjdft.jus.br/institucional/imprensa/campanhas-e-produtos/direito-facil/edicao-semanal/recurso-especial-x-recurso-extraordinario>. Citado 2 vezes nas páginas 27 e 28.

VALLE, V. C. L. L. Decisão judicial assistida por inteligência artificial e o sistema victor do supremo tribunal federal. *RINC – Revista de Informação Legislativa e Constitucional*, 2023. Disponível em: <https://www.scielo.br/j/rinc/a/YKZfQPLJqT7F3P445KkmwnC/>. Acesso em: 9 dez. 2025. Citado 2 vezes nas páginas 24 e 62.

VASWANI, A. et al. Attention is all you need. In: *Advances in Neural Information Processing Systems (NeurIPS)*. [S.l.: s.n.], 2017. p. 5998–6008. Citado 5 vezes nas páginas 39, 43, 44, 45 e 46.