Deteção de Texto Gerado por IA usando Modelos de Aprendizagem Profunda

Pedro Azevedo pg57897
José Fraga pg55970
Ruben Silva pg57900
Rui Pinto pg56010

Abril 2025

Resumo

Este relatório descreve o desenvolvimento de vários modelos para a deteção de textos gerados por Inteligência Artificial (IA) versus textos escritos por humanos. Foram exploradas implementações próprias (apenas com NumPy) bem como modelos desenvolvidos com TensorFlow/Keras. O pipeline inclui a construção dos datasets, treino dos modelos, avaliação e comparação final.

1 Introdução

Com o crescimento dos modelos de linguagem baseados em inteligência artificial (IA), como o ChatGPT ou Gemini, torna-se cada vez mais difícil distinguir entre textos gerados por IA e textos escritos por humanos. Esta distinção é relevante em várias áreas, desde a educação à deteção de desinformação.

O objetivo deste projeto é desenvolver modelos capazes de classificar automaticamente se um excerto de texto foi escrito por um humano ou gerado por IA. Para isso, foi construído um pipeline completo que inclui a criação de datasets, pré-processamento dos dados, treino de modelos com diferentes arquiteturas (implementações próprias e com TensorFlow/Keras) e avaliação comparativa dos resultados.

Este relatório descreve todas as etapas do processo, desde a recolha de dados até à análise dos modelos desenvolvidos.

2 Construção dos Datasets

O objetivo do nosso projeto é desenvolver um modelo capaz de identificar excertos de texto gerados por inteligência artificial (IA). Para isso, a qualidade e representatividade dos dados utilizados no treino são cruciais.

2.1 Utilização Inicial de Datasets Públicos

No início do projeto, foram utilizados datasets públicos disponibilizados no enunciado, provenientes do HuggingFace e Kaggle. Estes conjuntos de dados permitiram realizar uma primeira fase de experimentação e testes de abordagem. No entanto, de forma a aumentar a diversidade, controlar melhor as características dos textos e tornar o modelo mais robusto, optou-se por criar datasets personalizados.

2.2 Geração de Dados Humanos com a Wikipedia

Para a componente de texto humano, desenvolvemos um script que recorre à API da Wikipedia para extrair artigos de diversas áreas científicas. Através de uma lista de palavras-chave (por exemplo, *Physics*, *Genetics*, *Artificial Intelligence*), foi utilizada a função `wikipedia.search()` para obter tópicos relevantes, e, em seguida, extraiu-se o conteúdo completo das respetivas páginas.

2.3 Geração de Dados por IA com LLMs

Para a geração de textos de IA, foram utilizados modelos de linguagem de última geração, nomeadamente o *Gemini* (via API do Google Generative AI) e o *ChatGPT*. Os textos foram gerados com base em *prompts* cuidadosamente desenhados para simular o estilo, o tamanho e os temas dos textos humanos recolhidos da Wikipedia.

2.4 Tentativas de Expansão: Reddit e News API

Para enriquecer ainda mais o dataset humano, foram exploradas também outras fontes como o Reddit e a News API. Através da Reddit API (`PRAW`), procuraram-se comentários relevantes em subreddits científicos. Já com a News API, foram extraídos excertos de artigos jornalísticos sobre ciência.

No entanto, ambas as abordagens apresentaram limitações significativas. A Reddit API devolvia frequentemente comentários curtos, filtrados ou irrelevantes. Já a News API impunha limites de requisições e muitas vezes os textos estavam incompletos. Como resultado, a quantidade de dados obtida por estas fontes foi insuficiente para integrar o dataset final.

2.5 Resultado Final

O dataset final utilizado para treino e validação consistiu, assim, em excertos extraídos da Wikipedia (classificados como `Human`) e textos gerados por LLMs (classificados como `AI`). Esta abordagem permitiu construir um conjunto de dados com equilíbrio em tamanho, temática e estilo, garantindo uma base robusta para a tarefa de classificação proposta.

2.6 Limpeza e Pré-processamento

Antes de treinar os modelos, foi necessário aplicar um conjunto de técnicas de limpeza e pré-processamento aos textos. Foram removidos elementos irrelevantes como referências bibliográficas, cabeçalhos da Wikipedia, símbolos matemáticos e caracteres especiais.

Em seguida, os textos foram normalizados, corrigindo quebras de linha, espaços em excesso e garantindo uma estrutura gramatical uniforme. Para assegurar a coerência entre os exemplos e reduzir a complexidade computacional, os textos foram segmentados em excertos com um número de palavras entre 80 e 200.

Por fim, foi feita uma validação dos dados, eliminando textos vazios ou sem estrutura linguística, e, no caso dos textos gerados por IA, foram corrigidas automaticamente respostas truncadas utilizando modelos de linguagem.

2.7 Estatísticas do Dataset Final

O dataset final utilizado neste projeto contém um total de 10 000 excertos de texto. Cada exemplo foi rotulado como pertencente a uma de duas categorias:

- **Human** — textos escritos por autores humanos, extraídos de fontes como a Wikipedia;
- **AI** — textos gerados por modelos de linguagem artificial (como ChatGPT ou Gemini).

Os textos foram selecionados e ajustados de forma a manter características semelhantes entre classes, nomeadamente ao nível do tamanho e da temática. A tabela seguinte resume as principais estatísticas do conjunto de dados final:

Descrição	**Valor**
Número total de exemplos	10 000
Número de exemplos (Human)	5 000
Número de exemplos (AI)	5 000
Comprimento médio dos textos	152.66 palavras
Comprimento mínimo	64 palavras
Comprimento máximo	246 palavras

Tabela 1: Resumo estatístico do dataset final

3 Modelos com Implementação Própria (NumPy)

3.1 Arquitetura dos Modelos

Nesta fase do trabalho foram desenvolvidos três tipos principais de modelos, implementados exclusivamente com recurso à biblioteca NumPy, sem recorrer a frameworks de machine learning como Scikit-learn, TensorFlow ou PyTorch. Os modelos criados foram:

- **Regressão Logística**: modelo base (baseline), com uma única camada linear e ativação sigmoide.
- **Redes Neuronais Profundas (DNNs)**: redes compostas por várias camadas densas, ativação não linear (ReLU e Sigmoid), camadas de Dropout e otimizadores com momentum e regularização.
- **Redes Neuronais Recorrentes Simples (RNNs)**: arquitetura sequencial com uma camada recorrente baseada em tanh, permitindo aprender padrões temporais em sequências de texto.

3.2 Detalhes de Implementação

A implementação foi modular e orientada a objetos, garantindo reutilização e flexibilidade na definição das redes. Os principais componentes são:

- **Camadas**: Todas as camadas herdam de uma classe abstrata `Layer`. Foram implementadas `DenseLayer`, `DropoutLayer` e `RecurrentLayer`, com suporte para inicialização de pesos, forward e backward propagation.
- **Funções de ativação**: As funções `ReLU` e `Sigmoid` foram implementadas como camadas próprias, herdando de uma camada base `ActivationLayer`.
- **Funções de perda**: Foram utilizadas duas funções principais: `MeanSquaredError` e `BinaryCrossEntropy`, com as respetivas derivadas para retropropagação.
- **Otimizador**: Foi criado um otimizador com suporte a **momentum** e **regularização L2 (weight decay)**. Os pesos são atualizados com base nos gradientes acumulados.
- **Métricas**: Foram incluídas funções de `accuracy`, `mse` e `perplexity`, adaptadas para previsões contínuas e classificações binárias.

3.3 Configuração do Treino

A configuração do treino variou ligeiramente entre os modelos, mas de forma geral foram usados os seguintes parâmetros:

- **Epochs**: 100
- **Batch size**: 16
- **Train/validation split**: 70% / 30% (DNN) e 80% / 20% (RNN)
- **Optimizer**: SGD with learning rate $= 0.005$, momentum $= 0.9$ and weight decay $= 10^{-5}$
- **Loss function**: `BinaryCrossEntropy`
- **Evaluation metric**: `accuracy`
- **Early stopping** (only for RNN): patience of 5 epochs without improvement

3.4 Avaliação dos Modelos

Durante o treino, os modelos registaram métricas de desempenho por época, nomeadamente **loss** e **accuracy**, permitindo monitorizar o progresso e identificar sinais de overfitting.

Redes Densas (DNN): A DNN implementada possuía duas camadas ocultas com 32 e 16 unidades, seguidas por Dropout (30% e 40% respetivamente). Esta arquitetura alcançou bons resultados no conjunto de validação, com precisão superior a 90%.

Redes Recorrentes (RNN): A RNN foi treinada sobre sequências de texto transformadas via Bag-of-Words, agrupadas em janelas com comprimento fixo.

A arquitetura consistiu em uma `RecurrentLayer` com 32 unidades, seguida de camadas densas e Dropout. A utilização de early stopping permitiu melhorar a capacidade de generalização.

Adicionalmente, a saída da RNN foi usada como entrada para uma regressão logística, com o objetivo de avaliar a separabilidade linear dos vetores gerados pela rede.

4 Modelos com TensorFlow/Keras

Neste capítulo, abordamos os modelos desenvolvidos com recurso às bibliotecas *TensorFlow/Keras*. Para cada modelo foi seguida uma abordagem específica, com variação dos hiperparâmetros e aplicação de diferentes estratégias de representação textual e de processamento sequencial. O objetivo foi extrair o melhor desempenho possível de cada modelo na tarefa de distinguir entre texto gerado por Inteligência Artificial e texto escrito por um ser humano.

4.1 Modelos Desenvolvidos

Foram desenvolvidos e avaliados os seguintes modelos com TensorFlow/Keras, cada um com uma estratégia própria de representação e arquitetura.

1. DNN (Rede Neural Profunda)

Foi desenvolvida uma rede neural densa com várias camadas totalmente conectadas, aplicada sobre vetores *Bag-of-Words* gerados com `CountVectorizer`.

- **Arquitetura:**
  - 3 camadas `Dense` (128, 64, 32 neurónios) com `LeakyReLU`, `BatchNormalization`, regularização L2 ($\lambda = 0{,}001$) e `Dropout` (0.5)
  - Camada de saída: 1 neurónio com ativação `sigmoid`
- **Hiperparâmetros:**
  - Épocas: 100 Batch size: 32 Otimizador: Adam (lr=0.0001)
  - Função de perda: Binary Crossentropy
  - Callbacks: `EarlyStopping` e `ReduceLROnPlateau`

2. CNN (Rede Neural Convolucional)

Este modelo utiliza uma rede convolucional 1D simples para extrair padrões locais nas sequências de palavras, funcionando como um classificador de texto.

- **Representação textual:** Sequências tokenizadas e convertidas em embeddings treinados de raiz.
- **Arquitetura:**
  - Camada de Embedding
  - Camada Conv1D com 128 filtros e *kernel size* 5

- Camada GlobalMaxPooling
- Camada densa com 64 neurónios e função de ativação ReLU
- Camada de saída com 1 neurónio e ativação *sigmoid*

- **Hiperparâmetros:**
  - Épocas: 10
  - Tamanho do batch: 64
  - Otimizador: Adam
  - Taxa de aprendizagem: 0.001
  - Função de perda: Binary Crossentropy
  - Dropout: 0.5

3. LSTM (Long Short-Term Memory)

Foi implementado um modelo sequencial com duas camadas LSTM empilhadas para capturar padrões temporais em sequências de texto, utilizando embeddings treinados de raiz.

- **Arquitetura:**
  - Embedding (`input_dim`=30 000, `output_dim`=32)
  - LSTM (16 unidades, `return_sequences=True`), regularização L2 e `recurrent_dropout`=0.5
  - LSTM (8 unidades), regularização L2 e `recurrent_dropout`=0.5
  - `BatchNormalization`, seguido de `Dense`(16, ReLU) com L2 e `Dropout`(0.7)
  - Camada de saída: `Dense`(1, sigmoid)

- **Hiperparâmetros:**
  - Épocas: 20 Batch size: 32 Otimizador: Adam (lr=0.0001, `clipnorm`=1.0)
  - Função de perda: Binary Crossentropy
  - Callbacks: `EarlyStopping` e `ReduceLROnPlateau`

4. GRU (Gated Recurrent Unit)

Este modelo utiliza uma camada GRU para modelar sequências de texto, com forte regularização e normalização para mitigar *overfitting*.

- **Arquitetura:**
  - Embedding (`input_dim`=25 000, `output_dim`=32, `input_length`=100)
  - GRU (8 unidades) com `dropout`=0.6, `recurrent_dropout`=0.6, regularização L2 em pesos e recorrência
  - `BatchNormalization`
  - Saída: `Dense`(1, sigmoid) com L2

- **Hiperparâmetros:**
  - Épocas: 25 Batch size: 64 Otimizador: Adam (lr=0.0001, `clipnorm`=1.0)
  - Função de perda: Binary Crossentropy
  - Callbacks: `EarlyStopping` (patience=3), `ReduceLROnPlateau`

5. BERT + LSTM/CNN (Modelo Híbrido)

Usámos a versão `bert-base-uncased` da biblioteca HuggingFace para gerar embeddings contextuais de cada texto. Estes embeddings foram então processados por modelos simples (LSTM ou CNN) para a tarefa de classificação.

- **Representação textual:** Embeddings BERT
- **Arquiteturas testadas:**
  - BERT + LSTM (64 unidades)
  - BERT + CNN (Conv1D com 128 filtros)
- **Hiperparâmetros:**
  - Épocas: 5
  - Batch size: 16
  - Learning rate: 2e-5 (com scheduler)
  - Otimizador: AdamW

4.2 Metodologia de Treino

Todos os modelos seguiram um pipeline comum com pequenas variações conforme a arquitetura.

Pré-processamento: Os textos foram limpos, tokenizados e convertidos para minúsculas. Para o modelo DNN foi usada a representação *Bag-of-Words* com `CountVectorizer`. Nos modelos sequenciais (LSTM, GRU, CNN), os textos foram transformados em sequências inteiras e normalizados com `pad_sequences`.

Divisão de dados: Os dados foram divididos em treino (70%), validação (15%) e teste (15%), garantindo equilíbrio de classes.

Parâmetros comuns: Foi usada a função de perda `BinaryCrossentropy`, otimizador `Adam` (lr entre 0.0001 e 0.001), com `batch size` de 32 ou 64 e número de épocas entre 10 e 100.

Regularização: Recorremos a técnicas como `Dropout`, regularização L2, `BatchNormalization`, `EarlyStopping` e `ReduceLROnPlateau` para prevenir overfitting.

Avaliação: As métricas principais foram `accuracy` e `loss`, avaliadas em conjunto de teste separado.

4.3 Prompt Engineering

Para além do desenvolvimento de modelos supervisionados, explorámos também abordagens baseadas em *Prompt Engineering*, usando uma LLM pré-treinada (Gemini Pro) para distinguir entre textos gerados por IA e textos escritos por humanos. Testámos várias estratégias de construção de *prompts* e analisámos como estas afetavam a *accuracy* da LLM.

4.3.1 Estratégias de Prompt

As seguintes abordagens foram consideradas:

- **Zero-shot:** A estratégia mais simples, onde se pedia diretamente ao modelo para indicar se um texto foi escrito por um humano ou gerado por IA, sem fornecer qualquer exemplo. Foi a abordagem com pior desempenho.
- **One-shot:** Neste caso, forneceu-se um exemplo de cada classe (um texto de IA e um texto humano), já rotulados, antes da pergunta principal. O desempenho foi superior ao *zero-shot*.
- **Few-shot:** Semelhante ao *one-shot*, mas com múltiplos exemplos fornecidos para dar mais contexto ao modelo. O desempenho foi ligeiramente superior, mas muito próximo ao do *one-shot*.
- **RAG (Retrieval-Augmented Generation):** Mantendo a estrutura do *few-shot*, mas com exemplos escolhidos previamente com base em similaridade semântica, usando um *embedding model*. Os exemplos fornecidos eram os mais semelhantes ao texto a classificar, o que deu ao modelo um contexto mais relevante. Esta foi a abordagem que apresentou os melhores resultados.

5 Resultados e Discussão

- Comparação entre os modelos próprios e os com TensorFlow
- Discussão sobre desempenho, generalização, overfitting
- Impacto do tipo de dados e representação textual

Modelo	**Accuracy**
CNN	0.60
DNN	0.48
GRU	0.57
LSTM	0.67
simpleRNN	0.60
Ensemble	0.57

Tabela 2: Comparação da *accuracy* entre diferentes modelos no dataset 3

6 Conclusões

Este projeto permitiu explorar diferentes abordagens para a tarefa de deteção de textos gerados por inteligência artificial. Começámos pela construção de um dataset equilibrado e representativo, recorrendo a textos humanos extraídos da Wikipedia e a textos gerados por LLMs como o ChatGPT e Gemini.

Os melhores resultados foram obtidos com modelos sequenciais, em especial a LSTM, que alcançou uma accuracy de 67% no conjunto de teste, superando arquiteturas mais simples como a DNN (48%) ou CNN (60%). Estes valores

demonstram que, embora existam padrões identificáveis entre texto humano e IA, a tarefa continua desafiadora, sobretudo com o crescente realismo dos modelos generativos.

Adicionalmente, explorámos a técnica de *Prompt Engineering* como abordagem alternativa, testando estratégias zero-shot, one-shot, few-shot e RAG. A última revelou-se a mais eficaz, graças à personalização do contexto fornecido à LLM.

Vimos que a parte do projeto onde passamos mais tempo e onde no final também se verificou ser aquilo que trouxe melhores resultados foi a criação de um bom dataset para treinarmos os nosso modelos.

Como trabalho futuro, seria interessante expandir o dataset com textos provenientes de diferentes fontes e domínios, explorar fine-tuning de modelos pré-treinados e aplicar técnicas de explicabilidade para compreender melhor os critérios utilizados pelos modelos na distinção entre textos de origem humana e artificial.