Universidade da Beira Interior
Departamento de Informática

Nº 1 - 2024: *Quality and Emotion Visualization in Text*

Elaborado por:

Luís Alves a46107

Orientador:

Professor/a Doutor/a João Cordeiro

8 de julho de 2024

Agradecimentos

A realização deste projeto final de licenciatura em Engenharia Informática foi um desafio enriquecedor, que só foi possível com o apoio e colaboração de diversas pessoas e instituições. Gostaria de expressar a minha sincera gratidão a todos aqueles que, de alguma forma, contribuíram para a concretização deste trabalho.

Primeiramente, gostaria de agradecer ao meu orientador, João Cordeiro, cujo conhecimento, paciência e orientação foram fundamentais ao longo de todo o processo. As suas valiosas sugestões e críticas construtivas foram imprescindíveis para a evolução e melhoria deste projeto.

Agradeço também aos professores do Departamento de Engenharia Informática da UBI, por todo o conhecimento transmitido ao longo do curso, que formou a base indispensável para a execução deste trabalho.

Um agradecimento especial aos meus colegas e amigos, que estiveram ao meu lado nos momentos de dificuldade e celebração, oferecendo sempre apoio e encorajamento. A partilha de ideias e experiências com vocês foi essencial para o desenvolvimento deste projeto.

Não poderia deixar de agradecer à minha família, pelo suporte incondicional e por acreditarem sempre nas minhas capacidades. O vosso apoio emocional foi crucial para que eu pudesse dedicar-me inteiramente a este trabalho.

A todos, o meu muito obrigado.

Luís Alves

Conteúdo

Conteúdo **iii**

Lista de Figuras **v**

Lista de Tabelas **vii**

1 Introdução **1**
1.1 Enquadramento 1
1.2 Motivação 1
1.3 Objetivos 3
1.4 Organização do Documento 4

2 Estado da Arte **5**
2.1 Introdução 5
2.2 Contextualização 5
2.3 Soluções de *softwares* semelhantes 7
2.4 Conclusões 8

3 Tecnologias e Ferramentas Utilizadas **9**
3.1 Introdução 9
3.2 Python 9
3.3 SpaCy e NLTK 9
3.4 JavaScript 11
3.5 HTML e CSS 11
3.6 Framework Flask 11
3.7 GitHub 11
3.8 Conclusões 11

4 Trabalho Experimental **13**
4.1 Introdução 13
4.2 Amostras 13
4.3 Algoritmos 14
4.4 Métricas de Leitura 14
4.4.1 Simple Measure of Gobbledygook (SMOG) 14

4.4.2 Coleman–Liau Index 15
4.4.3 Flesch-Kincaid Grade Level 17
4.4.4 Flesch Reading Ease 18
4.4.5 Automated Readability Index (ARI) 19
4.5 Medidas de Complexidade Textual 21
4.5.1 Diversidade Lexical 21
4.5.2 Densidade Lexical 22
4.5.3 Tamanho Médio das Palavras 22
4.5.4 Tamanho Médio das Frases 23
4.5.5 Profundidade Sintática da Frase 24
4.6 Análise de Sentimentos 25
4.7 Heurísticas 26
4.7.1 Qualidade da Estrutura Textual 26
4.7.2 Visualização Sentimental 27
4.8 Modelo XGBoost 28
4.9 WebSite 31
4.10 Conclusões 31

5 Resultados **33**
5.1 Introdução 33
5.2 Visualização de Resultados 33
5.3 Modelo de Treino 35
5.4 Testes 36
5.5 Conclusões 39

6 Conclusões e Trabalho Futuro **41**
6.1 Conclusões Principais 41
6.2 Trabalho Futuro 42

A Anexo **43**
A.1 Funções Auxiliares 43

Bibliografia **51**

Lista de Figuras

4.1 Estrutura visual do site desenvolvido 31

5.1 Importância dos algoritmos usados. 34
5.2 Resultados da matriz de confusão. 34

Lista de Tabelas

4.1 Resultados do algoritmo SMOG. 15
4.2 Resultados do algoritmo *Coleman-Liau*. 17
4.3 Resultados do algoritmo *Flesch-Kincaid Grade Level*. 18
4.4 Resultados do algoritmo *Flesch Reading Ease*. 19
4.5 Resultados do algoritmo ARI. 21

5.1 Tabela de Sentimentos. 38
5.2 Tabela de Estrutura. 38

Lista de Excertos de Código

4.1 Trecho de código Smog. 14
4.2 Trecho de código Coleman. 16
4.3 Trecho de código Flesch-Kincaid Grade Level. 17
4.4 Trecho de código Flesch Reading Ease. 18
4.5 Trecho de código ARI. 20
4.6 Trecho de código Diversidade Lexical. 21
4.7 Trecho de código Densidade Lexical. 22
4.8 Trecho de código Tamanho Médio das Palavras. 23
4.9 Trecho de código Tamanho Médio das Frases. 23
4.10 Trecho de código Árvore Estrutural. 24
4.11 Trecho de código Análise Sentimental. 25
4.12 Trecho de código Heurísticas Estruturais Versão Final. 27
4.13 Trecho de código Heurísticas Sentimentais Versão Final. 28
4.14 Trecho de código para treino da IA. 29
5.1 Código responsável pelo número de erros gramaticais e ortográficos e possíveis correções. 37
5.2 Código usado para correção do erro no módulo csv. 37
5.3 Código usado para correção do erro do SpaCy. 37
A.1 Função utilizada para retirar amostras. 43
A.2 Função utilizada para calcular os algoritmos nas amostras retiradas. 44
A.3 Função utilizada para guardar as amostras juntamente com os resultados dos algoritmos. 45
A.4 Função utilizada para a aplicação das heurísticas. 47

Acrónimos

ARI Automated Readability Index
CSS Cascading Style Sheets
CSV Comma-Separated Values File
HTML HyperText Markup Language
IA Inteligência Artificial
PLN Processamento de Linguagem Natural
RAM Random Access Memory
SMOG Simple Measure of Gobbledygook
UBI Universidade da Beira Interior

Capítulo

1

Introdução

1.1 Enquadramento

A qualidade de um texto é composta por vários fatores, alguns explícitos e de cálculo fácil, como a riqueza de vocabulário, a complexidade de uma frase, ou o nível de subjetividade emocional de um texto. Outros são mais implícitos e podem envolver o estilo de escrita, coesão semântica e coerência, ou até mesmo arranjos lexicais específicos e expressões do próprio idioma que são menos evidentes numa avaliação superficial.

Este trabalho consiste em implementar um modelo de IA, que permite medir, caracterizar e visualizar a qualidade e a complexidade de um texto, como também as suas emoções. Este projeto também pretende avaliar a importância dos diferentes algoritmos existentes para a avaliação da qualidade textual, como também para avaliar a visualização sentimental. Na interação com o utilizador será desenvolvido uma aplicação *web* com diferentes funcionalidades úteis, para melhor entendimento deste tipo de tecnologia.

1.2 Motivação

Atualmente, a IA está muito presente no quotidiano de qualquer pessoa, sendo desta forma importante entender o seu funcionamento e desenvolvimento. Esta foi uma das grandes motivações para o desenvolvimento deste projeto, bem como, o aumento do impacto da área de processamento de linguagem natural nas tarefas feitas diariamente, sendo assim, necessário fazer progressos e novas pesquisas nessa área para um melhor aperfeiçoamento e desenvolvimento dos materiais entregues à comunidade.

Desta maneira, é relevante a qualidade e importância dos algoritmos usados, pois este ramo possibilita o desenvolvimento de tecnologias que apoiam muito a sociedade de hoje, o que motivou este projeto.

Texto cotado com "Estrutura Muito Complexa":

> In the private confines of the opulent penthouse, a profound sense of timelessness pervades as individuals immerse themselves in an immersive embrace of fervent ardor. Each pulsation echoes an orchestrated cadence of desire, resonating with a palpable intensity. Through tactile engagement, they traverse the intricate contours of their mutual affinity, delicately delineating the intricacies of their emotional landscape.

Texto cotado com "Estrutura Complexa":

> Adamson, a flight controller at JSC when selected, called New York state home. He received his first spaceflight assignment in January 1986 as a mission specialist on STS-61N, along with fellow Maggots McCulley and Brown, a Department of Defense mission aboard Columbia then planned for September 1986. The January 1986 Challenger accident resulted in the suspension of all flight and crew assignments.

Texto cotado com "Estrutura Pouco Complexa":

> “What a mess you are!” her two stepsisters laughed. That is why they called her “Cinderella.”
>
> One day, big news was announced in their village. It was time for the Prince to find a bride, and the King and Queen were going to have a ball! All of the young ladies of the land were invited to come. The stepsisters were wild with joy. They would wear their most beautiful gowns and fix their hair in the most splendid way. No doubt they would be the one to win the favor of the Prince!
>
> Cinderella now had extra work to do. She had to sew two fabulous gowns for her step-sisters in the latest fashion.

Texto cotado com "Estrutura Muito Simples":

> ONCE ON A FARM LONG AGO, a Mama Duck sat on her nest. “How long must I wait for my babies to hatch? I have to sit here all by myself!” But what could she do? A Mama duck must keep her eggs warm till they hatch.

> At last, the eggs began to crack. One yellow duckling stepped out of its shell, then another. Each little chick shook its wings. “Quack, quack!”

Tendo em conta os textos aqui apresentados, é importante ter em consideração a necessidade destas avaliações, pois notícias, pesquisas ou artigos necessitam de uma linguagem que abranja maior parte da sociedade, sendo uma boa cotação para esses tipos de textos "Estrutura Complexa"ou "Estrutura Pouco Complexa". Para textos dedicados ao público mais juvenil, é necessário um texto mais simples e claro, portanto uma boa cotação para esses textos seria "Estrutura Muito Simples"ou "Estrutura Pouco Complexa". Para textos escritos profissionalmente a estrutura textual é complexa e com uma gramática e ortografia mais robusta, pelo que é necessário uma atenção maior para seu entendimento, posto isto, seria avaliado com "Estrutura Muito Complexa".

Nestes textos também é necessário uma reflexão sentimental, pois textos na categoria de artigos e pesquisas tendem a ter um sentimento neutro, pois a sua escrita é direta e objetiva. Por outro lado em textos onde há uma história ou o desenvolvimento mais subjetivo tendem a ter um sentimento positivo ou negativo.

1.3 Objetivos

O projeto tem como objetivo principal a criação de um modelo inteligente com capacidade de prever e avaliar a qualidade textual e a visualização sentimental num texto, existindo vários objetivos subjacentes ao principal, como:

1. Agrupamento de diferentes textos, isto é, textos de diferentes qualidade e diferentes sentimentos, para a realização de análises e cálculos com os algoritmos desenvolvidos;

2. Criação de base de dados de diferentes idiomas, para ser possível a criação de um modelo inteligente capaz de analisar várias linguagens;

3. Desenvolvimento de diversos algoritmos, tanto para o cálculo da qualidade estrutural do texto, como para a previsão sentimental do mesmo;

4. Exploração de heurísticas para o modelo realizar uma previsão com boa taxa de acerto;

5. Realização de um *website* onde é possível observar os resultados obtidos do modelo de IA e interagir com o mesmo.

1.4 Organização do Documento

De modo a refletir o trabalho feito, este documento encontra-se estruturado da seguinte forma:

1. O primeiro capítulo – **Introdução** – apresenta o projeto, a motivação para a sua escolha, o enquadramento para o mesmo, os seus objetivos e a respetiva organização do documento.

2. O segundo capítulo – **Estado da Arte** – onde será feita uma análise detalhada e abrangente das pesquisas, descobertas e avanços mais recentes relacionados ao tema abordado neste relatório.

3. O terceiro capítulo – **Tecnologias e Ferramentas Utilizadas** – descreve os conceitos mais importantes no âmbito deste projeto, bem como as tecnologias utilizadas durante o desenvolvimento da aplicação.

4. O quarto capítulo – **Trabalho Experimental** – irá ser feita uma abordagem detalhada da implementação do código utilizado neste projeto.

5. O quinto capítulo – **Resultados** – onde é descrito os resultados obtidos ao longo do desenvolvimento do projeto, bem como, problemas e soluções encontrados.

6. O sexto capítulo – **Conclusão e Trabalho Futuro** – onde irá ser apresentada uma conclusão para o relatório assim como possíveis melhorias/-funcionalidades que poderiam ser implementadas em trabalhos futuros.

Capítulo

2

Estado da Arte

2.1 Introdução

Neste capítulo será abordada a área onde o trabalho desenvolvido está inserido, assim como, pesquisas realizadas no processamento de linguagem natural. Serão ainda descrito *softwares* semelhantes ao desenvolvido e como estes impactaram o desenvolvimento da IA e o quotidiano da sociedade atual.

2.2 Contextualização

No desenvolvimento deste projeto foi explorada uma grande área científica, a área de inteligência artificial, mais precisamente, processamento de linguagem natural, isto é, conseguir retirar métricas, contexto, compreensão semântica e sentimentos intrínsecos a uma certa palavra ou trecho de texto. Para um melhor entendimento dessa área descrita, foram analisados diversos artigos e trabalhos desenvolvidos, que serão expostos nesta secção.

Estes projetos e pesquisas feitas para o respetivo domínio é importante, pois permite compreender o desenvolvimento e evolução que esta área tem nos dias atuais, bem como o seu impacto e ajuda na atualidade.

De toda a literatura explorada, é de referir um artigo em específico, no qual se estudou a possível relação entre a repetição criada nos fractais com a similaridade das frases entre si. Neste artigo, desenvolvido pelos professores doutores João Cordeiro, Pedro Inácio e Diogo Fernandes, foi possível concluir que existe uma beleza fractal nos textos desenvolvidos, isto é, que a qualidade é diretamente proporcional ao grau de similaridade. [1]

O artigo citado apoiou no desenvolvimento do projeto proposto, pois ajudou a perceber melhor que no texto, como um todo, há uma similaridade na

sua formação, sendo essa informação transferida para a formação estrutural do texto que será analisado.

Ainda dentro da avaliação estrutural de um texto, foi ainda explorado o trabalho desenvolvido por Meri Coleman e Ta Lin Liau, onde nesta pesquisa é encontrado uma forma de avaliar os textos, tendo a fórmula:

$$0.0588 * (characters/words) - 0.296 * (sentences/words) - 15.8 \quad (2.1)$$

Com esta fórmula foi possível ditar um intervalo, [0, 11], que pode ser traduzido para o sistema de ensino americano. [2]

A pesquisa desenvolvida ajudou no objetivo de entender o funcionamento de diversos outros algoritmos com a mesma interpretação, isto é, a partir da avaliação da estrutura do texto conseguir ditar um intervalo que define a qualidade desse texto. Para além desse objetivo, este artigo permite concluir que definindo um intervalo, textos com uma pior estrutura tendem a encontrar-se com valores menores do que aqueles que apresentam uma melhor estrutura.

Ainda dentro do processamento de linguagem natural, temos o ramo de análise de sentimentos, onde é possível identificar o sentimento intrínseco de um texto ou partes desse mesmo texto.

A pesquisa realizada dentro desta área de inteligência artificial foi importante, pois é possível entender que nos dias atuais há um elevado fluxo de informação a circular, sendo necessário formas consistentes de filtrar essa informação. Para esta finalidade, é possível analisar o ramo estrutural do texto ou até o ramo sentimental, pois assim, dependendo do tipo de informação que é procurada é necessário essa informação ser precisa, consistente e confiável.

Para a avaliação dessas características o ramo de processamento de linguagem natural é de extrema importância importante, porque traz ferramentas desenvolvidas e aprimoradas ao longo de décadas para auxiliar no filtro do que é informação pertinente.

No que diz respeito à análise sentimental, esta tem um papel fundamental inerente, visto que uma forma comum de atingir um leitor, ou atrair atenção sobre um texto ou informação é apelar ao lado sentimental, dado que este é impactante no fator psicológico humano. Porém, notícias, pesquisas ou artigos, não apresentam qualquer ênfase sentimental, pois sendo descobertas ou avanços tecnológicos o interesse está relacionado com o projeto desenvolvido.

Para um melhor entendimento sobre a forma como os sentimentos são demonstrados no texto, foi estudado o artigo desenvolvido por Walaa Medhat, Ahmed Hassan e Hoda Korashy, no qual são abrangidos vários métodos de análise sentimental. Assim, alguns dos métodos referidos são utilizados pela

tecnologia escolhida, sendo usado, por exemplo, a visualização do léxico onde a palavra se insere, atribuindo um valor sentimental. [3]

Posto isto, o projeto desenvolvido encontra-se inserido na área de processamento de linguagem natural onde através de diversos mecanismos, auxilia na filtragem do que pode ser um texto bem estruturado. O trabalho realizado também ajuda na identificação sentimental, onde é possível entender se o texto produzido contém ou não sentimentos visualmente representados, o que pode ser um fator crucial na forma como o texto é recebido e partilhado entre a comunidade.

2.3 Soluções de *softwares* semelhantes

Nesta secção serão mostradas aplicações/*softwares* semelhantes ao elaborado, como também, será dada uma explicação sobre as aplicações apresentadas.

- **Grammarly** [4]: É uma IA desenvolvida para verificação de erros gramaticais, de ortografia, pontuação, escolha de palavras e estilo de escrita. Este *software* também indica possíveis problemas textuais e correções de contexto para os erros referidos a cima, gerando assim, um texto com uma qualidade textual elevada.
  Enquanto o *Grammarly* se foca mais em correção de erros e, com isso, gerar um texto de qualidade decente, o projeto desenvolvido foca-se mais em avaliar o texto apresentado sem causar alterações no mesmo, indicando também o sentimento principal desse texto, função que o *Grammarly* não apresenta.

- **Hemingway** [5]: É uma IA focada em legibilidade e estilo de escrita de um texto. Para além disso, a avaliação do texto é feita através de uma cotação, que quanto mais elevada mais bem escrito o texto se encontra. Esta aplicação encontra semelhanças com o projeto feito, pois ambas as IA usam métricas de leitura para a análise textual, sendo também, dada um cotação ao texto avaliado. Mas enquanto a *Hemingway* disponibiliza possíveis correções, o modelo de IA projetado avalia o texto estruturalmente e sentimentalmente, sendo que, a última função não se encontra disponível no *Hemingway*.

- **ProWritingAid** [6]: É uma IA que tem como objetivo avaliar o texto analisado através de relatórios sobre gramática, estilo de escrita, legibilidade, repetições de palavras, etc. Assim, é possível obter um relatório sobre a qualidade textual e possíveis alterações a ser feitas no texto.

A aplicação apresentada tem semelhanças com o projeto feito, pois ambas avaliam o texto tendo em conta a qualidade textual e a sua complexidade e fluência. Tendo a *ProWritingAid* a função de possíveis soluções a erros encontrados, enquanto que, o projeto desenvolvido tem a ferramenta de calcular o sentimento do texto.

- **Natural Language AI** [7]: É uma *Application Programming Interface* desenvolvida pela *Google Cloud* que providência uma IA para análise de sentimentos e reconhecimentos de entidades, sendo semelhante ao projeto desenvolvido, onde ambas analisam e demonstram o sentimento do texto. Apesar desta função, o trabalho desenvolvido ainda disponibiliza uma ferramenta, onde é possível prever a qualidade textual.

- **Lexalytics** [8]: Esta plataforma fornece análise de sentimentos, reconhecimento de entidades e o tema que o texto representa, sendo parecido ao trabalho feito no sentido de avaliar o texto sentimentalmente, onde é dado um resultado visual do sentimento do texto.

- **IBM Watson Natural Language Understanding** [9]: A IA oferece a capacidade de compreensão textual, como, análise de sentimentos, análise emocional e opções de visualização. Em comparação com o projeto desenvolvido ambas oferecem uma forma de análise de sentimentos.

2.4 Conclusões

Concluindo, neste capítulo foi explicada a área onde o projeto se encontra inserido, como também, foram citados artigos e pesquisas realizadas na área, que foram usados como suporte para o desenvolvimento do trabalho realizado.

Aqui, também foi explicado como o projeto se insere nesta área de pesquisa e também o que este projeto oferece tendo em conta ferramentas atualmente desenvolvidas e utilizadas.

Capítulo

3

Tecnologias e Ferramentas Utilizadas

3.1 Introdução

Neste capítulo serão abordadas todas as ferramentas e tecnologias utilizadas, como linguagens de programação, *framework* e plataformas de armazenamento de ficheiros. Assim, serão explicadas todas as funcionalidades e utilizações onde estas ferramentas estão inseridas.

3.2 Python

Para o desenvolvimento deste projeto, foi utilizada a linguagem de programação *Python*. Esta escolha foi feita devido à grande utilização desta linguagem na área de inteligência artificial, como também, à sua facilidade e melhor adequação para o trabalho desenvolvido.

Juntamente com esta linguagem, foram utilizadas diversas bibliotecas, sendo as mais relevantes para o desenvolvimento deste projeto, *NLTK* [10], *SpaCy* [11], *XGBoost* [12], *scikit-learn* [13] e *flask* [14].

3.3 SpaCy e NLTK

Sendo a base deste projeto o processamento de linguagem natural foram selecionadas duas grandes bibliotecas dessa área, sendo elas *NLTK* e *SpaCy*. Nesta secção será descrito funcionalidades disponíveis, bem como, diferenças entre estas bibliotecas.

As características do *SpaCy* são o seu desempenho e eficiência, pois este módulo foi desenvolvido para o processamento de grandes volumes de texto de forma rápida. A inclusão de modelos pré-treinados para diferentes idiomas que suportam tarefas como, reconhecimento de entidades e análise de dependências sintáticas. E a integração com *deep learning*, isto é, uma maior facilidade de integração de bibliotecas de *deep learning*, permitindo assim a extensão das suas capacidades com redes neuronais personalizadas.

O *SpaCy* também tem como principais funcionalidades:

- *Tokenização*: Segmentação de texto em palavras, pontuação, e outros elementos.
- *POS Tagging*: Atribuição de etiquetas gramaticais às palavras.
- *NER*: Identificação e classificação de entidades mencionadas no texto (pessoas, organizações, locais, etc.).
- Análise de Dependências: Estruturação das relações sintáticas entre palavras.
- Vetores de Palavras: Representação de palavras em vetores numéricos para cálculos semânticos.

As características do *NLTK* são a amplitude de ferramentas, que inclui uma vasta coleção de corpora, algoritmos de processamento de linguagem natural e ferramentas de visualização. Flexibilidade e extensibilidade que permite uma personalização e extensão de maneira fácil, sendo adequada para experimentações e desenvolvimento de novos algoritmos de PLN.

O módulo *NLTK* apresenta como principais funcionalidades:

- *Tokenização*: Diversos métodos de segmentação de texto em *tokens*.
- *POS Tagging*: Etiquetagem gramatical das palavras.
- *Stemming* e *Lematização*: Redução das palavras à sua forma raiz.
- Corpora e *Lexicons*: Acesso a vários conjuntos de dados textuais e léxicos.
- *Parsing*: Análise sintática de frases com diversas abordagens.

3.4 JavaScript

Linguagem utilizada para o apoio no *"client side"*, tendo como principal objetivo lidar com interações dinâmicas dentro do *website*, juntamente com o *Python*.

3.5 HTML e CSS

Sendo necessário desenvolver um *website* para a interação com o modelo de IA desenvolvida foi utilizado *HyperText Markup Language* (*HTML*) juntamente com *Cascading Style Sheets* (*CSS*) para desenvolver a parte visual do *client side*, onde interações dinâmicas são tratadas com *JavaScript* e *Python*, como referido a cima.

3.6 Framework Flask

Pensando na interação com o utilizador foi usado o *flask*, uma *framework* para criação de *websites*, onde é possível conectar *Python* com *HTML*. A partir desta *framework* foi desenvolvido o *client side*, onde é possível interagir com o modelo inteligente desenvolvido.

3.7 GitHub

Na partilha e armazenamento do projeto foi usada a tecnologia *GitHub*, uma plataforma de armazenamento de código-fonte e arquivos. Assim, com esta plataforma foi possível haver um melhor gerênciamento e organização, tanto das versões usadas do projeto, como também no registo e melhoramentos necessários ao longo do projeto.

3.8 Conclusões

Em suma, neste capítulo foi explicado de forma sucinta todas as tecnologias e ferramentas utilizadas, tendo como objetivo tornar o trabalho desenvolvido mais organizado, prático e eficiente.

Capítulo

4

Trabalho Experimental

4.1 Introdução

Neste capítulo será abordado de forma mais técnica e pormenorizada, a implementação realizada neste projeto. Este capítulo descreve, tanto os algoritmos desenvolvidos para a classificação estrutural do texto, como também a visualização sentimental. Também será abordado todo o procedimento até a fase final da criação do modelo de IA.

4.2 Amostras

Na realização deste projeto, foi decidido o uso de amostras do texto a ser analisado. Esta escolha deve-se ao facto que a análise de frações do texto é mais rápida e eficiente, é de referir, que o resultado utilizado do texto completo, a margem de erro na análise das amostras é de apenas 1%-10%. Estes valores foram calculados através da comparação dos algoritmos desenvolvidos com amostras e com o texto na sua totalidade.

Os algoritmos criados usam como argumento frases onde o contexto das mesmas não importa. As amostras retiradas do texto são 60 frases aleatórias, em que o seu tamanho tem de ser maior do que quatro palavras, pois para haver uma estrutura frásica válida para esta análise é necessário no mínimo quatro a cinco palavras. Foram usadas 60 frases porque a média normal de um parágrafo bem construido é de três a quatro frases, dando então um total de 12 a 20 parágrafos retirados do texto.

4.3 Algoritmos

Na análise textual e sentimental do texto foram usados vários algoritmos, que podem ser divididos em três categorias: Análise de sentimentos, medidas de complexidade textual e métricas de leitura. A categoria de análise de sentimentos diz respeito às métricas responsáveis pelo cálculo do valor sentimental inerente a cada frase retirada, podendo ser negativo, positivo ou neutro.

Já as métricas de leitura são responsáveis por medir a simplicidade de leitura de um texto, isto é, o quão fácil é a compreensão de um texto, considerando a complexidade das palavras, comprimento da frase e complexidade geral do texto, podendo ser traduzido para fórmulas matemáticas como *Simple Measure of Gobbledygook* [15] ou *Coleman–Liau Index* [16].

Apesar das medidas de complexidade textual terem um objetivo idêntico às métricas de leitura, os algoritmos usados para a determinação da complexidade e facilidade de compreensão e legibilidade são diferentes, podendo ser usado o cálculo da diversidade lexical ou até mesmo a profundidade média da árvore estrutural de um texto.

Os resultados obtidos destas funções para uma melhor precisão e uso da mesma escala foram normalizados, isto é, aplicada a fórmula:

$$x_{norm} = \frac{x - min(X)}{max(X) - min(X)} \tag{4.1}$$

A partir desta função é possível definir o resultado dos algoritmos no intervalo de [0, 1], sendo posteriormente mais fácil definir as heurísticas para o modelo de IA, citado na secção 4.7.

4.4 Métricas de Leitura

4.4.1 Simple Measure of Gobbledygook (SMOG)

Algoritmo responsável pelo cálculo da *SMOG* a partir da fórmula:

$$1.043 * \sqrt{ComplexWords * 30} + 3.1291 \tag{4.2}$$

A fórmula *SMOG* é usada para o cálculo da clareza e legibilidade de um texto, isto é, quanto mais elevado o resultado, mais complexo o texto será. Este cálculo é feito através do número de palavras poli-silábicas.

```
    def SMOGA(Samples):
        Soma = 0
        for Sample in Samples:
            Texto = str(Sample)
```

```
            ComplexWords = sum(1 for word in Texto.
                split() if syllapy.count(word) >= 3)

            Soma += 1.043 * math.sqrt(ComplexWords
                * 30) + 3.1291

        Normalizacao = Normalize(Soma/len(Samples)
            ,12,0)

        return round(Soma/len(Samples),3), round(
            Normalizacao,3)
```

Excerto de Código 4.1: Trecho de código Smog.

Para a aplicação da fórmula do *SMOG* foi utilizado um ciclo *for*, onde para cada frase analisada é contado o número de palavras poli-silábicas, isto é, palavras constituídas por mais de 3 sílabas, sendo esse valor calculado através de uma função disponibilizada pela biblioteca *syllapy* [17]. Após essa análise, é aplicada a fórmula do *SMOG* apresentada anteriormente.

Como retorno é feito a soma total da fórmula em todas as frases a dividir pelo número total de amostras, sendo também devolvido esse mesmo valor normalizado. O resultado dado por esta função pode ser observado na seguinte tabela:

Valor	Nível Escolar	Intervalo de Idades
0-1	1	3-7
1-5	1-6	7-11
5-8	6-9	11-14
8-11	9-12	14-17
>11	>12	>17

Tabela 4.1: Resultados do algoritmo SMOG.

4.4.2 Coleman–Liau Index

Algoritmo responsável pelo cálculo do *Coleman–Liau Index* a partir da fórmula:

$$0.0588 * (characters/words) - 0.296 * (sentences/words) - 15.8 \quad (4.3)$$

A fórmula descrita é usada para calcular o nível de leitura de um texto, ou seja, quanto mais maior for o resultado obtido, mais complexo é o texto e a sua leitura.

```
    def ColemanA(Samples):
        Soma = 0
        for Sample in Samples:
            Texto = str(Sample)

            words = len(Texto.split())

            characters = len(re.sub(r'[^a-zA-Z\s
               ]+|\s+', '', Texto))

            L = (characters/words) * 100
            S = (1/words) * 100
            Soma += 0.0588 * L - 0.296 * S - 15.8

        Normalizacao = Normalize(Soma/len(Samples)
           ,12,0)

        return round(Soma/len(Samples),3), round(
           Normalizacao,3)
```

Excerto de Código 4.2: Trecho de código Coleman.

Para aplicar o *Coleman–Liau Index* foi usado um ciclo *for*, onde em cada frase é contado o número de palavras, assim como, o número de caracteres totais da frase, em seguida é aplicada a fórmula para o cálculo do *Coleman–Liau Index*.

O retorno da função é dado pela soma total da fórmula aplicada a cada frase a dividir pelo número total de amostras do texto, como também esse mesmo valor normalizado. O resultado dado por esta função pode ser observado na seguinte tabela:

Valor	Nível Escolar	Intervalo de Idades
0-1	1	3-7
1-5	1-6	7-11
5-8	6-9	11-14
8-11	9-12	14-17
>11	>12	>17

Tabela 4.2: Resultados do algoritmo *Coleman-Liau*.

4.4.3 Flesch-Kincaid Grade Level

Algoritmo responsável pelo cálculo da *Flesch-Kincaid Grade Level* [18] a partir da fórmula:

$$0.39 * words + 11.8 * (syllables/words) - 15.59 \quad (4.4)$$

Flesch-Kincaid Grade Level avalia o texto dando uma cotação que pode ser traduzida para o sistema de avaliação educativa dos Estados Unidos da América. Na fórmula original, o número de palavras total é dividido pelo número de frases totais, mas como nas amostras analisadas é apenas avaliado uma frase de cada vez, essa divisão foi retirada, pois, em qualquer caso, o número de frases seria sempre um.

```
    def FleschGradeA(Samples):
        Soma = 0
        for Sample in Samples:
            Texto = str(Sample)

            words = len(Texto.split())

            syllables = sum(syllapy.count(word) for
                word in Texto.split())

            Soma += 0.39 * words + 11.8 * (
               syllables/words) - 15.59

        Normalizacao = Normalize(Soma/len(Samples)
           ,18,0)

        return round(Soma/len(Samples),3), round(
           Normalizacao,3)
```

Excerto de Código 4.3: Trecho de código Flesch-Kincaid Grade Level.

Para o desenvolvimento deste algoritmo foi utilizado um ciclo *for*, onde cada frase é separada numa lista de palavras e para cada palavra na lista é registado o número de sílabas, a partir do módulo *syllapy*. Em seguida, é aplicada a fórmula do *Flesch-Kincaid Grade Level*.

Esta função tem como resultado a soma total da fórmula aplicada em todas as frases a dividir pelo número total de amostras, como também esse mesmo valor normalizado. O resultado dado por esta função pode ser observado na seguinte tabela:

Valor	Nível Escolar	Intervalo de Idades
0-3	1-4	5-8
3-6	5-7	8-11
6-9	7-9	11-14
9-12	9-12	14-17
12-15	Faculdade	17-20
15-18	Licenciado	>20

Tabela 4.3: Resultados do algoritmo *Flesch-Kincaid Grade Level*.

4.4.4 Flesch Reading Ease

Algoritmo responsável pelo cálculo da *Flesch Reading Ease* [19] a partir da fórmula:

$$206.835 - 1.015 * words - 84.6 * (syllables/words) \tag{4.5}$$

Flesch Reading Ease faz uma avaliação textual de 0-100, em que valores mais elevados representam um texto mais simples e de fácil compreensão. Na fórmula original o número de palavras totais está a dividir pelo número total de frases na amostra, no entanto nesta versão foi retirado, pois cada frase é avaliada de forma independente, sendo essa divisão feita por um.

```
    def FleschReadingA(Samples):
        Soma = 0
        for Sample in Samples:
            Texto = str(Sample)

            words = len(Texto.split())
```

```
            syllables = sum(syllapy.count(word) for
                word in Texto.split())

            Soma += 206.835 - 1.015 * words - 84.6
               * (syllables/words)

        Soma1 = 100 - (Soma/len(Samples))
        Normalizacao = Normalize(Soma1,100,0)

        return round(Soma1,3), round(Normalizacao
           ,3)
```

Excerto de Código 4.4: Trecho de código Flesch Reading Ease.

No *Flesch Reading Ease* é utilizado um ciclo *for*, onde é contado o número total de palavras da frase, em seguida é analisado o número total de sílabas de cada palavra, para isso é utilizada a biblioteca *syllapy*. Após isso, é aplicada a fórmula para cada uma das frases analisadas. Saindo do ciclo *for* o valor da função é invertido, isto é, um valor baixo passa a avaliar um texto menos complexo, enquanto que, um de valor elevado será um texto mais complexo.

O retorno desta função é a soma total da fórmula aplicada em todas as frases a dividir pelo número total de amostras, como também esse mesmo valor normalizado. O resultado dado por esta função pode ser observado na seguinte tabela:

Valor	Nível Escolar	Intervalo de Idades
100-90	5	11
90-80	6	11-12
80-70	7	12-13
70-60	8-9	13-15
60-50	10-12	15-18
50-30	Faculdade	18-19
30-0	Licenciado	22-23

Tabela 4.4: Resultados do algoritmo *Flesch Reading Ease*.

4.4.5 Automated Readability Index (ARI)

Algoritmo responsável pelo cálculo da *Automated Readability Index* [20] a partir da fórmula:

$$4.71 * (characters/words) + 0.5 * words - 21.43 \quad (4.6)$$

Na fórmula do ARI, o valor resultante representa o nível escolar tendo em conta o sistema educativo americano. Na fórmula original o número de palavras é dividido pelo número de frases, sendo isso retirado, pois é apenas avaliado uma frase de cada vez, fazendo com que a divisão fosse sempre feita por um.

```
    def ARIA(Samples):
        Soma = 0
        for Sample in Samples:
            Texto = str(Sample)

            words = len(Texto.split())

            characters = len(re.sub(r'[^a-zA-Z\s
               ]+|\s+', '', Texto))

            Soma += 4.71 * (characters/words) + 0.5
                * words - 21.43

        Normalizacao = Normalize(Soma/len(Samples)
           ,14,1)

        return round(Soma/len(Samples),3), round(
           Normalizacao,3)
```

Excerto de Código 4.5: Trecho de código ARI.

No cálculo do *ARI* é utilizado um ciclo *for*, onde cada frase é dividida numa lista de palavras e em seguida, é contado o número de caracteres presentes em cada frase, após isso é aplicada a fórmula do *ARI*.

Como retorno da função é dado a soma total da fórmula aplicada em todas as frases a dividir pelo número total de amostras, como também esse mesmo valor normalizado. O resultado dado por esta função pode ser observado na seguinte tabela:

Valor	Nível Escolar	Intervalo de Idades
1	1	5-6
2	2	6-7
3	3	7-9
4	4	9-10
5	5	10-11
6	6	11-12
7	7	12-13
8	8	13-14
9	9	14-15
10	10	15-16
11	11	16-17
12	12	17-18
13	Faculdade	18-24
14	Licenciado	>24

Tabela 4.5: Resultados do algoritmo ARI.

4.5 Medidas de Complexidade Textual

4.5.1 Diversidade Lexical

Algoritmo responsável pelo cálculo da diversidade lexical [21] da amostra, isto é, o número de palavras diferentes a dividir pelo o número de palavras totais.

```
    def LexicalDiversityA(Samples):
        Text = ' '.join(Samples)
        Words = Text.split()
        return round(len(set(Words))/len(Words),3)
```

Excerto de Código 4.6: Trecho de código Diversidade Lexical.

Para o desenvolvimento deste algoritmo é necessário juntar todas as frases obtidas em uma única *string*, onde é separada cada palavra para uma lista. Após essa separação, é usada a função *set()*, que permite retirar da lista todas as palavras repetidas, fazendo em seguida a divisão pelo número total de palavras presentes na amostra e dando em retorno o valor pretendido.

O valor de retorno está contido no intervalo [0, 1], sendo o intervalo [0.6, 1] considerado uma diversidade lexical elevada. No intervalo [0.4, 0.6[a diversidade lexical é considerada normal e no intervalo [0, 0.4[a diversidade lexical é considerada baixa.

4.5.2 Densidade Lexical

Algoritmo responsável pelo cálculo da densidade lexical [21], isto é, o número de palavras lexicais (adjetivos, pronomes próprios, interjeições, verbos auxiliares, orações coordenadas, advérbios, preposições e substantivos) a dividir pelo número total de palavras.

```
    def LexicalDensityA(Samples):
        Soma = 0
        Count = 0
        for Sample in Samples:
            doc = nlp(str(Sample).lower())
            lexical_words = [token.text for token
               in doc if token.pos_ in {"ADP", "AUX
               ", "PART", "CCONJ", "NOUN", "INTJ",
               "PROPN", "ADJ"}]
            Count += len(str(Sample).split())
            Soma += len(lexical_words)

        return round(Soma/Count,3)
```

Excerto de Código 4.7: Trecho de código Densidade Lexical.

No desenvolvimento do algoritmo de densidade lexical, foi usado um ciclo *for* para analisar cada frase independentemente. Dentro desse ciclo é feito o tratamento com o módulo *SpaCy*, que a partir desta é possível atribuir a sua etiqueta sintática, isto é, atribuir à palavra a ser analisada se ela é pronome, adjetivo, verbo, entre outros. Após essa análise é guardado o número total de palavras na frase e o número de palavras pertencentes ao grupo lexical necessário (adjetivos, pronomes próprios, interjeições, verbos auxiliares, orações coordenadas, advérbios, preposições e substantivos), sendo o retorno final da função a divisão entre essas duas variáveis.

O valor devolvido tem como intervalo os valores entre [0, 1], sendo um texto considerado com uma boa densidade lexical aquele que estiver entre o intervalo [0.6, 1]. Um texto com um valor de densidade lexical dentro da média está no intervalo [0.4, 0.6[e um texto com um valor de densidade lexical abaixo da média encontra-se no intervalo [0, 0.4[.

4.5.3 Tamanho Médio das Palavras

Algoritmo responsável pelo cálculo do tamanho médio das palavras apresentadas nas amostras retiradas.

```
    def WordLengthA(Samples):
        Soma = 0
        for Sample in Samples:
            Sentence = str(Sample).split()
            WordLength = sum(len(Amostra) for
               Amostra in Sentence)
            Soma += WordLength/len(Sentence)

        Normalizacao = Normalize(Soma/len(Samples)
           ,10,1)

        return round(Soma/len(Samples),3), round(
           Normalizacao,3)
```

Excerto de Código 4.8: Trecho de código Tamanho Médio das Palavras.

Para medir o tamanho médio das palavras nas frases, foi utilizado um ciclo *for*, onde cada frase é dividida por palavras e por cada palavra é contado o número de caracteres. Em seguida, é somado o valor da divisão entre o número de caracteres e o tamanho total da frase em palavras.

O retorno da função é o valor somado de todas as divisões a dividir pelo número total de amostras utilizadas, como também esse mesmo valor normalizado.

O valor de retorno desta função está comprimido entre o intervalo [3, 7], sendo valores muito próximos de três normalmente pertencentes a um texto pouco complexo, textos com o intervalo médio de palavras entre [4,5] são onde maior parte dos textos analisados se inserem. Podendo haver exceções, onde o número de palavras supera a média de cinco, chegando raramente a um valor acima de seis, estes textos costumam obter uma complexidade textual elevada.

4.5.4 Tamanho Médio das Frases

Algoritmo responsável pelo cálculo do tamanho médio das frases apresentadas nas amostras retiradas.

```
    def SentenceLengthA(Samples):
        Soma = 0
        for Sample in Samples:
            Texto = str(Sample)
            TotalWords = len(Texto.split())
            Soma += TotalWords
```

```
        Normalizacao = Normalize(Soma/len(Samples)
            ,100,4)

        return round(Soma/len(Samples),3), round(
            Normalizacao,3)
```

Excerto de Código 4.9: Trecho de código Tamanho Médio das Frases.

O tamanho médio das frases é feito a partir de um ciclo *for*, onde cada frase é dividida numa lista de palavras que, em seguida, é utilizada uma variável que terá a soma total de palavras analisadas.

O retorno da função é dado pela divisão entre a variável com a quantidade total de palavras e o número de amostras retirado, tal como esse valor normalizado.

Apesar de não haver um intervalo definido para o tamanho máximo de uma frase, foi definido o intervalo [4,100] para ser possível a realização da normalização. O valor mínimo é quatro, pois como referido na secção 4.2 o tamanho mínimo das frases retirada é quatro.

4.5.5 Profundidade Sintática da Frase

Este algoritmo é responsável pelo cálculo médio da profundidade da árvore estrutural de cada frase.

```
    def average_depth(token, depth=0, count=0):
        if not list(token.children):
            return depth/max(1, count)
        else:
            total_depth = sum(average_depth(child,
                depth + 1, count + 1) for child in
                token.children)
            return total_depth

    def DepthAveA(Samples):
        Depth = 0
        for Sample in Samples:
            Texto = str(Sample)
            doc = nlp(Texto)
            for sent in doc.sents:
                Depth += average_depth(sent.root)
```

```
        Normalizacao = Normalize(Depth/len(Samples)
            ,30,1)

        return round(Depth/len(Samples),2), round(
            Normalizacao,3)
```

Excerto de Código 4.10: Trecho de código Árvore Estrutural.

Para o cálculo da média da árvore estrutural de cada frase é usada a livraria *SpaCy*, que é responsável por criar uma árvore em que cada nodo dessa árvore é uma palavra da frase em análise, após isso é usada a função *average_depth()* que calcula a profundidade máxima da árvore. Em seguida, é retornado pela função *DepthAveA()* a soma de todas as profundidades a dividir pelo número total de amostras, como também esse mesmo valor normalizado.

Como referido, o valor retornado indica, a profundidade da árvore da estrutura sintática da frase, fazendo com que quanto maior o valor da profundidade, mais complexa seja a amostra analisada. Para realizar a normalização foi criado o intervalo [1,30], pois dificilmente haverá uma frase com uma profundidade próxima à máxima.

4.6 Análise de Sentimentos

Na análise de sentimentos do texto foram utilizadas duas bibliotecas diferentes *SpaCy* e *NLTK*. Para uso final no modelo optou-se pelo *SpaCy* pela sua melhor precisão e atualidade.

```
    def SentimentA(Samples):
        nlp = SpaCy.blank('en')
        nlp.add_pipe('sentencizer')
        nlp.add_pipe('asent_en_v1')

        neg = 0
        neu = 0
        pos = 0
        compound = 0

        for Sample in Samples:
            doc = nlp(str(Sample))

            Sent = doc._.polarity
            neg += Sent.negative
            neu += Sent.neutral
```

```
            pos += Sent.positive
            compound += Sent.compound

        return round(neg/len(Samples),3), round(neu
           /len(Samples),3), round(pos/len(Samples)
           ,3), round(compound/len(Samples),3)
```

Excerto de Código 4.11: Trecho de código Análise Sentimental.

O cálculo do peso dos sentimentos é feito a partir *doc._.polarity*, função disponibilizada pela biblioteca *SpaCy*, que calcula para cada frase das amostras o sentimento negativo, positivo e neutro, como também o *compound*, que representa o sentimento geral do trecho de texto que se está a estudar. Assim, a função dá como retorno todos os sentimentos descritos anteriormente, juntamente com o *compound*.

4.7 Heurísticas

O modelo de IA desenvolvido é responsável pelo cálculo de duas medidas distintas: Cálculo da qualidade da estrutura textual e o valor sentimental do texto. Assim, foram pensadas diferentes heurísticas tendo em conta essas duas áreas.

4.7.1 Qualidade da Estrutura Textual

Para a criação do ficheiro *CSV* onde estão contidos todos os textos usados para o treino do modelo, foram utilizados textos de escritores profissionais, bem como *blogs*, que continham informação como: idade, género, signo, o texto desenvolvido, entre outros.

Numa primeira versão de heurística foi usada a idade e o nível profissional de escrita, havendo assim a criação de quatro grupos. O primeiro grupo continha todos os textos escritos por pessoas entre as idades de 13-22, o segundo grupo entre as idades de 23-32, o terceiro grupo entre as idades de 33-45 e o quarto grupo que continha todos os textos de escritores profissionais.

Esta primeira versão foi descartada, pois os resultados obtidos pelo modelo eram muito imprecisos, sendo apenas concluído que no grupo entre as idades de 13-22 a qualidade textual era baixa, no grupo de idades entre 23-32 e 33-45 a qualidade textual era semelhante, e no grupo de textos profissionais havia uma grande diferença de qualidade, pois muitos textos eram para públicos mais juvenis, fazendo com que a sua complexidade estrutural fosse mais simples.

Para uma segunda versão de heurísticas foi, para além da idade dos escritores, usada a média dos algoritmos utilizados, pois a característica comum entre todos os algoritmos é quanto maior o resultado, maior é a complexidade do texto analisado, sendo possível concluir que quanto maior a média de um texto, maior será a qualidade desse texto.

Na última versão de heurísticas, para uma melhor precisão e manipulação de dados foi adicionada a normalização dos valores dos algoritmos, fornecendo assim um intervalo comum de [0, 1].

Deste modo, foi possível concluir alguns intervalos de médias, sendo comum textos de nível muito baixo terem médias abaixo de 0.25, textos minimamente bem estruturados estão inseridos no intervalo [0.25, 0.50[, textos bem estruturados entre [0.50, 0.75[e textos considerados profissionais ou próximos disso têm uma média acima de 0.75.

```
    conditions = [
        df['Average'] < 0.25,
        (df['Average'] >= 0.25) & (df['Average'] <
           0.5),
        (df['Average'] >= 0.5) & (df['Average'] <
           0.75),
        df['Average'] >= 0.75
    ]
```

Excerto de Código 4.12: Trecho de código Heurísticas Estruturais Versão Final.

4.7.2 Visualização Sentimental

No processo da análise sentimental foram experimentadas duas versões de heurísticas, sendo as duas versões elaboradas usadas com o *SpaCy* e *NLTK*. Como referido anteriormente na secção 4.6, a versão final usada neste projeto usa o modelo desenvolvido com *SpaCy*.

Numa primeira versão de heurísticas, para a visualização sentimental é usada tanto o *compound*, isto é, a representação do sentimento geral da amostra no intervalo [-1, 1], onde valores negativos representam sentimentos negativos (tristeza, raiva, etc...) e valores positivos representam sentimentos agradáveis (felicidade, prazer, entre outros), como também a relação entre o sentimento negativo e positivo. Posto isto, foi verificado qual o sentimento predominante e, em seguida, verificado se o *compound* tinha a mesma relação, isto é, se o valor do sentimento predominante se encontrava no mesmo sentido do *compound*.

Esta versão foi descartada, pois em alguns casos o sentimento predominante e o *compound* não se encontravam no mesmo sentido, fazendo com que o resultado da análise textual fosse muito imprecisa.

Assim, na versão mais recente foi apenas usado o *compound* como argumento para o cálculo sentimental, pois, como referido, o *compound* constitui o sentimento principal geral das amostras analisadas, sendo um valor próximo de 1 um valor positivo (alegria, prazer, afeto, entre outros...), um valor próximo de -1 um valor negativo (raiva, tristeza, ansiedade, etc...) e para o sentimento neutro foram definidos valores próximos de 0, o que indica que o texto apresentado, muito provavelmente será um artigo, uma notícia ou uma pesquisa.

Numa abordagem mais técnica, para a determinação do valor sentimental de um texto foi usado o intervalo de [-1, -0.05[para identificar sentimentos negativos, o intervalo de [-0.05, 0.05] para identificar sentimentos neutros e o intervalo de]0.05, 1] para identificar sentimentos positivos.

```
    conditionss = [
        (dfs_num['Compound'] < -0.05),
        (dfs_num['Compound'] >= -0.05) & (dfs_num['
           Compound'] <= 0.05),
        (dfs_num['Compound'] > 0.05)
    ]
```

Excerto de Código 4.13: Trecho de código Heurísticas Sentimentais Versão Final.

4.8 Modelo XGBoost

Para modelo de treino foi usado o *XGBoost*. Este modelo foi escolhido pela sua maior taxa de acerto comparado com outros modelos, pois enquanto o *XGBoost* oferece uma taxa de acerto média de 95%, outros modelos, como árvore de decisão ou regressão linear oferece uma taxa de acerto de 93% e de 76%, respetivamente. Estes valores foram obtidos através de testes e comparações feitas entre estes algoritmos.

O treino realizado com este modelo foi feito com um ficheiro *CSV*, contendo mais de 406 textos desde a qualidade 0 (pior qualidade) à qualidade 3 (melhor qualidade).

Para a visualização da qualidade do modelo foi usada a função *confusion_matrix()*, onde indica a quantidade de textos usados e a quantidade de textos que foram corretamente e incorretamente avaliados. Além disso o *XGBoost* contem uma função, *plot_importance()*, que é possível obter e obser-

var quais os algoritmos mais relevantes para o treino do modelo, tendo assim uma noção sobre quais os algoritmos desenvolvidos foram mais relevantes no treino.

Relativamente, ao treino da visualização sentimental também foi utilizado o *XGBoost*, que comparada aos dois outros modelos aqui referidos, apresenta uma taxa de acerto média de 99%, enquanto que, o modelo de regressão linear e árvore de decisão apresentam uma taxa de acerto média de 41% e de 98%, respetivamente.

```
    def XGBoostTrain(FileIn, FileOut, Aux):
        warnings.filterwarnings('ignore', category=
           FutureWarning)

        data = pd.read_csv(FileIn)

        if Aux == 'Texto':
            X = data.drop(columns=['Text', '
               Classification','SentimentNeg','
               SentimentNeu','SentimentPos','
               Compound','ClassificationS'])
            y = data['Classification']
        elif Aux == 'Sentimento':
            X = data.drop(columns=['Text',      '
               Classification','ClassificationS','
               ARI','Coleman','Grade','
               LexicalDensity','LexicalDiversi  ty'
               ,'Reading','SentenceLength','Smog','
               Tree','WordLength'])
            y = data['ClassificationS']

        X_train, X_test, y_train, y_test =
           train_test_split(X, y, test_size=0.3)

        model = xgb.XGBClassifier(use_label_encoder
           =False, eval_metric='logloss')
        model.fit(X_train, y_train)

        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        report = classification_report(y_test,
           y_pred, output_dict=True)
```

```
        precision = report['weighted avg']['
           precision']
        recall = report['weighted avg']['recall']
        f1 = report['weighted avg']['f1-score']

        if Aux == 'Texto':
            if round(accuracy * 100,2) > 98: model.
               save_model(FileOut)
        elif Aux == 'Sentimento':
            if round(accuracy * 100,2) > 90: model.
               save_model(FileOut)

        cm = confusion_matrix(y_test, y_pred)
        plt.figure(figsize=(10, 6))
        sns.heatmap(cm, annot=True, fmt='d', cmap='
           Blues')
        plt.xlabel('Predicted Label')
        plt.ylabel('True Label')
        plt.title('Confusion Matrix')
        plt.show()

        data = pd.read_csv("DataV9_Spacy.csv")
        feature_names = data.drop(columns=['Text',
           'Classification','SentimentNeg','
           SentimentNeu','SentimentPos','Compound',
           'ClassificationS']).columns
        plot_importance(model, importance_type='
           gain', max_num_features=10)
        plt.gca()
        plt.show()

        return round(accuracy * 100,2), round(
           precision * 100,2), round(recall *
           100,2), round(f1 * 100,2)
```

Excerto de Código 4.14: Trecho de código para treino da IA.

4.9 WebSite

Front-End ou *client side* foi desenvolvido com a *framework flask*, como referido no Capítulo 3.6, a partir desta é possível fazer o tratamento do método *post*, que envia a informação colocada no formulário criado em HTML. Após recebida a informação do formulário, é tratada com as funções desenvolvidas e descritas na secção 4.8. Pelo que, após esse tratamento, será mostrado o resultado dessas funções juntamente com os algoritmos mais relevantes para a decisão mostrada. Para além destas informações, ainda são disponibilizados alguns textos exemplo para visualizar o funcionamento do modelo de IA criado.

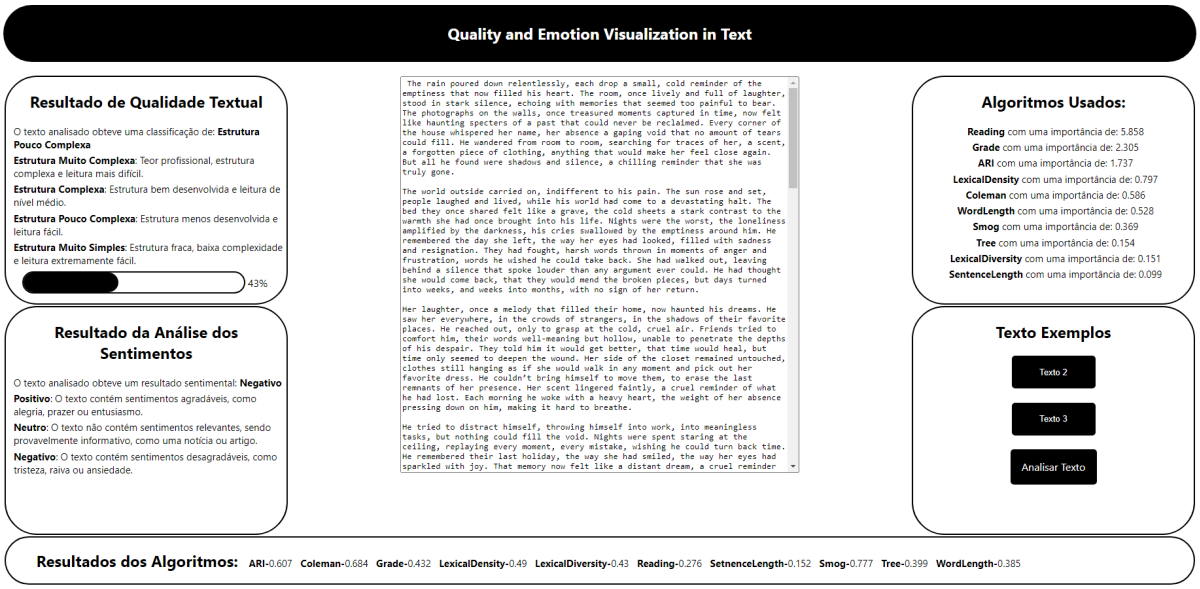

Figura 4.1: Estrutura visual do site desenvolvido

4.10 Conclusões

Resumindo, foram demonstradas todas as funcionalidades e implementações necessárias para o funcionamento e utilização do modelo inteligente, desde os algoritmos utilizados, como análise sentimental, métricas de complexidade textual e métricas de leitura, até aos modelos usados para treino e previsão de resultados. Aqui também foi relatado a importância de cada algoritmo para a implementação do modelo de IA.

Capítulo

5

Resultados

5.1 Introdução

Neste capítulo serão descritos testes realizados durante o projeto, bem como problemas encontrados ao realizá-los, também será explicado as soluções encontradas para os problemas citados.

Para além disso, serão de igual forma abordados as formas usadas para a visualização dos resultados, assim como o progresso realizado ao longo deste trabalho.

5.2 Visualização de Resultados

Para a visualização dos resultados de treino dos modelos criados da IA foram usadas ferramentas disponibilizados pelo *XGBoost*, descrito no Capítulo 4.8, e *scikit-learn*. Essas ferramentas são o *plot_importance()*, *confusion_matrix()*, *accuracy_score()* e *classification_report()*.

O *plot_importance()* é responsável por mostrar quais os algoritmos mais relevantes para o cálculo da previsão, adicionando um valor de importância, que quanto maior, mais relevante será o algoritmo. Esses resultados podem ser observados na seguinte imagem:

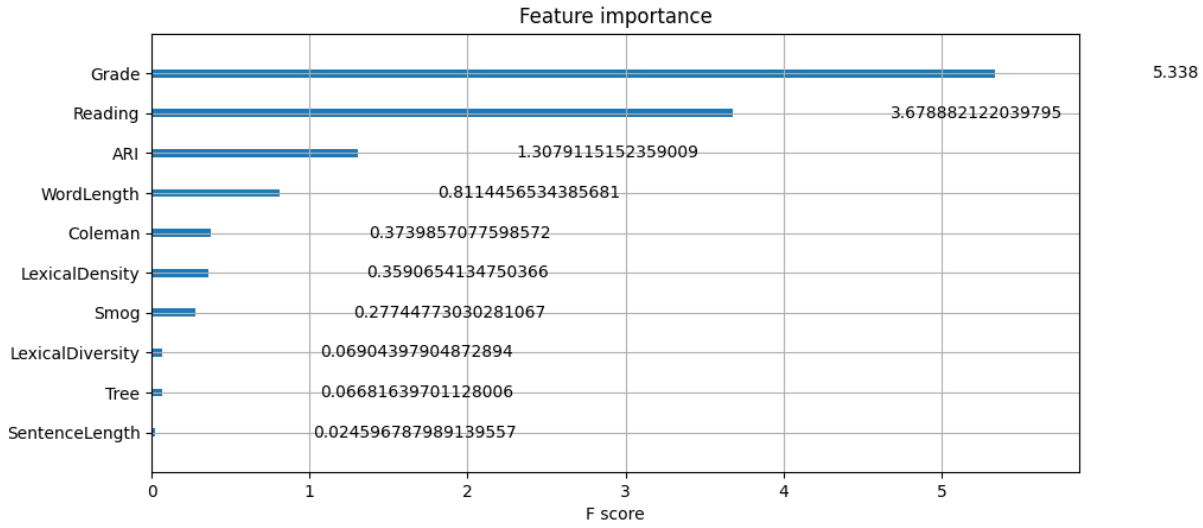

Figura 5.1: Importância dos algoritmos usados.

O *confusion_matrix()* é responsável por mostrar dentro dos dados de teste quais os textos bem previstos e mal previstos, isto é, feita a comparação do valor real da classificação dada a esse texto com a classificação prevista pelo modelo treinado. Sendo os resultados obtidos no treino os seguintes:

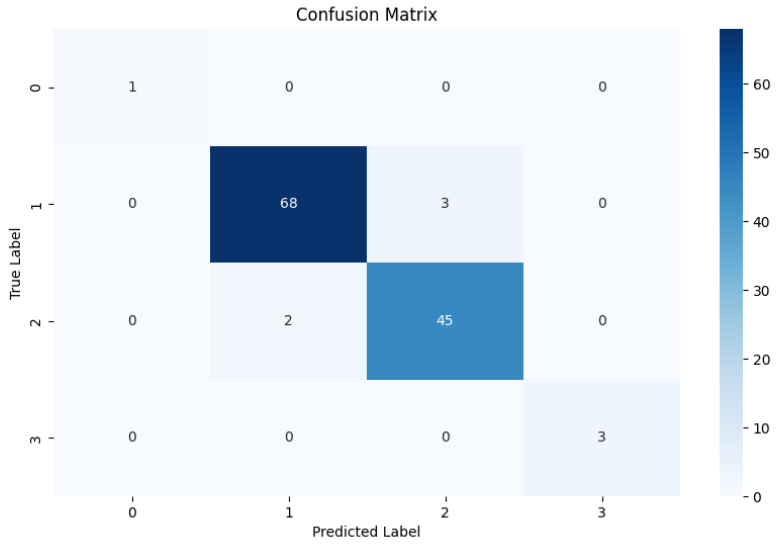

Figura 5.2: Resultados da matriz de confusão.

As ferramentas *accuracy_score()* e *classification_report()* permitem obter percentagens relevantes durante o treino do modelo, como precisão, taxa de acerto, *recall* e *F1 score*.

- Precisão é o valor que mede a proporção de verdadeiros positivos (TP) em relação à soma dos verdadeiros positivos com falsos positivos(FP), isto é, dos dados classificados quais eram realmente positivos. O modelo final usado para a classificação estrutural conta com uma precisão média de 96.27% e o modelo usado para a classificação sentimental obteve uma percentagem de 99.13%.

$$Precision = \frac{TP}{TP + FN} \tag{5.1}$$

- Taxa de acerto mede a proporção de todas as previsões corretas em relação ao número total de previsões. O modelo final conta com uma taxa de acerto média de 96.11% e 99.1% respetivamente, para o modelo de classificação estrutural e classificação sentimental.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.2}$$

- *Recall* é o valor que calcula a capacidade de um modelo identificar de forma correta todos os dados positivos. É a proporção entre verdadeiros positivos(TP) em relação à soma entre verdadeiros positivos(TP) e falsos negativos(FN). De forma simplificada o *recall*, trata de saber de todos os dados positivos quais foram realmente identificados corretamente. No modelo produzido, tanto a nível estrutural como a nível sentimental, apresenta um valor médio de *recall* de 96.11% e 99.11%, respetivamente.

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

- *F1 Score* é uma média entre a precisão e o *recall*, sendo o valor médio na classificação estrutural de 96.02%, enquanto que na classificação sentimental obteve um valor médio de 99.1%.

$$F1Score = 2 * \frac{Precison * Recall}{Precision + Recall} \tag{5.4}$$

5.3 Modelo de Treino

Durante o desenvolvimento deste projeto foram criados diferentes modelos de treino com diferentes heurísticas e base de dados.

Na primeira versão, foi usado como base de dados um ficheiro *CSV* com 209 textos analisados e como heurísticas a idade dos escritores, explicado no Capítulo 4.7.1. Para além disso foram usadas para treino as percentagens normais 30%-70%, isto é, 30% dos dados para teste e 70% dos dados usados para treino do modelo de IA. Com estas características o modelo feito obteve uma taxa de acerto de 48%.

Na segunda versão, o ficheiro *CSV* é constituído por 406 textos analisados, sendo usado como heurísticas a média dos algoritmos calculados e, por conseguinte, definidos intervalos de qualidade. Para treino foram usadas as percentagens referidas acima, 30%-70%, obtendo assim uma taxa de acerto de 75%.

Na última versão criada, houve a adição da normalização dos dados que, com as mesmas percentagens, foi possível obter uma taxa de acerto para o modelo produzido de 95%, sendo que, o modelo utilizado tem uma taxa de acerto de 98.36%.

Relativamente à previsão sentimental, foi realizada uma primeira versão com 406 textos analisados, sendo usado como heurística o sentimento predominante juntamente com o *compound*, citado no Capítulo 4.7.2. Obteve-se uma taxa de acerto média de 96% com as mesmas percentagens usadas nos testes anteriores, 30%-70%.

Na última versão realizada, apenas é usado o *compound* como heurística, sendo obtido assim uma taxa de acerto de 99%. Estes resultados devem-se ao facto da heurística utilizada ser simples, direta e também por uma elevada quantidade de textos positivos e neutros, o que causa uma maior taxa de acerto na realização do modelo de IA, tópico que será pormenorizado na secção 5.4. O modelo final utilizado conta com uma taxa de acerto de 99.18%.

5.4 Testes

Ao longo do desenvolvimento deste projeto foram efetuados diversos testes onde foi possível encontrar diversos problemas e possíveis soluções para os mesmos.

O primeiro teste realizado assentou nos algoritmos, onde foi verificada a eficiência do uso de amostras, isto é, a taxa de erro e o tempo usado comparado à análise do texto total. Aqui foram encontrados erros num algoritmo desenvolvido para a identificação de erros gramaticais e ortográficos e possíveis correções dos mesmos.

Este algoritmo foi descartado devido a um erro no módulo usado, *language_tool_python* [22], em que as *threads* da biblioteca não fechavam, gastando 100% da *random access memory* (RAM) o que, consequentemente, cau-

sava uma falha de sistema do computador usado. Outro problema encontrado neste algoritmo foi a sua imprecisão, pois comparando o algoritmo a calcular o texto completo com o cálculo das amostras os resultados eram muito distantes.

```
    def Grammar(Samples):
        tool = language_tool_python.LanguageTool('
           en-US')
        Soma = 0
        for Sample in Samples:
            matches = tool.check(Sample)
            Soma += len(matches)

        return Soma
```

Excerto de Código 5.1: Código responsável pelo número de erros gramaticais e ortográficos e possíveis correções.

Para além desse algoritmo, o algoritmo *DepthAveA()*, descrito no Capítulo 4.5.5, sofreu alterações, pois numa primeira versão o retorno desse algoritmo era dado pela profundidade máxima encontrada nas amostras. Esse retorno causava imprecisões, porque, dependendo das amostras retiradas, o resultado poderia variar drasticamente. Assim, a solução implementada foi realizar a média de profundidades de todas as amostras, dando assim um resultado mais consistente.

Devido ao grande volume de informação contido no ficheiro *CSV* um erro causado pelo módulo para manipulação do ficheiro, *csv* [23], foi necessária a adição da seguinte linha de código:

```
        csv.field_size_limit(2147483647)
```

Excerto de Código 5.2: Código usado para correção do erro no módulo csv.

O erro causado devia-se à necessidade de mais espaço em RAM, assim essa linha aumenta o espaço que a biblioteca pode utilizar para manipulação de ficheiros CSV.

Dentro da biblioteca *SpaCy* houve a origem de um erro devido à grande quantidade de palavras contidas no texto, sendo a seguinte linha de código utilizada para a resolução desse problema:

```
        nlp.max_length = 1600000
```

Excerto de Código 5.3: Código usado para correção do erro do SpaCy.

O erro era causado devido ao limite máximo de palavras que o *SpaCy* conseguia armazenar, sendo alterado para o seu máximo, 1.600.000 palavras.

Relativamente à forma como são retiradas as amostras, inicialmente eram retiradas do texto 1000 palavras aleatórias, como também, 10 parágrafos. Os parágrafos eram criados como agrupamentos de frases com um limite de 200 palavras. Esta forma de obter amostras causava erros, pois era comum haver parágrafos com frases incompletas e frases com apenas números e símbolos, causando, posteriormente, erros nos algoritmos.

Após isso, o retorno das amostras foi refeito, tendo sido retirado do texto 1000 palavras aleatórias e 60 frases aleatórias, com o apoio da biblioteca *SpaCy*, pois esta contém uma função capaz de dividir frases. Para além do uso dessa função, foram definidas frases que contivessem mais do que 4 palavras, pois caso houvesse símbolos ou números na frase retirada, há um contexto frásico a ser analisado.

Na versão mais recente são apenas retiradas 60 frases aleatórias, devido ao facto de que as 1000 palavras aleatórias eram apenas usadas pela função *WordLengthA()*, referida no Capítulo 4.5.3, assim, optou-se por usar as palavras presentes nas frases para fazer o cálculo do valor dado pela função.

Um outro teste realizado foi a tentativa de avaliar várias linguagens, português e inglês. Essa implementação não foi realizada devido à falta de dados para a análise em português, o que causava um resultado muito abaixo do pretendido na previsão da qualidade textual.

Para verificar a veracidade dos dados obtidos na análise da base de dados foi registada a quantidade de textos entre a qualidade textual de 0-3 e também registada a quantidade de textos com a visualização de sentimentos entre 0-2, sendo obtido a seguinte tabela:

Sentimento	**Negativo**	**Neutro**	**Positivo**
Nº Textos	51	141	212

Tabela 5.1: Tabela de Sentimentos.

Estrutura	**Muito Simples**	**Pouco Complexa**	**Complexa**	**Muito Complexa**
Nº Textos	8	238	138	20

Tabela 5.2: Tabela de Estrutura.

A partir dos dados da tabela relativa à qualidade textual, os dados obtidos estão dentro do expectável, pois, a principal fonte de dados são os *blogs* em que maior parte dos textos são escritos e produzidos pela comunidade no geral. Já os textos de muito boa qualidade textual, o número obtido é realista, pois foram colocados 36 textos com teor profissional, tendo já sido referido

que alguns desses textos eram para público mais juvenil, que entram na qualidade textual "Pouco Complexa".

Quanto aos dados alcançados a nível de sentimentos, estes encontram-se dentro do esperado, porque como referido anteriormente a fonte de dados é baseada em *blogs*. Posto isto, é frequente haver textos com sentimento positivo, sendo a intensidade de textos negativos e neutros menor.

5.5 Conclusões

Em suma, foi explicado e mostrado o progresso e atualizações realizadas durante o projeto, como também os testes feitos para suportar as alterações submetidas. Também foi apresentado o motivo das alterações e a importância das mesmas.

Capítulo

6

Conclusões e Trabalho Futuro

6.1 Conclusões Principais

Para a conclusão deste trabalho foi necessário uma série de algoritmos bem desenvolvidos e relevantes na área de processamento de linguagem natural. Juntamente com estes algoritmos, o modelo utilizado para treino do modelo de IA garantiu uma maior precisão nos resultados obtidos, tanto na sua avaliação textual, como na visualização de sentimentos.

Umas das conclusões principais deste projeto é a importância de algoritmos relevantes para a analise textual, tanto na sua estrutura, como a nível sentimental. Portanto é importante referir que de todos os algoritmos usados os que obtiveram um maior impacto no modelo utilizado foram: *Flesch Reading Ease*, *Flesch-Kincaid Grade Level*, *Automated Readability Index* e a densidade lexical. Na avaliação sentimental como apenas é usado o *compound* apenas essa é relevante.

Outro ponto a destacar, é a importância das heurísticas utilizadas, pois é a partir das heurísticas escolhidas que o modelo inteligente desenvolvido faz a sua decisão e, consequentemente, tem uma elevada precisão na sua avaliação, pois no desenvolvimento de várias versões das heurísticas foi possível ver uma alteração na precisão do modelo de IA. Em virtude deste modelo, este apresenta uma taxa de acerto com uma média de 96.11% na classificação estrutural e de 99.1% na classificação sentimental, este modelo apresenta ainda uma precisão média de 96.27% na classificação estrutural e de 99.13% na análise sentimental. Relativamente ao *recall* demonstra uma média de 96.11% no modelo estrutural e de 99.11% no modelo sentimental. *F1 Score* apresenta uma média de classificação estrutural e de análise sentimental de 96.02% e de 99.1%, respetivamente.

Em suma, o projeto desenvolvido alcançou quase todos os objetivos pretendidos, explorando assim o que é feito na área de processamento de linguagem natural. Nesta área pode ser concluído que há três características importantes: algoritmos utilizados, heurísticas desenvolvidas e o modelo de treino usado. No que se refere aos algoritmos utilizados, é possível concluir que há uma relação, pois a partir dos resultados obtidos é possível entender que quanto mais elevados são os resultados, mais elevada é a qualidade estrutural do texto.

6.2 Trabalho Futuro

Durante o desenvolvimento deste trabalho foram concluídas diversas tarefas, ficando apenas uma tarefa não concluída, sendo essa a implementação de processamento de textos de diferentes idiomas (Português, Inglês, entre outros). Isto deve-se ao facto da falta de textos nesses idiomas para treino da IA.

Tendo em conta o objetivo deste projeto, seria interessante avaliar outras áreas inerentes a um texto, como a sua área de inserção, isto é, a área onde o texto se encontra inserido, sendo ele, ciência, política, romance, entre outros. Adicionalmente, seria também possível considerar a criação de um algoritmo onde fosse possível a correção e avaliação gramatical e ortográfica.

Com o desenvolvimento deste projeto foi possível concluir algumas aplicações de interesse, como uma possível avaliação em tempo real de um texto a ser escrito no bloco de notas ou no *Microsoft Word*, dando assim uma ideia ao utilizador da qualidade textual e também a visualização de sentimentos no texto desenvolvido.

Apêndice

A

Anexo

A.1 Funções Auxiliares

Na criação do ficheiro *CSV* utilizado para o armazenamento do textos a ser analisados, foi criada uma série de funções de suporte.

Para o agrupamento de amostras foi utilizada uma função que consiste em receber o texto, onde, após isso, é criada uma lista com todas as frases com mais de 4 palavras, onde a divisão frásica é feita através do módulo *SpaCy*. É retirado no mínimo 4 palavras, pois para haver uma contexto frásico é necessário de 4 a 5 palavras. Em seguida, é escolhido de forma aleatória 60 frases, onde mais tarde serão usadas para a análise textual.

```
    def Amostras(Texto):
        doc = nlp(Texto)
        Sentences = [sent.text.strip() for sent in
           doc.sents if len(sent.text.split()) >=
           4]
        Frases = random.sample(Sentences,60)

        return Frases
```

Excerto de Código A.1: Função utilizada para retirar amostras.

Na manipulação de ficheiro *CSV* houve a criação de várias funções, havendo 3 com elevada importância, sendo elas: *Resultados()*, *CsvAlgo()* e *Heuristics()*.

Na função *Resultados()* é dado como argumento as amostras retiradas do texto e, após isso é feito o cálculo dos algoritmos desenvolvidos em diferentes

Threads, para uma melhor eficiência e organização. Em seguida, são colocados num dicionário, que é dado como retorno da função.

```
    def Resultados(Samples):
        ResultadosA = {}

        def TSMOGA(): ResultadosA['Smog'] = RM.
           SMOGA(Samples)
        def TColemanA(): ResultadosA['Coleman'] =
           RM.ColemanA(Samples)
        def TGradeA(): ResultadosA['Grade'] = RM.
           FleschGradeA(Samples)
        def TReadingA(): ResultadosA['Reading'] =
           RM.FleschReadingA(Samples)
        def TARIA(): ResultadosA['ARI'] = RM.ARIA(
           Samples)

        def TSentenceLengthA(): ResultadosA['
           SentenceLength'] = TCM.SentenceLengthA(
           Samples)
        def TWordLengthA(): ResultadosA['WordLength
           '] = TCM.WordLengthA(Samples)
        def TLexicalDensityA(): ResultadosA['
           LexicalDensity'] = TCM.LexicalDensityA(
           Samples)
        def TLexicalDiversityA(): ResultadosA['
           LexicalDiversity'] = TCM.
           LexicalDiversityA(Samples)

        def TTreeA(): ResultadosA['Tree'] = Tree.
           DepthAveA(Samples)

        def TSentimentA(): ResultadosA['Sentiment']
            = SA_NLTK.SentimentA(Samples)

        threads = [
            threading.Thread(target=
               TSentenceLengthA),
            threading.Thread(target=TWordLengthA),
            threading.Thread(target=
               TLexicalDensityA),
```

```
            threading.Thread(target=
               TLexicalDiversityA),
            threading.Thread(target=TTreeA),
            threading.Thread(target=TSentimentA),
            threading.Thread(target=TGradeA),
            threading.Thread(target=TSMOGA),
            threading.Thread(target=TColemanA),
            threading.Thread(target=TReadingA),
            threading.Thread(target=TARIA),
        ]

        for thread in threads: thread.start()

        for thread in threads: thread.join()

        return ResultadosA
```

Excerto de Código A.2: Função utilizada para calcular os algoritmos nas amostras retiradas.

Na função *CsvAlgo()* é dado como argumento dois ficheiros *CSV*, onde o primeiro ficheiro tem como conteúdo todos as amostras retiradas dos textos e o segundo argumento um ficheiro *CSV* onde serão colocadas as amostras com os respetivos algoritmos aplicados. Nesta função ambas as classificações usadas mais tarde para o modelo de IA são inicializadas a zero. Aqui é utilizada a função *Resultados()* e a função *Amostras()*.

```
    def CsvAlgo(input_csv, output_csv):
        filtered_texts = []

        with open(input_csv, 'r', encoding='utf-8')
            as file:
            reader = csv.DictReader(file)
            for row in reader:
                text = row['Text']
                filtered_texts.append(text)

        with open(output_csv, 'a', newline='',
           encoding='utf-8') as file:
            writer = csv.writer(file)
            writer.writerow([
                'Text',
                'ARI',
```

```
            'Coleman',
            'Grade',
            'LexicalDensity',
            'LexicalDiversity',
            'Reading',
            'SentenceLength',
            'SentimentNeg',
            'SentimentNeu',
            'SentimentPos',
            'Compound',
            'Smog',
            'Tree',
            'WordLength',
            'Classification',
            'ClassificationS'
            ])

        for Texto in filtered_texts:
            Samples = Amostras(Texto)
            ResultadosA = Resultados(Samples)

            writer.writerow([
                Texto,
                ResultadosA['ARI'][0],
                ResultadosA['Coleman'][0],
                ResultadosA['Grade'][0],
                ResultadosA['LexicalDensity'
                    ][0],
                ResultadosA['LexicalDiversity'
                    ][0],
                ResultadosA['Reading'][0],
                ResultadosA['SentenceLength'
                    ][0],
                ResultadosA['Sentiment'][0],
                ResultadosA['Sentiment'][1],
                ResultadosA['Sentiment'][2],
                ResultadosA['Sentiment'][3],
                ResultadosA['Smog'][0],
                ResultadosA['Tree'][0],
                ResultadosA['WordLength'][0],
                0,
```

```
                0
            ])
    return
```

Excerto de Código A.3: Função utilizada para guardar as amostras juntamente com os resultados dos algoritmos.

Na função *Heuristics()* é dado como argumento o ficheiro *CSV* produzido pela função anteriormente descrita. Esta função aplica as heurísticas desenvolvidas para a previsão da IA, sendo representadas no *CSV* como *Classification* e *ClassificationS*, dando como retorno um *CSV* que contém todas as amostras mais os seus algoritmos, juntamente como as classificações calculadas através das heurísticas.

```
def Heuristics(File):
    df = pd.read_csv(File)

    dfs_num = df.drop(columns=['Text','ARI','
       Coleman','Grade','LexicalDensity','
       LexicalDiversity','Reading','
       SentenceLength','Smog','Tree','
       WordLength','Classification','
       ClassificationS'])

    df_num = df.drop(columns=['Text','
       Classification','SentimentNeu','
       ClassificationS','Compound','
       SentimentPos','SentimentNeg'])
    df_num = df_num.apply(pd.to_numeric, errors
       ='coerce')
    df['Average'] = df_num.mean(axis=1).round
       (2)

    conditions = [
        df['Average'] < 0.25,
        (df['Average'] >= 0.25) & (df['Average'
           ] < 0.5),
        (df['Average'] >= 0.5) & (df['Average']
            < 0.75),
        df['Average'] >= 0.75
    ]
     choices = [0, 1, 2, 3]
```

```
conditionss = [
    (dfs_num['Compound'] < -0.05),
    (dfs_num['Compound'] >= -0.05) & (
       dfs_num['Compound'] <= 0.05),
    (dfs_num['Compound'] > 0.05)
]
choicess = [0, 1, 2]

df['Classification'] = np.select(conditions
   , choices)
df['ClassificationS'] = np.select(
   conditionss, choicess)
df.to_csv('B9.csv', index=False)

df = pd.read_csv('B9.csv')
df = df.drop(columns=['Average'])
df.to_csv('DataV9_Spacy.csv', index=False)
```

Excerto de Código A.4: Função utilizada para a aplicação das heurísticas.

Glossário

Application Programming Interface Interface de programação de aplicações (do inglês *application programming interface*, abreviado API) é um conjunto de serviços/funções que foram implementadas em um programa de computador que são disponibilizados para que outros programas possam utilizá-los diretamente de forma simplificada.. 8

blogs A maioria dos *blogs* são primariamente textuais, embora uma parte seja focada em temas exclusivos como arte, fotografia, vídeos, música ou áudio, formando uma ampla rede social.. 26, 38, 39

Cascading Style Sheets *Cascading Style Sheets* é um mecanismo para adicionar estilos a uma página web, aplicado diretamente nas *tags* HTML ou ficar contido dentro das *tags* <*style*>.. 11

client side Lado cliente, também conhecido como *front-end* refere-se ao ato de obter informações de outro computador chamado servidor no contexto do modelo cliente-servidor de redes de computadores.. 11, 31

deep learning A aprendizagem profunda, do inglês *Deep Learning* é um ramo de aprendizagem baseado em um conjunto de algoritmos que tentam modelar abstrações de alto nível de dados usando um grafo profundo com várias camadas de processamento, compostas de várias transformações lineares e não lineares.. 10

fractal Fractal é uma figura da geometria não clássica muito encontrada na natureza, isto é, um objeto em que suas partes separadas repetem os traços do todo completo.. 5

framework É uma abstração que une códigos comuns entre vários projetos de *software* provendo uma funcionalidade genérica.. 9, 11, 31

GitHub *GitHub* é uma plataforma de hospedagem de código-fonte e arquivos com controle de versão usando o *Git*.. 11

HyperText Markup Language HTML é uma linguagem de marcação utilizada na construção de páginas na *Web*.. 11

JavaScript Trata-se de uma linguagem de programação interpretada, que utiliza *scripts* em alto nível, caracterizada por uma tipagem dinâmica fraca e que suporta múltiplos paradigmas.. 11

Microsoft Word O *Microsoft Word* é um processador de texto produzido pela Microsoft Office/Microsoft 365.. 42

post O método *POST* é uma maneira de enviar dados de um formulário da Web para um servidor usando o protocolo HTTP. Os dados não são visíveis na URL, ao contrário do método *GET*, que acrescenta os dados como cadeias de caracteres de consulta.. 31

Python É uma linguagem de programação de alto nível, interpretada, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte.. 9, 11

random access memory A RAM é um banco de memória temporário onde o computador armazena os dados que precisa recuperar rapidamente.. 36

software *Software* é uma coleção de programas e dados que dizem a um computador como executar tarefas específicas. Isso contrasta com o *hardware*, a partir do qual o sistema é construído e que realmente executa o trabalho.. 7

threads É uma sequência de instruções que pode ser executada de forma independente dentro de um programa.. 36

website É um conjunto de páginas conectadas por hipertextos, acessíveis geralmente pelo protocolo HTTP ou pelo HTTPS na *internet*.. 11

Bibliografia

[1] Joao Cordeiro, Pedro RM Inácio, and Diogo AB Fernandes. Fractal beauty in text. In *Progress in Artificial Intelligence: 17th Portuguese Conference on Artificial Intelligence, EPIA 2015, Coimbra, Portugal, September 8-11, 2015. Proceedings 17*, pages 796–802. Springer, 2015.

[2] Meri Coleman and Ta Lin Liau. A computer readability formula designed for machine scoring. *Journal of Applied Psychology*, 60(2):283, 1975.

[3] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams engineering journal*, 5(4):1093–1113, 2014.

[4] Grammarly Inc. Grammarly, 2024. [Online] https://www.grammarly.com/.

[5] Hemingway. Hemingway, 2013. [Online] https://hemingwayapp.com/.

[6] Orpheus Technology. ProWritingAid, 2024. [Online] https://prowritingaid.com/.

[7] Google Cloud. Natural Language AI, 2024. [Online] https://cloud.google.com/natural-language.

[8] Lexalytics. Lexalytics, 2022. [Online] https://www.lexalytics.com/.

[9] IBM. IBM Watson Natural Language Understanding, 2021. [Online] https://www.ibm.com/products/natural-language-understanding.

[10] NLTK Project. NLTK, 2023. [Online] https://www.nltk.org/.

[11] Explosion. SpaCy, 2016. [Online] https://spacy.io/.

[12] xgboost developers. XGBoost, 2022. [Online] https://xgboost.readthedocs.io/en/stable/.

[13] scikit-learn developers. scikit-learn, 2007. [Online] https://scikit-learn.org/stable/.

[14] Pallets. Flask, 2010. [Online] https://flask.palletsprojects.com/en/3.0.x/.

[15] Readable. The SMOG Index, 2011. [Online] https://readable.com/readability/smog-index/.

[16] Readable. The Coleman Liau Readability Index, 2011. [Online] https://readable.com/readability/coleman-liau-readability-index/.

[17] Python Software Foundation. syllapy 0.7.2, 2024. [Online] https://pypi.org/project/syllapy/.

[18] Readable. Flesch Reading Ease and the Flesch Kincaid Grade Level, 2011. [Online] https://readable.com/readability/flesch-reading-ease-flesch-kincaid-grade-level/.

[19] Readable. Flesch Reading Ease and the Flesch Kincaid Grade Level, 2011. [Online] https://readable.com/readability/flesch-reading-ease-flesch-kincaid-grade-level/.

[20] Readable. The Automated Readability Index, 2011. [Online] https://readable.com/readability/automated-readability-index/.

[21] LINDA THORPE. What are Lexical Density and Lexical Diversity?, 2023. [Online] https://readabilityformulas.com/what-are-lexical-density-and-lexical-diversity/.

[22] Python Software Foundation. language-tool-python 2.8, 2024. [Online] https://pypi.org/project/language-tool-python/.

[23] Python Software Foundation. CSV File Reading and Writing, 2001. [Online] https://docs.python.org/3/library/csv.html.