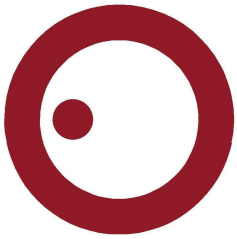


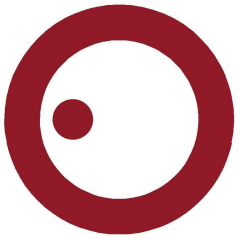
escola superior de tecnologia e gestão
instituto politécnico de leiria

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

METODOLOGIAS BASEADAS EM INTELIGÊNCIA ARTIFICIAL PARA A DETEÇÃO DE CORREIO ELETRÓNICO NÃO SOLICITADO

PEDRO VILAÇA CRUZ CERQUEIRA DOS SANTOS

Leiria, Setembro de 2025



escola superior de tecnologia e gestão
instituto politécnico de leiria

Instituto Politécnico de Leiria
Escola Superior de Tecnologia e Gestão
Departamento de Engenharia Informática
Mestrado em Cibersegurança e Informática Forense

METODOLOGIAS BASEADAS EM INTELIGÊNCIA ARTIFICIAL PARA A DETEÇÃO DE CORREIO ELETRÓNICO NÃO SOLICITADO

PEDRO VILAÇA CRUZ CERQUEIRA DOS SANTOS
Número: 2230464

Projeto realizado sob orientação do Professor Doutor Miguel Monteiro de Sousa Frade, Professor Doutor Patrício Rodrigues Domingues e Professor Doutor Miguel Cerdeira Marreiros Negrão.

Leiria, Setembro de 2025

AGRADECIMENTOS

Gostaria de agradecer aos meus orientadores por toda a ajuda que prestaram ao longo do curso deste trabalho.

RESUMO

O problema da deteção de correio eletrónico não solicitado (Spam) é um que esteve presente desde os primórdios da Internet levando a que, poucos anos após a introdução desta, começasse o desenvolvimento de sistemas e algoritmos para detetar e bloquear esta ameaça informática. Tentativas iniciais consistindo de moderação humana com auxílio de relatórios pelas vítimas e de sistemas simples baseados no bloqueio de certos termos ligados ao *spam* introduzidos numa *blacklist* manualmente pelos seus criadores. No entanto, à medida que a deteção de *spam* evoluía, o *spam* também evolui nas suas técnicas, desde táticas como a intencional introdução de erros na escrita para confundir os sistemas de deteção rudimentares que não conseguiam detetar corrupções de palavras contidas na sua *blacklist*, requerendo que os criadores introduzam manualmente todas as possíveis corrupções eles próprios. Uma das mais recentes inovações nesta área foi a de utilizar modelos de Inteligência Artificial, mais especificamente, de Processamento de Linguagem Natural, para tentar criar um modelo capaz de acompanhar indeterminadamente a evolução do *spam*, criando uma solução permanente a este problema. Neste trabalho, é o objetivo treinar um modelo NLP (NLP) baseado na arquitetura *Sentence BERT* (*Bidirectional Representations from Transformer*), que possa servir de exemplo da capacidade e potencial desta no combate ao *spam*, este modelo será treinado com um *dataset* de emails com mais de 90 mil emails sendo cerca de metade destes *spam* e a restante parte emails legítimos. O modelo obtido demonstrou resultados positivos, tendo a versão final alcançado, em combinação com um classificador Support Vector Classifier (SVC), métricas que ultrapassaram os 98.5%, com o modelo sBERT a consumir cerca de 30 micro segundos a vetorizar cada email.

A B S T R A C T

The issue of unsolicited electronic mail (spam) detection is one that has been present since the begginings of the Internet leading to, just a few years after its' introduction, the development of sistems and algorithms to detect and surpress this threat. Initial efforts were primarily consistent of human intervention aided by user reports, with rudimentary automated blocking systems based on a handcrafted word blacklist appearing soon after. However, as spam detection evolved, so too did spam itself evolve to counter these evolutions, with developments such as the intentional miswriting of suspicious word so as to avoid detection by blacklisting, requiring manual insertion of every possible corruption of a word into the blacklist to counter it. One of the most recent development on the side of spam detection has been the application Artificial Intelligence, more specifically, Natural Language Processing Models (NLP) to the task, in an attempt to utilize its self improving powers to create a system capable of trailing spams' own evolution and thus, a more permanent or, at least, longer lasting solution to the issue. In this work, the goal is to train one such NLP model, based on the Sentence BERT (Bidirectional Encoding Representations from Transformer) architecture, which may serve as an example of the capacity and potential of this avenue in detecting spam, the model in question will be trained with a dataset of various emails numbering just over 90 thousand with roughtly half of it consisting of spam and the remainder legitimate emails. The resulting model displayed positive results, with the final result having, in combination with an Support Vector Classifier (SVC), surpassed 98.5% in all metrics, with the model taking an average of 30 micro seconds per email vectorized.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

BERT	Bidirectional Encoder Representations from Transformer.
IA	Inteligência Artificial.
IDS	Intrusion Detection System.
KNN	K-Nearest Neighbors.
LLM	Large Language Model.
ML	Machine Learning.
NLP	Natural Language Processing.
RN	Rede Neuronal.
sBERT	Senteance Bidirectional Encoder Representation from Transformer.
SMS	Short Message Service.
SVM	Support Vector Machines.

1 INTRODUÇÃO

1.1 CONTEXTO

A aplicação de Inteligência Artificial (IA) ao campo da cibersegurança é um tópico que tem evoluído significativamente, tanto em interesse como no desenvolvimento técnico. Tal é motivado por uma crescente inovação por parte dos atacantes que leva à diminuição de capacidade de métodos convencionais de deteção e também um desejo de diminuir custos e aumentar a rapidez e eficácia dos sistemas existentes. Com várias diferentes empresas a oferecerem soluções para diversos problemas da área que incorporam ou são inteiramente constituídas por IA, desde soluções antivírus colocadas em *endpoints* específicos a mais abrangente soluções Intrusion Detection System (IDS) que filtram a rede de uma organização e detetam comportamentos anormais. Um dos vários temas da cibersegurança que tem recebido atenção por parte destes desenvolvimentos é o da deteção e combate ao correio eletrónico *spam*.

1.1.1 *Spam*

O *spam* é uma das ameaças mais antigas do mundo informático, tendo existido praticamente desde os primórdios da Internet e do correio eletrónico. O *spam* consiste em emails ou qualquer outro formato de mensagem enviada por meio informático em massa para um elevado número de alvos que não consentem em recebê-las, geralmente com objetivos financeiros ou criminosos e, em muitos casos, tentando imitar a aparência de mensagens legítimas para aumentar a probabilidade de o recetor interagir com os mesmos.

Em anos recentes, particularmente na era pós-COVID (2023-Presente), houve um aumento considerável no uso da Internet tanto por indivíduos como por empresas e organizações devido à popularização do teletrabalho (Aksoy et al., 2022). Consequentemente, o *spam* também viu um aumento explosivo na sua incidência e evolução (Guerra et al., 2010), tendo vindo a encontrar novos vetores de ataque

para ultrapassar as soluções existentes e utilizando táticas de engenharia social mais inteligentes para conseguir a atenção da vítima (Guerra et al., 2010).

As soluções ditas tradicionais para combater o *spam* sempre sofreram de problemas quanto à sua capacidade de lidar com táticas como a troca de letras por letras similares ou o uso de pontuação e sinais para disfarçar palavras suspeitas. Deve-se ter em conta também a dificuldade destes sistemas em detetar a intenção, sentimento e contexto por detrás de uma dada mensagem, o que dificulta a diferenciação entre *spam* e texto legitimo, levando a que estes sistemas tivessem que ser intencionalmente mais permissivos com o *spam*, de forma a evitar a interceção não intencional de email legitimo Fawcett, 2003. Para oferecer um exemplo prático, um atacante poderia distribuir uma mensagem de spam que menciona pagamentos em falta ou a necessidade de confirmar alguma ação noutro site, contendo um link malicioso. Tal email criaria um cenário difícil para um filtro de spam convencional dado que, não podendo o filtro considerar o contexto, a mensagem aparenta ser legitima de uma organização como a *Amazon* ou outro qualquer tipo de serviço, pelo que filtrar estas mensagens não seria desejoso, devido ao risco de perdas para o utilizador.

1.1.2 *Processamento de Linguagem Natural*

Uma das várias áreas que a IA veio a englobar é o processamento de linguagem natural. Este consiste no desenvolvimento de um modelo inteligente capaz de avaliar o significado de palavras escritas em linguagem comum, mas também determinar a intenção e sentimento por detrás das mesmas. Naturalmente houve, desde cedo, um interesse em aplicar este tipo de IA à deteção de *spam*, com o intuito que a capacidade de inferir a intenção de uma mensagem permitisse uma maior taxa de acerto comparativamente com os modelos não inteligentes que vieram previamente.

Em Al Saidat et al., 2024 são discutidos os resultados obtidos com implementações de modelos Natural Language Processing (NLP) (e também de *Machine Learning (ML)*) na deteção de *spam* em *Short Message Service (SMS)*. O trabalho conclui com resultados positivos que comprovam a utilidade e contributo da IA para o campo do combate ao *spam*, fornecendo também conselhos práticos para desenvolvimentos futuros na área. Já foram realizados trabalhos tanto práticos na implementação de sistemas NLP e de outros tipos de IA no combate ao *spam*, particularmente em áreas como emails, SMS e revisões de comercio online. Todos com resultados promissores, bem como trabalhos teóricos a apontar potenciais áreas que podem ser melhoradas e fornecendo uma visão do estado atual desta área.

1.2 MOTIVAÇÃO E OBJETIVOS

1.2.1 *Estatísticas Relevantes*

Statista, 2025b indica que o email, mesmo com o aparecimento de várias formas de comunicação rivais, continua a ver um uso elevado, com mais de 4 mil milhões de utilizadores globais em 2023, sendo previsto que este número suba em 400 milhões até 2027 sendo que em Statista, 2025a se demonstra que o *spam* tem acompanhado esse crescimento. Por exemplo, em Dezembro do ano 2024 foram detetados, nos Estados Unidos, 7,8 mil milhões de emails não solicitados por dia, isto é, sensivelmente, 100 mil milhões por mês. Tal demonstra que este meio de comunicação continua a ter elevada importância e necessita de precauções de segurança especiais dado o seu uso por um número elevado de pessoas, a maioria das quais não possui a competência técnica para detetar spam por iniciativa própria. O *spam* é uma ameaça que, apesar dos grandes esforços por várias organizações ao nível técnico e governos ao nível legal Ferrara, 2019 de o conter, continua a ver um nível de incidência no mundo gigante. Devido a tal, existe a necessidade de avaliar e melhorar as soluções existentes de combate ao *spam*, sendo que estas estão, atualmente, principalmente focadas em modelos NLP e de ML em geral. É o objetivo deste trabalho treinar várias implementações de classificadores de spam combinados com modelos NLP para detetar o *spam* e determinar a sua eficácia nesta função, bem como encontrar possíveis falhas e erros.

1.3 CONTRIBUTOS

Este trabalho visa oferecer as seguintes contribuições para a área:

1. Uma visão do estado atual da deteção de *spam*.
2. Levantar possíveis áreas que podem servir de foco em trabalhos futuros.
3. Comparar metodologias baseadas em IA com soluções clássicas.
4. Definir um modelo de classificação capaz de determinar, idealmente em tempo real, emails que são *spam*.

1.4 ORGANIZAÇÃO DO RELATÓRIO

Este relatório está organizado da seguinte forma.

No capítulo 2 são explicados conceitos de carácter técnico que vão ser utilizados neste trabalho de forma a facilitar a sua leitura.

No capítulo 3 é feita uma revisão da literatura relevante ao estado da arte na área de deteção de *spam*.

No capítulo 4 é descrita a metodologia seguida na elaboração deste trabalho, oferecendo uma visão geral do seu funcionamento.

No capitulo 5 são descritos: Os *datasets* utilizados neste projeto; O tratamento que foi feito sobre estes; As bibliotecas utilizadas e uso destas; Os modelos de classificação e vetorização utilizados e justificações para estes.

No capitulo 6 estão colocados os resultados dos testes práticos realizados, juntamente com gráficos para permitir a sua fácil análise e uma curta discussão pelo autor.

Finalmente, no capitulo 7 é concluído o relatório.

2

CONCEITOS

Para uma melhor compreensão deste relatório, é importante o leitor entender alguns conceitos base de IA, modelos NLP e *spam* em geral.

2.1 VETOR DE PALAVRAS

Um vetor de palavras é uma representação matemática de uma dada palavra ou frase num espaço multi-dimensional. Estes vetores, através da sua proximidade ou distância no espaço, permitem identificar a similaridade entre as palavras ou frases que representam.

2.2 SIMILARIDADE DE VETORES

No contexto de NLP, a similaridade entre vetores refere-se à proximidade espacial entre dois ou mais vetores. É uma métrica com vários usos nesta área incluindo a determinação de conexão (ou a sua ausência) entre dois vetores, a ligação no seu uso, significado ou tema, permitindo a um modelo NLP construir novas frases em resposta ao input do seu utilizador. Deve ser notado que, devido à quantidade elevada de dimensões que os vetores de palavras tendem a ocupar, é necessário utilizar funções tais como a distância Euclidiana ou distância de Manhattan que permitem calcular, com um certo nível de precisão (embora não perfeito), esta proximidade.

2.3 TOKENIZATION

A tokenização é um passo no processo vetorização de palavras, que consiste em repartir um texto maior ainda em língua humana em várias unidades mais pequenas, os ditos 'tokens'. Dependendo do específico tipo de tokenização a ser realizado, os tokens podem variar no tamanho, sendo palavras únicas ou frases inteiras, e

também na transformação final dos tokens para serem interpretáveis pelo modelo. A arquitetura sBERT utiliza a tokenização WordPiece da Google, que transforma o token numa representação numérica em que a cada primeira letra de uma palavra são atribuídas o número de vez que esta se encontra ligada a outras letras. Esta técnica, além de tornar o dataset interpretável para o modelo, também permite a este lidar com palavras sem significado ou com erro, podendo o modelo determinar a mais provável palavra que o escritor tinha intenção de escrever, dado que o modelo pode analisar pela letras presentes quais as letras mais prováveis de estarem a seguir até obter a palavra que mais provavelmente era a intenção do escritor.

2.4 SPAM

O termo *spam* refere-se a qualquer mensagem num contexto informático que tenha o objetivo de enganar ou manipular o recetor, geralmente com o objetivo de levar este a tomar uma ação que beneficia o remetente (por exemplo, enviar dinheiro ao mesmo). O *spam* distingue-se do *phishing* pela abrangência, ambos envolvem engenharia social, mas o *phishing* é direcionado a um indivíduo ou grupo pequeno de indivíduos, tentando obter sucesso com uma mensagem mais dirigida às vitimas, enquanto o *spam* tenta ser mais abrangente na mensagem de forma a conseguir atingir um maior número de vitimas, aumentando assim a probabilidade de pelo menos uma interagir com o *spam*.

2.5 HAM

No contexto da classificação de *spam* informático, o termo *ham*, é usado como uma forma mais curta de referir a mensagens legítimas, ou seja, aquelas que não são *spam*.

2.6 MODELO CLASSIFICADOR

Um modelo classificador procura criar um algoritmo ou qualquer outra estrutura de decisão capaz de classificar com a maior certeza possível o *dataset* usado durante o seu treino (chamado *fitting* no contexto de um modelo classificador), podendo isto ser feito com o recurso a um modelo de IA ou a um sistema de tentativa e erro automático. Os mecanismos tendem a variar de modelo para modelo, por exemplo,

num classificador de Regressão Linear, é utilizada a função de Regressão Linear, em que os valores de várias constantes dentro da mesma vão sendo alteradas até ser obtido um algoritmo que satisfaça as condições de treino.

2.7 NLP

NLP ou Natural Language Processing (Processamento de Linguagem Natural) é uma das áreas abrangidas pela IA, com um foco na construção de modelos capazes de processar e interpretar a linguagem humana, permitindo a estes modelos auxiliar em tarefas como a classificação de texto (p.e. Se um email é *spam* ou não) ou a sugestão de ações (p.e. Um modelo que analisa um texto escrito pelo utilizador e oferece dicas para o melhorar).

2.8 Rede Neuronal (RN)

Uma RN é uma implementação possível de IA, que visa emular o funcionamento do cérebro humano, sendo composta por um conjunto de "neurónios"interligados, em várias camadas sequenciais. Sendo que a partir de um dado *input*, os vários neurónios vão atribuindo-lhe um valor de *output*, este vai depois sendo passado ao longo das camadas da rede, sendo refinado a cada passagem por um neurónio até alcançar a camada final aonde sai o resultado final.

2.9 Large Language Model (LLM)

LLM são uma expansão do conceito NLP a uma escala maior, enquanto os modelos NLP tendem a ser treinados com foco num tópico e/ou linguagem específica, não tendo grande utilidade fora destes. Os modelos LLM consomem durante o seu treino informação relativa a uma grande variedade de diferentes áreas e linguagens com o objetivo de criar um modelo que consiga lidar com problemas em várias diferentes áreas. Tais modelos tendem a ter requisitos bastante elevados, não só quanto a armazenamento, mas também ao nível de processamento.

2.10 ENCODER

Um *Encoder* é um algoritmo ou programa que realiza a função de *encoding*. Isto trata-se do processo de converter informação que se encontra num estado difícil ou impossível de processar para uma máquina ou programa. por exemplo, conversão de texto de linguagem humana/tokens para um vetor de palavras, isto é, um formato interpretável para a máquina ou programa.

2.11 DECODER

Decoders realizam o processo inverso dos *encoders*, convertendo os dados de formato destinado a máquinas/programas para um formato interpretável pelo ser humano.

2.12 FINE TUNING

Fine Tuning é um conceito presente na área da IA que engloba várias diferentes pequenas modificações que podem ser realizadas a um modelo de IA de forma a alterar o algoritmo ou programa deste, com o objetivo de obter resultados melhores ao adequar mais o modelo ao problema a ser resolvido por este. O artigo Zhang et al., 2020 oferece uma explicação mais detalhada do *fine tuning*.

2.13 MODELO TRANSFORMADOR

Os modelos de vetorização conhecido como transformadores são uma inovação neste campo que têm vindo a oferecer resultados excecionais na criação de modelos NLP e LLM. O modelo transformador foi inicialmente proposto em Vaswani et al. (2017) e constitui um modelo baseado no método de IA *Deep Learning* que envolve treinar o modelo com base nos mecanismos de "atenção"descrita pelo autor. Especificamente, o modelo tenta determinar as partes do *token* que têm maior interesse e merecem "atenção"para conseguir criar os *embeddings* (termo para vetores de palavras no contexto de transformadores), também descritos inicialmente nesta referência, que conseguem reter não só as ligações entre palavras, mas também a entoação, significado e contexto por detrás delas.

Entrando numa definição mais técnica, os *tokens* são transformados num *encoding* ou vetor nos neurónios de entrada, passando depois estes *encodings* iniciais por várias camadas de neurónios intermédios que irão, cada um, refinar cada vez mais o *encoding* para melhorar progressivamente a sua captura das relações contextuais do *token* a ser avaliado pelo modelo, acabando o *encoding* por sair no fim da rede com uma elevada riqueza de contexto quanto ao *token* inicial. Tal permite ao modelo que utiliza os *encodings* um entendimento mais aprofundado da linguagem.

2.14 MODELO BERT

Os modelos BERT são, em si, uma recente evolução no campo dos modelos transformadores que veio a ser revolucionária no seu impacto, com a introdução do transformador bidirecional, capaz de processar uma palavra não só num único sentido (por exemplo, analisando apenas as palavras que estão atrás desta na frase) mas em ambos, conseguindo assim determinar com maior sucesso e competência o contexto da palavra ao analisar como esta está ligada a todo o texto a ser processado e estabelecendo a ligação da palavra com todas as restantes palavras no texto, independentemente da sua posição na frase. Para dar um exemplo da utilidade e inovação destes modelos, considere-se este simples exemplo: "O cão fugiu do gato porque ele tinha medo dele". Aqui um modelo unidirecional que esteja a interpretar da esquerda para a direita poderia determinar que a palavra ele, se refere ao gato na ausência do restante contexto, mas um modelo bidirecional, com o contexto adicional dado por 'tinha medo dele' consegue determinar que o pronome 'ele' se refere ao cão.

2.15 sBERT

Central a este trabalho, é a arquitetura sBERT. Esta assenta na arquitetura BERT para aumentar a eficácia no que toca à similaridade entre frases. A arquitetura sBERT foi desenvolvida com o treino de uma RN siamesa tripla sendo os pesos resultantes depois aplicados a um modelo base BERT. O desenvolvimento foi motivado pelas dificuldades inerentes ao BERT, que não gera vetores de palavras de tamanho fixo. Esta limitação dificulta tarefas como a pesquisa por similaridade e o treino não supervisionado, pois inviabiliza operações matemáticas como a similaridade de cossenos sem processar cada palavra da frase individualmente. Para contornar este problema, a arquitetura sBERT utiliza uma RN Siamesa (por vezes

tripla) durante o treino, que lhe permite analisar a frase no seu todo. Este processo resulta na criação de um vetor final único para a frase, ao invés da abordagem palavra a palavra do BERT. Com estas melhorias, o modelo sBERT demonstrou uma eficácia melhor que o BERT Reimers e Gurevych, 2019, especialmente no que toca a custo temporal, bem como oferecendo resultados melhores comparativamente com outros modelos rivais nos testes em Reimers e Gurevych, 2019.

2.16 VALIDAÇÃO CRUZADA

A Validação Cruzada é uma forma de avaliar um modelo de classificação com um único *dataset* sem encontrar os problemas habituais tal como o *overfitting*. Primeiramente, o utilizador define o valor de K, sendo o *dataset* dividido em K partes iguais (ou *folds*, na literatura anglo-saxónica). De seguida, o processo é repetido K vezes: em cada iteração, uma destas partes é alternadamente usada como *dataset* de teste (ou validação), enquanto as restantes K-1 partes são unidas para formar o *dataset* de treino. Os resultados de desempenho são registados após cada teste. Este ciclo repete-se até que todas as K partes tenham sido usadas exatamente uma vez como *dataset* de teste, fornecendo assim K conjuntos de resultados que são, tipicamente, usados para calcular uma média final do desempenho do modelo. A utilidade deste método de avaliação está na sua automatização sendo as K divisões feitas automaticamente e na sua capacidade de contrariar o *overfitting* ao permitir ao analista ver se os resultados do modelo são positivos independentemente da combinação de *datasets* usados ou se apenas ocorrem com específicas combinações.

3

TRABALHO RELACIONADO

Sendo a deteção de *spam* com recurso a sistemas de IA um assunto que já teve interesse por parte da comunidade cientifica durante vários anos, existe uma elevada variedade de trabalhos que forneceram valioso conhecimento não só ao nível teórico mas também prático para a realização deste trabalho, sendo nas seguintes secções mencionados vários estudos científicos que contribuíram para este trabalho.

3.1 HISTORIA E CARACTERÍSTICAS DO *SPAM*

Em Karim et al. (2019), os autores realizam uma análise ao estado atual do *spam*, recorrendo a estatísticas e fontes fiáveis para demonstrar o constante crescimento desta ameaça, como a perda de 107 milhões de dólares australianos no ano de 2018 devido a vários diferentes tipos de ataques de *spam*. Com um aumento anual médio de 9 milhões de dólares. Apesar dos esforços concentrados de várias organizações e governos para combater o *spam*, o problema persiste. Diversas soluções já foram sugeridas e implementadas globalmente. O trabalho faz previsões sobre a futura evolução do *spam* e das técnicas para o combater. Além disso, explica várias características menos conhecidas do *spam*, como as diferentes vertentes de ataque e as técnicas utilizadas. O estudo teve utilidade para estabelecer a motivação deste projeto, dado que o mesmo fornece informações sobre os possíveis vetores de ataque que podem ser utilizados no spam, auxiliando na componente prática do presente projeto.

Em S. Kumar e Gupta (2022) os autores realizam um estudo à evolução do *spam* como ameaça no contexto do Twitter (agora conhecido como X) e correio eletrónico nos cinco anos precedentes ao estudo, de 2017 a 2022. O estudo focou não só a perspetiva histórica em si mas também a evolução técnica do *spam*, incluindo os vários vetores utilizados pelos atacantes para desenvolver o *spam* de forma a melhor manipular o alvo, aumentar a possibilidade de interação com o *spam* e evitar deteção. Aborda também a evolução dos métodos de deteção e combate ao *spam*, demonstrando a constante corrida de armas entre estes dois lados. Este trabalho revelou-se útil ao oferecer uma visão detalhada da evolução do *spam* e das

várias diferentes formas em que este se demonstra no presente, conhecimentos úteis para treinar os modelos implementados neste trabalho. Adicionalmente, este estudo também serviu como um passo inicial para identificar o estado da arte atual do combate ao *spam*, apesar da sua publicação em 2022.

3.2 COMBATE AO *SPAM* COM IA EM GERAL

O trabalho Malode et al. (2024) teve importância, dado que este aborda a aplicação de algoritmos de IA a *datasets* utilizando a biblioteca NTLK para tentar obter o melhor modelo possível para classificação de *spam*, embora a linguagem de programação usada não tenha sido esclarecida pelos autores, pode-se inferir pelo uso da biblioteca NLTK que terá sido realizado em Python. Os autores usam a precisão e *accuracy* como métricas de avaliação dos modelos, uma abordagem também seguida no âmbito deste relatório de projeto. A metodologia seguida em Malode et al. (2024) acabou por servir de guia para este trabalho.

Outro estudo científico que entra neste tema foi Siddique et al. (2021). Os autores recorreram ao Python e a classificadores *Naive Bayes*, Support Vector Machines (SVM), LSTM e CNN, para classificar um *dataset* de *spam* na lingua Urdu, procurando criar um novo modelo para essa pouco explorada língua indo-iraniana, especialmente, quando escrita de forma nativa. Para o efeito, os autores traduziram vários *datasets* ingleses do Kaggle, obtendo 5000 emails que utilizaram no treino do modelo. O classificador LSTM obteve os melhores resultados, alcançando uma métrica de 98.4%. Este trabalho também explicou de forma esclarecedora a metodologia que foi seguida pelos autores para desenvolver os modelos finais, tendo isto, juntamente com informação quanto à obtenção de *datasets* e métricas mais adequadas para avaliar os modelos, sido a principal utilidade para este trabalho a aplicação de IA ao spam.

Em N. Kumar, Sonowal et al. (2020) são aplicados vários vetorizadores de machine learning, incluindo NLP, juntamente com diversos classificadores para avaliar o melhor par possível de vetorizador com classificador para obter os melhores resultados possíveis, tendo-se destacado três classificadores pelos seus resultados: O modelo *Random Forest*; Uma RN de *Perceptron* Multi-camada; Uma Rede de *Bayes*. Para além dos resultados, a mais valia do referido estudo científico está na metodologia empregue, com destaques para as várias combinações de testes, juntamente com as sugestões para trabalhos futuros.

3.3 USO DE MODELOS NLP PARA COMBATER O *SPAM*

No trabalho de Al Saidat et al. (2024), é abordada a implementação de modelos NLP na deteção de *spam* em SMS. Embora o foco em SMS seja diferente do deste trabalho, o artigo oferece um conhecimento abrangente da utilidade que estes avanços tecnológicos têm para a deteção de *spam*, demonstrando, sempre com o apoio de vários outros estudos citados, os resultados destas soluções na deteção de *spam*. O estudo reforça também a crescente necessidade para tais melhorias dada a evoluções dos vetores de ataque por *spam*, fornecendo sugestões para potenciais áreas de foco em trabalhos práticos que poderão resolver os problemas atuais que estão a limitar a eficácia destas tecnologias. A utilidade deste estudo veio do fornecimento de um melhor entendimento do desenvolvimento de um modelo de NLP e a sua orientação ao *spam*.

Também de interesse neste tópico foi Crawford et al. (2015). Este estudo apresenta o estado da arte até ao ano de publicação, 2015, com um foque em produtos comerciais disponíveis online e que recorreram à IA e a modelos NLP para deteção de avaliações falsas de produtos. O estudo demonstra os vários positivos e também as falhas destas implementações, concluindo com sugestões para a metodologia para futuras investigações da área, de interesse para este trabalho foi a análise em si para determinar as avenidas mais promissoras a explorar neste projeto, bem como a parte das recomendações que serviu para definir a metodologia a seguir neste trabalho.

3.3.1 *Modelos transformadores*

É também destacado Guo et al. (2023). Neste é abordado o uso de um transformador bidirecional, baseado na arquitetura estado da arte Bidirectional Encoder Representations from Transformer (BERT). Para o efeito é empregue um modelo pré-treinado nesta arquitetura para formar os vetores (chamados embeddings no contexto de transformadores) a serem usados depois com modelos de classificação IA. A vetorização com transformadores bidirecionais é uma técnica que tem vindo a trazer grandes avanços à interpretação de linguagem humana em IA, pelo que a sua implementação no combate ao *spam* foi um avanço significativo neste. Neste estudo foi utilizado um *dataset* com 36715 emails e a implementação, tal como a maioria dos restantes trabalhos neste capitulo foi realizada em Python. Os resultados de Guo et al. (2023) justificam o uso de modelos transformadores, dado que os

resultados superaram os de modelos convencionais também testados. O estudo é ainda pertinente pela explicação que providencia de conceitos relevantes para o trabalho deste projeto.

Jamal e Wimmer (2023) recorreram também à família de modelos BERT para a deteção de *spam*. Os autores desenvolveram o Improved Phishing and *spam* Detection Model (IPSDM), um modelo LLM com foco na deteção de *spam*. Este modelo LLM representa um avanço no conceito da IA NLP que tenta criar uma IA capaz de lidar com vários tópicos e problemas com alta perícia em vez de estar dedicada a um tema. Esta IA é baseada nos modelos DistilBERT e RoBERTA (ambos variações simplificadas do modelo BERT) com fine tuning extensivo e treino extra realizado pelos autores para obter o modelo final que oferece melhores resultados na classificação de *spam* comparado com os originais. Neste projeto, os autores usaram um *dataset* de 5761 emails com uma maioria de Ham, ao qual aplicaram *adaptive synthetic sampling*, uma técnica de *sampling* que corrige desequilíbrios entre *labels* ao gerar novos dados para os *labels* menos populados do *dataset* com base nos dados pré-existentes desse *label*. Como a maioria dos trabalhos práticos aqui revistos, os autores fizeram uma divisão 80/20 entre os dados de treino e teste com o *dataset* de treino sendo depois também dividido em 75/25 entre o *dataset* final de treino e o *dataset* de validação. Embora os autores não tenham mencionado a linguagem de programação usada, as imagens incluídas no relatório que mostram o código aparentam o uso de Python para programar o modelo. Não obstante o uso de LLM se afaste do foco em NLP deste projeto, Jamal e Wimmer (2023) forneceu vários conhecimentos adicionais sobre trabalhar com modelos transformadores que tiveram utilidade neste projeto.

O trabalho mais central a este projeto no entanto, foi Reimers e Gurevych (2019), este estudo relatou o desenvolvimento pelos autores da arquitetura sBERT ou Senteance Bidirectional Encoder Representation from Transformer (sBERT). Sendo uma inovação sobre o próprio modelo BERT que utiliza redes neuronais siamesas para otimizar o seu funcionamento com tarefas como a pesquisa por similaridade ou de aprendizagem não supervisionada, para apoiar as descobertas, os autores mostraram como o BERT demorou 65 horas a realizar uma tarefa de pesquisa por similaridade entre dez mil (10000) frases requerendo cerca de 50 milhões de comparações enquanto o modelo sBERT requereu apenas 5 segundos. Um dos maiores problemas do BERT nesta área vem do facto de este não criar vetores de tamanho fixo, causando problemas com a comparação semântica entre frases que causam perdas de tempo. Por sua vez, o sBERT com a sua RN siamesa consegue lidar com este problema ao manter, desde o início, os vetores limitados a um tamanho

fixo, o que possibilita o uso de métodos de pesquisa computacionalmente mais eficazes tais como a distância de Manhattan, distância Euclidiana ou a similaridade de cossenos para encontrar frases similares mais rapidamente que um modelo BERT.

Tabela 1: Principais Conclusões

Estudo	Principais Conclusões
Karim et al., 2019	• A combinação de vários métodos de detetar spam é melhor que o uso de apenas uma solução. • O uso de IA melhora a deteção do spam. • A necessidade de a deteção de spam continuar a evoluir e acompanhar a evolução do próprio spam.
S. Kumar e Gupta, 2022	• A deteção de spam como um processo em constante evolução. • O uso de IA melhora a deteção do spam.
Malode et al., 2024	• IA é mais eficaz no combate ao spam. • Combinação de múltiplos classificadores para melhores resultados.
Siddique et al., 2021	• A necessidade do tratamento dos dados para obter modelos eficazes. • Modelos *Deep-Learning* oferecem melhores resultados com maior custo temporal. • Soluções existentes demasiado focadas no alfabeto Latim.
N. Kumar, Sonowal et al., 2020	• A utilidade de IA na deteção de spam.
Al Saidat et al., 2024	• A utilidade de IA na deteção de spam. • Modelos de *Deep Learning* como o BERT são uma evolução positiva na deteção de spam. • A necessidade de classificação em tempo real.
Crawford et al., 2015	• A utilidade de IA na deteção de spam. • Demasiado foco por outros trabalhos em aprendizagem supervisionada, modelos não supervisionados mostram resultados promissores. • A necessidade do tratamento dos dados para obter modelos eficazes.
Reimers e Gurevych, 2019	• Desenvolvimento da arquitetura sBERT. • Demonstra as melhorias desta sobre os modelos competidores. • Formas de utilizar esta arquitetura para vários problemas, incluindo classificação.

4

METODOLOGIA

Para a implementação prática deste trabalho, houveram várias escolhas que tiveram que ser feitas, tais como definir os objetivos a alcançar, o modelo vetorizador específico e classificadores a utilizar, a metodologia de treino a seguir, bibliotecas específicas a usar bem como a aquisição de *datasets* para treinar os classificadores.

É importante mencionar que o código desenvolvido neste projeto encontra-se no seguinte repositório: https://github.com/labcif/AI4spam#.

4.1 OBJETIVOS A ALCANÇAR

Com este projeto, o autor tem dois objetivos em mente, primeiro, o desenvolvimento de um modelo classificador apoiado por tecnologia IA NLP capaz de distinguir emails legítimos de emails maliciosos com resultados satisfatórios, sendo que "satisfatórios"neste contexto, representam um baixo número de falsos positivos (emails legítimos classificados como spam), uma taxa de acerto ($\frac{Classificaçõescorretas}{totaldeclasificações}$) elevada que ultrapasse, pelo menos, os 90% e, por fim, ser suficientemente rápido para permitir classificar em tempo real. Neste âmbito,a metodologia foi definida com base na revisão da literatura e na consideração dos objetivos deste projeto.

4.2 FERRAMENTAS UTILIZADAS

Foi decidido utilizar computação em nuvem fornecida pela Google através do serviço Google Colab. Este é um serviço por subscrição com opções gratuitas que fornece computação de *notebooks Jupyter* em *Python*, que foi a implementação seguida para este trabalho. Para uma visão sumariada das ferramentas utilizadas, a Tabela 2 mostra o hardware utilizado, a Tabela 3 lista o software utilizado, enquanto a Tabela 4 indica as bibliotecas utilizadas.

No que toca ao hardware em si, o Google Colab oferece algumas opções gratuitas, incluindo GPUs da NVIDIA. Após alguns testes que demonstraram um aumento considerável em desempenho e velocidade do treino com GPU em comparação

com apenas CPU, foi escolhido o primeiro para a maioria dos testes, no entanto, a biblioteca utilizada para classificadores em GPU, cuML, não oferecia um classificador Extra Trees, pelo que os testes com este classificador especificamente foram realizados utilizando a biblioteca scikit-learn.

No que toca ao hardware em si, o Google Colab fornece algumas opções gratuitas, incluindo GPUs, sendo que a máquina utilizada para correr os modelos durante o treino e testes pode ser vista na Tabela 2

Tabela 2: Hardware Utilizado

Hardware	Específicos
CPU	Intel Xeon com 2 Cores virtuais a correr a 2.2 GHz
RAM	12,7 GiB
Armazenamento	112,6 GiB
GPU	NVIDIA Tesla T4 com 15GiB de RAM

4.3 SOFTWARE UTILIZADO

Para o desenvolvimento do modelo de classificação em *Python*, foram utilizadas várias bibliotecas específicas ao desenvolvimento de IA, vetorizadores NLP, classificadores e a utilidade Jupyter *notebook* para facilitar o desenvolvimento.

4.3.1 *Plataforma*

Devido à sua robustez e versatilidade, bem como à vasta comunidade de programadores de IA que fornece apoio e bibliotecas de código aberto, incluindo vetorizadores de última geração, a linguagem *Python* foi considerada a mais adequada para este projeto.

4.3.1.1 *Jupyter Notebook*

Os *Notebooks Jupyter* são utilitários de programação desenvolvidas pela *Project Jupyter* que foram utilizadas na elaboração deste trabalho. Estes são plataformas online *open-source* que podem ser instaladas e corridas localmente (no caso deste trabalho, foi utilizada a plataforma Google Colab), que são utilizadas para elaborar e executar em tempo real código nas linguagens *Python*, R ou *Julia*. Estes *Notebooks* são especialmente úteis para o desenvolvimento de programas orientados à analise de

dados ou desenvolvimento de IA. Tendo em conta a sua utilidade para desenvolver IA bem como a maior facilidade em testar o modelo classificador com a execução em tempo real, foi decidido que esta plataforma seria utilizada para desenvolver a componente prática deste trabalho em Python.

Tabela 3: Software Utilizado

Software	Versão	Motivos para escolha
Python	3.10	• Linguagem Versátil • Comunidade de IA dedicada • Várias bibliotecas de desenvolvimento de IA
JupyterLab *notebook*	N/A	• Plataforma útil para desenvolver modelos de análise de dados • Existem várias plataformas gratuitas de computação *Cloud* para treinar (p.e. Google Colab) • Funções ou partes do código podem ser testadas individualmente sem correr o código todo

4.3.2 *Bibliotecas*

Para desenvolver o código utilizado neste projeto, foram utilizadas diversas bibliotecas para simplificar o processo de desenvolvimento ao oferecerem funções já feitas, permitindo o foco principalmente no código mais relevante ao trabalho, a seguinte secção descreve as diversas bibliotecas usadas para este fim.

4.3.2.1 *Nativo*

Além das bibliotecas gerais como o Pandas e Numpy, que serviram para armazenar os dados, foram centrais as bibliotecas SciKit-Learn e Sentence Transformers ao desenvolvimento da IA deste trabalho.

A biblioteca SciKit-Learn é uma das mais populares no desenvolvimento geral da IA, fornecendo ferramentas para desenvolver diversos géneros de IA, incluindo vetorizadores, como o TF-IDF e Hash Vectorizer usados neste trabalho, classificadores, tendo todos os classificadores deste trabalho vindo desta mesma e também ferramentas que fornecem métricas de avaliação da IA tais como a revocação (Per-

centagem de Emails Maliciosos corretamente classificados em relação ao total de Emails Maliciosos) e F1 Score ($2 \cdot Precisão \cdot Revocação / Precisão + Revocação$) e $F\beta$ Score, bem como temporizadores e avaliadores de performance computacional, permitindo avaliar a competência da IA desenvolvida.

A biblioteca Sentence Transformers é uma implementação prática da arquitetura sBERT, sendo esta uma modificação da arquitetura estado da arte BERT destinada a formar vetores densos para frases. Esta biblioteca fornece vetorizadores transformadores que providenciam uma interpretação do texto mais completo com a inclusão de perceção do contexto e sentimento por detrás de dada mensagem de forma a permitir detetar spam mesmo quando este tenta imitar a linguagem de uma mensagem inofensiva através de indicadores subtis no contexto e sentimento por detrás da mesma. Adicionalmente, os vetorizadores sBERT são versáteis, podendo os seus *embeddings* ser usados para várias diferentes tarefas sem ser necessário voltar a vetorizar. É também possível combinar os resultados destes com classificadores de outras bibliotecas como a SciKit-Learn, característica que facilitou consideravelmente essa etapa deste projeto.

4.3.2.2 *Aceleração com GPU Nvidia*

A utilização de uma GPU da NVIDIA permite acelerar o processamento de dados através de bibliotecas alternativas desenvolvidas pela RAPIDS. Estas bibliotecas substituem o Pandas e o NumPy pela biblioteca cuDF, pois foram concebidas para interagir diretamente com a *framework* CUDA da NVIDIA. Isto maximiza a capacidade de processamento das placas gráficas da empresa, mas limita a sua utilização a máquinas que possuam hardware NVIDIA.

A biblioteca cuML foi utilizada para substituir a biblioteca SciKit-Learn, estando construída sobre o mesmo API que a biblioteca SciKit-Learn. Esta oferece a grande maioria das ferramentas oferecidas pela mesma, com a exceção do classificador extra trees em que foi necessário manter o classificador da SciKit-Learn, excluindo isto, todas as ferramentas, incluindo o pipeline, as métricas de avaliação, os restantes classificadores são oriundos da biblioteca cuML.

A biblioteca sBERT já faz uso de aceleração por GPU por omissão, não sendo necessária a sua substituição, embora seja necessário implementar também armazenamento de variáveis em bibliotecas que utilizam aceleração por GPU NVIDIA, como a cuDF, para ter um funcionamento mais rápido.

Uma sucinta descrição das bibliotecas empregues neste projeto encontra-se na Tabela 4.

Tabela 4: Bibliotecas Utilizadas

Biblioteca	Motivos para escolha / Uso
Sci-Kit Learn	• Utilizada para classificador Extra Trees, avaliação de resultados, métricas e gráficos • Funções para calcular métricas de classificação e desenhar gráficos dos resultados • Disponibiliza vários modelos classificadores • Disponibiliza algumas utilidades para testar classificadores, p.e. Validação Cruzada
Sentence Transformers	• Utilizada para obter modelo vetorizador sBERT pré-treinado e vetorizar *dataset* • Oferece vetorizadores IA pré-treinados de vários tamanhos diferentes • Oferece a possibilidade de finetuning para adequar vetorizadoes a problemas específicos • Modelos vetorizadores utilizam GPU, acelerando o processamento dos dados
PyTorch	• Computação Tensor (útil para GPUs do Google Colab) • Utilizada para usufruir de aceleração por GPU
NumPy	• Utilizada para armazenar os vetores • Quando combinada com a PyTorch, pode usufruir de aceleração por GPU • Oferece algumas transformações úteis vetores
Pandas	• Utilizada para armazenar dados tanto numéricos como nominais • Leitura de ficheiros csv para aceder ao *dataset* • Armazenamento de dados em formatos interpretáveis por todos os vetorizadores e classificadores utilizados
cuDF	• Utilizada para converter *datasets* em .csv para listas e conversões entre Pandas e NumPy • Análoga à biblioteca Pandas mas focada específicamente em aceleração com GPU • Fácil conversão entre bibliotecas Pandas, NumPy e cuDF
cuML	• Utilizada para obter classificadores usados (exceto Extra Trees) • Implementação quase total da componente de classificadores e vetorizadores da biblioteca Sci-Kit Learn com aceleração por GPU • Disponibiliza a maioria dos classificadores da Sci-Kit (com exceções como o Extra Trees) • Mais fácil de utilizar do que a combinação de Sci-Kit Learn com PyTorch para obter classificadores que usufruem da aceleração por GPU

4.4 METODOLOGIA A SEGUIR

A Figura 1 documenta a metodologia do treino

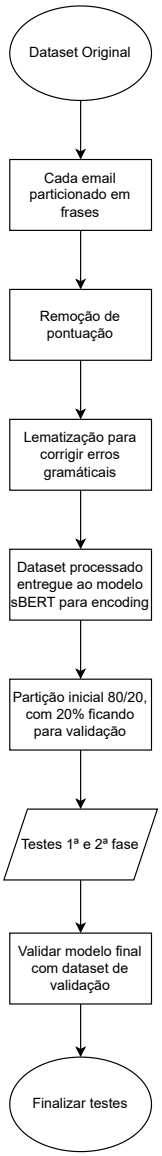


Figura 1: Diagrama com o processo de treino (ver Fig 3 para testes)

Para evitar criar um modelo classificador com fraca capacidade de lidar com novos dados que sejam diferentes dos de treino, foi decidido que o *dataset* de treino deveria ter pelo menos 50 mil emails. Para esse efeito, recorreu-se à plataforma *Kaggle* onde se obtiveram uma série de três *datasets* com um total de 90 mil emails.

Para testar o modelo classificador final, o *dataset* é inicialmente dividido numa porção de 80% para treino e 20% para teste. Esta divisão garante que os classificadores são avaliados com dados que não foram utilizados durante a fase de treino.

Inicialmente, o *dataset* passa pelo processo de tokenização em que o texto de cada email é particionado usando a tokenização *WordPiece* contida no próprio modelo sBERT. A tokenização procura prováveis terminações de frase tais como pontos finais ou espaços de tamanho maior que 1 no texto, particionando a partir destes as frases do email. Posteriormente, os *tokens* resultantes passam pelo processo de *encoding* e são transformados em *tensors* que o modelo consegue interpretar mais facilmente do que a linguagem comum. De seguida, é inserido o *padding* para garantir que os *tensors* tenham um tamanho pré-definido. Estes são então submetidos ao processo de vetorização, que gera os *encodings* finais (embeddings) de cada palavra. Finalmente, estes *embeddings* passam pelo processo de *pooling*, onde os *encodings* de palavras individuais são combinados para formar o *encoding* de toda a frase. Estes *embeddings* finais consistem em vetores de tamanho fixo (tipicamente 384 valores, ajustável nas configurações)

Por fim, os vetores obtidos do modelo vetorizador são introduzidos juntamente com os *labels* aos classificadores para realizar o *fitting*, ou treino, destes, isto consiste na geração de várias diferentes variantes do classificador a treinar com base no *dataset* de treino, sendo que estas várias iterações são comparadas utilizando os *labels* já conhecidos para determinar o melhor classificador de entre os candidatos. Os classificadores usados neste trabalho são cinco: *Support Vector Machine*, *Logistic Regression*, *K-Nearest Neighbours*, *Random Forest* e *Extra Trees*. Estando os classificadores treinados, é depois passado por estes uma pequena porção do *dataset* original que tinha sido previamente excluída dos dados de treino especificamente para servir depois para testar os modelos classificadores finais com dados novos.

4.5 A ESCOLHA DO sBERT SOBRE O BERT

Tendo em conta as explicações oferecidas no Capitulo 2, é agora necessário abordar qual a utilidade do sBERT e o que o torna uma melhor escolha que o BERT,

especialmente para este trabalho. Numa tarefa de pesquisa por similaridade entre duas frases, o BERT irá sempre ter que vetorizar o par como um conjunto para realizar a comparação, pelo que para se poder comparar todas as frases num *dataset* de n dados, terão que ser feitos embeddings para cada par possível: $n - 1!$. O sBERT, pelo outro lado, processa cada frase independentemente das restantes, produzindo encodings que podem depois ser comparados entre si sem requerer nova geração de encodings do par de frases em questão, esta diferença torna o sBERT superior em grandes *datasets* em que a necessidade de fazer encoding de cada par de frases, o número de embeddings a criar para um *dataset* de tamanho n seria, simplesmente, n. Embora para valores pequenos de n estas duas formulas não estejam particularmente distantes uma da outra, à medida que n vai aumentado, o número de embeddings que o modelo BERT teria que fazer irá ter um aumento exponencial enquanto o modelo sBERT terá um aumento linear que será consideravelmente mais pequeno quanto mais n aumentar. Para o desenvolvimento de um modelo cuja finalidade irá envolver um grande número de comparações, como é o caso deste projeto, a arquitetura sBERT irá oferecer um custo temporal bastante menor comparativamente a um modelo da arquitetura BERT. Concluindo, para os fins deste trabalho, que consiste em desenvolver um modelo classificador, tarefa que irá necessitar de comparações entre vetores, a capacidade de gerar estes com tamanho fixo do sBERT irá permitir poupar mais tempo do que se fosse utilizado o modelo BERT.

4.6 MÉTRICAS DE AVALIAÇÃO DO MODELO FINAL

Tendo sido realizado o desenvolvimento dos classificadores, é depois necessário determinar de forma empírica o desempenho destes, isto será feito com recurso a várias métricas de avaliação específicas a modelos classificadores, tais como a taxa de acerto, bem como métricas mais gerais, como o tempo que o classificador demorou a processar os dados.

4.6.1 *Conceitos base*

Para entender as métricas que se seguem, devem ser compreendidos primeiro quatro conceitos que formam a base da maioria das métricas relacionadas com problemas de classificação: Verdadeiros Positivos, Verdadeiros Negativos, Falsos Positivos e Falsos Negativos. Usando a deteção de spam como exemplo, os Verdadeiros Positivos (VP) são *emails* que são spam e foram corretamente identificados como tal. O Falso

Positivo (FP) é o oposto: uma mensagem legítima classificada erroneamente como spam. Inversamente, o Verdadeiro Negativo (VN) corresponde à classificação correta de um email legítimo como tal, enquanto o Falso Negativo (FN) ocorre quando uma mensagem de spam é classificada incorretamente como legítima.

4.6.2 *Métricas de classificação*

Para avaliar o modelo classificador final é primeiro preciso determinar quais as métricas mais relevantes a este problema, dado que existe uma grande variedade de métricas que podem ser usadas mas nem todas são úteis para os mesmos problemas. No que toca à questão de detetar spam, uma taxa de falsos positivos (emails legítimos classificados como spam) baixa é mais importante que a taxa de verdadeiros positivos (emails spam classificados como spam), isto devido à possibilidade de os emails legítimos classificados como spam poderem ser extremamente importantes ao recetor, dando um custo elevado ao falso positivo, pelo que, se for necessário fazer sacrifícios numa destas, deve ser preferida a taxa de verdadeiros positivos ($\frac{Emailsmaliciosos}{Todososemails}$).

4.6.3 *Precisão*

Para este problema então, as métricas mais importantes são a taxa de falsos positivos, seguida da taxa de verdadeiros positivos, já explicadas previamente. É também importante a métrica de precisão:

$$\frac{\text{VP}}{\text{VP} + \text{FP}} \tag{1}$$

Esta consiste no número de verdadeiros positivos sobre a soma de verdadeiros positivos com falsos positivos. Esta métrica é extremamente útil para problemas em que falsos positivos têm um elevado custo, sendo que ela é diretamente proporcional à taxa de falsos positivos, pelo que é uma das métricas mais úteis para os problemas do classificador a desenvolver.

4.6.3.1 *Fβ-Score*

Outra métrica importante é o *Fβ-Score*:

$$(1 + \beta^2) \cdot \frac{\text{precisão} \cdot \text{revocação}}{\beta^2 \cdot \text{precisão} + \text{revocação}} \tag{2}$$

Uma modificação do *F1-Score*:

$$\frac{\text{precisão} \cdot \text{revocação}}{\text{precisão} + \text{revocação}} \tag{3}$$

Com um peso β adicionado à precisão, consistindo na multiplicação da precisão sobre a revocação:

$$\frac{VP}{VP + FN} \tag{4}$$

Sobre a soma destes, na sua forma original, esta métrica dá valor igual aos verdadeiros positivos e falsos positivos, algo que não se aplica na classificação ao spam, pelo que é inserido o valor de β como peso para a precisão na equação de forma a aumentar o efeito desta para a equação, o que aumenta também o efeito de falsos positivos na equação, dando uma métrica mais útil para os fins deste projeto.

4.6.4 *Gráficos de classificação*

Por fim, para demonstrar de forma visual e mais fácil de interpretar os resultados serão utilizador alguns gráficos, principalmente, a matriz de confusão e a curva AUC-AOC.

4.6.4.1 *Matriz de Confusão*

A matriz de confusão é uma tabela ou matriz utilizada para demonstrar os resultados de um modelo de classificação, ela demonstra o número e percentagem de previsões corretas e incorretas pelo classificador, ou seja, demonstra os verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos e as suas respetivas taxas, sendo extremamente útil para visualizar estes rapidamente.

Tabela 5: Exemplo de Matriz de Confusão

		Esperado	
		Positivo	Negativo
Previsão	Positivo	Verdadeiro Positivo	Falso Positivo
	Negativo	Falso Positivo	Verdadeiro Negativo

4.6.4.2 *Curva AUC-ROC*

Uma curva de AUC ROC (Area Under the Curve-Receiver Operating Characteristic) é um gráfico que demonstra a competência do modelo classificador em distinguir entre os *labels* ou classes em análise (spam e ham no caso deste projeto), consistindo em dois componentes: O ROC que é uma representação da taxa de verdadeiros positivos sobre a taxa de falsos positivos, sendo que o seu valor demonstra o quão bem o classificador consegue classificar novos dados à medida que o limiar de decisão é alterado. Segundo é o valor AUC, este é definido pelo analista que está a rever o modelo classificador e serve de referencia visual para o desempenho do classificador, um AUC de 0,5 permitiria ver como o modelo classificador se compara com um classificador aleatório, sendo que uma uma curva ROC que está acima deste AUC indica que o classificador é superior ao classificador aleatório, enquanto o contrário indicaria que o modelo classificador é de qualidade inferior ao classificador aleatório.

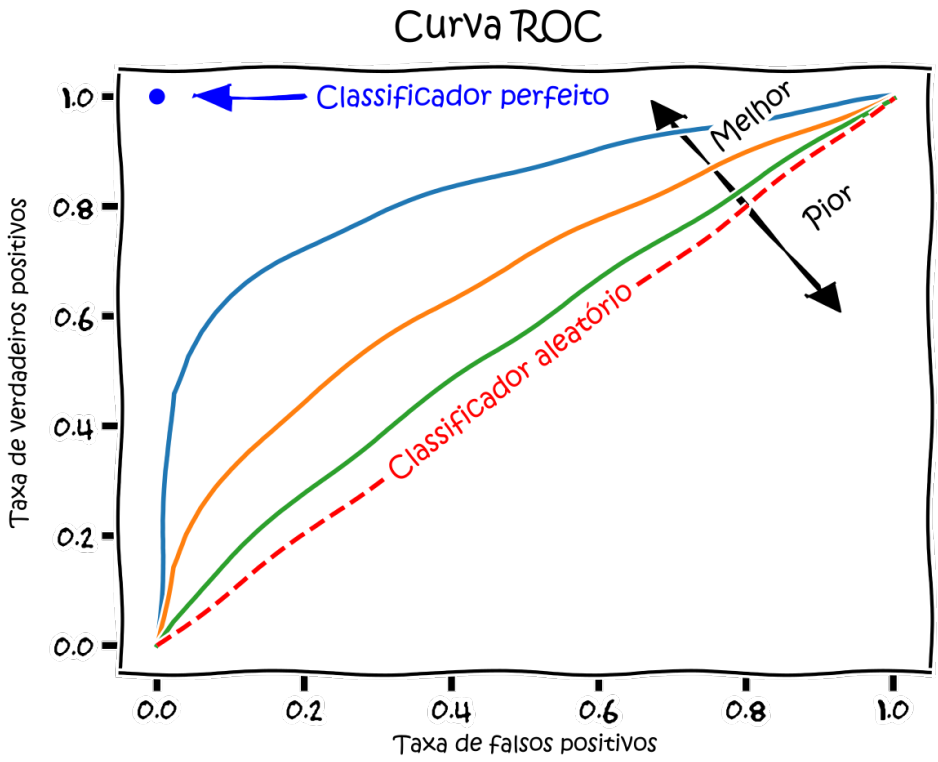


Figura 2: Ilustração de curva ROC e AUC 0,5 (tracejado vermelho) Dcbmariano, Acedido: 14-04-2025

4.7 PLANO DE TESTES E MELHORIA DE RESULTADOS

Para alcançar o classificador final, irá haver uma fase na metodologia que irá constar de um alternar entre testes do modelo classificador seguidos de tentativas de melhorar estes resultados com modificações a partes do classificador tais como os parâmetros do classificador.

4.7.1 *Testes e validação cruzada*

Antes de tocar na fase de testes é preciso esclarecer o porquê de serem necessários testes com *datasets* variados e diferentes entre si, bem como o que é a validação cruzada: No processo de treinar um classificador apoiado por IA, é possível que o *dataset* possua características, exemplificando, se uma grande percentagem dos dados do *dataset* incluírem uma vetor que não é, na realidade, particularmente relevante para os objetivos do classificador mas, por pura coincidência, está contida na grande maioria dos dados para uma das classes, é possível que estes acabem por afetar os valores finais do vetor devolvido, levando depois à criação um modelo classificador ao seu *dataset* de treino e é incapaz de lidar com novos dados ou, pelo menos, obter resultados parecidos com os obtidos com o *dataset* de treino. Uma forma de evitar este problema, conhecido como *overfitting* é testando o classificador com vários *datasets* diferentes, de modo a se verificar se o modelo classificador está a oferecer resultados bons geralmente ou se estes são apenas para um dos *datasets* ao qual o classificador ficou *overfitted*, note-se aqui que pelo termo '*datasets* diferentes' não quero dizer que se tem que obter de facto *datasets* inteiramente diferentes, de facto, é possível fazer divisões e baralhos com o mesmo *dataset* para obter *datasets* que aparentam ser diferentes entre para o propósito de treinar um modelo classificador. Uma das técnicas para realizar esta tarefa é a validação cruzada. A validação cruzada é uma técnica de testar um modelo classificador com vários possíveis *datasets* através da divisão e baralho do mesmo *dataset* original. Para explicar o funcionamento desta técnica, pode-se usar o seguinte exemplo: O analista define primeiro quantas validações (divisões) deseja fazer, suponhamos que este número é 5 para o fim desta explicação, o *dataset* é então dividido em cinco partes de tamanho igual, sendo que uma destas partes será usada como *dataset* de teste e as restantes como *dataset* de treino, para cada iteração, estas partes são trocadas, até que todas as partes tenham sido usadas para teste e treino, resultando em, efetivamente, cinco conjuntos de *datasets* treino e teste, todos eles com uma parte

dos dados diferente dos restantes. Isto irá permitir testar um classificador sem ter que procurar novos *datasets* para servirem de *dataset* de treino para verificar se o classificador não está a sofrer de *overfitting*.

4.7.2 *Plano de testes*

O plano de testes irá consistir em duas fases, uma primeira em que os testes são feitos com uma só versão dos *datasets* de teste e treino (ou seja, sem validação cruzada ou qualquer outro tipo de sorteio dos dados) e uma segunda que já se usa validação cruzada.

4.7.2.1 *Primeira fase*

Na primeira fase os testes serão principalmente para avaliar o uso de recursos e tempo que o modelo classificador demora, dado que estes testes só usam um *dataset*, não são tão úteis para avaliar a capacidade de classificar devido ao perigo de *overfitting* aos dados de treino, no entanto, os resultados destes também serão incluídos no capitulo de resultados. O custo temporal será avaliado através do próprio *notebook* Jupyter que indica quanto tempo a execução de uma célula durou, enquanto os recursos computacionais usados pelo classificador serão testados com recurso a ferramentas da biblioteca X, por fim, os resultados da própria classificação serão obtidas a partir da biblioteca Sci-Kit Learn que oferece um módulo 'metrics' repleto de ferramentas que realizam as operações matemáticas para determinar as diversas métricas a explorar de forma automatizada.

4.7.2.2 *Segunda fase*

A segunda fase consistira em testes já com validação cruzada, em geral, é desejável um número elevado de divisões de forma a poder garantir que o modelo classificador lida com novos dados bem, no entanto, é necessário ter em conta que um maior número de divisões leva a um maior custo temporal. Para este efeito, foi revisto o trabalho de Oyedele, 2023 em que foram realizadas diversos testes acabando por identificar que entre 10 a 20 iterações oferece o número ideal de resultados, com mais iterações levando à subestimação do classificador nos resultados e menos levando a um número pequeno demais de resultados para obter conclusões. Não tendo sido notados negativos quanto a tempo ou o número de resultados com 10 iterações, foi escolhido este número para os testes deste projeto. Para realizar os testes de

validação cruzada, será utilizada a solução oferecida pela biblioteca Sci-Kit Learn que, além de fornecer modelos classificadores úteis para este trabalho, também oferece várias ferramentas para testar os mesmos. A cada alteração feita aos modelos, estes irão depois passar por uma validação cruzada de 10 iterações, sendo verificada a média dos resultados dos classificadores para verificar como as alterações feitas estão a afetar o modelo final.

A Figura 3 documenta as duas fases de teste.

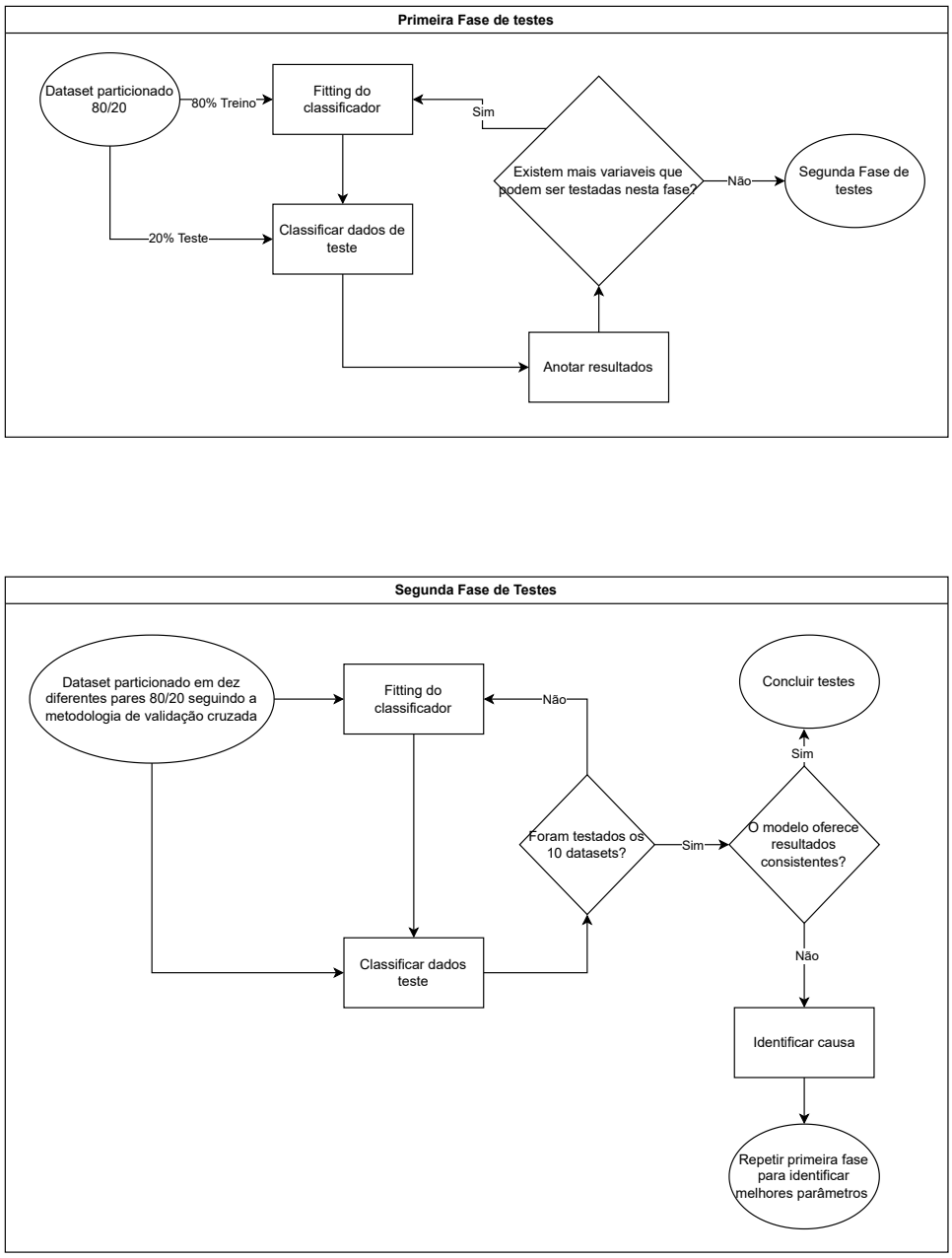


Figura 3: Diagramas das duas fases de teste

4.7.3 *Melhoria de Resultados*

Determinado pelos testes que os resultados ainda têm espaço para melhorar, existem duas principais componentes do processo de criar um modelo de classificação que podem ser alteradas: A fase de vetorização pode ser feito *fine-tuning* ou na fase de classificação em que as definições dos classificadores também podem ser alteradas para obter um resultado de maior qualidade.

4.7.3.1 *Definições do vetorizador*

Esta abordagem não foi adotada porque a única alternativa para melhorar os resultados através do vetorizador passaria pelo *fine-tuning* de um modelo sBERT existente. Essa tarefa, que permitiria especializar o modelo na deteção de *spam*, seria demasiado morosa no contexto deste trabalho, especialmente devido aos limitados recursos computacionais.

4.7.3.2 *Definições dos Classificadores*

Os classificadores empregues são modelos de classificação que possuem várias características que podem ser alteradas como se verifica no classificador *Random Forest* em que se pode alterar quantos ramos o modelo pode criar por cada atributo no máximo, os ramos totais, qual o limite de relevância para o classificador considerar dado atributo como importante para servir de nodulo. A mudança destas definições é algo que necessita de alguma tentativa e erro, dado que os valores 'perfeitos' tendem a variar de problema para problema não sendo completamente igual entre dois problemas diferentes. Por este motivo, fez-se uso dos conhecimentos aportados pela revisão da literatura para determinar potenciais direções a seguir (p.e. quais as definições que mais efeito parecem ter nos resultados finais), e não como guia sobre o que alterar.

4.8 INTERFACE GRÁFICA

Finalmente, embora interagir com o modelo classificador através do Google Collab em Python seja aceitável durante o treino, para uso corrente por um utilizador não familiarizado com a linguagem ou programação em geral seria útil o classificador ter incluída uma interface gráfica que permita utilizar o mesmo sem ter que programar

instruções em Python, para este fim, foi desenvolvida uma interface textual de linha de comando em que o utilizador preenche a informação a classificar, visível na figura 4.

```
Insira Sair a qualquer altura para fechar o programa
Introduza a diretoria ou texto do email a classificar
C:/shared/emails
Introduza o modelo que realizara a classificação (SVC, RFC, LR, KNN, ETC ou TODOS)
SVC
Esta a classificar um ficheiro? (S ou N)
S
Esta a classificar todos os ficheiros numa diretoria? (S ou N)
S
Deseja que as classificações sejam traduzidas para SPAM ou HAM em vez de números? (S ou N)
S
test1.eml
test2.eml
{'test1.eml': 'Ham', 'test2.eml': 'Spam'}
{'test1.eml': 'Ham', 'test2.eml': 'Spam'}
Se não quiser classificar mais emails, insira SAIR agora
```

Figura 4: Imagem da Interface CLI, tendo acabado de classificar um conjunto de dois ficheiros EML com o classificador SVC

5

DESENVOLVIMENTO

Para criar o produto final deste trabalho, foram feitas várias escolhas no que toca à aquisição do Dataset utilizado para treinar e testar o modelo bem como os detalhes específicos do funcionamento do modelo tais como qual o modelo vetorizador a utilizar, quais os melhores classificadores ou as bibliotecas a utilizar. Neste capitulo são enumeradas e justificadas as escolhas feitas para este fim.

5.1 DETERMINAÇÃO DE MODELO E FORMA DE APRENDIZAGEM

Com o inicio do desenvolvimento prático deste projeto, foi primeiro necessário determinar os específicos do modelo a implementar, bem como o tipo de aprendizagem mais adequado aos objetivos.

5.1.1 *Modelo*

No contexto de um modelo sBERT, existem dois possíveis caminhos a tomar quanto ao modelo a utilizar, pode ser escolhido um modelo pré-treinado ou pode ser treinado um modelo novo, sendo que ambas estas alternativas têm os seus prós e contras.

Com um modelo pré-treinado, as principais vantagens são a facilidade de uso ao não ser necessário interagir com o processo de treinar um novo modelo e todas as variáveis destes bem como o custo temporal, sendo que um modelo pré-treinado não necessita de treino. No entanto, esta opção também tem os seus problemas, modelos pré-treinados tendem a ser destinados a um uso generalista, isto leva a menor qualidade nos resultados quando comparados com os resultados de um modelo específicamente treinado para lidar com o problema em questão, finalmente, o facto que estes modelos pré-treinados tendem a ser dedicados a datasets inglês, o que impossibilita o seu uso para problemas noutras línguas.

Pelo outro lado, modelos treinados manualmente têm os seus pontos positivos especialmente no que toca à sua performance, tendendo a oferecer melhores resultados nos problemas em que foram treinados relativamente aos modelos pré-treinados

generalistas. Existe também a possibilidade de se criar modelos focados em linguagens ou tópicos para os quais não existem modelos pré-treinados disponíveis. Os pontos negativos do modelo treinado manualmente são no custo temporal e de recursos computacionais, sendo que estes modelos, para darem resultados aceitáveis necessitam de um tempo de treino e datasets grandes, podendo o treino demorar várias semana ou até meses, e requerer recursos computacionais que excedem os oferecidos por computadores pessoais, o modelo MiniLM-L6, por exemplo, demorou 4 minutos a realizar os encodings para 90000 emails.

Com tudo isto em consideração, foi escolhida a opção do modelo pré-treinado, escolha motivada pelo facto de o dataset em uso ser em inglês, língua para a qual não existe falta de modelos pré-treinados de qualidade, existem também preocupações temporais e de recursos, dado que a máquina que seria usada para o treino não disponibiliza recursos ao nível necessário para treinar um modelo sBERT dado que, mesmo um modelo relativamente pequeno como o All-MiniLM-L6-v2 requeriu 7 TPUs v3-8, oferecendo capacidade computacional consideravelmente superior à única GPU NVidia utilizada.

5.1.2 *Aprendizagem*

Sendo o problema aqui abordado um de classificação, a metodologia mais utilizada a para o treino do modelo é a da aprendizagem supervisionada, na qual o modelo recebe informação previamente classificada e depois estabelece quais as características do email que são relevantes para determinar a classificação do mesmo em novos casos.

Para este fim, no dataset final, todos os emails irão estar acompanhados de um label a indicar se são spam ou ham.

5.2 AQUISIÇÃO DOS DATASETS

O dataset final utilizado para treinar o modelo de classificação deste projeto consiste de um aglomerado de vários diferentes conjuntos de emails spam, obtidos a partir da plataforma online Kaggle. Esta é uma plataforma destinada a entusiastas do processamento de dados e competições do mesmo, estando focada no desenvolvimento de modelos IA para classificar dados. Para esse fim, utilizadores podem disponibilizar datasets já tratados e com labels pré-incluídos para vários diferentes tipos de

classificação, estando incluído entre estes vários datasets com emails de spam. Foram obtidos os seguintes datasets:

- Email Spam Classification Dataset CSV de Balaka Biswas Biswas (2019) com 5172 emails, daqui em diante referido como dataset 1.
- Spam Email Classification Dataset de Puru Singhvi Singhvi (2023) com 83446 emails, que é por si um aglomerado de dois datasets: o dataset enron Metsis et al. (2006) e o dataset TREC de 2007 al (2007). Daqui em diante este será referido como dataset 2
- Phishing Email Dataset de Naser Abdullah Alam e Amith Khandakar Alam (2024), constituído por 6 datasets: o dataset enron Metsis et al. (2006), o dataset Ling Sakkis et al. (2003) com 82486 emails, o dataset Nazario de Jozé Nazario, o dataset SpamAssassin pela organização do mesmo nome, o dataset CEAS e o dataset 'Nigerian Fraud'. A pedido dos autores deste conjunto, é também citado o trabalho Al-Subaiey et al. (2024). Daqui em diante este será referido como dataset 3.

Deve-se notar que o dataset Enron se encontra contido em duplicado no segundo e terceiro conjunto de datasets.

Como já foi mencionado no Capitulo 4, o dataset final será particionado seguindo a porção 80/20 para dataset de treino/teste, havendo na segunda fase dos testes um baralho dos mesmos para garantir a qualidade do modelo final.

5.3 TRATAMENTO DOS DATASETS

Após a obtenção dos datasets utilizados nestes testes, foi realizada uma série de transformações nestes. O primeiro passo tomado foi a exclusão de valores duplicados, resultantes da inclusão do dataset Enron em dois dos aglomerados usados: al, 2007 e Alam, 2024. Após esta remoção, foram notadas algumas discrepâncias entre os três conjuntos de datasets obtidos que impossibilitavam a sua junção:

- O dataset 1 continha 2 colunas irrelevantes para a análise, uma sem nome e outra com o nome `label_num` que continha o label em formato numérico, ambas foram excluídas do dataset final.
- Os datasets 2 e 3 continham o label em formato numérico (0 - Ham, 1 - Spam), sido tomada a decissão de utilizar labels em formato string, foi necessário mapear os labels numéricos para "Ham"ou "Spam".

- No dataset 3 a coluna que continha o texto tinha o nome `text_combined`, tendo este nome sido alterado para "text"de modo a estar em conformidade com os restantes datasets durante o processo de junção.

Estando estes conflitos resolvidos, foi realizada a concatenação dos três datasets para obter o resultado final: Um dataset com cerca de 100 mil emails.

A Figura 1, mostra uma porção do dataset final que foi utilizado.

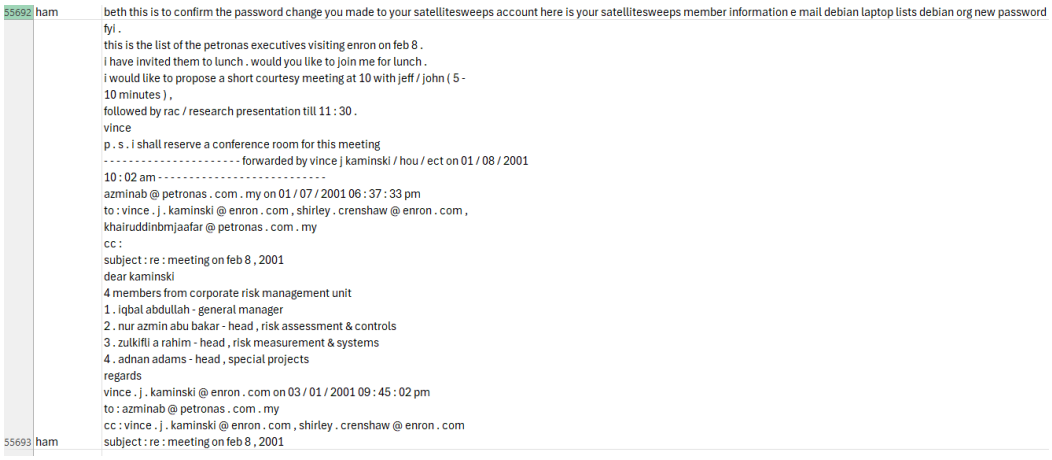


Figura 5: Porção do Dataset final usado

5.4 TOKENIZAÇÃO

A tokenização dos emails é realizada pelo próprio modelo vetorizador sBERT usado, este implementa uma tokenização WordPiece da Google, que transforma o token numa representação numérica em que a cada primeira letra de uma palavra são atribuídas o número de vezes que esta se encontra ligada a outras letras.

5.5 VETORIZAÇÃO

Estando criados os tokens, estes tem depois que ser passados pelo processo de encoding ou vetorização, em que o modelo vetorizador irá interpretar estes tokens e atribuir-lhes valores vetoriais que irão representar o seu significado espacialmente. Para esta fase foram testados 3 diferentes modelos em competição:

- Modelo Vetorizador All-MiniLM-L6-v2, treinado com um dataset de mas de 1 Mil Milhões de Pares de Frases em aprendizagem auto-supervisionada e desenvolvido através de Fine-Tuning do modelo MiniLM-L6-H384-uncased.

Com um tamanho de 80MBs este será o modelo mais pequeno em teste mas também o mais rápido, podendo processar 14200 frases por segundo com uma GPU V100.

- Modelo Vetorizador All-MiniLM-L12-v2, treinado com um dataset de mais de Mil Milhões de Pares de Frases em aprendizagem auto-supervisionada e desenvolvido através de Fine-Tuning do modelo MiniLM-L12-H384-uncased. Com um tamanho de 120MBs este é um modelo intermédio, podendo processar 7500 frases por segundo com uma GPU V100.
- Modelo Vetorizador All-mpnet-base-v2, treinado com um dataset de mais de 1 Mil Milhões de Pares de Frases em aprendizagem auto-supervisionada e desenvolvido através de Fine-Tuning do modelo mpnet-base da Microsoft. Com um tamanho de 420MBs este é o maior modelo em teste e o mais lento, processando apenas 2800 frases por segundo com uma GPU V100.

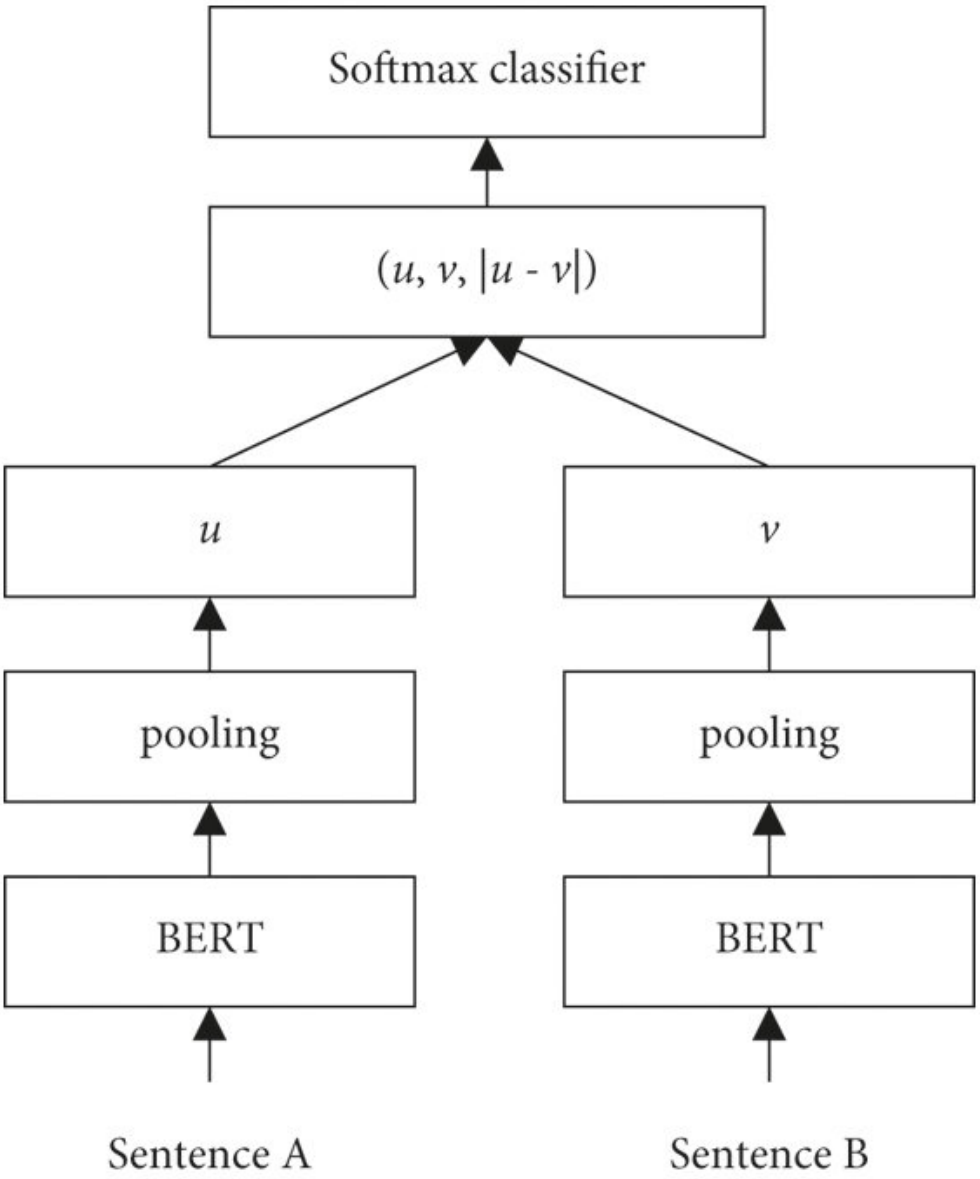


Figura 6: Estrutura base do sBERT, imagem obtida de Reimers e Gurevych, 2019

Destes três modelos, é esperado que o modelo mpnet, tendo um tamanho maior de 420MB, deverá oferecer as melhores métricas de classificação dado que este maior dataset permite uma melhor capacidade de classificar novos dados diferentes dos de treino enquanto o modelo All-MiniLM-L6-v2 deverá oferecer as piores métricas sendo o mais pequeno dos 3. De um ponto de vista temporal, é esperado o inverso: O modelo All-MiniLM-L6-v2 a ser o mais rápido dos 3 e o modelo mpnet o mais lento. Dos 3, é esperado que o modelo All-MiniLM-L12-v2 ofereça uma solução intermédia, com um custo temporal menor que o modelo mpnet e métricas de classificação superiores ao All-MiniLM-L6-v2.

5.6 CLASSIFICAÇÃO

Foram utilizados 5 modelos classificadores diferentes em conjunto com os modelos vetorizadores, estes são:

- K-Nearest Neighbors (KNN): Um modelo simples que classifica um novo dado com base nos seus K vizinhos mais próximos dos dados de treino (K sendo aqui, uma variável definida pelo utilizador), a medida utilizada para determinar a distancia e consequentemente, os vizinhos é, normalmente, a distancia Euclidiana ou de Mannhattan.
- SVM: Um modelo mais complexo que, através de uma função kernel (no caso deste projeto, uma função kernel linear), estabelece uma divisão no espaço ocupado pelos vetores, esta irá dividir os dados entre os possíveis labels (Ham ou Spam), sendo que um vetor a ser classificado irá ser colocado num dos dois lados deste divisor, recebendo essa classificação.
- Regressão Logística: Utiliza a função do mesmo nome para classificar um conjunto de dados, sendo que o modelo é treinado para "encaixar"os valores do vetor da palavra a ser classificada na função logística, obtendo depois a partir desta função um valor entre 1 e 0, sendo que 1 é o valor para um positivo (Spam) e 0 o valor para um negativo (Ham). O valor final é depois arredondado para se obter a previsão, servindo a proximidade do valor verdadeiro a este valor arredonda a certeza do algoritmo da sua classificação.
- Random Forest: Um modelo de classificação baseado na árvore de decisão que utiliza o valor de correlação entre as características do dataset de treino (p.e. quantas vezes a palavra "grátis"é usada) e os labels (Spam ou Ham), sendo a partir desta correlação possível estabelecer dois 'caminhos' ou 'ramos' a seguir para o label que está associado a esses valores com base na ausência ou presença da característica em questão. Caso não seja possível determinar o label a partir de um só ramo, o processo de ramificação pode ser continuado com as restantes variáveis até se obter um conjunto de ramos que levem a um único label possível no dataset de treino, sendo que depois, em novos datasets, são seguidos estes ramos com base nas características dos novos dados para alcançar a classificação mais provável para estes.
- Extra Trees: Similar ao modelo Random Forest, exceto que, em vez de os ramos das árvores consistirem no melhor ponto de divisão para classificar, a escolha envolve mais aleatoriedade, o que leva a uma melhor velocidade e diversidade no modelo oferecido.

5.7 TESTES E MELHORIAS

Tendo sido iniciados os testes, a primeira melhoria foi temporal, com a mudança para aceleração assistida por GPU NVIDIA de todos os componentes do modelo, levando a um aumento de velocidade considerável que ultrapassou os 90% de melhoria em alguns casos como o fitting de vetores gerados pelo modelo All-MiniLM-L6-v2 ao classificador SVC que melhorou de 2 minutos para 5 segundos, uma melhoria de cerca de 10000%, a próxima alteração foi a identificação do melhor valor de k para o classificador KNN, dado que este tem um forte impacto no resultado final, tendo sido descoberto que k=6 ofereceu os melhores resultados durante os testes.

Durante o processo da revisão da literatura, foi notado que outros testes utilizam datasets relativamente grandes comparados com o dataset em uso na altura de 5000 emails, levando à aquisição dos restantes dois datasets indicados previamente, a inclusão destes datasets adicionais levou a uma melhoria considerável não só dos resultados dos modelos (cerca de 10% na maioria dos casos) mas também da sua capacidade de classificar novos dados, dada a maior diversidade do treino de que estes agora usufruíam.

Um ponto que se demonstrou difícil de melhorar foi nos testes, específicamente a validação cruzada, não estando presente uma versão com aceleração por GPU nas bibliotecas usadas, tendo sido necessário encontrar uma outra biblioteca que disponibilizasse esta função, após a sua implementação, foi notado um aumento considerável na velocidade dos testes em cerca de 40%.

A última parte do modelo a passar para aceleração para GPU foram os modelos classificadores através da biblioteca cuML, embora esta não tenha ocorrido tão perfeitamente como as restantes mudanças, com um dos classificadores, Extra Trees, a não ter equivalente na biblioteca cuML, com o uso de alternativas com aceleração por GPU, o processo de classificar os dados também recebeu um aumento na sua velocidade considerável de cerca de 60% comparado com classificações em CPU.

5.7.1 *Parâmetros dos modelos classificadores*

Em geral, os modelos classificadores tendem a disponibilizar alguns parâmetros que podem ser alterados pelo utilizador de modo a adequar o modelo criado mais à tarefa em mão podendo estes levar a aumentos consideráveis nos resultados oferecidos pelo

modelo. Antes demais, é de notar que os parâmetros finais usados, constantes nas tabelas 6, 7, 8, 9 e 10 foram descobertos com recurso a tentativa e erro estando.

Tabela 6: Parâmetros do Modelo SVC

Parâmetro	Descrição	Original	Final
C	O parâmetro regularizador. A regularização é a importância que um modelo dá a evitar classificações falhadas, sendo uma menor regularização equivalente a menos importância a falsas positivos e negativos e maior relevância aos verdadeiros positivos e falsos.	1	3
Kernel	Define qual a função que o algoritmo irá utilizar para gerar o modelo final, estando disponíveis função lineares, polinomiais, sigmoide, base radial e "pré-computada"(utilizar uma função feita pelo utilizador).	Base Radial	Polinomial
Degrees	O número de graus da função polinomial	3	4
Gamma	Parâmetro relevante aos Kernels polinomiais, base radial e sigmoide que define o coeficiente destas funções.	Scale	Scale
Coef0	Parâmetro dos Kernels polinomiais e sigmoide que define o termo independente.	0.0	0.1
ToL	A tolerância que o algoritmo dá a erros para considerar que alcançou um modelo aceitável.	1e-3	1.2
Class Weigth	Um parâmetro especialmente importante para tarefas em que existem prioridades quanto à classificação dos labels, como é o caso deste trabalho. Este parâmetro efetivamente atribui valores separados a cada label, o que irá levar a que o algoritmo dê maior valor a modelos que melhor classificam os labels com maior valor mesmo que tenham piores resultados com os restantes labels.	Não usado	Ham - 1 Spam - 0.5

Tabela 7: Parâmetros do Modelo Regressão Logística

Parâmetro	Descrição	Original	Final
C	O parâmetro regularizador. A regularização é a importância que um modelo dá a evitar classificações falhadas, sendo uma menor regularização equivalente a menos importância a falsas positivos e negativos e maior relevância aos verdadeiros positivos e falsos.	1	8
Penalty	Parâmetro que define quanto um modelo deve ser punido por estar fora do esperado.	l2	Não utilizado
ToL	A tolerância que o algoritmo dá a erros para considerar que alcançou um modelo aceitável.	1e-3	1e-4
Class Weigth	Um parâmetro especialmente importante para tarefas em que existem prioridades quanto à classificação dos labels, como é o caso deste trabalho, este parâmetro efetivamente atribuir valores separados a cada label, o que irá levar a que o algoritmo de maior valor a modelos que melhor classificam os labels com maior valor mesmo que tenham piores resultados com os restantes labels.	Não usado	Ham - 2 Spam - 1

Tabela 8: Parâmetros do Modelo Random Forest

Parâmetro	Descrição	Original	Final
`n_estimators`	Parâmetro que define quantas árvores o algoritmo irá criar na floresta.	100	800
Critério de bifurcação	Parâmetro que define a métrica utilizada para avaliar uma dada bifurcação na árvore de decisão.	gini	entropia
Profundidade máxima	Define quantas bifurcações um dado ramo pode ter no máximo.	infinita	64
Bootstrap	Define se o algoritmo irá repartir o dataset para treinar cada árvore com partes diferentes ou se irá usar o dataset todo para cada árvore.	True	False
Mínimo de dados por bifurcação	Quantos dados devem estar contidos num dado nó da árvore para levar o modelo a realizar uma bifurcação deste.	2	3

Tabela 9: Parâmetros do Modelo kNearest Neighbors

Parâmetro	Descrição	Original	Final
n_neighbors	Quantos dos dados mais próximos ao que está a ser classificado devem ser considerados seus vizinhos pelo algoritmo ao determinar como o classificar.	5	3

Tabela 10: Parâmetros do Modelo Extra Trees

Parâmetro	Descrição	Original	Final
Critério	Métrica que o modelo usada para qualificar uma dada bifurcação na árvore. (gini, entropia ou perda logística)	gini	entropia
splitter	A estratégia usada para escolher a bifurcação a realizar em cada nodulo, existindo como opções a 'best', na qual é escolhida a melhor bifurcação possível de entre todas, e a 'random' que escolhe a melhor bifurcação aleatoriamente.	random	best
Máximo características	Número máximo de características usadas no processo de bifurcação.	Sem Limite	
Mínimo de dados para bifurcar	O mínimo de dados que um nódulo pode conter para justificar uma bifurcação deste.	2	4
Class Weigth	Um parâmetro especialmente importante para tarefas em que são atribuídas prioridades aos *labels*, como é o caso deste trabalho. Este parâmetro permite atribuir pesos separados a cada *label*, o que irá levar a que o algoritmo de maior valor a modelos que melhor classificam os *labels* com maior valor mesmo que tenham piores resultados com os restantes *labels*.	Não usado	Balanced

5.8 TESTES DE CLASSIFICAÇÃO USANDO PURAMENTE O VETORIZADOR

Houve interesse em testar o quão capaz o próprio encoder sBERT seria a atribuir vetores de modo a ser possível classificar novos dados usando puramente a distancia dos vetores da mensagem em análise aos vetores das mensagens mais próximos no dataset de treino, ou seja, uma classificação usando a similaridade de vetores sem utilizar um verdadeiro modelo classificador como o SVM ou RFC. Existem algumas formas de computar a distancia entre vetores, incluindo a similaridade de cosseno

$$\frac{x_1 \cdot x_2 + y_1 \cdot y_2}{\sqrt{x_1^2+y_1^2} \cdot \sqrt{x_2^2+y_2^2}} \tag{5}$$

e a distancia euclidiana

$$|\text{Vetor2-Vetor1}| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{6}$$

5.9 DESENVOLVIMENTO DE UMA INTERFACE PARA PERMITIR O USO DO MODELO FINAL POR UM NOVO UTILIZADOR

O modelo final que agrupa o modelo vetorizador e o modelo classificador têm uma falha, o facto de este apenas pode ser utilizado através do Notebook Python em que o modelo foi treinado, tal situação dificulta o uso do modelo por alguém alheio à informática bem como dificultando um uso corrente do modelo, para tratar deste problema, o modelo foi colocado num programa que serve de interface.

5.9.1 *Programa CLI (Command Line Interface)*

O programa é uma interface textual numa janela de comandos, este permite classificar tanto emails em formato de ficheiro .eml bem como permitindo o utilizador inserir o conteúdo do email manualmente na janela de comando para avaliar, adicionalmente, a primeira opção mencionada também funciona com vários ficheiros ao mesmo tempo.

Deve ser notado que a interface está dividida em dois ficheiro .Py: um com o nome "interface.py"que serve de interface para um utilizador com menos experiência, visível na figura 4, aqui os argumentos são inseridos um a um manualmente. Para automatização, o segundo ficheiro, chamado "main.py", pode receber todos os argumentos de uma vez com uma invocação da função classificar.

6

RESULTADOS

No fim, foi recolhida informação de métricas como a Precisão, Especificidade, Recall, F1 Score e Fβ Score e tempo, separadamente para cada par classificador-vetorizador bem como com aceleração assistida por GPU NVIDIA e sem. Neste capitulo, são primeiro mostrados os dados em bruto em formato de tabela, seguidos por gráficos que oferecem uma visão mais fácil de interpretar, finalizando com uma discussão dos resultados para inferir e estabelecer conclusões com base nestes resultados.

6.1 TESTES INDIVIDUAIS

Na primeira fase dos testes, os modelos foram testados com uma única iteração de vetorização do dataset. Para 7 classificadores (SVC, Regressão Logística, Random Forest, K-Nearest Neighbors, Extra Trees, sem classificador usando similaridade de 1 vizinho e sem classificador usando similaridade de 2 vizinhos) foram testados 21 pares vetorizador-classificador: Com o modelo vetorizador All-MiniLM-L6-v2 11; Com o modelo vetorizador All-MiniLM-L12-v2 12; Com o modelo vetorizador All-mpnet-base-v2 13.

Tabela 11: Resultados de classificação com All-MiniLM-L6-v2

	SVC	LR	RFC	KNN	ETC	Sim 1	Sim 2
Precisão	98.9%	97.0%	96.9%	98.6%	86.9%	98.8%	99.2%
Especificidade	98.8%	96.8%	96.6%	98.4%	85.6%	98.7%	99.2%
Recall	99.1%	93.5%	96.6%	98.3%	86.8%	98.3%	96.8%
F1 Score	99.0%	95.2%	96.8%	98.4%	86.9%	98.5%	98.0%
Fβ Score	99.0%	96.2%	96.8%	98.5%	86.9%	98.7%	98.7%

Tabela 12: Resultados de classificação com All-MiniLM-L12-v2

	SVC	LR	RFC	KNN	ETC	Sim 1	Sim 2
Precisão	98.9%	97.1%	97.0%	98.3%	86.6%	98.7%	99.3%
Especificidade	98.8%	96.9%	96.8%	98.1%	85.2%	98.6%	99.2%
Recall	98.8%	93.4%	95.8%	98.2%	86.8%	98.1%	96.5%
F1 Score	98.8%	95.2%	96.4%	98.2%	86.7%	98.4%	97.9%
Fβ Score	98.8%	96.3%	96.8%	98.3%	86.6%	98.6%	98.7%

Tabela 13: Resultados de classificação com All-mpnet-base-v2

	SVC	LR	RFC	KNN	ETC	Sim 1	Sim 2
Precisão	99.0%	97.9%	98.0%	98.7%	89.3%	98.9%	99.3%
Especificidade	98.9%	97.7%	97.8%	98.6%	88.1%	98.8%	99.3%
Recall	99.2%	96.4%	97.2%	98.7%	90.2%	98.7%	97.3%
F1 Score	99.1%	97.2%	97.6%	98.7%	89.7%	98.8%	98.3%
Fβ Score	99.0%	97.6%	97.8%	98.7%	89.5%	98.8%	98.9%

Aqui deve ser notado que o vetorizador mpnet, apesar de ter um tamanho cerca de 5 vezes maior que o modelo All-MiniLM-L6-v2, na maioria dos casos teve melhorias inferiores a 2%, com a exceção do classificador Extra Trees, em que houve uma melhoria de 3, facto que é enfraquecido pelos piores resultados deste classificador comparado com os restantes.

Tabela 14: Temporização

Para cada um destes pares, foi também registado o tempo que cada etapa do processo demorou, primeiro na ausência de aceleração por GPU NVidia, visível na tabela 14, depois com esta aceleração, visível na tabela 15. Para facilitar comparação, a diferença entre os dois valores pode ser visto na tabela 16 no formato Diferença Tempo/Diferença Percentagem, sendo que uma percentagem negativa indica que o custo temporal foi maior com aceleração por GPU.

	MiniLM-L6	MiniLM-L12	mpnet
Embeddings	04:16,8 Min	04:31,9 Min	26:22,1 Min
Fitting - SVC	02:03,6 Min	02:41,1 Min	14:56,1 Min
Fitting - LR	00:05,7 Min	00:06,3 Min	00:19,6 Min
Fitting - RFC	46:37,1 Min	46:17,5 Min	1:16:23,7 H
Fitting - KNN	0.01450 Seg	0,01607 Seg	0,02015 Seg
Fitting - ETC	01:10,1 Min	01:05,9 Min	02:37,9 Min
Fitting - Sim	0.07477 Seg	0,08903 Seg	0,14579 Seg
Cls - SVM	01:09,1 Min	01:12,6 Min	06:38,7 Min
Cls - LR	00:56,0 Min	00:58,7 Min	05:09,9 Min
Cls - RFC	00:58,3 Min	01:00,2 Min	05:20,2 Min
Cls - KNN	01:18,3 Min	01:22,0 Min	06:01,0 Min
Cls - ETC	00:55,6 Min	00:58,4 Min	05:11,0 Min
Cls - Sim 1	00:08,3 Min	00:08,6 Min	00:21,4 Min
Cls - Sim 2	00:08,9 Min	00:09,6 Min	00:16,3 Min

Tabela 15: Temporização GPU

	MiniLM-L6	MiniLM-L12	MPNet
Embeddings	05:05,7 Min	04:54,9 Min	37:35,2 Min
Fitting - SVC	00:05,7 Min	00:04,4 Min	00:07,1 Min
Fitting - LR	0,113 Seg	0,105 Seg	0,308 Seg
Fitting - RFC	00:32,3 Min	00:30,5 Min	00:41,2 Min
Fitting - KNN	0,042 Seg	0,046 Seg	0,074 Seg
Fitting - Sim	0,043 Seg	0,077 Seg	0,084 Seg
Cls - SVM	01:03,7 Min	01:02,9 Min	07:31,2 Min
Cls - LR	01:02,3 Min	01:00,8 Min	07:28,4 Min
Cls - RFC	01:25,8 Min	01:24,5 Min	07:49,6 Min
Cls - KNN	01:04,3 Min	01:02,9 Min	07:30,6 Min
Cls - Sim 1	00:12,7 Min	00:09,5 Min	00:22,5 Min
Cls - Sim 2	00:13,0 Min	00:09,0 Min	00:23,4 Min

Tabela 16: Temporização - Melhorias

	MiniLM-L6	MiniLM-L12	MPNet
Embeddings	-19.50%	-8,43%	-42,54%
Fitting - SVC	10710,81%	15515,74%	12532,49%
Fitting - LR	11529,89%	12007,69%	6271,11%
Fitting - RFC	8711.34%	10819,44%	11013,77%
Fitting - KNN	-250,57%	-178,24%	-267,55%
Fitting - Sim	74,42%	10,37%	73,91%
Cls - SVM	8,46%	59,99%	-13,18%
Cls - LR	-11.39%	-12,14%	-44,67%
Cls - RFC	-56,20%	-49,48%	-46,64%
Cls - KNN	16,94%	23,28%	-24,82%
Cls - Sim 1	-48,39%	-22,55%	-4,73%
Cls - Sim 2	-48,72%	-1,52%	-43.05%

Deve ser notada particularmente a melhoria no fitting dos classificadores SVC e RFC. Estas duas melhorias levaram a uma melhoria coletiva de 42 minutos em média para os modelos vetorizadores MiniLM e de 1 hora e 30 minutos no modelo vetorizador mpnet. Melhorias estas que superaram consideravelmente as perdas sofridas nas classificações. Deve também ser notado que os modelos vetorizadores não são afetados pelo uso de aceleração por GPU NVIDIA, pelo que a perda registada na geração de embeddings não é relevante à comparação.

6.2 VALIDAÇÃO CRUZADA COM 10 FOLDS

Os testes com validação cruzada tiveram o objetivo de obter resultados mais fieis quanto à performance do classificador ao permitir testar o mesmo com 10 *datasets* 'diferentes' (efetivamente estes eram só baralhos do dataset original) isto para comprovar que o modelo classificador não sofre de *overfitting* ao se obter resultados desejáveis para todos os conjuntos, caso apenas um dos conjuntos obtivesse resultados positivos, isto comprovaria que houve *overfitting* a este mesmo. Os resultados destes testes foram repartidos por classificador, estando os resultados para todos os pares de modelos vetorizadores com o classificador SVC na tabela 17, o classificador de Regressão Logística na tabela 18, o classificador Random Forest na tabela 19, o classificador K-Nearest Neighbors na tabela 20 e o classificador Extra Trees na tabela 21.

Tabela 17: SVM

	All-MiniLM-L6-v2	All-MiniLM-L12-v2	All-mpnet-base-v2
Precisão Média	98,37%	98,43%	98,82%
Melhor Precisão	98,55%	98,71%	99,02%

Tabela 18: Regressão Logística

	All-MiniLM-L6-v2	All-MiniLM-L12-v2	All-mpnet-base-v2
Precisão Média	93.86%	93.96%	94.06%
Melhor Precisão	94.21%	94.34%	94.63%

Tabela 19: Random Forest

	All-MiniLM-L6-v2	All-MiniLM-L12-v2	All-mpnet-base-v2
Precisão Média	95.07%	95.21%	95.64%
Melhor Precisão	95.57%	95.62%	95.92%

Tabela 20: KNN

	All-MiniLM-L6-v2	All-MiniLM-L12-v2	All-mpnet-base-v2
Precisão Média	98.00%	98.15%	98.34%
Melhor Precisão	98.14%	98.72%	99.02%

Tabela 21: Extra Trees

	All-MiniLM-L6-v2	All-MiniLM-L12-v2	All-mpnet-base-v2
Precisão Média	81.86%	82.05%	82.14%
Melhor Precisão	82.55%	82.51%	82.73%

6.3 DISCUSSÃO DOS RESULTADOS

6.3.1 *Modelos vetorizadores*

Observando os resultados dos testes individuais podem ser notados alguns pontos: O modelo all-MiniLM-L12 ofereceu resultados melhores com que os restantes nos classificadores KNN e por similaridade, comprovando que um dataset de treino maior (supondo que tal dataset foi propriamente selecionado e tratado) como este modelo tem, pode resultar em melhores resultados, devo notar que embora os

resultados individuais para os restantes classificadores não sejam tão positivos para este modelo, isto pode ser atribuído à aleatoriedade envolvida na partição do dataset, podendo-se verificar nos resultados da validação cruzada o mesmo. Além disto, deve ser também notado que, embora os seus resultados sejam ligeiramente piores que os do MiniLM-L12, pode-se verificar na tabela de temporização que o MiniLM-L6 resultou em menos custo temporal, tendo havido, no melhor caso, uma diminuição de 8 minutos no fitting do classificador RFC, isto, combinado com o facto de a perda de capacidade de classificação não ultrapassar, no pior caso, os 2% de perda, leva a que este modelo seja uma boa escolha para uso num sistema com menos recursos computacionais. Por fim, o modelo mpnet destacou-se na classificação em si tendo, com a exceção do classificador simples, tido melhores resultados, no entanto esta melhoria veio com um custo severo no tempo, tendo a tarefa de embedding demorado mais de trinta minutos relativamente aos modelos MiniLM testados.

6.3.2 *Classificadores*

Antes de mais, deve ser notado nos resultados do classificadores que os classificadores simples, que utilizam puramente a proximidade entre os vetores gerados pelos vetorizadores tiveram resultados geralmente superiores aos restantes classificadores, demonstrando o quão evoluída a arquitetura sBERT está para poder ser utilizada só por si para criar classificações sem ser necessário um algoritmo de classificação separado. Além disto, os classificadores SVC e KNN ambos ofereceram resultados a ultrapassar os 90% para todas as métricas (com a exceção da precisão do KNN para o vetorizador mpnet), embora deve aqui ser notado que, sem uso da biblioteca cuml, o primeiro demorou mais de 10 minutos a realizar o fitting.

6.3.3 *CPU vs GPU*

6.3.3.1 *Tempo*

No que toca a performance, tendo sido utilizados tanto classificadores sem aceleração por GPU Nvidia como com esta aceleração, é possível fazer uma comparação entre os resultados obtidos: Embora, inicialmente, a presença de alguns negativos na tabela possa fazer parecer que não ocorreu melhoria, deve-se notar que estes não se aproximam das melhorias que ocorreram, especialmente os fittings dos classificadores SVC, LR e RFC, tendo, no caso do classificador RFC com vetorizador mpnet, havido

uma diminuição de cerca de 1 hora e 16 minutos, estas melhorias salvam bastante mais tempo do que as perdas. Deve-se também notar que a maioria dos casos em que ocorreu uma perda de tempo, foram passos com duração inferior a um segundo, podendo não ser culpa da biblioteca em uso mas simplesmente erros aleatórios causados por problemas de hardware.

6.3.3.2 *Uso de recursos computacionais*

No tema de recursos computacionais, deve ser notado que as várias definições alteradas nos classificadores, apesar de levarem a aumentos nos resultados, também levaram, especialmente no classificador RFC, a aumentos severos no uso de recursos computacionais e também temporais, tendo, ocasionalmente, levado ao gasto de todos os recursos computacionais gratuitos do Google Colab e requerendo repetições dos testes. Além disto, deve ser notado que a aceleração com GPU NVidia teve um uso de recursos melhor comparado a sem o seu uso devido à biblioteca RAPIDS ter várias otimizações destinadas a placas gráficas NVIDIA que permitem um uso mais eficaz de recursos.

6.3.3.3 *Usabilidade*

Com tudo isto dito, a única área em que o modelo classificador com aceleração assistida por GPU foi ultrapassado pelo modelo normal foi na facilidade de uso, sendo que as bibliotecas da RAPID usadas para implementar os classificadores acelerados por GPU NVidia tem problemas tais como o não ser utilizável em maquinas sem estes GPUs, bem como tendo sido destinado a maquinas Linux, o que limita mais o seu uso.

6.3.4 *Escolha final de vetorizador*

Tendo em conta que os principais requisitos para o modelo delineados nos objetivos deste trabalho são uma mistura entre velocidade de classificação (o que exclui o modelo mpnet) e qualidade de resultados, analisando os resultados tanto individuais como de validação cruzada, é notável que os modelos MiniLM estão relativamente próximos nas suas métricas de classificação havendo uma perda negligente entre o modelo MiniLM-L6 e o modelo MiniLM-L12, por este motivo e também por ser mais rápido, foi decidido que o modelo All-MiniLM-L6-v2 seria o mais adequado para ser incluído no modelo de classificação final.

Tendo em conta esta escolha, os resultados podem analisados mais facilmente para este modelo vetorizador juntamente com os vários classificadores através de gráficos, para o classificador SVC a imagem 7 mostra a curva ROC e a imagem 13 mostra a matriz de confusão; Para o classificador de Regressão Logística a imagem 8 mostra a curva ROC e a imagem 14 a matriz de confusão; Para o classificador de Random Forest a imagem 9 mostra a curva ROC e a imagem 15 a matriz de confusão; Para o classificador de K-Nearest Neighbors a imagem 10 mostra a curva ROC e a imagem 16 a matriz de confusão; Para o classificador de Extra Trees a imagem 11 mostra a curva ROC e a imagem 17 a matriz de confusão; Para a classificação sem classificador a imagem 12 mostra a curva ROC e a imagem 18 a matriz de confusão;

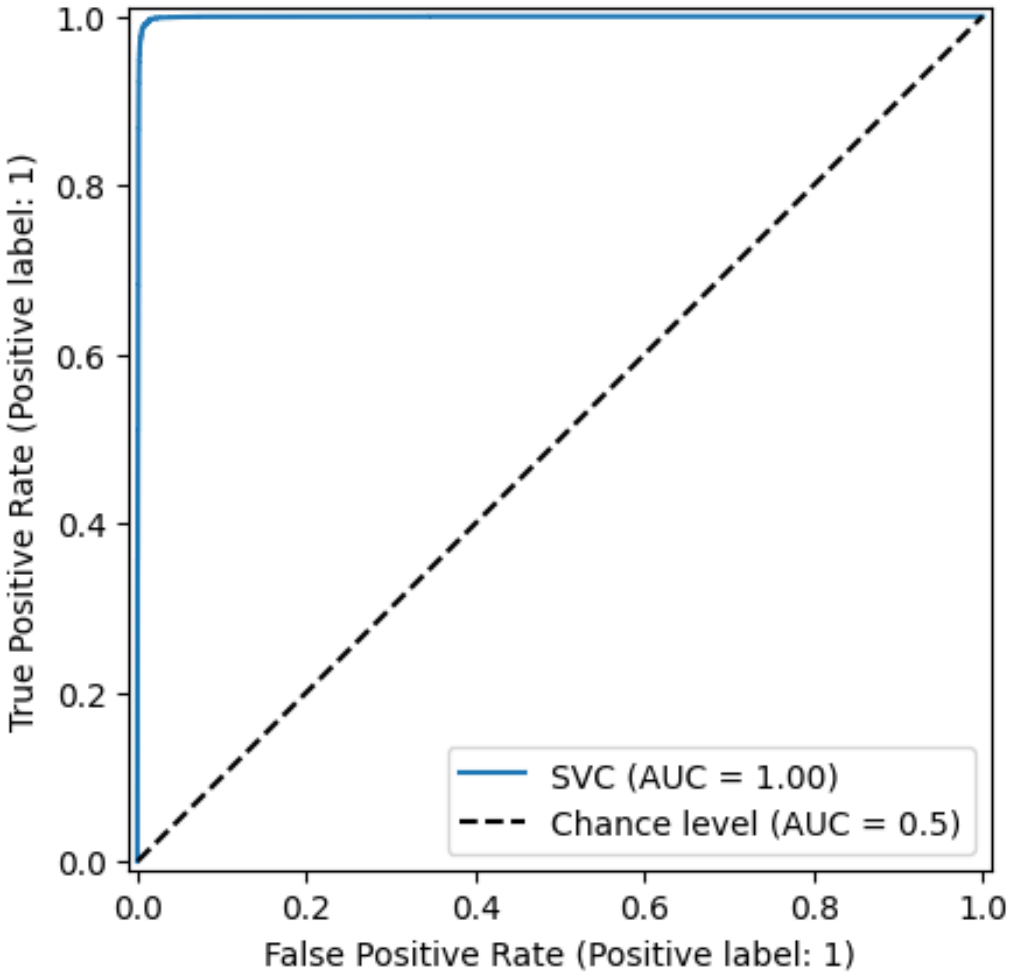


Figura 7: Curva ROC do classificador SVC

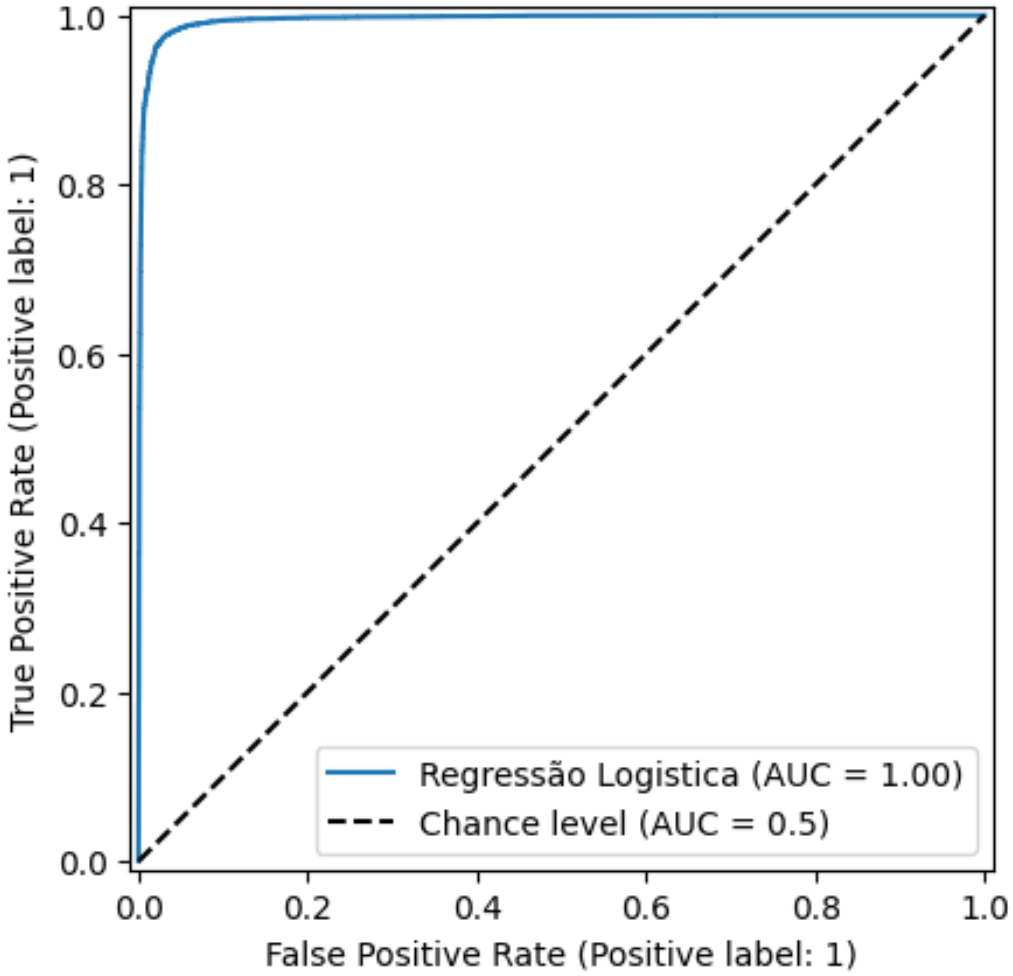


Figura 8: Curva ROC do classificador Regressão Logística

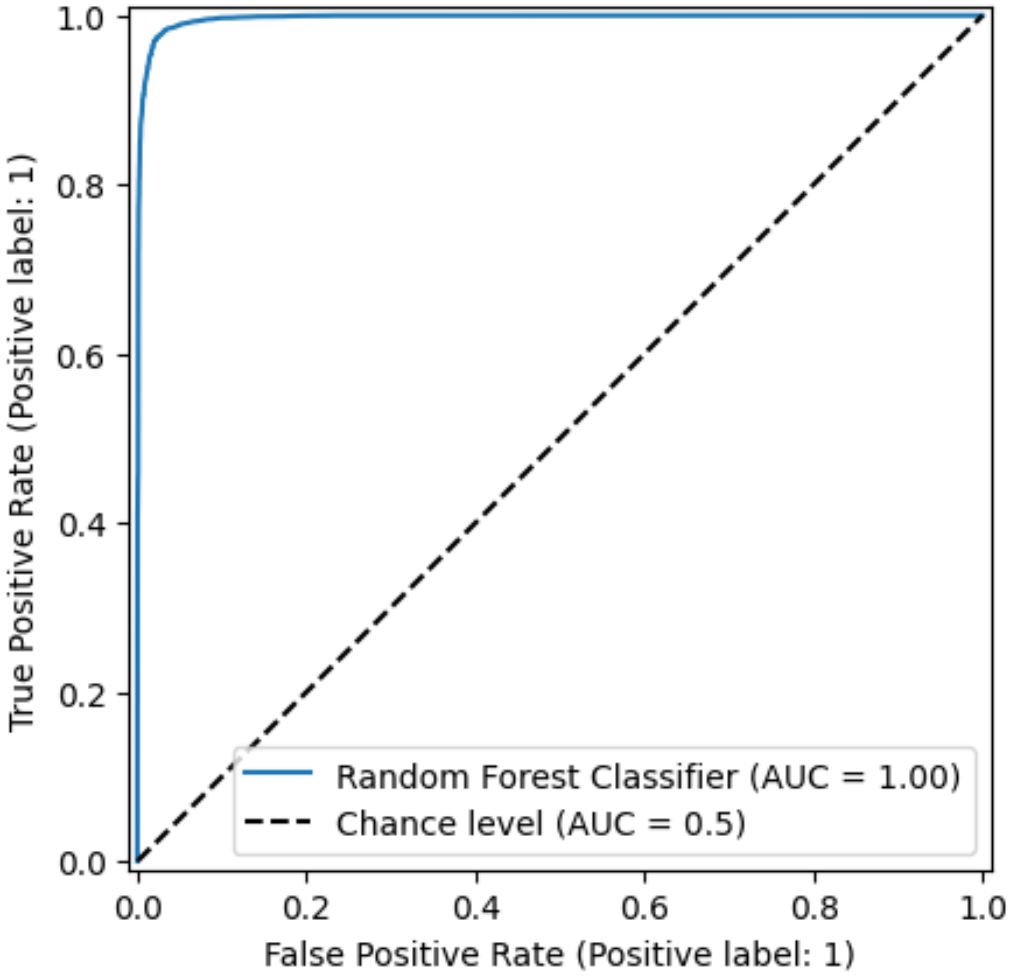


Figura 9: Curva ROC do classificador Random Forest

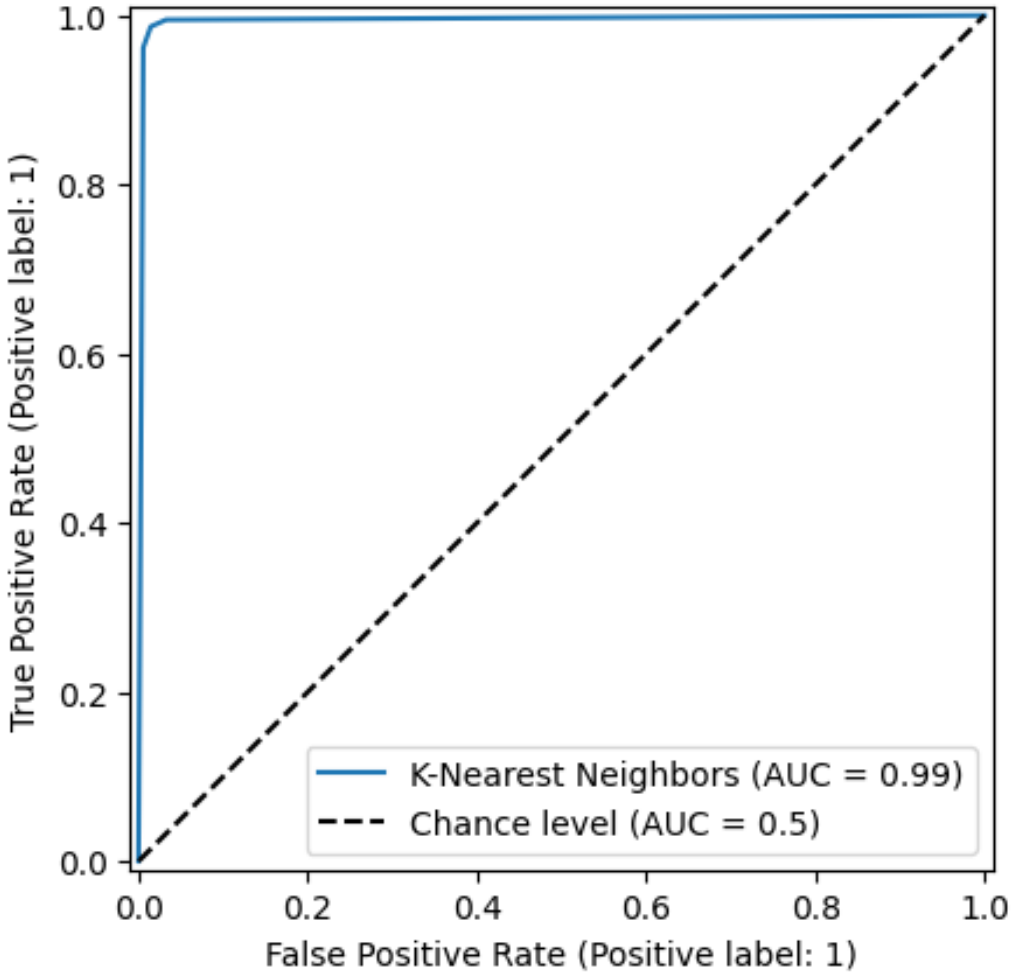


Figura 10: Curva ROC do classificador K-Nearest Neighbors

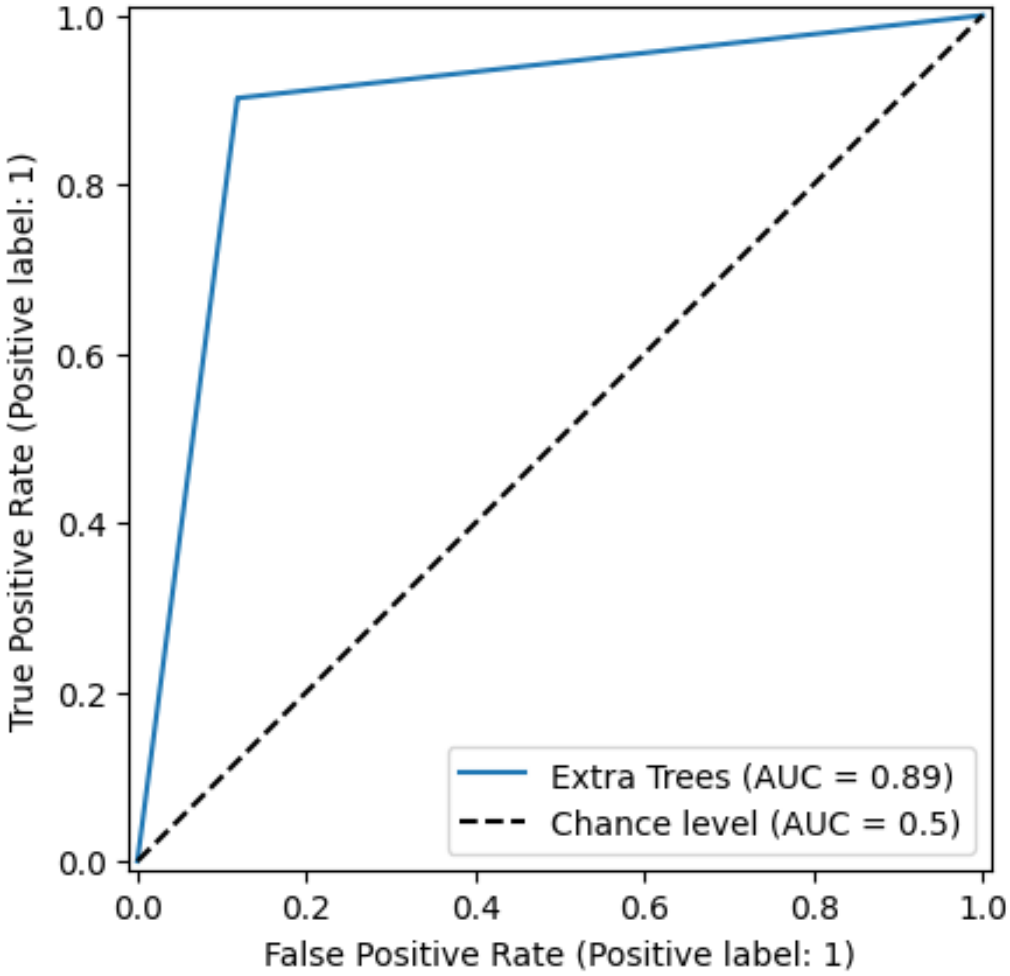


Figura 11: Curva ROC do classificador Extra Trees

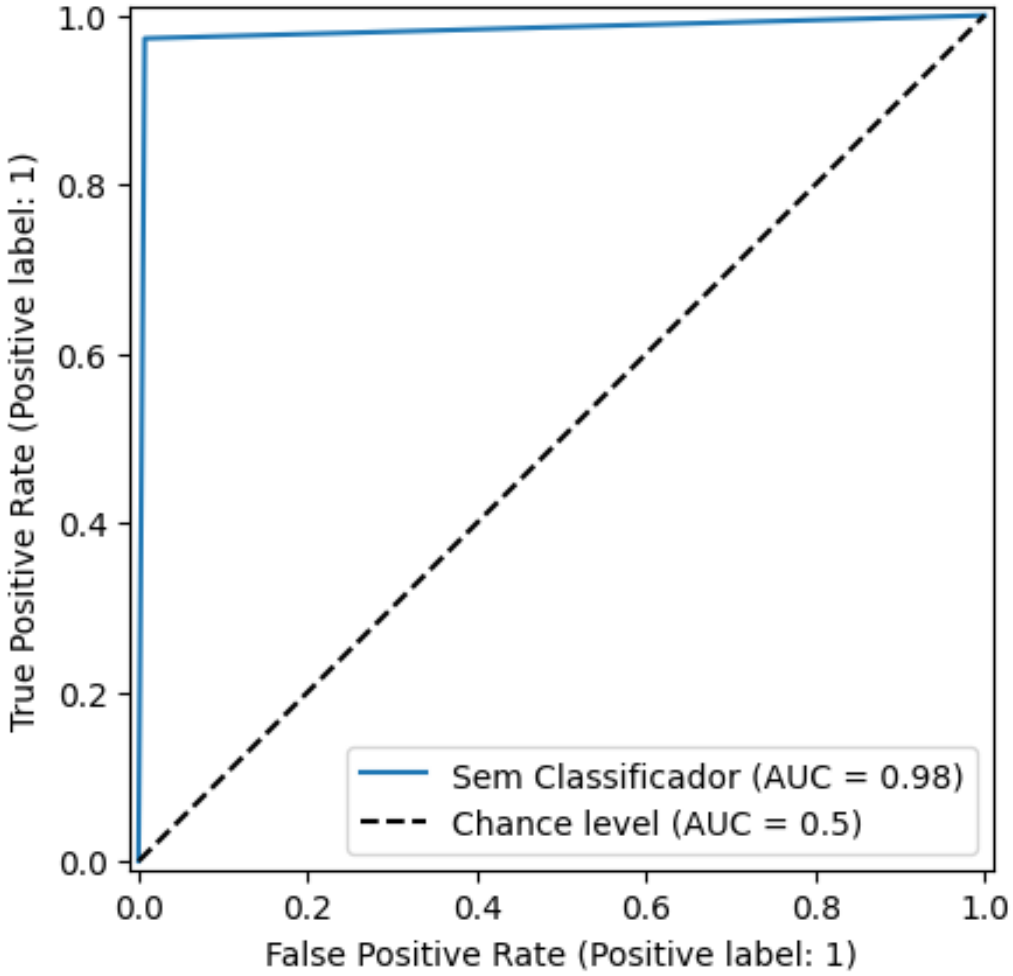


Figura 12: Curva ROC sem modelo classificador

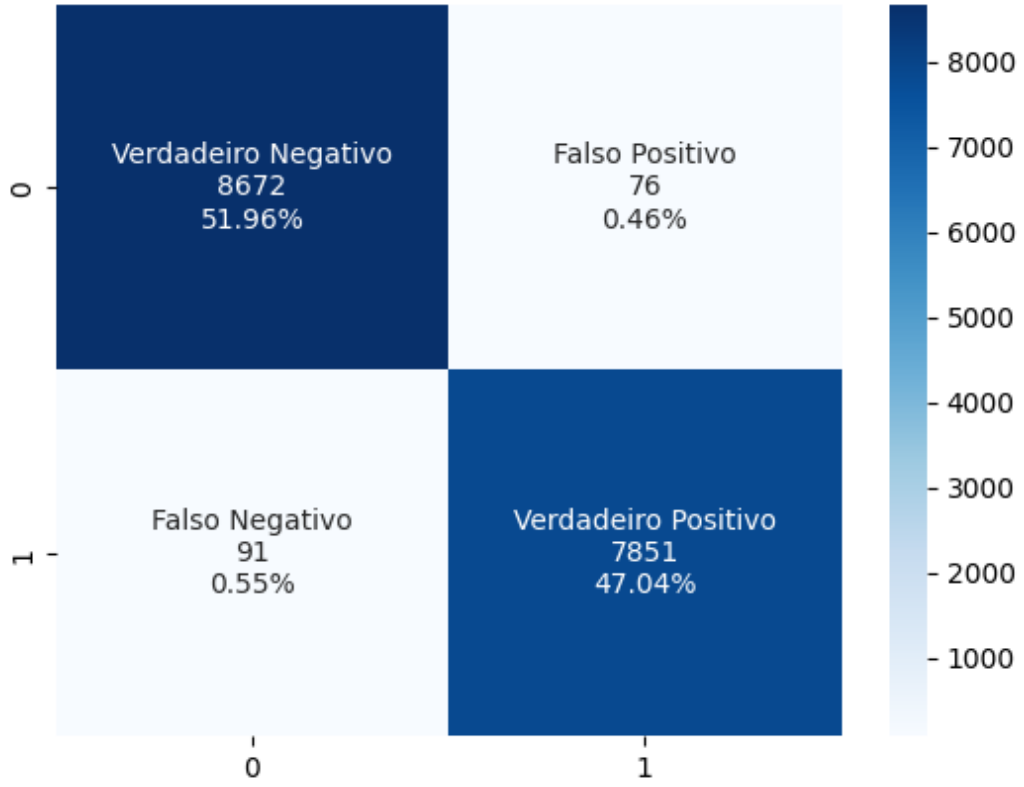


Figura 13: Matriz de confusão com classificador SVC

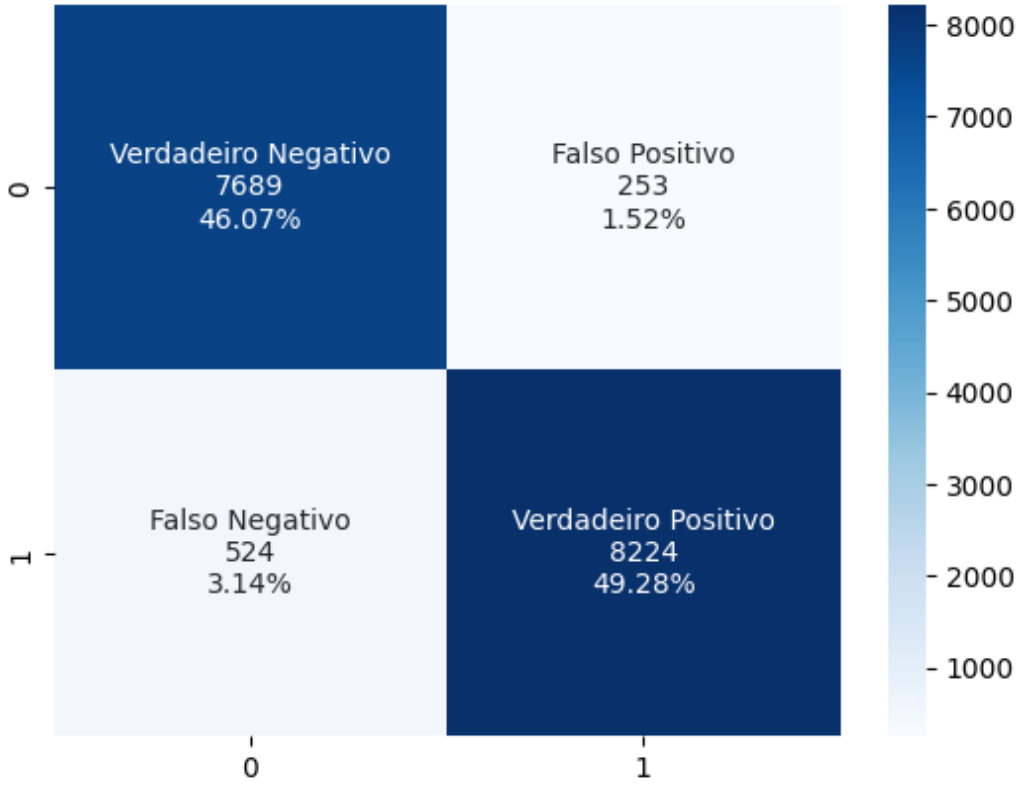


Figura 14: Matriz de confusão com classificador Regressão Logística

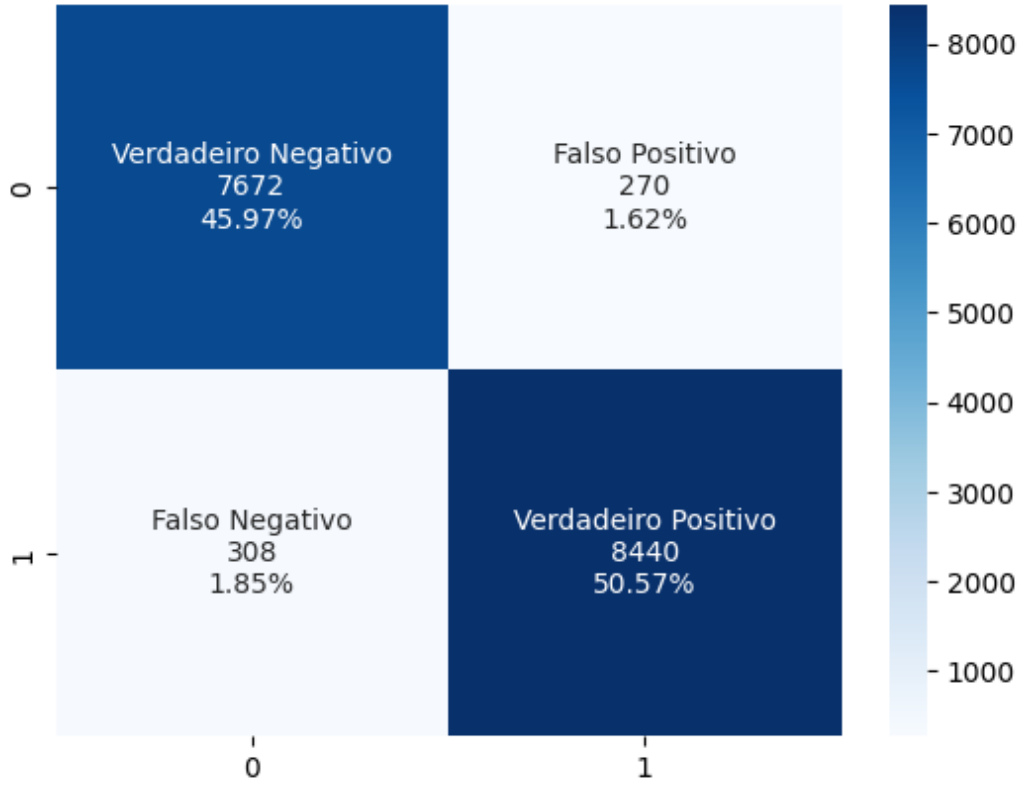


Figura 15: Matriz de confusão com classificador Random Forest

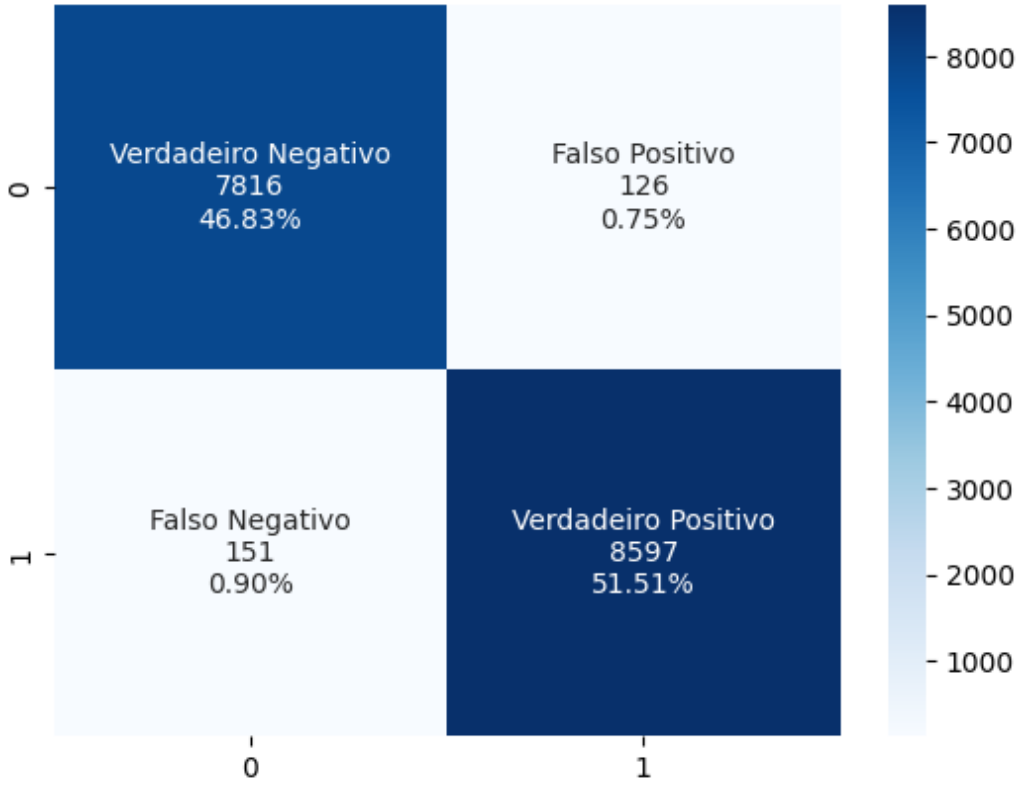


Figura 16: Matriz de confusão com classificador K-Nearest Neighbors

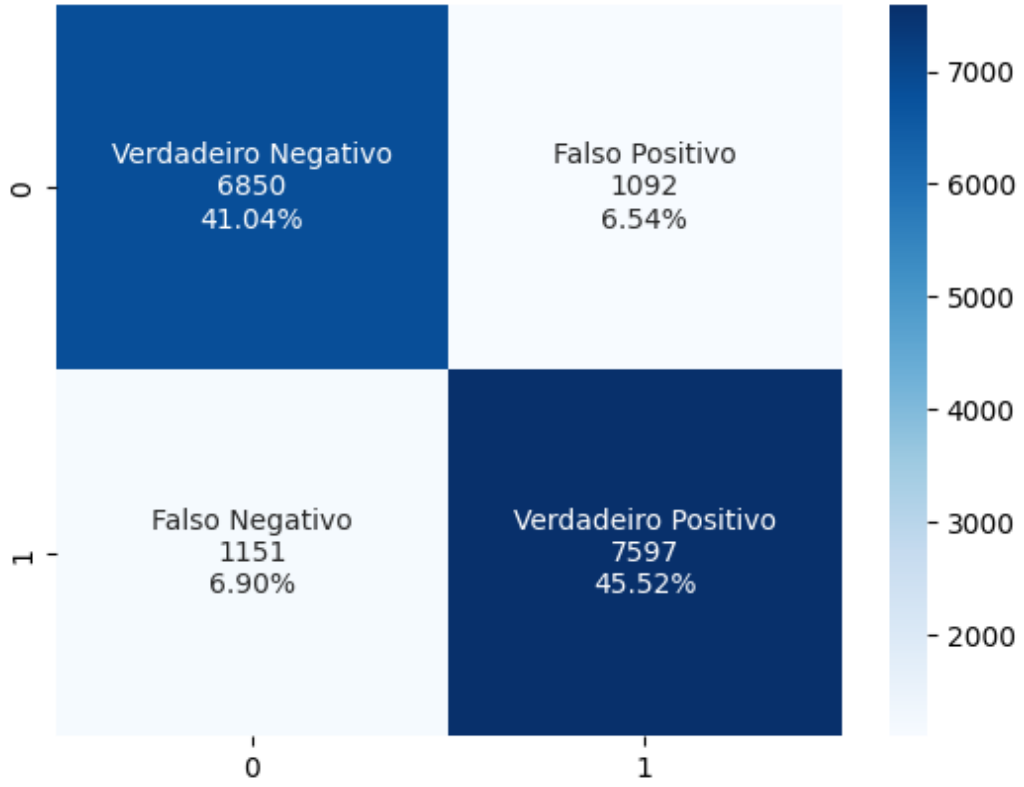


Figura 17: Matriz de confusão com classificador Extra Trees

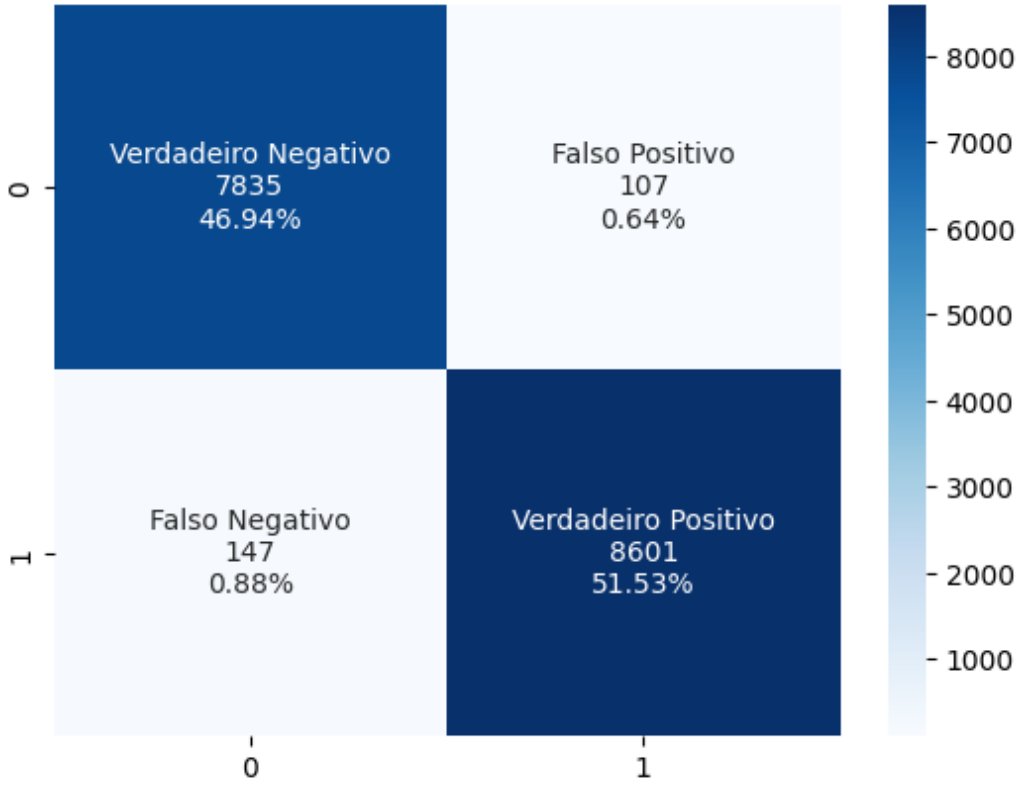


Figura 18: Matriz de confusão classificação sem modelo classificador

6.4 TRABALHOS FUTUROS

Analisando não só os resultados deste trabalho mas também os conhecimentos adquiridos durante a revisão da literatura, o autor recomendaria que futuros trabalhos tentem implementar modelos transformadores que sejam dedicados e treinados específicamente para a deteção de spam, dado que já foram obtidos resultados extremamente promissores utilizando já um transformador sBERT pré-treinado. Também foi notado que a maioria dos modelos não só testados neste estudo, mas também na revisão da literatura tendem a evitar modelos mais complexos (p.e. redes neuronais) no processo de classificação, deve ser tido em conta que a implementação destes modelos de classificação no processo de classificação dos dados vetorizados poderá, desde que com treino suficiente levar a uma melhoria considerável dos resultados embora com uma maior perda de tempo no treino bem como um decréscimo na percetibilidade e justificação da lógica por detrás das escolhas do modelo, podendo a ausência destas qualidades ser considerada um ponto contra o seu uso, dado que já foi comprovado em vários estudos que a falta de justificação pelas escolhas de um modelo pode levar a que estas apareçam menos confiáveis para os humanos

que as utilizam. Finalmente, tendo em conta que as táticas usadas para distribuir spam estão em constante evolução para contrariar as tentativas de o bloquear, é também recomendado um maior foco no desenvolvimento de modelos capazes de reagir a estas evoluções com menor intervenção por parte de humanos, conhecido como auto-melhoria recursiva ou Recursive Self-Improvement (RSI), o qual será, muito provavelmente, o próximo grande avanço na área da IA.

7

CONCLUSÕES

A tarefa de deteção de Spam estará sempre em constante evolução, sendo pouco provável que alguma vez seja desenvolvida uma forma de detetar o Spam perfeita e impossível de derrotar. No entanto, o desenvolvimento desta área tem que continuar para minimizar os danos causados por esta ameaça. Este trabalho demonstra uma possível solução moderna que faz uso de correntes avanços na tecnologia de IA NLP tais como a arquitetura BERT para implementar um modelo capaz de classificar e detetar spam com uma taxa de acerto bem acima dos 95% e alcançando os 99%. Embora, para uso corrente o modelo ainda esteja por otimizar, devido à falta de uma interface gráfica mais desenvolvida para ser usada por um utilizador alheio à área de informática, o modelo final aqui demonstrado já esta finalizado.

Em termos práticos, este modelo, no estado em que se encontra dado o fim do projeto, pode ser utilizado de forma prática em vários possíveis cenários: Desde um cenário privado em que um utilizador individual ou empresarial mantém o modelo como uma extensão no seu browser, sendo que quando acede ao seu email, todos os emails contidos são automaticamente passados pelo modelo para classificar, sendo o utilizador avisado quando um email possivelmente malicioso é encontrado; Outro possível cenário, mais prático, seria o modelo estar no Backend do sistema de emails a servir de filtro, sendo o Spam bloqueado antes de chegar ao utilizador alvo. Com estes dois exemplos, pode-se demonstrar a utilidade deste modelo, bem como mostrar que o modelo pode ver melhorias futuras para o tornar ainda mais competente na área. A deteção de spam é uma área que se encontra em constante evolução, pelo que este modelo irá eventualmente ser ultrapassado mas, na altura em que este relatório é escrito, o modelo aqui demonstrado encontra-se com resultados positivos.

BIBLIOGRAFIA

Aksoy, Cevat Giray et al. (2022). «Working from home around the world». Em: *Brookings Papers on Economic Activity* 2022.2, pp. 281–360.

al, V. Metsis et (2007). *Emails for spam or ham classification (Trec 2007)*. DOI: 10.34740/KAGGLE/DS/5074342. URL: https://www.kaggle.com/datasets/bayes2003/emails-for-spam-or-ham-classification-trec-2007.

Al Saidat, Mohammed Rasol, Suleiman Y Yerima e Khaled Shaalan (2024). «Advancements of SMS Spam Detection: A Comprehensive Survey of NLP and ML Techniques». Em: *Procedia Computer Science* 244, pp. 248–259.

Alam, Naser Abdullah (2024). *Phishing Email Dataset*. DOI: 10.34740/KAGGLE/DS/5074342. URL: https://www.kaggle.com/ds/5074342.

Biswas, Balaka (2019). *Email Spam Classification Dataset CSV*. URL: https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv.

Crawford, Michael et al. (2015). «Survey of review spam detection using machine learning techniques». Em: *Journal of Big Data* 2, pp. 1–24.

Dcbmariano (Acedido: 14-04-2025). *Ilustração de uma curva ROC*. https://commons.wikimedia.org/wiki/File:Curva_ROC.svg.

Fawcett, Tom (2003). «"In vivo"spam filtering: a challenge problem for KDD». Em: *ACM Sigkdd Explorations Newsletter* 5.2, pp. 140–148.

Ferrara, Emilio (2019). «The history of digital spam». Em: *Communications of the ACM* 62.8, pp. 82–91.

Guerra, Pedro H Calais et al. (2010). «Exploring the spam arms race to characterize spam evolution». Em: *Proceedings of the 7th Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS), Redmond, WA*.

Guo, Yanhui, Zelal Mustafaoglu e Deepika Koundal (2023). «Spam detection using bidirectional transformers and machine learning classifier algorithms». Em: *journal of Computational and Cognitive Engineering* 2.1, pp. 5–9.

Jamal, Suhaima e Hayden Wimmer (2023). «An improved transformer-based model for detecting phishing, spam, and ham: A large language model approach». Em: *arXiv preprint arXiv:2311.04913*.

Karim, Asif et al. (2019). «A comprehensive survey for intelligent spam email detection». Em: *Ieee Access* 7, pp. 168261–168295.

Kumar, Nikhil, Sanket Sonowal et al. (2020). «Email spam detection using machine learning algorithms». Em: *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*. IEEE, pp. 108–113.

Kumar, Shailender e Shweta Gupta (2022). «A Brief Survey of Detecting Spam Techniques: Twitter and Email». Em: *2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT)*. IEEE, pp. 563–567.

Malode, Chitra, Pallavi Lanjewar e Priyanka Gomase (2024). «Email Spam Classifier». Em: *International Journal of Advanced Research in Science, Communication and Technology*. URL: https://api.semanticscholar.org/CorpusID:269195862.

Metsis, Vangelis, Ion Androutsopoulos e Georgios Paliouras (2006). «Spam filtering with naive bayes-which naive bayes?» Em: *CEAS*. Vol. 17. Mountain View, CA, pp. 28–69.

Oyedele, Opeoluwa (2023). «Determining the optimal number of folds to use in a K-fold cross-validation: A neural network classification experiment». Em: *Research in mathematics* 10.1, p. 2201015.

Reimers, Nils e Iryna Gurevych (2019). «Sentence-bert: Sentence embeddings using siamese bert-networks». Em: *arXiv preprint arXiv:1908.10084*.

Sakkis, Georgios et al. (2003). «A memory-based approach to anti-spam filtering for mailing lists». Em: *Information retrieval* 6, pp. 49–73.

Siddique, Zeeshan Bin et al. (2021). «Machine Learning-Based Detection of Spam Emails». Em: *Scientific Programming* 2021.1, p. 6508784.

Singhvi, Puru (2023). *Spam Email Classification Dataset*. URL: https://www.kaggle.com/datasets/purusinghvi/email-spam-classification-dataset.

Statista (2025a). *Daily number of spam e-mails sent worldwide as of December 2024, by country*. Accessed: 4-1-2025. URL: https://www.statista.com/statistics/1270488/spam-emails-sent-daily-by-country/.

— (2025b). *Number of e-mail users worldwide from 2018 to 2027*. Accessed: 4-1-2025. URL: https://www.statista.com/statistics/255080/number-of-e-mail-users-worldwide/.

Al-Subaiey, Abdulla et al. (2024). «Novel interpretable and robust web-based AI platform for phishing email detection». Em: *Computers and Electrical Engineering* 120, p. 109625.

Vaswani, Ashish et al. (2017). «Attention is all you need». Em: *Advances in neural information processing systems* 30.

Zhang, Tianyi et al. (2020). «Revisiting few-sample BERT fine-tuning». Em: *arXiv preprint arXiv:2006.05987*.

D E C L A R A Ç Ã O

Declaro, sob compromisso de honra, que o trabalho apresentado nesta dissertação, com o título *"Metodologias baseadas em Inteligência Artificial para a deteção de correio eletrónico não solicitado"*, é original e foi realizado por Pedro Vilaça Cruz Cerqueira dos Santos (2230464) sob orientação de Professor Doutor Miguel Monteiro de Sousa Frade.

Leiria, Setembro de 2025

Pedro Vilaça Cruz Cerqueira dos Santos

Assinado por: Pedro Vilaça Cruz Cerqueira dos Santos
Identificação: BI15418146
Data: 2025-12-15 às 13:25:42