NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

PEDRO JORGE DE OLIVEIRA MARTINS DIAS DINIS

Licenciado em Economia

CLASSIFICADORES DE TEXTO: APRENDIZAGEM CLÁSSICA VS. APRENDIZAGEM PROFUNDA

CONTRIBUTO DE CADA ABORDAGEM NA MELHORIA DA EXPERIÊNCIA DO CLIENTE NA FORMALIZAÇÃO DE RECLAMAÇÕES

MESTRADO EM ANÁLISE E ENGENHARIA DE BIG DATA

Universidade NOVA de Lisboa
Julho, 2023

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

CLASSIFICADORES DE TEXTO: APRENDIZAGEM CLÁSSICA VS. APRENDIZAGEM PROFUNDA

CONTRIBUTO DE CADA ABORDAGEM NA MELHORIA DA EXPERIÊNCIA DO CLIENTE NA FORMALIZAÇÃO DE RECLAMAÇÕES

PEDRO JORGE DE OLIVEIRA MARTINS DIAS DINIS

Licenciado em Economia

Orientador: Joaquim Francisco Ferreira da Silva

Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade NOVA de Lisboa

Júri

Presidente: Paula Alexandra da Costa Amaral
Prof. Associada, Faculdade de Ciências e Tecnologia da Universidade NOVA de Lisboa

Arguente: Francisco José Moreira Couto
Prof. Associado com Agregação, Faculdade de Ciências da Universidade de Lisboa

Vogal: Joaquim Francisco Ferreira da Silva
Prof. Auxiliar, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

MESTRADO EM ANÁLISE E ENGENHARIA DE BIG DATA

Universidade NOVA de Lisboa
Julho, 2023

Classificadores de Texto: Aprendizagem Clássica vs. Aprendizagem Profunda
Contributo de cada abordagem na Melhoria da Experiência do Cliente na Formalização de Reclamações

Este documento foi gerado com o processador (pdf/Xe/Lua)LaTeX e o modelo NOVAthesis (v6.10.17) [23].

À minha Família.

Agradecimentos

Gostaria, em primeiro lugar, de agradecer ao Professor Joaquim Ferreira da Silva, pela ponderação e pragmatismo que sempre acrescentou ao planeamento e ao desenvolvimento desta dissertação, estando igualmente sempre presente e disponível para muito bem me orientar.

Agradeço ao Professor Pedro Barahona, a abertura e a consideração em compatibilizar os tempos de marcação da prova relativa à fase de preparação da dissertação com contra-tempo por mim sofrido, pelo qual incorria em risco de ver adiada essa fase.

Por fim, o meu profundo agradecimento aos Professores do nosso MAEBD, através dos quais tive o privilégio de consolidar e aprofundar conhecimentos de *machine learning* e da analítica avançada, mas fundamentalmente de adquirir novas competências e capacidade crítica em técnicas e metodologias aplicadas em projetos de inteligência artificial, nomeando subjetiva e (temo) inadvertidamente o Professor João Magalhães e o Professor Ludwig Krippahl como seus admiráveis representantes.

« Language can only deal meaningfully with a special, restricted segment of reality. The rest, and it is presumably the much larger part, is silence. »

— George Steiner

Resumo

A proliferação de textos e documentos digitais decorrente da profunda transformação digital que as sociedades modernas se encontram a sofrer há sensivelmente duas décadas, intensificada ainda nos tempos mais recentes, tem provocado um correspondente aumento da procura de processos e técnicas de classificação de texto para diferentes fins, tais como comunicacionais (e.g, suporte a interações entre humanos e organizações por meios digitais), operacionais (e.g., correção, validação e catalogação de documentos), organizacionais (e.g., automatização de centros de contacto e gestão da reputação) ou regulatórios (e.g., deteção de discriminações e manipulações da opinião em processos eleitorais). Esta procura está a ser acompanhada por progressos nos métodos de classificação de texto, protagonizados com inegável sucesso pelos Modelos de *Deep Learning* (DL), que se distanciam dos Modelo(s) Clássico(s) (MC) de *Machine Learning* (ML) por promoverem uma representação da linguagem através de uma extração automática das *features* relevantes, evitando assim que decorra de um processo de *feature engineering*, a cargo do *domain expertise*, quase sempre demorado e oneroso. No entanto, as tecnicidade e dimensão do *corpus* são críticas para a qualidade do desempenho de um Modelo de DL, tal como o requisito de justificação de cada classificação o é para a sua utilidade, dada a sua mais complexa explicabilidade. Uma comparação entre métodos adotados em MC de ML e em Modelos de DL é aqui realizada, quer a nível dos respetivos métodos e *pipelines* utilizados, quer a nível da qualidade e utilidade dos seus resultados obtidos sobre informação de reclamações de Clientes relativas a produtos e serviços financeiros, procurando-se identificar as circunstâncias em que se deverá optar por uma abordagem em detrimento de outra. É por fim avaliada a capacidade de virem a melhorar a experiência do Cliente no ato de formalização de uma reclamação, apoiando-o no preenchimento do tema em função do teor da reclamação, contribuindo, por um lado, para a sua literacia financeira e, por outro, para melhorar a eficiência do tratamento da reclamação.

Palavras-chave: Classificação Supervisionada de Documentos, Modelo(s) Clássico(s) de *Machine Learning*, Modelos de *Deep Learning*, *Feature Engineering*, Produtos e Serviços Financeiros, Formalização de Reclamação, Experiência do Cliente

Abstract

The proliferation of digital texts and documents resulting from the deep digital transformation that modern societies have been undergoing for about two decades, intensified even more recently, has caused a corresponding increase in the demand for text classification processes and techniques for different purposes, such as conversational (e.g., supporting interactions between humans and organizations through digital means), operational (e.g., document correction, validation and cataloguing), organisational (e.g., automating contact-centres and reputation management) or regulatory (e.g., detecting discriminations and opinion manipulations in electoral processes). This demand is being accompanied by advances in text classification methods, led with undeniable success by Deep Learning Models (DL), which differ from the Classical Machine Learning Models (ML) by promoting a language representation through an automatic extraction of the relevant features, thus avoiding a time-consuming and costly feature engineering process, performed by the domain expertise. However, the technicality and the size of the *corpus* are critical for the quality of the performance of a DL Model, as the requirement of justification of each classification is for its usefulness, given its more complex explainability. A comparison between methods adopted in Classical ML Models and DL Models is carried out herein, both in terms of the respective methods and pipelines used, and in terms of the quality and usefulness of their results obtained on information on customer complaints regarding financial products and services, seeking to identify the circumstances in which one approach should be chosen over another. Finally, their capacity to improve the Customer Experience when formalising a complaint is evaluated, supporting them in filling in the subject according to the content of the complaint, contributing, on the one hand, to their financial literacy and, on the other hand, to improving the efficiency of the complaint processing.

Keywords: Supervised Document Classification, Classical Machine Learning Models, Deep Learning Models, Financial Products and Services, Complaint Submission, Customer Experience

Índice

Índice de Figuras

Índice de Tabelas

Siglas

AE *Autoencoders (p. 6)*
ALBERT *A Lite BERT (p. 42)*

BERT *Bidirectional Encoder Representations from Transformers (pp. x, 11, 20, 26–28, 41–46)*
BM25 Okapi BM25 (BM é acrónimo de *Best Matching*) *(pp. 23, 24)*
BoW *Bag-of-Words (pp. 11, 12, 17, 23, 24, 26, 32, 37, 39, 40, 54)*
BPE *Byte Pair Encoding (p. 10)*

CFPB *Consumer Financial Protection Bureau (pp. xi, 1, 30–33)*
CNN *Convolutional Neural Network(s) (pp. 17, 25, 26)*
CV *Computer Vision (pp. 16–18)*

DeBERTa *Decoding-enhanced BERT with Disentangled Attention (pp. x, 27, 28, 42)*
DL *Deep Learning (pp. vi, vii, x, 1, 2, 4, 5, 7–12, 14, 16, 19, 23, 25, 26, 29, 31–33, 35, 41, 42, 44, 46, 47, 52, 57–61)*

FFNN *Feed-Forward Neural Network(s) (pp. 17, 18, 46, 47, 49, 52, 57)*
FM *Foundation Model (p. 28)*

GLUE *General Language Understanding Evaluation (pp. 42, 43)*
GPT *Generative Pre-trained Transformer (p. 20)*
GRNN *Gated Recurrent Neural Network (p. 25)*

LDA *Linear Discriminant Analysis (p. 6)*
LLM *Large Language Model (p. 28)*
LM *Language Model(s) (pp. 3, 10, 17, 19, 20, 23, 26, 27, 43, 45, 48)*
LSTM *Long Short-Term Memory (pp. 18, 25, 26)*

MAEBD Mestrado em Análise e Engenharia de Big Data *(p. iv)*

MC Modelo(s) Clássico(s) *(pp. vi, x, 1–5, 7, 8, 26, 31–33, 35, 61)*
ML *Machine Learning (pp. vi, vii, x, 1–5, 7, 8, 14, 23, 26, 29, 31–35, 40, 60, 61)*
MLM *Masked Language Model (pp. 20, 27, 28, 43)*

NB *Naïve Bayes (pp. 14, 23)*
NER *Name Entity Recognition (pp. 8, 20)*
NLG *Natural Language Generation (pp. 20, 27)*
NLP *Natural Language Processing (pp. 1, 2, 4, 8, 9, 11, 16–20, 26, 28, 29, 31, 33, 41–43, 45)*
NLU *Natural Language Understanding (pp. 20, 27, 42)*
NSP *Next Sentence Prediction (pp. 20, 28, 45)*

PCA *Principal Component Analysis (pp. 6, 13)*
PoS *Part-of-Speech (pp. 9, 12)*

QA *Question Answering (pp. 20, 28)*

ReLU *Rectified Linear Unit (p. 47)*
RNN *Recurrent Neural Network(s) (pp. 18, 19, 26)*
RoBERTa *Robustly Optimized BERT pre-training Approach (pp. 27, 28, 42)*

SVM *Support Vector Machine (pp. x, 15, 23–25, 40, 54–57, 59, 60)*

TF-IDF *Term Frequency-Inverse Document Frequency (pp. 11, 12, 17, 24–26)*

1

Introdução

A classificação de texto, tarefa de atribuição de uma classe ou categoria adequada a um texto, tem um papel fundamental e transversal nas aplicações de *Natural Language Processing* (NLP), tais como *Sentiment Analysis*, *Topic Labeling*, *Text Categorization* (de notícias, mensagens ou documentos), *Question Answering* ou *Intent Classification* (para *Conversational Agents*).
Para a realização desta tarefa é hoje possível adotar duas abordagens: uma que recorre a Modelo(s) Clássico(s) (MC) baseados em métodos de *Machine Learning* (ML) convencionais, e uma outra já recorrendo a métodos utilizados em Modelos de *Deep Learning* (DL). Para permitir a aprendizagem, supervisionada ou não supervisionada, estas duas abordagens são processualmente distintas, com os MC de ML mais dependentes do *domain expertise* para encontrar e/ou construir as *features* que melhor representem a linguagem e a semântica que estruturam os textos em estudo. Já os DL procuram mapear a estrutura e os contextos incorporados na sequência natural do texto, levando em conta, em métodos mais recentes, com os seus distintos posicionamentos, adotando assim uma lógica de *data-driven*, bastante mais exigente em dimensão de dados.
Este compromisso entre intervenção do *domain expertise* e a dimensão de dados (existindo ainda assim cada vez mais técnicas que permitem mitigar *datasets* mais reduzidos) pode traduzir-se em resultados potencialmente distintos em desempenho e na qualidade do apoio ao utilizador.

1.1 Problema/Motivação

Partindo de informação das reclamações provinda da *Consumer Financial Protection Bureau* (CFPB) [7], agência do governo federal dos Estados Unidos responsável pela recolha das reclamações de Clientes relativas aos produtos e serviços financeiros disponibilizados pelas instituições financeiras a operar no mercado norte-americano, este estudo pretende ser um contributo para a **Melhoria da Experiência do Cliente de produtos e serviços financeiros no ato de formalização de uma reclamação**, propondo-se apoiá-lo na identificação e **preenchimento de campo relativo ao** `Tipo de produto financeiro` **subjacente**

à reclamação (e.g., *Checking or savings account*, *Credit card and prepaid card*, *Mortgage*). Nem sempre detendo o nível de literacia financeira correspondente à diferenciação temática exigida pelas instituições financeiras nestes processos de tratamento de reclamações, nomeadamente para o seu encaminhamento à área interna responsável por lhe dar resposta, pretende-se investigar abordagens e técnicas alternativas de NLP capazes de lhe sugerir, em tempo real, qual o produto financeiro que motiva a sua reclamação, sugestão essa baseada na classificação do texto da reclamação por modelo de categorização de documentos[1]. Para além da melhoria da experiência do Cliente na classificação da sua reclamação, este apoio ao preenchimento permitirá também, por um lado, contribuir para melhorar a eficiência do seu tratamento pelas entidades responsáveis e, por outro, mitigar a eventual iliteracia financeira, reduzindo assim erros de classificação na *ground truth*, fundamental para se assegurar futuras atualizações e a manutenção do desempenho do modelo de categorização a níveis elevados.

1.2 Objetivos

Este estudo pretende avaliar abordagens técnicas distintas para a classificação das reclamações através da **comparação de dois modelos de categorização de documentos** desenvolvidos em aprendizagem supervisionada, um resultante da aplicação de métodos clássicos de *Machine Learning* (o MC de ML) e o outro desenvolvido com recurso a métodos de *Deep Learning* (o Modelo de DL), procurando caracterizá-los quanto a:

- **Processo de desenvolvimento** dos classificadores de texto, nomeadamente o grau de complexidade na estruturação da informação para treino do classificador;
- **Desempenho** global dos modelos e por `Tipo de produto financeiro` que classificará a reclamação;
- **Qualidade do apoio ao Cliente** conseguido pelos modelos, nomeadamente quanto à capacidade de cada um justificar o `Tipo de produto financeiro` mais provável para classificar a reclamação.

A comparação terá em conta os prós e contras de cada uma e a motivação para a realização deste desafio, i.e., a avaliação do compromisso entre esforço de engenharia na extração das *features* (mais automática nos Modelos de DL, mais demorada e onerosa nos MC de ML por dependente da intervenção humana) vs. desempenho proporcionado, medido quer na perspetiva computacional durante a aprendizagem e seleção do melhor modelo - especialmente sob infraestrutura tecnológica[6] de recursos limitados -, quer na qualidade dos resultados alcançados e grau de explicabilidade conseguido.
Assim, os principais objetivos centram-se na procura de respostas às seguintes questões:

[1] Subtarefa de classificação de texto que atribui automaticamente uma categoria pré-definida a documentos.

(i) Como se diferenciam as duas abordagens quanto a tarefas e métodos empregues ao longo das diferentes etapas do processo de classificação de texto?

- Do foco na estruturação do espaço das *features* enquanto representação da linguagem, intervindo-se no conteúdo do texto para que preserve apenas conteúdos informativos, à geração automática de mapas de *features* através da aplicação de *Language Model(s)* (LM)[2];
- Na construção, otimização e seleção do melhor modelo.

(ii) Quais os desafios específicos que se colocam a cada uma das abordagens, que deverão fazer parte das respetivas soluções técnicas?

- O grau de preocupação com a extração das *features* e a redução da dimensionalidade que assegurem, por um lado, a parcimónia do classificador e, por outro, a capacidade de computação exigida pelos algoritmos durante a aprendizagem;
- A procura da melhor parametrização do algoritmo e/ou a melhor arquitetura do classificador, com vista à otimização do nível de desempenho dos modelos.

(iii) Como comparam em resultados e que *outputs* consegue disponibilizar cada abordagem, capazes de justificar junto do Cliente a classificação que realizam?

- Na capacidade global dos modelos em classificar corretamente e em que medida o conseguem transversalmente às classes, nomeadamente nas menos representadas;
- Da distribuição semântica do texto da reclamação, e sua construção mais ou menos influenciada pelas classes, aos *embeddings* que relacionam os elementos do texto entre si e com as classes, proporcionando níveis de transparência dos resultados distintos, e por isso, com igualmente distintas capacidades de interpretar as respetivas classificações.

Refiram-se ainda assim, aspetos comuns aos processos analíticos aplicados nas duas abordagens implementados, quer decorrentes da estratégia adotada na fase de pré-processamento, onde se optou por intervir o menos possível na semântica do texto (abdicando-se por isso dos habituais métodos de normalização de texto aplicados nos MC de ML), quer resultantes das limitações da infraestrutura tecnológica[6] na qual esta investigação se realizou, pelas quais houve necessidade de se recorrer a processos de amostragem estratificada uniforme e de aplicação de ponderador de reposição da representatividade original das classes durante as fases de treino, avaliação e reporte de resultados. Se bem que, no primeiro caso, tal possibilitou a simplificação do processo, no segundo aumentou, em muito, a dificuldade de o programar, conseguindo-se no entanto ajustar a dimensão dos dados à complexidade dos algoritmos, dadas capacidade de processamento e memória existentes, sem perda notada de diversidade de informação, assegurando-se assim a devida consistência de resultados.

[2]Tarefa de previsão da probabilidade de surgimento de uma determinada *token* dada a sequência de *tokens* que a precede ou que a contextualizam na sua vizinhança.

1.3 Contribuições da Dissertação

Esta investigação de técnicas de classificação de texto sob abordagens distintas permitiu, por um lado, reconhecer a perícia do *feature engineering* presente no desenvolvimento de MC de ML, com maior possibilidade de retorno do investimento já em fase de interpretação de resultados e, por outro, experienciar a maior facilidade de mapeamento das *features* conseguido pela abordagem DL, com reduzida intervenção do *domain expertise*, exigindo em contrapartida maior abstração na interpretação das classificações. Já na perspetiva do desempenho computacional, observou-se uma complexidade "induzida" próxima entre os dois classificadores representantes de cada abordagem. Este controlo de complexidade computacional teve, no entanto, um impacto na qualidade dos resultados dos modelos de DL, devendo-se provavelmente menos ao volume de dados que lhes suprimimos do que ao tempo que lhes foi dado para treino.

1.4 Estrutura do Documento

Partindo do presente capítulo, onde se dá conta da motivação da investigação e objetivos a que se propõe dar resposta, este documento apresenta, no segundo capítulo, as duas abordagens, caracterizando-as de acordo com a diversidade e diferenciação de tarefas e de métodos aplicados em NLP ao longo das diferentes etapas do processo de classificação de texto, bem como o Estado da Arte onde se comprova esta sua contextualização.
No terceiro capítulo são detalhadas as propostas de processos analíticos desenvolvidas para cada abordagem, as quais procuram responder aos objetivos aqui elencados e ultrapassar os desafios concretos com que se deparam, consubstanciando-se assim na solução técnica preconizada desta dissertação. Revela, por fim, os resultados obtidos no quarto capítulo, concluindo sobre estes, já no quinto e último capítulo, à luz dos objetivos e motivação perspetivados, sugerindo caminhos futuros de aprofundamento desta investigação em trabalhos a desenvolver sobre temática semelhante.

2

Trabalho Relacionado

2.1 Classificação Supervisionada vs. Não Supervisionada

A tarefa de classificação de texto pode ser realizada através de uma aprendizagem supervisionada ou não supervisionada. O primeiro tipo de aprendizagem é usualmente utilizado em tarefas de classificação para as quais o conjunto de *labels* a atribuir a cada texto (e.g. notícia, mensagem, documento) é conhecido e pré-determinado, tarefas tais como: *Sentiment Analysis*, *Text Categorization* ou *Intent Classification*. Para que se realizem, requerem igualmente um vasto conjunto de exemplos já anotado. Quando tal não se verifica, quer seja por indefinição da classe ou classes a atribuir, quer seja por falta de anotação (processo muitas vezes demorado e dispendioso por envolver o recurso escasso do *domain expertise*), recorre-se à aprendizagem não supervisionada, o que ocorre habitualmente em tarefas como *Topic Modeling* e *Multi-Label Text Classification*.
A preponderância da aprendizagem supervisionada na realização da tarefa de classificação de texto é motivada por usualmente apresentar melhores resultados, estando o erro teoricamente apenas do lado do método de reprodução do *output* através dos exemplos já previamente classificados. Estas classes conduzirão assim o processo de treino do classificador por redução do erro de classificação, gerando-se por fim o modelo, a aplicar posteriormente a novos casos não classificados para os quais será previsto uma *label*. Este tipo de aprendizagem permite melhor selecionar e/ou construir as *features* relevantes, apresentando assim melhores resultados ao longo das distintas métricas de avaliação dos modelos.
Não existindo uma indicação da classe, o desafio da aprendizagem não supervisionada consiste justamente em detetar quantos e quais os principais tópicos ou grupos de temas implícitos em cada texto, capazes de construir atributos (*feature engineering*) especialmente caracterizadores de cada tema, permitindo a posterior classificação. Esta aprendizagem é assim mais exigente na identificação e seleção das *features*, normalmente com necessidade de intervenção do *domain expertise*, uma vez que não tem ajuda das classes para selecionar as mais discriminantes. De notar que este *feature engineering* mais exigente para a classificação não supervisionada, coloca-se tanto a MC de ML como a Modelos de DL, ambos a

recorrer a técnicas de redução da dimensionalidade para mitigar o *curse of dimensionality* e capturar o poder discriminante das *features*, quer seja através de *Principal Component Analysis* (PCA), *Linear Discriminant Analysis* (LDA) ou *Autoencoders* (AE). Uma abordagem usual para classificação de texto sob estas condições consiste assim em representar cada texto como um vetor resultante do *feature engineering* e aplicar-lhes um algoritmo de *clustering* (Allahyari et al. [2]). Os *clusters* resultantes são interpretados como as classes (idealmente a evidenciar cada um dos *clusters* uma forte coesão no seu conjunto de textos, bem como uma capacidade de se diferenciar face a cada um dos conjuntos dos demais *clusters*). Não existindo uma coesão entre exemplos de uma classe face às demais, este ruído de classificação pode ser mitigado por outros métodos através de heurísticas (Yu et al. [42], com a adição de amostragem negativa para introduzir contraste na aprendizagem), da intervenção do *domain expertise* associado a modelos pré-treinados (Meng et al. [24], a usufruir da semântica incorporada em modelos pré-treinados relacionando-a com a *label* de cada classe) ou apenas de modelos pré-treinados (Stammbach e Ash [34], com cada classe a emergir dos tópicos comuns dos textos mais próximos após a sua conversão em vetor de *embeddings* semântico "conhecido" pelos modelos pré-treinados).
Sabendo-se ser crítico para o processo de classificação de texto o acesso a um conjunto vasto de textos (bem) anotados, o que é em muitos casos difícil de assegurar, a adoção da aprendizagem não supervisionada ou da *self-supervised* com recurso a modelos pré-treinados tem elevado potencial, sendo que nestes últimos há que ter em conta que poderão ter dificuldades em lidar com textos de certos domínios de tecnicidade muito específica, entre estes a área de negócio do nosso caso de uso. Por esse facto e para melhor controlo do erro de classificação aquando da comparação entre modelos obtidos pelas distintas abordagens, opta-se aqui por realizar somente classificações de texto em aprendizagem supervisionada.

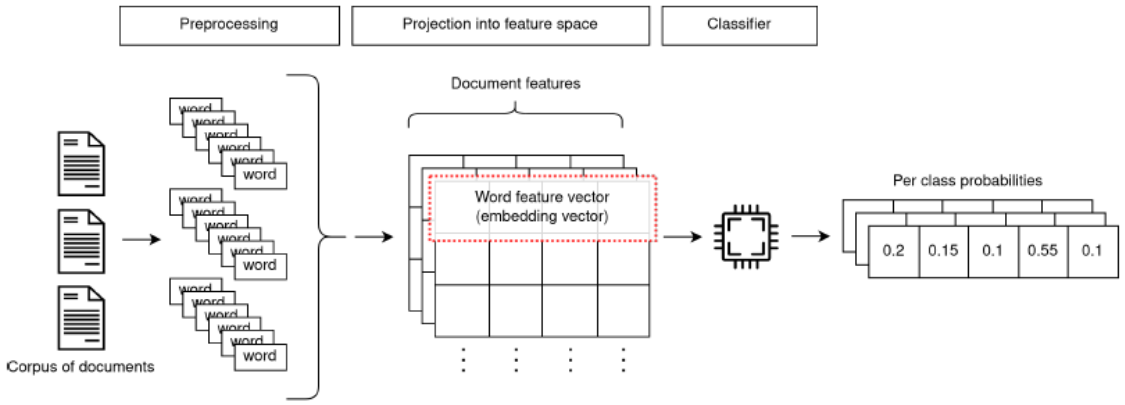
2.2 Abordagens Supervisionadas e Principais Diferenças

Com o crescimento exponencial de dados proveniente da transformação digital da sociedade e da forma como se organiza e relaciona, requerendo a geração de novos conhecimentos e instrumentos de apoio à gestão das organizações baseada neste vasto conjunto de dados em tempo útil, temos vindo a assistir ao surgimento de um novo paradigma [14] de procura e geração de conhecimento: o *data-driven*, i.e., basear a estratégia, atuação e gestão das organizações em função destes dados. No entanto, este paradigma coexiste com processos de geração de conhecimento liderados pelo *domain expertise*, quer seja pela complexidade e grau de especialização de uma área de negócio, quer seja pela necessidade de se justificar a atuação em função das características dos dados.
É possível assim observar-se duas abordagens aplicadas em tarefas de classificação de texto, consubstanciando-se em diferentes *pipelines* retratados por Gasparetto et al. [13] (ver figura 2.1), de etapas funcionalmente equivalentes mas de fluxos e métodos utilizados distintos.

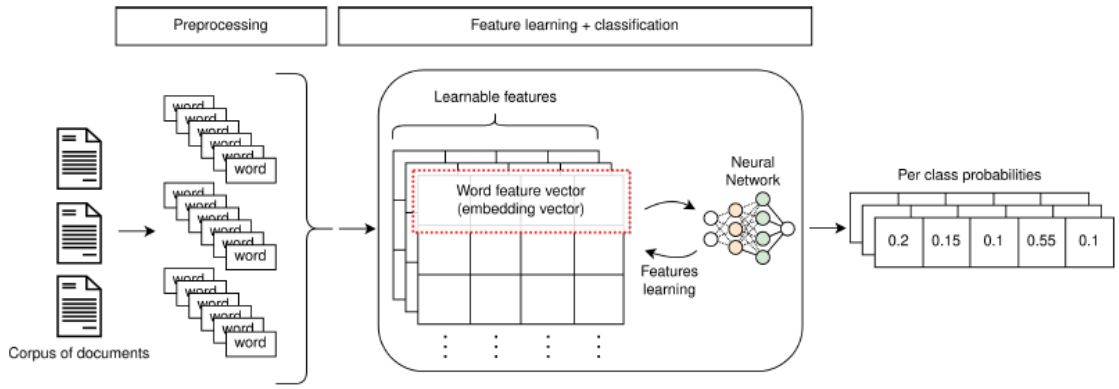
Em fluxos, diferenciam-se especialmente em:

(a) ***Pipeline* de Modelo(s) Clássico(s) de *Machine Learning***: fase de representação da linguagem, contendo métodos para extração de *features* e redução da dimensionalidade, separada da fase de treino do classificador, sendo a mais importante para a classificação por ser através dela que o classificador deterá a capacidade de diferenciar os textos e de possibilitar a interpretação dessa diferença. Requer a participação determinante do *domain expertise*;

(b) ***Pipeline* de Modelos de *Deep Learning***: fase de representação da linguagem interligada com o treino do classificador, permitindo capturar e relacionar amplamente a informação relevante nos dados de treino, e mapear por fim as *features*, gerando representações do texto contextualizadas e com significado semântico.



(a) Pipeline de MC de ML.



(b) Pipeline de Modelos de *Deep Learning*.

Figura 2.1: *Pipelines* de MC de ML e de Modelos DL.

2.3 Pré-processamento de Texto

Os dados de *input* da tarefa NLP de classificação de texto consistem justamente em documentos de texto cru, não estruturados. Carecendo de uma representação numérica que permita a atuação do classificador, deverão ser pré-processados e projetados num espaço que caracterize e diferencie os textos: o espaço das *features*. Como já referido, os MC de ML dependem fundamentalmente destes processos que se concluem com a *feature engineering* para a qualidade do seu desempenho, requerendo para tal de apurado tratamento de informação e do *domain expertise* para a interpretar. Já os DL conseguem automaticamente mapear e extrair as *features*. No entanto, também para estes modelos as decisões de pré-processamento em NLP se revelam importantes, sendo muitas vezes desvalorizadas. Se bem que Camacho-Collados e Pilehvar [8] argumentem, segundo estudos por si realizados, que uma simples delimitação de texto por *white spaces* poderá revelar-se igual ou melhor que a aplicação de técnicas de pré-processamento mais complexas (tais como a *lemmatization* ou *multiword grouping* via *n-grams*), quando nos deparamos com um *corpus* de natureza muito técnico tal simplicidade pode degradar desempenhos potenciais. Apresentam-se seguidamente os métodos de pré-processamento utilizados em classificação de texto, procurando identificar a abordagem que (mais) os aplica.

2.3.1 Normalização do Texto

Dados de textos contêm palavras desnecessárias e de diferentes formas de significado igual. Muitos algoritmos de análise e classificação, especialmente os estatísticos e probabilísticos, são muito sensíveis ao ruído e à redundância, com riscos para o seu desempenho ou mesmo para a sua credibilidade. Para mitigar estes riscos de qualidade de dados, utilizam-se vários métodos, apresentando-se de seguida um seu conjunto.

2.3.1.1 *Stop Words*

Os textos incluem um vasto conjunto de palavras não-informativas, i.e., sem importância semântica para os classificadores, que dificilmente definirão o significado do texto e a sua associação às classes, pelo que são habitualmente removidas. Esta decisão é muitas vezes controversa por enfraquecer/impedir o *Name Entity Recognition* (NER) ou a Extração de Expressões Relevantes, se realizada a remoção antes da sua aplicação, pois as *stop words* são por vezes parte integrante tanto de entidades como de expressões.

2.3.1.2 *Lemmatisation*

Lemmatisation é o processo de NLP de deflexionar toda a palavra para determinar a sua forma canónica ou lema, no pressuposto de que as formas flexionadas estão muito próximas ou fortemente associadas/correlacionadas.

2.3.1.3 *Stemming*

Utilizada com objetivo analítico similar ao da *lemmatisation*, este processo de NLP é ainda mais ambicioso ao reduzir cada palavra flexionada à sua raiz, removendo-lhe afixos. Difere assim por o seu resultado poder em alguns casos não ser verdadeiramente uma palavra (p.ex., a redução de *senate*, *senator* e *senators* a *senat*). Tanto este como o processo de *lemmatisation* podem alavancar o desempenho da classificação, mas trazem alguma controvérsia, pois palavras com significados distintos podem deter o mesmo lema ou raiz, passando a partir da aplicação destes processos a serem indistinguíveis.

2.3.1.4 *Part-of-Speech* tagging

Part-of-Speech (PoS) é a classificação das palavras do texto em função da posse de propriedades gramaticais semelhantes, tais como substantivos, verbos e adjetivos (do tipo *content words*). Esta operação é incluída no pré-processamento de texto sempre que se pretenda limitar o vocabulário para um subconjunto destas categorias, o que poderá ocorrer p.ex. na tarefa de Extração de Expressões Relevantes (ver 2.4.1.3 na página 12).

2.3.1.5 Outras Normalizações

Outros procedimentos de normalização de texto podem ser aplicados, tais como: tratamento de pontuação e de caracteres especiais, transformação de texto em *lowercase*, correção ortográfica, tratamento de calão.

2.3.2 Tokenização

A tokenização é o processo que parte o conteúdo da sequência do texto em pedaços, denominados as *tokens*, determinando a granularidade da posterior análise dos dados. Estabelece assim o vocabulário afeto ao *corpus* em estudo. Tradicionalmente geradas através de regras básicas de separação por espaço e pontuações, as *tokens* passam mais tarde a ser condicionadas por conceitos linguísticos.
Desenvolvimentos recentes ligados a modelos de DL passam a considerar este processo como *data-driven*, com melhores resultados, sendo que esta segmentação de frases em unidades (agora não totalmente correspondentes a *tokens* da língua original) já não tem uma motivação linguística mas sim analítica e computacional. No entanto, como os modelos de DL aprendem uma representação vetorial para cada *token* através dos dados de treino, e o conjunto destes vetores são matrizes de *embeddings* suficientemente dimensionadas para mapear toda a *token* às demais *tokens* pertencentes ao vocabulário (cada *token* é assim representada pelas demais *tokens* que a contextualizam nos dados do treino), existe a necessidade de limitar a dimensão do vocabulário de acordo com as condicionantes "tempo de processamento" e "memória disponível". Tem-se ainda assim especial atenção à minimização do número de palavras não representadas, as *out-of-vocabulary* (OOV). Apresenta-se de seguida algumas das técnicas desenvolvidas.

2.3.2.1 Segmentação de Texto (*N-grams*)

A técnica *N-gram* é a aplicação de regra de segmentação do texto para estabelecimento de conjuntos de n-*tokens* (de palavras ou sub-palavras[1]) que surgem em sequência no texto (um *2-gram* é um conjunto de 2 palavras que ocorrem consecutivamente no texto, e.g. "ocorrida em"; para *3-gram*, temos por exemplo "inflação superior a", etc.). Apesar de não ser uma representação do texto, pode caracterizá-lo e transmitir um contexto (a partir de *2-gram* as *tokens* nele integradas ficam "estruturalmente" relacionadas). Permite ainda promover análises sintáticas e fundamentar processos de extração de Expressões Relevantes (ver 2.4.1.3 na página 12).

2.3.2.2 *Byte Pair Encoding* (BPE)

Tokenização ligada aos modelos de DL, é uma segmentação do texto em sub-palavras (e.g, segmentação de *freestyle* em *free* e *style*). Decorre de um processo iterativo de aprendizagem do vocabulário de *tokens* contido no *corpus*, aquele inicialmente formado pelo conjunto de todos os caracteres únicos neste encontrados. Tendo por base o vocabulário conhecido/obtido no final da iteração anterior, o processo, em cada iteração, seleciona o par de *tokens* adjacentes mais frequente encontrado no *corpus* de treino (parte do *corpus* afeto à fase de treino a ser trabalhado pela parte do algoritmo denominada *token learner*), faz a sua junção em nova *token* e adiciona-a ao vocabulário, substituindo finalmente no *corpus* todas as ocorrências desse par de *tokens* adjacentes pela nova *token* (já em fase de segmentação do *corpus* a ser realizada pelo denominado *token segmenter*). O processo termina quando *k* junções de pares de *tokens*, valor definido para limitar a dimensão do vocabulário, se concretizarem.

2.3.2.3 *WordPiece*

A tokenização *WordPiece* [33] tem por objetivo a constituição de um vocabulário de sub-palavras que minimize a dimensão do *corpus* de treino à medida que o for segmentando em função do vocabulário que vai selecionando. Para tal, utiliza um *greedy algorithm*[2] para otimização do objetivo, o qual inicializa o vocabulário, tal como o utilizado pela tokenização BPE, com todos os caracteres únicos, e iterativamente vai construindo e aplicando versões de *Language Model(s)* (LM)[2] com base no vocabulário existente em cada passo, com vista a selecionar o par de sub-palavras que maximiza a verosimilhança do LM, realizando a sua junção num novo termo/*token* a incrementar o vocabulário. A verosimilhança de um par dever fazer parte do LM é medida pela frequência do par corrigida pela multiplicação das frequências individuais dos seus elementos (sub-palavras muito frequentes no *corpus* têm maior dificuldade de junção, pois fazem baixar o score). O algoritmo termina quando

[1] Desconstrução das palavras que abdica da "meta-informação" linguística que estas incorporam com o objetivo de "deixar os dados falar".

[2] Abordagem de resolução de problemas que escolhe em cada passo a melhor alternativa, sem levar em conta o impacto dessa escolha para os "ótimos" futuros e para o ótimo global final.

a verosimilhança esperada de uma nova junção cai abaixo de um determinado *threshold* ou a dimensão máxima do vocabulário, previamente fixada, é atingida.
É sobre esta tokenização que se baseia a correspondente utilizada pelo BERT (ver 2.5.2.4 na página 19), recente modelo de DL construído sobre a arquitetura *Transformer*, considerado um standard de *transfer learning* em NLP.

2.4 *Feature Engineering*

Segmentado o texto em *tokens* e mapeado em vocabulário, este deverá sofrer ainda uma conversão num espaço estruturado de *features* para que seja utilizado por um classificador, i.e., representado na forma vetorial para efeitos de modelação. Esta representação do texto pode procurar fazer essa conversão através da linguagem, da semântica, da captura dos contextos e/ou da caracterização dos tópicos dos documentos.

2.4.1 Extração de *Features*

A formalização do espaço de representação do texto nos documentos é realizada por distintos métodos de *feature extraction* distintos, desde os que registam, agrupam e ponderam as *tokens* (podendo ser *n-grams*, palavras ou sub-palavras) aos que constroem e atribuem vetores a cada *token* em função da sua relação com as demais *tokens*. Vamos ver seguidamente alguns exemplos.

2.4.1.1 *Bag-of-Words* (BoW)

Baseado na contagem de número de *tokens* em cada texto e sua alocação a um espaço de *features* derivado do vocabulário, este método apenas representa o texto como um conjunto não ordenado de *tokens*, ignorando estrutura e a sua proximidade (ainda que respeitando naturalmente a relação "forçada" entre "strings originais" pela utilização de *n-grams* em sede de tokenização, caso tenha sido assim requerida), bem como o relacionamento semântico entre *tokens* do texto para além da sua coexistência no texto. Pela sua simplicidade, é no entanto uma forma eficiente de extração que permite bons desempenhos a classificadores em NLP.

2.4.1.2 *Term Frequency-Inverse Document Frequency* (TF-IDF)

O clássico TF-IDF [17] representa o texto através de métrica que permite indicar a importância de uma *token* num texto (medido pela sua frequência absoluta), levando igualmente em conta quão rara é essa *token* no *corpus* (ver equação 2.1).

$$W_{t,d} = tf_{t,d} \cdot \log\left(\frac{N}{df_t}\right) \tag{2.1}$$

onde:

$tf_{t,d}$ = frequência do termo t no documento d;
df_t = total de documentos no *corpus* que contêm o termo t;
N = total de documentos no *corpus*;
$\frac{N}{df_t}$ = frequência inversa do documento (quanta informação dá o termo t ao documento d no contexto do *corpus*).

Esta métrica permite assim penalizar tanto as *tokens* pouco ou nada representadas no texto, como as que, ainda que com representação frequente no texto, estejam igualmente presentes num número elevado de outros textos.
Apesar do TF-IDF tratar o problema das *tokens* comuns a todos os textos, passando assim a relativizá-las, não evolui face ao BoW na representação do relacionamento semântico, uma vez que cada *token* é representada em cada texto descontextualizada.

2.4.1.3 Extração de Expressões Relevantes

Usualmente preparada com uma tokenização *n-gram*, eventualmente condicionada a PoS *tags* que detenham pelo menos umas das categorias substantivo e adjetivo, tem por objetivo encontrar automaticamente expressões (*keyphrases*) significativas em cada texto, que o descrevam e diferenciem, sendo assim relevantes para a sua classificação no conjunto de textos em estudo.
Tem como exemplos de métodos aplicados para realização da extração, o TF-IDF, o *Yet Another Keyword Extractor* (YAKE!) [9], o *TextRank* [25], baseado nos grafos de relacionamento das palavras nos textos, e o EmbedRank [4], já baseado em DL.

2.4.1.4 *Word Embedding*

Os métodos anteriores asseguram representações sintáticas das palavras mas não têm a capacidade de capturar o relacionamento semântico entre elas. O exemplo mais óbvio desta incapacidade é a representação de sinónimos realizada por estes, onde semanticamente são iguais ou muito próximas, mas cujas respetivas representações no espaço das *features* são ortogonais, o que significa, em matemática, que são totalmente diferentes.
Para solucionar esta incapacidade de relacionar semanticamente as palavras, surge a técnica de aprendizagem *self-supervised*[3] das *features* extraíveis de textos de um *corpus*, a denominada *Word Embedding*, através da qual, cada palavra[4] do vocabulário é mapeada num vetor multidimensional de números reais em função das demais palavras, tendo em conta as vizinhanças de palavras que esta palavra vai tendo ao longo dos textos do *corpus*. Em primeiras versões, o *embedding* de uma palavra era estático, i.e., uma palavra tinha uma única representação semântica, apesar de, muitas vezes, alterar o seu segundo

[3] O conceito de aprendizagem *self-supervised* decorre da autossuficiência do conjunto de dados de treino em promover a aprendizagem da informação que contém, habitualmente aprendendo a estrutura que lhe está subjacente. É muitas vezes promovida por uma *negative sampling* para aprender com o contraste face ao que (já) sabe.

[4] O conceito admite naturalmente *tokens* de outras tipologias, o que de resto é alcançado por métodos mais recentes.

significado apenas detetável via contexto (ou inclusivamente ser uma palavra homógrafa de outra).
Mikolov et al. [27, 26] são responsáveis pela primeira arquitetura desta técnica, o *Word2Vec*, utilizando para tal uma associação de duas redes neuronais para criar um vetor de elevada dimensionalidade para cada palavra, com a primeira a prever a palavra (que é do seu conhecimento, daí ser *self-supervised*) em função do conjunto de palavras que se encontram na sua vizinhança (modelo *Continuous Bag-of-Words* ou CBOW) e a segunda a encontrar palavras de elevada probabilidade de surgirem na vizinhança de determinada palavra (modelo *Skip-gram*). Surge, pouco depois, uma outra técnica de *Word Embedding*, a *Global Vectors for Word Representations* (GloVe) [29], agora baseada em modelo probabilístico sobre estatísticas de coocorrências de palavras, não evoluindo para modelo preditivo, como o *Word2Vec*, com a outra grande diferença de a aprendizagem da semelhança semântica entre palavras ser explicitamente (também) estendida para todo o *corpus*. O grande avanço conseguido por estes métodos é menos a previsão das palavras do que o mapeamento das palavras para *embeddings*, permitindo promover cálculos de distâncias entre palavras e com isso encontrar as que são próximas (sinónimos, palavras derivadas, ...) ou ainda qual a que dista de uma específica na projeção da distância estabelecida entre duas outras palavras (e.g., obter a palavra *Roupa* por projeção da distância entre as palavras *Cadeira-Mobília* à palavra *Camisa*).

2.4.2 Redução de Dimensionalidade

A aplicação de técnicas de redução da dimensionalidade ao resultado da formalização do espaço de representação do texto pode ser requerida em classificação de texto para as seguintes tarefas:

- Redução das *features* que geram o espaço de representação do texto a um número adequado às exigências do classificador em consumo de memória e tempo de processamento;
- Redução das *features* que geram um espaço de representação de texto de elevada dimensionalidade para mitigar o *overfitting* e evitar o *curse of dimensionality*;
- Conversão do espaço das *features* no espaço bidimensional (ou eventualmente tridimensional) para fins de visualização e interpretação do resultado do processo de aprendizagem das *features* e da classificação que por fim promovem.

2.4.2.1 *Principal Component Analysis* (PCA)

A PCA é uma técnica estatística que procura identificar um espaço de menor dimensão no qual os dados do espaço original caibam sem que percam demasiada variabilidade que dispunham inicialmente (diversidade de informação). Para tal, constrói um conjunto mais reduzido de novas *features* não correlacionadas (por projeção ortogonal, eliminando redundância de informação), como combinação linear das iniciais, maximizando no processo

a captura da variância original de forma a preservar a (quase) variabilidade/informação inicial detida. É da preservação da variabilidade que depende a qualidade da aplicação da técnica, ainda que a caracterização (e a eventual identificação do que mede) de cada uma das novas *features*, via projeção ponderada das originais, seja uma mais-valia.

2.4.2.2 *T-Distributed Stochastic Neighbor Embedding* (t-SNE)

O t-SNE é um método de redução da dimensionalidade de um espaço de elevada dimensão via *embeddings*. Para tal, o t-SNE converte as distâncias Euclideanas entre pontos do espaço original em probabilidades condicionadas que representam as similaridades entre eles. São estas similaridades que deverão ser preservadas na passagem para dimensionalidades mais baixas, passagem promovida por processo de minimização da divergência (função de custo) entre as distribuições das probabilidades condicionadas no espaço original e as do de destino, sendo a divergência medida pela *Kullback–Leibler divergence* [10].

2.5 Métodos de Classificação de Texto

A partir deste ponto fazemos a apresentação de um conjunto de classificadores de texto, iniciando por representantes da abordagem clássica e concluído com aqueles que se baseiam em redes neuronais profundas. A apresentação não será exaustiva e irá incidir em métodos e arquiteturas utilizados por classificadores singulares, não tratando, entre outros, os dois populares métodos de *ensemble* (combinação de classificadores) utilizados em ML, o *boosting* e o *bagging*, nem as técnicas de combinação ou concatenação de arquiteturas de DL standards.

2.5.1 Modelos Clássicos

Os métodos aplicados nos modelos clássicos eram, até recente, a abordagem adotada por defeito para classificação de texto. De aplicação agnóstica ao negócio em contexto, os seus principais desafios advinham da qualidade dos resultados conseguida na etapa anterior do pipeline de *Machine Learning*: a *feature extraction/engineering*, capaz de produzir *features* representantes da linguagem contida nos documentos do *corpus* devidamente interpretáveis, prontas a facilitar o trabalho do *classifier*. Nos pontos seguintes apresentam-se os métodos de classificação mais utilizados.

2.5.1.1 Classificador *Naïve Bayes* (NB)

O classificador *Naïve Bayes* (NB) é amiúde utilizado pela sua eficácia e simplicidade, tanto na perspetiva do algoritmo como da computação. A simplificação do cálculo advém do forte pressuposto de independência entre as *features* (daí o epíteto ao matemático Thomas Bayes que formulou o homónimo teorema sobre o qual o classificador se baseia). Esta independência permite ultrapassar a exigência computacional de um classificador *Bayes*, o qual, para cálculo da probabilidade conjunta entre cada classe e a combinação de *features*

para daí minimizar o erro de classificação, obrigaria ao produto de todas as probabilidades condicionadas de cada *feature*, dada toda a classe e todas as combinações de *features*.
É o método de classificação mais tradicional para categorização de texto, de demonstrada qualidade.

2.5.1.2 Classificador *K-Nearest Neighbors* (KNN)

O método *K-Nearest Neighbors* (KNN) é uma técnica não paramétrica utilizada para classificação de texto em vários domínios. Aplicado a uma amostra de textos não classificada, procura determinar a classe de cada um através da classe mais popular entre os k textos que se encontram mais próximos, tendo as *features* de cada texto como "coordenadas". É um classificador muito simples que não constrói qualquer modelo, nem existe qualquer treino, sendo todo o cálculo realizado durante a inferência (característica denominada *Lazy Learning*). A classificação pode ser muito rápida, na medida em que só depende da distância entre as coordenadas dos textos. No entanto, o tempo de computação depende da função de distâncias e ainda da dimensionalidade do espaço de procura dessas distâncias, com esta última a ter igualmente fortes impactos na noção de proximidade entre os diferentes *k-neighbors*, e consequentemente na qualidade da classificação ao longo do espaço (a conhecida *curse of dimensionality*).

2.5.1.3 Classificador Regressão Logística

A Regressão Logística foi um dos primeiros métodos de classificação, introduzida por David Cox em 1958. É um classificador linear, que estima as probabilidades das classes através da seleção e ponderação das *features* que, conjuntamente, mais diferenciam os casos exemplo. É usual ser utilizada para classificação binária, estimando-se assim a probabilidade de um acontecimento ou propensão a um comportamento, mas é facilmente aplicada ao caso multinomial através de formulação que inclui a função *softmax*.

2.5.1.4 Classificador *Support Vector Machine* (SVM)

Originalmente desenvolvido por Vapnik and Chervonenkis em 1963, o classificador *Support Vector Machine* (SVM) recebe já na década de 90, por B.E. Boser et al. [5], uma reformulação para versão não linear. Apesar de ter sido concebido para tarefas de classificação binária, é largamente utilizada em problemas multi-classe, gerando-se para o efeito múltiplos classificadores SVM (por ser um classificador binário, constroem-se soluções One-vs-One, um por cada par de classes).
Talvez a qualidade que mais o distingue seja justamente a capacidade que lhe é dada pela reformulação não linear, permitindo-lhe separar exemplos não linearmente separáveis, incapacidade que até então era mitigada com a permissão de violação das margens formadas pelos seus três *data points* que definem os vetores separadores do espaço que procedem à classificação. A ideia é proceder a uma transformação não linear dos dados iniciais, por aplicação de uma *kernel function*, i.e., uma função que mapeia os dados e os

projeta num espaço de maior dimensionalidade, expandindo-os de modo a permitir uma separação linear.

2.5.2 Modelos de *Deep Learning*

O surgimento dos métodos aplicados em Modelos de *Deep Learning* teve repercussão em todos os domínios da inteligência artificial, com especial incidência em processos ligados a *Computer Vision* (CV) e ao *Natural Language Processing*, o nosso NLP do qual faz parte a tarefa que aqui nos propomos realizar. Como atrás já referido, estes métodos foram suplantando os clássicos pela sua capacidade em gerar automaticamente mapas de *features* de elevada complexidade, com reduzida ou mesmo nenhuma intervenção do *domain expertise*, transformando sucessivamente os dados através de camadas de funções não lineares capazes de produzir classificadores de elevado desempenho. Estas transformações são consumadas nos múltiplos ponderadores (*weights*) - em *deep learning* comummente designados de parâmetros -, associados às ligações entre múltiplos neurónios[5] das várias camadas[5] (ou *layers*) da rede neuronal profunda (camada de *input*, camadas escondidas e camada de *output*), que vão sendo iterativamente alterados (as iterações são, em número, dependentes da quantidade de exemplos disponíveis e do *batch size*[5] de exemplos previamente definido a apresentar em cada iteração, bem como do número de épocas[5] igualmente pré-definido) pela parcela (gradiente) correspondente à sua responsabilidade no erro de classificação, na proporção da *learning rate*[5] definida, erro este (apurado via função *loss*[5]) que as suas combinações com as *features* (ajustadas ao longo da rede e durante o processo de otimização através de Otimizador[5]), enquadradas nas tais funções não lineares definidas ao nível dos neurónios e da camada de *output* (funções de ativação[5] respetivas), vão gerando, e que vai sendo reduzido ao longo do processo que o otimiza/minimiza.
Na classificação de texto e em desenvolvimentos mais recente, estes mapeamentos incorporados nas *features* passaram a permitir gerar representações semânticas, devidamente contextualizadas, para cada uma das unidades de um texto (ainda que possam deter iguais mapeamentos em situações particulares).
Nos pontos seguintes apresentam-se marcos na evolução dos modelos de DL, desde as três arquiteturas, até recentemente, mais utilizadas nos modelos, passando por modelos *encoder-decoder* que passam a permitir capturar (e armazenar) o contexto das sequências (uma das característica de textos) para utilização na fase de *decoding*, e pelo *attention mechanism* que permite ao *decoder* concentrar-se (ou ponderar) mais numa parte do texto de *input* quando se encontra a prever determinada *token output*, culminando na arquitetura *Transformer* que industrializou estes avanços, utilizando o *self-attention mechanism* para, em paralelo e simultaneamente, calcular para toda a *token* do texto um *attention score* (designado de *attention weight*) que mede a influência que cada *token* tem em todas as demais do texto.

[5]Hiperparâmetros de valor a definir e eventualmente a otimizar em função de desempenhos a medir em conjunto de validação e a avaliar em conjunto de teste após otimização.

Esta arquitetura permitiu constituir LM[2] pré-treinados em *corpora* de elevada dimensão para aprendizagem de representação de texto contextualizado através da previsão de palavras de acordo com contexto diversificados. Em posse da representação pré-treinada associada a cada palavra num vasto número de diferentes contextos, denominada de *context embeddings*, estes modelos pré-treinados são assim aplicáveis a outros domínios, promovendo-se assim a transferência do conhecimento da semântica (*transfer learning*), já capturada pelo LM em pré-treino, entre esses domínios, a custos baixos por apenas requerer a adaptação desse conhecimento à tarefa de classificação específica pretendida segundo as *labels* e exemplos para os quais o pretendemos especializar. Adicionalmente, é possível ainda reforçar esta especialização via treino profundo denominado *fine-tuning*, i.e., permitir que os *weights* pré-treinados também conheçam e se adaptem a estes exemplos, podendo comportar no entanto custos de processamento significativos.

2.5.2.1 Classificador *Feed-Forward Neural Network(s)* (FFNN)

Classificadores baseados em *Feed-Forward Neural Network(s)* (FFNN) encontram-se desenhados para aprender através de multi-camadas de número variado de neurónios totalmente ligadas (*full-connected* NN), onde cada camada escondida (*hidden layer*) apenas recebe ligação da precedente (a primeira *hidden* recebe da camada de *input*) e estabelece ligação com a seguinte (a última *hidden* com a camada de *output*). Vê o texto como um BoW, com a camada de *input* a ser preparada por um método de extração de *features*, tal como o TF-IDF ou o *Word Embedding*. Para a classificação multi-classe, o número de neurónios da camada de *output* deverá corresponder às *labels* a classificar, com função de ativação *softmax* e como função *loss* a *categorical cross entropy*.

2.5.2.2 Classificador *Convolutional Neural Network(s)* (CNN)

Classificadores baseados em *Convolutional Neural Network(s)* (CNN) foram originalmente desenvolvidos para tarefas de CV mas mais tarde também aplicados em NLP. A arquitetura de uma CNN consiste em:

1. Camadas de convolução, nas quais um *kernel* vai percorrendo um segmento de texto (ou uma região de uma imagem em tarefa CV) para dele extrair *features* (locais);
2. Camadas não lineares, onde uma função de ativação não linear é aplicada aos valores das *features* do ponto anterior;
3. Camadas de *pooling*, onde os diferentes conjuntos de *features* locais são agregados por operações *max-pooling* (ou *mean-pooling*) para apurar as *features* globais, mapeando-as;
4. Camadas *fully-connected* (*dense layers* da tipologia FFNN) no seu final para previsão/atribuição da classe ao exemplo (se *Full* CNN).

Uma das grandes vantagens da CNN é o mecanismo de partilha dos ponderadores por uso dos *kernels*, o que se traduz num reduzido número de parâmetros face ao que deteria uma

full-connected NN (ou FFNN), tornando-a de treino mais célere e por isso, amplamente utilizada em problemas de CV e NLP.

2.5.2.3 Classificador *Recurrent Neural Network(s)* (RNN)

Classificadores baseados em *Recurrent Neural Network(s)* (RNN) são uma escolha habitual em processamento de dados sequenciais, tais como texto (nosso caso) e vídeo. A arquitetura está capacitada para extrair a informação da estrutura das frases e assim capturar relações latentes entre palavras num mesmo contexto. O *input* de um modelo RNN para classificação de texto é habitualmente uma frase representada por uma sequência de *word embeddings*, com estes a entrarem no modelo um de cada vez.
A arquitetura da RNN original revelou-se não estar preparada para capturar dependências de longo prazo (relação de *timestamps* da sequência muito distantes), característica das sequências longas, o que ocorre em muitos casos práticos, devido principalmente a problemas de *vanishing gradient* (também por vezes de *explosion gradient*, com resultado prático análogo), i.e., os gradientes vão ficando próximos de zero à medida que o *back-propagation* do erro se vai aproximando das camadas mais próximas do *input*, deixando os ponderadores destas primeiras camadas praticamente inalterados em cada iteração[6], levando a que o otimizador não convirja.
É, por essa razão, desenvolvido o modelo *Long Short-Term Memory* (LSTM), a variante mais conhecida do RNN, desenhada para resolver este problema e poder capturar estas dependências de longo prazo. O LSTM consegue-o ao introduzir uma célula de memória para se recordar de valores por intervalos de tempo arbitrários, e ainda três *gates* (o *input gate*, o *output gate* e o *forget gate*) para regular o fluxo de entrada e saída de informação dessa célula.
O modelo RNN *Encoder-Decoder* foi outra importante realização evolutiva na arquitetura da RNN, particularmente importante em tarefas de tradução. Os modelos *encoder-decoder* aprendem a mapear o *input* na língua *A* no *output* da língua *B*, utilizando o *encoder* para comprimir o *input* num vetor de espaço-latente a utilizar pelo *decoder* para relacionar e prever o *output*. O espaço-latente deverá assim ter capturado a semântica subjacente ao *input*. Estes modelos são amplamente utilizados em tarefas de *sequence-to-sequence* (seq2seq) em NLP.
Ainda assim a RNN manteve a dificuldade de tratamento de textos longos, pelas limitações decorrentes a ser de natureza sequencial, quer seja pela exigência no carregamento em memória da sequência longa para computação do processo *encoder-decoder*, quer porque esta tende nestes casos, recorde-se, a esquecer as primeiras partes da sequência, dificultando a convergência.

[6] Por outras palavras, a rede tende a esquecer as partes iniciais da sequência, deixando a representação da semântica incompleta.

2.5.2.4 *Transformer*

A arquitetura *Transformer* (Vaswani et al. [36]), recente modelo *encoder-decoder* que implementa o *self-attention mechanism*[7] para poder processar todas as *tokens* simultaneamente (e não sequencialmente, como a RNN, evitando os problemas já descritos), permite ao *encoder* ter presente as demais *tokens* sempre que processa uma *token*, e assim aprender/capturar as dependências que as *tokens* têm entre si, sem incorrer em riscos de "esquecimento".
Uma outra sua especial implementação é o *Posicional Encoding*, aplicado antes de se iniciar o processamento do *encoder*, que impõe que os *embeddings* de uma mesmo *token* que surjam em diferentes posições nas frases/sequências detenham diferentes representações, assegurando assim que se incorpore a importante informação do posicionamento relativo das *tokens* (que de resto se perderia, uma vez que o processamento não é sequencial, como o da na RNN). A figura 2.2 apresenta a sua arquitetura, com o *encoder* (posicionado à esquerda) a produzir, como *output* de cada texto, *embeddings* que representam as suas *tokens* com a informação da dependência às demais *tokens* em determinado contexto, e com o *decoder* (ilustrado à direita) a gerar as probabilidades de as *tokens* surgirem em conjunto e em determinada posição dado um contexto ou tarefa de NLP em realização.
O Transformer (pela sua popularidade, vamos incluí-lo no nosso léxico) é um marco na área de NLP, levando a que os mais recentes modelos de DL sejam aplicações, construções e desenvolvimentos sobre esta arquitetura, e modelos seus representantes. A notoriedade destes modelos advém da qualidade do desempenho das suas versões pré-treinadas em distintas tarefas de NLP, tais como as ligadas à classificação de texto, quando aplicadas a outros domínios.
Como já referido, o *transfer learning* faz-se com o *Language Model(s)* (LM)[2], baseado num Transformer, pré-treinado em *corpora* genérica para desempenho de uma ou várias tarefas de classificação de texto, a ser especializado à tarefa, dados e *labels* pretendidos, podendo sê-lo ainda a um nível mais aprofundado através de *fine-tuning*. Na prática, ao LM é-lhe adicionado um *dense layer* de classificação (o nosso classificador específico) para esses fins. É assim possível (e muitas

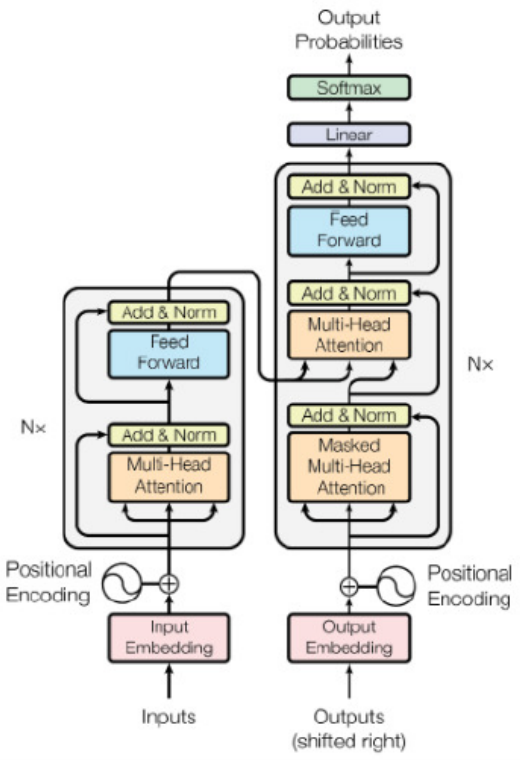


Figura 2.2: A arquitetura do Transformer.

[7] Baseado no *attention mechanism* introduzido por Bahdanau et al. [3] com o objetivo de disponibilizar ao *decoder* informação sobre a sequência completa do *input* do *encoder* que lhe permita diferenciar *tokens* de *input* e aprender quais delas são relevantes para a previsão da próxima *token* de *output*.

vezes desejável na perspetiva do desempenho sob novo domínio) permitir ao processo de *transfer learning* impactar na representação pré-treinada do LM através de *fine-tuning*, na expectativa de que este apenas sofra alterações mínimas, sob pena de não se aproveitar a sua eficiente capacidade de generalização, de se concretizar o risco de a perda dessa capacidade resultar em degradação do desempenho ou de vir a impossibilitar a (nova) convergência em tempo útil (um LM pré-treinado pode deter dezenas, centenas ou milhares de milhões de parâmetros, ou ultrapassar em alguns casos o bilião!).
Um dos Transformers mais utilizados é o *Bidirectional Encoder Representations from Transformers* (BERT) [11], cuja arquitetura é somente a componente *encoder* do modelo Transformer. O BERT foi pré-treinado como *Masked Language Model* (MLM), procedimento que mascara uma percentagem das *tokens* de *input* para que as tente prever (exemplo da sua capacidade de *self-supervised learning*). Como é bidirecional, permite-se aprender a relação entre as *tokens* mascaradas e os seus contextos dum lado e do outro. Também foi pré-treinado numa segunda tarefa, a *Next Sentence Prediction* (NSP), na qual são-lhe apresentadas duas frases e o BERT treina-se para prever a probabilidade de a segunda seguir a primeira (ser-lhe compatível nessa ordem), permitindo-lhe aprender "aprofundadamente" a relação das frases. O pré-treino neste último procedimento permite-lhe, por analogia, ter competências para realizar processos de *fine-tuning* em tarefas como *Name Entity Recognition* (NER), *Question Answering* (QA) ou *Text Classification*, o nosso caso, a utilizar como componente de *Natural Language Understanding* (NLU)[8].
Por fim, dá-se exemplo de uma outra família de Transformer - provavelmente o de maior notoriedade -, pré-treinado para a tarefa de *Text Generation* e capaz de transferir essa aprendizagem em processos de *fine-tuning* para o exercício justamente da componente de *Natural Language Generation* (NLG)[9], tarefa cujo objetivo é criar uma porção de texto coerente que dê continuidade a um dado contexto: os *Generative Pre-trained Transformer*, atualmente na quarta versão GPT-4[10] (suporte do conhecido *chatbot ChatGPT*), o qual, contrariamente ao BERT, utiliza somente o *decoder* da arquitetura Transformer.

2.6 Avaliação de Desempenho dos Classificadores

Seguidamente serão apresentadas as métricas de avaliação de desempenho dos classificadores a comparar. A avaliação é realizada em conjunto de teste previamente separado do *dataset*, com base em seleção de exemplos a respeitar uma amostragem aleatória estratificada segundo os temas/produtos em estudo.

Matriz de Confusão

Matriz que segmenta a frequência das classificações previstas por um classificador na ótica do seu cruzamento com a *ground truth*, quantificando e qualificando os segmentos nessa

[8] *Natural Language Understanding*, sub-domínio da NLP.
[9] *Natural Language Generation*, sub-domínio da NLP.
[10] É igualmente um *Multimodal Transformer*, i.e., capaz de executar tarefas que relacionam texto e imagem.

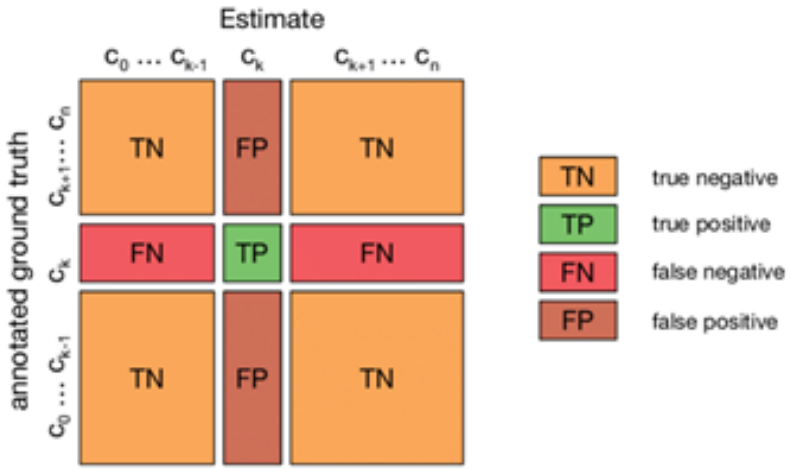


Figura 2.3: Matriz de confusão *multi-class* de $n+1$ níveis, na ótica da classe c_k.

perspetiva enquanto acertos (*True*) e erros (*False*). Na figura 2.3 pode ver-se a qualificação dos diferentes segmentos na ótica de uma classe c_k, onde no cruzamento entre a sua instância real e prevista encontramos os *true positive (TP)*, na coluna da previsão dessa classe cruzada com outras classes reais temos os *false positive (FP)*, na sua linha respeitante ao seu valor real cruzado com outras classes previstas os *false negative (FN)*, estando nos demais cruzamentos entre outras classes reais e respetivas classes previstas os *true negative (TN)*. Conforme já referido, a quase totalidade das métricas de desempenho utilizam estes segmentos da matriz, relacionando-os, com algumas a traçar a sua reação à variação do nível a partir do qual se classifica c_k como *positive* (*threshold*).

Accuracy

Métrica que avalia a proporção global de previsões corretas ($TP + TN$) do classificador. É dada pelo seguinte *score*:

$$\frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{2.2}$$

Recall

Mede a proporção de previsões corretas de determinada classe entre exemplos dessa classe.

$$\frac{TP}{(TP + FN)} \tag{2.3}$$

Precision

Mede a proporção de previsões corretas de determinada classe entre previsões nessa classe.

$$\frac{TP}{(TP + FP)} \tag{2.4}$$

F1-Score

Métrica que avalia simultaneamente o *precision* e o *recall* de um classificador, permitindo a seleção daquele que melhor equilibre a capacidade de realizar uma previsão correta de uma determinada classe de entre todas a previsões dessa classe com a capacidade de classificar corretamente os exemplos dessa classe. Pelo facto de calcular uma média harmónica entre estas métricas, é bastante sensível a um sofrível desempenho em umas das métricas, penalizando a avaliação.

$$\frac{2 \times Precision \times Recall}{Precision + Recall} \tag{2.5}$$

Curva *ROC* e a *AUC*

Denominada de ***R**eceiver **O**perating **C**haracteristic (ROC)*, avalia o compromisso entre o *recall* e proporção de falsos positivos (classificações incorretas em casos negativos, $\frac{FP}{(TN+FP)}$) à medida que se reduz o *threshold* para classificar cada exemplo como *positive*, i.e., se reduz a exigência da previsão de casos positivos permitindo que casos negativos sejam confundidos com positivos. Formada a curva *ROC*, apura-se a área que esta detém, denominada de ***A**rea **U**nder the **C**urve (AUC)*, indicando a capacidade de o modelo distinguir as classes estudadas.

Curva *Precision-Recall*

Exibe o compromisso entre a capacidade de realizar uma previsão correta de uma determinada classe de entre todas a previsões dessa classe e a capacidade de classificar corretamente os exemplos dessa classe à medida que se reduz o *threshold*.

2.7 Estado da Arte

Os métodos aplicados nos modelos clássicos eram, até recente, a abordagem adotada por defeito para classificação de texto. De aplicação agnóstica ao negócio em contexto, os seus principais desafios advinham da qualidade dos resultados conseguida na etapa anterior do pipeline de *Machine Learning*: a *feature extraction/engineering*, capaz de produzir *features* representantes da linguagem contida nos documentos do *corpus*, prontas a facilitar o trabalho do *classifier*.
Nos últimos anos, as arquiteturas utilizadas pelos modelos de DL têm dominado a tarefa de classificação de texto, pela sua capacidade em gerar automaticamente mapas de *features* de elevada complexidade por transformações não lineares sucessivas, com reduzido ou mesmo nenhuma intervenção do *domain expertise*, ao mesmo tempo que produzem classificadores de elevado desempenho.

2.7.1 Modelos Clássicos

Conscientes, por um lado, da popularidade do classificador NB na classificação de texto resultante da sua simplicidade e desempenho, nomeadamente a sua versão multinomial, o Multinomial Naïve Bayes, a qual toma um documento como uma sequência ordenada de ocorrências de palavras, mas cujas ocorrências são independentes (o pressuposto Naïve) e, por outro, do seu mais fraco desempenho face a classificadores com o SVM, principalmente quando treinados com poucos documentos, Kim, S.-B. et al. [18] deduzem e detetam um seu problema derivado do processo que formula a estimação dos parâmetros do teorema de Bayes, as probabilidades condicionadas das palavras w_i a cada uma das classes, ou sejam, $p(w_i \mid c)$ e $p(w_i \mid \bar{c})$, as quais permitem calcular a probabilidade de um determinado documento d_j pertencer à classe c, a $p(c \mid d_j)$: a formulação considera todos os documentos da classe c ou todos os documentos da classe $\bar{c}$ como um único documento, estimando assim estes parâmetros nesses "novos" documentos. Para reorientação desta estimação ao documento, propõem um processo de normalização da frequência das palavras por documento (a extração das *features* a ponderar palavras orientada ao documento, e não um único BoW) através da introdução do modelo *Multivariate poisson model for Naïve Bayes Text Classification*. Consideram que o uso da distribuição Poisson aplica-se porque a frequência da ocorrência de uma palavra num determinado documento (sendo a sua dimensão, o "tempo" de observação) pode ser explicada por esta distribuição, i.e., admitem que cada documento é gerado por uma Poisson multivariada (de palavras, naturalmente). Testam vários métodos de *normalização da frequência de cada palavra por documento* para estimação dos parâmetros das duas Poisson, entre eles, `RF` (frequência relativa), `SRF` (frequência relativa *smoothed*) e o score do conhecido LM utilizado em recuperação de informação `BM25` [30] (o método BM25 inclui tamanho do documento e saturação da frequência da palavra). Tomando na experimentação como *baseline* de desempenho o modelo Multinomial Naïve Bayes, indicado como `multi.`, e confrontando o seu com modelo *benchmark* baseado em

SVM, estado-da-arte em modelos clássicos na altura, aplicando os três modelos a *corpora* para classificação de texto que inclui `Reuters21578` (21 578 notícias de 1987 publicados pela Reuter Ltd) e `20 Newsgroups` (19 997 artigos recolhidos em 20 diferentes grupos de media), obtêm resultados que lhes permite concluir que:

- O modelo proposto apresenta melhor desempenho que o modelo tradicional multinomial, com pequeno acréscimo de custos em tempo e espaço, sendo assim útil para a construção de classificadores de texto probabilísticos;
- A frequência relativa e a frequência relativa *smoothed* são suficientes para normalizar a frequência dos termos face ao modelo tradicional. No entanto, observa-se um efeito substancial quando se utiliza o método BM25 como método de normalização da frequência dos termos;
- Apesar do classificador proposto ficar aquém do classificador SVM, consideram que terá espaço para utilização pela sua simplicidade e eficiência, o que lhe dará maior elasticidade para se manter atual.

H. Lodhi et al. [22] indicam que nem sempre os dados de *input* podem ser descritos por vetores de *features* explícitos, dando o exemplo de sequências biológicas, imagens e documentos de texto. Nestes casos, a *feature extraction* pode ser complexa e onerosa para solucionar um destes problemas, e que a intervenção do *domain expertise*, além de extensiva, pode conduzir a perdas de informação durante o processo. É aí que os métodos de *kernel* se colocam como alternativa para efetivar a *feature extraction*. Utilizando o classificador SVM, propõem um nova abordagem para categorizar documentos de texto baseada numa *kernel* especial, sendo esta um produto interno aplicado no espaço das *features* gerado através de todas as subsequências de texto de tamanho k. Promove assim uma arquitetura de classificação de texto baseada em *kernel* de subsequências de strings do texto, o qual mede a semelhança entre documentos justamente pelo produto interno de *features* extraídas via TF-IDF das subsequências de ordem k desses documentos. Esta abordagem compara bem com a *kernel* standard aplicado a *features* extraídas como BoW, o qual mapeia o documento, inicialmente representado em vetor onde cada entrada indica a presença ou ausência de cada *feature*, para vetor de maior dimensionalidade. Observam que apesar da *kernel* não incorporar qualquer conhecimento da linguagem utilizada no *corpus* `Reuters21578`, consegue capturar informação da semântica intrínseca da linguagem, a ponto de conseguir ultrapassar alguns classificadores *benchmark* sobre o mesmo *corpus*.

Já E. S. Tellez et al. [35] propõem-se inovar em *feature engineering* na classificação supervisionada de *Tweets* para identificação de género e idioma do utilizador Twitter que o enviou, introduzindo um esquema de ponderação de cada *feature* (cada *token* de conjunto extraído em processo de tokenização de *n-grams* de palavras e em processo de seleção de relações semânticas mais prováveis entre palavras pertencentes a vizinhança "próxima" segundo o

modelo *Skip-gram*) em função do valor da entropia subjacente à sua distribuição por classe, onde a um valor elevado corresponde uma sua distribuição uniforme ao longo das classes, resultando assim num ponderador baixo. O desempenho deste esquema de ponderação das *features* é comparado com a extração de *features* pelo TF-IDF através de utilização do classificador SVM com função de *kernel* linear, uma vez que pretendem selecionar um classificador de excelente desempenho num *input* de elevada dimensionalidade, mas não o pretendem "otimizar", já que o seu foco é avaliar o ponderador de entropia. O desempenho conseguido em identificação do idioma é bastante bom, obtendo acréscimos substanciais em *recall*, *f1-score* e *accuracy* face à extração TF-IDF.

2.7.2 Modelos de *Deep Learning*

Sustentando-se na hipótese de que a representação do texto, a consubstanciar-se numa arquitetura de um modelo de classificação de texto capaz de gerar automaticamente mapas de *features* para classificadores DL de elevado desempenho, pode beneficiar da incorporação da estrutura dos documentos, já que nem todas as partes do documento serão igualmente relevantes para a classificação e a procura das partes relevantes envolve a modelação da interação entre as palavras, Z. Yang et al. [41] propõem um modelo de DL de arquitetura *Hierarchical Attention Network* para a classificação. O modelo tem duas características que o distinguem:

(i) Tem uma estrutura hierárquica para espelhar a estrutura dos documentos (i.e., palavras formam frases e frases formam documentos), pelo que constrói a representação do documento por primeiro construir a representação das frases, e depois agregar estas representações na representação do documento;

(ii) Tem dois níveis de *attention mechanisms* aplicados aos níveis da palavra e frase, capacitando-o a diferenciar a atenção dada aos distintos conteúdos quando constrói a representação do documento.

De arquitetura *Hierarchical Attetion Networks* (HAN) e baseado na extração de *features Word Embedding*, o modelo constrói assim progressivamente um vetor do documento através da agregação de palavras importantes para dentro de vetores de frases, agregando depois importantes vetores de frases a vetores de documentos. As experiências conduzidas no *corpora Yelp*, *IMBD review*, *Yahoo Answer* e *Amazon review* demonstram que a arquitetura proposta ultrapassa métodos anteriores, com pares de *feature extraction* e de classificação transversais às abordagens[11], e por uma margem substancial.

Já X. Zhang et al. [43] abdicam de qualquer estrutura prévia nos documentos para estudo da sua representação, considerando que o texto neles contido pode ser tomado como um

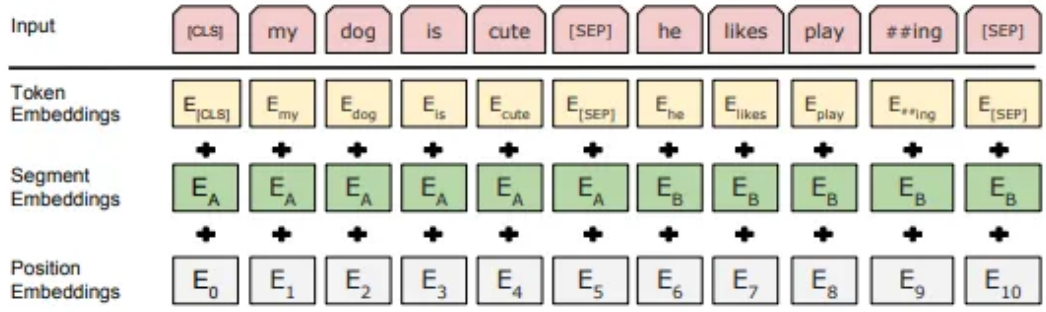
[11] Exemplos de pares de *feature extraction* e de classificação utilizados como *baselines*: SVM + Bigrams, SVM + TextFeatures, LSTM, CNN-char, CNN-word, CNN-GRNN, LSTM-GRNN.

sinal emitido ao nível do carácter, em série temporal (sequência), passível assim de ser representado por uma CNN. Este entendimento consideram-no admissível derivado a estudos anteriores, onde *convolutional networks* foram aplicadas na classificação de texto representado em "*word embeddings*" que não incorporavam conhecimento sintático ou semântico da linguagem. Esta sua experiência permite-lhes provar este entendimento, conseguindo demonstrar que, em *datasets* de elevada dimensionalidade, uma CNN profunda não requer qualquer conhecimento sobre as palavras, e logo de elementos hierárquicos superiores, pelo que a exploração ao nível dos caracteres é admissível. Entendem que esta abordagem de *feature engineering* será fundamental para um sistema único de representação do texto que se aplique às diferentes línguas.
Provam que as *character-level ConvNets*, as *convolutional networks* por si desenvolvidas que mapeiam *features* construídas sobre a sequências de caracteres, apresentam resultados bastante competitivos, comparativamente a MC de ML baseados em representações de texto que fazem recurso a estruturas da linguagem, tais como BoW, *n-grams* e TF-IDF, mas também a Modelos de DL, de arquitetura RNN (e a sua variante LSTM) ou CNN, que classifiquem espaços de *features* baseados em *Word Embeddings*. Consideram, por fim, como a mais importante conclusão da experiência o facto de as *character-level ConvNets* serem um método efetivo para a tarefa de classificação de texto, método que não necessita de trabalhar ao nível das palavras.

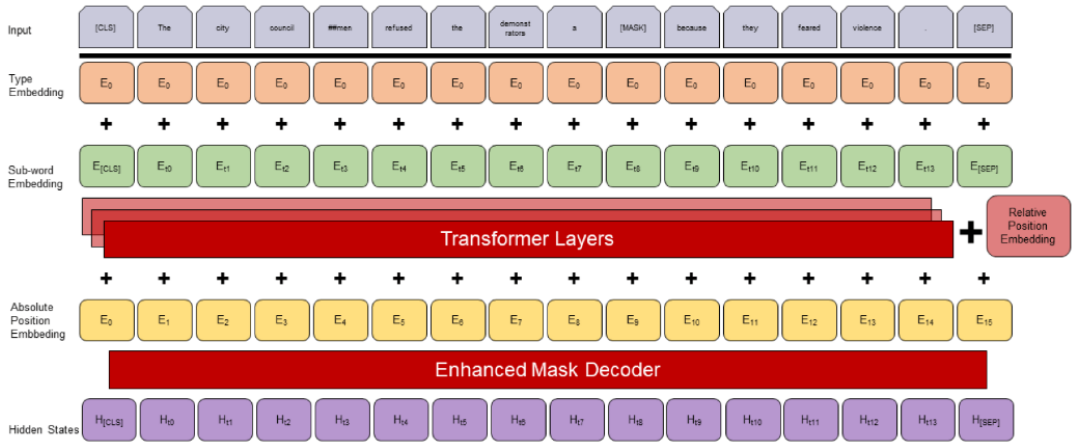
Procurando usufruir da representação da linguagem *state of the art*, preconizada pelo BERT, construída por modelação da linguagem via captura da sua semântica e com potencial de generalização e de aplicação a outros domínios, A. Adhikari et al. [1] promovem o que consideram ter sido a primeira aplicação deste Transformer na tarefa de classificação de documentos com o objetivo de igualmente capturar essa competência. Deparam-se, no entanto, com algumas dúvidas sobre a sua adequação (recorde-se que este modelo não foi inicialmente treinado para esta tarefa 2.5.2.4 (página 19), apesar de os seus autores [11], desde logo, lhe outurgarem essa capacidade de a vir a realizar com sucesso), nomeadamente por um documento poder ter características comuns a várias classes (ser *multi-label*). Por outro lado, sabendo-se que o plano seria aplicar o BERT, pré-treinado em tarefas de NLP distintas da pretendida, a *datasets* de documentos (*corpora benchmark*) de elevada dimensão, realizando assim o *fine-tuning* multi-domínio para captura da generalização, o facto de o BERT resultante, agora especialista em classificação de documentos, manter as centenas de milhões de parâmetros (o BERT_{large} tem 350 milhões!), tal iria certamente ter um custo de computação elevado quando fosse reutilizado como LM noutras aplicações. Para mitigar esta questão, aplicaram o processo de destilação de conhecimento em redes neuronais de G. Hinton et al. [16] para transferir conhecimento do BERT_{large}, já num primeiro passo adaptado por *fine-tuning* à classificação de documentos, a um modelo substancialmente mais pequeno e de arquitetura distinta (LSTM), com cerca de 100× menos parâmetros. Este modelo revelou-se conseguir um desempenho ao nível do BERT_{base} (110 milhões de parâmetros). Ao fazê-lo, conseguiram estabelecer um novo paradigma na

classificação de documentos e demonstrar que o conhecimento detido pelo BERT pode ser transferido para modelos de custo computacional bastante mais modesto. Confrontando o seu desempenho com resultados de um conjunto de modelos transversal a diferentes abordagens e arquiteturas, cujas aplicações aos *datasets benchmark* ao longo dos tempos se foram constituindo como *baselines* de classificação de documentos, os autores consideram que elevaram a *baseline* para esta tarefa ao realizar o *fine-tuning* ao BERT, ao mesmo tempo que conseguiram disponibilizar essa competência a custo bastante mais baixo ao inferir os parâmetros 40× mais rápido.

Fundamentalmente mais preocupados com a qualidade dos resultados dos LM pré-treinados, baseados na arquitetura Tranformer, e na sua capacidade de generalização a outros domínios via *fine-tuning*, quer em tarefas de NLU[8], quer em tarefas de NLG[9], He et al. [15] propõem um novo modelo, de arquitetura distinta, denominado *Decoding-enhanced BERT with Disentangled Attention* (DeBERTa), o qual faz evoluir os modelos BERT e RoBERTa[12] [21] via duas novas técnicas.



(a) A representação do *input* pelo BERT.



(b) A representação do *input* e a arquitetura do DeBERTa.

Figura 2.4: A representação do *input* no BERT e no DeBERTa.

A primeira "desembaraça" o *attention mechanism* preconizado pelo BERT, onde o *input* de

[12] Acrónimo de *Robustly Optimized BERT pre-training Approach*, considerado como uma melhor versão do BERT, mas (ainda mais) especializado na tarefa MLM.

cada *token* (ver figura 2.4(a)) é representado pelo vetor soma entre o seu *token embedding* (conteúdo) e o seu *embedding* posicional[13], para passar a representá-la através de dois vetores que codificam o conteúdo da *token* e a sua posição, sendo a aprendizagem realizada pelo *Disentangled Attention Mechanism* ao longo dos seus *multi-head attention layers* através do cruzamento das matrizes destas duas informações (consultar figura 2.4(b)).
A segunda, realizada pelo *enhanced mask decoder* (ver figura 2.4(b)), é usada para incorporar os posicionamentos absolutos das *tokens* no *layer* do *decoder*, substituindo o *layer* de *output softmax* do BERT que por fim produz a previsão da(s) *token(s)* "mascarada(s)" (recorde-se que tal como os seus precursores, é pré-treinado em MLM e NSP).
É igualmente melhorada a sua capacidade de generalização em processos de *fine-tuning* (com a introdução de método de regularização da variabilidade dos *embeddings* das *tokens* entre modelos), possibilitando uma melhor especialização a um domínio.
A incorporação destas técnicas reforçam a capacidade deste Transformer na compreensão do contexto e melhoram a qualidade dos seus resultados em distintas tarefas de NLP, superando claramente o desempenho do BERT, e consistentemente o do RoBERTa, feito obtido transversalmente aos *datasets benchmarks* de tarefas NLP.
Salienta-se que o DeBERTa encontra-se amplamente difundido no setor tecnológico e na academia, utilizado como ferramenta habitual para realização de tarefas de NLP. A sua versão mais robusta, o DeBERTa$_{large}$, detém 48 *layers* com 1.5 biliões de parâmetros, um ***Large Language Model* (LLM)** considerado um ***Foundation Model* (FM)** genérico, modelo capaz de estruturar treinos em *fine-tuning* sobre um conjunto muito vasto de dados de domínios diversos, e de o realizar, como pretendido, para diferentes tarefas NLP, gerando desse modo modelos NLP especializados, ou mesmo FMs de domínio mais específico.

Este é o caso do **Albertina PT-*** [31], desenvolvido, em parceira, por investigadores do NLX | Grupo da Fala e Linguagem Natural da Faculdade de Ciências da Universidade de Lisboa (FCUL) e investigadores do Laboratório de Inteligência Artificial e Ciência de Computadores da Faculdade de Engenharia da Universidade do Porto (FEUP), o qual foi lançado no passado mês de maio. Para alavancar a capacidade de codificar e representar a língua portuguesa através de um LLM, desenvolveu-se este **FM para Português** em 2 variantes da língua: o **Português Europeu de Portugal** (PT-PT) e o **Português Americano do Brasil** (PT-BR). Como indicado, este *encoder* parte do DeBERTa, sendo pré-treinado sobre *datasets* em língua portuguesa. A variante PT-PT, o Albertina PT-PT, detém 900 milhões de parâmetros, 24 *layers* e uma dimensionalidade nos *embeddings* de 1536 (o dobro do BERT), tendo já perto de 2000 downloads após cerca de 1 mês da sua publicação. Pela sua natureza, este *encoder* somente tem capacidade de intervir em atividades de NLP.

[13] Em tarefas de QA e NSP, acresce ainda o *embedding* de segmentação do texto.

3

Solução Técnica

3.1 Introdução

A solução técnica para a realização de teste comparativo entre abordagens de desenvolvimento de classificadores de texto, uma baseada na aplicação de métodos clássicos de *Machine Learning* e a outra a suportar-se em métodos de *Deep Learning*, destinados a apoiar o Cliente de produtos financeiros na formalização de reclamação, incorpora as habituais etapas de desenvolvimento destes modelos de NLP, mas implementados segundo *pipelines* diferenciados, conforme evidenciado no esquema seguinte (figura 3.1).

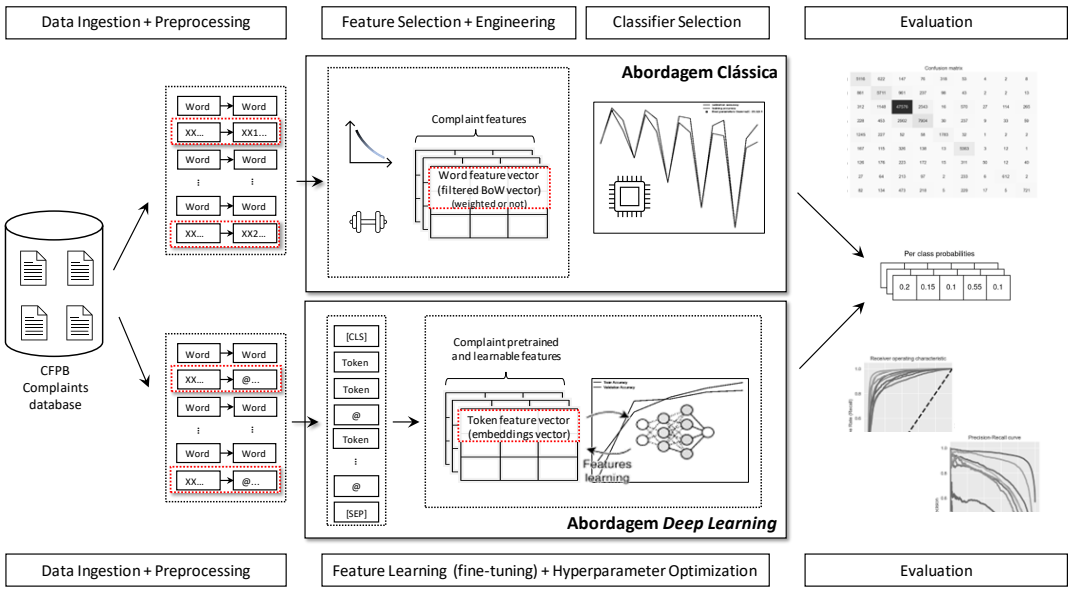


Figura 3.1: Arquitetura do teste comparativo de abordagens técnicas aplicadas no desenvolvimento de classificadores de texto.

Na perspetiva do processo, a etapa inicial, a respeitante ao pré-processamento do texto das reclamações, e a final, a relativa à avaliação dos classificadores, são idênticas, com ligeira nuance na primeira (mais à frente indicada). Já as fases nucleares de estruturação

do espaço das *features* e de treino de classificador apresentam *pipelines* e componentes distintos, a serem aprofundados nos pontos seguintes.
A apresentação desta solução técnica é assim estruturada segundo estes componentes, sendo que partirá do ponto comum relativo aos dados base e seu tratamento inicial, e culminará na comparação dos classificadores através da sua avaliação de desempenho, ponto igualmente de convergência do processo.

3.2 Fonte de Dados

Conforme indicado em 1.1, este trabalho parte de informação das reclamações provinda da *Consumer Financial Protection Bureau* (CFPB) [7], agência do governo federal dos Estados Unidos, responsável, desde julho de 2011, pela recolha das reclamações de Clientes relativas aos produtos e serviços financeiros disponibilizados pelas instituições financeiras a operar no mercado norte-americano. Dada a especificidade do dados em questão e a riqueza do *dataset* identificado, considerou-se desnecessária a existência de um outro *dataset* para generalização de resultados.
A base de dados CFPB disponibilizada é composta por um vasto conjunto de campos, do qual foi selecionado aquele que compõe o *dataset* de trabalho, de seguida tabelado (ver tabela 3.1).

Tabela 3.1: *Metadata* do *dataset* (recolhida na BD da CFPB).

Campo	Descrição	Formato	Nota
Data de receção	A data em que a CFPB recebeu a reclamação	*datetime*	—
Produto	Tipo produto identif. na reclamação p/ Cliente	texto plano	Variável categórica
Inst. Financeira (IF)	IF alvo de reclamação	texto plano	Variável categórica
Texto da reclamação	Texto submetido no campo da reclamação "*What happened*"	texto plano	Texto incluído após consentido do Cliente e ser retirada inf. pessoal
ID da reclamação	Número único que identifica a reclamação	numérico	—

A informação base do estudo aqui apresentada foi recolhida a 03 de dezembro de 2022. Apesar da série de reclamações se iniciar a meados de 2011, aplicou-se um filtro a partir de maio de 2017 por questões de uniformização de informação, uma vez que nessa altura a CFPB procedeu a uma alteração de como, entre outros, os produtos financeiros são agregados, advertindo que a *Consumer Complaint Database* apresenta no seu histórico os produtos de forma consistente às opções disponíveis no formulário na altura em que o Cliente submeteu a reclamação. A aplicação desse filtro permitiu, à data da recolha, ter-se

acesso a cerca de 1 milhão de reclamações.
Em prévia exploração de dados e análise da sua qualidade, detetou-se a existência de reclamações consideradas *outliers* por apresentarem valores extremos em número de palavras (p.ex., uma reclamação com 6314 palavras). Por dificilmente este caso (e outros análogos) ter condições de representar outras reclamações, procedeu-se à aplicação de um filtro de *outliers*, retirando-se todas as reclamações que estivessem nesta métrica para além do percentil 97,5%[1].
Por outro lado, foram igualmente detetados *outliers* em número reduzido de palavras. Neste caso, a criticidade decorre da importância de encontrar, para toda a reclamação, afinidades com as demais reclamações da mesma classe. Ora, existindo escassez de *features* que expressem essa afinidade, resolveu-se impor um posicionamento mínimo relevante em termos de frequência de palavras, devendo este ser superior ao quartil 2[2]. Exibe-se seguidamente algumas estatísticas sobre o *dataset* de partida e como fica após aplicação de filtro de *outliers*.

Tabela 3.2: Estatísticas do *dataset* original e após filtro *outliers*.

Dataset (filtro)	# Reclamações	# Classes	avg#w[a]	min#w[b]	max#w[c]
CFPB	955 192	9	185	1	6314
CFPB (s/ Q1 e outliers 97,5%)	689 453	9	201	65	772

[a] número médio de palavras por reclamação (inclui anonimizadas).
[b] número mínimo de palavras encontrado em todas as reclamações (inclui anonimizadas).
[c] número máximo de palavras encontrado em todas as reclamações (inclui anonimizadas).

Este não será, contudo, o conjunto de reclamações que será utilizado pelo estudo, uma vez que sofrerá nova redução, que será indicada no próximo ponto.

3.3 Pré-processamento do Texto

Esta primeira fase de estruturação do texto das reclamações, no caminho da definição do espaço das *features*, é, em NLP, habitualmente protagonizada pelas tarefas de normalização e tokenização. Por questões de abstração e uniformização processual entre abordagens, vamos colocar a tokenização na fase seguinte do processo de desenvolvimento dos classificadores, justificando-se por, no caso dos modelos de DL, ser um procedimento aplicado no âmbito da preparação e configuração do treino do classificador e por, no caso dos MC de ML, fazer desde logo parte do processo de *feature selection*, ao ser-lhe impostos limites

[1] Critério frequentemente usado em contextos estatísticos para exclusão de *outliers*.
[2] Este critério estabelece o valor mínimo de palavras que deverá deter uma reclamação, sendo mais à frente recuperado devido ao impacto da anonimização imposta pela CFPB neste "mínimo revelante" de palavras, conceito que nessa altura estará ligado à necessidade de se garantir uma "diversidade semântica mínima".

na formação do vocabulário logo numa sua primeira aplicação, ainda que nessa fase se trate de uma extração BoW.

3.3.1 Normalização

Relativamente à normalização, a nossa solução técnica optou por intervir o menos possível na semântica do texto contido nas reclamações, decisão tomada por questões de princípio, mas reforçada pela tecnicidade da terminologia esperada em reclamações relativas a produtos e serviços financeiros, onde termos técnicos ocorrem com frequência.
No entanto, houve necessidade de proceder a uma intervenção para tratamento dos "caracteres especiais" existentes nas reclamações, em virtude de lhes ter sido aplicada a devida anonimização. A anonimização realizada pela CFPB substitui toda a entidade a anonimizar pela *token* 'XXXX', podendo esta não estar separada de outra *token*, nomeadamente pontuação. Dada a extensão observada da anonimização e pretendendo evitar-se uma relevância exacerbada a dar a esta *token* durante a construção do espaço das *features*, procedeu-se ao seu tratamento em processo comum mas em duas versões distintas, adaptando-o aos processos de tokenização (e também de *Feature Engineering*), de tipologia igualmente distinta, que o texto das reclamações irá posteriormente ser alvo em cada uma das abordagens, a saber:

- Nos **MC de ML** promove-se uma substituição que torne cada instância anonimizada 'XXXX' como única, evitando que a tokenização inicial aplicada sob a técnica *N-grams* (ver 2.3.2.1) venha a dar-lhes atenção;

- Já nos **Modelos de DL**, promove-se a sua substituição pelo carácter especial '@', evitando-se que o processo de tokenização aqui aplicado, a *WordPiece* (ver 2.3.2.3), venha a dar demasiada importância aos caracteres da *token* 'XXXX', ao dividi-la em sub-palavras (corpo inicial mais prefixos ##xx) e, por sua vez, quantificá-las, pela esperada elevada frequência de ambas, como não agregáveis.

Como resultado deste processo de tratamento dos dados anonimizados, veio mais tarde, já durante a fase de *Feature Enginnering*, a identificar-se a necessidade de voltar a ser aplicado o critério de número mínimo de palavras relevantes, uma vez que um conjunto não despiciente de reclamações deixava de ser representado por *features*, ao deter elevada concentração de palavras anonimizadas sem que as demais compensassem esta perda por igualmente não serem selecionadas em processo correspondente.

3.3.2 *Dataset* para Desenvolvimento dos Modelos

Para se assegurar a diversidade semântica mínima, em risco por se ter identificado uma elevada proporção de reclamações com um número elevado de palavras anonimizadas[3],

[3] Uma reclamação tem, em média, 13 anonimizações; 25% das reclamações têm pelo menos 16 anonimizações; uma reclamação apresenta uma concentração média de palavras anonimizadas de 7%, mas para cerca de 1% das reclamações mais de 50% das suas palavras são anonimizações.

promove-se nova aplicação do critério "mínimo relevante" de 65 palavras, mas agora contabilizando apenas palavras "qualificadas", i.e., que tenham carga semântica.
Por fim, dadas as sentidas limitações da infraestrutura tecnológica[4,6] face à complexidade dos algoritmos aplicados, optou-se por limitar o *dataset* destinado ao desenvolvimento de modelos[5] às reclamações realizadas nos últimos dois anos, o biénio 2021-2022.
A tabela 3.3 apresenta o *dataset* sobre o qual o desenvolvimento dos modelos se realizará.

Tabela 3.3: Estatísticas do *dataset* de desenvolvimento.

Dataset (filtro)	# Reclamações	# Classes	avg#w[a]	min#w[b]	max#w[c]
CFPB (c/ diversidade semântica mínima)	659 902	9	194	65	771
CFPB (*DataSet* de desenvolvimento)	308 348	9	186	65	771

[a] número médio de palavras por reclamação (exclui anonimizadas).
[b] número mínimo de palavras encontrado em todas as reclamações (exclui anonimizadas).
[c] número máximo de palavras encontrado em todas as reclamações (exclui anonimizadas).

3.3.3 Partição do *Dataset* para Controlo do Erro

Para desenvolvimento dos modelos, tanto dos representantes de MC de ML e como de Modelos de DL, foi realizada a habitual partição do *dataset* em três outros conjuntos de dados para que desempenhem funções de treino, validação e teste. A proporção do *dataset* considerada para cada um foi:

- 40% destinado ao **treino**, que será acompanhado com outros 30% destinado à **validação**, sobre os quais se controlará o *overfitting* e se otimizarão hiperparâmetros dos modelos;

- 30% destinado ao **teste** para inferência do verdadeiro erro de cada modelo.

Assegurando que a distribuição original das classes fosse mantida nas três partições, foi realizada uma amostragem aleatória estratificada proporcional, de acordo com os tipos de produtos financeiros alvo de reclamação, as nossas 9 classes. A figura 3.2 atesta a qualidade da amostragem.
Importa aqui indicar que será sobre este conjunto de treino que se realizarão os processos de *feature extraction/selection* e *feature engineering*, etapas da abordagem clássica, bem como se testará e parametrizará a tokenização aplicada na abordagem *deep learning*. Na terminologia utilizada em NLP, este conjunto de treino será o ***corpus*** do nosso estudo.
Contudo, conforme referido nos objetivos do estudo (ver ponto 1.2), houve ainda a necessidade de se recorrer a novos processos de amostragem durante as fases de treino, avaliação

[4] Inclusivamente a nível de *Software*, onde algumas implementações de métodos analíticos advertem para a impossibilidade de serem executados sobre elevadas dimensões de dados por o formato das estruturas de dados que preveem não ter capacidade para as instanciar.
[5] Considera-se ainda que o conjunto das reclamações que não participam no desenvolvimento, poderá ser utilizado para realização de *backtest* com vista a avaliar-se a capacidade de generalização dos modelos desenvolvidos.

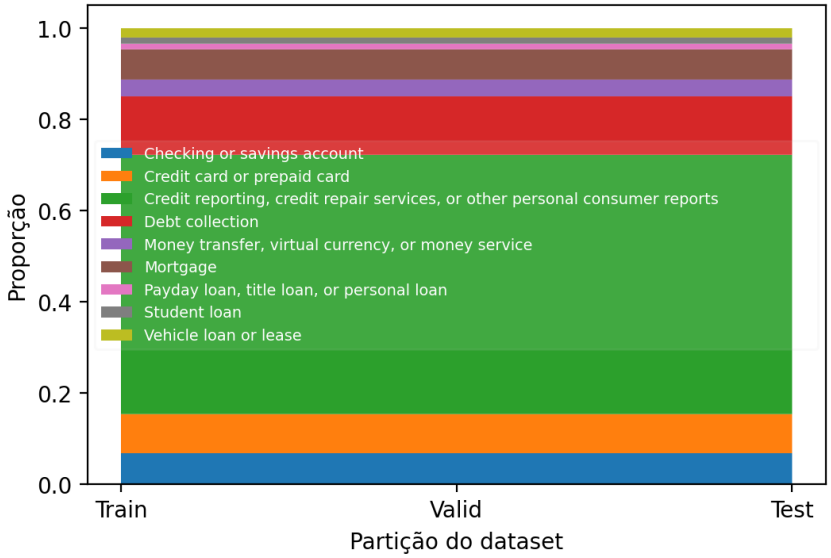


Figura 3.2: Distribuição relativa dos tipos de produto em cada partição do *dataset*.

e reporte de resultados, agora aplicados ao nível das partições, e à construção de respetivos ponderadores para reposição da representatividade original das classes, para fazer face aos recursos tecnológicos limitados[6]. Como premissa fundamental, procurou-se limitar esta intervenção ao estritamente necessário, mitigando perdas de robustez inerentes à redução de dados e potencial perda de diversidade de reclamações. Sabendo-se que o principal foco da ineficiência residia no treino, procedeu-se a uma amostragem aleatória estratificada uniforme sobre o conjunto de treino, justificando-se este tipo de estratificação para permitir aos algoritmos igual contacto com as características particulares de cada classe/tipo de produto alvo de reclamação. O processo de amostragem aplicado foi o seguinte:

i. Apurar a dimensão da classe de menor frequência;

ii. Devido ao método implementado[7] para proceder à amostragem, subtrai-se seguidamente 1 ao valor apurado em i e fixa-se o valor resultante como dimensão amostral para recolha de amostra em todas as classes, sabendo-se que as reclamações da classe de menor frequência terão probabilidade de inclusão na amostra (quase) igual a 1;

iii. Proceder a uma amostragem aleatória simples (sem reposição) em cada classe com a dimensão amostral fixada no ponto anterior;

[6] O equipamento informático utilizado apresenta as seguintes características: HP Intel Core i5, CPU 1.00Ghz, RAM 12.0 GB, GPU 2.0 GB.

[7] Para proceder à amostragem aleatória estratificada uniforme, onde ao estrato corresponde a classe/tipo de produto, adotou-se o método *sklearn.model_selection.train_test_split* do software de ML *scikit-learn* [28], o qual exige que os "copos" *train* e *test* apresentem pelo menos 1 elemento, gerando erro em caso contrário. Uma sua implementação alternativa seria, naturalmente, aplicá-lo às demais classes que não a classe de menor frequência, amostras às quais se juntaria toda a população desta última classe.

iv. Compilar as amostras de nove classes, consubstanciando-se na amostra final do conjunto (neste caso, o conjunto de treino).

Após a formação da amostra do conjunto de treino que o representará na fase de treino, há que proceder à construção do vetor de ponderação dos elementos da amostra que permitirá repor a representatividade original de cada classe nesse conjunto de treino. Para tal, segue-se o seguinte procedimento:

i. Calcular o ponderador de cada classe através da divisão da sua frequência absoluta do conjunto de treino (coluna "# original" da tabela 3.4) pela sua frequência na amostra deste conjunto (coluna "# amost. n. pond." da mesma tabela);

ii. Imputar o ponderador de cada classe, obtido no ponto anterior, a cada reclamação da amostra do conjunto de treino pertencente àquela classe, obtendo-se assim o vetor de ponderadores dos elementos da amostra, capaz de representar, neste caso, o conjunto de treino original.

A tabela 3.4 exibe a amostra que será ***processada*** no treino (amostra não ponderada, na coluna "# amost. n. pond."), mas que será vista nessa fase pelo algoritmo de classificação na forma ***ponderada*** (amostra ponderada, na coluna "# amost. pond.").

Tabela 3.4: Reclamações por Produto em cada versão conj. treino.

Produto	# original	# amost. n. pond.	# amostra pond.
Credit reporting, credit repair services, or other personal consumer reports	70093	1499	70093.0
Debt collection	15806	1499	15806.0
Credit card or prepaid card	10571	1499	10571.0
Mortgage	8462	1499	8462.0
Checking or savings account	8185	1499	8185.0
Money transfer, virtual currency, or money service	4535	1499	4535.0
Student loan	2513	1499	2513.0
Vehicle loan or lease	1675	1499	1675.0
Payday loan, title loan, or personal loan	1500	1499	1500.0

Importa recordar que esta metodologia foi aplicada tanto na aprendizagem dos MC de ML como na dos Modelos de DL. Se bem que, no primeiro caso, a sua aplicação tenha ficado pelo conjunto de treino, permitindo que o conjunto de validação e de teste produzissem métricas de desempenho sobre toda a sua extensão, já no caso da abordagem DL, a aplicação deste processo de amostragem foi estendida aos demais conjuntos, sendo tal justificado pela maior exigência dos recursos tecnológicos destes algoritmos, quer na validação do modelo em treino, quer na aplicação do modelo ao conjunto de teste para

avaliação de desempenho. Dá-se aqui nota que a aplicação de ponderadores para efeitos de treino, validação e teste, foi realizada com recurso aos parâmetros dos respetivos métodos dos algoritmos utilizados, para tal nestes já prevista. Mais se refere que a qualidade da amostragem foi igualmente assegurada, o que será mais tarde visível na consistência dos resultados apresentados.

3.4 Etapas das Soluções Orientadas às Abordagens

Como já anteriormente referido, pretende-se obter valores altos de *accuracy*, *precision* e *recall* (recordar métricas de avaliação em 2.6) na identificação da classe das reclamações feitas pelos Clientes. Para tal, é importante comparar o desempenho de modelos desenvolvidos sob abordagem clássica vs. modelos baseados em abordagem *deep learning*.
Tendo em conta que não é possível testar todas as suas respetivas possíveis variantes (quer seja no pré-processamento e *feature engineering* a adotar, quer no tipo de algoritmo de classificação a utilizar), procurou-se construir, em cada abordagem, um processo analítico e modelo que a representassem nas suas características específicas e nas opções por que passa na procura dos melhores resultados.

3.4.1 Abordagem Clássica

Considerando a família das abordagens clássicas, há quatro fases a considerar: a escolha das *features*; a redução do número de *features*; a (eventual) ponderação das *features*; a escolha do classificador.
Descreve-se seguidamente a atividade desenvolvida em cada uma.

3.4.1.1 A Extração e Escolha das *Features*

A qualidade da escolha das *features* é crucial pois reflete-se nos resultados a obter posteriormente na fase da classificação dos novos documentos de reclamação. Assim, o conjunto de *features* deverá apresentar, para cada classe conhecida, valores o mais distintos possível, de forma a que, cada grupo distinto de valores, identifique apenas uma das classes. Nesta fase, foram realizadas as seguintes tarefas:

a) **Obtenção de um conjunto de *features* candidatas**: consideraram-se como *features* candidatas, todos os possíveis *n-grams* no *corpus* até *3-grams* contíguos, isto é, todas as palavras distintas (*1-grams*) e todos os grupos distintos de *2-grams* e *3-grams* que lá existam (ver técnica em 2.3.2.1).
   Este conjunto de *features* candidatas foi, contudo, limitado em número aos 10 000 *n-grams* de maior frequência no *corpus* do estudo (o nosso conjunto de treino original), excluindo ainda *a priori* aqueles que se encontrem em mais de 95% das reclamações, bem como os demasiado raros, i.e., aqueles que não estejam presentes em pelo menos 3 reclamações.

O resultado desta tarefa é a atribuição de um vetor BoW de dimensão 10 000 *n-grams* a toda a reclamação, onde estão inscritas as frequências absolutas respeitantes a esses *n-grams* que esta contém;

b) **Construção da opção "BoW em frequências relativas"**: considerando que se deveria avaliar a opção de relativizar a presença de cada *n-gram* numa reclamação em função da dimensão da mesma, tornando assim comparáveis reclamações mais pormenorizadas com as mais resumidas, transformou-se o vetor produzido na tarefa anterior pelo seu somatório, onde cada *n-gram* apresenta agora a sua proporção no total de *n-grams* da reclamação;

c) **Medir a importância de cada *n-gram* para a diferenciação das classes**: esta tarefa constitui o instrumento fundamental na construção do espaço das *features*. Permitirá não só manter, como candidatas a *feature*, os *n-grams* que sejam estatisticamente significativos na perspetiva das classes, como ser responsável pela seriação das *features* com vista à redução do seu número (ver fase seguinte 3.4.1.2), como ainda permitir a atribuição de um peso a cada *feature*, "atraindo" dessa forma a atenção do classificador.
Esta importância é resultado da aplicação da análise de variância (ANOVA), com cada candidata a *feature*, na sua versão "frequência absoluta" ou "frequência relativa", a ser alvo de análise univariada com a variável das nossas classes para comparação da variância dessa frequência entre classes com a variância dessa frequência dentro das classes. A importância é apurada pela estatística *F-test* ou, como é comummente renomeada em indicador, o *F-ratio*, em baixo formulado na equação 3.1:

$$F\text{-}ratio = \frac{\text{variância da } \textit{feature } x \text{ entre classes}}{\text{variância da } \textit{feature } x \text{ dentro das classes}} = \frac{\sum_{i=1}^{K} n_i \left(\bar{x}_i - \bar{x}\right)^2 / (K-1)}{\sum_{i=1}^{K} \sum_{j=1}^{n_i} \left(x_{ij} - \bar{x}_i\right)^2 / (N-K)} \quad (3.1)$$

onde:

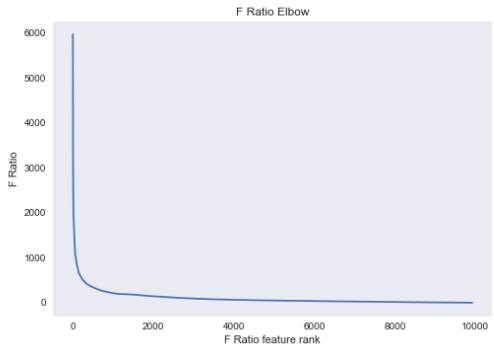
K = número de classes;
N = total de reclamações;
n_i = número de reclamações pertencentes à classe i;
x_{ij} = frequência (absoluta ou relativa, conforme opção BoW) do *n-gram* candidato a *feature* x na reclamação j pertencente à classe i;
$\bar{x}_i$ = frequência (absoluta ou relativa, conforme opção BoW) média do *n-gram* candidato a *feature* x em reclamações pertencentes à classe i;
$\bar{x}$ = frequência (absoluta ou relativa, conforme opção BoW) global média do *n-gram* candidato a *feature* x.

Esta análise gera valores distintos para as diferentes candidatas, com *n-grams* como *mortgage*, *card*, *debt*, *loan* ou *checking account* a serem decisivamente mais discriminantes do que *n-grams* como *How can*, *intend to* ou *am asking*.

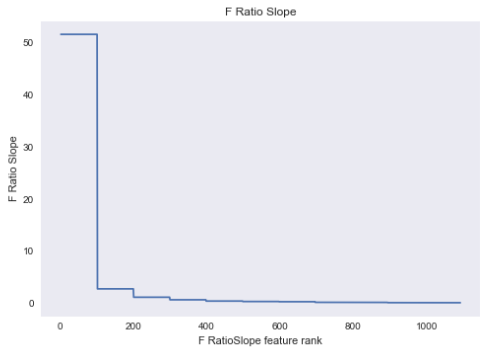
Nesta fase, o papel atribuído ao *F-ratio* foi de assegurar a significância (a 95%) das *features* face às classes, pelo que o resultado da seleção teve pouco impacto no número de *features* candidatas, mantendo 9 923, no caso da versão em frequência absoluta, e 9 876, no caso da relativa.

3.4.1.2 A Redução do Número de *Features*

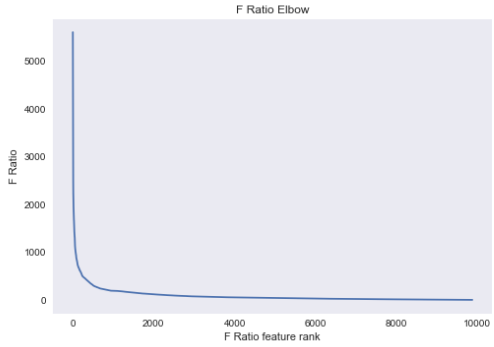
Para assegurar que o classificador irá procurar a diferenciação das classes com base em *features* que mais as discriminem, faz-se uso da importância de cada *n-gram*, atrás descrita, para considerar a importância limiar (*threshold*) abaixo da qual *features* inicialmente selecionadas são descartadas. Para apuramento desse limiar, considerou-se o uso de um método - inspirado no do cotovelo largamente utilizado em *clustering* - para procura encontrar o ponto a partir do qual o acréscimo marginal, em unidades de *F-ratio*, de manter uma *feature* é inferior a 1 (ver ponto de partida da análise nos gráficos 3.3). Caso não o seja, quer dizer que a *feature* ainda acrescenta um valor de *F-ratio* face a próxima *feature* superior a 1, pelo que se deverá manter por ainda se destacar.



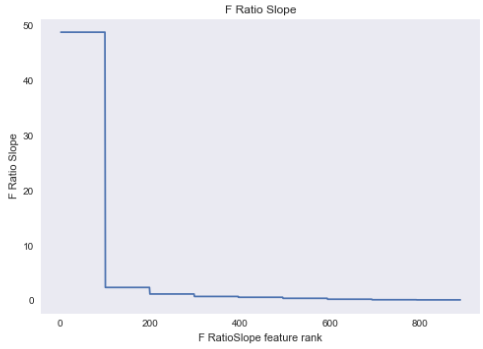
(a) Gráfico de cotovelo do *F-ratio* sobre FREQ. ABS.



(b) Gráfico de declive do *F-ratio* sobre FREQ. ABS.



(c) Gráfico de cotovelo do *F-ratio* sobre FREQ. REL.



(d) Gráfico de declive do *F-ratio* sobre FREQ. REL.

Figura 3.3: Gráficos cotovelo e de declive do *F-ratio* em cada unidade de medida.

Aplicando igual metodologia para cada opção de unidade de medida da frequência,

procede-se da seguinte forma:

i) Seriação das *features* em ordem decrescente em função do *F-ratio*;

ii) Cálculo da diferença entre de o *F-ratio* da *feature* que a antecede com o seu, dando conta do declive sofrido nesse ponto;

iii) Como a regularidade da série dos declives não é, em todo o seu domínio, monotónica decrescente, dividiu-se a distribuição do *F-ratio* em percentis (100 segmentos de igual número de *features*), calculando-se posteriormente a média do declive em cada percentil de *F-ratio* (*F-ratio Slope* a consultar nos gráficos 3.3(b) e 3.3(d));

iv) Procede-se à manutenção das *features* que, em média, detenham um declive superior a 1. No entanto, este (*threshold*) apenas retém cerca de 300 *features*, representando, por um lado, uma captura muito baixa de *F-ratio*, 26% em freq. absoluta e 29% em freq. relativa, mas também uma redução muito forte do número de *features*, que é o mesmo que dizer, de vocabulário de diferenciação das classes (apenas 3% do vocabulário de partida). Assim, optou-se por considerar um (*threshold*) de 0,1 (ou um declive unitário de 10%), permitindo capturar quase 50% da importância (*F-ratio*) e manter um vocabulário em 10% do de partida, conforme exibido na tabela 3.5.

Tabela 3.5: Proporção de *features* e de importância capturada.

Unid. medida base	Proporção # *features* (%)	Proporção de importância (%)
Frequência absoluta	11	48
Frequência relativa	9	48

3.4.1.3 Opção "Ponderação das *Features*" para Treino

Nesta fase, temos já duas opções para construção dos vocabulários, uma com frequências absolutas e outra de relativas, definindo assim dois possíveis espaços de *features*.
Pretendendo-se também poder optar por transformar esses espaços pela importância de cada um dos *n-grams* presentes nesses vocabulários, medida em função do poder discriminante que têm as classes sobre justamente a frequência de cada um, constroem-se duas alternativas para o valor a atribuir a um *n-gram* presente numa reclamação: a importância dada pela sua própria frequência (na unidade absoluta ou relativa, conforme se opte) e a importância medida através da multiplicação da frequência do *n-gram* pelo seu *F-ratio*, obtido na fase de escolha das *features* descrita em 3.4.1.1 e utilizado na fase de redução do espaço das *features* descrita em 3.4.1.2.
Para efeitos de industrialização em *pipelines*, estas opções de "construção do BoW em frequências relativas" e de "atribuição de importância ao BoW", previamente implementadas em funções, são alvo de transformação em métodos para que sejam executados, quando aplicável, durante o processo de treino sobre os conjuntos de treino e validação, e

em fase de avaliação sobre o conjunto de teste. Estes têm associados *datasets* de parâmetros obtidos nas fases anteriores já mencionadas, relativos às opções de "construção de vocabulário" e a opção de "atribuição de importância".

3.4.1.4 A Escolha do Classificador

Com os métodos e *datasets* de parâmetros correspondentes às 4 alternativas desenhadas para a construção do espaço das *features*, estamos capazes de passar para a fase de treino do classificador. Prévio ao treino, constrói-se um *pipeline* dinâmico que aplique métodos e procedimentos correspondentes às 4 alternativas de *feature engineering*, cruzando **unidade base do BoW** e **ponderador de importância da *feature***. Este *pipeline* irá, por sua vez e implementado em processo *grid search*, otimizar os parâmetros do algoritmo de classificação representante da abordagem clássica adotado: o **classificador *Support Vector Machine*** (ver ponto 2.5.1.4), na sua versão não linear (ver equação 3.2):

$$\begin{aligned} \underset{\vec{\alpha}}{\operatorname{argmax}} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m K\left(\vec{x}_n, \vec{x}_m\right) \\ \text{s.a.} \sum_{n=1}^{N} \alpha_n y_n = 0 \quad 0 \leq \alpha_n \leq C \end{aligned} \tag{3.2}$$

onde $K\left(\vec{x}_n, \vec{x}_m\right)$ é justamente a função de *kernel* que mapeia as nossas *features* e as expande para um espaço não linear de maior dimensionalidade, capaz assim de maiores desempenhos na separação das nossas classes. Para esta, considerámos a aplicação da função *Radial Basis Function* (RBF), única adotada, função em que o valor devolvido depende da distância ao centro do espaço de *input*, as nossas *features*. A forma mais comum é a *RBF Gaussiana*, mas no nosso caso utilizou-se a implementação do *software* de ML adotado, a saber:

$$K\left(\vec{x}_n, \vec{x}_m\right) = e^{-\gamma \left\| \overrightarrow{x_n} - \overrightarrow{x_m} \right\|^2} \tag{3.3}$$

O processo irá procurar otimizar:

i) O parâmetro γ (ver equação 3.3), parcela do expoente da exponencial, que define assim o raio de influência da escolha dos três pontos de *support* numa iteração, com comportamento do seu inverso (proveniente do sinal negativo deste expoente), i.e., para valores baixos a influência do vetor de *support* para com os demais exemplos faz-se a uma distância maior, e quando é mais alto, mais a influência se faz apenas sobre os que estão mais perto (por vezes tão perto que o vetor de *support* tenderá a influenciar-se só a si), e

ii) O parâmetro de regularização C (ver equação 3.2), que penaliza os erros de classificação, i.e., um elevado valor de C irá resultar numa elevada penalização pelo erro, pelo que irá levar a que a margem se reduza (no treino, elevando o risco de *overfitting*, monitorizado pela validação).

Não sendo de todo possível "varrer" no *grid search* um intervalo lato nos 2 parâmetros, procuraram-se exemplos que pudessem dar pistas de intervalos "vencedores". No caso do parâmetro de regulação, fixou-se o intervalo em $[0.01, 0.1, 1]$.
Já quanto ao γ, a utilização de exemplos não se revelou bem sucedida, até que (melhor) se interpretou este parâmetro na fórmula 3.3 e a sua comparação com a versão Gaussiana, tendo-se concluído pela utilização de um intervalo de 5 múltiplos de um γ de partida, cujo valor seria o inverso da variância global do conjunto de *features* multiplicado pelo seu número. Ora, a variância em denominador (a corrigir distâncias ao centro do espaço das *features*), ainda que "dimensionado" pelo número de *features*, assemelha-se justamente à versão Gaussiana (e seus princípios), o que fez todo o sentido e se revelou bem sucedido. De realçar somente que o intervalo testado para este parâmetro foi naturalmente distinto para cada uma das 4 alternativas de *feature engineering* por depender da variância do espaço de *features* que corresponde, com esta a depender igualmente da unidade de medida (absoluta ou relativa) ou da ampliação de *features* em níveis proporcionais às suas importâncias. No próximo capítulo apresentamos os resultados para o classificador selecionado (ver 4.2).

3.4.2 Abordagem de *Deep Learning*

Pretendendo colocar a arquitetura *state of the art* desta abordagem, o Transformer, ao serviço da classificação de reclamações de Clientes do setor financeiro, e usufruir do conhecimento da semântica, de carácter geral, capturada pelos modelos seus representantes *Encoders*, pré-treinados justamente para deterem a capacidade de *transfer learning* em várias tarefas de NLP, considerou-se a adoção de uma variante do modelo Transformer *Encoder BERT*. A variante do BERT deverá ser capaz de nos oferecer:

i) Um conhecimento abrangente das relações semânticas existentes na linguagem natural (representação da linguagem), adquirido no pré-treino em tarefas NLP de classificação de texto e passível de ser adaptado, com elevada eficácia, à classificação das reclamações do Cliente de produtos e serviços financeiros;

ii) Uma arquitetura e capacidade de computação compagináveis com recursos tecnológicos limitados, característica do ambiente tecnológico utilizado neste estudo.

A aplicação deste conhecimento ao serviço da classificação que pretendemos realizar, e que será representante da abordagem DL em classificação de texto, compreende quatro fases: a escolha do modelo pré-treinado da família de variantes BERT que o irá representar; a preparação dos dados para o modelo e à tarefa NLP que estamos a realizar; a configuração da camada específica de classificação e do processo de otimização dos hiperparâmetros do modelo; a seleção do classificador.

3.4.2.1 A Escolha do Modelo Pré-treinado BERT

A escolha do modelo pré-treinado BERT de entre um conjunto de variantes deste tipo de Transformers *Encoders* é fundamental para a qualidade da adaptação pretendida, uma vez que esta depende da natureza da tarefa de NLP, do domínio de aplicação, da capacidade de computação disponível para a adaptação e prevista para aplicação, entre outros fatores. Esta escolha foi bastante facilitada pelo acesso a uma vasta biblioteca destes modelos, com funcionalidades de pesquisa da sua documentação, proporcionada pela *Hugging Face*, comunidade especializada em *Artificial Intelligence* (AI).
Detalha-se de seguida alguns critérios que nortearam essa seleção:

a) **Experiência na tarefa**: é necessário que tenham sido pré-treinados para classificação de texto (*Sentence Classification*), ou para tarefa que lhe seja análoga (e.g., tarefas de NLU[8] avaliadas na coletânea *General Language Understanding Evaluation* (GLUE) [37], servindo, pela sua diversidade, como *benchmark* de avaliação de modelos de NLP);

b) **Compatibilidade com o *transfer learning***: os modelos pré-treinados devem ser compatíveis com a adição de camada específica, a treinar para a realização da tarefa da classificação de reclamações de produtos financeiros. Esta compatibilidade deriva de vários métodos, entre eles a tokenização aplicada pelo modelo às reclamações;

c) **Capacidade de adaptação ao domínio**: a maioria destes modelos são treinados sobre um *corpora* de domínio genérico. Sabendo-se que o setor financeiro tem terminologia própria, houve a preocupação em se assegurar a possibilidade de proceder a um treino do modelo mais profundo, impondo-lhe um *fine-tuning* (ver 2.5.2). Assim, as disponibilidade e flexibilidade do modelo a essa adaptação são críticas, uma vez ser desejável um seu (outro) treino orientado.

Partindo de uma lista na qual constavam, entre outros, o BERT (naturalmente), o ALBERT[8] [20], o RoBERTa[12] [21] ou mesmo o DeBERTa[9] [15], a opção recaiu sobre o DistilBERT [32]. Este foi proposto em 2019 para oferecer uma versão do BERT mais acessível em termos de exigência computacional, permitindo, por um lado, difundir o usufruto do conhecimento capturado por estas arquiteturas (*transfer learning*) e, por outro, a sua especialização a problemas concretos (*fine-tuning*), o que é o nosso caso. O DistilBERT consegue esta "democratização" por deter menos 40% dos parâmetros da versão do BERT da qual partiu (o BERT_{BASE}, com cerca de 110M de parâmetros), sendo por isso 60% mais rápido na execução, com perda de desempenho acomodável face ao seu "antecessor" (preserva 95%

[8] Versão mais leve do BERT para redução do tempo de treino, especializado na tarefa *Sentence Order Prediction* (SOP), também de classificação.

[9] Acrónimo de *Decoding-enhanced BERT with Disentangled Attention*, faz evoluir os modelos BERT e RoBERTa via nova arquitetura do *attention mechanism* (ver descrição detalhada no último exemplo de modelos DL apresentado na secção 2.7.2).

do desempenho[10]).
A metodologia que implementa esta redução de parâmetros passa justamente por aplicar uma técnica de compressão ao LM BERT, enquanto modelo de maior dimensão designado de "modelo do professor", num outro LM de menor dimensão, designado de "modelo do aluno", através da *destilação do conhecimento*[11] que aquele detém. A menor dimensão do "aluno" é resultante da alteração da arquitetura do modelo que pretende imitar (o BERT_{BASE}), fundamentalmente decorrente da redução do número de *layers* de 12 para 6 (cada *layer* mantém, contudo, os 12 *hidden layers* originais). O "aluno" é assim treinado para conseguir, com menos parâmetros, uma capacidade de generalização semelhante à do "professor" por via do mapeamento da distribuição do *output* deste (treina com as probabilidades estimadas pelo BERT, pré-treinado em tarefa *masked language modeling* (MLM), para toda a *token* do *corpus*, e não somente com essa verdadeira *token*) e do conhecimento já incorporada nesses parâmetros que herda do pré-treino do "professor".
A função *loss* a minimizar durante o treino é então composta por três componentes de *cross-entropy*, com a terceira a ser resultado das duas anteriores:

- A que compara as verdadeiras *tokens* "mascaradas" (denominadas pelos autores de *hard targets*) com as previsões realizadas pelo "aluno", não sendo mais do que a função *loss* habitual em classificação supervisionada, neste caso realizada por um *Masked Language Model* (MLM):

$$L_{\text{MLM}\,i} = -\sum_{i=1}^{n} t_i \log\left(st_i\right), \tag{3.4}$$

  onde:

  n = número de *tokens* (do vocabulário);
  t_i = *masked token*, i.e., a token que foi "mascarada";
  st_i = probabilidade estimada pelo "aluno" de a *token* i ser a *masked token*.

- A que compara a distribuição das *tokens* previamente previstas pelo "professor" BERT (denominadas pelos autores de *soft targets*) com a distribuição que vai sendo proposta (e otimizada), durante o treino, pelo "aluno" DistilBERT:

$$L_{\text{ce}\,i} = -\sum_{i=1}^{n} tr_i \log\left(st_i\right), \tag{3.5}$$

  onde:

[10] Proporção do GLUE *score* [37] do BERT_{BASE} que é coberto pelo GLUE *score* do DistilBERT. O *benchmark* GLUE tem 9 *datasets* para avaliação de outras tantas tarefas de NLP, via métricas de desempenho *accuracy* em 7 dos *datasets*, coeficientes de correlação de Pearson [38] e de Spearman [40], e ainda coeficiente de correlação de Matthews [39] nos demais 2 *datasets*, todas em escala de 100. O GLUE *score* de um modelo de NLP é obtido pela *macro-average* destas 9 métricas.
[11] Técnica introduzida por Bucila et. al. [6] e mais tarde generalizada por Hinton et al. [16].

n = número de *tokens* (do vocabulário);
tr_i = probabilidade estimada no pré-treino do "professor" de a *token* i ser a *masked token*;
st_i = probabilidade estimada pelo "aluno" de a *token* i ser a *masked token*.

- Saliente-se que para melhor mapear a distribuição das probabilidades estimadas das *tokens* do vocabulário face a cada *masked token*, recorre-se a parâmetro introduzido na *softmax* por Hinton et al.[16] no cálculo das probabilidades (apenas para efeito de treino), denominado *softmax-temperature* (ver equação 3.6):

$$p_i = \frac{\exp(z_i/T)}{\sum_{i=1}^{n} \exp(z_i/T)} \tag{3.6}$$

onde:

n = número de *tokens* (do vocabulário);
z_i = *embedding* médio após cada *step* de treino, estimado, tanto por "professor" como "aluno", de a *token* i ser a *masked token*;
T = *softmax-temperature*, fixado no intervalo $]\,0,\ +\infty\,[$ e aplicado em igual valor a "professor" e "aluno" durante o treino para "amplificar" o sinal de cada exemplo, conduzindo a distribuição das probabilidades das *tokens* a um vetor *target one-hot* a pender para a *token* de maior probabilidade quando $T \to 0$, ou a uma distribuição uniforme quando $T \to +\infty$.

- A que relaciona os vetores de cada umas das componentes anteriores, já com todas as *tokens*, através da aplicação do cosseno, pretendendo assim alinhar as direções do vetor de "aluno" com o do "professor" (torná-los semelhantes, uma vez que se trata justamente de "mimetismo"):

$$L_{\cos} = 1 - \cos(L_{\text{MLM}},\ L_{\text{ce}}) \tag{3.7}$$

O objetivo final do treino é então a minimização da combinação linear destas três componentes de distâncias.
Decidido o modelo representativo da abordagem de DL baseada na família BERT, parte-se para a fase de preparação do texto das reclamações de acordo com as expetativas do DistilBERT.

3.4.2.2 A Preparação dos Dados para Modelo e Tarefa de Classificação

O tratamento dos textos das reclamações para treino do DistilBERT segue os seguintes passos:

i) **Aplicação da tokenização específica do DistilBERT, baseada na *WordPiece*, às reclamações** (ver 2.3.2.3): tal como o BERT, a dimensão máxima da sequência de *tokens*

admitida pelo DistilBERT é de 512. Tendo em conta a tarefa NLP de classificação que vamos realizar, à sequência de *tokens*, resultante da aplicação da *WordPiece* ao texto de cada reclamação, é adicionada a *token* especial `[CLS]`[12] no início e a *token* `[SEP]` no seu final, pelo que apenas as primeiras 510 *tokens* são utilizadas. Por esse facto, parametriza-se o *tokenizer* para converter o conteúdo em *lowercase* para evitar que *tokens* no limiar da exclusão venham a ser truncadas devido à permanência de 2 ou mais *tokens* de semântica igual ou próxima (poder-se-á sustentar que uma letra em *uppercase* tem semântica própria, nomeadamente dando conta do início de uma frase, questão que é aqui mitigada, por estes algoritmos "valorizarem" o posicionamento da *token* na sequência de *input* e por a pontuação fazer parte da sequência);

ii) **Preparação das sequências de dados de *input* do DistilBERT com base na tokenização**: após tokenização realizada no passo anterior para que as sequências sejam processadas pelo *encoder*, com vista a representá-las em contexto de acordo com o seu conhecimento prévio e com o que for adquirindo com o treino, esta fase irá integrar **ids das *tokens*** (*tokens* na forma numérica para que sejam matematicamente processadas) e ***attention masks*** (informação das *tokens* às quais se deverá dar atenção, uma vez a dimensão de uma reclamação não ter conseguido preencher a dimensão máxima admitida; o que falta até ao máximo parametrizado é preenchido com a *padding token* `[PAD]`).

É com base neste tratamento que o DistilBERT irá, à semelhança dos demais Transformers, logo num primeiro passo, codificar toda a *token* em *input embedding* e *positional embedding*, os quais, em conjunto, formam o *contextual embedding* da *token*. No caso do DistilBERT, a dimensão destes *embeddings* é de 768, i.e., cada um absorve o contexto que o envolve, por atenção aos *embeddings* das *tokens* suas vizinhas (o *attention weight* observado em todas as reclamações), em 768 dimensões. Cada *embedding* vai sendo transformado ao longo dos *layers* do Transformer (no DistilBERT são 6 *layers*, cada com 12 *hidden layers*) através de uma combinação ponderada (por pesos ou parâmetros a otimizar ou pré-treinados) da sua versão de *embedding* obtida nos *layers* precedentes, concluindo-se o processo de aprendizagem, ou a aplicação do conhecimento adquirido, de cada *contextual embedding* no *output embedding* do DistilBERT.

3.4.2.3 Configuração da Camada de Classificação e do Processo de Otimização

Esta fase corresponde à configuração do *transfer learning* (e *fine-tuning*) do modelo, bem como de desenho do processo que irá promover a otimização dos seus hiperparâmetros para efeitos de classificação das reclamações de produtos financeiros. Dá-se aqui nota que o *fine-tuning* não chegou a realizar-se por limitações da infraestrutura tecnológica[6]

[12] *Token* especial utilizada no pré-treino do BERT na tarefa de NSP e em processos de *fine-tuning* para alargamento deste LM a outras tarefas de *Natural Language Inference* (NLI), tais como a nossa *Text Classification*, já que o *output embedding* deste *token* é uma representação agregada da sequência das demais *tokens* quando deriva desses processos.

utilizada. Estes trabalhos de infraestruturação do treino do DistilBERT foram realizados no **TensorFlow** via **Keras API**. Compreendem as seguintes atribuições:

i) **Definição da arquitetura da camada de classificação**, a adicionar ao modelo pré-treinado, responsável por transformar a representação de cada reclamação, agora já contextualizada, na classificação quanto ao produto que motivou a sua formalização. Na prática, desenha-se a arquitetura do processo de realização de um treino parametrizado através da composição do *pipeline* em camadas funcionais, iniciando pelos dados, passando para a instanciação do DistilBERT, capacitando-o de regulação (*dropout* nos seus diferentes *layers*) e indicando-lhe tratar-se de um treino menos ou mais profundo (*transfer learning* ou *fine-tuning*), e depois pela adição da estrutura de classificação, composta por:

- Somente aplicável em processo de *transfer learning*, um primeiro *layer* de densificação dos *output embeddings* do DistilBERT, correspondentes às 510 *tokens* da reclamação mais 2 relativas às especiais, num único *embedding*, o *embedding* médio da reclamação, que a representa (cada reclamação passa assim a ser representada por um *embedding* de dimensão 768). Caso tivesse sido possível a realização do (desejável) *fine-tuning*, ter-se-ia recorrido unicamente ao *output embedding* da *token* inicial `[CLS]`, a qual foi justamente *pré-treinada* pela "família" BERT para ser uma representação agregada da sequência das demais *tokens* quando processada em *fine-tuning* para tarefas de classificação. No entanto, só passaria a deter essa característica após eventual novo treino "dirigido" às nossas reclamações (ver J. Devlin et al [11], *footnote* 6);
- Um conjunto de dois *layers* de classificação, que pretende proceder à redução da dimensionalidade para as 9 classes em arquitetura *Deep Learning* protagonizada por um classificador *Feed-Forward Neural Network(s)* (ver 2.5.2.1) com *hidden layer* de neurónios, em número a determinar (hiperparâmetro a otimizar), ativados por função não linear, e com *output layer* com 9 neurónios correspondentes às 9 classes, ativados pela função *softmax*. Este conjunto permite constituir, no *layer* inicial, uma representação do conhecimento capturado nos *output embeddings* de menor dimensão, capaz assim de separar reclamações não separáveis linearmente e mitigar o *curse of dimensionality*, permitindo ao *output layer* melhor inferir a probabilidade de cada reclamação ser de cada classe e, por fim, ser estimada a sua classificação pela máxima probabilidade estimada de entre as 9.

Definida a camada de classificação, é então possível apresentar a arquitetura da solução de *Deep Learning*, indicada na figura 3.4.

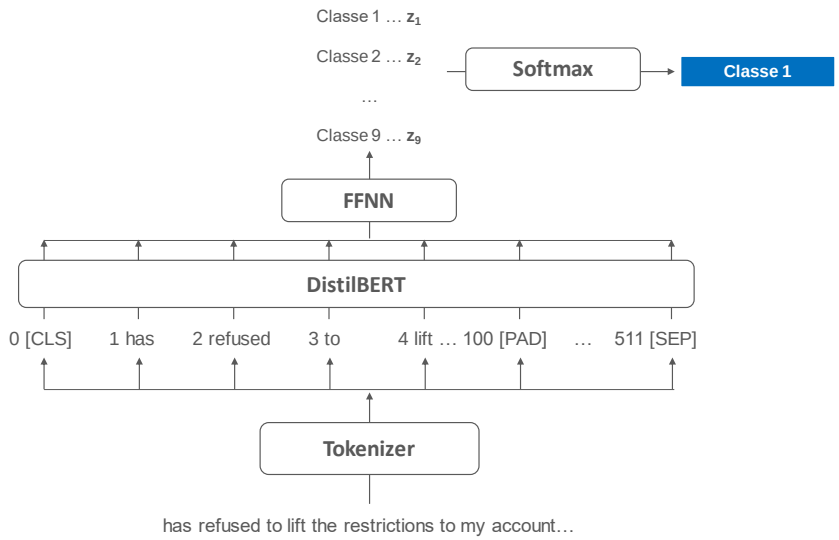


Figura 3.4: Arquitetura da solução de *Deep Learning*.

ii) **Configuração dos hiperparâmetros que condicionam o treino**, indo desde o próprio otimizador, a *learning rate*, o número de épocas e o *batch size*, até à função de ativação do *hidden layer* do classificador baseado na FFNN, à função de ativação do *output layer* e à função *loss*. Começando por estas últimas:

- Relativamente à **função de ativação do *hidden layer* da FFNN**, optou-se pela *Rectified Linear Unit* (ReLU)[13], função recomendada por evitar o *vanishing gradient problem*[14]. Já quanto à **função de ativação do *output layer*** e à **função *loss***, estas encontram-se estruturalmente ligadas e dependentes da tipologia de tarefa que se está a realizar, o que no nosso caso fixa a primeira na *softmax* (de resto, já indicada no ponto anterior), e a segunda na *categorical cross entropy*, uma vez estar a ser realizada uma classificação multi-classe;
- Passando agora aos hiperparâmetros diretamente ligados ao desempenho computacional do processo de otimização, fixou-se o ***batch size***[15] em 16 reclamações, dadas

[13]Função que reflete o *input* quando este apresente valores positivos e o "neutraliza", transformando-o para zero, quando já o for ou apresente valores negativos. Esta transformação é suficiente para tornar a função não linear, condição necessária para a eficiência da rede neuronal, conseguindo-se ao mesmo tempo manter as derivadas constantes e diferentes de zero sempre e quando os respetivos neurónios se ativam.

[14]Problema crónico das arquiteturas *Deep Learning* na propagação do erro de classificação (a *loss*) pelos parâmetros (os ponderadores ou *weights*) seus responsáveis quando utilizam funções de ativação clássicas como a sigmoide e a tangente hiperbólica, por as derivadas da função *loss* em ordem aos parâmetros, apesar dos neurónios se ativarem, convergirem para zero para elevados valores absolutos dos *inputs*. Este facto conduz a que os gradientes não informem o otimizador, em cada passo, da necessidade de reduzir a função *loss*, o que é ultrapassado pela função ReLU.

[15]Número de exemplos que são apresentados à rede em cada *step* (o número de *steps* depende justamente deste hiperparâmetro e do número de exemplos do conjunto de treino) para cálculo do erro e para a propagação desse erro em unidades de gradiente da responsabilidade de cada um dos seus parâmetros -

as limitações da infraestrutura tecnológica[6] e após terem sido testados esses limites (tendo em conta que várias classes no conjunto de treino original têm proporção inferior a $\frac{1}{16}$, poderá argumentar-se que a representatividade das classes em cada *batch* não é assegurada; no entanto, recorda-se que o conjunto de treino utilizado segue uma distribuição uniforme de classes, pelo que a probabilidade de inclusão de um exemplo de cada classe no *batch* é igual para todas as classes).
Já em relação ao **número de épocas**, hiperparâmetro que define quantas vezes a totalidade dos exemplos do conjunto de treino é apresentada ao otimizador, este foi fixado em 5, conduzindo a um tempo de treino de cada modelo de 5 horas, uma vez que cada época apresentou um tempo de execução a rondar uma hora;

- Falando agora dos hiperparâmetros da otimização, optou-se por utilizar o **otimizador** e a ***learning rate***[16] em conjunto, testando dois otimizadores, o ***AdaGrad*** e o ***Adam***, que os interligam para que a *learning rate* se adapte a cada parâmetro ao longo do processo de otimização, i.e., permita, em cada *step*, uma aprendizagem a diferentes velocidades em função do histórico das atualizações que o parâmetro vai sofrendo (ou potenciais atualizações) durante o processo de convergência. É assim comum a estes otimizadores o conceito de ***adaptive learning rates***.
Partem ambos do *Stochastic Gradient Descent* (SGD), cujo algoritmo começa por estimar, em cada *step* e para cada parâmetro a atualizar, o gradiente da função *loss* em ordem a esse parâmetro, tendo por base um conjunto aleatório (daí *Stochastic*) de exemplos contido no *batch* (ver equação 3.8), ...

$$\hat{g}_t = \nabla_\theta \left(\frac{1}{m} \sum_{i=1}^{m} L\left(f\left(x^{(i)}, \theta \right), y^{(i)} \right) \right) \tag{3.8}$$

onde:

θ	= parâmetro afeto ao *embedding* x a atualizar;
$x^{(i)}$	= *embedding* x observado no exemplo i do *batch* do *step* t;
$y^{(i)}$	= classe y observada no exemplo i do *batch* do *step* t;
$L\left(f\left(x^{(i)}, \theta \right), y^{(i)} \right)$	= função *loss*;
$\hat{g}_t$	= estimativa do gradiente da função *loss* em ordem ao parâmetro θ, formulada sobre os m exemplos do *batch* do *step* t.

como se realiza um *transfer learning*, os parâmetros do LM DistilBERT não se alteram, pelo que a redução do erro advém da variação sofrida nos parâmetros da camada de classificação. Quanto maior, mais rápido é o treino (menor, melhora a capacidade de generalização, tornando, contudo, o treino demasiado longo), mas a sua dimensão é limitada pela memória RAM.

[16] Hiperparâmetro que determina a proporção do gradiente da função *loss* da responsabilidade de cada parâmetro que lhe será efetivamente imputada, alterando o parâmetro nessa proporção para que prossiga o treino e se avalie o erro (*loss*) no próximo *step* no caminho da sua minimização. Se o valor deste hiperparâmetro for demasiado alto, poderá conduzir o otimizador a problemas de convergência. Se pequeno, poderá fazer com que o treino seja muito demorado. Por tentativa e erro, dever-se-á tentar que seja o mais alto possível desde que se perceba uma convergência na redução do erro.

... realizando de seguida a atualização desse parâmetro, subtraindo-lhe o gradiente (daí *Descent*) da função *loss* da sua responsabilidade, apurado na equação 3.8, à razão da *learning rate* fixada (ver equação 3.9).

$$\theta_{t+1} = \theta_t - \eta \hat{g}_t \tag{3.9}$$

onde:

θ_t = parâmetro afeto ao *embedding* x a atualizar, observado no início do *step* t;
$\hat{g}_t$ = estimativa do gradiente da função *loss* em ordem ao parâmetro θ, formulada sobre os m exemplos do *batch* do *step* t;
η = *learning rate* fixada;
θ_{t+1} = parâmetro afeto ao *embedding* x atualizado no *step* t para iniciar o *step* $t+1$.

Ora, dada a **elevada dimensionalidade** dos *embeddings* que encontramos nestes **problemas de otimização** da FFNN de classificação, habitualmente **não convexos**, o que conduz a diferentes sensibilidades das dimensões dos *embeddings* à *learning rate* fixada (demasiado pequena para algumas ou demasiado elevada para outras), surge o otimizador **AdaGrad**[12], abreviatura de *Adaptive Gradient*, que vem mitigar este problema ao adaptar a *learning rate* a cada parâmetro dessas dimensões, de acordo com o seu histórico de gradientes, controlando-a. Mais propriamente, acumula o quadrado dos gradientes ao longo do *steps* para a corrigir em função destes, pelo que promove uma redução mais rápida da agora *adaptive learning rate* em parâmetros que no passado tiveram uma maior responsabilidade pelo erro, tornando mais estáveis as suas atualizações, mitigando assim os seus problemas de convergência (ver equação 3.10).

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\varepsilon + \sum_{\tau=1}^{t} (\hat{g}_\tau)^2}} \odot \hat{g}_t \tag{3.10}$$

onde:

θ_t = parâmetro afeto ao *embedding* x a atualizar, observado no início do *step* t;
$\hat{g}_t$ = estimativa do gradiente da função *loss* em ordem ao parâmetro θ, formulada sobre os m exemplos do *batch* do *step* t;
η = *learning rate* fixada;
θ_{t+1} = parâmetro afeto ao *embedding* x atualizado no *step* t para iniciar o *step* $t+1$;
$\sum_{\tau=1}^{t} (\hat{g}_\tau)^2$ = soma dos quadrados dos gradientes estimados ao longo do treino até ao presente *step* t;
ε = termo para evitar a divisão por zero, especialmente se não for acautelada uma equilibrada inicialização dos parâmetros.

No entanto, o facto de a *adaptive learning rate* vir a reduzir-se nos parâmetros em função da acumulação da série histórica dos gradientes ocorrida desde o início do treino, pode, em alguns casos, levar a que a aprendizagem deixe de se fazer, uma vez aproximada rapidamente de zero. Ora, deixam assim igualmente de fazer repercutir os seus gradientes atuais (correspondentes a erro ainda a carecer de minimização) na correção dos seus parâmetros, impedindo a convergência global da rede. Quando, em simultâneo, coexistem outros parâmetros associados a dimensões dos *embeddings* que representam *tokens* específicas ligadas a classes menos frequentes, que requerem mais tempo de treino, a interrupção da capacidade do modelo em aprender deixa expostas as suas ainda convergências por realizar.
Este foi justamente o nosso caso, onde os exemplos das quatro classes menos representadas terão provocado atualizações a um ritmo mais lento, o que não permitiu ao modelo ganhar suficiente capacidade preditiva para as classificar corretamente.
Para ultrapassar esta questão, utilizou-se um outro método que igualmente calcula uma *adaptive learning rate* ao nível do parâmetro, este conceptualmente próximo do **AdaGrad**, ainda que incorpore o histórico de gradientes a um nível médio.
Trata-se do otimizador **Adam**[19], utilizado frequentemente no treino de redes neuronais profundas. A sua designação é a abreviatura de estimativa por *Adaptive Moment*, combinando os conceitos de ***momentum*** e o já discutido ***adaptive learning rate***. Tal como este último, também o *momentum* recorre ao histórico dos gradientes que foram sendo aplicados em cada parâmetro. Mas contrariamente a um controlo da convergência do processo de otimização do parâmetro instrumentalizado pela *adaptive learning rate*, o *momentum* tem por objetivo acelerar o processo de otimização em cada *step*, consistindo em fazer depender o "gradiente" a imputar ao parâmetro, da tendência da série de gradientes passados, ponderando ainda assim mais os gradientes de *steps* mais recentes (os mais antigos vão detendo cada vez menor peso à medida que os *steps* se realizam).
O **Adam** combina estes dois conceitos para proceder à atualização dos parâmetros da seguinte forma:

- ***Momentum***: calcula a média móvel exponencial (*exponentially decaying average*) da série de gradientes, estimando, por um lado, a sua tendência global e, por outro, alisando o efeito de gradientes "extremos", tornando as atualizações dos parâmetros mais robustas (ver equação 3.11). Para evitar o viés de m_t para zero, devido a ter sido inicializada a esse valor, procede-se à correção das suas versões indicada na equação 3.12.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, \hat{g}_t \tag{3.11}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^{\tau}} \tag{3.12}$$

  onde:

m_{t-1} = estimativa da média móvel exponencial dos gradientes calculada no *step* $t-1$;
$\hat{g}_t$ = estimativa do gradiente da função *loss* em ordem ao parâmetro θ, formulada nos exemplos do *batch* do *step* t;
β_1 = ponderador potência das médias móveis dos gradientes e do gradiente do *step* t (fixado usualmente em `0.9`);
m_t = estimativa da média móvel exponencial dos gradientes calculada no *step* t;
β_1^τ = ponderador potência acumulado das médias móveis dos gradientes e dos gradientes do histórico de atualizações τ;
$\hat{m}_t$ = estimativa da média móvel exponencial dos gradientes calculada no *step* t corrigida.

– ***Adaptive learning rate***: calcula a média móvel exponencial (*exponentially decaying average*) da série dos quadrados dos gradientes, estimando, por um lado, a magnitude e variabilidade dos gradientes e, por outro, funcionando como controlo da *learning rate* a aplicar em cada *step* (ver equação 3.13). Tal como acontece com m_t, também v_t necessita de ser corrigida pelo viés provocado na sua inicialização (ver equação 3.14).

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\,\hat{g}_t^2 \tag{3.13}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^\tau} \tag{3.14}$$

onde:

v_{t-1} = estimativa da média móvel exponencial dos quadrados dos gradientes calculada no *step* $t-1$;
$\hat{g}_t^2$ = estimativa do quadrado do gradiente da função *loss* em ordem ao parâmetro θ, formulada sobre os exemplos do *batch* do *step* t;
β_2 = ponderador potência das médias móveis dos quadrados dos gradientes e do quadrado do gradiente do *step* t (fixado usualmente em `0.999`);
v_t = estimativa da média móvel exponencial dos quadrados dos gradientes calculada no *step* t;
β_2^τ = ponderador potência acumulado das médias móveis dos quadrados dos gradientes e dos quadrados dos gradientes do histórico de atualizações τ;
$\hat{v}_t$ = estimativa da média móvel exponencial dos quadrados dos gradientes calculada no *step* t corrigida.

Levando em conta estes "estabilizadores", a atualização promovida pelo **Adam** é a

realizada da seguinte forma (ver equação 3.15):

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \varepsilon}\hat{m}_t \tag{3.15}$$

onde:

θ_t = parâmetro afeto ao *embedding* x a atualizar, observado no início do *step* t;
$\hat{m}_t$ = estimativa da média móvel exponencial dos gradientes calculada no *step* t ou *tendência do gradiente* (corrigida);
$\sqrt{\hat{v}_t}$ = raiz quadrada da estimativa da média móvel exponencial dos quadrados dos gradientes calculada no *step* t ou *magnitude do gradiente* (corrigida);
ε = termo para evitar a divisão por zero;
η = *learning rate* fixada;
θ_{t+1} = parâmetro afeto ao *embedding* x atualizado no *step* t para iniciar o *step* $t + 1$.

Conforme concluído na apresentação do **AdaGrad**, os seus resultados insatisfatórios levaram-nos a abdicar da sua utilização como hiperparâmetro de diferenciação de classificadores, tendo assim o **hiperparâmetro otimizador** sido fixado no **Adam**. Assim, o teste sistemático dos hiperparâmetros da otimização irá ser protagonizado pelo *Adam*, observando-se os impactos dos níveis da *learning rate* no seu desempenho.

Assim, esta investigação irá deixar variar, entre níveis pré-determinados, dois hiperparâmetros, um representante da arquitetura do modelo e outro que controla o processo de otimização realizado no treino:

- Número de neurónios na *hidden layer* da FFNN de classificação: [64, 128];
- *Learning rate*[16]: [8e-3, 1e-3, 3e-4].

3.4.2.4 A Seleção do Classificador

Estabelecida a arquitetura e as componentes alternativas para configuração de classificadores representantes da abordagem DL, foi desenhado um processo de *grid search*, com vista a encontrar a *accuracy* máxima que guiará a escolha do melhor modelo pré-treinado para classificação de reclamações de produtos financeiros. Os seus resultados são apresentados no capítulo seguinte (ver 4.3).

4

Resultados

4.1 Introdução

Neste capítulo apresentam-se os resultados da solução técnica deste estudo comparativo, divididos entre a sua exposição em cada uma das abordagens e a sua consolidação na derradeira análise comparativa. Recorde-se que é justamente no apuramento dos resultados que, na perspetiva operacional, o processo analítico volta a convergir, uma vez aplicarem-se os mesmos métodos de avaliação aos classificadores entretanto desenvolvidos.
Numa concretização das métricas de avaliação de desempenho de classificadores apresentadas no ponto 2.6, indicamos seguidamente as análises utilizadas na avaliação:

- Comparação da métrica *accuracy* (ver 2.6), obtida no conjunto de validação no final do treino por classificadores construídos segundo a mesma abordagem, para identificação daquele que detém o valor mais alto, consubstanciando-se assim no candidato a melhor classificador, bem como o valor dessa métrica obtida no conjunto de teste para estimação do verdadeiro valor da métrica, homologando "o melhor" se este resultado estiver "em linha" com o obtido na validação;

- Comparação da métrica *accuracy* (ver 2.6) obtida no conjunto de treino e validação ao longo do processo de treino por classificadores construídos segundo a mesma abordagem, procurando-se analisar, em cada um e globalmente, o percurso da maximização parametrizada justamente para descrever a influência dos parâmetros/hiperparâmetros nessa maximização da *accuracy* e seleção da melhor hipótese/melhor modelo em cada *set* de parâmetros, e finalmente na identificação do candidato a melhor classificador;

- Análise das métricas de desempenho por classe obtidas pelo homologado melhor classificador, procurando-se, por um lado, distinguir as classes onde a previsão do produto subjacente ao texto da reclamação terá maior e menor sucesso através da leitura da métrica *precision* (ver 2.6) e, por outro, as classes onde os Clientes serão mais e menos "incomodados" com o erro de classificação por via da interpretação da métrica *recall* (ver 2.6), qualificando o erro através das verdadeiras classes

(erradamente) previstas.

4.2 Abordagem Clássica

Após realização do processo analítico e de treino das distintas variantes de *feature engineering* do texto das reclamações sob parametrizações pré-estabelecidas, conforme indicado no capítulo anterior em 3.4.1.4, foi possível verificar que, globalmente, o nível de *accuracy* dos modelos baseados no SVM são elevados (ver figura 4.1).
O *set* de *feature engineering* que produz o candidato a melhor classificador é o **BoW dos *n-grams da reclamação* transformados em frequências relativas, com redução da dimensionalidade via capacidade das classes em discriminar a (agora) distribuição de frequências relativas de cada *n-gram* (medida pelo *F-ratio*) e sem ponderar as frequências relativas com esse poder discriminante**, com uma *accuracy* de 0.81 no conjunto de validação (ligeiramente mais de 4 acertos em cada 5 tentativas), corroborada em estimativa do acerto real no conjunto de teste.

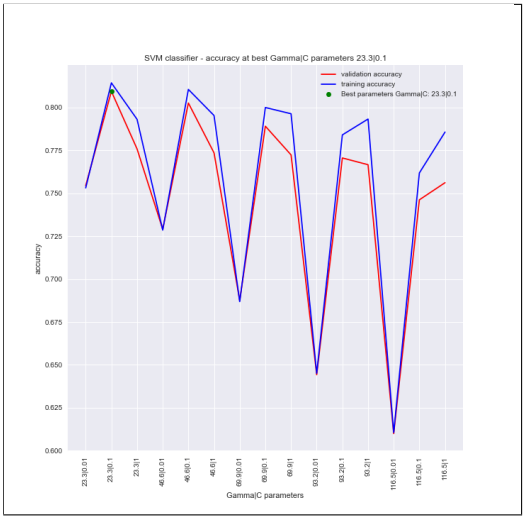
Classic Models

Model	Accuracy	
Algorithm | BoW type | Feature selection | Feature importance	Validation	Test
SVM | Abs.Freq.BOW | Elbow | NoFRatioWeighting	0,79	0,79
SVM | Rel.Freq.BOW | Elbow | NoFRatioWeighting	**0,81**	**0,81**
SVM | Abs.Freq.BOW | Elbow | FRatioWeights	0,80	0,79
SVM | Rel.Freq.BOW | Elbow | FRatioWeights	0,80	0,79

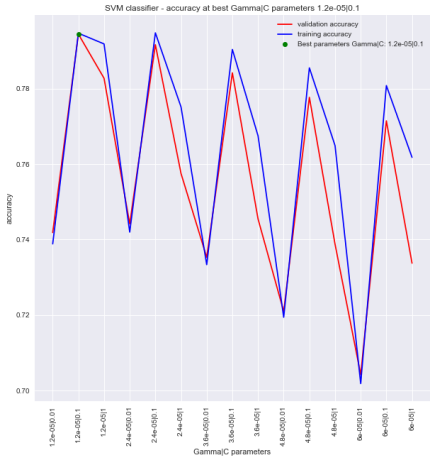
Figura 4.1: Abordagem clássica: resultados das variantes SVM.

Contextualizando inicialmente a análise relativa à comparação da métrica *accuracy* obtida no conjunto de treino e validação ao longo do processo de treino pelos classificadores (ver figura 4.2), dá-se aqui nota prévia de que a sequência de níveis dos parâmetros da SVM é crescente, tanto para C como para γ, pelo que cada passagem de nível em C significa penalizar mais o erro de classificação e em γ significa apertar o raio de influência do vetor de *support* para com os demais exemplos, i.e., estes têm que estar mais próximos para serem por aquele influenciados.
Situando-nos no classificador candidato a melhor (sub-gráfico (a)) e sabendo-se ser o *loop* do *grid search* fixado em cada nível de γ para percorrer todo o nível de C, indica-se que a *accuracy* de 0.81 alcançada no conjunto de validação foi atingida logo no 2.º nível de regulação ($C = 0.1$) do 1.º nível de γ testado ($\gamma = 23.3$). Apesar de tal confirmar que o γ de partida foi fixado (base para múltiplos dos níveis seguintes) no raio "ótimo" (na realidade, este foi fixado via "estimador da máxima verosimilhança"), indicia que para efeitos de melhor acompanhamento do processo de otimização, seria conveniente observar o desempenho a um valor mais baixo de γ, alargando assim o raio, e com isso permitir averiguar se não existiriam reclamações de uma classe que deixassem de ficar "isoladas" das suas pares pelo raio estar até então demasiado apertado. Verifica-se ainda que, à medida que

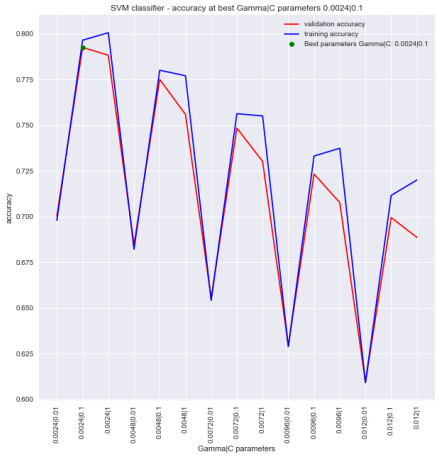
o γ sobe, o aumento da regulação, após regulação ótima, melhor funciona para o treino, mas tal não é validado pelo conjunto responsável por tal (só num nível de γ a atuação do regulador compensa).
Comparando todas as variantes de *feature engineering*, as variantes com ponderador (atribuição de importância), independentemente dos tipos de frequência e com o regulador mais eficiente ($C = 0.1$), parecem sofrer níveis de degradação de desempenho mais baixos em função do aumento de γ, podendo advir de a importância ter tornado as *features* mais desiguais, levando a que o SVM olhe "apenas" para algumas.



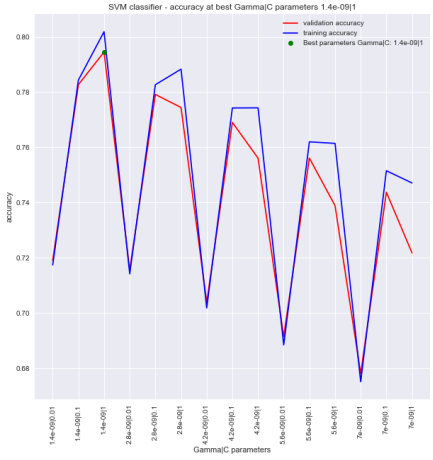
(a) Treino SVM Freq Rel N ponderada



(b) Treino SVM Freq Rel Ponderada



(c) Treino SVM Freq Abs N ponderada



(d) Treino SVM Freq Abs Ponderada

Figura 4.2: Abordagem clássica: desempenho do SVM ao longo do treino por variante.

Debruçando-nos agora nas métricas de desempenho por classe obtidas pelo "melhor" classificador (ver figura 4.3), é possível indicar:

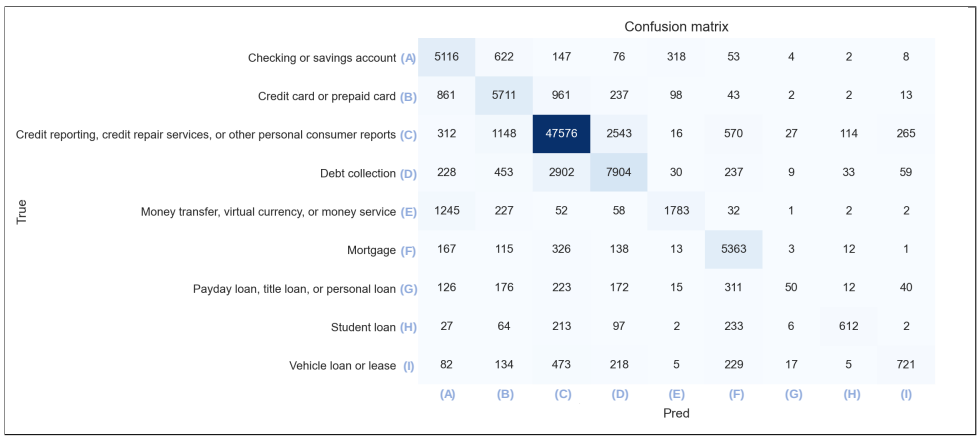
- As classes onde a qualidade da previsão do produto subjacente ao texto da reclamação é maior são: *Credit reporting, credit repair services, or other personal consumer reports* (a classe mais fácil de prever, mesmo sem modelo!), *Money transfer, virtual currency, or money service*, *Student loan* e *Mortgage*, todas com *precision* superiores a 0.7. Realce para as 3 últimas classes, pois detinham uma visibilidade reduzida em sede de treino;

- A classe onde o insucesso da previsão é maior e por isso mais crítico é a *Payday loan, title loan, or personal loan*, com *precision* de 0.42, i.e., equivocar-se-ia mais de metade das vezes sempre que previsse ser o texto de uma reclamação relativa a este produto;

- As classes onde os Clientes seriam mais "incomodados" com o erro de classificação são: *Payday loan, title loan, or personal loan* (novamente e agora drasticamente),*Vehicle loan or lease* e *Student loan*, todos com um *recall* inferior a 0.5. A este mau desempenho não é alheio o facto de a terminologia financeira utilizada (e ainda, problemas e ecossistemas de empresas envolvidas) nestes créditos ser semelhante.

	precision	recall	f1-score	support
Checking or savings account	0.63	0.81	0.71	6346
Credit card or prepaid card	0.66	0.72	0.69	7928
Credit reporting, credit repair services, or other personal consumer reports	0.90	0.90	0.90	52571
Debt collection	0.69	0.67	0.68	11855
Money transfer, virtual currency, or money service	0.78	0.52	0.63	3402
Mortgage	0.76	0.87	0.81	6138
Payday loan, title loan, or personal loan	0.42	0.04	0.08	1125
Student loan	0.77	0.49	0.60	1256
Vehicle loan or lease	0.65	0.38	0.48	1884

(a) Métricas de desempenho por classe



(b) Matriz de confusão

Figura 4.3: Abordagem clássica: desempenho do "melhor" SVM detalhado por classe.

4.3 Abordagem *Deep Learning*

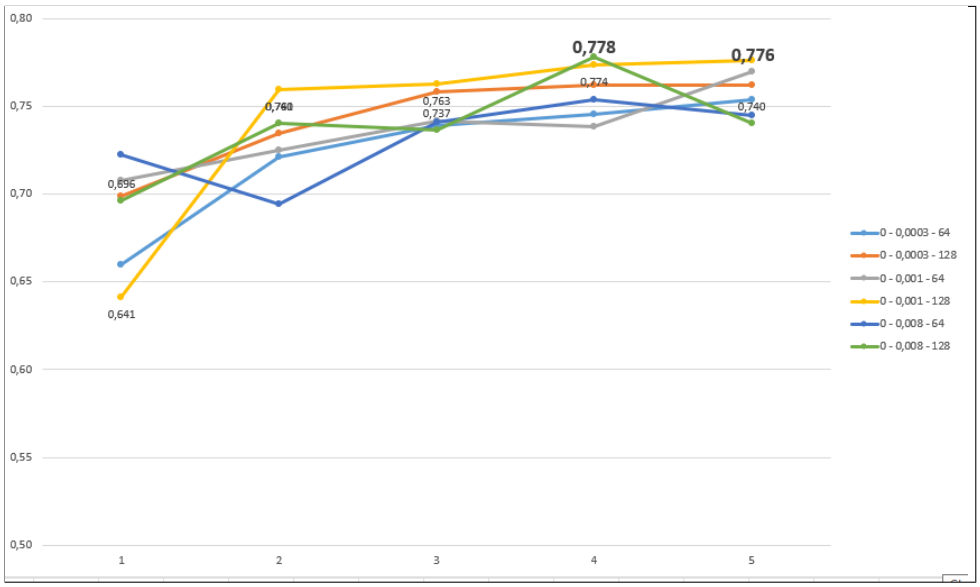
Após realização do processo analítico e de treino dos *sets* de hiperparâmetros aplicados no treino, conforme indicado no capítulo anterior em 3.4.2.3, foi possível verificar que,

globalmente, o nível de *accuracy* dos modelos baseados no DistilBERT são elevados (ver figura 4.4), ainda que aquém dos classificadores clássicos do SVM.
O *set* de hiperparâmetros que produz o candidato a melhor classificador é, marginalmente, o **DistilBERT com *learning rate* mais alta (0.008) e com 128 neurónios na *hidden layer* da FFNN responsável pelo *transfer learning* para a previsão das classes**, com uma *accuracy* 0.78 no conjunto de validação (ligeiramente menos de 4 acertos em cada 5 tentativas), corroborada em estimativa do acerto real no conjunto de teste. Este valor é atingido na 4.ª época. Outro classificador alcança valor praticamente igual - de igual arquitetura da camada de classificação, mas com *transfer learning* mais baixa, aparentando outra estabilidade no treino.

DL Models

Model (Transformer|Learning rate|Classifier #neurons|Optimizer)	Accuracy	
	Validation	Test
DistilBERT|0,001|64|Adam	0,77	**0,77**
DistilBERT|0,0003|64|Adam	0,75	0,75
DistilBERT|0,008|64|Adam	0,75	0,75
DistilBERT|0,001|128|Adam	**0,78**	**0,77**
DistilBERT|0,0003|128|Adam	0,76	0,76
DistilBERT|0,008|128|Adam	**0,78**	**0,77**

(a) Resultados das variantes DistilBERT

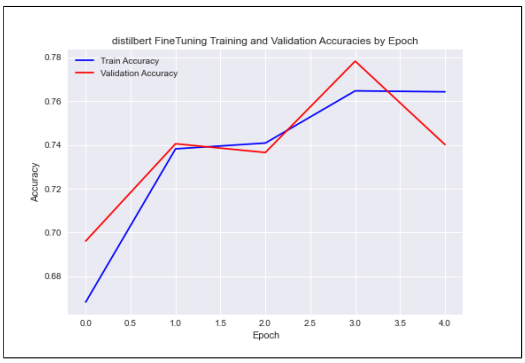


(b) Evolução do *accuracy* de validação das variantes durante o treino

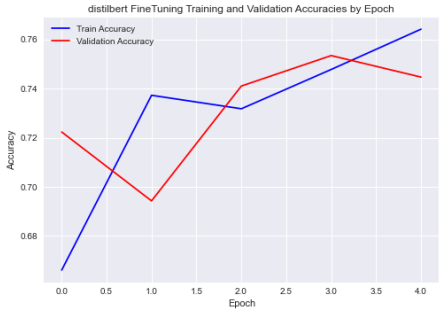
Figura 4.4: Abordagem *Deep Learning*: resultados das variantes DistilBERT.

Situando-nos no classificador candidato a melhor (ver sub-gráfico (a) da figura 4.5), é notória a sua menor consistência no treino, o que é acompanhado pelo seu par de nível de *transfer learning*, o que se justificará justamente pelo nível desta. O melhor desempenho

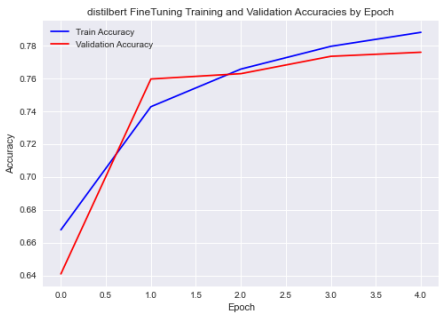
de validação parece ser um "acaso" no caminho da otimização. A estabilidade do treino consegue-se com a *transfer learning* mais baixa, mas a custo da qualidade da classificação. Necessitaria eventualmente de mais umas épocas para atingir o desempenho na validação dos seus "concorrentes", ainda que aparente estacionar a um nível mais baixo. Por outro lado, o aumento de neurónios da camada de classificação parece alavancar a classificação. Em suma, o classificador que provavelmente deveria ser escolhido, seria um que acumulasse melhor e mais estável desempenho, o que apontaria para o classificador DistilBERT de *learning rate* intermédia (0.001) e com 128 neurónios.



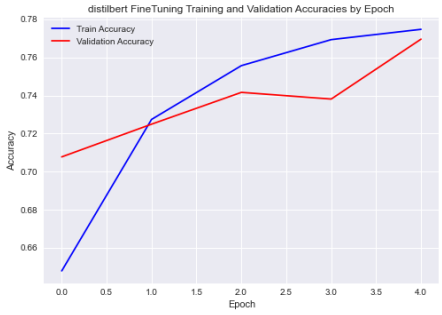
(a) Treino DistilBERT lr0.008nn128



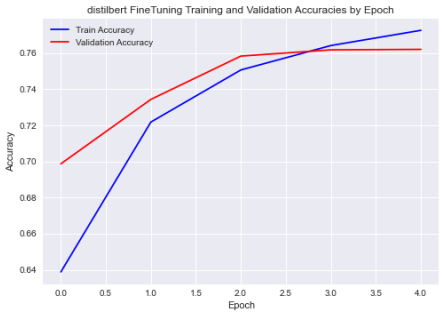
(b) Treino DistilBERT lr0.008nn64



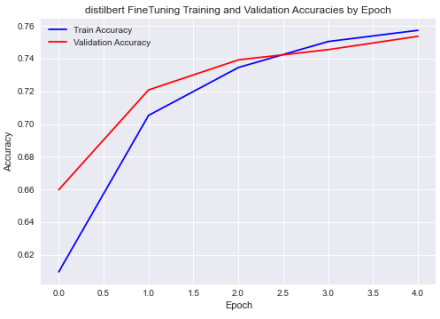
(c) Treino DistilBERT lr0.001nn128



(d) Treino DistilBERT lr0.001nn64



(e) Treino DistilBERT lr0.0003nn128



(f) Treino DistilBERT lr0.0003nn64

Figura 4.5: Abordagem *Deep Learning*: desempenho do DistilBERT ao longo do treino por variante.

Debruçando-nos agora nas métricas desempenho por classe obtidas pelo "melhor" classificador (ver figura 4.6), é possível indicar:

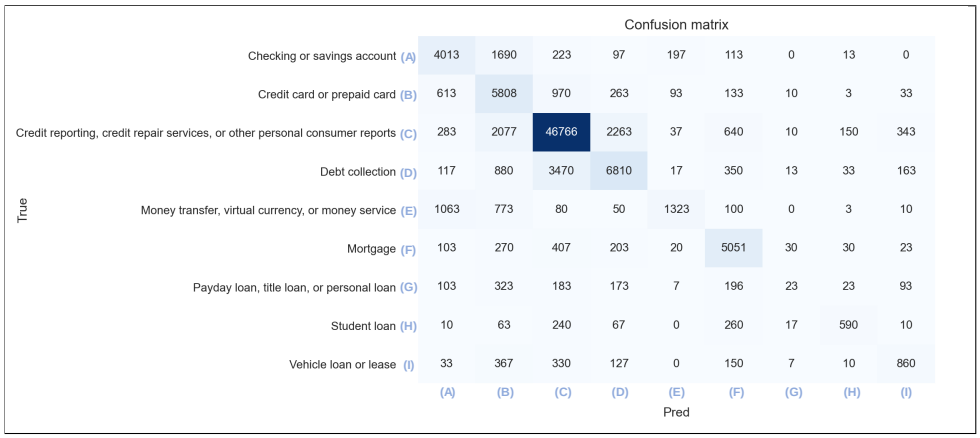
- As classes onde a qualidade da previsão do produto subjacente ao texto da reclamação é maior são: *Credit reporting, credit repair services, or other personal consumer reports* (a classe mais fácil de prever, mesmo sem modelo!), *Money transfer, virtual currency, or money service* e *Mortgage* (este ascende à 3.ª posição, por troca com *Student loan*, todas com *precision* superiores a 0.7;

- À semelhança com SVM, a classe onde o insucesso da previsão é maior e por isso mais crítico é a *Payday loan, title loan, or personal loan*, com *precision* de apenas 0.21;

- As classes onde os Clientes serão mais "incomodados" com o erro de classificação são: *Payday loan, title loan, or personal loan* (novamente...), *Money transfer, virtual currency, or money service*, *Vehicle loan or lease* e *Student loan*, todos com um *recall* inferior a 0.5. Temos novamente aqui o distúrbio provocado pela terminologia financeira comum aos créditos (veja-se na matriz de confusão - recordar ponto 2.6 - como *Mortgage* captura reclamações de *Student loan*, *Payday loan, title loan, or personal loan* e *Vehicle loan or lease*), mas surge um *Money transfer, virtual currency, or money service*, com equívocos de linguagem com *Checking or savings account*, o que também não surpreende por ser esta a "carteira" de origem ou destino daquele serviço.

	precision	recall	f1-score	support
Checking or savings account	0.63	0.63	0.63	6345.9999999998545
Credit card or prepaid card	0.47	0.73	0.58	7928.000000000249
Credit reporting, credit repair services, or other personal consumer reports	0.89	0.89	0.89	52569.99999999426
Debt collection	0.68	0.57	0.62	11853.999999999129
Money transfer, virtual currency, or money service	0.78	0.39	0.52	3402.000000000065
Mortgage	0.72	0.82	0.77	6138.000000000102
Payday loan, title loan, or personal loan	0.21	0.02	0.04	1124.9999999999948
Student loan	0.69	0.47	0.56	1256.0000000000005
Vehicle loan or lease	0.56	0.46	0.50	1884.000000000015

(a) Métricas de desempenho por classe



(b) Matriz de confusão

Figura 4.6: Abordagem *Deep Learning*: desempenho do "melhor" DistilBERT detalhado por classe.

4.4 Comparação de Resultados entre Abordagens

Em jeito de súmula de resultados comparativos entre abordagens, confronta-se as respetivas métricas de desempenho dos seus representantes eleitos por *accuracy*, na qual estão distanciados em 4 pontos percentuais (p.p.).
Fazendo deste delta o nosso referencial, é possível comentar que a abordagem clássica baseada em *Machine Learning* prevê significativamente melhor onde elas são globalmente mais difíceis de diferenciar (créditos) e incomoda menos em produtos tendencialmente mais "populares" em reclamações. A abordagem *Deep Learning* consegue ainda assim ter uma vitória de compensação em incomodar menos Clientes insatisfeitos de financiamento automóvel (mas sem deslumbrar!).

0,00	SVM superior ao DistilBERT em 4 ou mais p.p.
0,00	SVM inferior ao DistilBERT em 4 ou mais p.p.

		Precision		*Recall*		*F1-score*	
Product	p	SVM	DistilBERT	SVM	DistilBERT	SVM	DistilBERT
Checking or savings account	6,9%	0,63	0,63	**0,81**	0,63	**0,71**	0,63
Credit card or prepaid card	13,2%	**0,66**	0,47	0,72	0,73	**0,69**	0,58
Credit reporting, credit repair services, or other personal consumer reports	56,9%	0,90	0,89	0,90	0,89	0,90	0,89
Debt collection	10,9%	0,69	0,68	**0,67**	0,57	**0,68**	0,62
Money transfer, virtual currency, or money service	1,8%	0,78	0,78	**0,52**	0,39	**0,63**	0,52
Mortgage	7,6%	0,76	0,72	**0,87**	0,82	**0,81**	0,77
Payday loan, title loan, or personal loan	0,1%	**0,42**	0,21	0,04	0,02	**0,08**	0,04
Student loan	0,9%	**0,77**	0,69	0,49	0,47	0,60	0,56
Vehicle loan or lease	1,7%	**0,65**	0,56	**0,38**	0,46	0,48	0,50

Figura 4.7: Resultados comparativos SVM vs. DistilBERT.

5

Conclusões

Esta investigação de técnicas de classificação de texto sob abordagens distintas permitiu, por um lado, reconhecer a exigência do *feature engineering* presente no desenvolvimento de MC de ML, com maior possibilidade de retorno do investimento já em fase de interpretação de resultados e, por outro, experienciar a maior facilidade de mapeamento das *features* conseguido pela abordagem DL , com reduzida intervenção do *domain expertise*, exigindo em contrapartida maior abstração na interpretação das classificações.

No final, as dificuldades sentidas na classificação das classes foram relativamente as mesmas, com a abordagem clássica (MC de ML) a sair-se menos mal, ainda que a comparação possa ser menos robusta por não ter sido dado à abordagem DL, nem o tempo de processamento, nem o volume de dados a partir dos quais habitualmente consegue distinguir-se.

Já na perspetiva do desempenho computacional, observou-se uma complexidade "induzida" próxima entre os dois classificadores representantes de cada abordagem. É possível que este controlo de complexidade computacional tenha tido impacto na qualidade dos resultados dos modelos de *Deep Learning*, devendo-se provavelmente menos ao volume de dados que lhes suprimimos do que ao tempo que lhes foi dado para treino.

Levando em conta as condições de desenvolvimento deste trabalho e os resultados obtidos, fica a ideia que a abordagem clássica (MC de ML), sustentada num espírito artesanal alicerçado no *domain expertise* e suscitada em contexto de recursos escassos, de democraticidade de acesso aos recursos limitada ou de menor altruísmo na sua partilha, tem capacidade de acrescentar valor.

Bibliografia

[1] A. Adhikari et al. *DocBERT: BERT for Document Classification*. 2019. DOI: 10.48550/ARXIV.1904.08398. URL: https://arxiv.org/abs/1904.08398 (ver p. 26).

[2] M. Allahyari et al. *A Brief Survey of Text Mining: Classification, Clustering and Extraction Techniques*. 2017. DOI: 10.48550/ARXIV.1707.02919. URL: https://arxiv.org/abs/1707.02919 (ver p. 6).

[3] D. Bahdanau, K. Cho e Y. Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. DOI: 10.48550/ARXIV.1409.0473. URL: https://arxiv.org/abs/1409.0473 (ver p. 19).

[4] K. Bennani-Smires et al. «Simple Unsupervised Keyphrase Extraction using Sentence Embeddings». Em: *Proceedings of the 22nd Conference on Computational Natural Language Learning*. Brussels, Belgium: Association for Computational Linguistics, 2018-10, pp. 221–229. DOI: 10.18653/v1/K18-1022. URL: https://aclanthology.org/K18-1022 (ver p. 12).

[5] B. Boser, I. Guyon e V. Vapnik. «A Training Algorithm for Optimal Margin Classifier». Em: *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory* 5 (1996-08). DOI: 10.1145/130385.130401 (ver p. 15).

[6] C. Bucila, R. Caruana e A. Niculescu-Mizil. «Model compression». Em: *KDD* (2006). URL: https://www.cs.cornell.edu/~caruana/compression.kdd06.pdf (ver p. 43).

[7] C. F. P. Bureau. *CFPB Consumer Complaint Database*. 2011-2022. URL: https://www.consumerfinance.gov/data-research/consumer-complaints/ (acedido em 2022-12-03) (ver pp. 1, 30).

[8] J. Camacho-Collados e M. T. Pilehvar. *On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis*. 2017. DOI: 10.48550/ARXIV.1707.01780. URL: https://arxiv.org/abs/1707.01780 (ver p. 8).

[9] R. Campos et al. «YAKE! Keyword extraction from single documents using multiple local features». Em: *Information Sciences* 509 (2020), pp. 257–289. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2019.09.013. URL: https://www.sciencedirect.com/science/article/pii/S0020025519308588 (ver p. 12).

[10] W. contributors. *Kullback–Leibler divergence — Wikipedia, The Free Encyclopedia*. 2022-09. URL: https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence (acedido em 2022-10-13) (ver p. 14).

[11] J. Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805 (ver pp. 20, 26, 46).

[12] J. Duchi, E. Hazan e Y. Singer. «Adaptive subgradient methods for online learning and stochastic optimization». Em: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159. URL: http://jmlr.org/papers/v12/duchi11a.html (ver p. 49).

[13] A. Gasparetto et al. «A Survey on Text Classification Algorithms: From Text to Predictions». Em: *Information* 13.2 (2022). ISSN: 2078-2489. DOI: 10.3390/info13020083. URL: https://www.mdpi.com/2078-2489/13/2/83 (ver p. 6).

[14] C. Hansen et al. «The Fourth Paradigm: Data-Intensive Scientific Discovery». Em: 2009-01, pp. 153–163 (ver p. 6).

[15] P. He et al. *DeBERTa: Decoding-enhanced BERT with Disentangled Attention*. 2021. arXiv: 2006.03654 [cs.CL] (ver pp. 27, 42).

[16] G. Hinton, O. Vinyals e J. Dean. *Distilling the Knowledge in a Neural Network*. 2015. DOI: 10.48550/ARXIV.1503.02531. URL: https://arxiv.org/abs/1503.02531 (ver pp. 26, 43, 44).

[17] T. Joachims. «A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization.» Em: 1996-03 (ver p. 11).

[18] S.-B. Kim et al. «Some Effective Techniques for Naive Bayes Text Classification». Em: *IEEE Transactions on Knowledge and Data Engineering* 18.11 (2006), pp. 1457–1466. DOI: 10.1109/TKDE.2006.180 (ver p. 23).

[19] D. P. Kingma e J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG] (ver p. 50).

[20] Z. Lan et al. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. 2020. arXiv: 1909.11942 [cs.CL] (ver p. 42).

[21] Y. Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL] (ver pp. 27, 42).

[22] H. Lodhi et al. «Text Classification Using String Kernels». Em: *Journal of Machine Learning Research* 2 (2002-06), pp. 419–444. DOI: 10.1162/153244302760200687 (ver p. 24).

[23] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (ver p. ii).

[24] Y. Meng et al. *Text Classification Using Label Names Only: A Language Model Self-Training Approach*. 2020. DOI: 10.48550/ARXIV.2010.07245. URL: https://arxiv.org/abs/2010.07245 (ver p. 6).

[25] R. Mihalcea e P. Tarau. «TextRank: Bringing Order into Text». Em: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain: Association for Computational Linguistics, 2004-07, pp. 404–411. URL: https://aclanthology.org/W04-3252 (ver p. 12).

[26] T. Mikolov et al. *Distributed Representations of Words and Phrases and their Compositionality*. 2013. DOI: 10.48550/ARXIV.1310.4546. URL: https://arxiv.org/abs/1310.4546 (ver p. 13).

[27] T. Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. DOI: 10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781 (ver p. 13).

[28] F. Pedregosa et al. «Scikit-learn: Machine Learning in Python». Em: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (ver p. 34).

[29] J. Pennington, R. Socher e C. Manning. «GloVe: Global Vectors for Word Representation». Em: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014-10, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162 (ver p. 13).

[30] S. Robertson, S. Walker e M. Hancock-Beaulieu. «Okapi at TREC-7: Automatic ad hoc, filtering, VLC and interactive». Em: 1998-01, pp. 199–210 (ver p. 23).

[31] J. Rodrigues et al. *Advancing Neural Encoding of Portuguese with Transformer Albertina PT-**. 2023. arXiv: 2305.06721 [cs.CL] (ver p. 28).

[32] V. Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2020. arXiv: 1910.01108 [cs.CL] (ver p. 42).

[33] M. Schuster e K. Nakajima. «Japanese and Korean Voice Search». Em: *International Conference on Acoustics, Speech and Signal Processing*. 2012, pp. 5149–5152 (ver p. 10).

[34] D. Stammbach e E. Ash. «DocSCAN: Unsupervised Text Classification via Learning from Neighbors». Em: (2021). DOI: 10.48550/ARXIV.2105.04024. URL: https://arxiv.org/abs/2105.04024 (ver p. 6).

[35] E. S. Tellez et al. «Gender and language-variety Identification with MicroTC». Em: *CLEF*. 2017 (ver p. 24).

[36] A. Vaswani et al. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762 (ver p. 19).

[37] A. Wang et al. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. 2018. DOI: 10.48550/ARXIV.1804.07461. URL: https://arxiv.org/abs/1804.07461 (ver pp. 42, 43).

[38] Wikipedia contributors. *Pearson correlation coefficient — Wikipedia, The Free Encyclopedia*. [Online; acedido em 13-06-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Pearson_correlation_coefficient&oldid=1159533515 (ver p. 43).

[39] Wikipedia contributors. *Phi coefficient — Wikipedia, The Free Encyclopedia*. [Online; acedido em 13-06-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Phi_coefficient&oldid=1140138244 (ver p. 43).

[40] Wikipedia contributors. *Spearman's rank correlation coefficient — Wikipedia, The Free Encyclopedia*. [Online; acedido em 13-06-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Spearman%27s_rank_correlation_coefficient&oldid=1148440452 (ver p. 43).

[41] Z. Yang et al. «Hierarchical Attention Networks for Document Classification». Em: 2016-01, pp. 1480–1489. DOI: 10.18653/v1/N16-1174 (ver p. 25).

[42] Y. Yu et al. «Fine-Tuning Pre-trained Language Model with Weak Supervision: A Contrastive-Regularized Self-Training Approach». Em: (2020). DOI: 10.48550/ARXIV.2010.07835. URL: https://arxiv.org/abs/2010.07835 (ver p. 6).

[43] X. Zhang, J. Zhao e Y. LeCun. *Character-level Convolutional Networks for Text Classification*. 2015. DOI: 10.48550/ARXIV.1509.01626. URL: https://arxiv.org/abs/1509.01626 (ver p. 25).