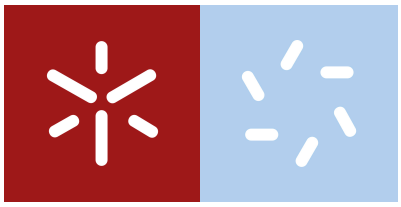


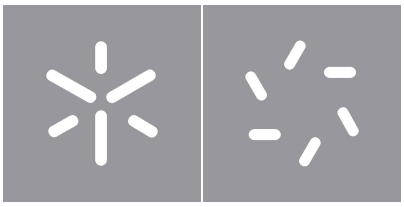
Universidade do Minho
Escola de Ciências

Projeto Integrado em Matemática e Computação

Análise de sentimento do discurso em transmissões televisivas de jogos de futebol

Eduardo Dias, Hugo Rocha, Simão Quintela, Tiago Monteiro

junho 2024



Universidade do Minho
Escola de Ciências

Projeto Integrado em Matemática e Computação

Análise de sentimento do discurso em transmissões televisivas de jogos de futebol

Eduardo Dias, Hugo Rocha, Simão Quintela, Tiago Monteiro

Trabalho efetuado sob a orientação de

António Oliveira (DTx)

Armindo Lobo (DTx)

Emanuel Gouveia (DTx)

Ricardo Teixeira (DTx)

Fernanda Costa (UMinho)

José Oliveira (UMinho)

junho 2024

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

No âmbito do Mestrado em Matemática e Computação da Universidade do Minho, foi realizado um projeto integrado na Associação Laboratório Colaborativo em Transformação Digital (DTx).

1.1 DTx

O Laboratório Colaborativo em Transformação Digital – DTx, é uma associação privada sem fins lucrativos, que desenvolve a sua atividade efetuando investigação aplicada em diferentes áreas associadas à transformação digital. Tendo, como membros associados, a Universidade do Minho, a Universidade Católica Portuguesa, o INL, o CEiiA, a Bosch, o IKEA, a NOS e a Primavera, entre muitos outros (DTx [2024]).

1.2 Objetivos do Projeto Integrado

O objetivo principal deste projeto consistiu em treinar e utilizar modelos de *deep learning* para classificação de texto para detetar momentos de destaque em transmissões TV de jogos de futebol. Mais especificamente, utilizando três tipos de abordagem: usando um classificador *zero-shot* pré-treinado, um classificador *few-shot* treinado com os nossos dados e, por último, utilizar um outro classificador BERT mas após a realização de *fine-tuning*.

Para tal, foram realizados diversos procedimentos que serão desenvolvidos ao longo do presente relatório:

- Obtenção do dataset de vídeos de jogos de futebol;
- Extração do aúdio desses vídeos;

- Fragmentação, tradução e transcrição do áudio resultante (em diversa línguas) para texto (em inglês);
- Classificação desses momentos dos jogos como relevantes ou não, usando as três abordagens referidas anteriormente;
- Análise dos resultados obtidos;
- Resultados finais e conclusões.

1.3 Estrutura do relatório e software utilizado

Ao longo deste relatório vamos procurar abordar todos os tópicos interessantes do projeto desde o tratamento da base de dados, os *benchmarks* realizados ao vários tipos de modelos *Whisper* até aos modelos de classificação utilizados e o respetivo *fine-tuning*. O *software* utilizado neste projeto foi a linguagem de programação *Python* a par de um ambiente virtual com todas as depêndencias necessárias instaladas (via *anaconda*). Adotámos o uso de *jupyter notebooks* para facilitar a compreensão de cada fase do projeto e alternámos a execução do código via local e via *Google Colab*, este último com vista a acelerar as transcrições dos ficheiros de áudio e os modelos de classificação através de execução do código com a GPU/TPU.

Capítulo 2

Base de dados SoccerNet e tratamento dos dados

2.1 Introdução

Neste capítulo iremos entrar em detalhe relativamente à resolução dos problemas que nos foram colocados. Iremos também explicar e mostrar os resultados obtidos.

2.2 SoccerNet

O *dataset* que utilizamos para a realização deste projeto foi o **SoccerNet**. O *dataset* contém cerca de 550 jogos de futebol (divididos em 1ª e 2ª parte) com relato em diferentes línguas. Inicialmente selecionamos apenas 4 jogos, de forma a conseguir trabalhar com um conjunto reduzido mas significativo de dados. Posteriormente, selecionamos 10 jogos de forma a testar com certeza e clareza os modelos produzidos.

2.3 Tratamento de dados

O *dataset* é fornecido em formato de vídeo com o respetivo relato do jogo em questão, mas, no entanto, a informação que queremos reter para os objetivos em questão é apenas o som do vídeo. Nesse sentido, extraímos o áudio de cada elemento da base de dados (vídeos) e ficamos com os respetivos ficheiros de áudio em formato .wav (dois ficheiros de áudio relativos à primeira e segunda parte respetivamente). Seguidamente, como o objetivo passa por classificar fragmentos de áudio, segmentámos os ficheiros de áudio inteiros em segmentos de cerca 30 segundos ficando com cerca de 90 a 100 fragmentos por parte. Para garantir que não se perde informação na transição de fragmentos, cada um dos fragmentos começa nos últimos 2 segundos do fragmento anterior com vista a garantir continuidade no áudio.

2.4 Transcrição dos fragmentos de áudio

Após obtenção dos fragmentos de áudio, procedemos à transcrição dos mesmos através do modelo de transcrição **Whisper** (HuggingFace [a]) fazendo a tradução para inglês visto que os áudios estão em várias línguas. Devido ao facto de existirem diversas versões deste modelo, realizamos um *benchmark* dos vários modelos (*tiny, small, base, medium, large, large-v2, large-v3*) sobre todos os fragmentos de um determinado jogo com vista a obter o tempo de execução de cada um deles e uma avaliação da qualidade comparando as diferentes transcrições de um fragmento em específico. Seguem abaixo duas tabelas referentes aos *benchmarks* dos diferentes modelos, corridos em CPU e em GPU do *Google Colab*. Toda o tratamento do áudio desde extração até à fragmentação foi possível através do uso da biblioteca *ffmpeg* em *Python* (Kroening [2017]).

jogo	model	run_time_per_game
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-tiny	7:52
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-base	13:34
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-small	34:52
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-medium	85:36
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-large	173:11
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-large-v2	187:16
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-large-v3	176:02

Figura 1: Comparação dos modelos corridos em CPU

jogo	model	run_time_per_game
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-tiny	2:55
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-base	3:24
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-small	4:23
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-medium	6:48
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-large	10:49
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-large-v2	10:05
2015-02-21 - 18-00 Chelsea 1 - 1 Burnley	openai/whisper-large-v3	9:42

Figura 2: Comparação dos modelos corridos em GPU do *Google Colab*

Como é observável nas tabelas acima, de um modo geral, existe um ganho de tempo extremamente significativo ao correr os modelos com GPU do *Google Colab* em comparação com o CPU. Além disso, o aumento de tempo parece ser gradual consoante o aumento da versão do modelo do *Whisper* utilizado

até à versão *large*. Dentro das diferentes versões *large* não há diferenças significativas nos tempos de execução.

Como dito anteriormente, para comparar a qualidade de todas estas versões do *Whisper*, foi selecionado um fragmento de áudio específico e comparámos a sua transcrição de todos os modelos. Segue abaixo os resultados obtidos nos diferentes modelos *Whisper*.

- **whisper-tiny**: *"Maran Koja, the second man, Manaoja, the father of the barley, has also said that in the first part of the league, and in which the second player won the Chelsea 1-3, the third player won the first one, and that he remembered that he was from the last one, and he ended up remontating, but he has to remember without any difficulties. Exactly, I just said that he was going to say, remember that very well, he started winning the barley, Mark Ungol, but the Chelsea won the second one, as well. That's why I had no problem with that, but as he said, the football is..."*

- **whisper-small**: *"Marco Jalduelo manages the ball's penalty. You have to say that they also gave the first game of Liga and in that match he won the Chelsea 1-3. The ball went forward. You have to remember that it went forward and ended up not remembering that much difficult. Exactly, what he was saying, I remember that game, he started the penalty and scored a goal, but the Chelsea won it. That's why he should have problems today, but as he said, football is..."*

- **whisper-base**: *"Moarran Coya, the duel, the Barley's ball is in control. They also got the first game of the league, and Chelsea won 1-3 in that match. The Barley got ahead of themselves. They had to remember that they got ahead of themselves and ended up getting back on the pitch. They had to remember that Chelsea had too many difficulties. Exactly, that's what I was going to say. I remember that match very well, the Barley started well, scored a goal, but Chelsea won it comfortably. That's why today there shouldn't be any problems either..."*

- **whisper-medium**: *"The duel has begun, Barley's ball is in the air. They also met in the first game of the league, and in that match Chelsea won 1-3. Barley went ahead. You have to remember that he went ahead in that game and ended up going back without too many difficulties. Exactly, that's what I was going to say. I remember that game very well. Barley started well, scored a goal, but Chelsea won comfortably. That's why today shouldn't have any problems either..."*

- **whisper-large**: *"The duel has started, Barley is in control of the ball. It must be said that they also met in the first league match and in that match Chelsea won 1-3. Barley was ahead. We must*

remember that he was ahead in that match and ended up going back without too many difficulties for Chelsea. Exactly, that's what I was going to say. I remember that match very well. Barley started well, scored a goal, but Chelsea won it comfortably. That's why today he shouldn't have any problems either..."

- **whisper-large-v2**: "*Barley started the game, he's in control of the ball. They also met in the first league game, and in that game Chelsea won 1-3. Barley came off the bench. We have to remember that Barley came off the bench in that game and ended up coming back without too many problems. Exactly, that's what I was going to say. I remember that game very well. Barley started well, he scored a goal, but Chelsea won comfortably.*"

- **whisper-large-v3**: "*The duel has started, Barley is in control. They also measured themselves in the first league match. In that match Chelsea won 1-3. Barley advanced. We have to remember that he advanced in that match and ended up going back without too many difficulties. Exactly, that's what I was going to say. I remember that match very well. Barley started well, scored a goal, but Chelsea won it comfortably.*"

Analisando os diferentes trechos de texto é possível detetar falhas na transcrição no início do áudio em todos os modelos até à versão *base* onde o modelo não consegue perceber a frase "Arrancou o duelo..."e atira, em vez disso, nomes próprios. Por haver falhas desse género optámos por deixar de lado este tipo de modelos e escolher a versão *large* que nos pareceu ser a transcrição mais fiável inclusivamente comparando com as versões 2 e 3 que não transcrevem uma última frase dita pelos comentadores. O *medium* também seria uma opção viável, no entanto, em pequenas frases também nos pareceu estar atrás do modelo *large* como por exemplo na frase "*Barley's ball is in the air.*"que não corresponde corretamente ao que foi dito realmente: "*Barley is in control of the ball.*"— transcrito pelo modelo *large*. Dessa forma, em termos de tempo de execução, o modelo demora cerca de 10 minutos por jogo até transcrever todos os áudios.

Capítulo 3

Construção do nosso próprio dataset

Sabendo que, no site do *SoccerNet*, é possível obter, para cada jogo, um ficheiro em formato *json* com as classificações de vários instantes de tempo, foi-nos possível classificar os vários excertos de texto com base nesses ficheiros *json.* Dessa forma, para facilitar o processo de compreensão e acesso aos dados, decidimos consolidar todos os fragmentos dos vários jogos (transcrições) com a respetiva classificação do *SoccerNet*, em dois *datasets*. Ambos os conjuntos de dados possuêm 8 colunas sendo as seguintes 7 colunas comuns entre os dois *datasets*:

- **id:** identificador único de cada linha do *dataset* (cada fragmento);
- **league:** liga de futebol a que pertence o jogo de onde foi retirado o fragmento;
- **game:** jogo de futebol de onde foi retirado o fragmento;
- **fragment_start_time:** instante de tempo de início do fragmento (em segundos);
- **fragment_end_time:** instante de tempo onde termina o fragmento (em segundos);
- **commentary:** transcrição feita pelo modelo *Whisper* do fragmento em questão;
- **model_id:** modelo *Whisper* utilizado para gerar as transcrições (versão *large* foi a escolhida).

Além destes atributos existe uma coluna correspondente à *label* de classificação dos fragmentos onde, no primeiro e principal *dataset*, temos como classificações possíveis: **Highlight** e **Non-Highlight**. A classificação, como dito anteriormente, foi possível através do site do *SoccerNet* que atribuía a certos instantes de tempo 16 *labels* possíveis: **Shots on target, Goal, Penalty** e **Red card** — que associámos no *dataset* como sendo classificações *Highlight*; e ainda as *labels*: **Ball out of play, Throw-in, Foul, Indirect free-kick, Clearance, Shots off target, Corner, Substitution, Kick-off, Direct free-kick, Offside** e **Yellow card** — correspondentes no *dataframe* a classificações *Non-Highlight*. No caso em que os fragmentos não têm nenhum instante de tempo classificado no *SoccerNet*, significa que o

fragmento é desinteressante e portanto foi também classificado como *Non-Highlight*. O *dataset* final contém **todos** os fragmentos e a sua respetiva classificação ficando com 2067 registos, 7 atributos e a respetiva *label* de classificação. De notar que, como era de esperar num jogo de futebol, o *dataset* se encontra desbalanceado, isto porque, contém **1915** registos associados a **Non-Highlight** e apenas **152** registos associados a **Highlight**.

No *dataset* secundário, esta coluna *label* é substituída por uma outra coluna mas com várias *labels* possíveis de classificação visto que, como temos fragmentos de 30 segundos, por vezes alguns fragmentos podem ter vários instantes de tempo com diferentes *labels* vindas do *SoccerNet*. Dessa forma, juntámos as possíveis classificações desse fragmento numa só *string* separados por barra (/) [1] e colocámos neste *dataset* secundário. Ao contrário do primeiro *dataset*, os fragmentos sem nenhum instante de tempo classificado no *SoccerNet* foram descartados, visto não terem nenhuma das 16 *labels* específicas associada. Devido a este último ponto, o *dataset* ficou com 1465 registos, 7 atributos e a *string* com as *labels* possíveis para esse fragmento.

Nota: No primeiro *dataset*, visto que podemos ter várias *labels* possíveis dentro do mesmo fragmento, seguimos a lógica mais natural: se dentro das classificações possíveis do fragmento, existir pelo menos uma *label* correspondente a *Highlight*, então o fragmento será considerado *Highlight*. Caso não existam instantes de tempo classificados pelo *SoccerNet* num determinado fragmento ou, caso existam, mas nenhuma das *labels* correspondentes a *Highlight* está presente, então o fragmento considera-se como sendo *Non-Highlight*.

[1] Exemplo: Corner/Shots on target/Ball out of play

Capítulo 4

Classificação Zero-Shot

4.1 Introdução

Classificação *Zero-Shot* é uma tarefa de processamento de linguagem natural onde um modelo é treinado num conjunto de exemplos rotulados, mas é capaz de classificar novos exemplos de classes que não foram previamente vistas.

Este método, que aproveita um modelo de linguagem pré-treinado, pode ser considerado uma forma de *transfer learning*, o qual é normalmente referenciado como o ato de realizar um *fine-tuning* a um modelo pré-treinado numa tarefa, para este ser capaz de realizar uma tarefa diferente/nova. A classificação *Zero-Shot* pode ser bastante útil quando a quantidade de dados rotulados é pequena.

4.2 Vantagens e desvantagens

4.2.1 Vantagens

Algumas das vantagens deste tipo de classificação são:

- **Flexibilidade** — trata-se de uma abordagem extremamente flexível devido ao facto de poder ser adaptada a novos domínios sem necessidade de treinar novamente o modelo;
- **Aplicações práticas** — pode ser aplicado em matérias onde não é possível obter dados de treino para todas as classes em questão;
- **Generalização** — o modelo aprende de forma generalizada podendo levar a uma melhor compreensão do contexto e das relações entre as classes.

4.2.2 Desvantagens

Alguns dos pontos não tão positivos deste tipo de abordagem são:

- **Desempenho inferior** — este tipo de abordagem pode obter resultados menos positivos em relação a outro tipo de modelos supervisionados, treinados com as classes em questão;
- **Dependência de boas representações semânticas** — o bom desempenho deste tipo de modelos pode estar fortemente dependente de boas representações semânticas e de descrições textuais precisas, o que pode não estar sempre disponível;
- **Complexidade computacional** — pode exigir recursos computacionais consideráveis para processar e generalizar a partir de descrições textuais de classes (quanto maior o número de classes maior será a complexidade computacional).

4.3 Aplicação do método

Neste tipo de classificação decidimos abordar três estratégias diferentes com diferentes tipos de *labels* candidatas dadas ao modelo pré-treinado por nós escolhido, no caso, **bart-large-mnli** — modelo de *zero-shot classification* treinado com base no *dataset* **MultiNLI (MNLI)** (HuggingFace [b]).

Inicialmente optámos por utilizar as *labels* mais naturais: **Highlight** e **Non-Highlight** sobre todo o conjunto de dados principal, no entanto, decidimos testar mais duas estratégias com o propósito de comparar resultados entre as mesmas. Uma delas passou por concentrar todas as *labels* vindas do *SoccerNet* associadas a *Highlight* numa só label (numa única *string* separadas por vírgula) [1] e o mesmo para as *labels* associadas a *Non-Highlight* [2]. Dessa forma ficámos com apenas duas *labels*, o que ajuda em termos de eficiência do modelo podendo ser utilizado todo o *dataset*, e o modelo fica com acesso às *labels* mais específicas vindas do *SoccerNet* associadas às duas classificações possíveis. Por fim, testamos uma outra estratégia que passou por pegar em 3 *labels* vindas do *SoccerNet* para *Highlight* (**Shots on target, Goal, Red card**) e outras 3 *labels* para *Non-Highlight* (**Ball out of play, Clearance, Corner**) totalizando 6 *labels* candidatas para o modelo. A escolha de manter as 16 *labels* como *labels* candidatas para o modelo tornou-se inviável do ponto de vista da eficiência e rapidez de execução devido ao tamanho do *dataset* e ao número de *labels* ser grande, daí a opção de tomar 3 *labels* para cada um dos

[1] *Label* candidata associada a *Highlight*: "Shots on target, Goal, Penalty, Red Card"

[2] *Label* candidata associada a *Non-Highlight*: "Ball out of play, Throw-in, Foul, Indirect free-kick, Clearance, Shots off target, Corner, Substitution, Kick-off, Direct free-kick, Offside, Yellow card"

momentos, tendo o cuidado de escolher *labels* que estão relativamente bem representadas no *dataset*. Visto que aqui se tratam de *labels* específicas utilizámos o *dataset* secundário e filtramos o mesmo de forma a ficar apenas com os registos que contêm na coluna *possible_labels*, pelo menos uma das 6 *labels* candidatas ao modelo. O *dataset* filtrado ficou com 981 registos e as mesmas 8 colunas.

4.4 Avaliação dos resultados das três abordagens

4.4.1 Métricas utilizadas

Devido ao desbalanceamento das duas *labels*, decidimos utilizar várias métricas para avaliar o resultado destas três diferentes abordagens, no caso: **accuracy**, **precision**, **recall** e **F1-Score**. Na avaliação dos modelos de classificação *Zero-Shot*, nos resultados destas métricas, como é de esperar, a *label Highlight* foi considerada a *label* positiva e a *label Non-Highlight* a negativa.

A fórmula associada à **accuracy** é a seguinte:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$ [3]

Esta famosa métrica calcula a percentagem de predições corretas de entre toda a amostra, no entanto, não é a mais interessante no nosso caso visto que esta métrica valoriza todas as classes da mesma forma e o nosso *dataset* encontra-se desbalanceado.

A fórmula associada à **precision** é a seguinte:

$$\text{precision} = \frac{TP}{TP + FP}$$

Com esta métrica conseguimos identificar quantas *predicts* que foram classificadas como *Highlight* são de facto *highlights*.

A fórmula associada à **recall** é a seguinte:

$$\text{recall} = \frac{TP}{TP + FN}$$

[3] TP: *True Positives*; TN: *True Negatives*; FP: *False Positives*; FN: *False Negatives*

Com esta métrica conseguimos identificar quantas *predicts* foram bem classificadas como *Highlight* tendo a noção de quantas foram mal classificadas como *Non-Highlight*.

A fórmula associada ao **F1-Score** é a seguinte:

$$\text{F1-Score} = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

Com esta métrica o objetivo passa por combinar as métricas *precision* e *recall* através de uma média harmónica entre as mesmas e é uma métrica boas para casos em que as classes estão desproporcionais no *dataset*.

Além destas métricas, decidimos fazer um *plot* da matriz de confusão dos diferentes modelos para melhor percepção visual dos resultados.

4.4.2 Resultados

Começando pelos resultados do modelos que utilizou como *labels* candidatas: **Highlight** e **Non-Highlight** (sobre todo o conjunto de dados).

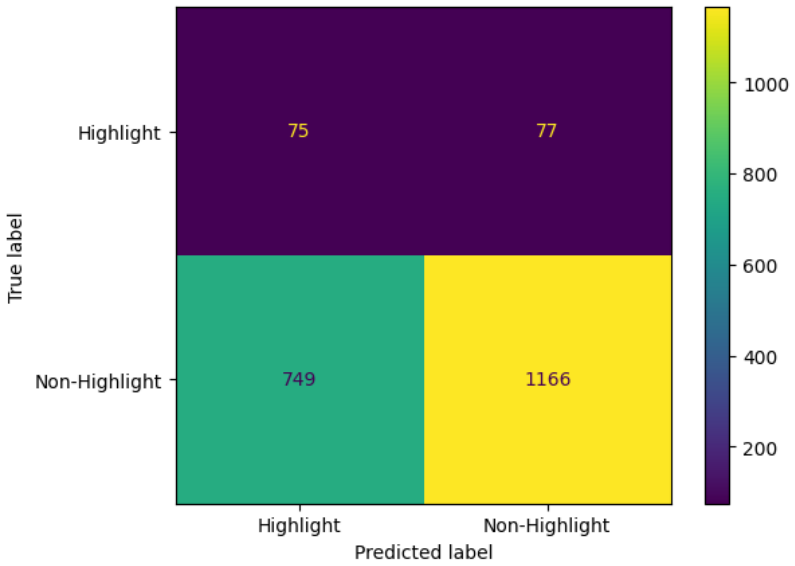


Figura 3: Matriz de confusão 1 - *Zero-Shot* (*Labels: Highlight e Non-Highlight)*

- **Accuracy: 60.04%**
- **Precision: 9.10%**

- **Recall: 49.34%**
- **F1-Score: 15.37%**

O modelo apresenta uma precisão baixa visto que apresenta um número alto de falsos positivos e, mesmo prevendo um número elevado de *Highlights*, o modelo apresenta um *recall* abaixo dos 50%, isto é, dos verdadeiros *highlights*, erra mais do que aqueles que acerta. Por consequência do valor da precisão ser baixo, o resultado do *F1-Score*, como expectável, também é baixo.

Passando para o modelo que utiliza na mesma 2 *labels* candidatas mas onde concentrámos as *labels* vindas do *SoccerNet* relativas a *Highlight* numa só *string/ label* e as relativas a *Non-Highlight* numa outra *string/ label* (separadas por vírgula). De realçar que, na matriz de confusão abaixo, aparecem como *labels* candidatas *Highlight* e *Non-Highlight* apenas para facilitar a visualização da matriz. Na realidade, as *labels* candidatas foram as *strings* com as *labels* do *SoccerNet*, como foi explicado anteriormente.

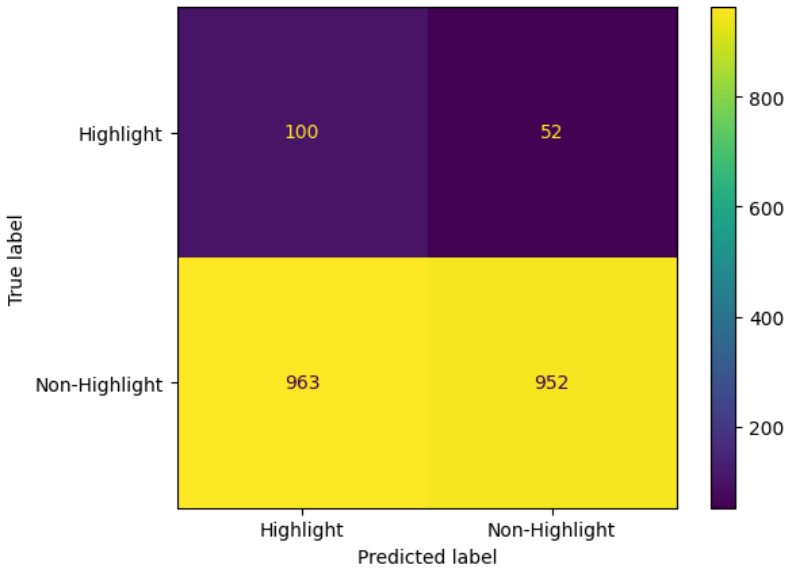


Figura 4: Matriz de confusão 2 - *Zero-Shot*

- **Accuracy: 50.09%**
- **Precision: 9.41%**
- **Recall: 65.79%**
- **F1-Score: 16.46%**

O modelo continua a apresentar uma precisão baixa visto que apresenta um número alto de falsos positivos mas, desta vez, apresenta um *recall* mais apelativo de cerca de 65%, isto é, o modelo foi capaz de acertar 65% dos verdadeiros *highlights*. No entanto, o maior problema do modelo anterior mantém-se que é o elevado número de falsos positivos. Por consequência do valor da precisão continuar a ser baixo, o resultado do *F1-Score*, como expectável, também é baixo.

Relativamente à abordagem com 6 *labels*, 3 relacionadas a *Highlight (Goal, Red card, Shots on target)* e outras 3 relacionadas a *Non-Highlight (Ball out of play, Clearance, Corner)*, os resultados foram os mostrados abaixo.

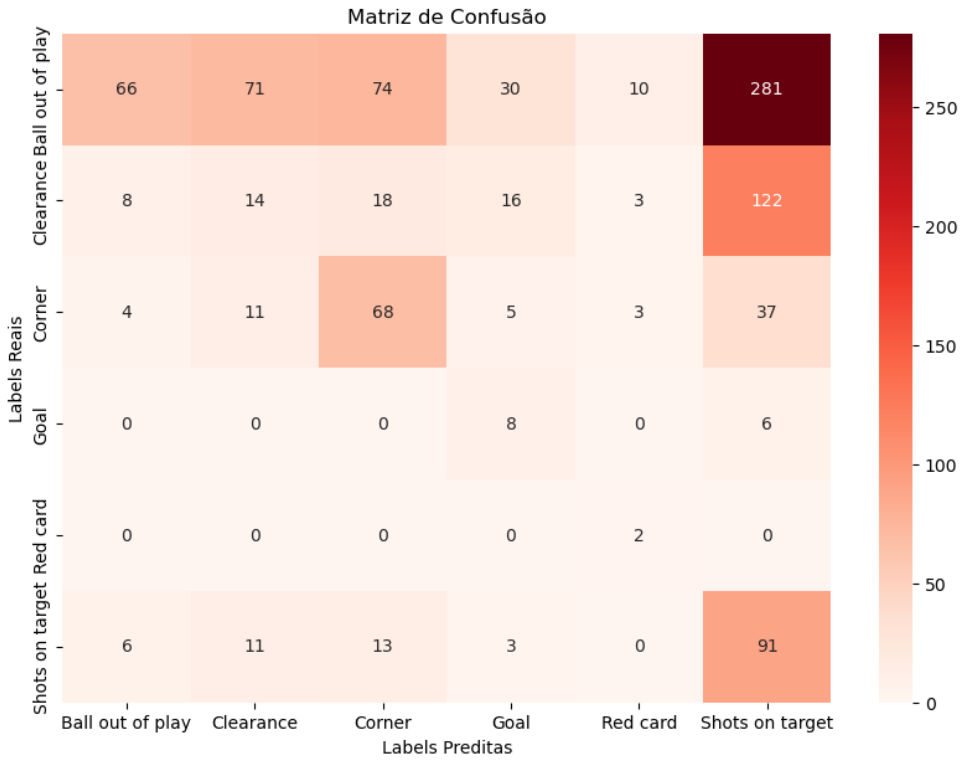


Figura 5: Matriz de confusão 3 - *Zero-Shot* (*Labels: Ball out of play, Clearance, Corner, Goal, Red card, Shots on target*)

- **Accuracy: 25.38%**
- **Precision: 16.37%**
- **Recall: 31.86%**
- **F1-Score: 21.63%**

Nota: As métricas *precision* e *recall* foram calculadas considerando como *labels* positivas – *"Shots on target, Goal, Red card"*; e como *labels* negativas – *"Ball out of play, Clearance, Corner"*. A título de exemplo, os *true positives* são o traço da submatriz 3x3 que representa as *labels* positivas e os *false positives* são a soma de todas as vezes que o modelo preveu *labels* positivas de forma errónea. O mesmo cenário acontece para as *labels* negativas.

Possivelmente devido ao número de *labels* ser maior, existe um decréscimo no valor da *accuracy* que já era expectável. Além disso, observa-se que o modelo prevê bastantes falsos *"Shots on target"* e como a maioria dessas previsões são relativas a *labels* de *Non-Highlight* e a *label "Shots on target"* é uma *label* de *Highlight*, torna-se um aspeto negativo do modelo. O mesmo acaba por se suceder com as *labels* relacionadas a *Highlight "Goal" e "Red card"* mas de forma menos evidente na matriz de confusão. No que toca às previsões de *labels Non-Highlight* o cenário descrito acima não acontece, embora existam algumas previsões falsas, a maioria permanece em *labels Non-Highlight* o que acaba por ser menos preocupante. No entanto, não nos pareceu ser a melhor abordagem pois, além da complexidade computacional ser maior devido ao aumento de *labels*, não houve ganho de desempenho.

Capítulo 5

Classificação Few-Shot

Tal como na classificação *Zero-Shot*, a classificação *Few-Shot* é uma tarefa processamento de linguagem natural onde é utilizado um modelo treinado num conjunto de exemplos rotulados. Porém, em vez de classificar observações de classes que o modelo nunca viu, é fornecida uma certa quantidade de exemplos rotulados dos novos dados ao modelo de forma a que ele seja capaz de utilizar estes dados limitados e generalizá-los efetivamente de forma a classificar os novos dados.

5.1 Aplicação do método

Na classificação *Few-Shot* foi aplicada uma técnica diferente, os modelos *SetFit*. Os modelos *SetFit* oferecem uma abordagem eficiente para o treino de modelos com poucos exemplos (*few-shot learning*), utilizando *sentence transformers*.

Vantagens dos modelos SetFit

1. **Abordagem sem prompt**: Ao contrário de métodos tradicionais que podem requerer prompts específicos para treino, os modelos *SetFit* eliminam essa necessidade. Isso simplifica o processo de configuração e utilização dos modelos.

2. **Modelos menores e mais rápidos**: *SetFit* utiliza modelos menores, o que reduz significativamente o tempo de treino. Esta eficiência é crucial quando se trabalha com grandes volumes de dados. Além disso, são necessários menos recursos computacionais, permitindo um desenvolvimento mais rápido e eficiente.

3. **Desempenho em ambientes de alta performance**: Devido ao seu tamanho reduzido e à eficiência do treino, os modelos *SetFit* são ideais para aplicações que exigem alta performance e

rapidez na inferência. Isso é particularmente vantajoso em cenários onde a aplicação de modelos em tempo real é crucial.

A estratégia abordada foi utilizar dois tipos de *labels* candidatas dadas ao modelo pré-treinado por nós escolhido, no caso, **all-MiniLM-L6-v2** – modelo utilizado no *few-shot classification* treinado com base num vasto conjunto de dados com mil milhões de pares de *sentences*.

Optámos por utilizar o *dataset* principal, que contém as *labels* binárias *Highlight* e *Non-Highlight*, para treino e teste deste modelo utilizando várias combinações dos mesmos para consequente avaliação dos resultados.

5.2 Avaliação dos resultados

5.2.1 Métricas utilizadas

Tal como na avaliação dos resultados do *Zero-Shot*, decidimos utilizar várias métricas para avaliar o resultado combinações de dados de treino e teste, sendo essas: **accuracy**, **precision**, **recall** e **F1-Score**.

Mais uma vez, além destas métricas, decidimos fazer um *plot* da matriz de confusão dos diferentes modelos para melhor percepção visual dos resultados.

5.2.2 Resultados

Inicialmente, começamos com um número relativamente pequeno de amostras, quase igualmente distribuídas entre ambas as classes, para treinar o modelo. Foram utilizadas 40 amostras para a classe *Highlight* e 38 amostras para a classe *Non-Highlight*.

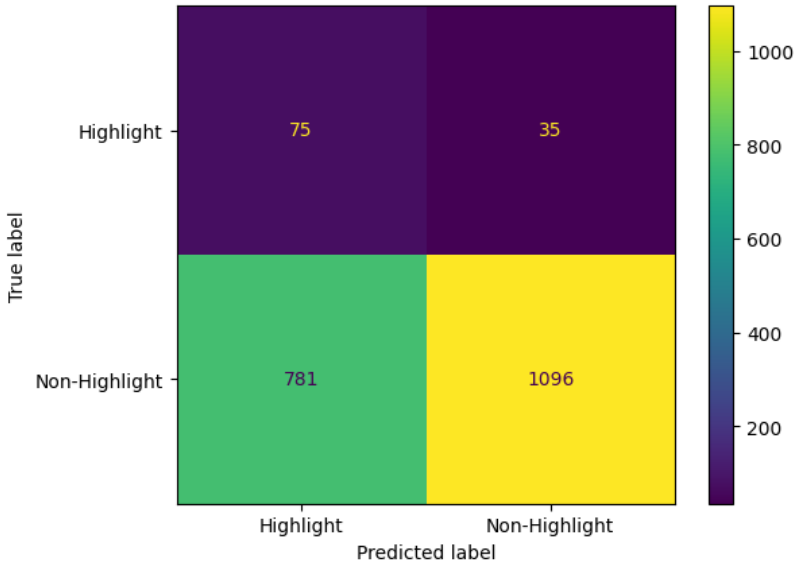


Figura 6: Matriz de confusão 1 - *Few-Shot*

	Precision	**Recall**	**F1-Score**	**Accuracy**
Highlight	8.76%	68.18%	15.5%	
Non-Highlight	96.9%	58.40%	72.90%	
Overall				58.93%

Tabela 1: Resultados das métricas de avaliação do modelo 1 - *Few-Shot*

O modelo apresenta uma precisão baixa para a classe *Highlight*, atingindo apenas 8.76%. Isso indica um número alto de falsos positivos, ou seja, muitas instâncias classificadas como *Highlight* que, na verdade, não pertencem a essa classe. No entanto, o *recall* é relativamente alto, com 68.18%, o que significa que, entre todos os verdadeiros *Highlights*, o modelo consegue identificar corretamente uma grande parte. No entanto, devido ao valor da precisão ser baixo, o resultado do *F1-Score*, como esperado, também é baixo, resultando em 15.5%.

Para a classe *Non-Highlight*, a precisão é muito alta, atingindo 96.9%. Isso indica que a maioria das previsões de *Non-Highlight* está correta. No entanto, o *recall* é de 58.40%, mostrando que o modelo falha em identificar uma parte significativa dos verdadeiros *Non-Highlights*, classificando-os incorretamente como *Highlight*. O *F1-Score* para esta classe é 72.90%, refletindo um bom equilíbrio entre precisão e *recall*. No entanto, aplicando mais as métricas para o problema em questão e visto que o que queremos

na verdade é detetar *Highlights*, os valores mais importantes a ter a conta são todas as métricas que consideram a *label Highlight* como positiva (primeira linha da tabela). A *accuracy* geral do modelo é 58,9%, indicando que o modelo está correto nas suas previsões num pouco mais de metade do número total de registos avaliados.

Na combinação seguinte, foram utilizadas 120 amostras para a classe *Highlight* e 150 amostras para a classe *Non-Highlight* para treino, com o objetivo de avaliar a performance do modelo ao aumentar o número de exemplos em ambas as classes.

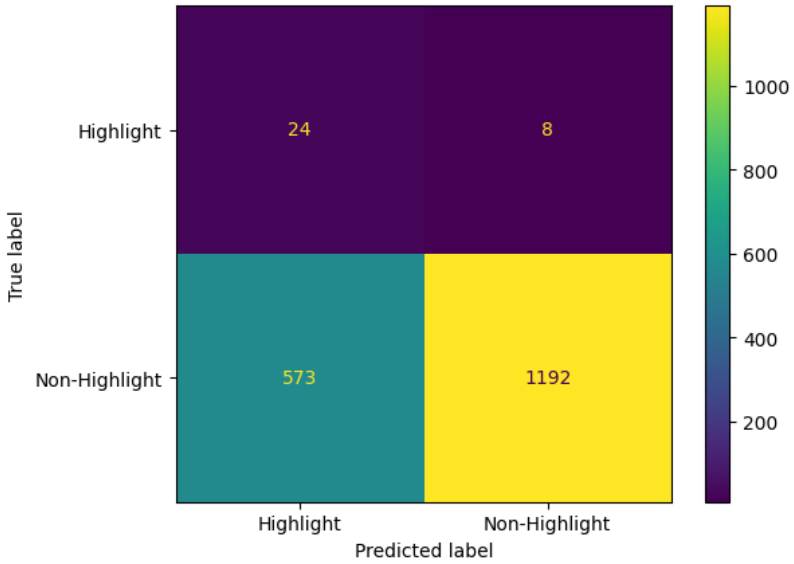


Figura 7: Matriz de confusão 2 - *Few-Shot*

	Precision	**Recall**	**F1-Score**	**Accuracy**
Highlight	4.02%	75.00%	7.66%	
Non-Highlight	99.33%	67.54%	80.48%	
Overall				67.67%

Tabela 2: Resultados das métricas de avaliação do modelo 2 - *Few-Shot*

O modelo apresenta uma precisão baixa para a classe *Highlight*, atingindo apenas 4,02%. Isso significa que há muitos falsos positivos, ou seja, várias instâncias classificadas como *Highlight* não pertencem realmente a essa classe. Por outro lado, o recall é alto, com 75%, indicando que o modelo consegue identificar a maioria dos *Highlights* reais. No entanto, devido à baixa precisão, o F1-Score também é baixo, resultando em 7,66%.

Para a classe *Non-Highlight*, a precisão é extremamente alta, chegando a 99,33%, o que significa que quase todas as previsões de *Non-Highlight* estão corretas. O recall, porém, é de 67,54%, mostrando que o modelo falha em identificar uma proporção significativa dos verdadeiros *Non-Highlights*, classificando alguns deles como *Highlight*. O F1-Score para esta classe é 80,48%, indicando um bom equilíbrio entre precisão e recall. A accuracy geral do modelo é 67,67%, o que indica que o modelo faz previsões corretas em pouco mais de dois terços dos casos avaliados.

Nesta combinação, decidimos continuar com uma boa quantidade de amostras para a classe *Highlight* e aumentar as amostras para a classe *Non-Highlight*, utilizando 100 amostras para *Highlight* e 300 amostras para *Non-Highlight* para treino.

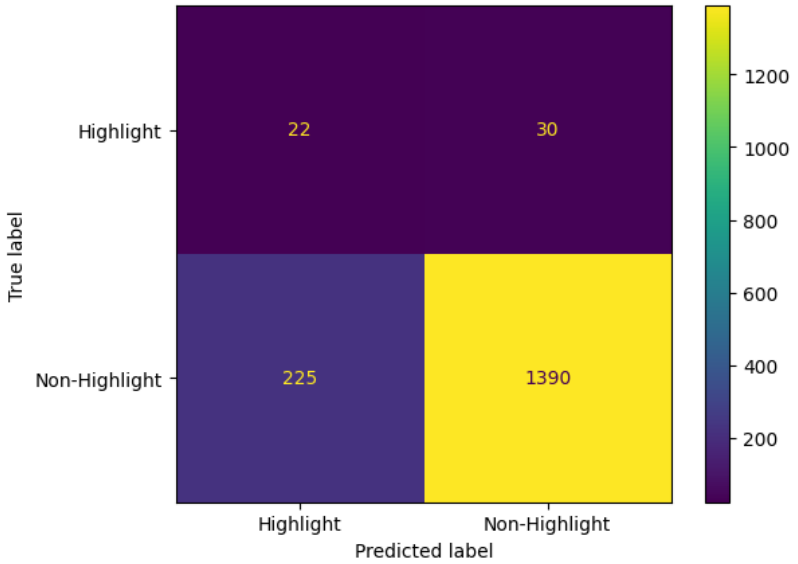


Figura 8: Matriz de confusão 3 - *Few-Shot*

	Precision	**Recall**	**F1-Score**	**Accuracy**
Highlight	8.90%	42.30%	14.71%	
Non-Highlight	97.90%%	86.00%	91.60%	
Overall				84.70%

Tabela 3: Resultados das métricas de avaliação do modelo 3 - *Few-Shot*

Apesar do modelo aumentar ligeiramente a precisão em relação às outras combinações para a classe *Highlight*, ele ainda apresenta uma precisão baixa, devido ao elevado número de falsos positivos. Desta

vez, o *recall* diminuiu drasticamente, correspondendo a 42,30%, ou seja, o modelo foi capaz de acertar apenas 42,30% dos verdadeiros *Highlights*. No entanto, o maior problema do modelo anterior permanece: o elevado número de falsos positivos. Como consequência, o valor do *F1-Score* é baixo.

Relativamente à classe *Non-Highlight*, a precisão permanece extremamente alta, chegando a 97,9%. No entanto, apresenta um *recall* mais apelativo comparado aos modelos anteriores, cerca de 86,00%, mostrando que o modelo identifica corretamente uma grande parte dos verdadeiros *Non-Highlights*. O F1-Score para esta classe é 91,60%, refletindo um bom equilíbrio entre precisão e recall. Por último, a *accuracy* aumentou, atingindo 84,70%, devido ao aumento do *recall* dos *Non-Highlight*.

Na última combinação, decidimos utilizar uma grande quantidade de amostras para a classe *Non-Highlight*, com 122 amostras para *Highlight* e 1532 amostras para *Non-Highlight*.

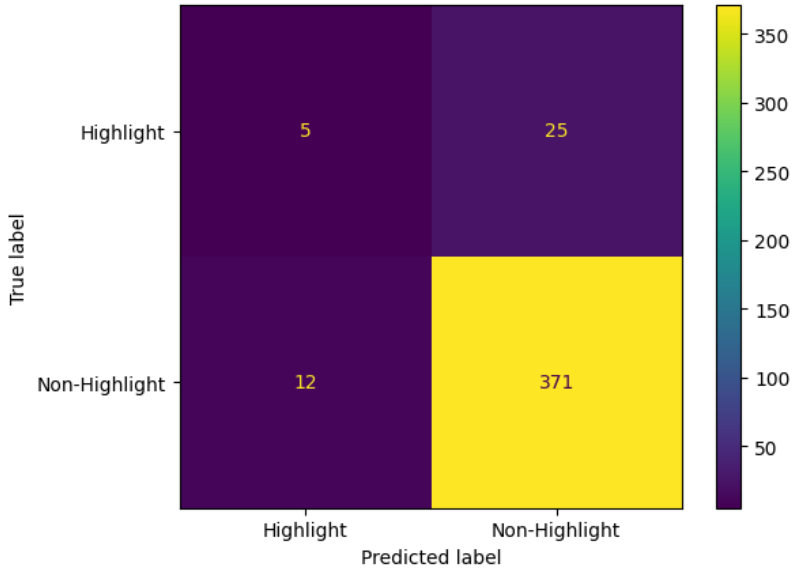


Figura 9: Matriz de confusão 4 - *Few-Shot*

	Precision	**Recall**	**F1-Score**	**Accuracy**
Highlight	29.41%	16.67%	21.28%	
Non-Highlight	93.70%	96.87%	95.26%	
Overall				91.04%

Tabela 4: Resultados das métricas de avaliação do modelo 4 - *Few-Shot*

Apesar do modelo aumentar a precisão em relação às outras combinações para a classe *Highlight*, ele ainda apresenta uma precisão relativamente baixa, de 29,41%. Desta vez, o *recall* é ainda mais baixo, correspondendo a 16,67%, ou seja, o modelo foi capaz de identificar apenas 16,67% dos verdadeiros *Highlights*. Embora continuem a existir falsos positivos, o valor baixo do *recall* espelha o aspeto mais negativo neste modelo que é a incapacidade de detetar verdadeiros *Highlights* de forma correta visto que mais de 80% são mal classificados.

Relativamente à classe *Non-Highlight*, a precisão permanece extremamente alta, chegando a 93,70%. Além disso, apresenta um *recall* muito alto, de 96,87%, o que é uma melhoria significativa em comparação com às tentativas anteriores. Isso mostra que o modelo identifica corretamente a grande maioria dos verdadeiros *Non-Highlights* devido às amostras de treino com essa *label* serem extremamente grandes. O F1-Score para esta classe é 95,26%, refletindo um excelente equilíbrio entre precisão e *recall*. Por último, a *accuracy* geral aumentou, atingindo 91,04%, devido ao aumento significativo do *recall* para a classe *Non-Highlight*. No entanto, não é o modelo indicado para o problema em questão visto que as métricas associadas à *label Highlight*, que é aquilo que queremos detetar, continuam insuficientes.

Capítulo 6

Fine-Tuning de um classificador

Na parte final do projeto, foi nos proposto realizar um *Fine-Tuning* de um classificador tradicional. Para isso utilizamos um modelo pré-treinado - **DistilBERT**, e adicionamos uma *classification head* no *output* do mesmo, de forma a conseguirmos obter um modelo mais ajustado aos novos dados.

6.1 Fine-Tuning

O *fine-tuning* é o processo de adaptação de um modelo pré-treinado para tarefas ou casos de uso específicos. Tornou-se uma técnica fundamental em *deep learning*, particularmente no processo de treino de modelos.

O *fine-tuning* pode ser considerado um subconjunto da técnica mais ampla de *transfer learning*: a técnica de aproveitar o conhecimento que um modelo existente já aprendeu como ponto de partida para aprender novas tarefas.

A intuição por trás do *fine-tuning* é que, essencialmente, é mais fácil e barato aprimorar as capacidades de um modelo base pré-treinado que já adquiriu conhecimentos amplos e relevantes para a tarefa em questão do que treinar um novo modelo a partir do zero para esse propósito específico. Isto é especialmente verdade para modelos de *deep learning* com milhões ou até mil milhões de parâmetros, como os grandes modelos de linguagem (LLMs) que se tornaram proeminentes no campo do *Natural Language Processing* (NLP) ou as *Convolutional Neural Networks* (CNNs).

Aproveitando o treino anterior do modelo através de *transfer learning*, o *fine-tuning* pode reduzir a quantidade de poder computacional caro e dados etiquetados necessários para obter grandes modelos adaptados a casos de uso e necessidades empresariais específicas. Por exemplo, o *fine-tuning* pode ser usado para simplesmente ajustar o tom conversacional de um LLM pré-treinado ou o estilo de ilustração de um modelo de geração de imagens pré-treinado; também pode ser usado para complementar os conhecimentos do conjunto de dados de treino original de um modelo com dados proprietários ou conhecimentos especializados e específicos de um domínio.

Assim, o *fine-tuning* desempenha um papel importante na aplicação real de modelos de aprendizagem automática, ajudando a democratizar o acesso a modelos e a personalização de modelos sofisticados.

6.2 Modelo BERT

O BERT (*Bidirectional Encoder Representations from Transformers*) é um modelo desenvolvido pela *Google* que revolucionou o campo de *Natural Language Processing* (NLP). Lançado em 2018, o BERT é projetado para entender o contexto das palavras duma forma bidirecional, o que significa que ele considera tanto as palavras à esquerda quanto à direita de uma palavra-alvo para compreendê-la em todo o seu contexto.

O modelo **Distilbert** é uma versão "destilada"do BERT que essencialmente é mais eficiente do ponto de vista computacional, não comprometendo o desempenho do modelo.

6.2.1 Modelo

Para realizar o treino do BERT, os desenvolvedores da *Google* optaram por duas vias: **Mask Language Model** e **Next Sentence Prediction**. Na 1ª a ideia seria dar uma frase ao BERT como *input* e colocar 15% das palavras como **masks**, ou seja, desconhecidas. Por exemplo, aplicando uma *mask* à frase "Estou a fazer o trabalho de reconhecimento de *highlights*", ficaríamos com algo deste tipo: "Estou a fazer o trabalho de [MASK] de highlights". O trabalho do BERT aqui seria prever qual a palavra que está *masked*.

Na 2ª técnica o objetivo seria prever qual a próxima frase que iria aparecer como *input*. Tomemos o seguinte exemplo: "Hoje vou a Guimarães com o meu grupo de trabalho da DTx". Seriam apresentadas ao modelo duas frases "Vamos apresentar o trabalho"e "Vamos ao *shopping*". Neste caso o BERT teria de prever qual a próxima frase a ser apresentada.

6.2.2 Tokenizer

Para conseguirmos inserir dados no *Distilbert*, o modelo exige que os dados estejam *encoded*. Para isso, utilizamos o DistilbertTokenizer. O *tokenizer* recebe como *input* uma frase e devolve duas listas, sendo a primeira uma lista de identificadores das palavras e a segunda uma lista com valores relativos à importância da palavra na frase, ao qual chamamos de *attention mechanism*.

6.3 Fine-tuning do BERT

Nesta secção vamos mostrar como conseguimos combinar as duas secções anteriores, de forma a obtermos um modelo mais ajustado aos nossos dados. Vamos também apresentar os resultados obtidos, bem como o histórico do treino dos diferentes modelos.

6.3.1 Arquitetura da rede

Na figura abaixo, podemos ver a forma como está montada a nossa rede neuronal.

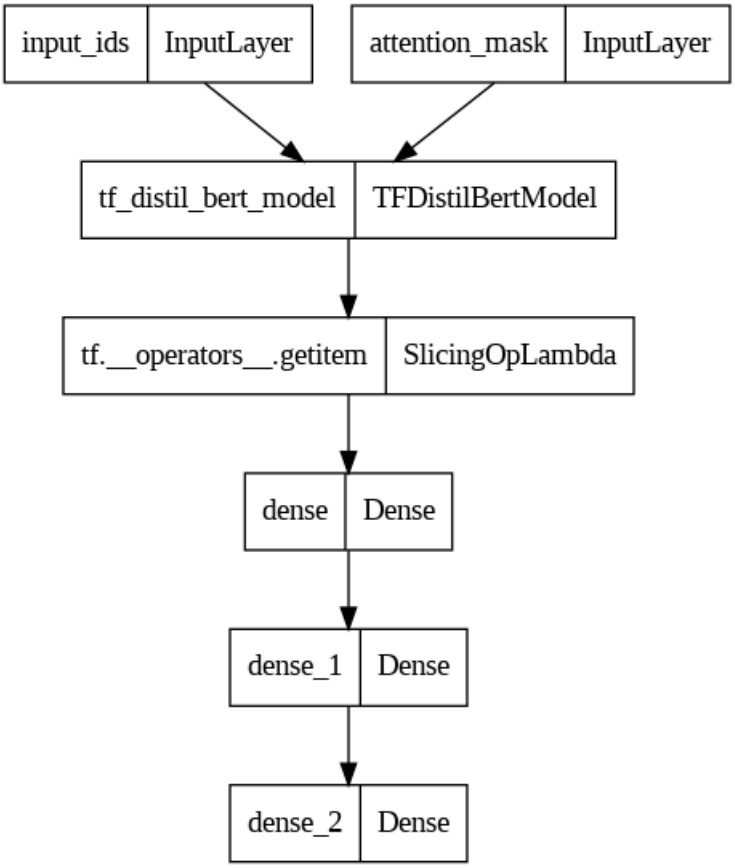


Figura 10: Arquitetura do modelo

Como podemos ver, a nossa rede tem duas *Input Layers*. A primeira *layer* corresponde à representação vetorial de cada palavra presente na frase passada como *input* enquanto que a segunda *layer* corresponde à *attention* atribuída a cada um destes *tokens*. De notar, que estes valores são obtidos através da aplicação do *tokenizer*, cujo comportamento foi explicado anteriormente.

Após os dados estarem devidamente preparados, passamo-los ao BERT e obtemos o seu *output* sem realizar a classificação do mesmo. Depois de obtermos o *output* do modelo, esse mesmo resultado é passado uma MLP composta por 3 camadas, onde vamos fazer um treino supervisionado. As primeiras duas camadas contêm 64 e 128 neurónios, respetivamente, enquanto que na terceira realizamos um *softmax* para obter o output desejado.

Abaixo podemos ver o código associado à construção da rede.

```
import tensorflow as tf
from transformers import DistilBertTokenizer, TFDistilBertModel
from sklearn.metrics import precision_score, recall_score
import numpy as np
from keras.utils import plot_model

# Create a custom model, Bert + MLP
def create_model(bert_model_name, num_classes):
    tokenizer = DistilBertTokenizer.from_pretrained(bert_model_name)
    bert_model = TFDistilBertModel.from_pretrained(bert_model_name)

    input_ids = tf.keras.Input(shape=(None,), dtype=tf.int64, name='input_ids')
    attention_mask = tf.keras.Input(shape=(None,), dtype=tf.int64,
    name='attention_mask')
    # Obter a saída do modelo transformer
    bert_output = bert_model([input_ids, attention_mask])
    sequence_output = bert_output.last_hidden_state
    # Selecionar a representação do token [CLS]
    cls_token = sequence_output[:, 0, :]
    # Adicionar camadas densas para classificação
    x = tf.keras.layers.Dense(64, activation='relu')(cls_token)
    x = tf.keras.layers.Dense(128, activation='relu')(x)
    output = tf.keras.layers.Dense(num_classes, activation='softmax')(x)

    model = tf.keras.Model(inputs=[input_ids, attention_mask], outputs=output)
    return model, tokenizer

model, tokenizer = create_model(bert_model_name='distilbert-base-uncased',
num_classes=2)

model.compile(
    optimizer='adam',
```

```
    loss=tf.keras.losses.categorical_crossentropy,
    metrics=['accuracy', tf.keras.metrics.Precision(), tf.keras.metrics.Recall()]
)
plot_model(custom_model)
```

6.3.2 Métricas utilizadas

Relativamente às métricas utilizadas neste problema, mais uma vez optamos por não olhar simplesmente à *accuracy*, por motivos óbvios. Olhando ao nosso *dataset*, temos cerca de 2067 registos dos quais 1915 são rotulados como *Non-Highlight* e 152 como *Highlight*.

Ora, olhando para este cenário vemos claramente que o *dataset* não está balanceado. Imaginando um cenário em que o modelo classifica todos os eventos como *Non-Highlight*, a sua **accuracy** seria aproximadamente 93%. Com este simples exemplo percebemos de imediato que apenas considerar a *accuracy* seria um erro.

As métricas utilizadas para avaliar resultados foram então a **accuracy**, **precision** e **recall**.

6.4 Treino do modelo

Nesta secção vamos mostrar como treinamos o nosso modelo, com as diferentes configurações testadas e vamos também apresentar e interpretar os resultados obtidos.

6.4.1 Encoding dos dados

Como dito na secção anterior, a nossa rede recebe como input os dados *encoded*. Portanto, definimos uma função que recebe os nossos dados, ou seja, os comentários em formato de texto e transforma-os em duas listas. A primeira referente à *tokenização* das palavras e a segunda referente à *attention* aplicada a cada palavra.

No código abaixo podemos ver a função *encode_data* responsável por este processo.

```
# Encode Bert output to pass as input to our MLP
def encode_data(data, tokenizer, max_length=512):
    encodings = tokenizer(list(data), truncation=True, padding=True,
    max_length=max_length)

    input_ids = np.array(encodings['input_ids'])
    attention_masks = np.array(encodings['attention_mask'])

    return input_ids, attention_masks
```

6.4.2 Modelo base

O primeiro modelo testado não envolveu grande cuidado no que toca a tratamento de dados ou otimização de hiperparâmetros. O código abaixo mostra o processo de treino do modelo.

```
x_train, y_train, x_test, y_test = load_data()

# Load input ids and attention masks
train_input_ids, train_attention_masks = encode_data(x_train, tokenizer)
test_input_ids, test_attention_masks = encode_data(x_test, tokenizer)

epochs = 10
batch_size = 64
# Train the model
history = custom_model.fit(
    x=[train_input_ids, train_attention_masks],
    y=y_train,
    validation_split=0.2,
    epochs=epochs,
    batch_size=batch_size
```

)

Neste modelo, treinamos durante 10 *epochs* com *batch_size* igual a 64 e utilizámos 20% dos dados para validação.

O resultado do treino é mostrado na figura abaixo:

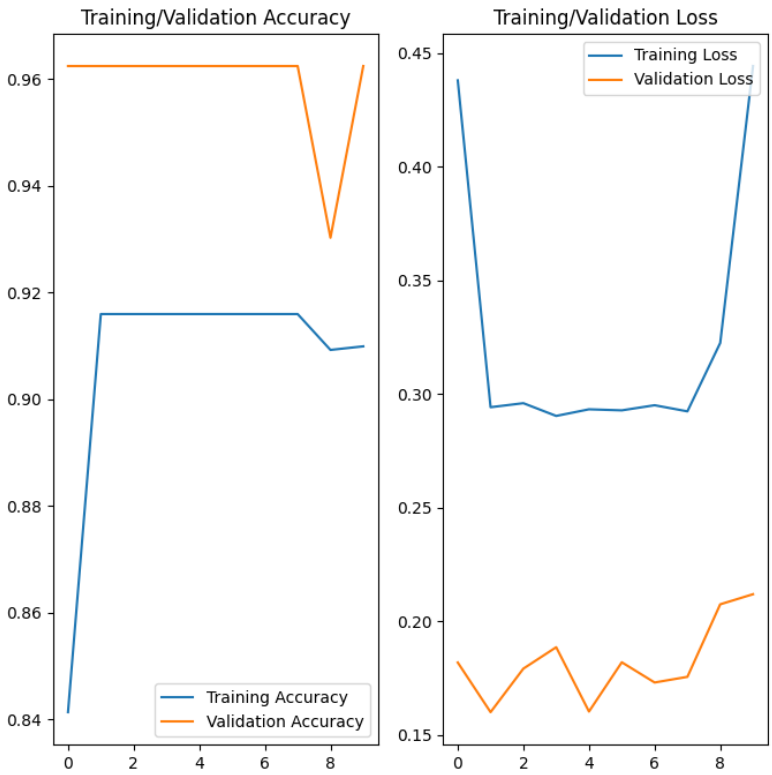


Figura 11: Histórico do treino do modelo base

Como podemos ver na figura, a *accuracy* do modelo foi bastante satisfatória. Vamos então olhar às *predictions* feitas para os dados de teste.

```
y_pred
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

y_test
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Figura 12: Previsões realizadas pelo modelo

Olhando à figura acima, vemos que embora que a *accuracy* seja bastante elevada, os resultados não são o esperado. Isto porque o modelo simplesmente está a prever que todo o input é um *Non-Highlight*. Analisando a probabilidade obtida após a realização do **softmax**, conseguimos perceber que o modelo estava a prever que uma frase é um *Non-Highlight* com uma probabilidade de **90%**.

Estes resultados levaram-nos a concluir que o problema poderia estar no balanceamento dos dados, portanto testamos uma nova abordagem, desta vez influenciando o resultado da previsão com **pesos de classes**.

6.4.3 Modelo com peso de classes

O conceito de atribuir pesos às diferentes classes do nosso problema é apresentado como forma de resolver o problema de balanceamento do *dataset*. A ideia subjacente a este conceito é atribuir um peso às diferentes classes do nosso problema, de modo a que estes valores tenham influência na importância que é dada às diferentes classes.

A fórmula utilizada para o cálculo do peso das classes é a seguinte:

$$peso_i = \frac{total}{2 \times total_i}, \qquad i = highlight, non\_highlight$$

Para o primeiro exemplo considerado, o peso da classe *Non-Highlight* foi aproximadamente **0.5** e o peso da classe *Highlight* foi aproximadamente de **6.8**.

Em termos de implementação, não tivemos grandes problemas como se pode ver no código abaixo:

```
total = counts[0] + counts[1]
weight_for_non_highlight = (1. / counts[0]) * (total / 2.0)
weight_for_highlight = (1. / counts[1]) * (total / 2.0)

class_weights = {
    0: weight_for_non_highlight,
    1: weight_for_highlight
}

# Train the model
history = custom_model.fit(
    x=[train_input_ids, train_attention_masks],
    y=y_train,
    validation_split=0.2,
    epochs=epochs,
    batch_size=batch_size,
    class_weight=class_weights
)
```

Os resultados obtidos foram os seguintes:

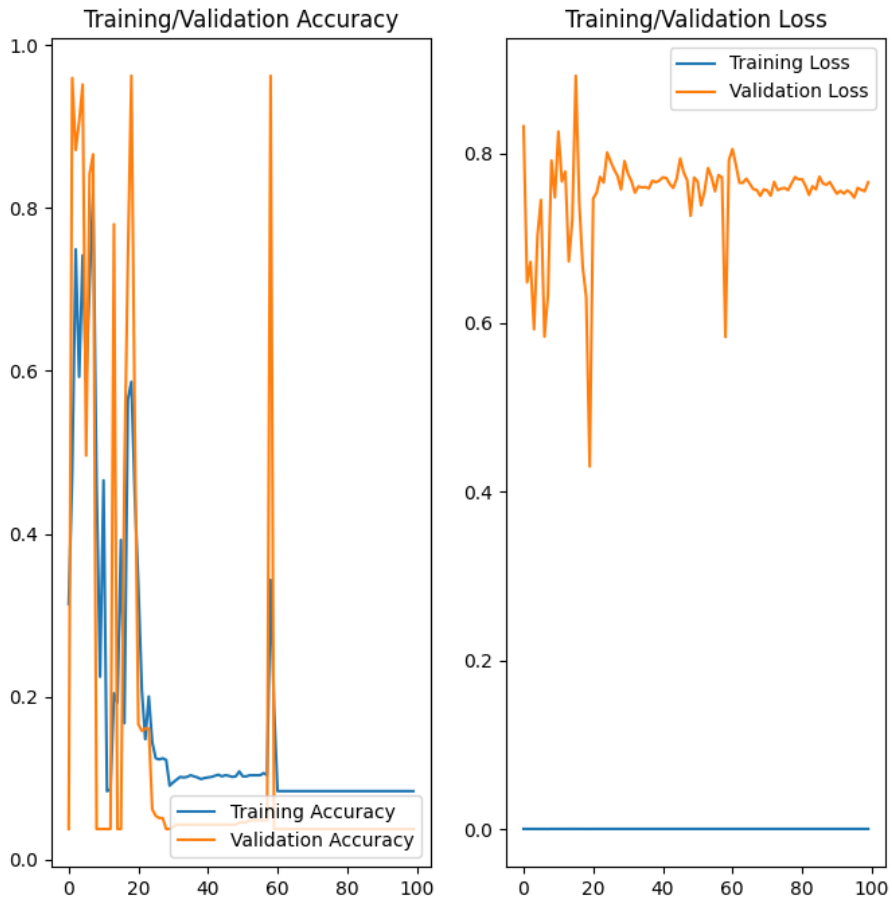


Figura 13: Histórico do treino do modelo com peso de classes

Desta vez optámos por realizar um treino mais extensivo, por isso treinamos durante 100 épocas como se pode ver na figura. Mais uma vez, os resultados não foram os mais encorajadores. Olhemos às *predictions* feitas pelo modelo.

```
y_pred
 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

y_test
 [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Figura 14: Previsões do modelo com peso de classes

Como se pode verificar pela figura, o modelo passou a prever todos os eventos como *Highlight*. Através da nossa análise das previsões percebemos que o resultado é dado com uma confiança de cerca de **51%**.

Capítulo 7

Conclusões e trabalho futuro

Este projeto tinha como objetivo principal treinar e utilizar modelos de *deep learning* para classificação de texto para detetar momentos de destaque em transmissões TV de jogos de futebol. Para atingir este objetivo foram realizadas diversas tarefas (indicadas no 1º capítulo), todas desafiantes à sua maneira.

Consideramos que os resultados obtidos através dos classificadores zero-shot e few-shot foram bons, tendo em conta que estes eram modelos pré-treinados para tarefas diferentes da pretendida para este projeto.

Relativamente ao fine-tuning de um classificador tradicional, acreditamos ter realizado um trabalho competente no seu desenvolvimento, porém os resultados ficaram um pouco aquém das nossas expectativas prevendo todos os eventos como sendo da mesma classe, apesar dos nossos esforços para que tal não acontecesse.

Do ponto de vista prático, o projeto permitiu-nos aplicar e desenvolver os conhecimentos adquiridos ao longo do mestrado e obter competências diversas, tais como a utilização de *ffmpeg* em *Python* para extração e fragmentação do audio dos vídeos dos jogos de futebol, a implementação de modelos pré-treinados para transcrição de texto e para tarefas de processamento de linguagem natural como a classificação *Zero-Shot* e classificação *Few-Shot* e, possivelmente a competência que advém da tarefa mais desafiante deste projeto, a capacidade de realização de *fine-tuning*, tarefa que conjugou os nossos conhecimentos de deep learning e de programação.

Acreditamos veemente que este projeto foi uma mais valia para todos nós, desenvolvendo-nos como pessoas e tornando-nos profissionais mais competentes.

7.1 Perspetiva de trabalho futuro

Em termos de trabalho futuro, seria interessante trabalhar com a totalidade da base de dados, os 505 jogos. Poderíamos também aplicar um modelo diferente na tarefa de transcrição de forma a tentar obter transcrições de qualidade superior, as quais poderiam impactar de forma positiva os resultados finais. Porém o maior objetivo para um projeto futuro seria aprofundar o *fine-tuning* realizado de forma a conseguir obter resultados de qualidade que permitissem a aplicação do modelo no mundo real.

Bibliografia

DTx. *Laboratório Colaborativo em Transformação Digital*. Apresentação, 2024. URL https://www.dtx-colab.pt/dtx-apresentacao/#1554389262917-beb0e9d5-3d3a.

HuggingFace. *Whisper documentation*. a. URL https://huggingface.co/openai/whisper-large.

HuggingFace. *bart-large-mnli documentation*. b. URL https://huggingface.co/facebook/bart-large-mnli.

Karl Kroening. *ffmpeg-python documentation*. 2017. URL https://kkroening.github.io/ffmpeg-python/.