Aprendizagem Profunda

Sistemas Inteligentes

David Teixeira PG55929
Eduardo Cunha PG55939
Jorge Rodrigues PG55966
Tiago Rodrigues PG56013

Março, 2025

Indice

1. Introdução

A crescente sofisticação dos modelos de inteligência artificial na geração de texto levanta um desafio crucial: como distinguir conteúdos escritos por humanos daqueles produzidos por IA? Esta questão torna-se particularmente relevante à medida que os modelos evoluem, tornando-se cada vez mais indistinguíveis da escrita humana.

Este relatório apresenta a nossa análise e implementação de diferentes abordagens de machine learning para abordar este problema. Exploramos desde modelos mais simples, como regressão logística, até redes neuronais profundas e *transformers* de grande escala, abrangendo tanto implementações feitas do zero como aquelas que recorrem a bibliotecas reconhecidas, como o TensorFlow. Além disso, investigamos técnicas avançadas, como engenharia de prompts, para melhorar a precisão na deteção de textos gerados por IA.

A metodologia adotada baseia-se na aplicação e estudo das implementações fornecidas pelo professor nas aulas práticas, complementadas por investigação adicional sobre técnicas avançadas. Por meio de uma abordagem prática e experimental, este estudo documenta os processos, decisões e resultados obtidos, contribuindo para um melhor entendimento desta área em constante evolução.

2. Metodologias para Criação do Dataset

A construção do *dataset* para a deteção de textos gerados por inteligência artificial seguiu um processo estruturado, englobando a obtenção de dados de diferentes fontes, a sua limpeza e a preparação para o seu uso nos modelos criados.

2.1. Agregação e Normalização dos Dados

A primeira etapa consistiu na recolha de dados provenientes de diversas fontes, incluindo conjuntos de dados públicos, como "artem9k/ai-text-detection-pile" e "NicolaiSivesind/human-vs-machine". Conjuntos como o quinto (data_set.csv) e o sexto (en_news_human_features_df.pkl, en_news_gpt_features_df.pkl,...) foram escolhidos e utilizados principalmente por abordarem temas científicos. Considerando que o objetivo do estudo era focado em artigos científicos, acreditamos que estes conjuntos de dados apresentavam frases semelhantes às que poderiam ser encontradas no *dataset* de teste utilizado pelo docente para avaliar os modelos.

Cada conjunto de dados foi transformado num *DataFrame* padronizado com as colunas ***text*** (conteúdo textual) e ***source*** (origem do texto: "human" ou "ai"). Foi realizado um mapeamento dos valores das *labels* para garantir uniformidade entre diferentes fontes, garantindo que "human" fosse representado como 0 e "ai" como 1.

Posteriormente, priorizamos o uso do quinto *dataset* em conjunto com um conjunto gerado pelo grupo, via *apis* como o "arXiv", para recolher conteúdo de *papers*, e o "OpenRouter" para geração por meio de modelos *open source* conhecidos como o "mistral-nemo".

2.2. Processamento e Particionamento dos Dados

Com o *dataset* consolidado, procedeu-se à sua separação em diferentes conjuntos, nomeadamente df_tail e df_head, para reduzir o tamanho do dataset sem comprometer a distribuição

das classes. As labels foram convertidas de "source" para targetLabel para evitar que a palavra "source", frequentemente presente nos textos, causasse problemas no encoding. Utilizou-se a ferramenta CountVectorizer para transformar o texto em representações binárias, limitadas a 10.000 features. Após a transformação dos textos em representações numéricas, os dados foram divididos em treino (70%), validação (10%) e teste (20%), garantindo a remoção de duplicados para evitar redundâncias. A representação dos dados variou consoante os modelos, pelo que o uso de *Bag-of-words* não foi a única opção adotada, como será abordado nas secções seguintes.

2.3. Armazenamento e Disponibilização

Os *datasets* finais foram guardados em ficheiros CSV normalizados, facilitando a integração em modelos nas fases seguintes. Adicionalmente, o mapeamento de palavras utilizado na tokenização foi guardado num ficheiro JSON para referência futura. Optamos ainda por gerar um *subdataset* do original, pois dada a complexidade do *datase* gerado seria inviável desenvolver e fazer *debugging* de modelos num *dataset* tão complexo, sendo este então usado apenas depois do modelo estar devidamente implementado.

3. Modelos Raiz

Os modelos raiz foram definidos com base na implementação incompleta fornecida pelo professor, que posteriormente finalizámos.

3.1. DNNs

As redes neuronais profundas (DNNs) foram a abordagem mais explorada nesta etapa do trabalho. Para melhorar a capacidade de generalização dos modelos, implementámos diferentes funções de ativação, indo além da *ReLU*. Testámos alternativas como a *Tanh*, que não obterem resultados particularmente melhores que a *ReLU* e a *Sigmoid*. Permitimos também a escolha da configuração de inicialização de pesos, dado que diferentes funções de ativação funcionam melhor ou pior dependendo do método de escolha dos pesos iniciais. Dada a natureza binária do problema, a última camada utilizava sempre a função de ativação *Sigmoid*, como a *loss function* predileta seria a *Binary Cross Entropy*

Além disso, incorporámos técnicas de regularização para evitar *overfitting*. Utilizámos *Dropout Layers*, que desativam aleatoriamente uma fração dos neurónios durante o treino, e regularização *L1* e *L2*, que adicionam penalizações aos pesos da rede para promover soluções mais robustas. O impacto destas estratégias foi analisado ao longo de diferentes iterações.

Para avaliar o desempenho das redes, introduzimos métricas adicionais além da *Accuracy*. Considerámos métricas como *Precision*, *Recall* e *F1-score*, que são particularmente úteis num contexto de deteção de textos gerados por IA, onde o equilíbrio entre falsos positivos e falsos negativos pode ser crítico.

No que diz respeito à otimização, comparámos o desempenho de diferentes algoritmos, incluindo o *Adam* e o *RMSProp*. Cada um destes métodos apresentou vantagens consoante a complexidade do modelo e a estrutura dos dados, sendo que o *Adam*, pela sua capacidade adaptativa, demonstrou ser a escolha mais eficaz na maioria dos cenários.

Por fim, utilizámos estratégias de melhoria do treino, como a implementação de *Callbacks* para monitorizar o desempenho do modelo ao longo das épocas. Entre estes, destacamos o uso do *Early Stopping*, que interrompe automaticamente o treino quando não há melhorias significativas na validação, evitando *overfitting*

Paralelamente, explorámos diferentes arquiteturas de redes neuronais, variando o número de camadas, a quantidade de neurónios por camada e a combinação de funções de ativação, para encontrar a configuração mais eficiente para o nosso problema. A opção escolhida para a competição será discutida posteriormente.

3.2. RNNs

As Redes Neuronais Recorrentes (RNNs) foram exploradas como uma abordagem alternativa, mas enfrentámos desafios significativos na sua implementação devido à incompatibilidade entre a nossa abstração e a estrutura dos dados esperada pelo código base fornecido pelo professor.

Inicialmente, a nossa abstração utilizava uma abordagem *1D*, onde cada camada recorrente trabalhava apenas com o número de unidades especificado. No entanto, o código disponibilizado pelo professor estava estruturado para uma abordagem *3D*, exigindo entradas no formato (tamanho do *batch*, *timesteps*, número de unidades). Esta diferença estrutural causou um conflito entre a nossa implementação e a arquitetura exigida para as RNNs, tornando necessária uma adaptação.

Para resolver este problema, utilizámos *embeddings* como um mecanismo de conversão para ajustar os dados à estrutura exigida pelas RNNs. Criámos uma adaptação que transformava as representações de texto, permitindo que a entrada da rede passasse a conter as três dimensões necessárias. Além disso, adicionámos a opção de *return sequence*, que permite à rede devolver sequências completas em vez de apenas o último estado oculto. Esta funcionalidade possibilitou a conjugação das RNNs com as restantes camadas da abstração, aumentando a flexibilidade da nossa implementação.

Apesar da adaptação técnica bem-sucedida, a performance das RNNs revelou-se ineficiente. O tempo de treino era excessivamente elevado, limitando-nos a trabalhar apenas com conjuntos de dados muito pequenos. Além disso, a qualidade dos *embeddings* utilizados foi um fator crítico na degradação do desempenho. Esta combinação de fatores resultou em modelos com fraca capacidade de generalização e desempenho insuficiente para a competição.

Diante destes desafios, concluímos que, embora as RNNs fossem uma abordagem teoricamente interessante, a sua aplicação prática neste contexto não se revelou viável, levando-nos a priorizar outras arquiteturas nesta etapa do projeto.

3.3. Embeddings

Para modelos baseados em *embeddings*, foi aplicada uma *tokenização* básica, convertendo palavras em índices numéricos. Posteriormente, as sequências foram padronizadas para um comprimento fixo. Utilizou-se a base de *embeddings* GloVe (100 dimensões) para gerar representações vetoriais das palavras.

Dada as diferenças entre DNN e RNN, foram criadas duas classes, *EmbeddingLayer* e *EmbeddingLayerRNN*, que funcionam como camadas de *embeddings*. A principal diferença entre *EmbeddingLayer* e *EmbeddingLayerRNN* reside na forma como processam os dados de entrada e saída. A *EmbeddingLayer* foi projetada para trabalhar com palavras dados do tipo *Binary Bag of Words*, convertidos no formato de índices, e retornando um vetor de *embedding* por palavra encontrada. Por outro lado, a *EmbeddingLayerRNN* foi concebida para processamento sequencial, recebendo sequências de índices de palavras e retornando uma matriz tridimensional, onde cada palavra na sequência recebe o seu respetivo vetor de *embedding*.

Outra diferença importante está na forma como o *back propagating* é realizada. Enquanto a *EmbeddingLayer* acumula gradientes para cada palavra individualmente, a EmbeddingLayerRNN precisa iterar sobre todas as posições dentro de cada sequência para atualizar corretamente os *embeddings* usados em diferentes *timesteps*.

Além disso, ambas as camadas oferecem a opção de serem treináveis, permitindo que os *embeddings* sejam ajustados durante o treino da rede, ou de não serem treináveis, o que é particularmente útil quando se pretende utilizar *embeddings* pré-treinados, como o GloVe.

3.4. Regressão Logística

4. Tensorflow

A segunda etapa do projeto envolve o uso da conhecida biblioteca *Tensorflow* para a construção de Redes Neuronais profundas, melhorando e aprofundando os conceitos explorados na nossa implementação.

4.1. DNNs

Numa primeira abordagem, o processo começou com o pré-processamento dos dados textuais, utilizando um *Tokenizer* do *Keras* para converter as palavras em sequências numéricas. Foi definida uma rede neuronal densa com camadas de dropout para mitigar o overfitting e utilizou-se *Early Stopping* para interromper o treino caso a validação deixasse de melhorar, juntamente com *Model Checkpoint* para salvar o melhor modelo.

O desempenho inicial foi modesto, com cerca de 54% de *accuracy* no teste, sugerindo subajuste ou problemas na representação dos dados. Numa segunda tentativa, substituiu-se a tokenização por vetores **TF-IDF**, que capturam a importância relativa das palavras no texto. Isso melhorou drasticamente os resultados, atingindo cerca de 94% de *accuracy*, mostrando que a representação dos dados é crucial para o sucesso do modelo.

Seguidamente, noutra abordagem, foi definida uma rede neuronal densa. O foco desta abordagem foi testar diferentes numeros de *epochs*, *batch sizes*, otimizadores e topologias, de forma a procurar afinar os hiperparâmetros.

Posto isto explorámos ainda diferentes estratégias de representação de texto, com particular foco no uso de *embeddings*. Testámos duas abordagens principais:

- ***Embeddings* treinados do zero**: Desenvolvemos o nosso próprio *embedding*, permitindo que a rede aprendesse representações vetoriais para as palavras à medida que o treino avançava. Embora esta abordagem desse à rede maior flexibilidade na adaptação ao problema

específico, revelou-se computacionalmente dispendiosa e exigia uma abundância de dados para gerar *embeddings* de qualidade.

- ***Embeddings* pré-treinados com *GloVe***: Optámos por utilizar *embeddings* pré-treinados do modelo *GloVe* (Global Vectors for Word Representation). Ao integrar estas representações no nosso modelo, conseguimos capturar relações semânticas mais ricas desde o início do treino.

A comparação entre ambas as abordagens mostrou que os modelos treinados com *GloVe* apresentaram desempenho significativamente superior relativamente aos modelos que aprenderam os *embeddings* do zero. A utilização de *embeddings* pré-treinados permitiu que os modelos convergissem mais rapidamente e alcançassem melhores classificações com menos dados, uma vez que já incorporavam conhecimento semântico sobre as palavras.

4.2. RNNs

A *SimpleRNN* foi a abordagem mais básica que testámos. Esta arquitetura consiste numa camada recorrente simples, onde cada neurónio recebe a entrada anterior e a atual, permitindo uma propagação sequencial da informação. No entanto, esta abordagem revelou-se ineficaz para o nosso problema. O principal problema da *SimpleRNN* foi a dissipação, um fenómeno comum quando redes recorrentes simples lidam com sequências mais longas. Como os textos analisados podem conter frases relativamente extensas, esta limitação impediu a camada de capturar dependências de longo prazo entre palavras

A segunda abordagem testada foi as redes *GRU*, uma versão mais avançada das redes recorrentes, que introduz um mecanismo de *gates* para melhorar a retenção de informação ao longo do tempo. As GRU utilizam dois *gates*, um *gate* de atualização, que decide quanta informação da entrada atual deve ser mantida, e um de redefinição, que determina quanta informação do passado deve ser esquecida. Uma das principais vantagens das *GRU* em relação às *SimpleRNN* foi a capacidade de aprender padrões a longo prazo sem sofrer tanto com dissipação. Comparado com *LSTM*, as *GRU* tem uma estrutura mais simples e menos parâmetros, tornando-as computacionalmente mais eficiente e permitindo tempos de treino mais rápidos.

A terceira abordagem testada foram as *LSTM*, uma das arquiteturas mais populares para redes recorrentes. Esta arquitetura resolve o problema da dissipação de gradientes ao introduzir um mecanismo mais sofisticado de gestão da memória, por meio de três *gates* principais, um *gate* de *input*, que decide quais informações devem ser adicionadas à memória, um *gate* de esquecimento, que controla quais informações podem ser descartadas e um *gate* de *output*, que regula a informação que será propagada para os próximos estados da rede. Esta estrutura permitiu às redes *LSTM* preservar contexto por períodos mais longos do que as outras abordagens. Durante os testes, verificámos que o *LSTM* teve um desempenho ligeiramente superior às *GRU*, mas com um treino mais demorado.

Optámos por utilizar um *embedding* treinado por nós, porque quisemos testar representações com dimensões superiores a 100. Mais dimensões permitem capturar relações semânticas mais detalhadas, fornecendo às redes uma representação mais expressiva do texto, o que esperamos que seja determinante no desempenho na tarefa de classificação. Este aspeto revelou-se particularmente relevante para modelos como *LSTM* e *GRU*, que beneficiam de *embeddings* de maior dimensão para melhorar a sua capacidade de aprendizagem de dependências temporais.

Após testar as três arquiteturas, concluímos que a *SimpleRNN* não é competitiva devido às suas limitações na retenção de contexto. Entre *GRU* e *LSTM*, a *LSTM* obteve melhores resultados, devido à sua maior capacidade de preservar informação a longo prazo e à estabilidade durante o treino. No entanto, reconhecemos que as *GRU* podem ser uma alternativa viável, dado que os recursos computacionais são limitados, sendo esta abordagem mais eficiente computacionalmente, e também porque o seu desempenho foi relativamente próximo ao das *LSTM*.

4.3. Transformers

No que toca a *Deep Learning* os *Transformers* representam uma das arquiteturas mais avançadas e populares na área. Baseiam-se em mecanismos de atenção que permitem ao modelo capturar dependências contextuais de longo alcance entre palavras num texto (sem limitação sequencial característica de outras arquiteturas como as RNNs). O *Transformer* implementado segue uma estrutura típica: preparação e tokenização dos dados, carregamento do modelo, definição do otimizador e função de perda, e finalmente, o processo de treino e avaliação.

O modelo foi treinado durante 10 *epochs*, permitindo uma avaliação mais robusta do seu comportamento. No entanto, rapidamente se constatou que esta abordagem apresenta um custo computacional considerável. Cada *epoch* exigiu um tempo de processamento elevado, resultando num total aproximado de 80 minutos para concluir o treino completo. Este fator torna a utilização de *transformers* menos prática em contextos com recursos computacionais limitados ou onde rapidez de execução é um requisito.

Apesar do tempo investido no treino, os resultados obtidos não se revelaram substancialmente superiores quando comparados com abordagens mais leves e eficientes. O modelo apresentou uma melhoria modesta na capacidade de distinguir textos gerados por IA, mas os ganhos marginais não justificaram, nesta experiência, o custo computacional elevado.

Para testar a eficácia do *BERT* nesta tarefa, treinámos dois modelos distintos, um modelo com menos *epochs* de treino e um modelo com mais *epochs* de treino.

O primeiro modelo foi treinado com um número reduzido de *epochs*, permitindo uma avaliação inicial do desempenho do *BERT* na tarefa. Como esperado, mesmo com poucas *epochs*, o modelo já demonstrou um desempenho competitivo, beneficiando do conhecimento adquirido durante o seu pré-treino.

O segundo modelo foi treinado durante mais *epochs*, embora o número total de *epochs* tenha sido ainda relativamente baixo devido ao elevado custo computacional do treino do *BERT*. Cada época *epoch* exigia um tempo de processamento significativo, tornando inviável uma otimização extensiva sem acesso a recursos computacionais mais potentes. No entanto, mesmo com um número limitado de *epochs*, o modelo conseguiu melhorar ligeiramente os seus resultados, aprendendo padrões mais refinados na deteção de textos gerados por IA.

4.4. LLMs

Atualmente, existem diversas abordagens baseadas em LLMs no mercado. Considerando as suas capacidades, decidimos implementar algumas estratégias, como Zero-shot, One-shot, engenharia de prompt e RAG (Retrieval-Augmented Generation), utilizando modelos pré-existentes para avaliar a eficácia dessas abordagens na melhoria dos resultados.

4.4.1. Zero Shot

Começámos com a abordagem Zero-shot, na qual um modelo é capaz de classificar amostras em categorias que não foram vistas durante o treino. Isso é possível porque modelos pré--treinados, como **BART**, **RoBERTa** e **DistilRoBERTa**, já possuem um entendimento geral da linguagem e podem ser utilizados para comparar novos textos com descrições previamente conhecidas.

Para a sua aplicação, começamos por definir as classes. Foram criadas duas descrições curtas que representavam textos humanos e textos gerados por IA:

- **Texto Humano:** "Texto escrito por um ser humano, criativo e expressivo."
- **Texto de IA:** "Texto gerado por inteligência artificial, previsível e estruturado."

A seguir, procedemos à extração de embeddings, utilizando os modelos para converter tanto os textos do conjunto de dados como as descrições das classes em vetores numéricos, representando as suas características semânticas. Em seguida, calculamos a similaridade do cosseno entre o embedding de cada texto e os embeddings das descrições das classes.

Com base na maior similaridade, o texto foi classificado como humano ou gerado por IA. A avaliação foi realizada nas primeiras 100 amostras do conjunto de dados. A *accuracy* foi então calculada, comparando as previsões com os rótulos reais, obtendo, no entanto, resultados que não foram particularmente convincentes.

4.4.2. One Shot

De seguida, implementamos uma estratégia de One-Shot Learning, uma técnica de *machine learning* em que o modelo é treinado para classificar ou reconhecer uma classe com apenas um exemplo de treino. O modelo faz uma comparação entre o exemplo conhecido e as novas entradas para realizar a classificação.

Para implementar esta estratégia com os modelos **BERT**, **RoBERTa** e **BART**, seguimos os seguintes passos:

Primeiro, selecionamos um exemplo de cada classe no conjunto de dados: um exemplo de um texto rotulado como "humano" e outro rotulado como "IA". Estes exemplos foram usados como representações das classes.

Em seguida, calculamos os embeddings de cada exemplo selecionado. Os embeddings foram obtidos através dos tokenizers e das funções dos respetivos modelos, que convertem o texto em vetores de alta dimensionalidade.

Para classificar um novo texto, o modelo compara o embedding do novo texto com os embeddings dos exemplos das classes "humano" e "IA". A classificação é realizada com base na similaridade do cosseno entre o embedding do texto e os embeddings das classes.

A precisão do modelo foi avaliada utilizando os primeiros 100 textos do conjunto de dados, obtendo-se *accuracies* bastante elevadas.

4.4.3. Engenharia de Prompt

A Engenharia de Prompt consiste na prática de criar prompts de forma a orientar o modelo a gerar saídas de alta qualidade. O objetivo é fornecer ao modelo um contexto claro e instruções específicas, de modo a obter a resposta desejada.

Na implementação da Engenharia de Prompt, foi utilizado o modelo **BERT** para classificar textos como "humano" ou "IA". Para a criação do prompt, foi desenvolvido o seguinte prompt estruturado para a tarefa de classificação: **"Classify the following text as 'human' or 'AI': {text}"**. Ele define claramente a tarefa e orienta o modelo a classificar o texto como "humano" ou "IA". Como o modelo foi treinado principalmente em inglês, a instrução é fornecida em inglês para garantir uma interpretação mais precisa pela sua arquitetura.

O texto de entrada foi tokenizado utilizando o tokenizador BERT, o que envolve a conversão do texto em tokens numéricos processáveis pelo modelo. Além disso, foi aplicado truncamento e preenchimento para garantir que todos os textos tivessem o mesmo comprimento, uma exigência do modelo BERT.

Few Shot

Os resultados de accuracy obtidos com a Engenharia de Prompt não foram particularmente interessantes, pelo que decidimos implementar também a técnica de Few-Shot Learning. Nesta abordagem, incluímos um pequeno número de exemplos no prompt. O Few-Shot Learning é uma técnica na qual o modelo é alimentado com poucos exemplos para aprender a realizar a tarefa de forma eficaz.

Neste caso, o modelo BERT foi ajustado para o Few-Shot Learning, incluindo três exemplos de textos classificados como "humano" e três exemplos de texto classificados como "IA" no prompt, com o objetivo de fornecer contexto ao modelo. O prompt foi estruturado para incluir esses exemplos antes do novo texto a ser classificado, orientando o modelo a fazer a previsão com base nesses exemplos.

A *accuracy* foi calculada comparando as previsões do modelo com os rótulos reais dos dados. Utilizando as 100 primeiras amostras do conjunto de dados, o *accuracy* obtido foi instável, dependendo dos exemplos fornecidos. No entanto, no resultado final, o desempenho revelou-se satisfatório. Ainda assim, devido a essa instabilidade, o modelo não se mostrou totalmente fiável. Por esse motivo, decidimos aplicar a mesma abordagem a um modelo mais recente e avançado, o GPT-2. No entanto, os resultados permaneceram instáveis.

4.4.4. RAG (Retrieval-Augmented Generation)

O RAG (Retrieval-Augmented Generation) é uma abordagem que combina a geração de texto com a recuperação de informações externas. O modelo consulta uma base de dados para enriquecer suas respostas, sendo especialmente útil em tarefas que exigem conhecimento factual específico, como em question answering. No entanto, decidimos não implementá-lo no nosso caso de estudo por vários motivos. A implementação do RAG exigiria a criação de um sistema de recuperação de dados externos, o que adicionaria complexidade ao processo e exigiria recursos significativos para sua manutenção. Além disso, o problema que estávamos a resolver, a classificação de textos como "humano" ou "IA", não necessita de informações externas, podendo ser bem resolvido com modelos como BERT, RoBERTa e BART. A utilização

do RAG também traria um aumento no tempo de treino e nos requisitos computacionais, o que não justificaria os ganhos no desempenho para o nosso caso específico. Por estes motivos, optámos por abordagens mais simples e eficazes, como os modelos de classificação anteriormente mencionados.

5. Resultados Obtidos

Este capítulo tem como objetivo apresentar e analisar os resultados obtidos nas diferentes submissões, discutindo o impacto de cada uma na melhoria dos modelos desenvolvidos.

5.1. Primeira Submissão

Na primeira submissão, correspondente aos modelos raiz, alcançámos o segundo lugar, com uma taxa de acerto de 73% em ambas as submissões. Optámos por não atualizar o código destes modelos, uma vez que considerámos o desempenho aceitável e, além disso, não haveria novas submissões de modelos raiz. Assim, concentrámos os nossos esforços no desenvolvimento de novos modelos, explorando abordagens mais avançadas com recurso ao TensorFlow.

5.2. Segunda Submissão

Na segunda submissão, os resultados não foram os esperados: obtivemos 67% e 69% nas submissões com um Transformer BERT e uma LSTM RNN. Os resultados foram ligeiramente inferiores aos da primeira submissão, o que não era suposto. Ainda assim, acreditamos que realizámos um bom trabalho no desenvolvimento dos modelos, especialmente tendo em conta que os nossos resultados de validação são muito semelhantes aos obtidos. Consideramos, por isso, que conseguimos mitigar problemas como overfitting.

Para a próxima submissão, cabe-nos testar exaustivamente outros modelos já desenvolvidos e ajustá-los de forma a melhorar os resultados na submissão final.

5.3. Terceira Submissão

primeiro lugar

6. Conclusão

Neste trabalho, desenvolvemos tanto modelos raiz como modelos baseados em bibliotecas consolidadas na indústria, com o objetivo de determinar se um texto foi escrito por um humano ou gerado por inteligência artificial. Este relatório descreve as abordagens adotadas, as implementações realizadas e os resultados obtidos ao longo do processo.

Acreditamos ter cumprido todos os objetivos propostos, uma vez que conseguimos desenvolver um conjunto de modelos robustos e eficazes para abordar este problema, que continua a ser um tema de grande relevância e ainda em investigação. No contexto da competição, os resultados alcançados foram bastante competitivos, o que reflete a qualidade do estudo e da preparação dedicados ao desenvolvimento dos modelos.

Além disso, este trabalho revelou-se extremamente valioso para a consolidação e aprofundamento dos conceitos abordados nas aulas, permitindo-nos aplicar na prática os conhecimentos adquiridos e explorar de forma mais aprofundada os temas estudados.