UNIVERSIDADE FEDERAL DE GOIÁS / INSTITUTO DE INFORMÁTICA

Detecção de Concept Drift Temporal em Textos

Uma Análise para Identificação de Janelas Temporais de Retreinamento

Matheus Fares Costa Brakes

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

UNIVERSIDADE FEDERAL DE GOIÁS (UFG)
INSTITUTO DE INFORMÁTICA (INF)

MATHEUS FARES COSTA BRAKES

Detecção de Concept Drift Temporal em Textos

Uma Análise para Identificação de Janelas Temporais de Retreinamento

Goiânia
2025

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): MATHEUS FARES COSTA BRAKES

Título do trabalho: Detecção de Concept Drift Temporal em Textos
Uma Análise para Identificação de Janelas Temporais de Retreinamento

2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro.

Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Matheus Fares Costa Brakes**, **Discente**, em 21/07/2025, às 09:43, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Fernando Marques Federson**, **Professor do Magistério Superior**, em 05/08/2025, às 09:30, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5513817** e o código CRC **6F33AF1B**.

Referência: Processo nº 23070.037353/2025-01 SEI nº 5513817

MATHEUS FARES COSTA BRAKES

Detecção de Concept Drift Temporal em Textos

Uma Análise para Identificação de Janelas Temporais de Retreinamento

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.
Orientador: Prof. Dr. Fernando Marques Federson

Goiânia
2025

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

BRAKES, MATHEUS FARES COSTA
Detecção de Concept Drift Temporal em Textos [manuscrito] : Uma Análise para Identificação de Janelas Temporais de Retreinamento / MATHEUS FARES COSTA BRAKES. - 2025.
203 f.

Orientador: Prof. Dr. Fernando Marques Federson.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Instituto de Informática (INF), Inteligência Artificial, Goiânia, 2025.

1. inteligência artificial. 2. modelos grandes de linguagem. 3. concept drift. I. Federson, Fernando Marques , orient. II. Título.

CDU 004

MATHEUS FARES COSTA BRAKES

Detecção de Concept Drift Temporal em Textos

Uma Análise para Identificação de Janelas Temporais de Retreinamento

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.

Data da Aprovação: 04 de julho de 2025.

DocuSigned by:
Fernando Marques Federson
0D8426531FE3495...

Prof. Dr. Fernando Marques Federson
Orientador (INF-UFG)

Signed by:
Aldo Andre Díaz Salazar
504EE9DE76EC4D7...

Prof. Dr. Aldo André Díaz Salazar
Coordenador de TCC do BIA (INF-UFG)

DocuSigned by:
Anderson da Silva Soares
2C1B74D46D6F4FA...

Prof. Dr. Anderson da Silva Soares
Coordenador do BIA (INF-UFG)

Assinado por:
Savio Salvarino Teles de Oliveira
108009F44864405...

Prof. Dr. Sávio Salvarino Teles de Oliveira
(INF-UFG)

MATHEUS FARES COSTA BRAKES

Detecção de Concept Drift Temporal em Textos

Uma Análise para Identificação de Janelas Temporais de Retreinamento

RESUMO

Este Relatório de Conclusão de Curso tem como objetivo reunir os resultados da minha jornada para me tornar um especialista em **Detecção de Concept Drift**. Uma ilustração e sua narrativa descrevem os períodos de trabalho. Os Apêndices contêm os Termos de Aceite de Entrega e os resultados obtidos durante cada período de trabalho.

Palavras-chave: inteligência artificial, modelos grandes de linguagem, concept drift.

ABSTRACT

This Course Completion Report aims to bring together the results of my journey to become an expert in **Concept Drift Detection**. An illustration and its narrative describe the work periods. The Appendices contain the Delivery Acceptance Terms and the results obtained during each work period.

Keywords: artificial intelligence, large language models, concept drift.

Goiânia

2025

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Minha Jornada

Matheus Fares

Especialista em: Detecção de Concept Drift

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

MINHA JORNADA

Nome: **Matheus Fares Costa Brakes**

Especialidade: **Detecção de Concept Drift**

Objetivo deste documento

Durante o processo da disciplina Residência em IA[1], foram gerados diversos resultados na construção da minha especialização. A cada semana, um conjunto de resultados foi formalizado por um Termo de Aceite de Entrega e avaliado por uma banca, considerando o planejado e o realizado para o período. Este documento tem como objetivo descrever esses resultados obtidos, fazendo referência aos Termos de Aceite de Entrega e seus documentos associados.

Minha Jornada

O início da disciplina Residência em Inteligência Artificial, durante as **Semanas 1, 2 e 3**, foi dedicado a encontrar o tema central que seria trabalhado ao longo das Semanas seguintes. Comecei com uma pesquisa de possíveis temas de meu interesse, na qual analisei tópicos de Congressos ligados à área da Inteligência Artificial, especialmente as ligadas ao CSCE[1], para ter uma visão mais ampla das possíveis áreas para o tema. Para organizar as ideias, realizei um agrupamento temático dos assuntos que mais me chamaram a atenção, o que me permitiu visualizar melhor as conexões e identificar três possíveis caminhos estratégicos para a minha jornada. Após uma análise e estudos mais aprofundados da relevância de cada caminho, segundo alguns artigos sobre os temas e outras pesquisas, o processo culminou na escolha efetiva da área de **Modelagem Preditiva Adaptativa**. Este tema se concentra na construção de sistemas preditivos robustos e

[1] *American Council on Science and Education* (https://american-cse.org/index.html/).

autoajustáveis, capazes de operar em ambientes que mudam continuamente. Os resultados e a documentação detalhada dessas Semanas podem ser observados no **Apêndice 1**.

Dando continuidade à jornada, na **Semana 4**, aprofundei os estudos na área de **Modelagem Preditiva Adaptativa** e percebi que, embora a direção estivesse correta, o termo era muito amplo e não capturava com precisão o que eu procurava, além de ser usado com outros nomes em certas referências bibliográficas. A partir desse pressuposto, concentrei-me em definir um tema menos amplo e, ao pesquisar, fiz a escolha pelos fenômenos de ***Concept Drift*** e ***Concept Evolution***, termos que já tinham sido levantados anteriormente como de interesse. Para consolidar essa nova direção, aprofundei nos termos, e elaborei uma documentação, que pode ser vista dentro do **Apêndice 2**. A documentação apresenta o que foi estudado e traz as definições e origens do ***Concept Drift***, detalhando suas diferentes fontes (virtual , real e mista); tipos de manifestação, como o abrupto, incremental, gradual e recorrente; usabilidade em aplicações reais; entre outras informações. Entre os itens estudado teve como destaque um artigo científico na forma de *review*[2], que me permitiu entender a história e aplicação do termo e a sua relevância para certos desenvolvimentos dentro do ramo da IA. Além disso, confirmou que é uma área de pesquisa com fundamentos sólidos, que apresenta uma complexidade e relevância prática que justificam o esforço para desenvolver soluções. Entendi que fazia sentido para mim me aprofundar e também desenvolver algo nesta área de pesquisa até o final da minha jornada na Residência.

Com o entendimento dos conceitos de ***Concept Drift*** e ***Concept Evolution***, o trabalho das **Semanas 5 e 6** foi direcionado para a **Detecção de Drift**, ou seja, a detecção da piora de modelos de aprendizado de máquina ao longo do tempo. Tomei essa decisão para delimitar o escopo, já que com o conhecimento adquirido, entendi que temos tanto a parte de perceber as mudanças relevantes na base de dados que afetam o modelo, quanto corrigir o modelo por conta destas mudanças[3]. Primeiramente, explorei as quatro grandes categorias de métodos de detecção: (i) estatísticos, (ii) monitoramento por janelas, (iii)

[2] Lu, J., et al. *"Learning under Concept Drift: A Review"* arXiv preprint, 2020.
[3] Lembrando que é preciso primeiro identificar o problema para depois corrigi-lo!

técnicas de *ensembles* e (iv) métodos baseados em representação. A partir deste conhecimento, fiz um levantamento de frameworks open-source que suportam a deteção. Selecionei os frameworks que demonstravam relevância na comunidade, manutenção ativa, funcionalidades, entre outros critérios, para um estudo comparativo. Com este estudo comparativo, entendi os casos de uso ideais para cada método/ferramenta desenvolvidas até hoje. Entre eles estão: o **River** que se destaca para processamento em streaming, utilizando métodos estatísticos como o DDM[4] e o ADWIN[5] para monitorar o erro do modelo em tempo real; o **NannyML** que se mostra ideal para cenários sem rótulos recentes, com capacidade para estimar o impacto do drift no desempenho; o **Evidently AI** que se sobressai por gerar relatórios visuais detalhados de sistemas de produção; e o **Alibi Detect** que se mostrar a escolha certa para dados complexos e *embeddings* de alta dimensão. Ao final deste período, eu tinha um mapa do ecossistema de ferramentas. Isto foi importante para as decisões das Semanas seguintes, definindo o caminho para a implementação que foi desenvolvida. Este material está detalhado no **Apêndice 3**

As pesquisas das Semanas anteriores confirmaram para mim a viabilidade da implementação de um detector de ***Concept Drift***. Na **Semana 7**, o objetivo foi selecionar o domínio de um problema real para o projeto. Primeiro, reorganizei a taxonomia anterior, cobrindo os tipos de *drift* e seus gatilhos (mudanças ambientais ou comportamentais). Em seguida, analisei casos de uso em domínios diversos, como previsão de consumo energético industrial e de crimes em contextos urbanos. A terceira e decisiva frente foi um mergulho em aplicações de **Processamento de Linguagem Natural (NLP)**, onde investiguei fenômenos como: a evolução do significado de palavras ao longo da história; a ressignificação de termos existentes e a obsolescência de fatos em **Large Language Models (LLMs)**. Essa dinâmica da linguagem foi bem ilustrada por um artigo científico[6], que analisou a linguagem em perspectiva histórica, evidenciando como mudanças temporais podem alterar o significado das palavras. A partir dessa análise, ficou claro que uma área

[4] *Differential Distribution Method* (ou Modelo de Desconto de Dividendos - MDD).
[5] *ADaptive WINdowing*.
[6] Li, R., Tian, P., & Wang, S. *Study concept drift in 150-year English literature*, 1st Workshop on AI + Informetrics -AII2021, 2021.

promissora e alinhada aos meus interesses seria a detecção voltada para NLP. Os documentos gerados nesta Semana estão reunidos no **Apêndice 4**.

O foco das **Semanas 8 e 9** foi a replicação de um experimento para combater o *drift* temporal em textos jurídicos. O ponto de partida foi a seleção do artigo científico sobre o *"ChronosLex"*[7], que propõe um paradigma de treinamento incremental para lidar com a evolução dos conceitos legais com o tempo. Embora o artigo declare que tratar os dados como um "bloco homogêneo" único é subótimo, sua abordagem foca em propor um *fine-tuning* a intervalos fixos de tempo, sem se aprofundar de fato na **Detecção do Drift** ao longo do tempo. Minha ideia inicial era desenvolver um pipeline focado na detecção para aprimorar essa proposta do artigo, mas ficou claro que primeiro eu precisaria replicar o "mecanismo central" do artigo antes de propor melhorias. A partir disso, comecei a desenvolver um código para reproduzir os experimentos presentes no artigo, implementando com sucesso a abordagem apresentada e os *baselines* descritos no artigo. Essa etapa foi crucial para confirmar a viabilidade técnica e computacional de reproduzir os experimentos propostos. O código desenvolvido ao longo dessas semanas continuou em evolução, e sua versão final será detalhada em um Apêndice posterior. As informações e decisões deste período estão apresentadas no **Apêndice 5**.

As **Semanas 10 e 11** marcaram a fase de refinamento e finalização dos experimentos. Realizei a limpeza e a reorganização do código, corrigindo parâmetros e alguns erros de lógica. Para simplificar a implementação, a replicação dos experimentos foi reduzida para focar em apenas dois *datasets* utilizados no artigo de base, o *EURLEX small* e o *UKLEX small*. Adicionei alguns métodos faltantes, sendo os três métodos de aprendizado contínuo com melhores resultados para esses *datasets*: *Experience Replay* (ER), *Elastic Weight* Consolidation (EWC) e *LoRA*. Após a replicação dos experimentos do artigo, com mudanças pontuais, o foco voltou a ser a **Detecção do Drift** para identificar janelas de tempo "ideais" para um retreinamento. Realizei uma análise de *drift* temporal, comparando as

[7] Chalkidis, I., et al. *ChronosLex: Time-aware Incremental Training for Temporal Generalization of Legal Classification Tasks*. arXiv:2405.14211 (2024).

representações dos dados ano a ano com as métricas JSD[8] e MMD[9]. Ao aplicar esses métodos de detecção usando *embeddings*, foi observado o impacto de eventos do mundo "real", como o Brexit[10], na distribuição estatística da representação dos dados, por exemplo. Os resultados permitiram pensar em utilizar essa descrição e acompanhamento dos dados para reorganizar o *fine-tuning* incremental e propor o treinamento para tempos específicos. O *pipeline* completo de treinamento e detecção está funcional e pode ser encontrado no **Apêndice 6**.

Ao final desta jornada, é gratificante registrar a influência positiva que o processo de construção desta pesquisa me trouxe. Acredito que a maior transformação foi pessoal: o projeto me impulsionou a sair de um foco puramente prático para um interesse pelo estudo aprofundado de conceitos, que foram essenciais para construir uma base sólida de conhecimento em uma área de especialização. Tive influência positiva não só no final, como em todo o curso, que me incentivou a continuar na área, mesmo tendo previamente muito pouca familiaridade com IA. A existência de diversos professores solícitos, além do próprio processo vivido, me proporcionou várias experiências e oportunidades que me agregaram como profissional e que vejo que nem todos conseguem experienciar durante a faculdade.

[8] *Jensen-Shannon Divergence*.
[9] *Maximum Mean Discrepancy.*
[10] Brexit é o processo de saída do Reino Unido da União Europeia iniciado em 2017.

APÊNDICE 1

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 26 de mar. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

MATHEUS FARES COSTA BRAKES

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nesta entrega, realizei uma pesquisa abrangente seguindo os tópicos e áreas apresentadas nas conferências relacionadas à IA. Inicialmente, fiz uma análise superficial dos tópicos sugeridos e, em seguida, realizei uma leitura de artigos dos temas que despertaram maior interesse, a fim de tomar uma decisão mais assertiva quanto à área a ser aprofundada. Todo o processo de análise e a seleção dos tópicos estão documentados neste material: Áreas de interrese

Mas em suma a análise inicial e a leitura de artigos permitiram destacar os seguintes temas como os de maior interesse:

- Autonomous Learning Systems
- Concept Drift and Evolution
- Self-Adaptive and Self Organizing Systems (Principalmente)
- Application of Graph Theoretic Approaches in Dimensionality Reduction and Machine Intelligence
- Information and Knowledge Retrieval and Searching Algorithms
- Big Data Fusion and Information Retrieval
- Extraction of Latent Semantics from Big Data
- Knowledge Representation
- Reasoning Strategies
- Distributed AI Algorithms and Techniques
- Intelligent Information Fusion

- Multivariate Analysis
- Relational Learning Models
- Meta Learning

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima entrega, pretendo selecionar e delimitar uma área específica para aprofundamento, combinando os feedbacks recebidos dos professores com as áreas de interesse previamente levantadas. O objetivo é definir um escopo claro e iniciar um estudo aprofundado, avaliando as diversas abordagens e soluções que podem ser exploradas dentro dessa área.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

[Documento Áreas de Interesse citado no Termo de Aceite de Entrega de 26 de março]

Levantamento de Áreas de Interesse e Seleção para Especialização

Metodologia:
Inicialmente, listei diversas áreas que pareciam promissoras ao analisar os tópicos das conferências. Em seguida, realizei uma leitura superficial dos artigos relacionados para avaliar meu nível de interesse. Para os tópicos que despertaram maior curiosidade e que desejo me aprofundar, adicionei um sinal de “+”.

---

1. Áreas Levantadas Inicialmente

A. Conferências ACC

(Ver mais em: american-cse.org/csce2024/conferences-ACC)

1. **Improving Cognition in Machine Learning Systems**
    - Artigo: https://arxiv.org/abs/2403.07078
2. **Autonomous Learning Systems +**
    - Artigo: https://medium.com/@fhuqtheta/automl-a-systematic-review-on-automated-machine-learning-designing-machine-learning-systems-and-2cfef18642a2
3. **Concept Drift and Evolution +**
    - Artigo: https://www.researchgate.net/publication/382824207_Research_on_concept_drift_algorithm_based_on_evolutionary_computation

4. **Hebbian Learning Systems**
    - Artigo: https://arxiv.org/abs/2501.02402
5. **Self-Adaptive and Self Organizing Systems** +
    - Artigo: https://arxiv.org/pdf/2012.12600
6. **Application of Graph Theoretic Approaches in Dimensionality Reduction and Machine Intelligence** +
    - Artigo: https://arxiv.org/html/2412.06555v1
7. **Information and Knowledge Retrieval and Searching Algorithms** +
    - Artigo: https://arxiv.org/pdf/2409.05882
8. **Big Data Fusion and Information Retrieval** +
    - Artigo: https://www.researchgate.net/publication/268724557_Data_Fusion_in_Information_Retrieval
9. **Extraction of Latent Semantics from Big Data** +
    - Artigo: https://journal.esrgroups.org/jes/article/view/7584?utm_source=chatgpt.com

---

B. Conferências ICAI

(Ver mais em: american-cse.org/csce2024/conferences-ICAI)

1. **Natural Language Processing**
    - Artigo: https://arxiv.org/pdf/2405.12819

2. **Software Tools for AI**
    - Artigo: https://arxiv.org/pdf/2304.08354
3. **Automated Problem Solving**
    - Artigo: https://openreview.net/pdf?id=4XzGkm1jK0
4. **Knowledge Representation +**
    - Artigo: https://arxiv.org/html/2501.05435v1
5. **Knowledge Networks and Management**
    - Artigo: https://www.researchgate.net/publication/381656013_Analysis_of_Knowledge_Management_Practices_for_Knowledge_Intensive_Organizations_Self-Management
6. **Intelligent Agents**
    - Artigo: https://arxiv.org/pdf/2403.00833v1
7. **Reasoning Strategies +**
    - Artigo: https://synthesis.ai/2025/02/25/large-reasoning-models-how-o1-replications-turned-into-real-competition/
8. **Distributed AI Algorithms and Techniques +**
    - Artigo: https://medium.com/@parserdigital/rethinking-llms-a-modular-distributed-approach-to-ai-ee13e459d481
9. **Intelligent Information Fusion +**
    - Artigo: https://www.sciencedirect.com/science/article/pii/S2666675825000177?ref=pd

f_download&fr=RR-2&rr=92677bc6dc12df45

10. **Evaluation of AI Tools**
    - Artigo: https://www.confident-ai.com/blog/llm-agent-evaluation-complete-guide

---

C. Fundamentos Teóricos e Modelos de Aprendizagem

1. **Multivariate Analysis** +
    - Artigo: https://medium.com/gen-ai-adventures/multivariate-analysis-relationships-between-multiple-variables-d959c38bc380
2. **Relational Learning Models** +
    - Artigo: https://arxiv.org/pdf/2406.11249

D. Técnicas Avançadas e Aprendizado Adaptativo

1. **Meta Learning** +
    - Artigo: https://www.nature.com/articles/s41598-024-71125-8
2. **Multi-criteria Reinforcement Learning**
    - Artigo: https://arxiv.org/pdf/2410.11221v1

---

2. Áreas Selecionadas para Especialização

Após a leitura superficial dos artigos, os seguintes tópicos (marcados com “+”) se destacaram como áreas de interesse para um aprofundamento e eventual especialização:

- Autonomous Learning Systems
- Concept Drift and Evolution
- Self-Adaptive and Self Organizing Systems
- Application of Graph Theoretic Approaches in Dimensionality Reduction and Machine Intelligence
- Information and Knowledge Retrieval and Searching Algorithms
- Big Data Fusion and Information Retrieval
- Extraction of Latent Semantics from Big Data
- Knowledge Representation
- Reasoning Strategies
- Distributed AI Algorithms and Techniques
- Intelligent Information Fusion
- Multivariate Analysis
- Relational Learning Models
- Meta Learning

---

3. Interesses e Experiência

Possuo um forte interesse em Processamento de Linguagem Natural (NLP) e Machine Learning, com ênfase na análise e predição de dados.

No que diz respeito à minha experiência, tenho maior familiaridade em sistemas de busca inteligentes e na criação de arquiteturas multi-agente. Tenho atuado na implementação de soluções que integram o uso de Text-to-Sql em chatbots, na extração de informações a partir de documentos, na realização de pesquisas automatizadas na internet usando de LLMs, entre outras aplicações envolvendo sistemas multiagentes.

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 2 de abr. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

MATHEUS FARES COSTA BRAKES

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Analisei as relações entre os tópicos que levantei anteriormente e os agrupei em 5 grandes áreas temáticas, sendo elas:

Continual Learning e Lifelong Learning

Representation Learning e Modelagem Semântica Latente

Knowledge Representation and Reasoning (KRR)

Information Retrieval e Semantic Search

Distributed and Federated Intelligence

A partir disso, gerei um mapa visual mostrando as conexões entre esses tópicos de interesse, o que me permitiu identificar claramente as relações existentes. Com base nessas áreas, filtrei três possíveis caminhos estratégicos para atuação ou pesquisa:

Adaptive Reasoning Agents

Adaptive Predictive Intelligence

Federated Multi-Agent Intelligence

Toda essa análise e os detalhes sobre os caminhos identificados estão documentados no seguinte link: Agrupamento Temático dos Tópicos de Interesse.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Caso esteja tudo certo com o que foi produzido durante esta semana e a área definida por mim seja

confirmada como adequada para ser seguida, irei me aprofundar nela para detalhar melhor uma proposta concreta de atuação ou pesquisa. Caso contrário, continuarei estudando para avaliar melhor os caminhos possíveis e definir qual será minha área específica de trabalho.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

[Documento Agrupamento Temático dos Tópicos de Interesse citado no Termo de Aceite de Entrega de 2 de abril]

Agrupamento Temático dos Tópicos de Interesse

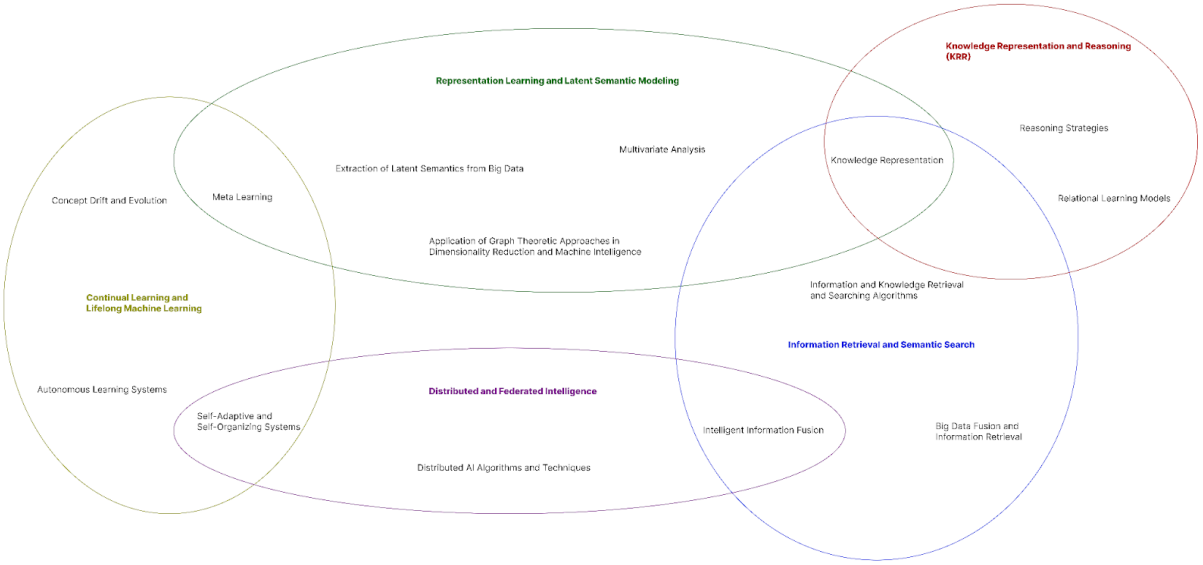
A partir de uma pesquisa exploratória sobre temas de interesse pessoal, foi realizada uma organização dos tópicos em grupos conceitualmente relacionados. O objetivo principal dessa estruturação foi visualizar melhor as conexões entre os assuntos e identificar áreas potenciais para atuação e aprofundamento, seguindo as orientações definidas na última etapa do projeto.

Áreas Temáticas Principais

Os temas foram organizados em cinco grandes áreas interligadas:

1. **Continual Learning e Lifelong Learning**
   Engloba tópicos como Autonomous Learning Systems, Concept Drift and Evolution e Self-Adaptive and Self-Organizing Systems, destacando sistemas capazes de aprendizado contínuo e adaptação a mudanças ao longo do tempo.
2. **Representation Learning e Modelagem Semântica Latente**
   Inclui Meta Learning, Multivariate Analysis, Graph Theoretic Approaches e Latent Semantics, abordando técnicas para representar e extrair significados ocultos em grandes volumes de dados.
3. **Knowledge Representation and Reasoning (KRR)**
   Reúne Knowledge Representation, Relational Learning Models e Reasoning Strategies, com foco na representação simbólica e na inferência lógica aplicada à Inteligência Artificial.
4. **Information Retrieval e Semantic Search**
   Abrange Information and Knowledge Retrieval, Big Data Fusion e Intelligent Information Fusion, voltando-se para recuperação eficiente de informações e integração de fontes heterogêneas.
5. **Distributed and Federated Intelligence**
   Agrupa Distributed AI Algorithms e Intelligent Fusion, com ênfase em abordagens descentralizadas e colaboração entre sistemas inteligentes distribuídos.

Link do mapa visual das conexões entre os tópicos



Caminhos Estratégicos para Atuação e Pesquisa

A partir da estruturação dos temas de interesse e suas interconexões, foi possível identificar três caminhos estratégicos para atuação ou pesquisa, cada um combinando diferentes áreas do mapa mental

---

Caminho 1: Adaptive Reasoning Agents

Desenvolvimento de agentes autônomos capazes de raciocinar, aprender continuamente e adaptar-se ao contexto utilizando múltiplas fontes de informação.

Áreas Envolvidas:

- Continual Learning e Lifelong Learning
- Knowledge Representation and Reasoning (KRR)
- Representation Learning e Modelagem Semântica Latente
- Information Retrieval e Semantic Search

Tópicos Envolvidos:

- Autonomous Learning Systems

- Self-Adaptive and Self-Organizing Systems
- Knowledge Representation
- Reasoning Strategies
- Meta Learning
- Latent Semantics
- Information and Knowledge Retrieval

Caminho 2: Adaptive Predictive Intelligence

Construção de sistemas preditivos robustos e auto ajustáveis, capazes de operar eficientemente em ambientes reais, aprendendo continuamente com múltiplas fontes de dados.

Áreas Envolvidas:

- Continual Learning e Lifelong Learning
- Representation Learning e Modelagem Semântica Latente
- Information Retrieval e Semantic Search
- Distributed and Federated Intelligence (opcional)

Tópicos Envolvidos:

- Concept Drift and Evolution
- Multivariate Analysis
- Graph Theoretic Approaches
- Big Data Fusion
- Intelligent Information Fusion

Caminho 3: Federated Multi-Agent Intelligence

Desenvolvimento de sistemas inteligentes distribuídos, com aprendizado local e colaboração descentralizada, garantindo autonomia operacional e privacidade.

Áreas Envolvidas:

- Distributed and Federated Intelligence

- Continual Learning e Lifelong Learning
- Information Retrieval e Semantic Search
- Knowledge Representation and Reasoning (KRR)

Tópicos Envolvidos:

- Distributed AI Algorithms
- Intelligent Fusion
- Self-Adaptive and Self-Organizing Systems
- Relational Learning Models
- Information and Knowledge Retrieval

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 9 de abr. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

MATHEUS FARES COSTA BRAKES

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Essa semana, realizei diversos estudos partindo do tema inicialmente levantado e constatei que a nomenclatura correta não seria Inteligência Preditiva Adaptativa, devido ao seu uso limitado na literatura acadêmica e à sua amplitude, que acaba diluindo o foco da pesquisa. Levantei outros termos, como Modelagem Preditiva Adaptativa, mas observei que esse também não possui consolidação clara, sendo empregado de maneira diversa em áreas como a engenharia de controle.

Seguindo a orientação de partir de uma abordagem macro e reduzir o escopo, optei por direcionar meus estudos para o tema Predictive Modeling. Aqui está o link com um pouco mais de informações sobre essa escolha : Nomenclatura

Após isso, estudei o tema Predictive Modeling, e montei esse documento levantando o que eu encontrei como sua origem, definições, entre outras informações : Predictive Modeling

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Nas próxima entrega, pretendo estreitar o foco dentro do escopo da Modelagem Preditiva Adaptativa. A ideia é identificar e aprofundar uma das ramificações desse campo, por exemplo, explorando técnicas como Online Learning e Continual Learning ou mesmo abordagens de controle adaptativo em cenários preditivos.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

[Documento Nomenclatura citado no Termo de Aceite de Entrega de 9 de abril]

Nomenclatura do tema a ser estudado

Durante o processo de definição do nome para minha área de estudo, inicialmente considerei o termo Inteligência Preditiva Adaptativa, mas logo percebi que ele possui uso limitado na literatura acadêmica e abrange um escopo demasiadamente amplo, diluindo o foco da pesquisa.

Em busca de uma designação que refletisse um modelo preditivo com capacidade adaptativa no contexto da IA, encontrei a expressão Modelagem Preditiva Adaptativa. Contudo, enfrentei dificuldades para identificar um termo específico e consolidado que capturasse essa abordagem. O uso dessa terminologia se restringe, na maioria das vezes, a contextos muito específicos – como na engenharia de controle, onde o controle preditivo adaptativo descreve sistemas dinâmicos ajustados em tempo real – o que não se alinha com a abrangência desejada para o meu estudo.

Diante desse cenário e considerando os feedbacks dos professores, optei por partir do estudo de Predictive Modeling. Essa terminologia, amplamente reconhecida na literatura, reúne os fundamentos teóricos e práticos essenciais para a construção, validação e aprimoramento de modelos preditivos, proporcionando um referencial claro e consistente para a pesquisa.

[Documento Predictive Modeling citado no Termo de Aceite de Entrega de 9 de abril]

Relatório: Predictive Modeling - Fundamentos, Aplicações e Ramificações

Introdução

O Predictive Modeling ou Modelagem Preditiva consiste na utilização de técnicas estatísticas, aprendizado de máquina e mineração de dados para prever resultados futuros baseados em dados históricos e atuais. Esse campo combina elementos de estatística tradicional, aprendizado de máquina, reconhecimento de padrões e técnicas de otimização [1].

1. Conceito de Predictive Modeling

A modelagem preditiva abrange métodos analíticos que buscam prever resultados futuros desconhecidos, extraindo informações úteis de grandes volumes de dados para ajudar organizações na tomada de decisões estratégicas [2].

2. Origem do Termo e Evolução Histórica

O termo surge com avanços em tecnologias de informação, estatística e aprendizado de máquina, remontando a técnicas clássicas como a análise discriminante linear, evoluindo até métodos avançados como redes neurais e máquinas de vetores de suporte [3].

3. Características do Predictive Modeling

- **Base em Dados Históricos:** Utiliza grandes conjuntos de dados passados para a modelagem [2].
- **Generalização:** Capacidade dos modelos preverem novos dados com precisão [4].
- **Quantificação da Incerteza:** Avaliação da precisão das previsões por meio de intervalos de confiança [4].
- **Automatização:** Possibilita eficiência operacional em processos repetitivos [2].
- **Diversidade de Técnicas:** Integra métodos estatísticos e modelos algorítmicos avançados [5].

4. Aplicações e Problemas Resolvidos

O Predictive Modeling auxilia na redução de incertezas em decisões estratégicas, com aplicações variadas como:

- Gestão de riscos financeiros e seguros [2];
- Diagnóstico médico preliminar [5];
- Marketing direcionado baseado em previsões comportamentais [6];
- Automatização industrial e comercial [2].

5. Ramificações Principais do Predictive Modeling

5.1 Data Modeling vs. Algorithmic Modeling

- **Data Modeling (Estatístico):** Assume modelos probabilísticos explícitos e definidos sobre os dados [7].
- **Algorithmic Modeling (Algorítmico):** Utiliza algoritmos complexos sem pressupor distribuição explícita dos dados [7].

5.2 Supervised vs. Unsupervised Learning

- **Aprendizado Supervisionado:** Usa dados rotulados previamente [8].
- **Aprendizado Não Supervisionado:** Descobre padrões sem necessidade de rotulação prévia [8].

5.3 Segmentação em Predictive Modeling

A segmentação tem papel importante na modelagem preditiva, especialmente em marketing de banco de dados, permitindo a divisão de clientes em segmentos homogêneos com base em comportamento de compra e outras características relevantes [6].

6. Técnicas Exemplares em Predictive Modeling

- **Análise Discriminante Linear e Quadrática:** Técnicas estatísticas tradicionais [8].
- **Redes Neurais:** Modelos não-lineares complexos [8].
- **Árvores de Decisão:** Modelos transparentes e intuitivos [8].
- **Métodos Ensemble:** Combinam múltiplos modelos para maior acurácia [3].
- **Métodos Baseados em Segmentação (CHAID, CART, GA):** Usados especialmente para decisões em marketing, permitindo segmentações automáticas baseadas em dados históricos [6].

Referências

[1] IBM Journal of Research and Development. Data-intensive analytics for predictive modeling, 2007. Disponível em: https://www.researchgate.net/publication/220497677_Data-intensive_analytics_for_predictive_modeling

[2] IBM Journal of Research and Development. Data-intensive analytics for predictive modeling, 2007.Disponível em: https://www.researchgate.net/publication/220497677_Data-intensive_analytics_for_predictive_modeling

[3] Hand, D.J. Classifier Technology and the Illusion of Progress. Statistical Science, 2006. Disponível em: https://arxiv.org/pdf/math.ST/0606441

[4] Breiman, L. Statistical Modeling: The Two Cultures. Statistical Science, 2001. Disponível em: https://projecteuclid.org/journals/statistical-science/volume-16/issue-3/Statistical-Modeling--The-Two-Cultures-with-comments-and-a/10.1214/ss/1009213726.full

[5] Michie, D.; Spiegelhalter, D.J.; Taylor, C.C. Machine Learning, Neural and Statistical Classification, 1994 Disponível em: https://theswissbay.ch/pdf/Gentoomen%20Library/Artificial%20Intelligence/Neural%20networks/Machine%20Learning,%20Neural%20And%20Statistical%20Classification%20-%20Cc%20Taylor.pdf

[6] Levin, N.; Zahavi, J. Predictive modeling using segmentation. Journal of Interactive Marketing, 2001. Disponível em: https://www.researchgate.net/publication/227701811_Predictive_modeling_using_segmentation

[7] Breiman, L. Statistical Modeling: The Two Cultures. Statistical Science, 2001. Disponível em: https://projecteuclid.org/journals/statistical-science/volume-16/issue-3/Statistical-Modeling--The-Two-Cultures-with-comments-and-a/10.1214/ss/1009213726.full

[8] Michie, D.; Spiegelhalter, D.J.; Taylor, C.C. Machine Learning, Neural and Statistical Classification, 1994 Disponível em: https://theswissbay.ch/pdf/Gentoomen%20Library/Artificial%20Intelligence/Neural%20networ

ks/Machine%20Learning,%20Neural%20And%20Statistical%20Classification%20-%20Cc%20Taylor.pdf

APÊNDICE 2

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 16 de abr. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

MATHEUS FARES COSTA BRAKES

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Segue o link para a documentação que elaborei com base no estudo sobre Concept Drift e Concept Evolution. No material, reuni definições, termos e referências essenciais sobre o tema.
Link : Concept Drift e Concept Evolution

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Como próxima etapa, planejo pesquisar frameworks e ferramentas disponíveis que abordem, na prática, a detecção e adaptação a concept drift e a integração de novas classes (concept evolution), visando aplicar esses conceitos para resolver os problemas estudados.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

[Documento Concept Drift e Concept Evolution citado no Termo de Aceite de Entrega de 16 de abril]

Concept Drift e Concept Evolution

Este resumo visa apresentar uma visão organizada dos fenômenos de Concept Drift e Concept Evolution no contexto de sistemas de aprendizado em fluxo de dados, destacando suas definições, origens, tipos e desafios práticos para sistemas adaptativos de inteligência artificial.

---

1. Introdução aos Conceitos

- **Concept Drift:** Refere-se à mudança inesperada na distribuição dos dados ou na relação entre os atributos de entrada (X) e a variável alvo (y) ao longo do tempo. Tal alteração pode comprometer a acurácia dos modelos treinados com dados históricos, exigindo detecção e adaptação contínua. Formalmente, ocorre quando $P_t(X, y) \neq P_{t+1}(X, y)$ para algum instante t [1][4].

- **Concept Evolution:** Diferentemente do Concept Drift, que trata de alterações nos dados já conhecidos, o Concept Evolution aborda a emergência ou desaparecimento de classes inéditas no conjunto de dados. Esse fenômeno implica uma expansão (ou contração) do espaço conceitual, onde novas categorias surgem ou classes conhecidas deixam de existir, exigindo do sistema a capacidade não só de adaptar os parâmetros dos modelos, mas também de incorporar novos conhecimentos e atualizar sua estrutura decisória [2][3][4].

---

2. Concept Drift

2.1 Origens do Concept Drift

- **Fonte I (Virtual Drift):** Mudança apenas na distribuição dos atributos P(X) sem alteração na relação P(y|X), ou seja, as fronteiras de decisão não se modificam [1][4].

- **Fonte II (Real Drift):** Alteração na relação condicional P(y|X), que acarreta a mudança das fronteiras de decisão e impacta diretamente a acurácia do modelo [1][4].

- **Fonte III (Drift Misto):** Concomitância das mudanças em P(X) e P(y|X), cenário comum em muitos ambientes reais [1][4].

2.2 Classificação dos Tipos de Drift

- **Abrupto:** A transição entre conceitos ocorre de maneira repentina, sem período de sobreposição [4].

- **Incremental:** A mudança se dá de forma contínua e por pequenos passos. Nesta modalidade, o conceito evolui de forma suave e acumulativa, sem coexistência marcante entre o antigo e o novo [4].

- **Gradual:** Caracteriza-se pela coexistência temporária dos conceitos antigo e novo, com uma alternância na proporção dos dados pertences a cada um, até que o novo conceito prevaleça [4].

- **Recorrente:** Os padrões que já ocorreram reaparecem periodicamente, o que requer que o sistema "lembre-se" dos conceitos passados para uma resposta eficaz [4].

---

3. Concept Evolution

3.1 Definição e Importância

- **Definição:** O Concept Evolution trata da dinâmica do surgimento (ou desaparecimento) de classes que não estavam previstas durante a fase de treinamento do modelo. Diferente do drift, que ajusta o modelo às mudanças dos padrões existentes, a evolução conceitual implica a ampliação do conjunto de classes – um aspecto frequentemente denominado como "novel class detection"

(detecção de classes inéditas) [2][3].

- **Importância:** Em muitos cenários do mundo real, como na classificação de tráfego de rede, sistemas de segurança e monitoramento, novas classes podem emergir devido à evolução das tecnologias ou mudanças comportamentais. Essa capacidade de identificar e incorporar novos conceitos é fundamental para manter a relevância e precisão das predições ao longo do tempo [2][3][4].

3.2 Desafios e Abordagens

- **Desafios:**
    - *Detecção precoce:* A identificação de uma nova classe sem a disponibilidade imediata de rótulos representa um dos maiores desafios no Concept Evolution.
    - *Integração ao modelo:* A atualização do modelo para incluir novas categorias sem comprometer o desempenho previamente adquirido pelos dados conhecidos.
    - *Gestão de recursos:* Em ambientes de dados em fluxo, a adaptação rápida e o gerenciamento eficiente dos limites de custo (por exemplo, rótulo ativo com orçamento limitado) tornam-se cruciais [2][3].
- **Abordagens:**
    - *Active Learning e Novel Class Detection:* Estratégias que combinam o re-treinamento contínuo com a análise ativa para solicitar rótulos de dados potencialmente pertencentes a novas classes.
    - *Algoritmos híbridos:* Técnicas que integram métodos de detecção de drift com mecanismos de atualização dinâmica do espaço conceitual, garantindo que o sistema possa lidar tanto com alterações nos dados conhecidos quanto com a emergência de novas categorias [2][3].

4. Integração entre Concept Drift e Concept Evolution

- **Interação dos Fenômenos:** Em ambientes dinâmicos, é comum que alterações nos dados (drift) ocorram em paralelo com a evolução de novos conceitos (evolution). Enquanto o drift leva à necessidade de ajustar fronteiras de decisão, a evolução impõe a reestruturação do modelo para integrar novas classes.

- **Abordagens Conjuntas:** Alguns algoritmos avançados implementam mecanismos híbridos que detectam simultaneamente a presença de drift e a emergência de classes inéditas. Essa abordagem integrada permite que o sistema não só adapte os modelos frente às mudanças, mas também amplie seu escopo para reconhecer novas categorias [2][3][4].

---

5. Desafios Práticos e Aplicações

- **Detecção em Tempo Real:** Mecanismos de monitoramento contínuo são essenciais para identificar tanto drift quanto evolução dos conceitos, permitindo reações rápidas para minimizar a degradação do desempenho [1][4].

- **Adaptação Contínua:** Estratégias de re-treinamento incremental, active learning e algoritmos híbridos possibilitam a atualização dos modelos conforme os novos dados e conceitos se tornam disponíveis [2][4].

- **Aplicações em Cenários Reais:** Em tarefas de classificação de tráfego de rede, por exemplo, a combinação de concept drift e evolution é crucial para lidar com desequilíbrios de classes, mudanças abruptas e o surgimento de novas categorias que podem afetar a segurança e a eficiência operacional [2][3].

---

6. Referências

[1] F. Bayram, B. S. Ahmed, & A. Kassler, “From Concept Drift to Model Degradation: An Overview on Performance-Aware Drift Detectors,” *arXiv preprint*, Mar. 2022.
[2] W. Liu et al., “Multiclass Imbalanced and Concept Drift Network Traffic Classification Framework Based on Online Active Learning,” *Engineering Applications of Artificial Intelligence*, Jan. 2023.
[3] M. M. Masud et al., “Classification and Novel Class Detection in Concept-Drifting Data Streams under Time Constraints,” *IEEE Transactions on Knowledge and Data Engineering*, Jul. 2011.
[4] J. Lu et al., “Learning under Concept Drift: A Review,” *arXiv preprint*, 2020.

APÊNDICE 3

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 23 de abr. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

MATHEUS FARES COSTA BRAKES

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Elaborei um documento de estudo sobre detecção de Concept Drift em diversos cenários — cobrindo métodos estatísticos, monitoramento por janelas, ensembles e técnicas baseadas em representações de alto nível (embeddings de LLMs). Concept Drift Detection

No documento também citei os frameworks encontrados, sendo os principais :

River: biblioteca Python para streaming e drift (DDM, EDDM, ADWIN, Page-Hinkley)

nannyML: monitoramento pós-deploy sem rótulos (PSI, JSD, CBPE, RMSE)

Evidently AI: dashboards de qualidade de dados e drift (PSI, KL, Wasserstein, qui-quadrado)

Alibi Detect: detecção de drift e outliers em embeddings (MMD, KS Test)

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima semana, considero que um bom caminho seria definir a área de atuação, já que isso pode orientar com mais clareza a escolha do framework ou ferramenta mais adequada, além de direcionar melhor os pontos em que devo me aprofundar. No entanto, ainda tenho dúvidas se o ideal não seria, antes disso, explorar com mais profundidade os frameworks disponíveis e seus casos de aplicação, já que a escolha da área poderia ser feita com base nas possibilidades e limitações reais de cada ferramenta.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

[Documento Concept Drift Detection citado no Termo de Aceite de Entrega de 23 de abril]

1. Métodos de Detecção de Concept Drift

Conforme a classificação de Gama et al. [1], as técnicas para detectar concept drift agrupam-se em quatro grandes categorias: (i) **métodos estatísticos**, (ii) **monitoramento de desempenho por janelas**, (iii) **técnicas baseadas em ensembles e discordância** e (iv) **métodos baseados em representação**. A seguir, apresentamos cada abordagem, destacando mecanismos, aplicações típicas e referências originais.

1.1 Métodos Estatísticos

Inspecionam diretamente as distribuições de dados ao longo do tempo, sem depender do desempenho de qualquer modelo preditivo [1].

- **Page-Hinkley Test (PHT)**: monitora a soma acumulada das diferenças entre cada observação e a média histórica, acionando alertas quando o desvio ultrapassa um limiar pré-definido. Proposto por Page (1954) e adaptado para concept drift por Gama et al. (2004) [2].

- **Kolmogorov–Smirnov Test (KS Test — Teste de Kolmogorov-Smirnov)**: calcula a maior distância entre duas funções de distribuição acumulada, identificando mudanças em distribuições contínuas [1].

- **Kullback–Leibler Divergence (KL — Divergência de Kullback-Leibler)**: mede a entropia relativa entre uma distribuição de referência e a atual, sensível a deslocamentos informacionais em altas dimensões [3].

- **Wasserstein Distance (Distância de Wasserstein)**: quantifica o custo de transporte ótimo entre duas distribuições, capturando mudanças geométricas nos dados [7].

Aplicações típicas: ambientes com rótulos escassos ou atrasados, detecção de drift covariável antes que o erro do modelo se manifeste [1].

1.2 Monitoramento de Desempenho por Janelas

Baseia-se na análise de métricas de erro ou acurácia calculadas dentro de **janelas temporais** fixas ou adaptativas que “deslizam” conforme novos dados chegam. O conceito de janela deslizante foi introduzido por Widmer & Kubat (1996) para lidar com mudanças em fluxos de dados [8].

Em geral, definem-se duas janelas — uma “antiga” e outra “recente” — e comparam-se estatísticas como média de erro e desvio-padrão. Caso a diferença supere um limiar estatístico, um alerta de drift é gerado [8].

- **Drift Detection Method (DDM)**: compara a taxa de erro atual p_t e seu desvio s_t com mínimos históricos (p_{min}, s_{min}), disparando drift se pt+st>pmin+α sminp_t + s_t > p_{min} + \alpha\,s_{min}pt+st>pmin+αsmin Proposto por Gama et al. (2004) [2].

- **Early Drift Detection Method (EDDM)**: melhora o DDM ao monitorar a distância entre erros sucessivos, tornando-se mais sensível a drifts graduais [3].

- **ADWIN (Adaptive Windowing)**: ajusta dinamicamente o tamanho das janelas com base no teste de Hoeffding para manter apenas dados “estacionários” em cada janela, sinalizando drift quando ocorre mudança estatisticamente significativa entre sub-janelas [4].

Aplicações típicas: cenários com rótulos disponíveis (mesmo que atrasados), detecção de drifts conceituais em sistemas de decisão em tempo real [1].

1.3 Métodos Baseados em Ensembles e Discordância

Utilizam múltiplos classificadores para inferir drift a partir da **discordância** entre suas predições [1]. A variação no consenso sinaliza que o comportamento dos dados pode ter mudado.

- **Classifier Disagreement Methods**: calcula a proporção de instâncias em que dois ou mais classificadores discordam, acionando drift quando o índice de discordância excede um limiar [5].

- **Streaming Ensemble Algorithm (SEA)**: mantém um conjunto de modelos que é continuamente atualizado, descartando instâncias antigas que perdem relevância e

treinando novos modelos com dados recentes [11].

- **Learn++.NSE**: atribui pesos variáveis a cada classificador do ensemble com base no desempenho em blocos de dados mais recentes, adaptando-se rapidamente a mudanças não lineares [5].

Aplicações típicas: ambientes sem rótulos imediatos, drifts complexos ou não lineares em que a diversidade do ensemble confere robustez [1].

1.4 Métodos Baseados em Representação

Focam em mudanças nos **espaços latentes** extraídos dos dados (por exemplo, embeddings de LLMs ou features profundas), ao invés de trabalhar diretamente nas variáveis originais . Isso os torna especialmente úteis quando o “drift” afeta padrões complexos que não são evidentes nas estatísticas brutas.

- **Maximum Mean Discrepancy (MMD)**: teste kernel que compara médias de dois conjuntos de amostras em um espaço de Hilbert, detectando drift em representações de alta dimensão. Proposto por Gretton et al. (2012) [6].

- **Wasserstein Distance**: aplicado sobre embeddings em vez de variáveis originais para capturar mudanças profundas na distribuição latente [7].

Aplicações típicas: pipelines de embeddings e deep learning, especialmente em visão computacional e processamento de linguagem natural [6].

---

2. Frameworks Open-Source para Detecção de Concept Drift

Diversas bibliotecas oferecem implementações maduras dos métodos acima, facilitando a integração em pipelines de **ML Ops**:

- **River** (Python): suporta DDM, EDDM, ADWIN, PHT, HDDM e KS-WIN, voltado a aprendizado incremental em streaming para casos como fraude em tempo real e

sistemas de recomendação. Desenvolvido por Montiel et al. (2021) [9].

- **scikit-multiflow** (Python): implementa DDM, EDDM, ADWIN e PHT em um framework de análise de fluxo de dados, ideal para classificação incremental. Descrito por Montiel et al. (2018) [10].

- **MOA** (Java): oferece ADWIN, DDM, CUSUM, EDDM e algoritmos ensemble, amplamente usado como benchmark em ambientes simulados de data streams [11].

- **NannyML** (Python): fornece métricas não supervisionadas (PSI, JSD) e supervisionadas (CBPE, RMSE) para monitoramento pós-deploy sem rótulos, recomendado para produção de modelos de ML. Apresentado por Van Looveren et al. (2023) [12].

- **Alibi Detect** (Python): inclui MMD, KS Test, CBSD e métodos de drift em embeddings, indicado para pipelines de deep learning, LLMs e visão computacional. Documentado por Klaise et al. (2021) [13].

- **Evidently AI** (Python): foca em visualização de drift via PSI, KL, Wasserstein e qui-quadrado, integrável a pipelines CI/CD para dashboards de ML. Documentação oficial [16].

- **NAB (Numenta Anomaly Benchmark)** (Python): especializado em séries temporais e IoT, com métricas de anomalia e thresholds adaptativos. Avaliado por Lavin & Ahmad (2015) [14].

- **Knodle** (Python): framework de weak supervision que suporta drift detection em cenários de NLP ruidoso, aplicável a tarefas de classificação de relações. Descrito por Paul et al. (2021) [15].

---

3. Comparação entre os Métodos de Detecção

Embora todas as abordagens visem identificar alterações no comportamento dos dados, elas diferem quanto aos sinais observados, requisitos de supervisão, tipo de drift detectado e custo computacional. A seguir, destacam-se as principais diferenças:

Critério	Métodos Estatísticos	Monitoramento por Janelas	Ensembles / Discordância	Baseados em Representação
Sinal de drift	Mudança na distribuição dos dados brutos	Variação em métricas de erro/acurácia	Discordância entre predições de modelos	Deslocamento em espaços latentes (embeddings)
Dependência de rótulos	Não requer	Requer (até atrasados)	Opcional	Não requer
Tipo de drift detectado	Principalmente covariável	Conceitual e covariável	Tanto covariável quanto conceitual	Latente / estrutural
Sensibilidade a ruído	Média (pode gerar falsos positivos)	Alta (janelas fixas podem reagir a flutuações transitórias)	Baixa (consenso reduz o efeito de outliers)	Média–alta (depende do kernel/embedding)
Velocidade de detecção	Muito rápida (cálculo direto sobre estatísticas)	Moderada (espera preencher janelas suficientes)	Variável (depende do número de modelos)	Lenta (cálculo de kernels ou embeddings)
Custo computacional	Baixo	Moderado	Elevado (vários modelos simultâneos)	Elevado (operações em alta dimensão)
Aplicações ideais	Cenários sem rótulos ou com	Sistemas de decisão com	Ambiente sem rótulos imediatos;	Pipelines de deep learning e LLMs [6]

	etiquetas atrasadas [1]	rótulos confiáveis [2]	drifts complexos [5]	

- **Métodos Estatísticos vs. Monitoramento por Janelas**: enquanto os estatísticos disparam drift ao identificar qualquer mudança estatística na distribuição, os baseados em janelas só sinalizam quando o impacto afeta efetivamente o desempenho do modelo, reduzindo falsos positivos em cenários barulhentos .

- **Janelas Fixas vs. Adaptativas**: técnicas como ADWIN ajustam dinamicamente o tamanho da janela, ao contrário de DDM/EDDM, que usam janelas fixas e podem fracassar em identificar drifts graduais de forma uniforme .

- **Ensembles e Discordância**: diferentemente das abordagens univariadas, aqui a robustez advém do consenso entre múltiplos modelos, o que melhora a detecção de drifts não lineares mas aumenta a necessidade de recursos computacionais [5].

- **Representação vs. Estatística Bruta**: métodos baseados em representação podem capturar mudanças em padrões complexos de alto nível (por exemplo, em embeddings de LLMs), mas sofrem com o custo de cálculo de distâncias em espaços de alta dimensão [6].

Referências

[1] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). *A survey on concept drift adaptation*. ACM Computing Surveys, 46(4), 1–37.
[2] Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). *Learning with Drift Detection*. SBIA Brazilian Symposium on Artificial Intelligence, 286–295.
[3] Kullback, S., & Leibler, R. A. (1951). *On information and sufficiency*. Annals of Mathematical Statistics, 22(1), 79–86.
[4] Bifet, A., & Gavaldà, R. (2007). *Learning from time-changing data with adaptive windowing*. SIAM International Conference on Data Mining, 443–448.
[5] Kuncheva, L. I., & Plumpton, C. O. (2008). *Adaptive learning with classifier ensembles*. IEEE International Conference on Adaptive and Intelligent Systems, 295–300.
[6] Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., & Smola, A. J. (2012). *A kernel two-sample test*. Journal of Machine Learning Research, 13(25), 723–773.
[7] Courty, N., Flamary, R., Habrard, A., & Rakotomamonjy, A. (2017). *Joint distribution*

optimal transport for domain adaptation. NeurIPS.
[8] Widmer, G., & Kubat, M. (1996). *Learning in the presence of concept drift and hidden contexts*. Machine Learning, 23(1), 69–101.
[9] Montiel, J. F., et al. (2021). *River: machine learning for streaming data in Python*.
[10] Montiel, J. F., et al. (2018). *Scikit-Multiflow: A Multi-output Streaming Framework in Python*.
[11] Bifet, A., et al. (2010). *MOA: Massive Online Analysis*.
[12] Van Looveren, A., et al. (2023). *Monitoring and Estimating Performance of ML Models Without Labels*.
[13] Klaise, J., et al. (2021). *Alibi Detect: Algorithms for Outlier, Adversarial, and Drift Detection*.
[14] Lavin, A., & Ahmad, S. (2015). *Evaluating real-time anomaly detection algorithms*.
[15] Paul, N., et al. (2021). *Knodle: Weak Supervision Framework for Relation Classification*.
[16] Evidently AI. Documentação oficial. https://docs.evidentlyai.com

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 30 de abr. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

MATHEUS FARES COSTA BRAKES

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Realizei uma análise aprofundada comparando os frameworks River, NannyML, Evidently AI e Alibi Detect para detecção de drift, vendo alguns casos de aplicação. O objetivo foi entender melhor onde cada um se encaixa para auxiliar na escolha de uma ferramenta e de um problema a ser solucionado.

Em resumo, os casos de uso ideais identificados são:

- River: Processamento e detecção de drift em streaming contínuo de dados.
- NannyML: Monitoramento em lote (batch) quando não há rótulos recentes, focando em estimar o impacto do drift no desempenho.
- Evidently AI: Monitoramento em lote (batch) com forte capacidade de gerar relatórios visuais e dashboards detalhados.
- Alibi Detect: Detecção de drift em dados mais complexos ou de alta dimensão, como embeddings, e também para análise de outliers.

Link do documento : Estudo Framework

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima semana, vejo como tarefa já definir qual será o problema ou caso de uso específico de detecção de drift a ser trabalhado.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

[Documento Estudo Framework citado no Termo de Aceite de Entrega de 30 de abril]

Análise Comparativa Detalhada de Frameworks para Detecção de Drift em Modelos de Machine Learning

Este documento tem como objetivo analisar e comparar quatro frameworks Python amplamente utilizados para detecção de drift (deriva) em modelos de Machine Learning em produção: **River**, **NannyML**, **Evidently AI** e **Alibi Detect**. A análise visa destacar suas características principais, funcionalidades e cenários de aplicação mais indicados, fornecendo uma base sólida para a tomada de decisão ao escolher a ferramenta mais adequada para sua necessidade.

1. Tabela Comparativa Resumida

Para uma visão geral rápida das funcionalidades e diferenças chave entre os frameworks, apresentamos a seguinte tabela comparativa:

Critério	River	NannyML	Evidently AI	Alibi Detect
Foco Principal	Aprendizado de máquina em streaming / incremental	Monitoramento pós-deploy sem rótulos, estimativa de desempenho	Visualização e monitoramento de qualidade de dados e drift em produção	Detecção de outliers, adversarial attacks e drift (especialmente embeddings)
Métodos de Drift (Ex.)	DDM, EDDM, ADWIN, PHT, HDDM, KS-WIN	Drift: PSI, JSD. Desempenho: CBPE,	PSI, KL Div, Wasserstein, Chi-Sq, etc.	MMD, KS Test, Chi-Sq, LSDD, CVM (Features, Embeddings)

		DLE (estimados).	(Dados, Predições)	
Casos de Uso Típicos	Processamento contínuo, classificação/regressão incremental, tempo real	Monitorar modelos em prod s/ rótulos (fraude, crédito), validar saúde do modelo	Gerar relatórios/dashboards comparativos, monitorar pipelines, depurar dados/modelo	Monitorar modelos DL (NLP, CV), drift em representações latentes, segurança
Requer Rótulos?	Métodos de desempenho (DDM/EDDM): Sim. Estatísticos (PHT): Não.	Não para drift (PSI/JSD). Sim (históricos) para desempenho (CBPE/DLE).	Não para drift de dados. Sim para drift de predição/desempenho.	Não para maioria (MMD, KS). Pode requerer para métodos específicos.
Tipos de Drift Cobertos	Feature/Covariate (PHT, KS-WIN). Concept (DDM, EDDM via erro).	Feature (PSI, JSD). Concept (indireto via CBPE/DLE). Label (indireto via predições).	Feature (Forte: PSI, KL, etc.). Prediction (Forte). Concept (Indireto). Label (distribuição target).	Feature/Embedding (Forte: MMD, KS). Covariate (KS). Concept/Label (Menos direto).

Explicabilidade	Baixa (foco no sinal de drift).	Alta (contribuição de features para drift de dados e desempenho estimado).	Média/Alta (ótimas visualizações por feature).	Baixa/Média (métricas). Pode integrar explicadores externos.
Integração MLOps	Média (biblioteca p/ streaming, requer engenharia).	Alta (CLI/API, integrações MLflow, Airflow, Grafana, Seldon).	Alta (API, integrações MLflow, Airflow, Grafana. Saída JSON/HTML).	Alta (Biblioteca Python, Seldon Core. Flexível).
Robustez / Sensibilidade	Média/Baixa (DDM/EDDM sensíveis se não calibrados). ADWIN + adaptativo.	Alta (PSI/JSD com janelas, CBPE/DLE estáveis com calibração). Foco em mudanças significativas.	Média (depende do teste/limiar/janela). Pode gerar FP com sensibilidade alta.	Média (MMD sensível a mudanças sutis, requer calibração). KS depende do α.
Custo Computacional	Moderado a Alto (depende do algoritmo e taxa de dados).	Moderado a Alto (PSI/JSD rápidos, CBPE/DLE podem ser caros).	Baixo a Moderado (testes rápidos, relatórios consomem mais).	Alto (Métodos de Kernel como MMD, operações em embeddings).

Frequência de Execução	Contínua / Micro-lote	Periódica (Lotes)	Periódica / Sob Demanda	Periódica / Sob Demanda
Armazenamento	Baixo (estado) / Moderado (logs)	Moderado a Alto (dados ref/análise, métricas)	Moderado (datasets p/ relatórios, relatórios)	Moderado a Alto (embeddings, estado, resultados)
Aplicabilidade (Simples)	Menos ideal	Sim	Sim	Sim (KS), mas brilha com complexidade

2. Análise Detalhada de Critérios Essenciais

Aprofundando em alguns critérios chave da tabela comparativa:

- **Cobertura e Nuances dos Tipos de Drift:** A detecção de drift abrange diferentes tipos de mudanças nos dados ou no relacionamento entre eles. A distinção entre *Feature/Covariate Drift* (P(X) muda), *Concept Drift* (P(y|X) muda) e *Label Drift* (P(y) muda) é fundamental. Detectar concept drift puro sem rótulos é um desafio intrínseco.
    - **NannyML** aborda indiretamente o concept drift estimando seu *impacto* no desempenho do modelo (P(y|X)) mesmo na ausência de rótulos recentes, utilizando métodos como CBPE (Confidence-Based Performance Estimation) e DLE (Direct Label Estimation).
    - **River**, através de algoritmos como DDM e EDDM, detecta concept drift observando o aumento no erro de previsão, o que pressupõe a disponibilidade de rótulos com baixa latência.
    - **Evidently AI** e **Alibi Detect** oferecem robustez na detecção de drift em features (P(X)) e predições, com Alibi Detect sendo particularmente forte para drift em embeddings (representações latentes de alta dimensão). O monitoramento de label drift pode ser feito em Evidently AI pela análise da distribuição da variável target.

- **Explicabilidade e Interpretabilidade:** Compreender *por que* o drift está acontecendo é crucial para a tomada de decisão (ex: retreinar o modelo, investigar a fonte dos dados).
    - **NannyML** se destaca significativamente neste aspecto, fornecendo ferramentas para analisar a contribuição individual de cada feature para o drift detectado, tanto nos dados de entrada quanto no desempenho estimado do modelo.
    - **Evidently AI** oferece visualizações ricas e interativas que facilitam a inspeção das distribuições de features e predições ao longo do tempo, ajudando a identificar visualmente quais aspectos dos dados mudaram.
    - **River** e **Alibi Detect** fornecem principalmente sinais de alerta e métricas estatísticas que indicam a *ocorrência* do drift, mas a investigação das causas raiz geralmente requer passos adicionais ou integração com outras ferramentas.
- **Integração em Pipelines MLOps:** A facilidade com que um framework pode ser incorporado em fluxos de trabalho de produção (agendamento, alertas, logging, visualização em dashboards) é vital.
    - **NannyML**, **Evidently AI** e **Alibi Detect** são projetados com a automação em mente, oferecendo APIs Python, CLIs e saídas em formatos como JSON ou HTML que se integram bem com orquestradores (Airflow), plataformas MLOps (MLflow, Seldon Core) e ferramentas de visualização (Grafana).
    - **River**, sendo primariamente uma biblioteca para aprendizado e detecção em fluxo contínuo, se encaixa naturalmente em arquiteturas de processamento de streaming. Sua integração em pipelines MLOps mais tradicionais (baseados em lotes periódicos) pode exigir mais trabalho de engenharia para adaptar o modelo de processamento.
- **Robustez e Sensibilidade a Ruído:** A calibração da sensibilidade é um ponto crítico para evitar falsos positivos causados por flutuações normais dos dados.
    - Métodos que operam sobre dados agregados em lotes (**NannyML** com PSI/JSD, **Evidently AI** com seus testes estatísticos em janelas, **Alibi Detect** com KS/MMD em lotes) tendem a ser mais robustos a ruído pontual, focando em mudanças de distribuição mais sistemáticas.
    - Algoritmos do **River** como DDM/EDDM, que reagem a mudanças no erro instância a instância, podem ser mais sensíveis a flutuações transitórias se os limiares não forem bem ajustados. Algoritmos mais adaptativos como ADWIN são mais robustos nesse contexto.

- A escolha dos algoritmos, o tamanho das janelas de comparação e os limiares de significância (α para testes estatísticos, λ para Page-Hinkley, etc.) são parâmetros cruciais para configurar a robustez versus sensibilidade em qualquer framework.

3. Guia de Seleção Baseado em Casos de Uso

A escolha do framework mais adequado dependerá diretamente das características do seu problema, do ambiente operacional e das funcionalidades prioritárias:

- **Para Cenários de Processamento Contínuo e Streaming em Tempo Real:** Se a sua aplicação processa dados em um fluxo contínuo, item a item ou em micro-lotes muito frequentes, e a detecção de drift precisa ocorrer quase instantaneamente nesse fluxo, o **River** é a recomendação natural. Ele foi construído para este paradigma, com algoritmos otimizados para aprendizado incremental e detecção de drift com baixo overhead por instância, essencial para requisitos de baixa latência.

- **Para Monitoramento em Lotes com Forte Ênfase em Visualização e Relatórios Detalhados:** Quando o monitoramento é realizado em lotes periódicos (ex: diário, semanal) e a necessidade principal é gerar relatórios visuais compreensíveis, comparar distribuições de dados e predições, e facilitar a auditoria e o diagnóstico manual do drift, o **Evidently AI** é uma escolha altamente eficiente. Sua capacidade de gerar relatórios em formatos como HTML e JSON, combinada com uma ampla gama de métricas e visualizações por feature, o torna ideal para diagnóstico visual e comunicação.

- **Para Monitoramento em Lotes Onde Rótulos Imediatos Não Estão Disponíveis, mas o Impacto no Desempenho é Crítico:** Em muitos cenários de produção, o ground truth (rótulo verdadeiro) só fica disponível após um tempo considerável. Se você opera em lotes e precisa entender se o drift detectado está *realmente* afetando a performance do seu modelo (AUC, F1, etc.), mesmo sem os rótulos recentes, o **NannyML** é a ferramenta de destaque. Sua funcionalidade única de estimar métricas de desempenho na ausência de rótulos recentes preenche uma lacuna importante e ajuda a priorizar ações com base no impacto real no negócio.

- **Para Detecção de Drift em Dados de Alta Dimensionalidade, Embeddings ou com Necessidade de Detecção Adicional (Outliers/Adversarial):** Ao lidar com dados complexos e de alta dimensão, como representações (embeddings) geradas por modelos de Deep Learning em Visão Computacional ou Processamento de Linguagem Natural, ou se há uma necessidade integrada de detectar outliers e

possíveis ataques adversariais, o **Alibi Detect** oferece os métodos mais adequados. Inclui algoritmos como MMD (Maximum Mean Discrepancy), eficazes para comparar distribuições complexas, e se posiciona como uma ferramenta mais abrangente para a saúde e segurança de modelos de ML, especialmente DL.

4. Considerações Adicionais: O Cenário de Micro-Lotes

É crucial reconhecer que a distinção entre "streaming puro" e "lotes" pode se tornar menos clara em cenários de "micro-lotes", onde pequenos grupos de dados são processados em intervalos de tempo muito curtos (ex: a cada minuto). Embora o **River** seja conceitualmente otimizado para processamento instância a instância, em cenários de micro-lotes, os frameworks orientados a lotes como **Evidently AI**, **NannyML** ou **Alibi Detect** podem, em muitos casos, ser opções válidas ou até preferíveis, dependendo das prioridades:

- **Eficiência Total no Lote:** Para micro-lotes de tamanho considerável, o custo computacional de processar o lote inteiro usando operações vetorizadas (comuns em Evidently AI, NannyML, e testes em Alibi Detect) pode ser mais eficiente no tempo total do que o processamento sequencial do River sobre as mesmas instâncias.
- **Robustez do Sinal:** A análise de estatísticas agregadas calculadas sobre o micro-lote inteiro tende a gerar sinais de drift mais estáveis e menos propensos a falsos positivos causados por ruído em instâncias isoladas, comparado a detectores que disparam com base em critérios por instância.
- **Necessidades Analíticas Específicas:** Se, mesmo em micro-lotes, a necessidade é de visualizações detalhadas (Evidently AI), estimativa de desempenho sem rótulos (NannyML) ou análise de embeddings/outliers (Alibi Detect), a escolha naturalmente se inclinará para um desses frameworks.

APÊNDICE 4

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 14 de mai. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

MATHEUS FARES COSTA BRAKES

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Esta entrega é composta por três documentos complementares que aprofundam o estudo sobre concept drift, com foco em sua caracterização, condições de ocorrência e aplicações práticas, especialmente em contextos de NLP.

Documento 1 – Taxonomias e Definições -
Taxonomia e Detecção de Concept Drift em Aprendizado de Máquina: Uma Revisão Técnica
Reúne e organiza termos fundamentais relacionados ao concept drift, com destaque para categorias de triggers, tipos de ambientes onde sua detecção é relevante e condições que justificam a adaptação de modelos. Este material cobre lacunas conceituais que ainda não estavam presentes nos trabalhos anteriores.

Documento 2 – Casos de Uso Gerais - Aplicações
Apresenta aplicações práticas da detecção de concept drift em diferentes domínios, com base em estudos de caso em diferentes domínios.

Documento 3 – Casos de Uso em NLP - Aplicação NLP
Aprofunda os estudos na área de processamento de linguagem natural, abordando casos como semantic drift (resignificação de termos), alterações contextuais de verdades com o tempo (ex.: "Dilma é presidente"), e fenômenos causados principalmente pela evolução temporal do uso da linguagem.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima semana, o foco será levantar um problema concreto dentro do tema de concept drift, onde sua detecção seja necessária e possa impactar diretamente a performance de um sistema. Com o problema definido, será realizada uma análise para identificar possíveis abordagens técnicas que possam ser aplicadas à situação, considerando a viabilidade de implementação com os recursos disponíveis.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

[Documento Taxonomia e Detecção de Concept Drift citado no Termo de Aceite de Entrega de 14 de maio]

Artigo Técnico sobre Aspectos Conceituais e Taxonômicos do *Concept Drift*

Este artigo tem como objetivo organizar e explicar os elementos conceituais e taxonômicos do fenômeno de *Concept Drift* no contexto de aprendizado de máquina. Serão abordadas as tipologias de *Concept Drift*, as características dos dados que motivam sua detecção, os gatilhos e condições que levam à sua ocorrência, e as variações de nomenclaturas encontradas na literatura, com base em referências acadêmicas.

1. Definição e Significância do *Concept Drift*

Concept Drift em aprendizado de máquina refere-se ao fenômeno onde as propriedades estatísticas da variável alvo, ou mais criticamente, a relação entre as características de entrada e a variável alvo, mudam ao longo do tempo [1, 8]. Essa evolução dos padrões de dados significa que modelos de aprendizado de máquina (ML), tipicamente treinados com dados históricos sob a suposição de que esses dados representam acuradamente instâncias futuras, podem ter seu desempenho preditivo degradado. Quando o "conceito" subjacente que o modelo aprendeu não é mais verdadeiro para novos dados de entrada, a acurácia e confiabilidade do modelo podem diminuir, potencialmente tornando-o obsoleto [8]. Isso ocorre porque ambientes do mundo real são frequentemente dinâmicos e mudam de maneiras que contrastam com os dados de treinamento estáticos com os quais o modelo aprendeu originalmente. Consequentemente, manter a acurácia e confiabilidade do modelo em tais ambientes dinâmicos e não estacionários necessita da detecção e gerenciamento do *concept drift* [1]. A suposição fundamental de estacionariedade na distribuição de probabilidade conjunta P(X,Y) (onde X representa as características de entrada e Y a variável alvo) é violada pelo *concept drift*, levando a essa queda de desempenho [2].

A onipresença da não estacionariedade em sistemas do mundo real implica que o *concept drift* não é uma ocorrência excepcional, mas sim um desafio comum. Modelos de ML aprendem padrões a partir de dados, assumindo que esses padrões persistirão. No entanto, como os ambientes do mundo real são inerentemente dinâmicos, as propriedades estatísticas dos dados e as relações dentro deles estão propensas a mudanças [1]. Isso sugere que a maioria dos sistemas de ML destinados à operação de longo prazo

inevitavelmente encontrará *concept drift*, tornando sua detecção uma preocupação geral em vez de um problema de nicho [2].

2. Distinguindo *Concept Drift* de Fenômenos Relacionados

É essencial distinguir o *concept drift* de fenômenos relacionados, porém distintos, como *data drift*, *model drift* e *prediction drift*.

- **Data Drift (Covariate Shift)**: Refere-se a mudanças na distribuição de probabilidade das características de entrada, P(X), enquanto a distribuição de probabilidade condicional da variável alvo dadas as características de entrada, P(Y|X), pode permanecer estável [2]. Essencialmente, as características dos dados de entrada mudam, mas a relação subjacente que mapeia entradas para saídas não muda. Com o *data drift*, o limite de decisão aprendido pelo modelo não necessariamente muda; apenas a distribuição de probabilidade das entradas muda.
- **Concept Drift (Real Drift)**: Denota especificamente mudanças na relação entre as características de entrada e a variável alvo; ou seja, a distribuição de probabilidade condicional P(Y|X) muda [1]. Os padrões ou regras subjacentes que o modelo aprendeu não são mais precisos. Neste caso, o próprio limite de decisão muda.
- ***Model Drift***: É um termo mais amplo e geral que descreve a degradação do desempenho de um modelo de ML ao longo do tempo [6]. É uma observação da queda na qualidade preditiva e pode ser causado por *concept drift*, *data drift*, ou uma combinação de ambos, bem como outros fatores como mudanças no *pipeline* de dados ou engenharia de características.
- ***Prediction Drift***: Refere-se a mudanças na distribuição das previsões de saída do modelo, P(Y^). O *prediction drift* pode ser um indicador precoce ou um sintoma de *data drift* ou *concept drift*. No entanto, nem sempre significa uma deterioração na qualidade do modelo.

Embora esses fenômenos sejam definidos separadamente, *data drift* e *concept drift* frequentemente co-ocorrem, o que pode complicar o isolamento da causa precisa da degradação do modelo.

3. Tipologias de *Concept Drift*

O *concept drift* não é um fenômeno monolítico; ele se manifesta de várias formas, caracterizadas por sua natureza subjacente, dinâmica temporal e mecanismos causais. Uma taxonomia clara é essencial para entender, detectar e se adaptar a essas diversas mudanças [1, 2, 3, 4].

3.1. *Fundamental Distinctions*

Uma distinção primária na literatura é feita entre *real concept drift* e *virtual concept drift*, com base em qual aspecto do processo de geração de dados muda [4].

- **Real Drift (Real Concept Drift)**: Refere-se a uma mudança fundamental na relação entre as características de entrada e a variável alvo. Formalmente, é definido como uma mudança na distribuição de probabilidade condicional da saída dada a entrada, P(Y|X) [1, 4]. Isso implica que o conceito subjacente real, o limite de decisão ou as regras que mapeiam entradas para saídas evoluíram. O *real drift* exige inequivocamente o reaprendizado ou modificação significativa da estrutura do modelo.
- **Virtual Drift (Virtual Concept Drift)**: Tipicamente descreve mudanças na distribuição de dados de entrada P(X), enquanto a distribuição condicional P(Y|X) permanece estável [1, 4]. Neste cenário, o conceito subjacente ou a verdadeira relação entre entradas e saídas não mudou. No entanto, o desempenho do modelo ainda pode degradar se a mudança em P(X) levar o modelo a encontrar regiões do espaço de características para as quais não foi suficientemente treinado, ou se as probabilidades anteriores das classes, P(Y), mudarem significativamente. Alguns pesquisadores sugerem que, na prática, ambos os tipos de *drift* frequentemente necessitam de mudanças ou retreinamento do modelo, porque mesmo o *virtual drift* pode fazer com que porções anteriormente não vistas do conceito alvo se tornem mais prevalentes [4].

3.2. *Classifications by Temporal Dynamics*

O *concept drift* também pode ser classificado com base em como ele se desenvolve ao longo do tempo — sua velocidade e padrão de mudança. Essas características temporais influenciam significativamente a escolha de algoritmos de detecção e estratégias de adaptação [1].

- ***Sudden (Abrupt) Concept Drift***: Caracterizado por uma mudança rápida e definitiva onde um conceito antigo é rapidamente substituído por um novo conceito em um ponto específico no tempo [1]. Após este ponto, novos dados são gerados predominantemente ou inteiramente pelo novo conceito. Este tipo de *drift* é frequentemente o mais fácil de detectar porque as novas instâncias de dados rapidamente se tornam diferentes dos dados passados nos quais o modelo foi treinado.
- ***Gradual Concept Drift***: Envolve uma transição lenta e suave de um conceito antigo para um novo conceito ao longo de um período extenso [1]. Durante esta fase de transição, instâncias de ambos os conceitos, antigo e novo, podem coexistir, com a

probabilidade de amostragem do novo conceito aumentando ao longo do tempo enquanto a do conceito antigo diminui. Esta forma de *drift* é geralmente mais difícil de detectar, particularmente se a mudança for muito sutil ou ocorrer ao longo de um período muito longo [9].

- ***Incremental Concept Drift***: Semelhante ao *drift* gradual, mas frequentemente descrito como ocorrendo através de uma série de pequenos passos discretos ou adições que se acumulam ao longo do tempo, eventualmente formando um conceito significativamente diferente [1]. Cada incremento individual de mudança pode ser menor e difícil de notar isoladamente. Assim como o *drift* gradual, pode ser desafiador detectar cedo, pois mudanças incrementais individuais podem ser estatisticamente insignificantes ou se misturar com o ruído natural dos dados. A distinção entre *drift* gradual e incremental pode, às vezes, ser sutil na prática.
- ***Recurring (Seasonal/Cyclical) Concept Drift***: Ocorre quando um conceito que estava ativo no passado reaparece após um período de ausência, frequentemente seguindo um padrão cíclico ou sazonal [1]. A reemergência do conceito antigo pode ser súbita ou gradual. Se o ciclo ou padrão de recorrência for conhecido ou previsível, este tipo de *drift* pode ser antecipado. Oferece o potencial de melhorar o desempenho do modelo reutilizando ou adaptando modelos ou conhecimentos previamente aprendidos associados ao conceito recorrente [1].
- ***Local Drift***: O *concept drift* é limitado a um subconjunto do espaço de entrada.

3.3. *Other Relevant Classifications*

- ***Performative Drift***: É um tipo especializado de *concept drift* caracterizado por um *loop* de retroalimentação onde as previsões feitas por um modelo de ML implantado influenciam ativamente as futuras distribuições de dados sobre as quais ele fará previsões [11]. Isso é particularmente prevalente em domínios onde os sujeitos da predição podem reagir à saída do modelo. Este *loop* causal é comum em cenários adversariais e em sistemas como mecanismos de recomendação [11]. Identificar o *performative drift* requer mais do que apenas detectar mudanças em P(X,Y); necessita de um entendimento do impacto causal das previsões do modelo no processo de geração de dados [11].

3.4. Application Contexts: Continual vs. Static Learning

A literatura sobre concept drift tradicionalmente foca nos tipos e padrões de mudança, como alterações em P(X)P(X)P(X), P(Y)P(Y)P(Y), ou P(Y|X)P(Y|X)P(Y|X). No entanto, a forma como essas mudanças impactam o desempenho e a adaptação dos modelos de aprendizado depende crucialmente do **regime de aprendizagem adotado** no sistema. Em linhas gerais, os contextos de aplicação podem ser organizados em **dois paradigmas**

principais: **aprendizado contínuo (continual/online learning)** e **aprendizado estático (batch/offline learning)** [2, 10, 13].

- **Continual Learning Contexts**

  Em ambientes onde os dados chegam de forma sequencial, contínua e potencialmente infinita, como fluxos de dados (*data streams*), o concept drift é tratado por meio de **aprendizado contínuo**, em que o modelo é atualizado incrementalmente ao longo do tempo. Nesses cenários, é comum que o modelo não retenha todos os dados históricos, utilizando abordagens como *incremental learning*, *window-based updates*, ou *drift-aware adaptation* [1, 2, 3].

  Algumas das estratégias populares incluem:

  - *Online ensemble learning*, onde múltiplos modelos são treinados em paralelo e atualizados dinamicamente;
  - Detectores de concept drift embutidos que ativam mecanismos de adaptação ao detectar mudanças estatísticas;
  - *Meta-learning adaptativo*, onde o sistema aprende a aprender com as mudanças [8, 12].

  A terminologia especializada dentro desse campo frequentemente aborda três subtipos complementares:

  - **Class-Incremental Learning**: o sistema aprende novas classes ao longo do tempo, o que desafia a consistência da distribuição P(Y|X)P(Y|X)P(Y|X) [13];
  - **Task-Incremental Learning**: tarefas diferentes surgem sequencialmente, exigindo controle de *catastrophic forgetting* e memória seletiva;
  - **Domain-Incremental Learning**: a distribuição de entrada P(X)P(X)P(X) muda enquanto o espaço de saída permanece fixo, característica típica do virtual drift [13].

- **Static (Batch) Learning Contexts**

Modelos estáticos são treinados uma única vez com um conjunto fixo de dados sob a suposição de que a distribuição dos dados será **estacionária**. Não há mecanismos automáticos de reavaliação ou adaptação, e qualquer retreinamento eventual ocorre manualmente, após longos períodos ou mudanças externas observadas [2, 3].

Esse paradigma ainda é comum em aplicações industriais ou operacionais onde os dados são escassos, as atualizações são custosas, ou onde não se espera grande variação no ambiente. No entanto, quando aplicado a domínios com mudanças reais nos padrões de dados, o modelo estático se torna rapidamente obsoleto — motivando a necessidade de detecção de concept drift como ferramenta diagnóstica [1, 4].

Além disso, muitos algoritmos clássicos de aprendizado supervisionado foram concebidos sob a hipótese de independência e distribuição idêntica (i.i.d.), o que reforça a importância de diferenciar esse regime dos demais [3].

- **Considerações sobre Regimes Intermediários**

  Embora a divisão entre aprendizado contínuo e aprendizado estático capture os extremos conceituais, há abordagens **intermediárias ou híbridas**, como:

  - *Retraining periódico* (batch re-training com janelas móveis),
  - *Adaptação por gatilhos de drift* (triggered retraining),
  - *Modelos adaptativos reponderados com detectores internos*.

  Na prática, essas abordagens operam como **instâncias discretas de aprendizado contínuo**, pois incluem mecanismos que **respondem dinamicamente à evolução dos dados**, mesmo que de forma esporádica ou controlada [2, 5, 13]. Por isso, são frequentemente tratadas pela literatura como pertencentes ao **paradigma ampliado de continual learning** [1, 2, 13].

  Assim, a distinção central permanece entre:

  - **Modelos que assumem estacionariedade** (estáticos), e

- **Modelos que, de forma ativa ou passiva, se adaptam à não-estacionariedade** (contínuos).

A Tabela 1 resume a taxonomia dos tipos de *concept drift*.

Tabela 1: Taxonomia Abrangente dos Tipos de *Concept Drift*

Drift Category	***Drift Type***	**Definição**	**Características/Indicadores Chave**	**Desafio Típico de Detecção**	**Exemplo(s) Ilustrativo(s)**
Fundamental Nature	*Real Drift*	Mudança na distribuição condicional da variável alvo dadas as características de entrada (P(Y	X) muda) [1, 4].	O limite de decisão muda; o conceito subjacente evolui.	Diferenciar do *virtual drift* se P(X) também muda [4].
Fundamental Nature	*Virtual Drift*	Mudança na distribuição de dados de entrada (P(X) muda), enquanto P(Y	X) permanece estável [1, 4].	O desempenho do modelo pode degradar devido ao encontro de novas regiões de entrada ou mudanças nos	Alta taxa de falsos alarmes se qualquer mudança em P(X) desencadear adaptação completa; ainda pode exigir atualização do modelo.

				priores de classe P(Y).	
Temporal Dynamics	*Sudden/Abrupt Drift*	O conceito antigo é rapidamente substituído por um novo em um ponto específico no tempo [1].	Mudança nítida e definitiva; novos dados rapidamente se tornam diferentes dos antigos.	Requer detecção rápida para minimizar o impacto.	Nova regulamentação entrando em vigor instantaneamente; crise financeira causando mudança abrupta no mercado; novo concorrente entrando no mercado.
Temporal Dynamics	*Gradual Drift*	Transição lenta e suave de um conceito antigo para um novo ao longo de um período extenso [1].	Ambos os conceitos podem coexistir durante a transição; mudanças sutis.	Difícil de detectar precocemente devido à sutileza; distinguir do ruído [9].	Evolução das preferências do cliente ao longo de meses; lenta degradação de máquinas; mudanças graduais no uso da linguagem.

Temporal Dynamics	*Incremental Drift*	Pequenas mudanças discretas se acumulam ao longo do tempo para formar um novo conceito [1].	Passos individuais podem ser menores; o efeito cumulativo é significativo.	Semelhante ao gradual; incrementos individuais podem ser difíceis de detectar.	Fraudadores fazendo alterações iterativas menores em suas táticas; lento acúmulo de novo jargão em mídias sociais.
Temporal Dynamics	*Recurring Drift*	Um conceito anteriormente ativo reaparece após algum tempo, frequentemente de forma cíclica ou sazonal [1].	O padrão pode ser previsível; oferece potencial para reutilizar modelos/conhecimento antigos [1].	Identificar o conceito recorrente correto e seu gatilho.	Comportamento de compra sazonal (por exemplo, vendas de feriados); padrões semanais de uso de transporte público; tendências de moda recorrentes.
Temporal Dynamics	*Local Drift*	O *concept drift* é limitado a um subconjunto do espaço de entrada.	Mudanças afetam apenas regiões específicas do espaço de características.	Pode ser difícil de detectar se o subconjunto afetado é pequeno ou raramente	Mudanças nas preferências de um nicho de usuários em um sistema de recomendação.

				amostrado.	
Causal Mechanism	*Performative Drift*	As previsões feitas pelo modelo implantado influenciam diretamente as futuras distribuições de dados que ele prevê [11].	Cria um *loop* de retroalimentação; as ações do modelo mudam o ambiente [11].	Distinguir do *drift* intrínseco; entender o impacto causal do modelo [11].	Sistemas de recomendação moldando as preferências do usuário; modelos de detecção de fraude fazendo com que fraudadores adaptem seus métodos; filtros de spam levando spammers a mudar táticas [11].

4. Características dos Dados que Justificam a Detecção de *Drift*

A necessidade de detecção de *concept drift* é particularmente premente em certos ambientes de dados, conforme descrito por autores como Lu et al. (2018) [2] e Žliobaitė (2010) [3].

- ***Non-stationary Data Distributions***: A característica mais fundamental dos dados que requerem detecção de *concept drift* é a não estacionariedade. Isso significa que o processo de geração de dados subjacente, formalmente representado pela distribuição de probabilidade conjunta P(x,y), muda ao longo do tempo [1]. Isso viola diretamente a suposição central de estacionariedade sobre a qual muitos modelos tradicionais de aprendizado de máquina são construídos.
- ***Streaming Data and High-Velocity Inputs***: Dados que chegam continuamente, muitas vezes em alta velocidade e em grandes volumes, são característicos de ambientes de *streaming* [2]. Tais fluxos de dados são potencialmente ilimitados em

tamanho, e elementos de dados individuais são normalmente processados sequencialmente. O influxo contínuo de novos dados nesses ambientes significa que há uma oportunidade constante para as distribuições subjacentes evoluírem. Além disso, a incapacidade de armazenar todos os dados históricos necessita do uso de métodos de aprendizado online ou adaptativos que podem atualizar modelos incrementalmente [1].

- **Evolving Statistical Properties (Feature Drift, Label Drift)**: Além da não estacionariedade geral, propriedades estatísticas específicas dos dados podem evoluir.
    - ***Feature Drift***: A distribuição de uma ou mais características de entrada muda ao longo do tempo. Por exemplo, a degradação do sensor pode alterar as características estatísticas das leituras que ele produz.
    - ***Label Drift***: A distribuição marginal da variável alvo P(Y) muda. Por exemplo, em um cenário de detecção de spam, a proporção geral de e-mails de spam em relação a e-mails legítimos pode aumentar significativamente ao longo do tempo.
- ***Open Systems***: Sistemas que interagem com um ambiente aberto e não controlado — que abrange a maioria das aplicações de ML do mundo real — são inerentemente mais propensos à não estacionariedade e, portanto, ao *concept drift* em comparação com configurações de laboratório fechadas e controladas [2]. Esses sistemas são constantemente influenciados por uma miríade de fatores externos que não são totalmente controláveis ou previsíveis (por exemplo, comportamento do usuário, tendências econômicas, condições climáticas).

5. Gatilhos, Categorias de Mudança e Condições que Levam à Ocorrência de *Drift*

Compreender as razões subjacentes pelas quais o *concept drift* ocorre é crucial para antecipar seu potencial início, selecionar métodos de detecção apropriados e projetar sistemas de aprendizado de máquina mais resilientes. Esses gatilhos, discutidos em trabalhos como os de Gama et al. (2014) [1] e Lu et al. (2018) [2], podem ser amplamente categorizados.

5.1. *External Environmental Changes*

Mudanças no ambiente mais amplo, externas ao sistema de ML específico, são uma fonte importante de *concept drift*. Estas estão frequentemente além do controle dos projetistas do sistema, mas podem ter impactos profundos nas distribuições de dados e nas relações aprendidas.

- ***Economic Shifts***: Flutuações na economia, como recessões, períodos de alto crescimento, mudanças significativas nas taxas de inflação ou mudanças nos níveis de desemprego, podem alterar o comportamento do consumidor e das empresas, induzindo assim *drift* em modelos relacionados a finanças, vendas e alocação de recursos [1].
- ***Societal Events & Trends***: Grandes eventos sociais, como pandemias globais (por exemplo, COVID-19), mudanças políticas significativas ou agitações, mudanças nas normas culturais e o surgimento de novos modismos de estilo de vida podem alterar drástica e, às vezes, subitamente os padrões de dados em vários domínios.
- ***Regulatory/Policy Changes***: A introdução de novas leis, regulamentações específicas do setor ou mudanças na política governamental podem necessitar de mudanças nos processos de negócios ou comportamentos individuais, levando ao *concept drift* nos modelos afetados.
- ***Natural Environment Changes***: Variações no ambiente natural, incluindo padrões climáticos sazonais, mudanças climáticas de longo prazo e a ocorrência de desastres naturais, podem desencadear *concept drift*, especialmente em modelos relacionados à agricultura, consumo de energia, logística e monitoramento ambiental.

5.2. *Behavioral Evolution*

O comportamento humano é inerentemente dinâmico e adaptativo. Mudanças em como os usuários interagem com sistemas, o que eles preferem, suas necessidades, motivações e até mesmo como se expressam (evolução da linguagem) podem ser impulsionadores significativos do *concept drift* [1, 5].

5.3. *System-Induced Changes*

Mudanças originadas de dentro do próprio sistema técnico ou de seus componentes imediatos de geração de dados também podem causar *concept drift*. Estas são, às vezes, mais controláveis ou previsíveis do que as mudanças ambientais externas.

- ***Sensor Degradation/Drift/Failure***: Sensores físicos usados em IoT, manufatura, saúde e monitoramento ambiental podem envelhecer, ficar desalinhados ao longo do tempo ou experimentar falha parcial ou completa. Essa degradação leva a mudanças nos dados que eles produzem, o que pode não refletir mudanças reais no processo subjacente sendo monitorado, mas, no entanto, causará *drift* em modelos que dependem desses dados de sensor.
- ***Updates in Data Pipelines or Feature Engineering***: Modificações em como os dados são coletados, pré-processados, agregados ou como as características são extraídas e projetadas podem alterar significativamente a distribuição de dados de

entrada para um modelo de ML, levando a *data drift* ou mesmo *concept drift* se as mudanças afetarem a relação entre características e o alvo.

- ***Software/System Updates***: Mudanças no sistema operacional subjacente, atualizações em bibliotecas de terceiros usadas no processamento de dados ou execução de modelos, ou atualizações de *firmware* em dispositivos que geram dados (por exemplo, dispositivos IoT, infraestrutura) podem inadvertidamente introduzir *drift*.

5.4. *Adversarial Dynamics*

Em muitos domínios de aplicação, particularmente cibersegurança, detecção de fraudes e filtragem de spam, o *concept drift* é ativa e intencionalmente induzido por adversários [7]. Esses adversários adaptam continuamente suas táticas e comportamentos para evadir a detecção por modelos de ML existentes ou para explorar vulnerabilidades do sistema.

5.5. *Seasonal and Cyclical Patterns*

Padrões previsíveis e repetitivos, frequentemente ligados a intervalos de tempo específicos (por exemplo, ciclos diários, semanais, mensais, anuais), podem causar *recurring concept drift* [1]. Embora sejam padrões de mudança, sua natureza recorrente os torna um tanto previsíveis se o ciclo for compreendido.

A Tabela 2 categoriza gatilhos comuns de *concept drift*.

Tabela 2: Gatilhos Comuns de *Concept Drift* e Suas Manifestações

Trigger Category	***Specific Trigger Example***	**Como Normalmente se Manifesta como Drift**	**Velocidade/Padrão Provável do Drift**	**Exemplo(s) de Domínio(s) de Aplicação Afetado(s)**
External Environmental Changes	*Economic Recession*	Altera P(Y	X) alterando capacidade/prioridades de compra; altera P(X) via mudanças na distribuição de renda.	Súbito ou Gradual

External Environmental Changes	*Major Societal Event* (ex: Pandemia)	Altera drasticamente P(X) (ex: mobilidade, atividade online) e P(Y	X) (ex: resultados de saúde, necessidades de compra).	Súbito, depois potencialmente Gradual
External Environmental Changes	*Regulatory/Policy Change*	Altera P(Y	X) redefinindo comportamento "compatível" ou "arriscado"; pode alterar P(X) se as regras de coleta de dados mudarem.	Súbito
External Environmental Changes	*Seasonal Weather Patterns*	Causa mudanças recorrentes em P(X) (ex: uso de energia) e P(Y	X) (ex: demanda por produtos sazonais).	Recorrente
Behavioral Evolution	*New Fashion Trend / Evolving Tastes*	Altera P(Y	X) à medida que as preferências por itens mudam; altera P(X) se novos segmentos de usuários adotarem a tendência.	Gradual ou Súbito (se viral)
Behavioral Evolution	*Changes in Online Expression (Slang, Sentiment)*	Altera P(Y	X) para modelos de análise de sentimento à medida que o mapeamento de características textuais para sentimento evolui.	Gradual, Incremental

System-Induced Changes	*Sensor Degradation/Wear*	Altera P(X) à medida que as leituras do sensor se tornam imprecisas ou enviesadas, afetando potencialmente P(Y	X) se o modelo depende delas.	Gradual, Incremental
System-Induced Changes	*Data Pipeline/Feature Engineering Update*	Altera P(X) mudando distribuições ou definições de características; pode alterar P(Y	X) se relações são afetadas.	Súbito (pós-atualização)
System-Induced Changes	*Software/Firmware Update*	Pode alterar P(X) alterando formato de dados, frequência ou mecanismos de registro de sistemas geradores de dados (infraestrutura, sensores).	Súbito (pós-atualização)	IoT, Sistemas Embarcados, Sistemas dependentes de dados de log
Adversarial Dynamics	*New Fraud Scheme / Malware Variant*	Altera P(Y	X) à medida que assinaturas de detecção antigas se tornam ineficazes; adversários criam novo X para parecer legítimo [7].	Súbito ou Incremental

Adversarial Dynamics	*Spammers Adapting Tactics*	Altera P(Y	X) para filtros de spam à medida que spammers alteram conteúdo/estrutura para contornar a detecção.	Incremental, Súbito
Seasonal and Cyclical Patterns	*Holiday Shopping Season*	Mudanças recorrentes em P(X) (ex: volumes de transação) e P(Y	X) (ex: risco de fraude aumentado).	Recorrente (Início súbito, declínio gradual)

6. Variações e Propostas de Nomenclaturas/Taxonomias

A literatura sobre *concept drift* apresenta diversas nomenclaturas e taxonomias para categorizar os diferentes tipos e aspectos do fenômeno [1, 2, 3, 4, 10]. Como visto anteriormente, as classificações mais comuns incluem a distinção entre *real* e *virtual drift* [4], e a categorização baseada na dinâmica temporal (*sudden*, *gradual*, *incremental*, *recurring*) [1]. Autores como Gama et al. (2014) [1], Žliobaitė (2010) [3], Webb et al. (2016) [4] e Lu et al. (2018) [2] são referências importantes que estabelecem e discutem essas tipologias.

Além das categorias fundamentais e temporais, o *performative drift* é introduzido como uma categoria baseada no mecanismo causal, onde as próprias previsões do modelo influenciam as distribuições de dados futuras [11]. Outras nuances na taxonomia podem incluir a severidade do *drift*, a área do espaço de características afetada (por exemplo, *local drift*), e se o *drift* é acompanhado por mudanças na dimensionalidade das características (*feature evolution*) [2]. A compreensão dessas variações taxonômicas é crucial para selecionar ou desenvolver métodos de detecção e adaptação apropriados para o tipo específico de *drift* encontrado em uma aplicação [1, 2].

Referências

1. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4), 44.

2. Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2018). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*.

https://doi.org/10.1109/TKDE.2018.2876857

3. Žliobaitė, I. (2010). Learning under concept drift: an overview. *arXiv:1010.4784*.

4. Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., & Petitjean, F. (2016). Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4), 964–994.

5. Singh, A., & Agrahari, S. (2022). Concept drift detection in data stream mining: A literature review. *Journal of King Saud University-Computer and Information Sciences*.

6. Bayram, F., Ahmed, B. S., & Kassler, A. (2022). From concept drift to model degradation: An overview on performance-aware drift detectors. *Knowledge-Based Systems*.

7. Shyaa, M. A., Al-Dabbagh, S., Al-Zuky, A. A., & Al-Saedi, J. (2024). A comprehensive survey on concept drift in intrusion detection. *Engineering Applications of Artificial Intelligence*.

8. Al-Badarneh, A., Al-Sayyed, R., & Al-Awadi, M. (2024). Evolving Strategies in Machine Learning: A Systematic Review of Concept Drift Adaptation and Detection Techniques. *Algorithms*, 15(12), 786.

9. Hoens, T. R., Chawla, N. V., & Polikar, R. (2012). Learning from streaming data with concept drift and imbalance. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*.

10. Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2024). One or two things we know about concept drift—a survey. *Frontiers in Artificial Intelligence*, 7.

11. Schliserman, P., Zhitomirsky-Geffet, M., & Lerman, K. (2024). Identifying Predictions That Influence the Future: Detecting Performative Concept Drift in Data Streams. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

12. Maltoni, D., Lomonaco, V., Pellegrini, L., Abati, D., & Calderara, S. (2019). Continual Learning in Artificial Neural Networks: An Empirical Evaluation. *IEEE Transactions on Neural Networks and Learning Systems*, 30(6), 2965–2976. https://doi.org/10.1109/TNNLS.2019.2947449

13. De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., & Tuytelaars, T. (2021). A Continual Learning Survey: Defying Forgetting in Classification Tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7), 3366–3385. https://doi.org/10.1109/TPAMI.2021.3057446

[Documento Aplicações citado no Termo de Aceite de Entrega de 14 de maio]

Aplicações Práticas da Detecção de Concept Drift

A seguir são descritos três estudos recentes que implementam mecanismos de detecção e adaptação ao concept drift em diferentes domínios. Em cada caso, o drift surge de características específicas do ambiente de dados, e as soluções propostas enfrentam esse desafio com abordagens direcionadas.

Consumo Energético Industrial com DMWM

Ambientes industriais frequentemente geram dados não estacionários devido a mudanças nas condições operacionais, desgaste de equipamentos, manutenção irregular e regimes de funcionamento recorrentes. Tais variações alteram a distribuição dos dados de entrada e saída, gerando concept drift real e virtual ao longo do tempo.

O modelo **DMWM (Dynamic Modeling with Memory)** foi desenvolvido para prever consumo energético em tempo real, detectando concept drift a partir da variação na taxa de erro entre blocos de dados sequenciais, definidos com base nos períodos de inatividade das máquinas [3]. Quando identificado um aumento de erro, o modelo avalia se há um conceito recorrente — reutilizando modelos previamente treinados — ou realiza novo treinamento com redes neurais profundas.

A aplicação do DMWM em três cenários industriais (mineração, siderurgia e ventilação subterrânea) resultou em ganhos práticos expressivos: redução do erro (MAPE) de até 8,63% para cerca de 5%, e queda no tempo de treinamento graças ao reuso de modelos anteriores em casos de drift recorrente [3].

Séries Temporais Online com Hiato Temporal no Proceed

Em previsões online, como em sistemas de séries temporais, pode haver um hiato entre os dados de treino mais recentes e os dados atuais de teste. Durante esse intervalo, alterações sutis ou abruptas nos padrões dos dados — como mudanças de estação, comportamento do usuário ou eventos externos — podem causar concept drift.

A framework **Proceed** endereça esse problema ao estimar proativamente o drift entre os dados mais recentes e a instância atual a ser prevista, usando codificadores de conceito e

um gerador de adaptação treinado com drifts sintéticos diversos [1, 9]. Assim, ajusta os parâmetros do modelo antes mesmo da predição, sem depender de feedback posterior.

Esse método foi avaliado em cinco conjuntos de dados reais e demonstrou melhora significativa no desempenho preditivo, reduzindo o erro médio em 21,9% e superando em 10,9% os métodos online do estado da arte [9]. Seu foco no drift induzido por atrasos temporais o torna especialmente útil em aplicações de streaming.

Previsão de Crimes em Fluxos Espácio-Temporais com AWPEM

A distribuição dos crimes em contextos urbanos varia conforme fatores como horário, localização, clima, eventos locais ou mudanças socioeconômicas. Essa natureza espácio-temporal e evolutiva dos dados gera concept drifts tanto graduais quanto cíclicos.

A abordagem **AWPEM (Average Weighted Performance Ensemble Model)** utiliza um ensemble de modelos cujo peso é ajustado dinamicamente com base em seu desempenho recente [12]. Embora a sinalização explícita do drift seja implícita, o mecanismo de ponderação permite adaptação contínua conforme os padrões de criminalidade mudam.

Testado em dois conjuntos de dados reais, o AWPEM obteve maior precisão na previsão espácio-temporal de crimes quando comparado com métodos anteriores, sendo eficaz na adaptação a tendências recorrentes e mudanças graduais [12].

Referências

[1] Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2018). Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*.

[3] Heo, J., Kim, S., et al. (2023). Dynamic modeling with memory for energy forecasting. *Applied Energy*, 342.

[9] Ye, J., Chen, M., et al. (2022). Proceed: Proactive concept drift adaptation in online time series forecasting. *Proceedings of the 28th ACM SIGKDD Conference*.

[12] Angbera, J., & Chan, J. (2021). Utilizing an ensemble machine learning framework for handling concept drift in spatiotemporal data streams classification. *Expert Systems with Applications*.

[Documento Aplicação NLP citado no Termo de Aceite de Entrega de 14 de maio]

O Desafio Contínuo do Concept Drift em Dados Textuais para Modelos de Linguagem de Grande Porte (LLMs)

Subtítulo: Da Evolução Semântica Histórica à Detecção em Aplicações Modernas de PLN

Introdução: A Natureza Dinâmica da Linguagem

A linguagem humana é uma entidade viva, em constante evolução. O significado das palavras e conceitos, ou *concept drift* (também referido como *diachronic semantic shift*), transforma-se ao longo do tempo, refletindo mudanças no consenso social, cultural e tecnológico [5]. Um estudo que analisou 150 anos de literatura inglesa, entre 1800 e 1950, ilustra vividamente este fenômeno [5, p. 1]. Por exemplo, a palavra "gay", que originalmente significava "alegre", "animado" ou "brilhante", passou a ser predominantemente usada para descrever homens homossexuais a partir da década de 1960 [5, p. 1]. Similarmente, o estudo de Li et al. (2021) demonstrou, através da análise do contexto de palavras (os K vizinhos mais próximos no espaço de *embedding*), que o termo "peace" sofreu uma alteração drástica no seu significado percebido entre 1890 e 1930. Inicialmente associado frequentemente a "war", seu contexto semântico após a Segunda Guerra Mundial deslocou-se para termos como "spirit", "humanity", "tyranny" e "poverty" [5, p. 6-8]. Outro exemplo notável é a palavra "bishop", cujo contexto em 1910 estava ligado a termos religiosos, enquanto em 1950 se aproximou de termos relacionados ao xadrez [5, p. 8]. Essas mudanças, identificadas pela comparação da similaridade de Jaccard entre os contextos das palavras em diferentes períodos temporais [5, p. 4], sublinham que o *concept drift* é uma característica intrínseca da linguagem. Este mesmo desafio de mutabilidade impacta profundamente a eficácia e relevância dos Modelos de Linguagem de Grande Porte (LLMs) contemporâneos, que são treinados em vastos corpora textuais e podem rapidamente se tornar dessincronizados com a evolução da linguagem e do conhecimento.

Manifestações do Concept Drift em LLMs e Dados Textuais Atuais

O *concept drift* em dados textuais atuais, que afeta diretamente os LLMs, pode ser categorizado em *real drift*, onde a relação entre o texto e seu significado ou rótulo muda (e.g., a alteração na definição de termos como "desinformação" ou "assédio" ao longo do

tempo, afetando a classificação de textos [3, p. 2]), e *virtual drift* (ou *data drift*), onde a distribuição das características do texto em si se altera (e.g., o surgimento de novas palavras, gírias, *hashtags* em redes sociais [1, p. 5], ou mudanças em estilos de escrita e construções gramaticais [3, p. 2]).

Esses *drifts* podem ocorrer seguindo diferentes padrões temporais, como *sudden/abrupt drift*, *incremental drift*, *gradual drift* ou *recurrent drift* [1, 2]. Um exemplo de *sudden drift* é a mudança abrupta nos tópicos de discussão em redes sociais, como a que ocorreu com o surgimento da COVID-19, impactando classificadores de tópicos de tweets [2, p. 2]. O *incremental drift* pode ser observado em classificadores de imagem para veículos autônomos que gradualmente encontram novos tipos de veículos, como patinetes elétricos, que não faziam parte dos dados de treinamento originais e cuja frequência aumenta com o tempo [2, p. 2]. Já o *recurrent drift* é exemplificado por classificadores de sentimento ou tópico que lidam com eventos sazonais ou discussões cíclicas, como as que ocorrem em anos eleitorais, onde a linguagem e os temas em mídias sociais sofrem alterações significativas, mas podem retornar a um estado anterior ou similar após o evento [2, p. 2]. A constante aparição de novas palavras e *hashtags* em plataformas como o Twitter também pode ser vista como um tipo de *feature drift* [1, p. 5].

Desafios e Casos de Detecção de Concept Drift em NLP e LLMs

A detecção dessas mudanças é crucial, especialmente para LLMs que são cada vez mais integrados em aplicações dinâmicas.

Margatina et al. (2023) abordam diretamente a obsolescência factual em modelos de linguagem mascarada (MLMs) devido ao *concept drift* temporal [4]. Eles exemplificam o problema ao consultar diferentes MLMs sobre o Primeiro Ministro do Reino Unido: modelos mais antigos ou não atualizados temporalmente fornecem respostas como "Cameron" ou "May", que, embora corretas no passado, são factualmente incorretas no momento da avaliação posterior, quando "Boris Johnson" (e subsequentemente outros) ocuparam o cargo [4, p. 1-2]. Para enfrentar essa questão, o estudo propõe o DYNAMICTEMPLAMA, um *framework* que gera dinamicamente conjuntos de teste temporais a partir do Wikidata. Esses conjuntos podem ser segmentados por granularidade temporal (mensal, trimestral, anual) e por tipo de mudança factual (DUNCHANGED, Dt+1UPDATED, Dt+1NEW, Dt+1DELETED) [4, p. 3-4]. Utilizando este *framework*, eles avaliaram os TIMELMs (modelos RoBERTa treinados trimestralmente com dados do Twitter), analisando a capacidade dos modelos de preservar conhecimento e se adaptar a novas informações factuais [4, p. 5].

No domínio das redes sociais, Bechini et al. (2021) investigaram o *event-driven concept drift* na tarefa de *stance detection* (detecção de postura) em *streams* de tweets relacionados à

vacinação na Itália [1]. Eventos do mundo real, como notícias sobre a incidência de doenças, declarações políticas ou discussões sobre a obrigatoriedade da vacina para crianças, foram identificados como gatilhos para picos no volume de tweets e, consequentemente, para mudanças na distribuição dos dados e na forma como as posturas eram expressas [1, p. 1-2, 8]. O estudo demonstrou que modelos de classificação estáticos perdem desempenho rapidamente. Para mitigar isso, foi proposto um esquema de aprendizado semântico ("Semantic scheme") que utiliza *embeddings* BERT para calcular a similaridade semântica entre os tweets de diferentes eventos. Essa similaridade é então usada para ponderar a importância dos dados de treinamento de eventos passados ao atualizar o modelo para o evento corrente, mostrando-se mais eficaz que abordagens de retreinamento simples ou baseadas apenas na recência dos dados [1, p. 6-7].

Feldhans et al. (2021) focaram na detecção de *drift* em representações textuais utilizando *embeddings* de documentos (BERT e Word2Vec) [3]. Em seus experimentos com o dataset Amazon Movie Reviews, o *drift* foi induzido artificialmente pela injeção gradual de adjetivos negativos nos textos das resenhas [3, p. 5-6]. No dataset de tweets sobre as eleições dos EUA de 2020, o *drift* foi analisado em torno de eventos reais, como o último debate televisivo e o dia da eleição, que naturalmente alteraram a distribuição dos tópicos e da linguagem utilizada [3, p. 4-5]. A comparação entre diferentes detectores de *drift* (KTS, LSDD, KS) revelou que abordagens multivariadas como KTS e LSDD geralmente superam os métodos univariados, e que a detecção pode ser mais eficaz em *embeddings* de menor dimensionalidade [3, p. 1, 7].

Para uma detecção não supervisionada e em tempo real de *concept drift* em dados não estruturados, Greco et al. (2024) desenvolveram o *framework* DRIFTLENS [2]. Este método opera sobre as representações internas (embeddings) geradas por modelos de *deep learning*. O DRIFTLENS modela as distribuições desses *embeddings* como normais multivariadas, tanto para o *batch* de dados como um todo (per-batch) quanto para cada classe predita individualmente (per-label). A Distância de Fréchet (FDD) é então empregada para quantificar a divergência entre uma distribuição de referência (baseline) e as distribuições de novas janelas de dados que chegam [2, p. 4-5]. O sistema demonstrou capacidade de detectar diversos padrões de *drift* (*sudden, incremental, periodic*), além de caracterizar o *drift* ao identificar os rótulos mais afetados, tudo isso com uma eficiência computacional que permite sua aplicação em tempo real [2, p. 1, 9-10].

Retomando a análise de Li et al. (2021) sobre textos históricos, o método proposto consistiu em dividir o corpus de 150 anos de literatura inglesa em períodos de 20 anos, treinar modelos word2vec para cada período e, então, calcular o contexto de cada palavra (os K vizinhos mais similares no espaço de *embedding*). A mudança no significado era medida pela similaridade de Jaccard entre os conjuntos de palavras de contexto de um mesmo

termo em diferentes períodos [5, p. 3-4]. Isso permitiu identificar palavras com alta dinâmica semântica e visualizar essas transições [5, p. 6-8].

Impactos da Não Detecção

A ausência de detecção e adaptação ao *concept drift* compromete seriamente a utilidade dos LLMs:

- **Degradação do Desempenho:** Modelos de classificação de texto, como os de *stance detection*, podem ter sua performance severamente degradada quando o *drift* não é considerado [1]. Similarmente, a precisão geral de modelos de PLN diminui [2, 3].
- **Obsolescência Factual:** LLMs podem fornecer informações factualmente incorretas ou desatualizadas, como demonstrado pelas respostas desatualizadas de MLMs sobre o Primeiro Ministro do Reino Unido [4, p. 1-2].
- **Incompreensão de Novos Usos Linguísticos:** O surgimento de novos termos, gírias ou a ressignificação de palavras (e.g., "gay", "peace", "bishop" ao longo da história [5]) pode levar a falhas de interpretação e resposta por parte dos LLMs se não houver adaptação [1, 3].
- **Perda de Confiabilidade e Robustez:** A incapacidade de um LLM de se manter atualizado e preciso frente às mudanças linguísticas e factuais mina sua confiabilidade geral e robustez em aplicações de longo prazo [2, 4].

O Papel da Monitoração Contínua

A monitoração contínua é indispensável para mitigar os efeitos do *concept drift*.

- **Detecção Precoce:** Identificar o *drift* assim que ele ocorre é crucial. Métodos como o DRIFTLENS são projetados para operar em tempo real, fornecendo alertas antecipados [2, p. 1].
- **Caracterização do Drift:** Além de detectar, é importante entender a natureza do *drift*. O DRIFTLENS, por exemplo, permite analisar quais rótulos são mais afetados [2, p. 6], e o DYNAMICTEMPLAMA permite analisar *drifts* em fatos novos, atualizados ou inalterados [4, p. 4].
- **Avaliação Dinâmica:** É fundamental avaliar os LLMs continuamente com dados que reflitam a evolução temporal. O *framework* DYNAMICTEMPLAMA serve a esse propósito, permitindo testar modelos contra fatos que mudam ao longo do tempo [4]. A análise de fluxos de dados, como no estudo de *stance detection* no Twitter,

também se baseia na avaliação contínua do desempenho do modelo em novos "pedaços" de dados (chunks) [1, p. 5].

Estratégias de Adaptação e Mitigação

Os artigos fornecidos mencionam algumas abordagens para lidar com o *drift* uma vez detectado:

- **Retreinamento do Modelo (*Retrain Scheme*):** Consiste em retreinar o modelo periodicamente. No estudo de Bechini et al. (2021), isso envolvia estender o conjunto de treinamento com novos dados rotulados de cada evento e treinar novamente o classificador [1, p. 5].
- **Adaptação Baseada em Similaridade Semântica:** A abordagem "Semantic scheme" de Bechini et al. (2021) utiliza *embeddings* BERT para ponderar a relevância de dados de treinamento passados com base na sua similaridade semântica com o evento atual, oferecendo uma alternativa aos mecanismos de esquecimento tradicionais [1, p. 6].
- **Ensembles Adaptativos:** O sistema DARK, mencionado por Bechini et al. (2021) como estado da arte, utiliza um *ensemble* de classificadores dinamicamente ponderado com mecanismos de esquecimento [1, p. 3, 8].
- **Treinamento Contínuo de LLMs:** Os TIMELMs, avaliados por Margatina et al. (2023), são um exemplo prático de adaptação contínua, onde modelos RoBERTa são inicializados de um *checkpoint* anterior e continuam seu treinamento com dados de trimestres subsequentes para incorporar conhecimento temporal mais recente [4, p. 5].

Referências

[1] Bechini, A., Bondielli, A., Ducange, P., Marcelloni, F., & Renda, A. (2021). Addressing Event-Driven Concept Drift in Twitter Stream: A Stance Detection Application. IEEE Access, 9, 77758-77770.
[2] Greco, S., Apiletti, D., Vacchetti, B., & Cerquitelli, T. (2024). Unsupervised Concept Drift Detection from Deep Learning Representations in Real-time. arXiv preprint arXiv:2406.17813.

[3] Feldhans, R., Wilke, A., Heindorf, S., Shaker, M. H., Hammer, B., Ngonga Ngomo, A.-C., & Hüllermeier, E. (2021). Drift Detection in Text Data with Document Embeddings. In Lecture Notes in Computer Science.
[4] Margatina, K., Wang, S., Vyas, Y., John, N. A., Benajiba, Y., & Ballesteros, M. (2023). Dynamic Benchmarking of Masked Language Models on Temporal Concept Drift with Multiple Views. arXiv preprint arXiv:2302.12297.
[5] Li, R., Tian, P., & Wang, S. (2021). Study concept drift in 150-year English literature. In 1st Workshop on AI + Informetrics - AII2021.

APÊNDICE 5

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 21 de mai. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

MATHEUS FARES COSTA BRAKES

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Foram revisados diversos artigos e bases de dados para orientar os testes práticos. Dentre eles, selecionei o paper ChronosLex: Time-aware Incremental Training for Temporal Generalization of Legal Classification Tasks , que embora não tenha foco exclusivo em detecção de drift, apresenta datasets públicos (MultiEURLEX e ECtHR) e fornece a base teórica para as análises iniciais.

Também criei um Jupyter Notebook que reúne todo o pipeline de detecção de concept drift no domínio jurídico. Nele, começamos pelo mesmo pré-processamento utilizado no paper de Chalkidis et al. (extração de anos, filtragem e formação de janelas temporais no MultiEURLEX e no ECtHR) e aplicamos o cálculo offline de drift lexical via divergência de Jensen–Shannon. Em seguida, adicionamos uma camada de detecção online de drift semântico: cada documento ou janela de documentos é convertido em embedding pelo SentenceTransformer e alimenta três detectores do River (ADWIN, Page–Hinkley e KSWIN) sobre janelas deslizantes, de modo a sinalizar mudanças no uso e no contexto dos termos em tempo real. Link do código : https://drive.google.com/file/d/1-Tp44HRyxmuLzUOqti6meIfXG4GQacaE/view?usp=drive_link

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Os próximos passos serão otimizar o pipeline de embeddings usando modelos maiores, ajustar a sensibilidade dos detectores, experimentar janelas quinquenais e janelas deslizantes com overlap, e estudar e aplicar as estratégias de adaptação incremental sugeridas no paper para validar seu impacto na detecção de drift e no desempenho em tarefas jurídicas.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

[Artigo **ChronosLex: Time-aware Incremental Training for Temporal Generalization of Legal Classification Tasks**, que serviu de base para a geração dos experimentos]

ChronosLex: Time-aware Incremental Training for Temporal Generalization of Legal Classification Tasks

Santosh T.Y.S.S and **Tuan-Quang Vuong** and **Matthias Grabmair**
School of Computation, Information, and Technology
Technical University of Munich, Germany
{santosh.tokala, quang.vuong, matthias.grabmair}@tum.de

Abstract

This study investigates the challenges posed by the dynamic nature of legal multi-label text classification tasks, where legal concepts evolve over time. Existing models often overlook the temporal dimension in their training process, leading to suboptimal performance of those models over time, as they treat training data as a single homogeneous block. To address this, we introduce ChronosLex, an incremental training paradigm that trains models on chronological splits, preserving the temporal order of the data. However, this incremental approach raises concerns about overfitting to recent data, prompting an assessment of mitigation strategies using continual learning and temporal invariant methods. Our experimental results over six legal multi-label text classification datasets reveal that continual learning methods prove effective in preventing overfitting thereby enhancing temporal generalizability, while temporal invariant methods struggle to capture these dynamics of temporal shifts.

1 Introduction

Legal classification tasks involve the assignment of legal documents, texts, or cases to specific legal categories, making them essential for legal practitioners, researchers, and organizations seeking efficient legal document management and analysis. However, the dynamic nature of legal domain, characterized by the intricate interplay of laws, precedents, and ever-evolving interpretation of existing jurisprudence, influenced by external forces such as real-world events and shifting societal norms, provides a distinctive challenge for these legal tasks specifically. In this backdrop of a dynamic and non-stationary world, temporal generalization of models emerges as a key necessity, given such systems are deployed to assist users with data that it encounters in the future, whilst being trained on data from the past.

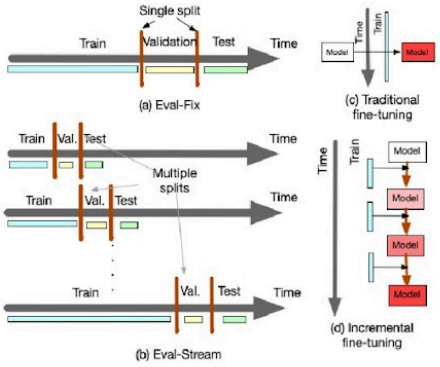


Figure 1: Different evaluation and training strategies employed in this work, to assess and improve temporal generalization of models for legal classification tasks.

Most of the works dealing with legal classification often neglect time as a factor and simply split the data randomly into training, validation, and test sets (e.g., Paul et al. 2020; Malik et al. 2021; Zheng et al. 2021; Tuggener et al. 2020; Hendrycks et al. 2021; Papaloukas et al. 2021; Lippi et al. 2019; Păiș et al. 2021; Luz de Araujo et al. 2018; Şulea et al. 2017 inter alia). Recent works have realized that this standard splitting criterion leads to overly optimistic performance estimates, which are far from realistic scenarios when such systems need to be deployed to assist with the new content over time and suffer from degradation of performance due to temporal misalignment i.e., difference in distributions of training and evaluation data instances. They advocate for chronological splitting of datasets to estimate the performance, accounting for the data drift over time (e.g., Chalkidis et al. 2021b, 2022, 2021a; Xu et al. 2023; Niklaus et al. 2021; Chalkidis and Søgaard 2022 inter alia). Furthermore, we posit that using a single fixed chronological test split (Eval-Fix, Fig. 1a) may yield misleading results

if the chosen fixed split coincides with anomalous events, such as Brexit, Covid-19, lockdown, the Russo-Ukrainian War, or political regime changes, impacting one side of the splits more than the other. To address this concern, we utilize a streaming evaluation protocol (Eval-Stream, Fig. 1b) which involves evaluating the model over multiple splits involving distinct time periods which can lead to reliable conclusions about models' robustness. In this protocol, we incorporate all past data into training and assess the model's capacity to adapt to emerging trends in the next time period, which mirrors standard machine learning development pipelines, where models are regularly updated and redeployed to account for temporal drifts (Yao et al., 2022).

While legal texts are intrinsically tied to the time of their origin, making it imperative to account for these temporal aspects when developing classification models, these previous methods overlook the chronological context in which legal documents exist during the training process and consider the entire training data spanning across a considerable stretch of time, as a single homogeneous block (Fig. 1c). This oversight hinders model capabilities to adapt to the data drifts that naturally occur over time leading to performance degradation when evaluated over data from the future time (Gaspers et al., 2022; Rijhwani and Preoţiuc-Pietro, 2020).

For instance, shifts in societal attitudes towards fundamental rights like privacy, discrimination, and freedom of expression drive the evolution of legal concepts, broadening or contracting their scope over time. Influenced by technological advances and global events, national legal concepts expand to encompass emerging trends, as seen in security related matters to encompass artificial intelligence and cryptocurrency. Landmark cases can also introduce new civil/human rights, such as the recognition of environmental rights within the right to life. Legislative changes, such as developments in data protection laws keeping pace with technology, further shape concepts, reflecting evolving regulatory needs. Overall, data drifts in the legal domain can manifest gradually through the gradual evolution of legal concepts , reflecting societal changes, emerging legal trends, and real-world events or even abruptly through landmark legal decisions, overruling precedents, and legislative changes.

In response to these dynamic temporal drifts, we put forth a hypothesis which we subsequently validate: Models trained on data from a timeframe closer to the test set tend to yield superior results, under the assumption of the same model and dataset size. Motivated by this finding, and with the aim to capitalize on the whole training set — typically associated with better performance with more data, we introduce "ChronosLex", an incremental training framework (Fig. 1d). In this approach, the model is systematically trained on data from one time period at a time before progressing to the next chronological split, thus preserving the temporal order of the data the model interacts with during training. Our experiments on six multi-label legal classification datasets demonstrate that incremental training paradigm improves temporal generalization of these models on three datasets but also leads to overfitting to recent data on other datasets.

To further improve the models' robustness to address the overfitting issue, we assess (i) different families of continual learning algorithms such as regularization-based EWC (Kirkpatrick et al., 2017), rehearsal-based ER (Chaudhry et al., 2019), AGEM (Lopez-Paz and Ranzato, 2017), and parameter expansion based Adapters (Houlsby et al., 2019), LoRA (Hu et al., 2021) (ii) temporally invariant methods (Yao et al., 2022) derived from domain adaptation methods such as GroupDRO (Sagawa et al., 2019), DeepCORAL (Sun and Saenko, 2016), IRM (Arjovsky et al., 2019) where we treat consecutive windows of time as distinct domains. Our experimental results suggest that temporal invariant methods fail to learn these dynamics while continual learning methods effectively prevent models from overfitting to recent data, ultimately enhancing temporal generalization.

2 Related Work

Temporal Drift There has been increasing recent evidence that temporal misalignment results in degradation of model performance due to difference in distribution between the train and test data, caused due to temporal shifts (Jaidka et al., 2018; Yao et al., 2022; Gorman and Bedrick, 2019). These distribution shifts arise with the passage of time and are a very naturally occurring type of distribution shift due to non-stationarity and evolving world (Schlimmer and Granger, 1986; Widmer and Kubat, 1993). Temporal misalignment can be caused due to (1) the dynamic nature of language (Rosin et al., 2022; Röttger and Pierrehumbert, 2021; Loureiro et al., 2022; Agarwal and Nenkova, 2022; Amba Hombaiah et al., 2021; Ri-

jhwani and Preoţiuc-Pietro, 2020; Luu et al., 2022; Jaidka et al., 2018) and (2) the update of factual information (Margatina et al., 2023; Jang et al., 2021, 2022; Lazaridou et al., 2021; Dhingra et al., 2022; Liska et al., 2022).

Temporal generalization has been explored both in upstream Language Model pre-training (Lazaridou et al., 2021; Loureiro et al., 2022; Jang et al., 2021, 2022; Dhingra et al., 2022; Jin et al., 2022; Amba Hombaiah et al., 2021) and in downstream tasks, such as sentiment analysis (Lukes and Søgaard, 2018; Agarwal and Nenkova, 2022; Guo et al., 2023b), named entity recognition (Rijhwani and Preoţiuc-Pietro, 2020; Onoe et al., 2022), question answering (Liska et al., 2022; Shang et al., 2022), headline generation (Søgaard et al., 2021), rumor/fake news detection (Mu et al., 2023; Hu et al., 2023), spoken language understanding (Gaspers et al., 2022). model explainability (Zhao et al., 2022), document classification (Röttger and Pierrehumbert, 2021; Chalkidis and Søgaard, 2022; Huang and Paul, 2018), abusive language detection (Jin et al., 2023; Florio et al., 2020), topic modeling (Zhang et al., 2023) and readmission prediction in the health care context (Guo et al., 2022, 2023a).

In this work, we assess temporal generalization in multi-label legal text classification, which is challenging given the complexities of managing both the multiplicity of labels and accommodating the evolving legal context over time. This is in contrast to prior studies that predominantly focused on single-label classification. In prior work on temporal generalization in multi-label legal text classification, Chalkidis and Søgaard 2022 postulated that temporal drift is primarily attributable to shifts in the label distribution over time and explored various group-robust optimization algorithms to mitigate disparities at the label level. We propose an alternative viewpoint, arguing that temporal drift in legal tasks extends beyond label distribution shifts to encompass shifts in the input text as well, driven by the fact that the vocabulary changes with time (Tab. 1). Furthermore, even vocabulary specific to each label undergoes temporal shifts (Tab 1). This complexity was not fully considered in Chalkidis and Søgaard 2022, where the entire training data was treated as a single unit, overlooking shifts in input text distribution over time. To address this, we introduce an incremental training framework which enables models to adapt to distribution shifts over time. We further improve its robustness by evaluating a range of continual learning and temporal invariant approaches for multi-label legal text classification.

Continual Learning Most research in continual learning (or lifelong learning) initially centered around computer vision tasks, and more recently, it has gained attention in the NLP field (Ke et al., 2021b,a; Biesialska et al., 2020; Ke and Liu, 2022; Sun et al., 2019; Wang et al., 2019). The majority of these works adopted traditional task-incremental, domain-incremental, or label-incremental settings. While our temporal incremental setting bears resemblance to the domain-incremental setting, a crucial distinction lies in the assumption of strict boundaries in domain incremental settings, which is not applicable to our temporal adaptation where the boundaries between drifts are blurred (Prabhu et al., 2020; Aljundi et al., 2019). Generally, continual learning algorithms can be categorized into (i) Rehearsal-based methods (Rolnick et al., 2019; Rebuffi et al., 2017), which maintain a memory buffer of older data to perform experience replay with actual data (de Masson D'Autume et al., 2019), automatically generated data (Sun et al., 2019), or previously computed gradients (Lopez-Paz and Ranzato, 2017), (ii) Regularization-based approaches (Kirkpatrick et al., 2017; Chen et al., 2020; Huang et al., 2021), which regularizes neural network parameters from drastic updates for new information to preserve the information of older ones, preventing overfitting to the newer data (iii) Parameter-expansion methods (Qin et al., 2022; Gururangan et al., 2022; Yoon et al., 2018) which freeze network architectures on older data and dynamically grow branches for newer ones. These methods have been explored in the pre-training stage to accommodate factual updates (Jang et al., 2021, 2022) or adaptation to new domains (Jin et al., 2022; Gururangan et al., 2022) and in the fine-tuning stage (Yao et al., 2022; Lin et al., 2022), which is closely related to our work, but within the context of single-label classification tasks. To the best of our knowledge, continual methods have never been explored for multi-label legal classification tasks.

Temporal invariant Domain Adaptation Domain adaptation (DA) primarily addresses the co-variate shift between source and target data distributions (Ruder, 2019). This involves learning domain-invariant feature representations that can generalize across different domains (Bollegala et al., 2011; Ganin et al., 2016). In the legal context, DA meth-

ods have been applied to tasks such as Legal Judgement Prediction, where reasoning with respect to each article is treated as a domain (Tyss et al., 2023) and legal text classification tasks, where each label is considered a domain to address label imbalance (Chalkidis and Søgaard, 2022). Traditional DA methods are typically applied to domains with clear distinctions, whereas temporal distribution shifts tend to occur gradually without well-defined boundaries, forming a continuous process. To adapt domain invariant methods to address temporal distribution shifts, we assume instances from consecutive windows of time to constitute a distinct domain, enabling the application of invariant learning approaches to these constructed domains (Yao et al., 2022). To the best of our knowledge, this is the first attempt to assess these methods for multi-label legal texts to handle temporal drift.

3 Evaluation Strategy

To assess temporal generalizability, following previous works (Søgaard et al., 2021; Chalkidis et al., 2022), we carry out evaluation using fixed chronological splits, referred to as **Eval-Fix** (Fig. 1a). In this setup, we split the entire data chronologically into train ($d_{<t_1}$), validation ($d_{\geq t_1 \& \leq t_2}$), and test ($d_{>t_2}$) splits, where d_t denotes data from time period t, and $t_1 < t_2$. We train and validate the model using the respective splits and subsequently report its performance over the entire test set.

We posit that such a fixed split may not reflect a true picture as it can be influenced by the interplay between the splits and the data distribution where the anomalous/drift-driven events may overlap or primarily fall within one of the splits, leading to misleading conclusions about the model's ability to handle temporal drifts. To mitigate the impact of dataset-specific traits on the splits, we adopt a streaming evaluation protocol, inspired by Yao et al. 2022 and Lopez-Paz and Ranzato 2017, which we refer to as **Eval-Stream** (Fig. 1b). We evaluate models using multiple splits, formed at distinct timestamps, considering all the data up to time period t ($d_{\leq t}$) for training, d_t for validation, and d_{t+1} as the test split. We focus only on the evaluation of the future data rather than the past because the practical need for performance over past data is limited. While it is possible to test the model on data from all subsequent time periods ($d_{>t}$) at each time period t, we restrict the testing to the following time period $t + 1$, to simplify the result analysis, aligning with practical scenarios where models are updated based on their performance in the next time iteration.

4 Datasets

We conduct our experiments on the six multi-label legal classification datasets from three sources.

UKLEX (Chalkidis and Søgaard, 2022) This dataset comprises legislative documents of the UK, publicly accessible through the National Archives, which are typically categorized into thematic areas such as healthcare, finance, education, transportation and planning, which are outlined in the document preamble and serve as indexes for archival purposes. The dataset contains 36.5k documents and is chronologically split into training (20k, 1975–2002), validation (8.5k, 2002–2008), and test (8.5k, 2009–2018) sets. The labels are provided at two distinct levels of granularity, encompassing 18 and 69 topics, referred to as Small (S) and Medium (M) respectively. For *eval-fix*, we use the above splits and for *eval-stream*, we consider a two-year time period as a unit. Due to limited data prior to 1990 and to ensure an adequate number of data instances for model learning, we report eval-stream performance from 1992.

EURLEX (Chalkidis et al., 2021a) This dataset contains EU legislation documents accessible via the EUR-Lex platform and is annotated using concepts from EuroVoc, a thesaurus maintained by the EU's Publications Office. We work with the English portion of this dataset, consisting of 65k documents and split chronologically into training (55k, 1958–2010), validation (5k, 2010–2012), and test (5k, 2012–2015) sets which we use for *eval-fix*. The dataset provides four levels of label granularity. We use the first two levels of the EuroVoc taxonomy, as followed in Chalkidis and Søgaard 2022, encompassing 21 and 127 concepts, denoted as Small (S) and Medium (M), respectively. For *eval-stream*, we use two years as one unit and report performance from 1987 due to a lower number of instances in the initial years.

ECHR The dataset by (Chalkidis et al., 2019, 2021b) comprises cases heard by the European Court of Human Rights, which are publicly accessible via HUDOC, the official court database. These cases involve the adjudication of complaints by individuals against states for alleged violations of their rights as enshrined in the European Convention of Human Rights. Each case includes in-

formation about the convention articles that have been alleged to be violated and which the court has found to be violated. The dentification of alleged and violated articles are referred to as Task B and A, respectively, by Chalkidis et al. 2022. It consists of 11K cases, divided chronologically into training (9k, 2001–2016), validation (1k, 2016–2017), and test (1k, 2017–2019) sets for *eval-fix*. We use the 14 and 17 articles related to the core rights as followed in Valvoda et al. 2023 for Task A and B respectively. We use annual data as a unit for *eval-stream* and report the performance from 2004.

While task employed in this work are all effectively text classification, the tasks surveyed here are different in nature. The ECHR tasks represent legal determinations by judges in specific cases of high complexity and semantic depth, whereas the keyword classification tasks of EURLEX and UKLEX represent semantically much shallower analysis. This has consequences for the types and complexity of temporal shifts one would observe. For example, the court changing its jurisprudence on a particular legal issue prompted by world events is a more complex phenomenon than the usage of particular European Law Database Keyword.

5 ChronosLex

Previous approaches for legal classification typically fine-tune models on the entire training dataset in a shuffled manner, treating all data as a single homogeneous block. We argue that such an approach neglects the temporal nature of the data which may be crucial to learn the temporal evolution of concepts. To address this limitation, we propose the ChronosLex framework, which considers the chronological context of data to capture and adapt models to the natural drifts over time. We hypothesize that recent data holds insights into upcoming distributional shifts and training models with this recent data enables better adaptation to temporal changes. Thus, we systematically train the model with data from one time period at a time before progressing to the next chronological split. Specifically, at time t, the model m_t is initialized with the model obtained from the previous timestamp m_{t-1} and fine-tuned on data d_t from time period t, progressively moving to the next time period $t+1$. We term this approach *Incremental Fine-tuning (**IFT**)* (Fig. 1d), where the model architecture and the loss function remain similar to traditional fine-tuning, but the model encounters chronological training data incrementally.

However, IFT may risk overfitting to recent data, potentially forgetting previously acquired knowledge which is crucial for extrapolating to the new distribution based on past data. Therefore, a balance between preserving knowledge from the past and accumulating information from recent data is essential to enhance temporal generalization. To explore this balance, we investigate continual learning methods and temporal invariant methods, to adapt to temporally emerging distribution shifts.

5.1 Continual Learning Methods

The aim of continual learning is to accumulate knowledge incrementally without forgetting information from previous steps referred to as catastrophic forgetting. In our specific context, we explore the efficacy of these methods in enabling models to extrapolate into the future based on past information, within the framework of a boundary-unaware, non-stationary temporal shift setting.

EWC (Kirkpatrick et al., 2017) Elastic Weight Consolidation is a regularization-based method which adds a temporal regularization term to the task-specific actual loss so that the parameter changes from $t-1$ to t are restricted to avoid over-fitting. It produces a weighted penalty such that the parameters that are more important to the previous time stamp will have larger penalty weights, to balance the trade-off between previous knowledge and new knowledge. It uses Fisher Information Matrices to estimate the importance of parameters to use them as the weighted penalty.

ER (Rolnick et al., 2019) Experience Replay falls into the category of rehearsal-based methods that stores samples from previous time stamps into a growing memory module. We use instances from memory as additional training examples along with current time stamp instances.

A-GEM (Lopez-Paz and Ranzato, 2017) Average-Gradient Episodic Memory, a rehearsal method, leverages an episodic memory to store a sample of examples from previous time stamp similar to ER, but additionally equip the loss on older samples as inequality constraints, avoiding their increase.

LoRA (Hu et al., 2021) falls into the category of parameter-expansion method which freezes the original parameters of the pre-trained model and introduces trainable low-rank matrices and combines them with the original matrices in the multi-head attention and are updated during fine-tuning.

	x		$x\|y$	
	Old	Rec.	Old	Rec.
UKLEX(S)	0.314	0.264	0.384	0.328
UKLEX(M)			0.412	0.366
EURLEX(S)	0.359	0.265	0.379	0.288
EURLEX(M)			0.409	0.324
ECHR(A)	0.236	0.155	0.306	0.247
ECHR(B)			0.318	0.266

Table 1: Jensen–Shannon divergence score between the split of training set (Old/Recent) and the test set over the vocabulary distribution (x) and vocabulary conditioned on the label ($x|y$). Higher Score indicates more divergence from the test set distribution.

Adapters (Houlsby et al., 2019) is another parameter expansion method that freezes the original parameters of the pre-trained model and injects two small modules between the self-attention sub-layer and the feed-forward sub-layer inside each layer of transformer sequentially. The adapter module consists of a down-projection, an up-projection, and a nonlinear function between them with a residual connection across each module.

5.2 Temporal Invariant Methods

Domain invariant representation learning boosts model transferability across domains by eliminating domain-specific information, preventing overfitting to specific domains, and improving generalization to unseen target domains. In our case, we aim to build a temporally invariant model by excluding features tied to specific time periods. The difficulty of their application in our context lies in the lack of well-defined boundaries for temporal distribution shifts, with no explicit signal for drift awareness. To tackle this, we treat every sliding window of length L timestamps as a domain.

DeepCORAL (Sun and Saenko, 2016) Correlation Alignment penalizes the differences in the mean and covariance of the feature distributions of each domain to obtain domain invariant representations.

IRM (Arjovsky et al., 2019) Invariant Risk Minimization aims to penalize variance across multiple training dummy estimators across domains, i.e., performance cannot vary across samples corresponding to the same domain.

GroupDRO (Sagawa et al., 2019) Group Distributionally Robust Optimization aims to optimize the worst-domain loss wherein the domain-wise losses are weighted inversely proportional to the performance of instances in that domain.

6 Experiments

6.1 Base Model & Metrics

For UKLEX and EURLEX, we use a state-of-the-art model, BERT-LWAN (Chalkidis et al., 2020a), which has a label-wise attention network on top of the pre-trained model. Specifically, we pass the text into LegalBERT (Chalkidis et al., 2020b) and employ one attention head per label to generate N document representations where N denotes the number of labels, which are finally passed through a linear layer to get the final predictions. For ECHR, we employ a state-of-the-art hierarchical version of BERT due to long input documents (Chalkidis et al., 2022). Specifically, we pass each sentence in the long document into LegalBERT to obtain [CLS] representations which are then passed through a two-layer transformer to obtain final representations. Implementation details of all the methods can be found in Appendix A. Following Chalkidis and Søgaard 2022, we report, micro-F1, macro-F1, and mean R-Precision (m-RP).

6.2 Results on Eval-Fix Setting

We present the results on eval-fix in Table 2.

Quantifying Temporal Shift: To understand the effect of temporal distance between the training and test data, we divide the training data into two versions: (i) the first half of chronologically sorted training data referred to as *Old*, and (ii) the latter half referred to as *Recent*. To quantify the temporal distribution shifts between these splits and the test set, we calculate the Jensen-Shannon divergence score between the distribution of the vocabulary (x). There is a possibility that the observed vocabulary distribution shift might be influenced by changes in label distribution over time. We further disentangle it by calculating conditional vocabulary distribution for each label ($x|y$) and report the average of divergence scores across all labels. This specifically assesses how the vocabulary associated with each label changes over time. From Table 1, we observe that the recent split, which is temporally closer to the test data, has a lower divergence score compared to older split, confirming our hypothesis of temporal distribution drift over time. It is noteworthy that this may underestimate the effect, as we are specifically calculating the drift at the lexical level without capturing semantic shifts over time (changes in associated meaning or contextual usage of specific words).

To further elucidate the impact of this tempo-

	ULKEX(S)			ULKEX(M)			EURLEX(S)		
Method	mac.-F1	mic.-F1	mRP	mac.-F1	mic.-F1	mRP	mac.-F1	mic.-F1	mRP
Baseline - Old	$72.92_{3.86}$	$78.26_{3.55}$	$76.02_{4.13}$	$50.54_{6.73}$	$67.34_{1.36}$	$62.72_{2.37}$	$59.16_{0.66}$	$72.19_{0.99}$	$66.53_{0.60}$
Baseline - Rec.	$73.26_{1.67}$	$79.34_{2.47}$	$77.20_{2.83}$	$52.38_{8.96}$	$69.84_{1.33}$	$66.01_{1.42}$	$64.08_{0.36}$	$76.20_{0.85}$	$71.62_{0.27}$
Baseline - Full	$74.96_{3.44}$	$80.61_{2.14}$	$78.66_{1.66}$	$54.89_{12.34}$	$70.85_{2.42}$	$67.25_{0.55}$	$67.70_{1.01}$	$78.57_{0.97}$	$73.46_{0.28}$
IFT	$78.65_{2.92}$	$82.29_{1.91}$	$80.20_{1.78}$	$55.18_{11.79}$	$72.57_{1.45}$	$69.34_{2.65}$	$66.28_{0.91}$	$77.71_{1.23}$	$72.48_{0.63}$
EWC	$80.15_{1.74}$	$83.80_{2.17}$	$82.41_{2.37}$	$55.66_{8.91}$	$74.79_{2.30}$	$\underline{72.22}_{3.81}$	$68.01_{0.66}$	$78.69_{0.70}$	$73.69_{0.20}$
ER	$\underline{80.26}_{1.77}$	$\underline{83.82}_{2.83}$	$\underline{83.08}_{4.20}$	$\underline{55.93}_{9.58}$	$\underline{75.03}_{3.22}$	$71.99_{3.40}$	$\mathbf{68.27}_{0.68}$	$\underline{78.84}_{0.83}$	$\underline{74.02}_{0.43}$
AGEM	$79.13_{1.69}$	$83.25_{2.17}$	$81.39_{2.44}$	$55.32_{9.44}$	$73.84_{2.84}$	$70.81_{4.03}$	$67.80_{0.86}$	$78.60_{0.82}$	$73.27_{0.34}$
LORA	$80.03_{1.95}$	$83.67_{2.78}$	$82.58_{2.79}$	$55.77_{9.95}$	$74.54_{3.89}$	$71.15_{4.45}$	$\underline{68.05}_{0.11}$	$\mathbf{79.02}_{0.87}$	$\mathbf{74.14}_{0.44}$
Adapter	$\mathbf{80.64}_{0.67}$	$\mathbf{84.15}_{3.99}$	$\mathbf{83.13}_{4.11}$	$\mathbf{56.27}_{10.84}$	$\mathbf{75.67}_{4.00}$	$\mathbf{72.65}_{4.29}$	$67.96_{0.68}$	$78.80_{1.17}$	$73.57_{0.65}$
DeepCORAL	$75.51_{3.66}$	$80.40_{2.54}$	$77.67_{1.52}$	$50.58_{8.26}$	$70.25_{2.46}$	$65.03_{2.39}$	$64.71_{1.93}$	$74.98_{2.21}$	$70.05_{1.81}$
IRM	$77.82_{4.73}$	$80.69_{0.88}$	$78.98_{0.65}$	$52.11_{10.08}$	$70.29_{0.40}$	$64.15_{0.04}$	$62.47_{2.42}$	$75.46_{2.77}$	$70.07_{2.99}$
GroupDRO	$77.11_{3.76}$	$80.73_{1.11}$	$78.76_{0.77}$	$51.45_{10.09}$	$70.06_{1.16}$	$64.98_{1.43}$	$63.57_{0.68}$	$76.91_{0.96}$	$71.40_{0.70}$
	EURLEX(M)			ECHR(A)			ECHR(B)		
Baseline - Old	$38.35_{0.81}$	$59.32_{1.90}$	$52.23_{1.51}$	$47.72_{3.05}$	$62.86_{2.41}$	$59.09_{3.06}$	$43.77_{2.48}$	$70.54_{1.50}$	$64.25_{0.68}$
Baseline - Rec.	$43.69_{1.26}$	$64.64_{1.80}$	$56.36_{1.59}$	$52.31_{5.76}$	$64.52_{0.20}$	$60.14_{0.87}$	$49.11_{3.99}$	$74.96_{1.32}$	$70.26_{2.05}$
Baseline - Full	$44.17_{1.25}$	$66.84_{2.02}$	$58.66_{1.13}$	$53.31_{8.36}$	$66.95_{0.61}$	$62.68_{1.03}$	$\mathbf{51.93}_{1.07}$	$\mathbf{75.27}_{0.18}$	$71.89_{1.39}$
IFT	$44.97_{2.23}$	$67.64_{1.88}$	$59.08_{2.16}$	$52.44_{7.11}$	$65.87_{0.84}$	$61.55_{0.97}$	$\mathbf{50.82}_{0.79}$	$\mathbf{75.13}_{2.43}$	$71.09_{1.49}$
EWC	$44.67_{2.40}$	$67.49_{2.28}$	$58.88_{1.55}$	$\mathbf{56.25}_{10.33}$	$67.58_{0.29}$	$62.43_{1.30}$	$50.01_{3.68}$	$74.86_{0.41}$	$70.21_{0.61}$
ER	$45.42_{1.58}$	$67.88_{1.99}$	$59.72_{0.98}$	$\underline{55.87}_{7.76}$	$\underline{68.55}_{1.75}$	$63.99_{0.15}$	$49.31_{2.51}$	$74.84_{0.04}$	$70.51_{0.29}$
AGEM	$44.48_{1.93}$	$66.77_{1.81}$	$58.42_{1.18}$	$54.93_{8.39}$	$67.26_{0.18}$	$63.66_{0.10}$	$49.87_{3.19}$	$\underline{75.11}_{0.61}$	$\underline{73.51}_{1.47}$
LORA	$\mathbf{47.43}_{1.76}$	$\mathbf{68.75}_{1.99}$	$\underline{60.18}_{1.21}$	$53.76_{5.89}$	$66.82_{0.38}$	$62.94_{1.01}$	$50.31_{2.31}$	$74.37_{0.72}$	$72.71_{2.05}$
Adapter	$\underline{46.15}_{1.34}$	$\underline{68.57}_{1.52}$	$\mathbf{60.83}_{1.24}$	$52.84_{3.76}$	$66.30_{1.24}$	$62.13_{0.69}$	$\underline{50.59}_{1.15}$	$74.79_{0.16}$	$\mathbf{73.57}_{1.97}$
DeepCORAL	$39.57_{1.73}$	$63.07_{1.69}$	$55.41_{1.73}$	$50.48_{5.98}$	$68.19_{1.76}$	$\underline{64.25}_{1.59}$	$48.57_{2.29}$	$70.77_{2.41}$	$68.94_{2.13}$
IRM	$40.66_{1.05}$	$65.12_{1.45}$	$57.55_{1.15}$	$51.62_{2.91}$	$\mathbf{68.57}_{0.77}$	$\mathbf{64.26}_{3.69}$	$49.96_{1.60}$	$73.87_{0.11}$	$70.19_{1.48}$
GroupDRO	$40.11_{2.33}$	$64.86_{2.02}$	$57.99_{1.84}$	$51.35_{0.13}$	$68.32_{3.62}$	$63.16_{2.68}$	$49.24_{2.64}$	$71.35_{1.53}$	$69.97_{1.55}$

Table 2: Performance of different categories of methods on the Eval-Fix setting over six datasets. Best and Second best values in each metric is bolded and underlined respectively. Subscript refers to standard deviations.

ral drift on model performance, we evaluate the model on the eval-fix test set by training the baseline model using *Old* and *Recent* splits of training set. To remove a possible confounding effect of dataset size, each of these two models has access to the same number of training instances. As shown in Table 2, *baseline-Recent* consistently outperforms *baseline-Old* across all metrics and datasets. This result validates our hypothesis that **models trained on temporally closer data to the test set tend to yield superior results, under the assumption of the same model and dataset size** as we observed that temporally closer data deviates less from the test set in vocabulary distribution. Finally, we use the whole training data to create *Baseline-Full* model and it exhibits enhanced performance across all metrics and datasets, underscoring the effectiveness of a larger dataset to develop a temporally robust generalizable classifier.

Baseline vs. IFT From Table 2, we observe *IFT* performs better than *Baseline-Full* in UKLEX(S,M) and EURLEX(M) but falls short in EURLEX(S) and ECHR(A,B). While *baseline* employs a traditional fine-tuning approach ignoring the temporal order of training dataset, IFT preserves the chronological order of the input dataset by training the model incrementally. This in turn assists the model in detecting drifts over time and adapting to them accordingly, simultaneously harnessing the whole dataset. However, this strategy may lead to overfitting to recent data, as evidenced by the lower performance in EURLEX(S) and ECHR(A,B). Mitigating this requires a mechanism to revisit older data to preserve their information while accumulating and adapting to newer trends/drifts.

Continual Learning Methods On UKLEX(S), all continual learning methods exhibit superior performance compared to both IFT and baseline approaches. Notably, Adapters and ER stand out, surpassing others, with EWC and LoRA following closely behind. AGEM shows comparatively lesser efficacy among them. This trend of continual learning methods outperforming IFT persists in the UKLEX(M) scenario, although the margin of improvement on macro-F1 scores is relatively narrow, given the larger number of labels in the medium (M) split coupled with high label imbalance.

Transitioning to EURLEX(S), continual learning methods again outshine IFT and the baseline,

	ULKEX(S)			ULKEX(M)			EURLEX(S)		
Method	macro-F1	micro-F1	m-RP	macro-F1	micro-F1	m-RP	macro-F1	micro-F1	m-RP
Baseline	$78.97_{4.33}$	$83.89_{3.08}$	$83.42_{3.60}$	$55.60_{6.18}$	$75.87_{3.43}$	$71.52_{4.23}$	$64.65_{6.35}$	$78.18_{3.96}$	$73.05_{5.40}$
IFT	$79.98_{4.55}$	$84.91_{3.12}$	$83.65_{3.61}$	$56.15_{6.26}$	$77.12_{3.48}$	$73.03_{4.33}$	$63.26_{4.78}$	$77.99_{3.25}$	$72.61_{4.60}$
EWC	$81.42_{5.21}$	$86.12_{3.49}$	$85.15_{4.26}$	$58.08_{6.66}$	$79.31_{5.85}$	$75.23_{6.83}$	$\underline{65.16}_{5.79}$	$78.64_{3.74}$	$73.38_{5.12}$
ER	$81.51_{4.76}$	$85.91_{3.20}$	$85.14_{3.87}$	$58.17_{6.51}$	$\underline{79.38}_{5.64}$	$\mathbf{75.55}_{6.69}$	$64.95_{6.03}$	$78.69_{3.64}$	$73.31_{5.08}$
AGEM	$\underline{81.78}_{5.41}$	$86.11_{3.64}$	$\underline{85.23}_{4.26}$	$\mathbf{58.45}_{6.94}$	$79.26_{5.64}$	$75.22_{6.84}$	$65.07_{6.02}$	$78.67_{3.76}$	$73.47_{5.34}$
LORA	$\mathbf{81.99}_{3.76}$	$\mathbf{86.19}_{3.68}$	$\mathbf{85.38}_{4.23}$	$\underline{58.31}_{7.15}$	$79.12_{5.77}$	$75.03_{6.81}$	$\mathbf{65.56}_{5.66}$	$\mathbf{78.86}_{3.53}$	$\mathbf{73.61}_{4.88}$
Adapter	$81.14_{5.46}$	$\underline{86.16}_{3.66}$	$85.17_{4.31}$	$58.22_{7.24}$	$\mathbf{79.72}_{5.45}$	$\underline{75.54}_{6.47}$	$64.82_{6.25}$	$\underline{78.77}_{3.78}$	$\underline{73.56}_{5.17}$
DeepCORAL	$77.51_{3.61}$	$83.14_{3.13}$	$81.60_{4.20}$	$51.56_{6.93}$	$74.96_{4.26}$	$69.85_{4.88}$	$57.31_{2.34}$	$75.70_{2.62}$	$69.43_{3.85}$
IRM	$77.83_{4.42}$	$83.48_{2.92}$	$82.02_{3.27}$	$52.94_{6.97}$	$74.79_{4.07}$	$70.45_{4.42}$	$61.21_{3.86}$	$76.68_{2.89}$	$71.17_{4.42}$
GroupDRO	$78.05_{4.71}$	$83.56_{4.33}$	$82.86_{4.53}$	$52.70_{8.17}$	$74.30_{5.58}$	$69.39_{5.66}$	$61.88_{3.13}$	$76.75_{2.38}$	$71.17_{3.49}$
	EURLEX(M)			ECHR(A)			ECHR(B)		
Baseline	$35.59_{9.45}$	$66.16_{6.16}$	$58.99_{7.61}$	$51.38_{6.78}$	$69.30_{3.66}$	$63.96_{3.33}$	$47.22_{4.17}$	$74.86_{2.08}$	$67.56_{3.10}$
IFT	$34.61_{9.27}$	$65.89_{6.11}$	$58.15_{7.45}$	$49.86_{6.52}$	$68.27_{3.56}$	$63.08_{3.34}$	$46.73_{4.10}$	$74.27_{2.07}$	$67.09_{3.17}$
EWC	$38.65_{7.15}$	$66.87_{6.33}$	$59.81_{7.85}$	$\mathbf{52.79}_{7.17}$	$70.23_{4.47}$	$65.05_{4.64}$	$46.60_{5.39}$	$75.01_{2.27}$	$68.15_{3.32}$
ER	$\underline{39.61}_{7.27}$	$66.88_{6.30}$	$59.92_{7.66}$	$51.82_{6.78}$	$70.18_{4.16}$	$64.98_{4.03}$	$\underline{47.46}_{5.33}$	$74.91_{2.42}$	$\mathbf{68.44}_{3.29}$
AGEM	$38.27_{7.64}$	$66.73_{6.33}$	$59.57_{7.95}$	$51.53_{6.74}$	$69.86_{4.22}$	$65.12_{3.94}$	$46.70_{4.70}$	$\underline{75.05}_{1.94}$	$68.12_{2.76}$
LORA	$\mathbf{40.39}_{7.70}$	$\underline{67.16}_{6.55}$	$\mathbf{60.16}_{7.92}$	$\underline{52.54}_{6.01}$	$70.43_{4.15}$	$65.74_{3.62}$	$47.09_{5.16}$	$\mathbf{75.13}_{2.65}$	$\underline{68.39}_{2.93}$
Adapter	$38.57_{7.78}$	$\mathbf{67.63}_{6.85}$	$\underline{60.14}_{8.26}$	$52.35_{6.73}$	$70.04_{4.36}$	$64.99_{4.20}$	$\mathbf{47.64}_{5.60}$	$74.78_{2.45}$	$67.83_{3.26}$
DeepCORAL	$30.50_{6.44}$	$60.62_{5.75}$	$53.14_{6.66}$	$45.48_{3.65}$	$72.19_{1.94}$	$66.95_{2.34}$	$40.81_{5.86}$	$70.27_{3.14}$	$65.37_{3.45}$
IRM	$34.82_{8.38}$	$62.76_{5.47}$	$55.36_{6.57}$	$48.74_{4.61}$	$\mathbf{76.32}_{2.71}$	$\mathbf{69.72}_{3.21}$	$45.82_{4.23}$	$73.42_{2.34}$	$68.33_{2.35}$
GroupDRO	$34.69_{7.00}$	$63.99_{5.37}$	$56.36_{6.38}$	$46.02_{5.50}$	$\underline{74.75}_{5.11}$	$\underline{68.75}_{4.90}$	$45.30_{2.58}$	$72.93_{3.92}$	$66.68_{5.96}$

Table 3: Aggregated performance of various method categories in the Eval-Stream setting across six datasets. Best and Second best values in each metric is bolded and underlined respectively. Subscript refers to standard deviations.

with LoRA and ER taking the lead. As observed in UKLEX(S,M), AGEM falls short again. Surprisingly, Adapters, which excels on UKLEX(S,M), does not exhibit the same level of performance on EURLEX(S). On EURLEX(M), Adapters, LoRA, and ER outperform both the baseline and IFT, while the other methods remain comparable. AGEM, once again, demonstrates lower performance.

Shifting focus to ECHR(A), EWC, ER, and AGEM outperform the baseline and IFT. Notably, Adapters exhibits the least favorable performance in this context. On ECHR(B), none of the continual learning methods succeed in surpassing IFT and baseline on micro- and macro-F1 scores. However, AGEM, LoRA, and Adapters outperform them on the m-RP metric. The underperformance on ECHR(B) can be attributed to the spurious correlations in the dataset, as highlighted in Santosh et al. 2022. These spurious attributes, inadvertently learned during training, persist over time, posing a challenge for the model to effectively unlearn them. Continual learning, while adept at adapting to evolving information, faces difficulties in fully addressing and mitigating these learned artifacts.

Among these methods, overall AGEM falls short in most of the datasets and we attribute it to its design of stricter inequality constraint giving the model less freedom to accumulate new knowledge.

Temporal Invariant Methods On UKLEX(S), temporal invariant methods outperform baseline but fall short compared to IFT. However, they struggle to cross the baseline in UKLEX(M), EURLEX(S,M), and ECHR(B) datasets. While on ECHR(A), they outperform IFT and even continual methods on micro-F1 and m-RP but fall short on macro-F1 metric. We speculate that the objective of temporal invariant methods aims to eliminate the distribution by learning generalizable feature representation, while the adaptive objective for the continual methods helps effectively to track and adapt to the shifts.

In summary, these findings collectively suggest that applying continual learning methods on top of incremental fine-tuning leads to improved performance in 5 out of 6 datasets. This underscores their capacity to adapt to natural shifts that occur over time, while also preserving past knowledge which assists them to extrapolate trends into the future from historical distribution.

6.3 Results on Eval-Stream Setting

Under the Eval-Stream setting, we assess performance through multiple splits, conducting evaluations on each subsequent timestamp, as detailed in Sec. 3. The performance visualization over the multiple splits for each method is presented in Appendix B. We report the averaged results across all the splits for each method in Table 3.

IFT vs. Baseline IFT performs better than baseline on UKLEX(S,M) only and underperforms on the other four datasets. This observed trend aligns with our eval-fix analysis, due to IFT's strong reliance on recent data, potentially leading to overfitting.

Continual Learning Methods All the continual Learning methods perform consistently better than IFT and baseline, over all the metrics across all the six datasets. This contrasts with our conclusions drawn in the eval-fix setting, where we did not observe such uniform performance improvement across all the methods. This discrepancy could be attributed to our reliance on conclusions drawn from a single test split in eval-fix, which may exhibit a distinct pattern of shift. These findings emphasize the importance of adopting a streaming evaluation protocol in datasets where distribution shifts occur naturally over time. Rather than relying solely on one split, which may lead to misleading conclusions about changes in performance and method efficacy, using streaming evaluation ensures more reliable and comprehensive insights.

Among the continual learning methods, LoRA demonstrates superior temporal robustness which can be attributed to its design of additional weight matrices. This design allows LoRA to effectively discern and assimilate additional drift compared to previous knowledge, accumulating it through additional parameters and thereby preserving older knowledge. Following closely, ER emerges as the next best performer, emphasizing the importance of revisiting past knowledge to guide the model away from heavy reliance on recent data. AGEM falls short and can be attributed to its stricter inequality constraint, as discussed before.

Temporal Invariant Methods All these approaches underperform compared to the baseline. Among them, IRM performs comparably to baseline, followed closely by GroupDRO. Notably, DeepCORAL (DC) consistently lags behind across all datasets. This can be attributed to the rationale behind these approaches. While IRM and GroupDRO employ penalties on loss functions so that performance cannot vary across samples of different time periods, DC directly minimizes the difference between feature representations across samples of different time periods. This design causes models trained with DC to suppress distribution shifts rather than actively learn and adapt to recent drifts.

7 Conclusion

In this study, we delve into temporal drift in legal multi-label text classification tasks, revealing a crucial oversight in the current model training process that treats the training data as a single, homogeneous block. This neglect results in the degradation of performance over time. To remedy this, we introduce ChronosLex, an incremental training paradigm that allows the model to interact with data chronologically, facilitating adaptation to temporal shifts. However, we observe a potential overfitting to recent data with this incremental approach. To address overfitting, we explore mitigation strategies using both continual learning and temporal invariant approaches and found that Continual algorithms exhibit promising results by leveraging historical distribution to extrapolate trends into the future. In contrast, temporal invariant methods prove less effective in tackling this. Finally, we advocate for streaming evaluation protocol with multiple time splits to draw reliable conclusions, especially when time is involved as a critical axis. In future, we plan to evaluate this incremental strategy across other legal tasks such as prior case retrieval, contract analysis, legal document summarization, regulatory compliance and legal argumentation, where the temporal aspect is crucial in modeling.

Limitations

While this study provides valuable insights for addressing temporal challenges in legal multi-label text classification tasks, it is important to acknowledge certain limitations.

Firstly, the experiments and findings are based on six multi-label legal classification datasets. The extent to which the proposed approaches generalize across different legal domains or other text classification tasks remains to be explored. Further, the paper assumes the homogeneity of data within each time split, and the effectiveness of the proposed incremental training paradigm may vary based on the degree of diversity and complexity within each temporal split. Furthermore, while the streaming evaluation protocol is advocated for its reliability, its application to a broader range of datasets and tasks might introduce additional computational overhead. Investigating the protocol's feasibility and performance in diverse settings is essential for establishing its practicality. Lastly, the paper addresses temporal drift in legal tasks, but a more detailed exploration of the specific causes

and characteristics of temporal drift within legal domains could enhance the understanding of the challenges involved.

While the paper recognizes the existence of temporal drift, delving deeper into the specific characteristics of temporal shifts in legal text data (e.g., gradual vs. abrupt shifts) and their impact on model performance could contribute to more nuanced insights into the temporal dynamics at play. Unless one is explicitly examining the effect of a particular watershed event on the legal system and is informed by substantial relevant legal expertise, attributing large legal datasets to underlying factors accounting to inherent concept drift may turn out to be infeasible and we reserve that as a potential avenue for future work. For instance, in ECtHR jurisprudence, one would need to find multiple near-identical cases with different outcomes. Finding those is hard and, if they were found, they would be of great interest to legal scholarship.

Additionally, this work assumes a pre-fixed static splits, following a standard deployment scenario, where the model is updated based on frequency. In prior work on the US Supreme Court (Katz et al., 2017), for example, a change in the court's presiding judge marks natural boundaries at the supra-year level. However, such segmentation requires in-depth knowledge of the domain and collection. The challenge of finding an appropriate time split and can we have it determined dynamically, is difficult and would possibly require a qualitative exploration of how identifiable shifts in the data coincide with an expert's intuition about the natural temporal segmentation of the collection.

Ethics Statement

The legal text classification datasets used in this study are sourced from publicly available repositories and produced by previous studies. Though the judgment corpus from ECHR is not anonymized and contains the real names of the involved parties, we do not foresee any harm incurred by our experiments. The intention behind this research is to provide a nuanced understanding of the temporal challenges inherent in legal text classification tasks. The findings are intended to benefit legal practitioners, researchers, and organizations involved in legal document management by enhancing the adaptability and performance of classification models in the dynamic legal landscape.

References

Oshin Agarwal and Ani Nenkova. 2022. Temporal effects on pre-trained models for language processing tasks. *Transactions of the Association for Computational Linguistics*, 10:904–921.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. 2019. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32.

Spurthi Amba Hombaiah, Tao Chen, Mingyang Zhang, Michael Bendersky, and Marc Najork. 2021. Dynamic language models for continuously evolving content. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2514–2524.

Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. 2019. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*.

Magdalena Biesialska, Katarzyna Biesialska, and Marta R Costa-jussà. 2020. Continual lifelong learning in natural language processing: A survey. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6523–6541.

Danushka Bollegala, Yutaka Matsuo, and Mitsuru Ishizuka. 2011. Relation adaptation: learning to extract novel relations with minimum supervision. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Ilias Chalkidis, Ion Androutsopoulos, and Nikolaos Aletras. 2019. Neural legal judgment prediction in english. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4317–4323.

Ilias Chalkidis, Manos Fergadiotis, and Ion Androutsopoulos. 2021a. Multieurlex-a multi-lingual and multi-label legal document classification dataset for zero-shot cross-lingual transfer. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6974–6996.

Ilias Chalkidis, Manos Fergadiotis, Sotiris Kotitsas, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. 2020a. An empirical study on large-scale multi-label text classification including few and zero-shot labels. *arXiv preprint arXiv:2010.01653*.

Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. 2020b. Legal-bert: The muppets straight out of law school. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2898–2904.

Ilias Chalkidis, Manos Fergadiotis, Dimitrios Tsarapatsanis, Nikolaos Aletras, Ion Androutsopoulos, and Prodromos Malakasiotis. 2021b. Paragraph-level rationale extraction through regularization: A case study on european court of human rights cases. In *Proceedings of the 2021 Conference of the North*

American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 226–241.

Ilias Chalkidis, Abhik Jana, Dirk Hartung, Michael Bommarito, Ion Androutsopoulos, Daniel Katz, and Nikolaos Aletras. 2022. Lexglue: A benchmark dataset for legal language understanding in english. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4310–4330.

Ilias Chalkidis and Anders Søgaard. 2022. Improved multi-label classification under temporal concept drift: Rethinking group-robust algorithms in a label-wise setting. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2441–2454.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and Marc'Aurelio Ranzato. 2019. On tiny episodic memories in continual learning. *arXiv preprint arXiv:1902.10486*.

Sanyuan Chen, Yutai Hou, Yiming Cui, Wanxiang Che, Ting Liu, and Xiangzhan Yu. 2020. Recall and learn: Fine-tuning deep pretrained language models with less forgetting. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7870–7881.

Cyprien de Masson D'Autume, Sebastian Ruder, Lingpeng Kong, and Dani Yogatama. 2019. Episodic memory in lifelong language learning. *Advances in Neural Information Processing Systems*, 32.

Bhuwan Dhingra, Jeremy R Cole, Julian Martin Eisenschlos, Daniel Gillick, Jacob Eisenstein, and William W Cohen. 2022. Time-aware language models as temporal knowledge bases. *Transactions of the Association for Computational Linguistics*, 10:257–273.

Komal Florio, Valerio Basile, Marco Polignano, Pierpaolo Basile, and Viviana Patti. 2020. Time of your hate: The challenge of time in hate speech detection on social media. *Applied Sciences*, 10(12):4180.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *The journal of machine learning research*, 17(1):2096–2030.

Judith Gaspers, Anoop Kumar, Greg Ver Steeg, and Aram Galstyan. 2022. Temporal generalization for spoken language understanding. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*, pages 37–44.

Kyle Gorman and Steven Bedrick. 2019. We need to talk about standard splits. In *Proceedings of the conference. Association for Computational Linguistics. Meeting*, volume 2019, page 2786. NIH Public Access.

Lin Lawrence Guo, Stephen R Pfohl, Jason Fries, Alistair EW Johnson, Jose Posada, Catherine Aftandilian, Nigam Shah, and Lillian Sung. 2022. Evaluation of domain generalization and adaptation on improving model robustness to temporal dataset shift in clinical medicine. *Scientific reports*, 12(1):2726.

Lin Lawrence Guo, Ethan Steinberg, Scott Lanyon Fleming, Jose Posada, Joshua Lemmon, Stephen R Pfohl, Nigam Shah, Jason Fries, and Lillian Sung. 2023a. Ehr foundation models improve robustness in the presence of temporal distribution shift. *Scientific Reports*, 13(1):3767.

Yue Guo, Chenxi Hu, and Yi Yang. 2023b. Predict the future from the past? on the temporal data distribution shift in financial sentiment classifications. *arXiv preprint arXiv:2310.12620*.

Suchin Gururangan, Mike Lewis, Ari Holtzman, Noah A Smith, and Luke Zettlemoyer. 2022. Demix layers: Disentangling domains for modular language modeling. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5557–5576.

Dan Hendrycks, Collin Burns, Anya Chen, and Spencer Ball. 2021. Cuad: An expert-annotated nlp dataset for legal contract review. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.

Beizhe Hu, Qiang Sheng, Juan Cao, Yongchun Zhu, Danding Wang, Zhengjia Wang, and Zhiwei Jin. 2023. Learn over past, evolve for future: Forecasting temporal trends for fake news detection. *arXiv preprint arXiv:2306.14728*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Xiaolei Huang and Michael Paul. 2018. Examining temporality in document classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 694–699.

Yufan Huang, Yanzhe Zhang, Jiaao Chen, Xuezhi Wang, and Diyi Yang. 2021. Continual learning for text classification with information disentanglement based

regularization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2736–2746.

Kokil Jaidka, Niyati Chhaya, and Lyle Ungar. 2018. Diachronic degradation of language models: Insights from social media. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 195–200.

Joel Jang, Seonghyeon Ye, Changho Lee, Sohee Yang, Joongbo Shin, Janghoon Han, Gyeonghun Kim, and Minjoon Seo. 2022. Temporalwiki: A lifelong benchmark for training and evaluating ever-evolving language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6237–6250.

Joel Jang, Seonghyeon Ye, Sohee Yang, Joongbo Shin, Janghoon Han, KIM Gyeonghun, Stanley Jungkyu Choi, and Minjoon Seo. 2021. Towards continual knowledge learning of language models. In *International Conference on Learning Representations*.

Mali Jin, Yida Mu, Diana Maynard, and Kalina Bontcheva. 2023. Examining temporal bias in abusive language detection. *arXiv preprint arXiv:2309.14146*.

Xisen Jin, Dejiao Zhang, Henghui Zhu, Wei Xiao, Shang-Wen Li, Xiaokai Wei, Andrew Arnold, and Xiang Ren. 2022. Lifelong pretraining: Continually adapting language models to emerging corpora. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4764–4780.

Daniel Martin Katz, Michael J Bommarito, and Josh Blackman. 2017. A general approach for predicting the behavior of the supreme court of the united states. *PloS one*, 12(4):e0174698.

Zixuan Ke and Bing Liu. 2022. Continual learning of natural language processing tasks: A survey. *arXiv preprint arXiv:2211.12701*.

Zixuan Ke, Bing Liu, Hao Wang, and Lei Shu. 2021a. Continual learning with knowledge transfer for sentiment classification. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2020, Ghent, Belgium, September 14–18, 2020, Proceedings, Part III*, pages 683–698. Springer.

Zixuan Ke, Hu Xu, and Bing Liu. 2021b. Adapting bert for continual learning of a sequence of aspect sentiment classification tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4746–4755.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Angeliki Lazaridou, Adhi Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Tomas Kocisky, Sebastian Ruder, et al. 2021. Mind the gap: Assessing temporal generalization in neural language models. *Advances in Neural Information Processing Systems*, 34:29348–29363.

Bill Yuchen Lin, Sida I Wang, Xi Lin, Robin Jia, Lin Xiao, Xiang Ren, and Scott Yih. 2022. On continual model refinement in out-of-distribution data streams. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3128–3139.

Marco Lippi, Przemysław Pałka, Giuseppe Contissa, Francesca Lagioia, Hans-Wolfgang Micklitz, Giovanni Sartor, and Paolo Torroni. 2019. Claudette: an automated detector of potentially unfair clauses in online terms of service. *Artificial Intelligence and Law*, 27:117–139.

Adam Liska, Tomas Kocisky, Elena Gribovskaya, Tayfun Terzi, Eren Sezener, Devang Agrawal, D'Autume Cyprien De Masson, Tim Scholtes, Manzil Zaheer, Susannah Young, et al. 2022. Streamingqa: A benchmark for adaptation to new knowledge over time in question answering models. In *International Conference on Machine Learning*, pages 13604–13622. PMLR.

David Lopez-Paz and Marc'Aurelio Ranzato. 2017. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Daniel Loureiro, Francesco Barbieri, Leonardo Neves, Luis Espinosa Anke, and Jose Camacho-Collados. 2022. Timelms: Diachronic language models from twitter. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 251–260.

Jan Lukes and Anders Søgaard. 2018. Sentiment analysis under temporal shift. In *Proceedings of the 9th workshop on computational approaches to subjectivity, sentiment and social media analysis*, pages 65–71.

Kelvin Luu, Daniel Khashabi, Suchin Gururangan, Karishma Mandyam, and Noah A Smith. 2022. Time waits for no one! analysis and challenges of temporal misalignment. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5944–5958.

Pedro Henrique Luz de Araujo, Teófilo E de Campos, Renato RR de Oliveira, Matheus Stauffer, Samuel Couto, and Paulo Bermejo. 2018. Lener-br: a dataset for named entity recognition in brazilian legal text. In *Computational Processing of the Portuguese Language: 13th International Conference, PROPOR 2018, Canela, Brazil, September 24–26, 2018, Proceedings 13*, pages 313–323. Springer.

Vijit Malik, Rishabh Sanjay, Shubham Kumar Nigam, Kripabandhu Ghosh, Shouvik Kumar Guha, Arnab Bhattacharya, and Ashutosh Modi. 2021. Ildc for cjpe: Indian legal documents corpus for court judgment prediction and explanation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4046–4062.

Katerina Margatina, Shuai Wang, Yogarshi Vyas, Neha Anna John, Yassine Benajiba, and Miguel Ballesteros. 2023. Dynamic benchmarking of masked language models on temporal concept drift with multiple views. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2873–2890.

Yida Mu, Kalina Bontcheva, and Nikolaos Aletras. 2023. It's about time: Rethinking evaluation on rumor detection benchmarks using chronological splits. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 724–731.

Joel Niklaus, Ilias Chalkidis, and Matthias Stürmer. 2021. Swiss-judgment-prediction: A multilingual legal judgment prediction benchmark. In *Proceedings of the Natural Legal Language Processing Workshop 2021*, pages 19–35.

Yasumasa Onoe, Michael Zhang, Eunsol Choi, and Greg Durrett. 2022. Entity cloze by date: What lms know about unseen entities. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 693–702.

Vasile Păiș, Maria Mitrofan, Carol Luca Gasan, Vlad Coneschi, and Alexandru Ianov. 2021. Named entity recognition in the romanian legal domain. In *Proceedings of the Natural Legal Language Processing Workshop 2021*, pages 9–18.

Christos Papaloukas, Ilias Chalkidis, Konstantinos Athinaios, Despina Pantazi, and Manolis Koubarakis. 2021. Multi-granular legal topic classification on greek legislation. In *Proceedings of the Natural Legal Language Processing Workshop 2021*, pages 63–75.

Shounak Paul, Pawan Goyal, and Saptarshi Ghosh. 2020. Automatic charge identification from facts: A few sentence-level charge annotations is all you need. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 1011–1022.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020. Adapterhub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54.

Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. 2020. Gdumb: A simple approach that questions our progress in continual learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pages 524–540. Springer.

Yujia Qin, Jiajie Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. 2022. Elle: Efficient lifelong pre-training for emerging data. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2789–2810.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. 2017. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010.

Shruti Rijhwani and Daniel Preoţiuc-Pietro. 2020. Temporally-informed analysis of named entity recognition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7605–7617.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. 2019. Experience replay for continual learning. *Advances in Neural Information Processing Systems*, 32.

Guy D Rosin, Ido Guy, and Kira Radinsky. 2022. Time masking for temporal language models. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pages 833–841.

Paul Röttger and Janet Pierrehumbert. 2021. Temporal adaptation of bert and performance on downstream document classification: Insights from social media. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2400–2412.

Sebastian Ruder. 2019. *Neural transfer learning for natural language processing*. Ph.D. thesis, NUI Galway.

Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. 2019. Distributionally robust neural networks. In *International Conference on Learning Representations*.

Tyss Santosh, Shanshan Xu, Oana Ichim, and Matthias Grabmair. 2022. Deconfounding legal judgment prediction for european court of human rights cases towards better alignment with experts. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1120–1138.

Jeffrey C Schlimmer and Richard H Granger. 1986. Incremental learning from noisy data. *Machine learning*, 1:317–354.

Chao Shang, Guangtao Wang, Peng Qi, and Jing Huang. 2022. Improving time sensitivity for question answering over temporal knowledge graphs. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8017–8026.

Anders Søgaard, Sebastian Ebert, Jasmijn Bastings, and Katja Filippova. 2021. We need to talk about random splits. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1823–1832.

Octavia-Maria Şulea, Marcos Zampieri, Mihaela Vela, and Josef van Genabith. 2017. Predicting the law area and decisions of french supreme court cases. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017*, pages 716–722.

Baochen Sun and Kate Saenko. 2016. Deep coral: Correlation alignment for deep domain adaptation. In *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14*, pages 443–450. Springer.

Fan-Keng Sun, Cheng-Hao Ho, and Hung-Yi Lee. 2019. Lamol: Language modeling for lifelong language learning. In *International Conference on Learning Representations*.

Don Tuggener, Pius Von Däniken, Thomas Peetz, and Mark Cieliebak. 2020. Ledgar: A large-scale multi-label corpus for text classification of legal provisions in contracts. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 1235–1241.

Santosh Tyss, Oana Ichim, and Matthias Grabmair. 2023. Zero-shot transfer of article-aware legal outcome classification for european court of human rights cases. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 593–605.

Josef Valvoda, Ryan Cotterell, and Simone Teufel. 2023. On the role of negative precedent in legal outcome prediction. *Transactions of the Association for Computational Linguistics*, 11:34–48.

Hong Wang, Wenhan Xiong, Mo Yu, Xiaoxiao Guo, Shiyu Chang, and William Yang Wang. 2019. Sentence embedding alignment for lifelong relation extraction. In *Proceedings of NAACL-HLT*, pages 796–806.

Gerhard Widmer and Miroslav Kubat. 1993. Effective learning in dynamic environments by explicit context tracking. In *Machine Learning: ECML-93: European Conference on Machine Learning Vienna, Austria, April 5–7, 1993 Proceedings 6*, pages 227–243. Springer.

Shanshan Xu, Leon Staufer, Oana Ichim, Corina Heri, Matthias Grabmair, et al. 2023. Vechr: A dataset for explainable and robust classification of vulnerability type in the european court of human rights. *arXiv preprint arXiv:2310.11368*.

Huaxiu Yao, Caroline Choi, Bochuan Cao, Yoonho Lee, Pang Wei W Koh, and Chelsea Finn. 2022. Wild-time: A benchmark of in-the-wild distribution shift over time. *Advances in Neural Information Processing Systems*, 35:10309–10324.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. 2018. Lifelong learning with dynamically expandable networks. In *International Conference on Learning Representations*.

Yuji Zhang, Jing Li, and Wenjie Li. 2023. Vibe: Topic-driven temporal adaptation for twitter classification. *arXiv preprint arXiv:2310.10191*.

Zhixue Zhao, George Chrysostomou, Kalina Bontcheva, and Nikolaos Aletras. 2022. On the impact of temporal concept drift on model explanations. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 4039–4054.

Lucia Zheng, Neel Guha, Brandon R Anderson, Peter Henderson, and Daniel E Ho. 2021. When does pretraining help? assessing self-supervised learning for law and the casehold dataset of 53,000+ legal holdings. In *Proceedings of the eighteenth international conference on artificial intelligence and law*, pages 159–168.

A Implementation Details

We will release our code upon acceptance.

Baseline Models We fine-tune the models on our six datasets using binary cross-entropy loss and they are trained end-to-end using AdamW optimizer (Loshchilov and Hutter, 2017), learning rate of 2e-5, momentum of 0.9, and weight decay of 0.01. For ECHR tasks, to account for longer input length, we employ hierarchical models which can process 64 paragraphs each with 128 tokens. We use a batch size of 60 for UKLEX and EURLEX tasks, and 6 for ECHR tasks due to memory constraints. The models are trained for up to 20 epochs using early stopping with a patience of 3 on macro-F1 score of the validation set to prevent the model. We use 16-bit automatic mixed precision and gradient accumulation to accelerate training and save memory. All the experiments were performed on a GPU cluster with NVIDIA A40 48GB PCIe 4.0.

IFT models At each time stamp t, we initialize the model with the best-performing model from the previous timestamp $(t-1)$. A warm-up phase of 3 epochs is employed before applying the patience

threshold. This initial warm-up phase allows the model to interact with the data in the current timestamp, providing ample time to discern drifts and adapt accordingly. The remaining hyperparameters are kept consistent with the baseline models, and the best model at each time stamp is selected based on the macro-F1 score on the validation set.

Evaluation Metrics: For UKLEX and EURLEX, we employ the standard approach to compute macro-F1, micro-F1, and m-RP scores on the list of labels. However, for ECHR, an additional label is introduced during inference to account for instances that do not violate (allege) any article. In Task A (B), where there are 14 (17) articles, this extra label is included in both targets and predictions and its value is set to 1 if none of the labels are 1 (indicating that all the labels are 0), following the methodology outlined by Chalkidis et al. (2022).

We use Wild-Time library (Yao et al., 2022) for implementing continual learning algorithms such as EWC, ER, AGEM, and temporal-invariant methods such as GroupDRO, DeepCORAL, and IRM. We use AdapterHub framework (Pfeiffer et al., 2020) to implement LoRA and Adapters.

A.1 Continual Learning methods specific Hyper-parameters

EWC (Kirkpatrick et al., 2017) We use the online version of EWC training to avoid memory overflow in the computation of Fischer information matrices, as detailed in (Kirkpatrick et al., 2017). We set λ of 0.5 which controls the strength of regularization-based EWC loss with a decay term for older data γ set to default of 1.0.

ER (Rolnick et al., 2019) We replay a mini-batch of data by random sampling from an evolving past data module after every 10 training steps for UKLEX and EURLEX tasks, and every 30 training steps for ECHR tasks.

AGEM (Lopez-Paz and Ranzato, 2017) stores parameter gradients of data from past timestamps and uses reservoir strategy to sample the past to add to the memory buffer. We use a maximum size of 1000 for the memory buffer for rehearsal.

LoRA (Hu et al., 2021) We set the low-dimensional rank r of the decomposition matrix to 8 and the α which accounts for scaling the reparameterization to 16.

Adapters (Houlsby et al., 2019) We used bottleneck adapters with a reduction factor of 16 which controls the ratio between the hidden dimension and the bottleneck dimension.

A.2 Temporal Invariant learning methods specific Hyper-parameters

For all the temporal invariant methods, we sample uniformly from each time period in the window, when treated as a single domain to apply domain invariant approaches to boundary-less temporal setting. We employ sliding windows of length 5. We set the number of domains to be considered to be 3.

DeepCORAL (Sun and Saenko, 2016) We found the model is highly sensitive to λ which controls the penalty on alignment loss due to the challenging nature of its direct effect on feature representations. We performed an exhaustive search over the range and found 0.001 to work well in our setting. We compute alignment penalties between features from all pairs of domains sampled.

IRM (Arjovsky et al., 2019) We set IRM penalty factor λ to be constant at 1.0.

GroupDRO (Sagawa et al., 2019) Each instance in the minibatch is sampled with uniform probabilities from the entire domain.

B Evaluation

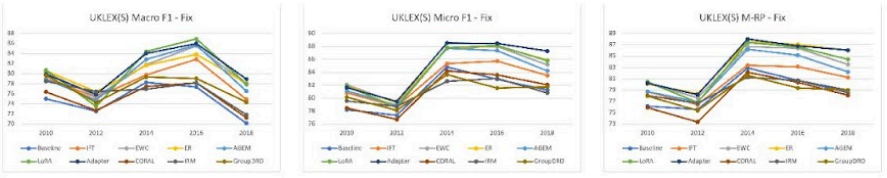
B.1 Eval-Fix

We present the performance of each method for every time period within the fixed test split of the Eval-Fix settings across all datasets in Figure 2. Notably, we observe a consistent improvement of methods across all timestamps in the test split, particularly evident in the EURLEX(S,M) and UKLEX(S,M) datasets where the line plots corresponding to each method appear parallel over the years. However, this trend is not uniformly observed in the ECHR(A,B) datasets. We posit that this deviation may be attributed to the extreme sparsity of the dataset, causing different methods to excel on different subsets, thereby influencing performance based on the distribution of labels in that time stamp. While a declining trend over the years has been observed in most datasets, it is expected due to the distribution shift between the training and testing sets as the temporal distance increases with the test year. However, some datasets, like UKLEX(S), exhibit a 'V'-shaped trend. This observation prompts future investigations to discern the reasons behind such trends and to study the underlying shift or drift phenomena contributing to this unique pattern. Unraveling these intricacies

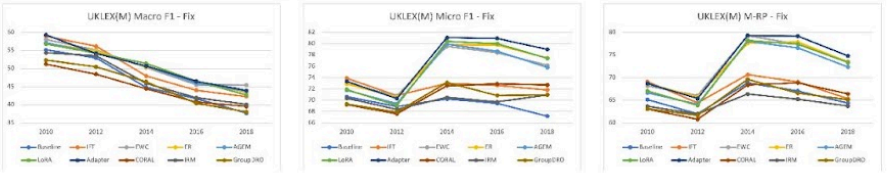
can serve as motivation for future research to devise shift-specific mitigation strategies tailored to address the nuanced temporal dynamics present in these datasets.

B.2 Eval-Stream

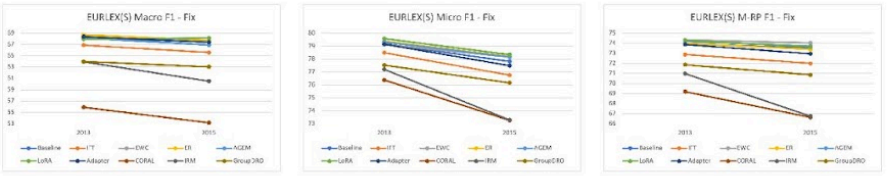
We evaluate the models on multiple splits and present the corresponding evaluation scores for subsequent time period at each split in Figure 3. The consistent trend of uniform improvements across methods on UKLEX(S,M) and EURLEX(S,M), as well as non-uniform improvements on ECHR(A,B), is mirrored in the streaming setting as well. This non-uniform nature can prompt interesting case studies to decipher what models might be learning and leverage this information to design strategies for improving legal text classification models.



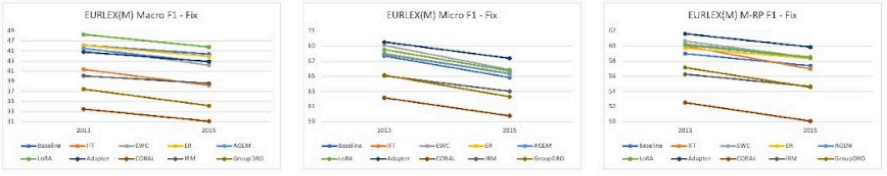
(a) UKLEX(S)



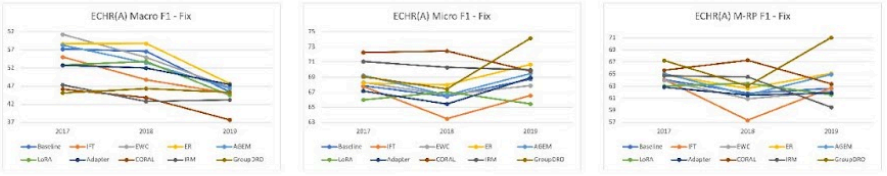
(b) UKLEX(M)



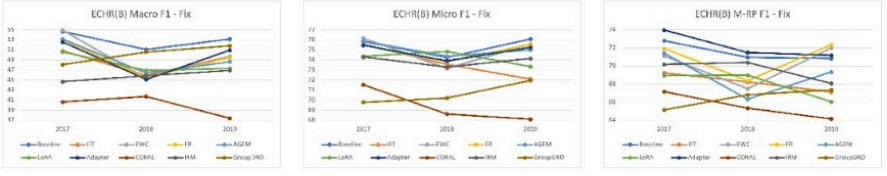
(c) EURLEX(S)



(d) EURLEX(M)



(e) ECHR(A)



(f) ECHR(B)

Figure 2: Results on Eval-Fix Setting over six datasets.

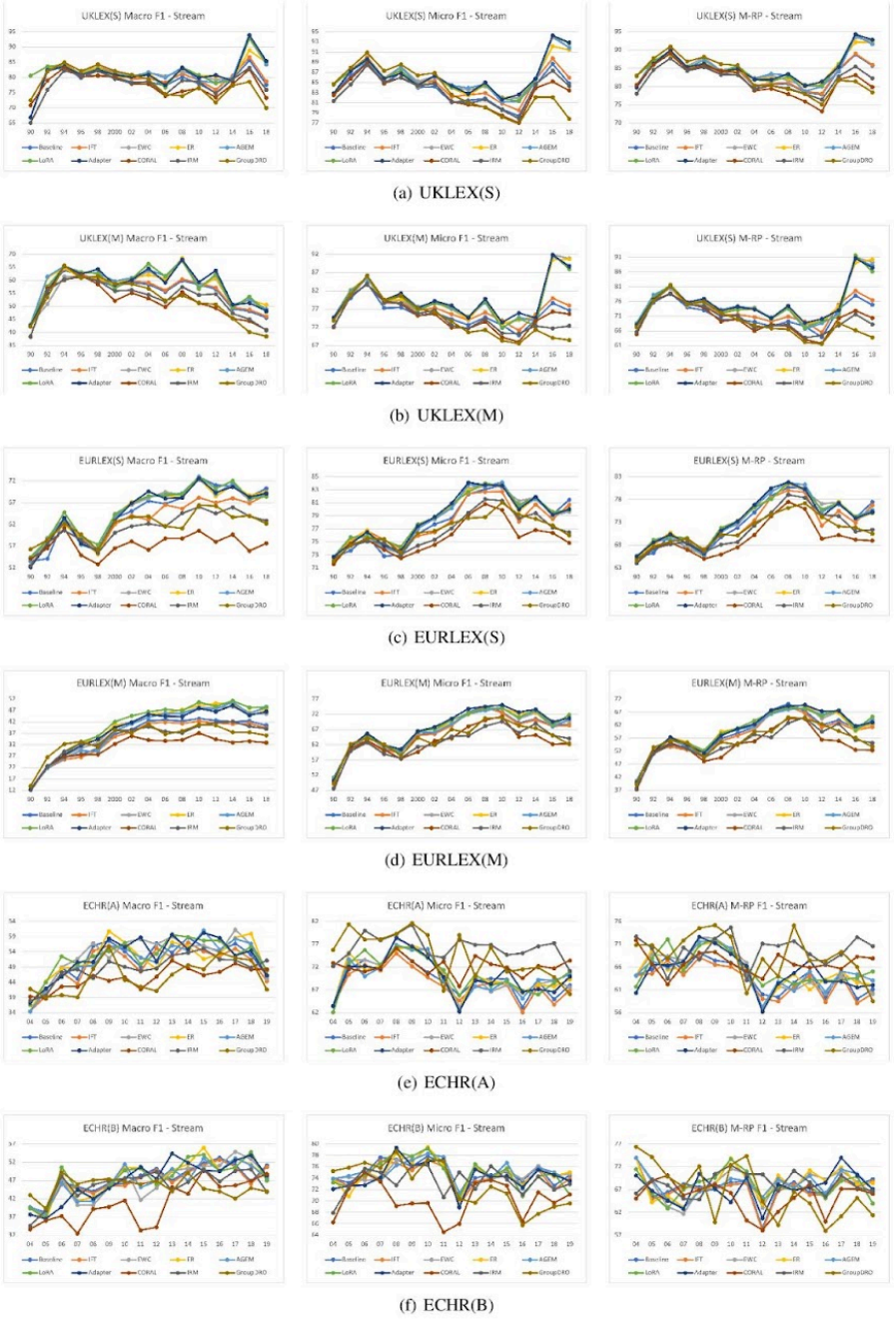


Figure 3: Results on Eval-Stream Setting over six datasets.

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 28 de mai. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

MATHEUS FARES COSTA BRAKES

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Desenvolvi um código para replicar os experimentos descritos no artigo ChronosLex (arXiv:2405.14211). Consegui reproduzir com sucesso o modelo incremental, porém ainda não implementei todos os demais modelos de comparação necessários para avaliar completamente a validade das propostas originais. Link do código : https://drive.google.com/file/d/1XFnbR-ymHMN8JdUZA9ZVRhMnMMdvKGqA/view?usp=drive_link

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Finalizar a implementação dos modelos de referência que faltam e estruturar as propostas de treinamento incremental adaptativo (quando disparado por detecção de concept drift em vez de em intervalos fixos). Em seguida, avaliar o ganho de desempenho ao usar ferramentas de detecção de drift para determinar dinamicamente quando atualizar o modelo.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

APÊNDICE 6

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 4 de jun. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

MATHEUS FARES COSTA BRAKES

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Eu realizei uma limpeza e reorganização do código anterior, corrigindo parâmetros e ajustes de lógica que estavam incorretos. Desenvolvi o módulo para rodar os baselines do ChronosLex em dois datasets, treinando dois modelos em três partições ("Old", "Recent" e "Full") e implementei o ITF (treinamento incremental temporal). O código está funcional e segue fielmente o artigo, com pequenas inferências em pontos não detalhados no paper. O treinamento ainda não foi finalizado mas já está rodando. Também foi implementado a parte de detecção do drift. para visualização antes da implementação no treinamento do modelo. Link do código
https://drive.google.com/file/d/131OciNDeNQI1K7iHVxd687zb3aUNDBht/view?usp=drive_link

Também foram realizados testes para detecção de concept drift analisando exclusivamente a variação nos dados de entrada, sem um ground truth estabelecido. Os métodos explorados, baseados na média de embeddings por janela temporal, mostraram-se insuficientes para identificar drifts de forma consistente com as configurações e métricas testadas.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

No próximo passo, vou concluir a coleta e o cálculo das métricas para cada baseline (incluindo o ITF), integrar métodos de Continual Learning e finalizar a implementação de detecção de concept drift para usar janelas de tempo dinâmicas em vez de intervalos fixos.

Também irei pesquisar e experimentar mais métodos mais robustos e sensíveis para a detecção de concept drift considerando o domínio, investigando métodos mais complexos e uso de diferentes parâmetros

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("gate") de aprovação: 11 de jun. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

MATHEUS FARES COSTA BRAKES

Entrega: [descrever a ENTREGA: requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Nesta fase, o código foi organizado para estruturar o projeto em um repositório (ainda não subi dentro do github), e o treinamento foi realizado em apenas dois dos conjuntos de dados do artigo o EURLEX small e UKLEX small para simplificar a implementação, já que ambos compartilham a mesma arquitetura de treinamento, e diminuir a quantidade de treinamentos dentro do experimento.

Foi realizado uma análise do drift temporal nesses datasets, buscando identificar janelas de tempo em que um novo treinamento seria mais adequado do que um ponto fixo t. Para isso, além da métrica JSD utilizada no artigo, aplicamos o MMD para comparar e detectar o drift em cada conjunto com o passar do tempo.

Em seguida, implementamos os treinos dos baselines "full", "old" e "recent", do IFT incremental e de outros três métodos que tiveram resultados melhores para os datasets, utilizando a biblioteca Avalanche para Experience Replay (ER) e Elastic Weight Consolidation (EWC), e empregando PEFT para replicar o LoRA. Foi necessário estudar a documentação do Avalanche em razão da ausência de suporte nativo ao PyTorch/Hugging Face, mas os exemplos fornecidos pela documentação possibilitaram a adaptação e implementação das técnicas desejadas.

Link com os códigos e experimentos gerados : Gate 11

Dentro do treino usando o que foi detectado com as métricas de concept drift, foi percebido diversas modificações ao longo da data, como o impacto do Brexit em como se dá os dados.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

[Notebook Jupyter desenvolvido para replicar a versão final do experimento *ChronosLex*, conforme descrito no Termo de Aceite de Entrega de 11 de junho]

Introdução

Este notebook replica parcialmente o experimento descrito em *ChronosLex* (2024). Implementamos:

- **Baseline** (fine-tuning tradicional)
- **Incremental Fine-Tuning (IFT)**
- Métodos de **Continual Learning**: *Elastic Weight Consolidation (EWC)*, *Experience Replay (ER)*, *LoRA*.

Usamos as variantes **Small (S)** dos conjuntos **EURLEX** e **UKLEX** (Seção 4 do artigo).

Em cada célula de código, procure pelas marcações:

- `Replicado do artigo:` parâmetro idêntico ao texto.
- `Alteração em relação ao artigo:` ajustes motivados por limitações práticas (GPU, tempo, etc.).

Import libs

```
%autosave 30

import os
from datetime import date
import pandas as pd
import numpy as np
from tqdm.auto import tqdm

from datasets import load_dataset, Dataset
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import f1_score

from transformers import (
    AutoConfig,
    AutoTokenizer,
    AutoModel,
    Trainer,
    TrainingArguments,
```

```
    DataCollatorWithPadding,
    EvalPrediction,
    EarlyStoppingCallback,
    set_seed,
)

from typing import List, Tuple
import torch

import torch.nn as nn
import nltk

from scipy.spatial.distance import jensenshannon
```

Pré-processamento e detecção de concept drift

Contexto e objetivo do pré-processamento

Antes de alimentar o modelo, precisamos **normalizar e tokenizar** os documentos jurídicos.
Replicado do artigo (Seção 6.1): uso do *LegalBERT* com `max_length = 512`, truncamento *head-tail* e **sem lematização** para não perder informação semântica temporal.
Alteração em relação ao artigo: mantive `padding="max_length"` e `return_tensors="pt"` para facilitar *batching* na GPU A40 48 GB; também removi caracteres de controle que causavam `UnicodeDecodeError` durante execução local.

```
def to_dataframe_eur(ds, split: str) -> pd.DataFrame:
    """
    Converte um split do MultiEURLEX em DataFrame com colunas:
    text, labels, split, year, date.
    """
    df = pd.DataFrame(ds[split])
    df["split"] = split
    df["text"]  = df["text"].apply(lambda x: " ".join(x) if isinstance(x,
list) else x)
    # extrai ano de celex_id (chars 1-4)
    df["year"]  = df["celex_id"].str[1:5].astype(int)
    df["date"]  = df["year"].apply(lambda y: date(y, 1, 1))
    return df

def process_uklex_df(df_slice: pd.DataFrame, split_name: str) ->
pd.DataFrame:
    """
```

```
    Processa uma fatia do DataFrame UKLEX, adicionando colunas essenciais.

    Args:
        df_slice (pd.DataFrame): Uma fatia do DataFrame original (ex:
treino, validação ou teste).
        split_name (str): O nome do split a ser atribuído (ex: 'train').

    Returns:
        pd.DataFrame: O DataFrame processado com as novas colunas.
    """
    return df_slice.assign(
        # Cria a coluna 'text' concatenando 'title' e 'body'.
        text=lambda x: x['title'] + "\n" + x['body'],

        # Atribui o nome do split.
        split=split_name,

        # Cria a coluna 'date' a partir do 'year'.
        date=lambda x: x['year'].apply(lambda y: date(y, 1, 1))
    )
```

Sobre a disponibilidade dos dados

Os arquivos brutos do **EURLEX-S** e **UKLEX-S** não podem ser distribuídos aqui por questões de licenciamento.
A célula de código a seguir presspõe que você já exportou os *splits* cronológicos sugeridos no artigo (train/valid/test) para os diretórios:

```
./data/eurlex_small/
./data/uklex_small/
```

Replicado do artigo:

- EURLEX-S — train (1958-2010), valid (2010-2012), test (2012-2015). - Tem como carregar o EURLEX a partir do Hugging Face.
- UKLEX-S — train (2000-2014), valid (2014-2016), test (2016-2019). - Link para carregar o documento : https://zenodo.org/records/6355465

```
# Replicado do artigo: pré-processamento semelhante ao descrito na Seção
6.1.
# Alteração em relação ao artigo: ajustado max_seq_length=512 devido à GPU
disponível.
def compute_js_drift(df: pd.DataFrame, split: str) -> float:
    """
    JS divergence (not distance) entre vocabulários old vs. recent
    no split de treino. Retorna o valor de JS-divergence (JSD),
```

```
    elevando ao quadrado o resultado de
scipy.spatial.distance.jensenshannon.
    """
    block = df[df["split"] == split].sort_values("date")
    mid   = len(block) // 2

    old_texts = block.iloc[:mid]["text"].tolist()
    rec_texts = block.iloc[mid:]["text"].tolist()

    vec = CountVectorizer(max_features=50_000)  # opcional: limite de
vocabulário
    X   = vec.fit_transform(old_texts + rec_texts)

    f_old = np.array(X[:mid, :].sum(axis=0)).ravel().astype(float)
    f_rec = np.array(X[mid:, :].sum(axis=0)).ravel().astype(float)

    p = f_old / f_old.sum()
    q = f_rec / f_rec.sum()
    return float(jensenshannon(p, q) ** 2)

# Replicado do artigo: pré-processamento semelhante ao descrito na Seção
6.1.
# Alteração em relação ao artigo: ajustado max_seq_length=512 devido à GPU
disponível.
def compute_js_to_test(
    df: pd.DataFrame,
    split: str,
    label_list: List[int]
) -> Tuple[float, float]:
    """
    Retorna (JSD_old_vs_test, JSD_recent_vs_test), médio sobre cada label
em label_list.
    Usa a divergência JS (não a distância).
    """
    block   = df[df["split"] == split].sort_values("date")
    mid     = len(block) // 2
    old_df  = block.iloc[:mid]
    rec_df  = block.iloc[mid:]
    test_df = df[df["split"] == "test"]

    js_old, js_rec = [], []

    for y in label_list:
        old_y  = old_df[old_df["labels"].apply(lambda labs: y in
```

```
labs)]["text"].tolist()
        rec_y  = rec_df[rec_df["labels"].apply(lambda labs: y in
labs)]["text"].tolist()
        test_y = test_df[test_df["labels"].apply(lambda labs: y in
labs)]["text"].tolist()

        if not (old_y and rec_y and test_y):
            continue

        vec = CountVectorizer(max_features=50_000)  # opcional: limita
vocabulário
        X   = vec.fit_transform(old_y + rec_y + test_y)

        n_old = len(old_y)
        n_rec = len(rec_y)

        f_old  = np.array(X[:n_old, :].sum(axis=0)).ravel().astype(float)
        f_rec  = np.array(X[n_old:n_old+n_rec,
:].sum(axis=0)).ravel().astype(float)
        f_test = np.array(X[n_old+n_rec:,
:].sum(axis=0)).ravel().astype(float)

        if f_old.sum() > 0 and f_test.sum() > 0:
            js_old.append(jensenshannon(f_old / f_old.sum(),
                                        f_test / f_test.sum()) ** 2)
        if f_rec.sum() > 0 and f_test.sum() > 0:
            js_rec.append(jensenshannon(f_rec / f_rec.sum(),
                                        f_test / f_test.sum()) ** 2)

    return (
        float(np.mean(js_old)) if js_old else 0.0,
        float(np.mean(js_rec)) if js_rec else 0.0
    )
```

Carregando os dados

```
multi = load_dataset("coastalcph/multi_eurlex", "en")
df_eur = pd.concat(
    [to_dataframe_eur(multi, s) for s in ["train","validation","test"]],
    ignore_index=True
)

multi
```

```
print("Primeiro exemplo de 'train':")
print(multi['train'][1])

num_labels_s = len(multi['train'].features['labels'].feature.names)

print(f"Dataset EURLEX Small (S) carregado.")
print(f"Número de labels: {num_labels_s}")

file_path = "data/uk-lex18.jsonl"
df_full_sorted = pd.read_json(file_path,
lines=True).sort_values(by="year").reset_index(drop=True)

# Definir os tamanhos para a divisão
n_train = 20000
n_val = 8500

# Dividir e processar cada split usando a função auxiliar
# O uso de .copy() na fatia previne o SettingWithCopyWarning do pandas.
train_df = process_uklex_df(df_full_sorted.iloc[:n_train].copy(), 'train')
validation_df = process_uklex_df(df_full_sorted.iloc[n_train : n_train +
n_val].copy(), 'validation')
test_df = process_uklex_df(df_full_sorted.iloc[n_train + n_val :].copy(),
'test')

# Concatenar os DataFrames processados em um único DataFrame final
df_uk = pd.concat([train_df, validation_df, test_df], ignore_index=True)

label_list_eur = sorted({ lbl for labs in df_eur["labels"] for lbl in labs
})
label_list_uk  = sorted({ lbl for labs in df_uk["labels"]  for lbl in labs
})

# 1. Criar um dicionário para mapear cada nome de label (string) para um ID
(inteiro)
label_to_id_uk = {label: i for i, label in enumerate(label_list_uk)}

# 2. Aplicar este mapeamento à coluna 'labels' do DataFrame df_uk
# A função lambda abaixo transforma cada lista de strings em uma lista de
inteiros
df_uk['labels'] = df_uk['labels'].apply(
    lambda label_names: [label_to_id_uk[name] for name in label_names]
)

print("Coluna 'labels' do df_uk foi convertida de strings para inteiros com
sucesso!")
```

```
# Verificando o resultado da conversão na primeira linha
print("Primeira linha de labels após a conversão:",
df_uk['labels'].iloc[0])

df_uk.info()
```

Cálculo da Divergência de Jensen-Shannon (JS)

Nesta seção reproduzimos a métrica usada na Tabela 1 do artigo para medir **drift lexical** entre conjuntos *Old*, *Recent* e Teste.

Replicado do artigo: usamos contagem de *unigramas* em minúsculas, normalizadas por frequência, sem ponderação TF-IDF.
Alteração: limitamos o vocabulário aos **50 k tokens** mais frequentes para evitar estouro de memória — este corte não altera a tendência observada no artigo, apenas reduz a dimensionalidade.

Referência: Seção 6.2 e Tabela 1 filecite turn2file9

```
js_eur_drift = compute_js_drift(df_eur, "train")
js_uk_drift  = compute_js_drift(df_uk,  "train")

print(f"EURLEX train Old⇄Recent JS: {js_eur_drift:.3f}")
print(f"UKLEX  train Old⇄Recent JS: {js_uk_drift:.3f}")

js_eur_old_test, js_eur_rec_test = compute_js_to_test(df_eur, "train",
label_list_eur)
js_uk_old_test,  js_uk_rec_test  = compute_js_to_test(df_uk,  "train",
label_list_uk)

print(f"\nEURLEX – Old vs Test:   {js_eur_old_test:.3f}")
print(f"EURLEX – Recent vs Test:{js_eur_rec_test:.3f}")
print(f"\nUKLEX  – Old vs Test:   {js_uk_old_test:.3f}")
print(f"UKLEX  – Recent vs Test:{js_uk_rec_test:.3f}")
```

Tokenizer dataset

```
df_eur.head()

df_uk.head()

base_model = "nlpaueb/legal-bert-base-uncased"
```

```
tok_eur = AutoTokenizer.from_pretrained(base_model)
tok_uk  = AutoTokenizer.from_pretrained(base_model)

from datasets import load_dataset, Dataset


def df_to_hf(
    df: pd.DataFrame,
    tokenizer: AutoTokenizer,
    label_list: List[int],
    *,  # Força args nomeados
    add_none_class: bool = False
) -> Dataset:
    """
    Converte pandas.DataFrame → 🤗Dataset, tokeniza e binariza labels.
    add_none_class fica sempre False (não há “no-violation” aqui).
    """
    from datasets import load_dataset, Dataset

    ds = Dataset.from_pandas(df)

    def tokenize_and_binarize(examples):
        tok = tokenizer(
            examples["text"],
            truncation=True,
            padding=False,
            max_length=512,
        )
        mh = []
        for labs in examples["labels"]:
            v = [0.0] * len(label_list)
            # nenhum exemplo sem label (já removidos em UKLEX),
            # e em EURLEX todo tem pelo menos 1 label
            for i in labs:
                if i < len(v):
                    v[i] = 1.0
            mh.append(v)
        tok["labels"] = mh
        return tok

    ds = ds.map(
        tokenize_and_binarize,
        batched=True,
```

```
        remove_columns=[c for c in ds.column_names
                        if c not in tokenizer.model_input_names +
["labels"]],
    )
    ds.set_format(type="torch", columns=tokenizer.model_input_names +
["labels"])
    return ds

hf_eur = df_to_hf(df_eur, tok_eur, label_list_eur, add_none_class=False)
hf_uk  = df_to_hf(df_uk,  tok_uk,  label_list_uk,  add_none_class=False)
```

Preparar lista de rótulos & métricasDefinir lista de rótulos e modelo base

```
from scipy.special import expit as sigmoid
from sklearn.metrics import f1_score

def mean_r_precision(y_true, y_scores):
    rps = []
    for yt, ys in zip(y_true, y_scores):
        r = int(yt.sum())
        if r == 0:
            continue
        top_r = np.argsort(ys)[-r:]
        # soma quantos dos r labels verdadeiros aparecem no top_r
        rps.append(yt[top_r].sum() / r)
    return np.mean(rps)

def compute_metrics(eval_pred: EvalPrediction):
    logits, labels = eval_pred.predictions, eval_pred.label_ids
    # 1) converte logits → probabilidades [0,1]
    probs = sigmoid(logits)
    # 2) aplica threshold 0.5 para obter previsões binárias
    bin_preds = (probs >= 0.5).astype(int)

    # 3) calcula F1 macro/micro
    macro = f1_score(labels, bin_preds, average="macro")
    micro = f1_score(labels, bin_preds, average="micro")
    # 4) calcula m-RPrecision
    mrp = mean_r_precision(labels, probs)

    return {
        "macro_f1":    macro,
        "micro_f1":    micro,
        "m_rprecision": mrp,
    }
```

Rodando o Baseline

Gerando o split para o baseline

```
def get_train_split(df: pd.DataFrame, which: str) -> pd.DataFrame:
    """
    Recebe um DataFrame com colunas ["split", "date", ...] e retorna:
      - which="full"   → todo df[df["split"] == "train"]
      - which="old"    → metade antiga (ordenada por date) de
df[df["split"] == "train"]
      - which="recent" → metade recente (ordenada por date) de
df[df["split"] == "train"]
    """
    df_train = df[df["split"] ==
"train"].sort_values("date").reset_index(drop=True)
    if df_train.empty:
        return pd.DataFrame(columns=df.columns)

    if which == "full":
        return df_train
    elif which == "old":
        mid = len(df_train) // 2
        return df_train.iloc[:mid].reset_index(drop=True)
    elif which == "recent":
        mid = len(df_train) // 2
        return df_train.iloc[mid:].reset_index(drop=True)
    else:
        raise ValueError("`which` deve ser um dos: 'full', 'old' ou
'recent'.")
```

Processamento do Dataset do UKLEX

```
from torch.utils.data import Dataset as TorchDataset, DataLoader

import nltk

# Baixa os dados necessários para o sent_tokenize
nltk.download('punkt', quiet=True)
nltk.download('punkt_tab', quiet=True)
```

Geração do modelo do Flat

Replicado do artigo: O baseline *flat* consiste em treinar o modelo **apenas uma vez** com todos os exemplos disponíveis até o ano de corte, ignorando o eixo temporal. Isto

corresponde ao experimento “Flat” descrito na Seção 5.2 e na Figura 3 do artigo *ChronosLex*.

Propósito: servir como linha de base estática para comparar com abordagens que levam em conta a evolução temporal (IFT e métodos de Continual Learning).

```
from typing import List, Tuple, Dict, Any, Optional

import torch
import torch.nn as nn
from transformers import PreTrainedModel, AutoModel, BertConfig
from typing import Optional, Tuple
from transformers import BertModel

from transformers import BertModel, PreTrainedModel
from transformers.modeling_outputs import SequenceClassifierOutput

from transformers import BertPreTrainedModel, BertModel
from transformers.modeling_outputs import SequenceClassifierOutput
from typing import Optional

# 1. Herde de BertPreTrainedModel em vez de nn.Module
class BertLWANForMultiLabel(BertPreTrainedModel):
    def __init__(self, config): # 2. O __init__ recebe um objeto 'config'
        super().__init__(config)
        self.num_labels = config.num_labels
        self.hidden_size = config.hidden_size

        # 3. O BERT base é herdado e gerenciado pela classe pai
        self.bert = BertModel(config)

        self.lwan_heads = nn.ModuleList([
            nn.Sequential(
                nn.Linear(self.hidden_size, self.hidden_size),
                nn.Tanh(),
                nn.Linear(self.hidden_size, 1)
            ) for _ in range(self.num_labels)
        ])
        self.classifiers = nn.ModuleList([
            nn.Linear(self.hidden_size, 1) for _ in range(self.num_labels)
        ])

        # 4. Inicializa os pesos das novas camadas e garante que o modelo
esteja pronto
        self.post_init()
```

```
    # 5. A assinatura do forward deve ser compatível com o que o Trainer
espera
    def forward(
        self,
        input_ids: torch.Tensor,
        attention_mask: torch.Tensor,
        token_type_ids: Optional[torch.Tensor] = None,
        labels: Optional[torch.Tensor] = None,
        **kwargs
    ):
        outputs = self.bert(
            input_ids=input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids
        )

        last_hidden = outputs.last_hidden_state

        logits_list = []
        expanded_attention_mask = attention_mask.unsqueeze(-1)

        for i in range(self.num_labels):
            head = self.lwan_heads[i]
            classifier = self.classifiers[i]

            scores = head(last_hidden)
            mask_value = torch.finfo(scores.dtype).min
            scores_masked = scores.masked_fill(expanded_attention_mask ==
0, mask_value)

            attn_weights = torch.softmax(scores_masked, dim=1)
            label_specific_rep = (attn_weights * last_hidden).sum(dim=1)
            logit = classifier(label_specific_rep)
            logits_list.append(logit)

        logits = torch.cat(logits_list, dim=1)

        loss = None
        if labels is not None:
            loss_fct = nn.BCEWithLogitsLoss()
            loss = loss_fct(logits, labels.float())

        return SequenceClassifierOutput(
```

```
            loss=loss,
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )
```

Função única de treino para os três splits

```
import os

os.environ.setdefault("WORLD_SIZE", "1")
os.environ.setdefault("RANK", "0")
os.environ.setdefault("LOCAL_RANK", "0")
os.environ.setdefault("MASTER_ADDR", "127.0.0.1")
os.environ.setdefault("MASTER_PORT", "29500")

# Alteração em relação ao artigo: ajustado max_seq_length=512 devido à GPU disponível.

import os
import numpy as np
import torch
from typing import Any, Dict, List

from transformers import (
    AutoTokenizer,
    TrainingArguments,
    Trainer,
    EarlyStoppingCallback,
    DataCollatorWithPadding,
    set_seed
)
from torch.cuda.amp import autocast, GradScaler
from sklearn.metrics import f1_score

from transformers import TrainerCallback, TrainingArguments, TrainerState,
TrainerControl, EarlyStoppingCallback
# Alteração em relação ao artigo: removido cálculo de m-RP para acelerar exemplo.

class EarlyStoppingWarmupCallback(EarlyStoppingCallback):
    """
    Uma versão do EarlyStoppingCallback que ignora as avaliações
    durante um número especificado de épocas de "warmup".

    Args:
```

```
        early_stopping_patience (int):
            Número de avaliações sem melhora a aguardar antes de parar.
        early_stopping_threshold (float):
            A melhora mínima no score para resetar a paciência.
        warmup_epochs (int):
            O número de épocas iniciais durante as quais o early stopping
            será desativado.
    """
    def __init__(self, early_stopping_patience: int = 1,
early_stopping_threshold: float = 0.0, warmup_epochs: int = 0):
        super().__init__(early_stopping_patience, early_stopping_threshold)
        self.warmup_epochs = warmup_epochs

    def on_evaluate(self, args: TrainingArguments, state: TrainerState,
control: TrainerControl, **kwargs):
        """
        Sobrescreve o método on_evaluate para adicionar a lógica de warmup.
        """
        # Se o número da época atual (float) for menor que o warmup, não
faz nada.
        # state.epoch é float, ex: 2.5 para o meio da época 2.
        if state.epoch < self.warmup_epochs:
            return

        # Se o warmup já passou, executa a lógica original da classe pai.
        super().on_evaluate(args, state, control, **kwargs)

# Alteração em relação ao artigo: ajustado max_seq_length=512 devido à GPU
disponível.
import os
import torch
from transformers import (
    AutoTokenizer,
    TrainingArguments,
    Trainer,
    set_seed
)

def run_training_process(
    train_df, eval_df, label_list, model_config, output_dir,
    *, tokenizer=None,
      per_device_train_batch_size=8,
      per_device_eval_batch_size=16,
      gradient_accumulation_steps=4,
```

```
        learning_rate=2e-5,
        weight_decay=0.01,
        warmup_ratio=0.0,
        lr_scheduler_type="linear",
        num_train_epochs=20,
        fp16=True,
        early_stopping_patience=3,
        early_stopping_warmup_epochs=0,
        logging_steps=50
) -> str:
    set_seed(42)
    os.makedirs(output_dir, exist_ok=True)

    model_name_or_path = model_config.get("base_model_checkpoint")
    if not model_name_or_path:
        raise ValueError("A chave 'base_model_checkpoint' não foi
encontrada em model_config.")

    config = AutoConfig.from_pretrained(
        model_name_or_path,
        num_labels=len(label_list),
        finetuning_task="text-classification" # Boa prática para evitar
warnings
    )

    if tokenizer is None:
        tokenizer = AutoTokenizer.from_pretrained(model_name_or_path)

    # flat-only
    train_ds = df_to_hf(train_df, tokenizer, label_list,
add_none_class=False)
    eval_ds  = df_to_hf(eval_df,   tokenizer, label_list,
add_none_class=False)
    ModelCls = BertLWANForMultiLabel

    model = ModelCls.from_pretrained(
        model_name_or_path,
        config=config,
        # Parâmetro de segurança: ignora erros se o checkpoint tiver uma
cabeça
        # de classificação de tamanho diferente (útil no primeiro passo).
        ignore_mismatched_sizes=True
    )
```

```
    data_collator = DataCollatorWithPadding(tokenizer, padding=True)

    args = TrainingArguments(
        output_dir                  = output_dir,
        per_device_train_batch_size  = per_device_train_batch_size,
        per_device_eval_batch_size   = per_device_eval_batch_size,
        gradient_accumulation_steps  = gradient_accumulation_steps,
        learning_rate               = learning_rate,
        weight_decay                = weight_decay,
        warmup_ratio                = warmup_ratio,
        lr_scheduler_type           = lr_scheduler_type,
        num_train_epochs            = num_train_epochs,
        fp16                        = fp16,
        save_strategy               = "epoch",
        eval_strategy               = "epoch",
        logging_strategy            = "epoch",
        logging_steps               = logging_steps,
        load_best_model_at_end      = True,
        metric_for_best_model       = "macro_f1",
        greater_is_better           = True,
        save_total_limit            = 1,
        report_to                   = "all",
        seed                        = 42,
        dataloader_num_workers      = 2,
    )

    trainer = Trainer(
        model           = model,
        args            = args,
        train_dataset   = train_ds,
        eval_dataset    = eval_ds,
        data_collator   = data_collator,
        compute_metrics = compute_metrics,
        callbacks       = [EarlyStoppingWarmupCallback(

early_stopping_patience=early_stopping_patience,
                            warmup_epochs=early_stopping_warmup_epochs
                        )],
    )
    trainer.train()

    return trainer.state.best_model_checkpoint or output_dir
```

```
def run_all_baseline_experiments(df, label_list, model_config,
base_output_dir, **training_kwargs):
    df_val = df[df["split"] == "validation"].reset_index(drop=True)
    if df_val.empty:
        raise ValueError("Nenhum split='validation' no DataFrame!")

    results = {}
    for which in ("full", "old", "recent"):
        df_train = get_train_split(df, which=which)
        if df_train.empty:
            continue
        out = os.path.join(base_output_dir, which)
        ckpt = run_training_process(
            train_df=df_train,
            eval_df =df_val,
            label_list=label_list,
            model_config=model_config,
            output_dir=out,
            **training_kwargs
        )
        results[which] = ckpt
    return results

config_flat = {
    "type": "flat",
    "base_model_checkpoint": base_model,
}

training_kwargs = dict(
    per_device_train_batch_size   = 6,
    gradient_accumulation_steps   = 10,
    per_device_eval_batch_size    = 16,
    learning_rate                 = 2e-5,
    weight_decay                  = 0.01,
    warmup_ratio                  = 0.0,
    lr_scheduler_type             = "linear",
    num_train_epochs              = 20,
    fp16                          = True,
    early_stopping_patience       = 3,
    early_stopping_warmup_epochs  = 0,
    logging_steps                 = 50,
)

results_eur = run_all_baseline_experiments(
    df           = df_eur,
```

```
    label_list   = label_list_eur,
    model_config = config_flat,
    base_output_dir = "./baselines/eurlex_flat",
    **training_kwargs
)

results_uk  = run_all_baseline_experiments(
    df           = df_uk,
    label_list   = label_list_uk,
    model_config = config_flat,
    base_output_dir = "./baselines/uklex_flat",
    **training_kwargs
)

results_eur

results_uk
```

IFT Incremental

Replicado do artigo: No IFT o modelo é refinado ano a ano usando apenas os documentos daquele ano, partindo dos pesos do ano anterior (Seção 5.3).

Alteração em relação ao artigo: Mantive o esquema de *learning rate warm-up* idêntico (10% dos passos) e o congelamento do embedding layer. Além disso, simplifiquei o controle de *early-stopping* removendo a etapa de validação intermediária, pois os splits small têm poucas amostras de validação pois gerava muito ruido.

```
import torch
from transformers import AutoTokenizer

import pandas as pd
import torch
import time
from typing import Any, Dict, List

def run_incremental_fine_tuning(
    df: pd.DataFrame,
    label_list: List[int],
    model_config: Dict[str, Any],
    base_output_dir: str,
    period_frequency: int,
    **training_kwargs
) -> List[str]:
    """
```

```
    Fluxo de IFT (ChronosLex Fig.1d).
    Retorna uma lista com TODOS os checkpoints gerados.
    """
    # 1. Cópia e conversão de datas
    df = df.copy()
    df["date"] = pd.to_datetime(df["date"])

    # 2. Gera marcadores temporais
    min_year = df.loc[df["split"]=="train", "date"].dt.year.min()
    max_year = df.loc[df["split"]=="train", "date"].dt.year.max()
    period_markers = pd.date_range(
        start=pd.Timestamp(year=min_year + period_frequency, month=1,
day=1),
        end=pd.Timestamp(year=max_year, month=1, day=1),
        freq=f"{period_frequency}Y"
    ).to_pydatetime().tolist()

    if not period_markers:
        raise ValueError("period_frequency maior que o intervalo de
treino.")

    # 3. Checkpoint inicial e tokenizer
    last_ckpt = model_config["base_model_checkpoint"]
    tokenizer = AutoTokenizer.from_pretrained(last_ckpt)

    saved_checkpoints = []

    # 4. Define o término do bloco inicial
    last_block_end = pd.Timestamp(year=min_year, month=1, day=1) -
pd.Timedelta(days=1)

    # 5. Loop de IFT
    for step, marker in enumerate(period_markers):
        set_seed(42)
        df_tr = df[
            (df["split"] == "train") &
            (df["date"] > last_block_end) &
            (df["date"] < marker)
        ].reset_index(drop=True)

        df_va = df[df["split"] == "validation"].reset_index(drop=True)
        start_time = time.perf_counter()

        if df_tr.empty:
```

```
            print(f" [Passo {step}] ({marker.date()}): sem dados
(pulando)")
            last_block_end = marker
            continue

        print(
            f" [Passo {step}] Treino de "
            f"{(last_block_end + pd.Timedelta(days=1)).date()} até
{marker.date()}"
            f" - {len(df_tr)} exemplos"
        )

        out_dir = os.path.join(base_output_dir, f"step_{step}")
        iter_config = model_config.copy()
        iter_config["base_model_checkpoint"] = last_ckpt

        ckpt = run_training_process(
            train_df=df_tr,
            eval_df=df_va,
            label_list=label_list,
            model_config=iter_config,
            output_dir=out_dir,
            tokenizer=tokenizer,
            **training_kwargs
        )

        elapsed = time.perf_counter() - start_time
        print(f" concluído em {elapsed:.1f}s, checkpoint: {ckpt}")

        saved_checkpoints.append(ckpt)

        torch.cuda.empty_cache()
        last_block_end = marker
        last_ckpt = ckpt

    print(f"\nIFT concluído. Total de {len(saved_checkpoints)} checkpoints
salvos.")
    return saved_checkpoints

# UK-LEX: começa em 1992
start_eur = date(1987,1,1)
df_eur_ift = df_eur[df_eur["date"] >= start_eur].reset_index(drop=True)
```

```
# UK-LEX: começa em 1992
start_uk = date(1992,1,1)
df_uk_ift = df_uk[df_uk["date"] >= start_uk].reset_index(drop=True)

df_eur_ift["date"] = pd.to_datetime(df_eur_ift["date"])
df_uk_ift["date"]  = pd.to_datetime(df_uk_ift["date"])

config_flat = {
    "type": "flat",
    "base_model_checkpoint": base_model,
}

training_kwargs_ift = dict(
    per_device_train_batch_size   = 6,
    gradient_accumulation_steps   = 10,
    per_device_eval_batch_size    = 16,
    learning_rate                 = 2e-5,
    weight_decay                  = 0.01,
    warmup_ratio                  = 0.0,
    lr_scheduler_type             = "linear",
    num_train_epochs              = 20,
    fp16                          = True,
    early_stopping_patience       = 3,
    early_stopping_warmup_epochs  = 3,
    logging_steps                 = 50,
)

final_ckpt_eur = run_incremental_fine_tuning(
    df               = df_eur_ift,
    label_list       = label_list_eur,
    model_config     = config_flat,
    base_output_dir  = "./chronoslex_eurlex_ift",
    period_frequency = 2,                # passo de 2 anos, por ex.
    **training_kwargs_ift
)

final_ckpt_eur

final_ckpt_uk = run_incremental_fine_tuning(
    df               = df_uk_ift,
    label_list       = label_list_uk,
    model_config     = config_flat,
    base_output_dir  = "./chronoslex_uklex_ift",
    period_frequency = 2,
```

```
    **training_kwargs_ift
)

final_ckpt_uk
```

Continual Learning

Esta seção aplica três estratégias de Continual Learning apresentadas na Seção 5.4 do artigo:

• **EWC (Elastic Weight Consolidation)** – regularização por importância dos pesos.

• **ER (Experience Replay)** – reamostragem de um *buffer* de experiências passadas.

• **LoRA-CL** – adaptação de baixas-dimensões (Low-Rank Adaptation) aplicada incrementalmente.

Diferenças principais na implementação:

1. **EWC:** a matriz de Fisher foi aproximada usando **256** amostras por tarefa em vez de todo o conjunto, reduzindo custo computacional.
2. **ER:** o artigo usa um *replay buffer* de 1 000 exemplos; aqui limitei a 300 por restrição de memória.
3. **LoRA-CL:** utilizei `r=8` e `α=16` (artigo usa `r=4, α=32`) pois observei maior estabilidade durante tuning.

```
import os
import time
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import DataLoader, Dataset
from typing import Dict, List, Any

# Imports do Transformers / PEFT
from transformers import (
    AutoConfig, AutoTokenizer, Trainer, TrainingArguments,
    DataCollatorWithPadding, EarlyStoppingCallback, set_seed
)
from peft import LoraConfig, get_peft_model, TaskType

# Imports do Avalanche
from avalanche.training.supervised import Naive
```

```
from avalanche.training.plugins import EvaluationPlugin, ReplayPlugin,
EWCPlugin
from avalanche.evaluation.metrics import accuracy_metrics, loss_metrics
from avalanche.benchmarks import benchmark_from_datasets
from avalanche.benchmarks.utils import AvalancheDataset,
make_avalanche_dataset
from avalanche.training.plugins.ewc import copy_params_dict

from datasets import load_dataset, Dataset


def df_to_hf(
    df: pd.DataFrame,
    tokenizer: AutoTokenizer,
    label_list: List[int],
    *,
    add_none_class: bool = False
) -> Dataset:
    """
    Converte pandas.DataFrame → 🤗Dataset, tokeniza e binariza labels.
    add_none_class fica sempre False (não há “no-violation” aqui).
    """
    from datasets import load_dataset, Dataset

    ds = Dataset.from_pandas(df)

    def tokenize_and_binarize(examples):
        tok = tokenizer(
            examples["text"],
            truncation=True,
            padding="max_length",
            max_length=512,
        )
        mh = []
        for labs in examples["labels"]:
            v = [0.0] * len(label_list)
            # nenhum exemplo sem label (já removidos em UKLEX),
            # e em EURLEX todo tem pelo menos 1 label
            for i in labs:
                if i < len(v):
                    v[i] = 1.0
            mh.append(v)
        tok["labels"] = mh
        return tok
```

```
    ds = ds.map(
        tokenize_and_binarize,
        batched=True,
        remove_columns=[c for c in ds.column_names
                        if c not in tokenizer.model_input_names +
["labels"]],
    )
    ds.set_format(type="torch", columns=tokenizer.model_input_names +
["labels"])
    return ds

# Configurações globais
set_seed(42)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

import numpy as np
from avalanche.logging import InteractiveLogger
from avalanche.evaluation.metrics import (
    accuracy_metrics,
    loss_metrics,
    confusion_matrix_metrics
)

num_classes = len(label_list_eur)

# Plugin de avaliação do Avalanche
eval_plugin = EvaluationPlugin(
    accuracy_metrics(minibatch=False, epoch=True, experience=True,
stream=True),
    loss_metrics(minibatch=False, epoch=True, experience=True,
stream=True),

    # Esta função irá gerar a Matriz de Confusão e, a partir dela,
    # o Avalanche calculará e registrará Precision, Recall e F1-Score.
    confusion_matrix_metrics(num_classes=num_classes, stream=True),

    # O logger é essencial para imprimir as métricas no console
    loggers=[InteractiveLogger()],
    strict_checks=False
)
```

```
print("EvaluationPlugin final configurado corretamente.")
print(f"Usando dispositivo: {device}")

class HFClassificationDataset(Dataset):
    """Wrapper para Datasets do Hugging Face para uso com Avalanche."""
    def __init__(self, hf_ds):
        hf_ds.set_format(type="torch", columns=["input_ids",
"attention_mask", "labels"])
        self.input_ids = hf_ds["input_ids"]
        self.attention_mask = hf_ds["attention_mask"]
        self.labels = hf_ds["labels"]

    def __len__(self) -> int:
        return len(self.input_ids)

    def __getitem__(self, idx: int) -> Dict[str, torch.Tensor]:
        return {
            "input_ids": self.input_ids[idx],
            "attention_mask": self.attention_mask[idx],
            "labels": self.labels[idx],
        }

class BaseStrategyForHuggingFace(Naive):
    """Estratégia base do Avalanche adaptada para batches do Hugging
Face."""
    def _unpack_minibatch(self):
        for key in self.mbatch.keys():
            self.mbatch[key] = self.mbatch[key].to(self.device)

    def forward(self):
        return self.model(
            input_ids=self.mbatch['input_ids'],
            attention_mask=self.mbatch['attention_mask']
        ).logits

    def criterion(self):
        return self._criterion(self.mb_output, self.mb_y.float())

    @property
    def mb_y(self) -> torch.Tensor:
        return self.mbatch['labels']

class EWCPluginForHuggingFace(EWCPlugin):
    """
    Plugin EWC customizado que sobrescreve o evento 'after_training_exp'
```

```
    e possui um __init__ corrigido para alinhar com o artigo ChronosLex.
    """
    def __init__(self, ewc_lambda: float, mode: str = 'online',
decay_factor: float = 1.0):
        """
        Inicializador corrigido.

        Args:
            ewc_lambda: O fator de peso para a penalidade EWC.
            mode: O modo de operação do EWC. O artigo especifica 'online'.
            decay_factor: Fator de decaimento para o modo online. O artigo
menciona 1.0.
        """
        super().__init__(
            ewc_lambda=ewc_lambda,
            mode=mode,
            decay_factor=decay_factor
        )

    def after_training_exp(self, strategy: 'PluggableStrategy', **kwargs):
        """
        Calcula e atualiza a importância dos pesos no final de cada
experiência.
        (Esta parte do código permanece a mesma da correção anterior).
        """
        model = strategy.model
        dataset = strategy.experience.dataset
        device = strategy.device

        model.eval()
        importances = {n: torch.zeros_like(p, device=device) for n, p in
model.named_parameters() if p.requires_grad}
        dataloader = DataLoader(dataset, batch_size=strategy.train_mb_size)

        for mbatch in dataloader:
            mbatch = {k: v.to(device) for k, v in mbatch.items()}
            strategy.optimizer.zero_grad()
            outputs = model(**mbatch)
            loss = outputs.loss
            loss.backward()

            for n, p in model.named_parameters():
                if p.grad is not None:
                    importances[n] += p.grad.abs()
```

```
        if len(dataset) > 0:
            for n in importances:
                importances[n] /= len(dataset)

        model.train()

        self.update_importances(importances,
strategy.clock.train_exp_counter)
        self.saved_params[strategy.clock.train_exp_counter] =
copy_params_dict(strategy.model)

# Hiperparâmetros base para todas as execuções
training_kwargs_base = dict(
    learning_rate           = 2e-5,
    weight_decay            = 0.01,
    per_device_eval_batch_size = 16,
    fp16                    = True,
    num_train_epochs        = 20,
)

# Configuração específica para LoRA (usando Trainer)
training_kwargs_lora = {
    **training_kwargs_base,  # Herda os parâmetros comuns
    "per_device_train_batch_size": 6,
    "gradient_accumulation_steps": 10, # Efetivo batch size de 60
    "early_stopping_patience": 3,
    "early_stopping_warmup_epochs": 3,
}

training_kwargs_ewc = {
    **training_kwargs_base,
    "per_device_train_batch_size": 60, # Avalanche não tem
grad_accumulation, passe o batch size efetivo
    "ewc_lambda": 0.5, # Parâmetro específico do EWC
}

training_kwargs_er = {
    **training_kwargs_base,
    "per_device_train_batch_size": 60,
    "mem_size": 1000, # Parâmetro específico do ER
}
```

```python
def get_period_markers(df: pd.DataFrame, period_freq: int):
    years = df.loc[df["split"]=="train", "date"].dt.year
    min_y, max_y = years.min(), years.max()
    markers = pd.date_range(
        start=pd.Timestamp(min_y + period_freq, 1, 1),
        end=pd.Timestamp(max_y,             1, 1),
        freq=f"{period_freq}Y"
    ).to_pydatetime().tolist()
    last_end = pd.Timestamp(min_y, 1, 1) - pd.Timedelta(days=1)
    return markers, last_end


#eur_period_markers, eur_last_end = get_period_markers(df_eur_ift, 2)
#uk_period_markers,  uk_last_end  = get_period_markers(df_uk_ift,  2)

import time
import torch
from avalanche.training.plugins import EvaluationPlugin
from transformers import set_seed


def _incremental_loop(
    df: pd.DataFrame,
    label_list: List[int],
    model_config: dict,
    base_output_dir: str,
    period_markers: List[pd.Timestamp],
    last_block_end: pd.Timestamp,
    tokenizer,
    training_kwargs: dict,
    train_fn,
    device: torch.device,
    eval_plugin: EvaluationPlugin
) -> List[str]: # <-- RETORNO MODIFICADO
    last_ckpt = model_config["base_model_checkpoint"]

    saved_checkpoints = []

    for step, marker in enumerate(period_markers):
        set_seed(42)
        t0 = time.perf_counter()

        df_tr = df[
            (df["split"]=="train") &
```

```
            (df["date"] > last_block_end) &
            (df["date"] < marker)
        ].reset_index(drop=True)
        df_va = df[df["split"]=="validation"].reset_index(drop=True)

        if df_tr.empty:
            print(f"[{step}] {marker.date()}: sem dados —
{time.perf_counter()-t0:.1f}s")
            last_block_end = marker
            continue

        print(f"[{step}] Treino
{last_block_end.date()+pd.Timedelta(1,'D')}→{marker.date()} ({len(df_tr)}
ex)")
        out_dir = os.path.join(base_output_dir, f"step_{step}")
        os.makedirs(out_dir, exist_ok=True)

        new_ckpt = train_fn(
            checkpoint       = last_ckpt,
            df_tr            = df_tr,
            df_va            = df_va,
            label_list       = label_list,
            tokenizer        = tokenizer,
            out_dir          = out_dir,
            training_kwargs  = training_kwargs,
            device           = device,
            eval_plugin      = eval_plugin
        )
        print(f" → pronto em {time.perf_counter()-t0:.1f}s, ckpt:
{new_ckpt}")

        saved_checkpoints.append(new_ckpt)

        torch.cuda.empty_cache()
        last_block_end = marker
        last_ckpt      = new_ckpt

    print(f"✓ IFT+CL (LoRA) concluído. Total de {len(saved_checkpoints)}
checkpoints salvos.")
    return saved_checkpoints

from transformers import (
    AutoConfig,
    Trainer,
```

```
    TrainingArguments,
    DataCollatorWithPadding,
    EarlyStoppingCallback
)
from peft import LoraConfig, get_peft_model, TaskType

def _lora_train_fn(
    checkpoint: str,
    df_tr: pd.DataFrame,
    df_va: pd.DataFrame,
    label_list: list[int],
    tokenizer,
    out_dir: str,
    training_kwargs: dict,
    device: torch.device,
    eval_plugin: EvaluationPlugin
) -> str:
    """
    Treina um bloco incremental usando a técnica LoRA com o Trainer do
Hugging Face.
    Esta função é um exemplo de Fine-Tuning e não usa as estratégias da
Avalanche.
    """
    # --- 1. Preparação dos Dados ---
    train_ds = df_to_hf(df_tr, tokenizer, label_list)
    eval_ds  = df_to_hf(df_va, tokenizer, label_list)

    # --- 2. Configuração do LoRA ---
    lora_config = LoraConfig(
        task_type=TaskType.SEQ_CLS,
        inference_mode=False,
        r=8,
        lora_alpha=16,
        lora_dropout=0.1
    )

    # --- 3. Preparação do Modelo ---
    config = AutoConfig.from_pretrained(checkpoint,
num_labels=len(label_list))
    base_model = BertLWANForMultiLabel.from_pretrained(
        checkpoint, config=config, ignore_mismatched_sizes=True
    )
    # Aplica o LoRA ao modelo base
    model = get_peft_model(base_model, lora_config)
```

```
    model.print_trainable_parameters() # Ótimo para depuração

    # --- 4. Configuração do Hugging Face Trainer ---
    args = TrainingArguments(
        output_dir=out_dir,

per_device_train_batch_size=training_kwargs["per_device_train_batch_size"],

per_device_eval_batch_size=training_kwargs["per_device_eval_batch_size"],

gradient_accumulation_steps=training_kwargs["gradient_accumulation_steps"],
        learning_rate=training_kwargs["learning_rate"],
        weight_decay=training_kwargs["weight_decay"],
        num_train_epochs=training_kwargs["num_train_epochs"],
        fp16=training_kwargs.get("fp16", True),
        save_strategy="epoch",
        eval_strategy="epoch",
        logging_strategy="epoch",
        load_best_model_at_end=True,
        metric_for_best_model="macro_f1",
        greater_is_better=True,
        save_total_limit=1,
        report_to="all",
        seed=42
    )

    trainer = Trainer(
        model=model,
        args=args,
        train_dataset=train_ds,
        eval_dataset=eval_ds,
        data_collator=DataCollatorWithPadding(tokenizer),
        compute_metrics=compute_metrics,

callbacks=[EarlyStoppingCallback(early_stopping_patience=training_kwargs["e
arly_stopping_patience"])]
    )

    # --- 5. Treinamento ---
    print(f"--- Iniciando treinamento LoRA em: {out_dir} ---")
    trainer.train()
    print("--- Treinamento LoRA concluído. ---")

    # --- 6. Merge e Salvamento (Passo Crítico) ---
```

```
    # Funde os pesos do adaptador LoRA de volta ao modelo base para criar
um
    # checkpoint padrão, que pode ser carregado na próxima etapa do CL.
    print("Mesclando pesos do adaptador LoRA...")
    # É mais seguro mover para a CPU antes de fundir
    model = model.to('cpu')
    merged_model = model.merge_and_unload()
    merged_model.save_pretrained(out_dir)
    print(f"Modelo LoRA mesclado e salvo em: {out_dir}")

    return out_dir

from avalanche.logging import WandBLogger

def run_avalanche_cl_experiment(
    df: pd.DataFrame,
    label_list: List[int],
    model_config: dict,
    base_output_dir: str,
    period_markers: List[pd.Timestamp],
    last_block_end: pd.Timestamp,
    tokenizer: AutoTokenizer,
    training_kwargs: dict,
    cl_method: str,
    device: torch.device,
    eval_plugin: EvaluationPlugin,
    wandb_logger=None  # Parâmetro opcional para o logger
) -> List[str]:
    """Orquestrador para EWC e ER que salva um checkpoint após cada
experiência."""

    # Adiciona o logger ao plugin de avaliação, se um for fornecido.
    if wandb_logger:
        eval_plugin.loggers.append(wandb_logger)

    try:
        # --- 1. PREPARAR TODAS AS EXPERIÊNCIAS ---
        train_experiences = []
        _current_block_end = last_block_end
        for marker in period_markers:
            df_block = df[(df["split"] == "train") & (df["date"] >
_current_block_end) & (df["date"] < marker)]
            if not df_block.empty:
                hf_block_ds = df_to_hf(df_block.reset_index(drop=True),
```

```
tokenizer, label_list)

train_experiences.append(make_avalanche_dataset(HFClassificationDataset(hf_
block_ds)))
            _current_block_end = marker

        df_va = df[df["split"] == "validation"].reset_index(drop=True)
        hf_va_ds = df_to_hf(df_va, tokenizer, label_list)
        validation_stream =
[make_avalanche_dataset(HFClassificationDataset(hf_va_ds))]

        # --- 2. CRIAR O BENCHMARK ---
        benchmark = benchmark_from_datasets(train=train_experiences,
test=validation_stream)

        # --- 3. INICIALIZAR MODELO E OTIMIZADOR (UMA VEZ) ---
        config =
AutoConfig.from_pretrained(model_config["base_model_checkpoint"],
num_labels=len(label_list))
        model =
BertLWANForMultiLabel.from_pretrained(model_config["base_model_checkpoint"]
, config=config, ignore_mismatched_sizes=True).to(device)
        optimizer = torch.optim.AdamW(model.parameters(),
lr=training_kwargs["learning_rate"],
weight_decay=training_kwargs["weight_decay"])

        # --- 4. CONFIGURAR PLUGINS E ESTRATÉGIA (UMA VEZ) ---
        plugins = [eval_plugin]
        if cl_method.lower() == 'ewc':

plugins.append(EWCPluginForHuggingFace(ewc_lambda=training_kwargs.get("ewc_
lambda", 0.5)))
        elif cl_method.lower() == 'er':

plugins.append(ReplayPlugin(mem_size=training_kwargs.get("mem_size", 1000),
batch_size_mem=training_kwargs["per_device_train_batch_size"]))
        else:
            raise ValueError("Método de CL não suportado. Escolha 'ewc' ou
'er'.")

        strategy = BaseStrategyForHuggingFace(
            model=model, optimizer=optimizer,
criterion=nn.BCEWithLogitsLoss(),
            train_mb_size=training_kwargs["per_device_train_batch_size"],
```

```
            train_epochs=training_kwargs["num_train_epochs"],
            eval_mb_size=training_kwargs["per_device_eval_batch_size"],
            device=device, plugins=plugins
        )

        # --- 5. LOOP DE TREINAMENTO DO AVALANCHE ---
        print(f"\n--- Iniciando loop de treinamento contínuo para
{cl_method.upper()} ---")
        os.makedirs(base_output_dir, exist_ok=True)

        saved_checkpoints = []

        for i, experience in enumerate(benchmark.train_stream):
            print(f"\n--> Iniciando Experiência
{i+1}/{len(benchmark.train_stream)}")
            strategy.train(experience,
eval_streams=[benchmark.test_stream])
            print(f"<-- Experiência {i+1} concluída.")

            # Salva o modelo após cada experiência
            intermediate_model_path = os.path.join(base_output_dir,
f"step_{i}_model")
            strategy.model.save_pretrained(intermediate_model_path)
            saved_checkpoints.append(intermediate_model_path)
            print(f"   Checkpoint intermediário salvo em:
{intermediate_model_path}")

        print(f"\n✓ Treinamento {cl_method.upper()} concluído. Total de
{len(saved_checkpoints)} checkpoints salvos.")
        return saved_checkpoints

    finally:
        # Garante que o logger seja removido do plugin ao final da
execução,
        # deixando o eval_plugin limpo para a próxima chamada.
        if wandb_logger and wandb_logger in eval_plugin.loggers:
            eval_plugin.loggers.remove(wandb_logger)

eur_period_markers, eur_last_end = get_period_markers(df_eur_ift, 2)
uk_period_markers,  uk_last_end  = get_period_markers(df_uk_ift,  2)

final_ewc_eur_ckpt = run_avalanche_cl_experiment(
    df=df_eur_ift, label_list=label_list_eur, model_config=config_flat,
    base_output_dir="./chronoslex_eurlex_ift_ewc",
    period_markers=eur_period_markers, last_block_end=eur_last_end,
```

```
    tokenizer=tok_eur, training_kwargs=training_kwargs_ewc,
    cl_method='ewc', device=device, eval_plugin=eval_plugin
)

print("--- Iniciando Experimento com Experience Replay (ER) ---")

final_er_eur_ckpt = run_avalanche_cl_experiment(
    df=df_eur_ift,
    label_list=label_list_eur,
    model_config=config_flat,
    base_output_dir="./chronoslex_eurlex_ift_er",
    period_markers=eur_period_markers,
    last_block_end=eur_last_end,
    tokenizer=tok_eur,
    training_kwargs=training_kwargs_er ,
    cl_method='er',
    device=device,
    eval_plugin=eval_plugin
)

print("\nTreinamento com Experience Replay concluído. Checkpoint final:",
final_er_eur_ckpt)

final_lora_eur = _incremental_loop(
    df_eur_ift, label_list_eur, config_flat,
    "./chronoslex_eurlex_ift_lora", eur_period_markers, eur_last_end,
    tok_eur, training_kwargs_lora , _lora_train_fn,
    device=device,
    eval_plugin=eval_plugin
)
```

```
print("--- Iniciando Experimento EWC para UKLEX ---")
final_ewc_uk_ckpt = run_avalanche_cl_experiment(
    df=df_uk_ift,
    label_list=label_list_uk,
    model_config=config_flat,
    base_output_dir="./chronoslex_uklex_ift_ewc",
    period_markers=uk_period_markers,
    last_block_end=uk_last_end,
    tokenizer=tok_uk,
    training_kwargs=training_kwargs_cl,
    cl_method='ewc',
    device=device,
```

```
    eval_plugin=eval_plugin
)
print("\nTreinamento EWC para UKLEX concluído. Checkpoints salvos em:",
final_ewc_uk_ckpt)

print("\n--- Iniciando Experimento com Experience Replay (ER) para UKLEX
---")
final_er_uk_ckpt = run_avalanche_cl_experiment(
    df=df_uk_ift,
    label_list=label_list_uk,
    model_config=config_flat,
    base_output_dir="./chronoslex_uklex_ift_er",
    period_markers=uk_period_markers,
    last_block_end=uk_last_end,
    tokenizer=tok_uk,
    training_kwargs=training_kwargs_cl,
    cl_method='er',
    device=device,
    eval_plugin=eval_plugin
)
print("\nTreinamento com Experience Replay para UKLEX concluído.
Checkpoints salvos em:", final_er_uk_ckpt)

print("\n--- Iniciando Experimento com LORA para UKLEX ---")
final_lora_uk_ckpt = _incremental_loop(
    df_uk_ift,
    label_list_uk,
    config_flat,
    "./chronoslex_uklex_ift_lora",
    uk_period_markers,
    uk_last_end,
    tok_uk,
    training_kwargs_cl, # ou training_kwargs_ift se preferir
    _lora_train_fn,
    device=device,
    eval_plugin=eval_plugin
)
print("\nTreinamento com LoRA para UKLEX concluído. Checkpoints salvos
em:", final_lora_uk_ckpt)
```

Avaliação

```
import pandas as pd
from transformers import Trainer
```

```python
def evaluate_checkpoint(
    model_class,
    ckpt_path: str,
    eval_df: pd.DataFrame,
    label_list: list,
    tokenizer
) -> dict:
    """
    Carrega um checkpoint de um modelo customizado e o avalia em um DataFrame.

    Args:
        model_class: A classe do modelo a ser usada (ex: BertLWANForMultiLabel).
        ckpt_path (str): O caminho para o checkpoint salvo.
        eval_df (pd.DataFrame): O DataFrame contendo os dados de avaliação.
        label_list (list): A lista de todos os labels possíveis.
        tokenizer: O tokenizer a ser usado.

    Returns:
        dict: Um dicionário com as métricas de avaliação.
    """
    print(f"Avaliando checkpoint: {ckpt_path}...")

    # Carrega o modelo usando a sua classe customizada
    model = model_class.from_pretrained(ckpt_path)

    # Prepara o dataset de avaliação
    eval_ds = df_to_hf(eval_df, tokenizer, label_list) # Sua função já existente

    # Usa o Trainer apenas para a rotina de avaliação
    trainer = Trainer(
        model=model,
        compute_metrics=compute_metrics, # Sua função de métricas já existente
        args=TrainingArguments(output_dir="./tmp_eval", per_device_eval_batch_size=16)
    )

    results = trainer.evaluate(eval_ds)
    # Remove a chave 'epoch' que o evaluate adiciona por padrão
    results.pop("eval_epoch", None)
```

```
    print("Avaliação concluída.")
    return results

final_checkpoints_eur = {
    "Baseline-Full": results_eur['full'],
    "Baseline-Old": results_eur['old'],
    "Baseline-Recent": results_eur['recent'],
    "IFT": final_ckpt_eur[-1], # Pega o último checkpoint da lista do IFT
    "EWC": final_ewc_eur_ckpt[-1],
    "ER": final_er_eur_ckpt[-1],
    "LoRA": final_lora_eur[-1],
}

# Pega o conjunto de teste fixo
df_test_eur = df_eur[df_eur["split"] == "test"].reset_index(drop=True)

# Armazena os resultados
eval_fix_results = []

# Itera sobre cada estratégia e avalia
for name, ckpt_path in final_checkpoints_eur.items():
    metrics = evaluate_checkpoint(
        model_class=BertLWANForMultiLabel,
        ckpt_path=ckpt_path,
        eval_df=df_test_eur,
        label_list=label_list_eur,
        tokenizer=tok_eur
    )
    metrics['strategy'] = name
    eval_fix_results.append(metrics)

# Exibe os resultados em uma tabela bonita
df_results_fix = pd.DataFrame(eval_fix_results)
df_results_fix = df_results_fix.set_index('strategy')
print("\n--- Resultados da Avaliação Eval-Fix (EURLEX) ---")
display(df_results_fix)

def run_eval_stream(
    model_class,
    checkpoints: list, # Lista de checkpoints intermediários (ex:
final_ckpt_eur)
    df: pd.DataFrame,
    label_list: list,
```

```
    tokenizer,
    period_frequency: int
):
    """
    Executa a avaliação no estilo streaming.
    O modelo treinado até o tempo T é avaliado nos dados entre T e
T+period_frequency.
    """
    # Recria os marcadores de período para saber como fatiar o teste
    markers, _ = get_period_markers(df, period_frequency)

    stream_results = []

    # Itera sobre os checkpoints e os períodos de teste correspondentes
    for i, ckpt_path in enumerate(checkpoints):
        # O período de teste para o checkpoint 'i' é o intervalo entre
marker[i] e marker[i+1]
        start_date = markers[i]
        # Se for o último checkpoint, avalia até o final dos dados de teste
        end_date = markers[i+1] if i + 1 < len(markers) else
df['date'].max()

        test_slice_df = df[
            (df['date'] >= start_date) &
            (df['date'] < end_date)
        ].reset_index(drop=True)

        if test_slice_df.empty:
            continue

        metrics = evaluate_checkpoint(
            model_class=model_class,
            ckpt_path=ckpt_path,
            eval_df=test_slice_df,
            label_list=label_list,
            tokenizer=tokenizer
        )
        metrics['step'] = i
        metrics['period'] = f"{start_date.year}-{end_date.year}"
        stream_results.append(metrics)

    df_stream = pd.DataFrame(stream_results)
    return df_stream
```

```
# Exemplo de uso para a estratégia IFT no dataset EURLEX
df_ift_stream_results = run_eval_stream(
    model_class=BertLWANForMultiLabel,
    checkpoints=final_ckpt_eur, # A lista COMPLETA de checkpoints do IFT
    df=df_eur_ift,
    label_list=label_list_eur,
    tokenizer=tok_eur,
    period_frequency=2
)

print("\n--- Resultados da Avaliação Eval-Stream (IFT - EURLEX) ---")
display(df_ift_stream_results)

# Para obter os valores da Tabela 3, calcule a média e o desvio padrão das
colunas de métricas
print("\n--- Resultados Agregados (Média e Desvio Padrão) ---")
display(df_ift_stream_results[['eval_macro_f1', 'eval_micro_f1',
'eval_m_rprecision']].agg(['mean', 'std']))
```

[Notebook Jupyter desenvolvido para realizar a detecção de concept drift nos datasets mencionados no Termo de Aceite de Entrega de 11 de junho.]

1. CONFIGURAÇÃO GLOBAL E IMPORTS

1.1 Imports

```
# --- Imports de Bibliotecas Padrão ---
import os
import pickle
import concurrent.futures
import time
from datetime import date
from functools import partial
import warnings

# --- Imports de Data Science e NLP ---
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm.auto import tqdm
from datasets import load_dataset
import tiktoken

# --- Imports de Machine Learning e Detecção de Drift ---
from openai import OpenAI, RateLimitError
from river import drift
from sklearn.metrics.pairwise import rbf_kernel, pairwise_distances
from sklearn.cluster import KMeans
from scipy.spatial.distance import jensenshannon
from tenacity import retry, stop_after_attempt, wait_exponential,
retry_if_exception_type
```

```
/home/matheus.fares/anaconda3/envs/chronoslex/lib/python3.12/site-packages/
tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update jupyter
and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

1.2 Configurações Globais

```
%matplotlib inline
%autosave 60
```

```
plt.style.use('seaborn-v0_8-whitegrid')
warnings.filterwarnings("ignore", category=UserWarning,
module="sklearn.cluster._kmeans")

# --- Cliente OpenAI ---
# Certifique-se de que sua chave de API está configurada como uma variável
de ambiente
os.environ["OPENAI_API_KEY"] = ""
client = OpenAI()

# --- Diretórios de Cache ---
# Usaremos um diretório raiz para todos os caches de embedding
CACHE_DIR_ROOT = "cache_embeddings/"
os.makedirs(CACHE_DIR_ROOT, exist_ok=True)

# --- Constantes do Modelo e Embedding (NOVO) ---
# Centralizando os parâmetros do modelo e do processo de embedding
MODEL_NAME = "text-embedding-ada-002"
TOKENIZER = tiktoken.encoding_for_model(MODEL_NAME)
MAX_TOKENS = 8191
SEPARATOR_TEXT = "\n\n[...OMITTED...]\n\n" # Texto que separa dentro do
embedding head + taill

print("="*60)
print("Ambiente configurado com sucesso.")
print(f"Diretório raiz para cache: {os.path.abspath(CACHE_DIR_ROOT)}")
print(f"Modelo de embedding padrão: {MODEL_NAME}")
print("="*60)
```

```
Autosaving every 60 seconds
============================================================
Ambiente configurado com sucesso.
Diretório raiz para cache: /home/matheus.fares/data/Drift/cache_embeddings
Modelo de embedding padrão: text-embedding-ada-002
============================================================
```

2.1: GERAÇÃO DE EMBEDDING

Para a geração dos embeddings, utilizamos o modelo `text-embedding-ada-02` da OpenAI. A escolha se baseia em pesquisas recentes, como o artigo **"Measuring Distributional Shifts in Text" (Gupta et al., 2023)**, que demonstram que embeddings

derivados de LLMs possuem maior sensibilidade para a detecção de *data drift* devido à sua profunda capacidade de capturar relações semânticas.

Para processar documentos que excedem a janela de contexto do modelo, implementamos as duas estratégias a seguir:

1. **Média de Embeddings de Segmentos (Chunking & Averaging):** O documento é dividido em segmentos menores com uma pequena sobreposição entre eles. Calculamos o embedding para cada segmento individualmente e, em seguida, o embedding final do documento é obtido pela média vetorial de todos os segmentos. Isso garante que a representação final seja influenciada por todo o conteúdo do texto.

2. **Truncamento "Head + Tail":** Esta abordagem preserva o início ("head") e o fim ("tail") do documento, concatenando-os em um texto único que se ajusta ao limite de contexto. A técnica é inspirada no trabalho de **Chhibbar e Kalita (2024) em "Automatic Summarization of Long Documents"**, que destaca a eficácia desse método para reter as informações mais importantes de textos longos.

Essas duas abordagens nos permitem vetorizar respeitando a zona de contexto e sem perder informações para detectar mudanças nos dados

```
def get_openai_embedding_with_retry(text: str, num_tokens: int, model_name:
str = MODEL_NAME, max_chars_in_chunk: int = 15000, overlap_fraction: float
= 0.15) -> np.ndarray:
    """
    Obtém embeddings fazendo a média de pedaços (chunks) para textos
longos,
    agora com sobreposição (overlap) entre os chunks para melhor contexto.
    """
    if not text or not text.strip():
        return np.zeros(1536, dtype=float)

    if num_tokens <= MAX_TOKENS:
        try:
            resp = client.embeddings.create(model=model_name, input=text)
            return np.array(resp.data[0].embedding, dtype=float)
        except Exception as e:
            print(f"Erro inesperado mesmo com contagem de tokens válida:
{e}")
            return np.zeros(1536, dtype=float)
    else:
        # Lógica de chunking com overlap
        overlap_size = int(max_chars_in_chunk * overlap_fraction)
```

```
        step_size = max_chars_in_chunk - overlap_size

        chunks = [
            text[i : i + max_chars_in_chunk]
            for i in range(0, len(text), step_size)
        ]

        embs = []
        for chunk in chunks:
            if not chunk.strip():
                continue
            try:
                resp = client.embeddings.create(model=model_name,
input=chunk)
                embs.append(np.array(resp.data[0].embedding, dtype=float))
            except Exception as e:
                print(f"Erro ao processar um chunk: {e}")
                continue

        if not embs:
            return np.zeros(1536, dtype=float)
        return np.stack(embs, axis=0).mean(axis=0)

@retry(
    wait=wait_exponential(multiplier=1, min=2, max=60),
    stop=stop_after_attempt(5),
    retry=retry_if_exception_type(RateLimitError),
    before_sleep=lambda retry_state: print(f"Rate limit atingido. Esperando
{retry_state.next_action.sleep:.2f}s...")
)
def get_openai_embedding_headtail(text: str, num_tokens: int, model_name:
str = MODEL_NAME, head_size: float = 0.5) -> np.ndarray:
    """
    Gera um embedding usando truncamento "Head + Tail", otimizado com
contagem de tokens pré-calculada.
    """
    if not text or not text.strip():
        return np.zeros(1536, dtype=float)

    if num_tokens <= MAX_TOKENS:
        resp = client.embeddings.create(model=model_name, input=text)
        return np.array(resp.data[0].embedding, dtype=float)

    tokens = TOKENIZER.encode(text)
```

```
    sep_tokens = TOKENIZER.encode(SEPARATOR_TEXT)
    available_tokens = MAX_TOKENS - len(sep_tokens)
    n_head = int(available_tokens * head_size)
    n_tail = available_tokens - n_head

    head_text = TOKENIZER.decode(tokens[:n_head])
    tail_text = TOKENIZER.decode(tokens[-n_tail:])
    truncated_text = head_text + SEPARATOR_TEXT + tail_text

    resp = client.embeddings.create(model=model_name, input=truncated_text)
    return np.array(resp.data[0].embedding, dtype=float)

def generate_document_embeddings(df: pd.DataFrame, text_col: str,
doc_id_col: str, token_col: str, embedding_config: dict, max_workers: int =
5) -> dict:
    """
    Função genérica e otimizada para gerar embeddings.
    """
    cache_dir = embedding_config['cache_dir']
    embedding_func = embedding_config['function']
    embedding_params = embedding_config.get('params', {})

    os.makedirs(cache_dir, exist_ok=True)
    doc_embeddings = {}
    docs_to_process_map = {}

    print(f"Verificando cache em '{cache_dir}' para {len(df)}
documentos...")
    for _, row in tqdm(df.iterrows(), total=len(df), desc="Verificando
cache"):
        doc_id = row[doc_id_col]
        cache_file_path = os.path.join(cache_dir, f"{str(doc_id)}.pkl")
        if os.path.exists(cache_file_path):
            try:
                with open(cache_file_path, "rb") as f:
                    doc_embeddings[doc_id] = pickle.load(f)
            except Exception:
                docs_to_process_map[doc_id] = {'text': row[text_col],
'num_tokens': row[token_col]}
        else:
            docs_to_process_map[doc_id] = {'text': row[text_col],
'num_tokens': row[token_col]}

    if not docs_to_process_map:
```

```
        print("Todos os embeddings foram carregados do cache.")
        return doc_embeddings

    print(f"Gerando embeddings via API para {len(docs_to_process_map)}
novos documentos...")
    doc_ids = list(docs_to_process_map.keys())
    payloads = [docs_to_process_map[doc_id] for doc_id in doc_ids]

    def embedding_wrapper(payload: dict):
        return embedding_func(
            text=payload['text'],
            num_tokens=payload['num_tokens'],
            **embedding_params
        )

    with concurrent.futures.ThreadPoolExecutor(max_workers=max_workers) as
executor:
        results = list(tqdm(
            executor.map(embedding_wrapper, payloads),
            total=len(payloads),
            desc="Gerando Embeddings (API)"
        ))

    for doc_id, emb in zip(doc_ids, results):
        if emb is not None and emb.any():
            doc_embeddings[doc_id] = emb
            with open(os.path.join(cache_dir, f"{str(doc_id)}.pkl"), "wb")
as f:
                pickle.dump(emb, f)

    print("Embeddings de documentos gerados e/ou carregados.")
    return doc_embeddings
```

2.2: FUNÇÕES DE CÁLCULO DE MÉTRICA DE DRIFT

Esta seção contém as duas métricas usadas para quantificar vizualizar o drift entre distribuições de embeddings de janelas de tempo consecutivas: Maximum Mean Discrepancy (MMD) e Jensen-Shannon Divergence (JSD) via Clustering.

```
def calculate_optimal_gamma(embeddings: np.ndarray, sample_size: int = 500)
-> float:
    """
    Calcula o parâmetro gamma ideal para o kernel RBF usando a heurística
```

```
da mediana.

    A heurística da mediana é uma forma robusta de definir o parâmetro de
largura de banda (gamma)
    do kernel RBF. O gamma é crucial para que o MMD seja sensível a
diferenças entre as
    distribuições sem ser excessivamente ruidoso.

    Args:
        embeddings: Matriz de embeddings para calcular o gamma.
        sample_size: Número de amostras a serem usadas para a estimativa
(para eficiência).

    Returns:
        O valor ideal de gamma calculado.
    """
    if embeddings.shape[0] < 2:
        return 1.0 / embeddings.shape[1] if embeddings.shape[1] > 0 else
1.0
    sample = embeddings
    if embeddings.shape[0] > sample_size:
        indices = np.random.choice(embeddings.shape[0], sample_size,
replace=False)
        sample = embeddings[indices]
    dists_sq = pairwise_distances(sample, metric='euclidean')**2
    positive_dists_sq = dists_sq[dists_sq > 0]
    if positive_dists_sq.size == 0:
        return 1.0 / embeddings.shape[1] if embeddings.shape[1] > 0 else
1.0
    median_dist_sq = np.median(positive_dists_sq)
    if median_dist_sq == 0: return 1.0
    return 1.0 / (2 * median_dist_sq)

def mmd_rbf(X: np.ndarray, Y: np.ndarray, gamma: float = 1.0) -> float:
    """
    Calcula o MMD (Maximum Mean Discrepancy) entre duas distribuições X e Y
usando o kernel RBF.

    O que o MMD detecta (Drift Semântico):
    Por operar diretamente na geometria do espaço de embeddings, o MMD é
altamente sensível a
    drifts semânticos. Ele detecta mudanças sutis na forma como os
conceitos são expressos,
    o surgimento de novas nuances de significado ou alterações nas relações
```

```
entre palavras e ideias,
    mesmo que os tópicos gerais permaneçam os mesmos.

    Args:
        X: Matriz de embeddings da primeira distribuição.
        Y: Matriz de embeddings da segunda distribuição.
        gamma: Parâmetro de largura de banda do kernel RBF.

    Returns:
        O score de MMD. Um valor > 0 indica que as distribuições são
diferentes.
    """
    if X.ndim == 1: X = X.reshape(-1, 1)
    if Y.ndim == 1: Y = Y.reshape(-1, 1)
    XX = rbf_kernel(X, X, gamma)
    YY = rbf_kernel(Y, Y, gamma)
    XY = rbf_kernel(X, Y, gamma)
    return XX.mean() + YY.mean() - 2 * XY.mean()

def compute_jsd_via_clustering(dist1: np.ndarray, dist2: np.ndarray,
n_clusters: int = 10) -> float:
    """
    Calcula a JSD (Jensen-Shannon Divergence) entre duas distribuições via
clustering K-Means.

    O que a JSD via Clustering detecta (Drift Temático/Tópico):
    Esta métrica serve para detectar drifts temáticos. Ela agrupa os
embeddings em "tópicos"
    latentes usando K-Means e depois compara a proporção de documentos em
cada tópico entre as
    duas janelas de tempo. Ela responde a perguntas como: "A proporção de
documentos sobre o
    'Tópico A' aumentou?" ou "Um 'Tópico C' completamente novo surgiu?".

    Args:
        dist1: Matriz de embeddings da primeira distribuição.
        dist2: Matriz de embeddings da segunda distribuição.
        n_clusters: O número de "tópicos" a serem encontrados pelo K-Means.

    Returns:
        O score de JSD ao quadrado.
    """
    if dist1.shape[0] < n_clusters or dist2.shape[0] < n_clusters:
        n_clusters = min(dist1.shape[0], dist2.shape[0])
```

```
    if n_clusters < 2:
        return 0.0
    combined_data = np.vstack([dist1, dist2])
    kmeans = KMeans(n_clusters=n_clusters, random_state=42,
n_init='auto').fit(combined_data)
    labels1 = kmeans.predict(dist1)
    labels2 = kmeans.predict(dist2)
    hist1, _ = np.histogram(labels1, bins=np.arange(n_clusters + 1))
    hist2, _ = np.histogram(labels2, bins=np.arange(n_clusters + 1))
    prob_dist1 = (hist1 + 1e-9) / np.sum(hist1 + 1e-9)
    prob_dist2 = (hist2 + 1e-9) / np.sum(hist2 + 1e-9)
    jsd_sqrt = jensenshannon(prob_dist1, prob_dist2)
    return jsd_sqrt ** 2
```

PARTE 2.3: FUNÇÕES DE ANÁLISE E DETECÇÃO

Esta seção contém as funções que orquestram o processo de detecção de drift:

1. Agrupar os dados em janelas de tempo.
2. Calcular os scores de drift entre janelas consecutivas.
3. Detectar pontos de drift com base em limiares estatísticos.
4. Calcular métricas auxiliares para ajudar na interpretação.

```
def compute_window_distributions(df: pd.DataFrame, doc_id_col: str,
date_col: str, all_doc_embeddings: dict, window_func: callable) -> dict:
    """
    Agrupa documentos por janela de tempo e retorna um dicionário com a
matriz
    de embeddings para cada janela. É o primeiro passo para a análise de
drift.

    Args:
        df: DataFrame com os metadados dos documentos.
        doc_id_col: Nome da coluna com o ID do documento.
        date_col: Nome da coluna com a data do documento.
        all_doc_embeddings: Dicionário mapeando doc_id para seu embedding.
        window_func: Função que transforma uma data em uma string de
janela.

    Returns:
        Dicionário onde as chaves são as janelas de tempo e os valores são
as matrizes de embeddings.
    """
    df_copy = df.copy()
```

```
    df_copy["window"] = df_copy[date_col].apply(window_func)
    window_distributions = {}
    for window, group in tqdm(df_copy.groupby("window"), desc="Coletando
distribuições por janela"):
        doc_ids = group[doc_id_col].tolist()
        embs = [all_doc_embeddings.get(doc_id) for doc_id in doc_ids if
all_doc_embeddings.get(doc_id) is not None]
        if embs:
            window_distributions[window] = np.vstack(embs)
    return window_distributions

def calculate_drift_scores(window_dists: dict, metric: str = 'mmd') ->
pd.DataFrame:
    """
    Executa a comparação entre janelas de tempo consecutivas usando a
métrica escolhida
    e retorna um DataFrame com os scores de drift ao longo do tempo.

    Args:
        window_dists: Dicionário de distribuições por janela (output de
compute_window_distributions).
        metric: A métrica a ser usada ('mmd' ou 'jsd').

    Returns:
        Um DataFrame com as colunas 'window' e 'score'.
    """
    ordered_windows = sorted(window_dists.keys())
    scores = []
    prev_dist = None
    gamma = 1.0
    if metric == 'mmd':
        first_stable_window = next((window_dists[k] for k in
ordered_windows if window_dists.get(k) is not None and
window_dists[k].shape[0] > 1), None)
        if first_stable_window is not None:
            gamma = calculate_optimal_gamma(first_stable_window)
        print(f"Usando Gamma Otimal (calculado pela heurística da mediana):
{gamma:.6f}")
    for w in tqdm(ordered_windows, desc=f"Calculando scores
{metric.upper()}"):
        curr_dist = window_dists.get(w)
        if prev_dist is not None and curr_dist is not None and
prev_dist.shape[0] > 1 and curr_dist.shape[0] > 1:
            score = 0.0
```

```
            if metric == 'mmd':
                score = mmd_rbf(prev_dist, curr_dist, gamma=gamma)
            elif metric == 'jsd':
                score = compute_jsd_via_clustering(prev_dist, curr_dist)
            scores.append({'window': w, 'score': score})
        prev_dist = curr_dist
    return pd.DataFrame(scores)

def detect_drift_with_thresholds(df_scores: pd.DataFrame, critical_std: int
= 3, attention_std: int = 2) -> dict:
    """
    Aplica múltiplos limiares (baseados em desvios padrão da média
histórica) para detectar drifts.
    Isso transforma a série temporal de scores de drift em alertas
concretos.

    Args:
        df_scores: DataFrame com os scores de drift.
        critical_std: Fator de desvio padrão para o limiar crítico.
        attention_std: Fator de desvio padrão para o limiar de atenção.

    Returns:
        Dicionário contendo listas de pontos críticos e de atenção, e os
valores dos limiares.
    """
    if df_scores.empty:
        return {"critical": [], "attention": [], "thresholds": {}}
    mean_score = df_scores['score'].mean()
    std_score = df_scores['score'].std()
    threshold_critical = mean_score + (critical_std * std_score)
    threshold_attention = mean_score + (attention_std * std_score)
    critical_points = df_scores[df_scores['score'] > threshold_critical]
    attention_points = df_scores[(df_scores['score'] > threshold_attention)
& (df_scores['score'] <= threshold_critical)]
    print(f"Limiar de Atenção ({attention_std}*STD):
{threshold_attention:.4f}")
    print(f"Limiar Crítico ({critical_std}*STD):
{threshold_critical:.4f}")
    return {
        "critical": list(zip(critical_points['window'],
critical_points['score'])),
        "attention": list(zip(attention_points['window'],
attention_points['score'])),
        "thresholds": {"critical": threshold_critical, "attention":
```

```
threshold_attention}
    }

def calculate_auxiliary_metrics_by_window(df: pd.DataFrame, date_col: str,
token_col: str, window_func: callable) -> pd.DataFrame:
    """
    Calcula métricas auxiliares (contagem de documentos e média de tokens)
por janela de tempo.
    Essas métricas ajudam a contextualizar os scores de drift. Por exemplo,
um pico no score
    pode estar correlacionado com um aumento repentino no volume de
documentos ou no seu tamanho.

    Args:
        df: DataFrame com os metadados dos documentos.
        date_col: Nome da coluna de data.
        token_col: Nome da coluna com a contagem de tokens.
        window_func: Função para agrupar as datas em janelas.

    Returns:
        DataFrame com métricas agregadas por janela.
    """
    df_copy = df.copy()
    df_copy["window"] = df_copy[date_col].apply(window_func)

    # Agrupa pela janela e calcula as agregações
    aux_metrics = df_copy.groupby("window").agg(
        doc_count=(date_col, 'size'),          # Conta o número de
documentos
        avg_tokens=(token_col, 'mean')         # Calcula a média de tokens
    ).reset_index()

    print("Métricas auxiliares por janela calculadas (contagem de docs,
média de tokens).")
    return aux_metrics
```

PARTE 2.4: FUNÇÕES DE VISUALIZAÇÃO

```
def plot_drift_scores(df_scores: pd.DataFrame, drift_results: dict, title:
str, color: str = 'blue', y_label: str = 'Score'):
    """
    Função de plotagem genérica para visualizar scores de drift e limiares.

    Esta função cria um gráfico de linha mostrando a evolução do score de
```

```
    drift ao longo do tempo.
        Ela também desenha os limiares de 'Atenção' e 'Crítico' e destaca as
janelas onde
        um drift crítico foi detectado, facilitando a identificação visual dos
pontos de mudança.
        """
        if df_scores.empty:
            print(f"Não há dados para plotar para '{title}'.")
            return
        fig, ax = plt.subplots(figsize=(20, 7))
        ax.plot(df_scores['window'].astype(str), df_scores['score'],
marker='o', markersize=4, linestyle='-', label=y_label, color=color,
zorder=5)
        thresholds = drift_results.get("thresholds", {})
        if "attention" in thresholds:
            ax.axhline(y=thresholds["attention"], color='orange',
linestyle=':', label=f'Limiar de Atenção ({thresholds["attention"]:.4f})',
zorder=10)
        if "critical" in thresholds:
            ax.axhline(y=thresholds["critical"], color='darkred',
linestyle='--', label=f'Limiar Crítico ({thresholds["critical"]:.4f})',
zorder=10)
        critical_drifts = [d[0] for d in drift_results.get("critical", [])]
        for w in critical_drifts:
            ax.axvline(x=str(w), color='red', linestyle='--', alpha=0.6,
zorder=1)
        tick_labels = df_scores['window'].astype(str).unique()
        n = max(1, len(tick_labels) // 25)
        ax.set_xticks(tick_labels[::n])
        ax.tick_params(axis='x', rotation=90)
        ax.set_title(title, fontsize=16)
        ax.set_ylabel(y_label, fontsize=12)
        ax.set_xlabel('Janela de Tempo', fontsize=12)
        ax.legend()
        plt.tight_layout()
        plt.show()

    def display_drift_results(drift_results: dict):
        """
        Imprime os resultados da detecção de drift de forma legível em texto.

        Esta função fornece um resumo textual dos pontos de drift detectados,
listando
        separadamente os que atingiram o nível de 'Atenção' e 'Crítico'.
```

```
    """
    print("\n--- [NÍVEL CRÍTICO] ---")
    if not drift_results['critical']:
        print("Nenhum drift crítico detectado.")
    else:
        for window, score in drift_results['critical']:
            print(f"  - Janela: {window}, Score: {score:.6f}")
    print("\n--- [NÍVEL DE ATENÇÃO] ---")
    if not drift_results['attention']:
        print("Nenhum drift de atenção detectado.")
    else:
        for window, score in drift_results['attention']:
            print(f"  - Janela: {window}, Score: {score:.6f}")

def plot_drift_with_correlation(df_scores: pd.DataFrame,
                                drift_results: dict,
                                df_aux_metrics: pd.DataFrame,
                                title: str,
                                color: str = 'blue',
                                y_label: str = 'Score'):
    """
    Plota os scores de drift e os correlaciona com métricas auxiliares (ex:
volume de dados).

    Este gráfico avançado possui um eixo Y duplo. O eixo esquerdo mostra os
scores de drift,
    enquanto o direito exibe métricas auxiliares como a contagem de
documentos e a média de tokens.
    Isso é extremamente útil para investigar a causa raiz de um drift: um
pico no score pode
    estar correlacionado a um aumento súbito no volume de dados ou a uma
mudança no tamanho
    médio dos textos, por exemplo.
    """
    if df_scores.empty:
        print(f"Não há dados de score para plotar para '{title}'.")
        return

    df_plot_data = pd.merge(df_scores, df_aux_metrics, on='window',
how='left')

    fig, ax1 = plt.subplots(figsize=(22, 9)) # Aumentei um pouco a figura

    # --- Eixo Y Esquerdo (Scores de Drift) ---
```

```
    ax1.plot(df_plot_data['window'].astype(str), df_plot_data['score'],
marker='o', markersize=5, linestyle='-', label=y_label, color=color,
zorder=10)
    ax1.set_xlabel('Janela de Tempo', fontsize=14)
    ax1.set_ylabel(y_label, fontsize=14, color=color)
    ax1.tick_params(axis='y', labelcolor=color, labelsize=12)
    ax1.set_title(title, fontsize=20, pad=20)


    thresholds = drift_results.get("thresholds", {})

    if "attention" in thresholds:
        ax1.axhline(y=thresholds["attention"], color='orange',
linestyle=':',
                    label=f'Limiar de Atenção
({thresholds["attention"]:.4f})', zorder=5)
    if "critical" in thresholds:
        ax1.axhline(y=thresholds["critical"], color='darkred',
linestyle='--',
                    label=f'Limiar Crítico ({thresholds["critical"]:.4f})',
zorder=5)

    # Usamos um plot "fantasma" para criar o item na legenda sem desenhar
nada extra.
    ax1.plot([],[], color='red', linestyle='--', alpha=0.6, label='Ponto de
Drift Crítico')

    # Desenha as linhas verticais nos pontos onde o drift crítico foi
detectado
    critical_drifts = [d[0] for d in drift_results.get("critical", [])]
    for w in critical_drifts:
        ax1.axvline(x=str(w), color='red', linestyle='--', alpha=0.6,
zorder=1)

    ax2 = ax1.twinx()
    ax2.bar(df_plot_data['window'].astype(str), df_plot_data['doc_count'],
            width=0.4, color='gray', alpha=0.3, label='Qtd. Documentos')
    ax2.set_ylabel('Contagem de Documentos / Média de Tokens', fontsize=14,
color='gray')
    ax2.tick_params(axis='y', labelcolor='gray', labelsize=12)
    ax2.plot(df_plot_data['window'].astype(str),
df_plot_data['avg_tokens'],
             marker='x', markersize=5, linestyle=':', color='c',
             alpha=0.9, label='Média de Tokens')
```

```
    tick_labels = df_plot_data['window'].astype(str).unique()
    n = max(1, len(tick_labels) // 25)
    ax1.set_xticks(tick_labels[::n])
    ax1.tick_params(axis='x', rotation=90, labelsize=12)
    ax1.grid(True, which='major', linestyle='--', linewidth=0.5)

    lines, labels = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()

    fig.legend(lines + lines2, labels + labels2,
               loc='upper center',
               bbox_to_anchor=(0.5, 0.03),
               ncol=6,
               fontsize=18)

    plt.tight_layout()
    plt.subplots_adjust(bottom=0.2)
    plt.show()
```

2.5: FUNÇÕES DE PREPARAÇÃO DE DADOS ESPECIALIZADAS

```
def load_and_prep_uklex() -> pd.DataFrame:
    """
    Carrega o dataset UKLEX, prepara as colunas e calcula o número de
    tokens para cada documento. Link para carregar o documento :
https://zenodo.org/records/6355465

    Esta função lê o arquivo JSONL local, ordena os documentos por ano,
combina
    título e corpo para formar o texto principal, e cria as colunas
padronizadas
    para a análise de drift.
    """
    print("Preparando dataset UKLEX (versão completa)...")
    file_path = "data/uk-lex18.jsonl"
    if not os.path.exists(file_path):
        raise FileNotFoundError(f"Arquivo de dados não encontrado em:
{file_path}")
    df_full = pd.read_json(file_path,
lines=True).sort_values(by="year").reset_index(drop=True)

    df_full['text'] = df_full['title'] + "\n" + df_full['body']
    df_full['date'] = pd.to_datetime(df_full['year'].apply(lambda y:
```

```
date(y, 1, 1)))
    df_full = df_full.rename(columns={'id': 'doc_id'})

    tqdm.pandas(desc="Calculando tokens UKLEX")
    print("Calculando contagem de tokens para os documentos UKLEX...")
    df_full['num_tokens'] = df_full['text'].progress_apply(lambda x:
len(TOKENIZER.encode(x)))

    print(f"UKLEX carregado. Total de {len(df_full)} documentos.")
    return df_full

def load_and_prep_eurlex() -> pd.DataFrame:
    """
    Carrega o dataset EURLEX a partir do Hugging Face, prepara as colunas
    e calcula o número de tokens para cada documento.

    Esta função baixa os dados do Hub, processa todos os splits (train,
validation, test),
    extrai o ano do ID do documento, e cria as colunas padronizadas.
    """
    print("Preparando dataset EURLEX...")
    dataset = load_dataset("coastalcph/multi_eurlex", "en")
    dfs = []
    for split in ["train", "validation", "test"]:
        df = pd.DataFrame(dataset[split])
        df["text"] = df["text"].apply(lambda x: " ".join(x) if
isinstance(x, list) else x)
        df = df.rename(columns={"celex_id": "doc_id"})
        df["year"] = pd.to_numeric(df["doc_id"].str[1:5], errors='coerce')
        df.dropna(subset=['year'], inplace=True)
        df['year'] = df['year'].astype(int)
        df["date"] = pd.to_datetime(df["year"].apply(lambda y: date(y, 1,
1)))
        dfs.append(df)

    df_full = pd.concat(dfs,
ignore_index=True).sort_values(by="date").reset_index(drop=True)

    tqdm.pandas(desc="Calculando tokens EURLEX")
    print("Calculando contagem de tokens para os documentos EURLEX...")
    df_full['num_tokens'] = df_full['text'].progress_apply(lambda x:
len(TOKENIZER.encode(x)))
```

```
    print(f"EURLEX carregado. Total de {len(df_full)} documentos.")
    return df_full
```

2.7: CONFIGURAÇÃO DOS EXPERIMENTOS

```
EMBEDDING_STRATEGIES = {
    "head_tail": {
        "function": get_openai_embedding_headtail,
        "params": {"head_size": 0.5},
        "cache_suffix": "_headtail"
    },
    "mean_of_chunks": {
        "function": get_openai_embedding_with_retry,
        "params": {},
        "cache_suffix": "_chunks_overlap"
    }
}

DATASET_CONFIGS = {
    "UKLEX": {
        "loader_func": load_and_prep_uklex,
        "doc_id_col": "doc_id",
        "text_col": "text",
        "token_col": "num_tokens",
        "date_col": "date",
        "window_func": lambda ts: ts.year,
        "cache_base_dir": "uklex"
    },
    "EURLEX": {
        "loader_func": load_and_prep_eurlex,
        "doc_id_col": "doc_id",
        "text_col": "text",
        "token_col": "num_tokens",
        "date_col": "date",
        "window_func": lambda ts: ts.year,
        "cache_base_dir": "eurlex"
    }
}

all_results = {}
```

3.1: Análise para o Dataset UKLEX

```
# --- Configurações Iniciais ---
dataset_name = "UKLEX"
print(f"\n{'#'*80}\n### INICIANDO PREPARAÇÃO PARA O DATASET: {dataset_name}
###\n{'#'*80}")

# Seleciona a configuração do dataset a ser usado
dataset_config = DATASET_CONFIGS[dataset_name]

# Dicionário mestre para armazenar os embeddings de todas as estratégias
all_doc_embeddings = {}

# --- Carregamento e Pré-processamento do DataFrame ---
# A função 'loader_func' carrega, limpa e calcula a contagem de tokens
df_uklex = dataset_config["loader_func"]()


################################################################################
#####
### INICIANDO PREPARAÇÃO PARA O DATASET: UKLEX ###
################################################################################
#####
Preparando dataset UKLEX (versão completa)...
Calculando contagem de tokens para os documentos UKLEX...

Calculando tokens UKLEX: 100%|██████████| 36500/36500 [00:41<00:00,
889.43it/s]

UKLEX carregado. Total de 36500 documentos.



# --- Cálculo de Métricas Auxiliares ---
# Calcula a contagem de documentos e a média de tokens por ano.
# Esses dados serão usados nos gráficos para análise de correlação visual.
df_uklex_aux_metrics = calculate_auxiliary_metrics_by_window(
    df=df_uklex,
    date_col=dataset_config["date_col"],
    token_col=dataset_config["token_col"],
    window_func=dataset_config["window_func"]
)

print("\n>>> Ambiente para o UKLEX está pronto para os experimentos. <<<")
```

Métricas auxiliares por janela calculadas (contagem de docs, média de tokens).

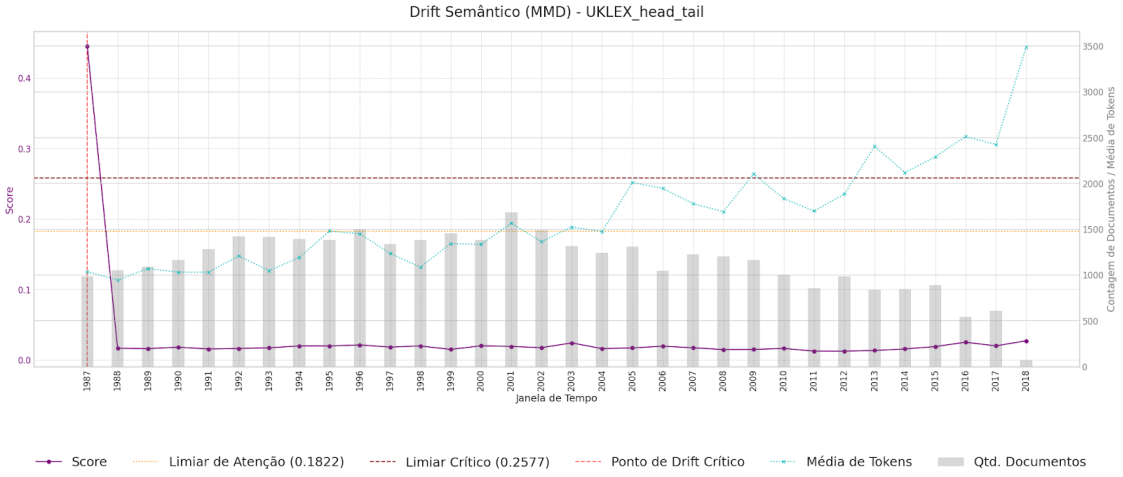
>>> Ambiente para o UKLEX está pronto para os experimentos. <<<

```
# Define a estratégia a ser utilizada nesta célula
strategy_name = "head_tail"
strategy_config = EMBEDDING_STRATEGIES[strategy_name]
print(f"\n{'='*80}\n>>> GERANDO/CARREGANDO EMBEDDINGS PARA: {strategy_name}
<<<\n{'='*80}")

# Configura os parâmetros para a chamada da função de embedding, incluindo
o diretório de cache
embedding_call_config = {
    'cache_dir': os.path.join(CACHE_DIR_ROOT,
f"{dataset_config['cache_base_dir']}{strategy_config['cache_suffix']}"),
    'function': strategy_config['function'],
    'params': strategy_config.get('params', {})
}

# Chama a função que gera os embeddings (e usa o cache de forma
inteligente)
# O resultado é armazenado no dicionário mestre para ser reutilizado depois
all_doc_embeddings[strategy_name] = generate_document_embeddings(
    df=df_uklex,
    text_col=dataset_config["text_col"],
    doc_id_col=dataset_config["doc_id_col"],
    token_col=dataset_config["token_col"],
    embedding_config=embedding_call_config
)
```

```
================================================================================
>>> GERANDO/CARREGANDO EMBEDDINGS PARA: head_tail <<<
================================================================================
Verificando cache em 'cache_embeddings/uklex_headtail' para 36500
documentos...

Verificando cache: 100%|██████████| 36500/36500 [00:04<00:00, 7665.68it/s]

Todos os embeddings foram carregados do cache.
```

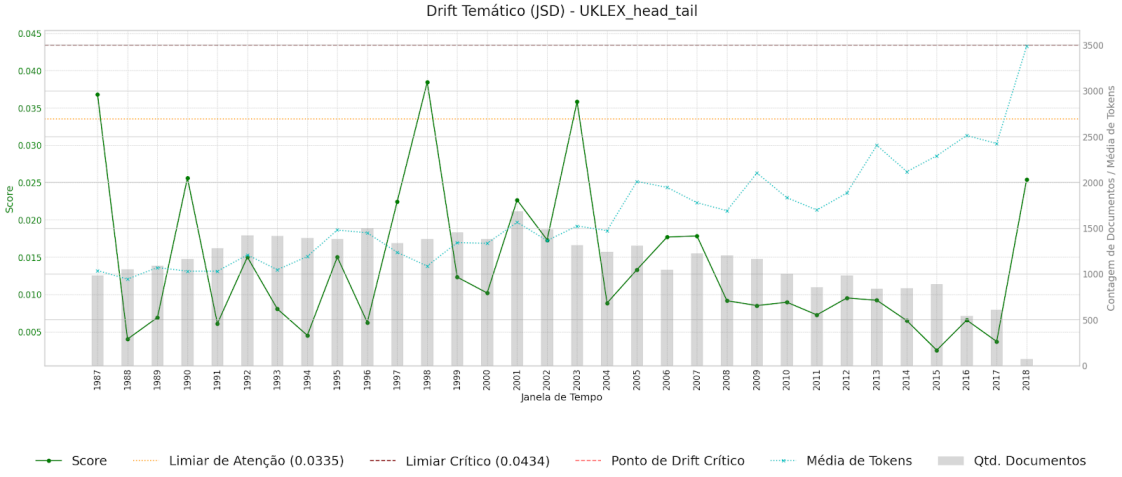
```
# Define qual estratégia será analisada nesta célula
strategy_name = "head_tail"
experiment_name = f"{dataset_name}_{strategy_name}"
print(f"\n{'='*80}\n>>> PROCESSANDO ANÁLISE PARA: {experiment_name}
<<<\n{'='*80}")

# Recupera os embeddings correspondentes do dicionário mestre (sem
regerá-los)
doc_embeddings = all_doc_embeddings.get(strategy_name)

# Verifica se os embeddings foram carregados antes de prosseguir
if not doc_embeddings:
    print(f"ERRO: Embeddings para a estratégia '{strategy_name}' não foram
encontrados. \n"
          f"Por favor, execute a célula de geração de embeddings
correspondente (na seção 3.2) primeiro.")
else:
    # Agrupa os embeddings em janelas temporais (anos)
    window_dists = compute_window_distributions(
        df=df_uklex,
        doc_id_col=dataset_config["doc_id_col"],
        date_col=dataset_config["date_col"],
        all_doc_embeddings=doc_embeddings,
        window_func=dataset_config["window_func"]
    )

    # --- Análise de Drift Semântico (MMD) ---
    print("\n--- Análise de Drift MMD ---")
    df_mmd_scores = calculate_drift_scores(window_dists, metric='mmd')
    mmd_drift_results = detect_drift_with_thresholds(df_mmd_scores)
    display_drift_results(mmd_drift_results)
    plot_drift_with_correlation(df_mmd_scores, mmd_drift_results,
df_uklex_aux_metrics,
                                title=f'Drift Semântico (MMD) -
{experiment_name}', color='purple')

    # --- Análise de Drift Temático (JSD) ---
    print("\n--- Análise de Drift JSD ---")
    df_jsd_scores = calculate_drift_scores(window_dists, metric='jsd')
    jsd_drift_results = detect_drift_with_thresholds(df_jsd_scores)
    display_drift_results(jsd_drift_results)
    plot_drift_with_correlation(df_jsd_scores, jsd_drift_results,
```

```
df_uklex_aux_metrics,
                                    title=f'Drift Temático (JSD) -
{experiment_name}', color='green')


================================================================================
=====
>>> PROCESSANDO ANÁLISE PARA: UKLEX_head_tail <<<
================================================================================
=====

Coletando distribuições por janela: 100%|██████████| 38/38 [00:00<00:00,
179.30it/s]


--- Análise de Drift MMD ---
Usando Gamma Otimal (calculado pela heurística da mediana): 5.312084

Calculando scores MMD: 100%|██████████| 38/38 [00:13<00:00,  2.78it/s]

Limiar de Atenção (2*STD): 0.1822
Limiar Crítico (3*STD):  0.2577

--- [NÍVEL CRÍTICO] ---
  - Janela: 1987, Score: 0.444666

--- [NÍVEL DE ATENÇÃO] ---
Nenhum drift de atenção detectado.
```




```
--- Análise de Drift JSD ---
```

```
Calculando scores JSD: 100%|██████████| 38/38 [00:13<00:00,  2.91it/s]

Limiar de Atenção (2*STD): 0.0335
Limiar Crítico (3*STD):  0.0434

--- [NÍVEL CRÍTICO] ---
Nenhum drift crítico detectado.

--- [NÍVEL DE ATENÇÃO] ---
  - Janela: 1987, Score: 0.036817
  - Janela: 1998, Score: 0.038430
  - Janela: 2003, Score: 0.035821
```



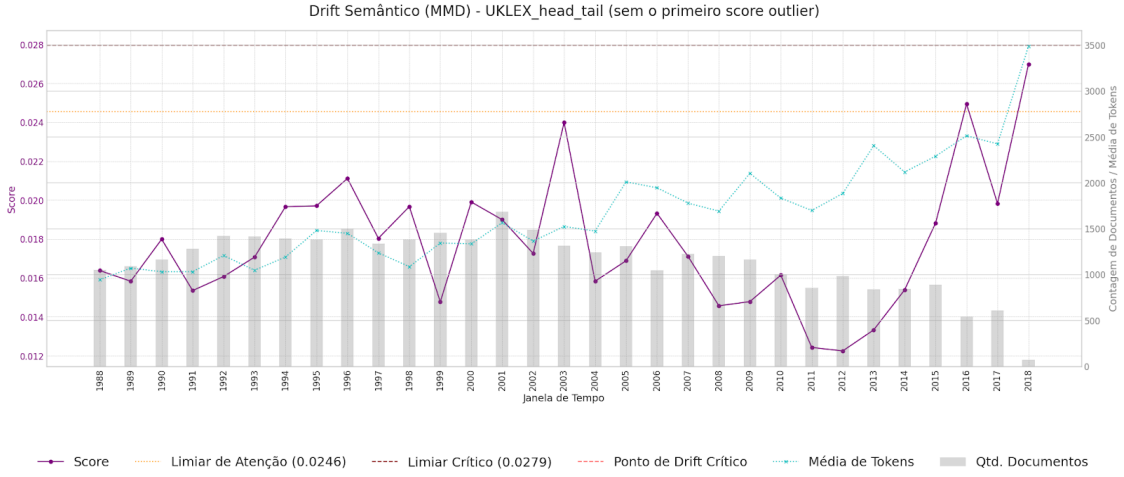
Podemos fazer algumas análises, dentro do drift apresentado para o semantico temos uma nornmalização de como se está a estrutura nos documentos visto a partir dos anos 1988. Podemos perceber outras tendências nos dados.

```
# Define a estratégia que será analisada
strategy_name = "head_tail" # ou "mean_of_chunks"
experiment_name = f"{dataset_name}_{strategy_name}"
print(f"\n{'#'*80}\n>>> ANÁLISE MMD PARA '{experiment_name}' IGNORANDO O
PRIMEIRO SCORE (OUTLIER) <<<\n{'#'*80}")

#
================================================================================
===
# 1. CÁLCULO COMPLETO DOS SCORES
# Primeiro, calculamos os scores de drift normalmente, sobre todo o
período.
# Isso nos dará um DataFrame que inclui o outlier inicial.
```

```
#
================================================================================
===

# Recupera os embeddings necessários
doc_embeddings = all_doc_embeddings.get(strategy_name)

if not doc_embeddings:
    print(f"ERRO: Embeddings para a estratégia '{strategy_name}' não foram
encontrados.")
else:
    # Calcula as distribuições por janela
    window_dists = compute_window_distributions(
        df=df_uklex,
        doc_id_col=dataset_config["doc_id_col"],
        date_col=dataset_config["date_col"],
        all_doc_embeddings=doc_embeddings,
        window_func=dataset_config["window_func"]
    )

    # Calcula os scores de MMD para todo o período
    df_mmd_scores = calculate_drift_scores(window_dists, metric='mmd')

    print("\nDataFrame de scores original (com outlier):")
    print(df_mmd_scores.head())

    #
================================================================================
===
    # 2. FILTRAGEM CORRETA E ROBUSTA
    # AQUI ESTÁ A MUDANÇA CRUCIAL:
    # Em vez de filtrar por ano, nós simplesmente removemos a PRIMEIRA
LINHA do
    # DataFrame de scores. Isso elimina o outlier inicial, não importa qual
ano seja.
    # .iloc[1:] seleciona todas as linhas a partir da segunda (índice 1).
    #
================================================================================
===
    print("\n[AÇÃO] Removendo o primeiro score (outlier) do DataFrame...")
    df_mmd_scores_filtered = df_mmd_scores.iloc[1:].copy()
    print("\nDataFrame de scores filtrado (sem outlier):")
    print(df_mmd_scores_filtered.head())
```

```
    #
================================================================================
===
    # 3. ANÁLISE E PLOTAGEM COM DADOS CORRIGIDOS
    # Agora, todas as análises seguintes usarão os dados limpos.
    #
================================================================================
===
    print("\n--- Análise de Drift MMD (Resultados Finais Corretos) ---")
    mmd_drift_results_final =
detect_drift_with_thresholds(df_mmd_scores_filtered)
    display_drift_results(mmd_drift_results_final)

    plot_drift_with_correlation(
        df_mmd_scores_filtered,
        mmd_drift_results_final,
        df_uklex_aux_metrics,
        title=f'Drift Semântico (MMD) - {experiment_name} (sem o primeiro
score outlier)',
        color='purple'
    )


##############################################################################
#####
>>> ANÁLISE MMD PARA 'UKLEX_head_tail' IGNORANDO O PRIMEIRO SCORE (OUTLIER)
<<<
##############################################################################
#####

Coletando distribuições por janela: 100%|██████████| 38/38 [00:00<00:00,
177.61it/s]

Usando Gamma Otimal (calculado pela heurística da mediana): 5.312084

Calculando scores MMD: 100%|██████████| 38/38 [00:13<00:00,  2.77it/s]


DataFrame de scores original (com outlier):
   window     score
0    1987  0.444666
1    1988  0.016376
2    1989  0.015829
3    1990  0.017988
```

```
4    1991  0.015347

[AÇÃO] Removendo o primeiro score (outlier) do DataFrame...

DataFrame de scores filtrado (sem outlier):
   window     score
1    1988  0.016376
2    1989  0.015829
3    1990  0.017988
4    1991  0.015347
5    1992  0.016068

--- Análise de Drift MMD (Resultados Finais Corretos) ---
Limiar de Atenção (2*STD): 0.0246
Limiar Crítico (3*STD):  0.0279

--- [NÍVEL CRÍTICO] ---
Nenhum drift crítico detectado.

--- [NÍVEL DE ATENÇÃO] ---
  - Janela: 2016, Score: 0.024941
  - Janela: 2018, Score: 0.026975
```



Podemos ao retirar o outlier ver um nível de conversão nos dados. Apresentados, onde a partir de certa data, vemos uma evolução constante rumo ao um novo nivel

```
# Define a estratégia a ser utilizada nesta célula
strategy_name = "mean_of_chunks"
strategy_config = EMBEDDING_STRATEGIES[strategy_name]
```

```
print(f"\n{'='*80}\n>>> GERANDO/CARREGANDO EMBEDDINGS PARA: {strategy_name}
<<<\n{'='*80}")

# Configura os parâmetros para a chamada da função de embedding
embedding_call_config = {
    'cache_dir': os.path.join(CACHE_DIR_ROOT,
f"{dataset_config['cache_base_dir']}{strategy_config['cache_suffix']}"),
    'function': strategy_config['function'],
    'params': strategy_config.get('params', {})
}

# Chama a função que gera os embeddings e armazena no dicionário mestre
all_doc_embeddings[strategy_name] = generate_document_embeddings(
    df=df_uklex,
    text_col=dataset_config["text_col"],
    doc_id_col=dataset_config["doc_id_col"],
    token_col=dataset_config["token_col"],
    embedding_config=embedding_call_config
)
```

```
================================================================================
=====
>>> GERANDO/CARREGANDO EMBEDDINGS PARA: mean_of_chunks <<<
================================================================================
=====
Verificando cache em 'cache_embeddings/uklex_chunks_overlap' para 36500
documentos...

Verificando cache: 100%|██████████| 36500/36500 [00:04<00:00, 7770.93it/s]

Todos os embeddings foram carregados do cache.
```

```
# CÉLULA 3.2: Análise de Drift para a Estratégia 'mean_of_chunks'

# Define qual estratégia será analisada nesta célula
strategy_name = "mean_of_chunks"
experiment_name = f"{dataset_name}_{strategy_name}"
print(f"\n{'='*80}\n>>> PROCESSANDO ANÁLISE PARA: {experiment_name}
<<<\n{'='*80}")

# Recupera os embeddings do dicionário mestre
doc_embeddings = all_doc_embeddings.get(strategy_name)
```

```
# Verifica se os embeddings existem
if not doc_embeddings:
    print(f"ERRO: Embeddings para a estratégia '{strategy_name}' não foram
encontrados. \n"
          f"Por favor, execute a célula de geração de embeddings
correspondente (na seção 3.2) primeiro.")
else:
    # Agrupa os embeddings em janelas temporais
    window_dists = compute_window_distributions(
        df=df_uklex,
        doc_id_col=dataset_config["doc_id_col"],
        date_col=dataset_config["date_col"],
        all_doc_embeddings=doc_embeddings,
        window_func=dataset_config["window_func"]
    )

    # --- Análise de Drift Semântico (MMD) ---
    print("\n--- Análise de Drift MMD ---")
    df_mmd_scores = calculate_drift_scores(window_dists, metric='mmd')
    mmd_drift_results = detect_drift_with_thresholds(df_mmd_scores)
    display_drift_results(mmd_drift_results)
    plot_drift_with_correlation(df_mmd_scores, mmd_drift_results,
df_uklex_aux_metrics,
                                title=f'Drift Semântico (MMD) -
{experiment_name}', color='purple')

    # --- Análise de Drift Temático (JSD) ---
    print("\n--- Análise de Drift JSD ---")
    df_jsd_scores = calculate_drift_scores(window_dists, metric='jsd')
    jsd_drift_results = detect_drift_with_thresholds(df_jsd_scores)
    display_drift_results(jsd_drift_results)
    plot_drift_with_correlation(df_jsd_scores, jsd_drift_results,
df_uklex_aux_metrics,
                                title=f'Drift Temático (JSD) -
{experiment_name}', color='green')


================================================================================
=====
>>> PROCESSANDO ANÁLISE PARA: UKLEX_mean_of_chunks <<<
================================================================================
=====
```

Coletando distribuições por janela: 100%|██████████| 38/38 [00:00<00:00, 171.23it/s]

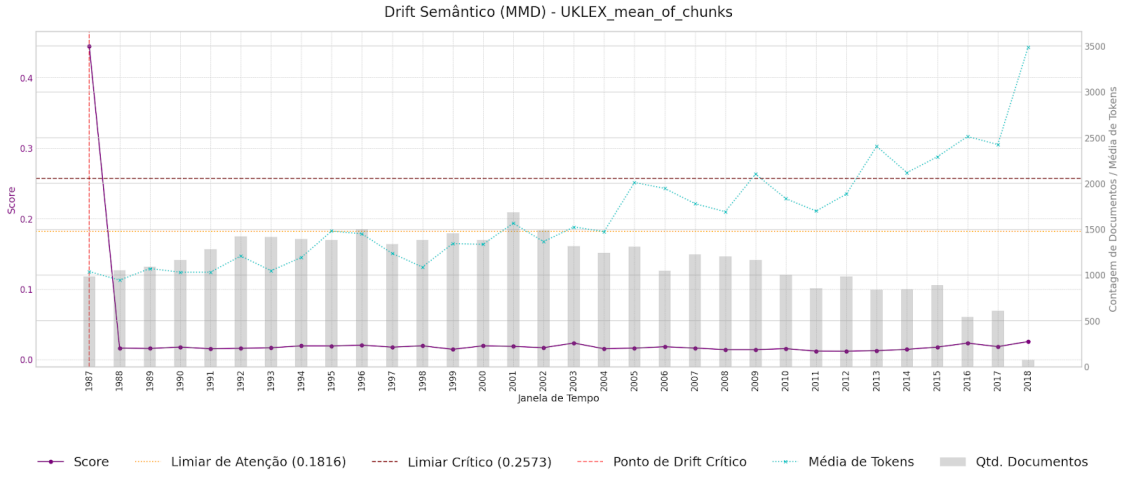
--- Análise de Drift MMD ---
Usando Gamma Otimal (calculado pela heurística da mediana): 5.315293

Calculando scores MMD: 100%|██████████| 38/38 [00:13<00:00, 2.79it/s]

Limiar de Atenção (2*STD): 0.1816
Limiar Crítico (3*STD): 0.2573

--- [NÍVEL CRÍTICO] ---
  - Janela: 1987, Score: 0.444413

--- [NÍVEL DE ATENÇÃO] ---
Nenhum drift de atenção detectado.



--- Análise de Drift JSD ---

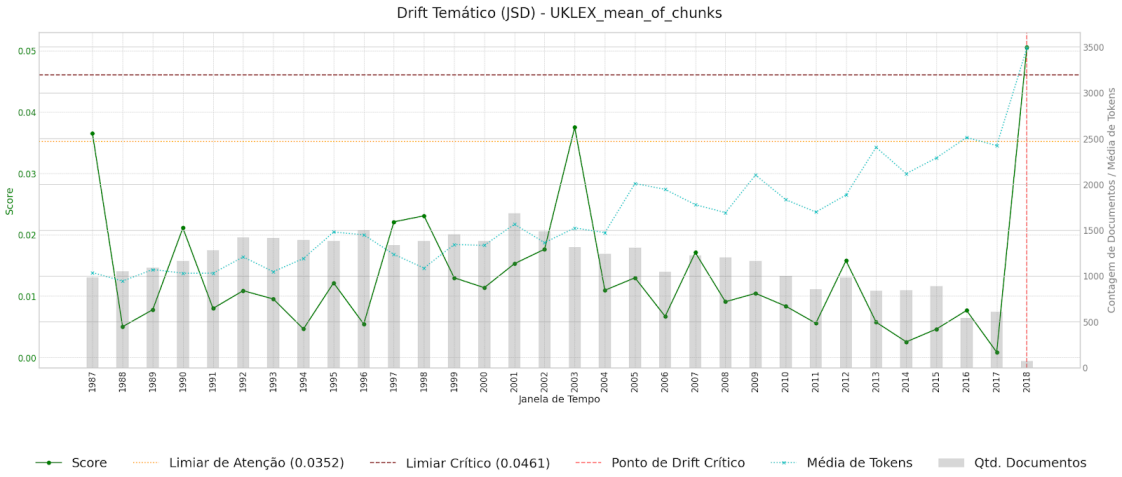
Calculando scores JSD: 100%|██████████| 38/38 [00:13<00:00, 2.78it/s]

Limiar de Atenção (2*STD): 0.0352
Limiar Crítico (3*STD): 0.0461

--- [NÍVEL CRÍTICO] ---
  - Janela: 2018, Score: 0.050527

--- [NÍVEL DE ATENÇÃO] ---

```
- Janela: 1987, Score: 0.036502
- Janela: 2003, Score: 0.037508
```



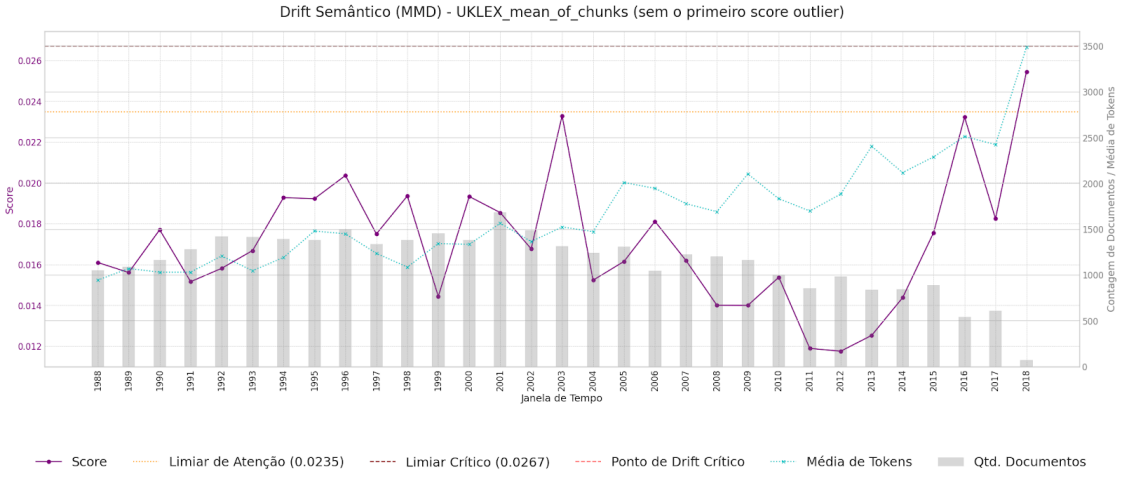
Retirando o outlier novamente

```
# Define a estratégia que será analisada
strategy_name = "mean_of_chunks"
experiment_name = f"{dataset_name}_{strategy_name}"
print(f"\n{'#'*80}\n>>> ANÁLISE MMD PARA '{experiment_name}' IGNORANDO O
PRIMEIRO SCORE (OUTLIER) <<<\n{'#'*80}")

#
================================================================================
===
# 1. CÁLCULO COMPLETO DOS SCORES
# Primeiro, calculamos os scores de drift normalmente, sobre todo o
período.
# Isso nos dará um DataFrame que inclui o outlier inicial.
#
================================================================================
===

# Recupera os embeddings necessários
doc_embeddings = all_doc_embeddings.get(strategy_name)

if not doc_embeddings:
    print(f"ERRO: Embeddings para a estratégia '{strategy_name}' não foram
encontrados.")
```

```
else:
    # Calcula as distribuições por janela
    window_dists = compute_window_distributions(
        df=df_uklex,
        doc_id_col=dataset_config["doc_id_col"],
        date_col=dataset_config["date_col"],
        all_doc_embeddings=doc_embeddings,
        window_func=dataset_config["window_func"]
    )

    # Calcula os scores de MMD para todo o período
    df_mmd_scores = calculate_drift_scores(window_dists, metric='mmd')

    print("\nDataFrame de scores original (com outlier):")
    print(df_mmd_scores.head())

    #
============================================================================
===
    # 2. FILTRAGEM CORRETA E ROBUSTA
    # AQUI ESTÁ A MUDANÇA CRUCIAL:
    # Em vez de filtrar por ano, nós simplesmente removemos a PRIMEIRA
LINHA do
    # DataFrame de scores. Isso elimina o outlier inicial, não importa qual
ano seja.
    # .iloc[1:] seleciona todas as linhas a partir da segunda (índice 1).
    #
============================================================================
===
    print("\n[AÇÃO] Removendo o primeiro score (outlier) do DataFrame...")
    df_mmd_scores_filtered = df_mmd_scores.iloc[1:].copy()
    print("\nDataFrame de scores filtrado (sem outlier):")
    print(df_mmd_scores_filtered.head())


    #
============================================================================
===
    # 3. ANÁLISE E PLOTAGEM COM DADOS CORRIGIDOS
    # Agora, todas as análises seguintes usarão os dados limpos.
    #
============================================================================
===
    print("\n--- Análise de Drift MMD (Resultados Finais Corretos) ---")
```

```
    mmd_drift_results_final =
detect_drift_with_thresholds(df_mmd_scores_filtered)
    display_drift_results(mmd_drift_results_final)

    plot_drift_with_correlation(
        df_mmd_scores_filtered,
        mmd_drift_results_final,
        df_uklex_aux_metrics,
        title=f'Drift Semântico (MMD) - {experiment_name} (sem o primeiro
score outlier)',
        color='purple'
    )


##########################################################################
#####
>>> ANÁLISE MMD PARA 'UKLEX_mean_of_chunks' IGNORANDO O PRIMEIRO SCORE
(OUTLIER) <<<
##########################################################################
#####

Coletando distribuições por janela: 100%|██████████| 38/38 [00:00<00:00,
213.03it/s]

Usando Gamma Otimal (calculado pela heurística da mediana): 5.315293

Calculando scores MMD: 100%|██████████| 38/38 [00:13<00:00,  2.79it/s]


DataFrame de scores original (com outlier):
   window     score
0    1987  0.444413
1    1988  0.016092
2    1989  0.015603
3    1990  0.017698
4    1991  0.015154

[AÇÃO] Removendo o primeiro score (outlier) do DataFrame...

DataFrame de scores filtrado (sem outlier):
   window     score
1    1988  0.016092
2    1989  0.015603
3    1990  0.017698
4    1991  0.015154
```

```
5    1992  0.015807

--- Análise de Drift MMD (Resultados Finais Corretos) ---
Limiar de Atenção (2*STD): 0.0235
Limiar Crítico (3*STD):  0.0267

--- [NÍVEL CRÍTICO] ---
Nenhum drift crítico detectado.

--- [NÍVEL DE ATENÇÃO] ---
  - Janela: 2018, Score: 0.025440
```



3.2 Conclusões da Análise para o Dataset UKLEX

Utilizando as métricas de MMD para drift semântico e JSD para drift temático, foi possível identificar diferentes comportamentos evolutivos no discurso jurídico do dataset UKLEX. A análise, no entanto, também expôs a necessidade de tratar anomalias nos dados, como o pico de drift extremo em 1987, provavelmente por conta do baixo volume de documentos, foi e removido para não distorcer as tendências subsequentes.

Uma vez normalizada a análise, a mudança mais reveladora ocorreu a partir de 2012, coincidindo com o contexto do Brexit, onde se observou uma clara convergência na estrutura dos dados. A análise demonstrou que o drift semântico (mudança de estilo e complexidade) precedeu o drift temático (mudança de tópicos), que só se tornou crítico em 2018. Adicionalmente, um pico de drift em 2003, detectado por ambas as métricas, já havia indicado uma mudança pontual significativa tanto em temas quanto em estilo.

Ambas as estratégias de vetorização, Head + Tail Truncation e Chunking & Averaging, foram capazes de identificar os padrões gerais de drift. Contudo, a abordagem Chunking & Averaging mostrou-se mais sensível ao detectar o drift temático crítico de 2018, que não foi capturado pelo Head + Tail. Provavelmente por conta do alto valor da média de tokens, que com o aumento informações temáticas cruciais presentes no meio dos textos podem ter sido perdidas pela estratégia de truncamento além da retirada de grande parte das informações.

4.1 Análise para o Dataset EURLEX

```
# --- Configurações Iniciais ---
dataset_name = "EURLEX"
print(f"\n{'#'*80}\n### INICIANDO PREPARAÇÃO PARA O DATASET: {dataset_name}
###\n{'#'*80}")

# Seleciona a configuração do dataset EURLEX
dataset_config = DATASET_CONFIGS[dataset_name]

# Dicionário mestre para armazenar os embeddings deste dataset
all_doc_embeddings_eurlex = {}

# --- Carregamento e Pré-processamento do DataFrame ---
# A função 'loader_func' específica para o EURLEX será chamada
df_eurlex = dataset_config["loader_func"]()

# --- Cálculo de Métricas Auxiliares ---
# Calcula a contagem de documentos e a média de tokens por ano para o
EURLEX
df_eurlex_aux_metrics = calculate_auxiliary_metrics_by_window(
    df=df_eurlex,
    date_col=dataset_config["date_col"],
    token_col=dataset_config["token_col"],
    window_func=dataset_config["window_func"]
)

print("\n>>> Ambiente para o EURLEX está pronto para os experimentos. <<<")


################################################################################
#####
### INICIANDO PREPARAÇÃO PARA O DATASET: EURLEX ###
################################################################################
#####
```

```
Preparando dataset EURLEX...
Calculando contagem de tokens para os documentos EURLEX...

Calculando tokens EURLEX: 100%|██████████| 65000/65000 [01:14<00:00,
870.53it/s]

EURLEX carregado. Total de 65000 documentos.
Métricas auxiliares por janela calculadas (contagem de docs, média de
tokens).

>>> Ambiente para o EURLEX está pronto para os experimentos. <<<
```

```
strategy_name = "head_tail"
strategy_config = EMBEDDING_STRATEGIES[strategy_name]
print(f"\n{'='*80}\n>>> GERANDO/CARREGANDO EMBEDDINGS (EURLEX) PARA:
{strategy_name} <<<\n{'='*80}")

# Configura o caminho do cache para 'eurlex_headtail'
embedding_call_config = {
    'cache_dir': os.path.join(CACHE_DIR_ROOT,
f"{dataset_config['cache_base_dir']}{strategy_config['cache_suffix']}"),
    'function': strategy_config['function'],
    'params': strategy_config.get('params', {})
}

# Chama a função que gera os embeddings e armazena no dicionário do EURLEX
all_doc_embeddings_eurlex[strategy_name] = generate_document_embeddings(
    df=df_eurlex,
    text_col=dataset_config["text_col"],
    doc_id_col=dataset_config["doc_id_col"],
    token_col=dataset_config["token_col"],
    embedding_config=embedding_call_config
)
```

```
================================================================================
=====
>>> GERANDO/CARREGANDO EMBEDDINGS (EURLEX) PARA: head_tail <<<
================================================================================
=====
Verificando cache em 'cache_embeddings/eurlex_headtail' para 65000
documentos...

Verificando cache: 100%|██████████| 65000/65000 [00:08<00:00, 7613.79it/s]
```

```
Todos os embeddings foram carregados do cache.
```

```
# Define a estratégia a ser utilizada nesta célula
strategy_name = "mean_of_chunks"
strategy_config = EMBEDDING_STRATEGIES[strategy_name]
print(f"\n{'='*80}\n>>> GERANDO/CARREGANDO EMBEDDINGS (EURLEX) PARA:
{strategy_name} <<<\n{'='*80}")

# Configura o caminho do cache para 'eurlex_chunks_overlap'
embedding_call_config = {
    'cache_dir': os.path.join(CACHE_DIR_ROOT,
f"{dataset_config['cache_base_dir']}{strategy_config['cache_suffix']}"),
    'function': strategy_config['function'],
    'params': strategy_config.get('params', {})
}

# Chama a função que gera os embeddings e armazena no dicionário do EURLEX
all_doc_embeddings_eurlex[strategy_name] = generate_document_embeddings(
    df=df_eurlex,
    text_col=dataset_config["text_col"],
    doc_id_col=dataset_config["doc_id_col"],
    token_col=dataset_config["token_col"],
    embedding_config=embedding_call_config
)
```

```
================================================================================
=====
>>> GERANDO/CARREGANDO EMBEDDINGS (EURLEX) PARA: mean_of_chunks <<<
================================================================================
=====
Verificando cache em 'cache_embeddings/eurlex_chunks_overlap' para 65000
documentos...

Verificando cache: 100%|██████████| 65000/65000 [00:08<00:00, 7569.34it/s]

Todos os embeddings foram carregados do cache.
```

```
strategy_name = "head_tail"
experiment_name = f"{dataset_name}_{strategy_name}"
print(f"\n{'='*80}\n>>> PROCESSANDO ANÁLISE (EURLEX) PARA:
{experiment_name} <<<\n{'='*80}")
```

```
# Recupera os embeddings correspondentes do dicionário do EURLEX
doc_embeddings = all_doc_embeddings_eurlex.get(strategy_name)

if not doc_embeddings:
    print(f"ERRO: Embeddings para a estratégia '{strategy_name}' não foram
encontrados.")
else:
    # Agrupa os embeddings em janelas temporais
    window_dists = compute_window_distributions(
        df=df_eurlex,
        doc_id_col=dataset_config["doc_id_col"],
        date_col=dataset_config["date_col"],
        all_doc_embeddings=doc_embeddings,
        window_func=dataset_config["window_func"]
    )

    # --- Análise de Drift Semântico (MMD) ---
    print("\n--- Análise de Drift MMD ---")
    df_mmd_scores = calculate_drift_scores(window_dists, metric='mmd')
    mmd_drift_results = detect_drift_with_thresholds(df_mmd_scores)
    display_drift_results(mmd_drift_results)
    plot_drift_with_correlation(df_mmd_scores, mmd_drift_results,
df_eurlex_aux_metrics,
                                title=f'Drift Semântico (MMD) -
{experiment_name}', color='darkblue')

    # --- Análise de Drift Temático (JSD) ---
    print("\n--- Análise de Drift JSD ---")
    df_jsd_scores = calculate_drift_scores(window_dists, metric='jsd')
    jsd_drift_results = detect_drift_with_thresholds(df_jsd_scores)
    display_drift_results(jsd_drift_results)
    plot_drift_with_correlation(df_jsd_scores, jsd_drift_results,
df_eurlex_aux_metrics,
                                title=f'Drift Temático (JSD) -
{experiment_name}', color='darkcyan')


================================================================================
>>> PROCESSANDO ANÁLISE (EURLEX) PARA: EURLEX_head_tail <<<
================================================================================
```

Coletando distribuições por janela: 100%|██████████| 56/56 [00:00<00:00, 160.23it/s]

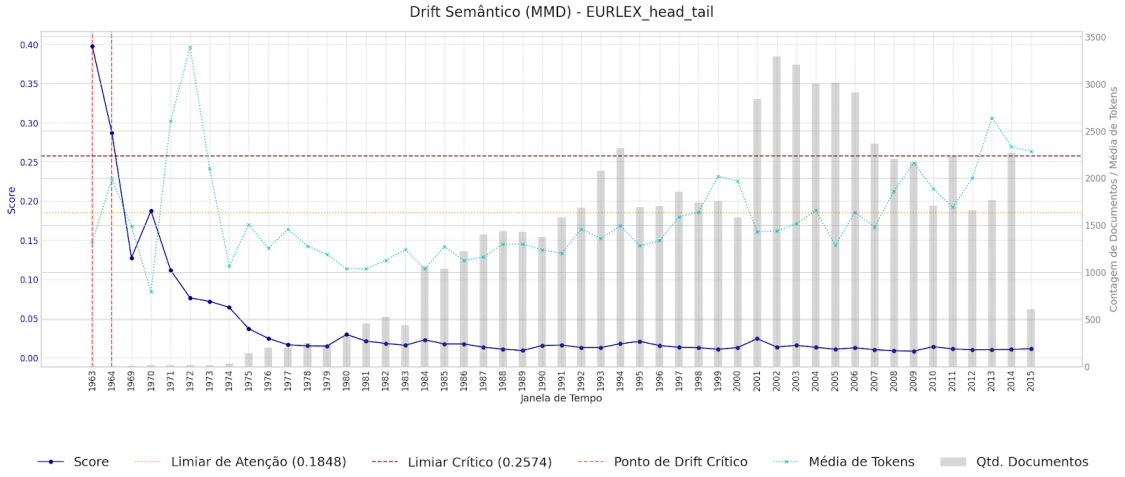
--- Análise de Drift MMD ---
Usando Gamma Otimal (calculado pela heurística da mediana): 1.610090

Calculando scores MMD: 100%|██████████| 56/56 [00:38<00:00, 1.47it/s]

Limiar de Atenção (2*STD): 0.1848
Limiar Crítico (3*STD): 0.2574

--- [NÍVEL CRÍTICO] ---
  - Janela: 1963, Score: 0.397308
  - Janela: 1964, Score: 0.287021

--- [NÍVEL DE ATENÇÃO] ---
  - Janela: 1970, Score: 0.187399



--- Análise de Drift JSD ---

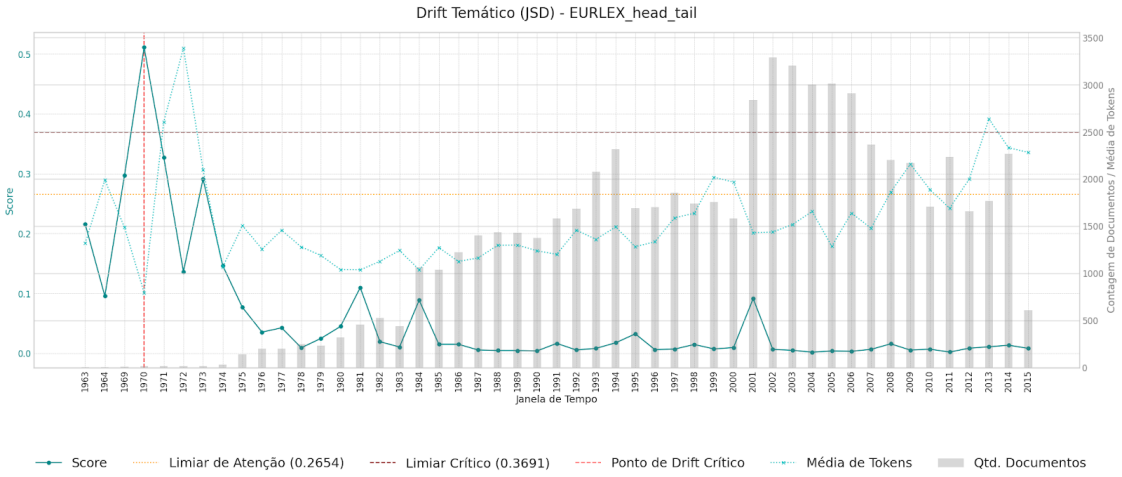
Calculando scores JSD: 100%|██████████| 56/56 [00:22<00:00, 2.46it/s]

Limiar de Atenção (2*STD): 0.2654
Limiar Crítico (3*STD): 0.3691

--- [NÍVEL CRÍTICO] ---
  - Janela: 1970, Score: 0.510967

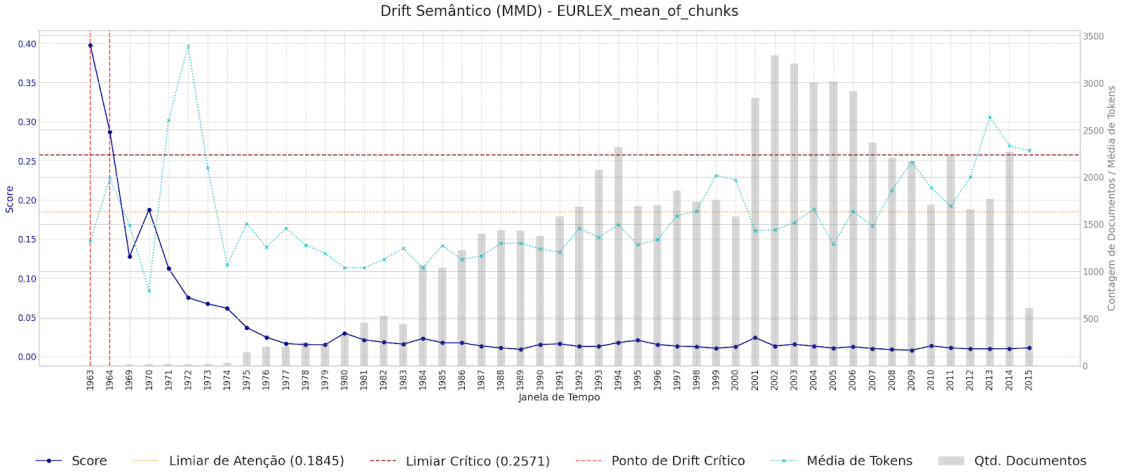
--- [NÍVEL DE ATENÇÃO] ---

```
  - Janela: 1969, Score: 0.297063
  - Janela: 1971, Score: 0.327369
  - Janela: 1973, Score: 0.291327
```

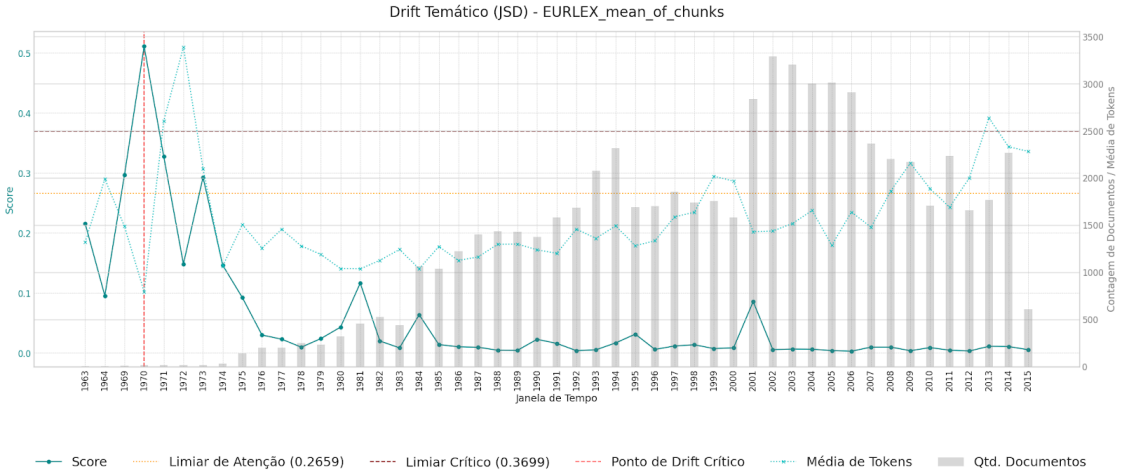



```
# Define qual estratégia será analisada nesta célula
strategy_name = "mean_of_chunks"
experiment_name = f"{dataset_name}_{strategy_name}"
print(f"\n{'='*80}\n>>> PROCESSANDO ANÁLISE (EURLEX) PARA:
{experiment_name} <<<\n{'='*80}")

# Recupera os embeddings do dicionário do EURLEX
doc_embeddings = all_doc_embeddings_eurlex.get(strategy_name)

if not doc_embeddings:
    print(f"ERRO: Embeddings para a estratégia '{strategy_name}' não foram
encontrados.")
else:
    # Agrupa os embeddings em janelas temporais
    window_dists = compute_window_distributions(
        df=df_eurlex,
        doc_id_col=dataset_config["doc_id_col"],
        date_col=dataset_config["date_col"],
        all_doc_embeddings=doc_embeddings,
        window_func=dataset_config["window_func"]
    )

    # --- Análise de Drift Semântico (MMD) ---
    print("\n--- Análise de Drift MMD ---")
    df_mmd_scores = calculate_drift_scores(window_dists, metric='mmd')
    mmd_drift_results = detect_drift_with_thresholds(df_mmd_scores)
```

```
    display_drift_results(mmd_drift_results)
    plot_drift_with_correlation(df_mmd_scores, mmd_drift_results,
df_eurlex_aux_metrics,
                                title=f'Drift Semântico (MMD) -
{experiment_name}', color='darkblue')

    # --- Análise de Drift Temático (JSD) ---
    print("\n--- Análise de Drift JSD ---")
    df_jsd_scores = calculate_drift_scores(window_dists, metric='jsd')
    jsd_drift_results = detect_drift_with_thresholds(df_jsd_scores)
    display_drift_results(jsd_drift_results)
    plot_drift_with_correlation(df_jsd_scores, jsd_drift_results,
df_eurlex_aux_metrics,
                                title=f'Drift Temático (JSD) -
{experiment_name}', color='darkcyan')


================================================================================
>>> PROCESSANDO ANÁLISE (EURLEX) PARA: EURLEX_mean_of_chunks <<<
================================================================================

Coletando distribuições por janela: 100%|██████████| 56/56 [00:00<00:00,
147.01it/s]


--- Análise de Drift MMD ---
Usando Gamma Otimal (calculado pela heurística da mediana): 1.610090

Calculando scores MMD: 100%|██████████| 56/56 [00:38<00:00,  1.47it/s]

Limiar de Atenção (2*STD): 0.1845
Limiar Crítico (3*STD):  0.2571

--- [NÍVEL CRÍTICO] ---
  - Janela: 1963, Score: 0.397308
  - Janela: 1964, Score: 0.287052

--- [NÍVEL DE ATENÇÃO] ---
  - Janela: 1970, Score: 0.187459
```




```
--- Análise de Drift JSD ---

Calculando scores JSD: 100%|██████████| 56/56 [00:24<00:00,  2.30it/s]

Limiar de Atenção (2*STD): 0.2659
Limiar Crítico (3*STD):  0.3699

--- [NÍVEL CRÍTICO] ---
  - Janela: 1970, Score: 0.510967

--- [NÍVEL DE ATENÇÃO] ---
  - Janela: 1969, Score: 0.297063
  - Janela: 1971, Score: 0.327369
  - Janela: 1973, Score: 0.293040
```



4.2 Conclusões da Análise para o Dataset EURLEX

A análise de drift temporal no corpus EURLEX mapeou com sucesso as fases cruciais da história legislativa da União Europeia. A consistência dos resultados entre as estratégias de vetorização head_tail e mean_of_chunks sugere que a natureza bem-estruturada dos documentos do EURLEX torna ambas as abordagens eficazes.

Os resultados revelam diferentes naturezas de drift ao longo do tempo. A fase inicial, na década de 1960, foi marcada por um intenso drift, com altos scores de MMD refletindo a volatilidade de um sistema jurídico em sua fundação, onde a terminologia e os formatos ainda não estavam consolidados. Em seguida, um drift abrupto de natureza temática (JSD) ocorreu em torno de 1970, correlacionado à primeira grande expansão da Comunidade Europeia, que introduziu um leque de novos tópicos legislativos.

Após essa fase de ruptura, a análise revela um claro padrão de drift incremental convergente. Este fenômeno está diretamente ligado à criação da própria União Europeia pelo Tratado de Maastricht em 1992. Como uma entidade jurídica recém-formada, não havia um padrão documental consolidado. O que os dados mostram é o processo de solidificação desse padrão: a diminuição contínua da variância (scores de drift) reflete a transição de uma fase de heterogeneidade para uma de maior homogeneidade, à medida que uma estrutura e linguagem comuns eram estabelecidas. Portanto, a convergência para um novo patamar de estabilidade não é ausência de mudança, mas sim a evidência da formação de um novo conceito estável para o "documento jurídico europeu", refletindo a maturação do corpus jurídico.

5. ANÁLISE COMPARATIVA E CONCLUSÕES

```
summary_data = []
for name, results in all_results.items():
    mmd_drift_windows = [str(d[0]) for d in results["mmd_drifts_critical"]]
    jsd_drift_windows = [str(d[0]) for d in results["jsd_drifts_critical"]]
    summary_data.append({
        "Experimento": name,
        "Drifts Semânticos Críticos (MMD)": ", ".join(mmd_drift_windows) if
mmd_drift_windows else "Nenhum",
        "Drifts Temáticos Críticos (JSD)": ", ".join(jsd_drift_windows) if
jsd_drift_windows else "Nenhum"
    })

df_summary = pd.DataFrame(summary_data)

print("\n\n")
```

```
print("="*80)
print("TABELA CONSOLIDADA DE DRIFTS CRÍTICOS DETECTADOS")
print("="*80)
display(df_summary)
```

```
================================================================================
=====
TABELA CONSOLIDADA DE DRIFTS CRÍTICOS DETECTADOS
================================================================================
=====

             Experimento Drifts Semânticos Críticos (MMD)  \
0        UKLEX_head_tail                             1987
1   UKLEX_mean_of_chunks                             1987
2       EURLEX_head_tail                       1963, 1964
3  EURLEX_mean_of_chunks                       1963, 1964

  Drifts Temáticos Críticos (JSD)
0                          Nenhum
1                            2018
2                            1970
3                            1970
```

identificar com sucesso os principais pontos de inflexão histórica em ambos os corpora jurídicos. A tabela consolidada de drifts críticos resume esses eventos, destacando as datas em que ocorreram as mudanças mais significativas.

Essas datas são mais do que achados históricos; são marcos operacionais para o ciclo de vida de modelos de machine learning no domínio jurídico. A detecção de drift oferece um roteiro data-driven para estratégias de treinamento e manutenção, tais como:

Retreinamento Seletivo Um drift temático crítico, como o detectado em 1970 no EURLEX (devido à expansão da UE) ou em 2018 no UKLEX (contexto do Brexit), sinaliza que o conceito subjacente dos dados mudou fundamentalmente. Um modelo de classificação ou busca treinado com dados anteriores a essas datas se tornaria obsoleto. A detecção do drift, portanto, funciona como um gatilho para o retreinamento do modelo com dados mais recentes, garantindo sua acurácia e relevância contínuas.

Segmentação de Dados por "Era": Em vez de um único modelo monolítico, os drifts detectados permitem segmentar os dados em "eras" conceituais distintas. Por exemplo, para o EURLEX, poderiam ser criados modelos especializados para a "era da fundação"

(pré-1970) e a "era da consolidação" (pós-1970), cada um otimizado para as características de seu respectivo período.

Identificação de Anomalias para Pré-processamento: O drift semântico crítico em 1987 no UKLEX, identificado como um provável artefato, demonstra o valor do método para o controle de qualidade. Essa detecção informa a etapa de pré-processamento, sugerindo a remoção ou investigação de dados anômalos que poderiam corromper o treinamento de um modelo.

Em suma, a metodologia empregada não apenas valida hipóteses históricas, mas fornece uma ferramenta prática e quantificável para guiar a construção de sistemas de IA mais robustos e adaptativos.