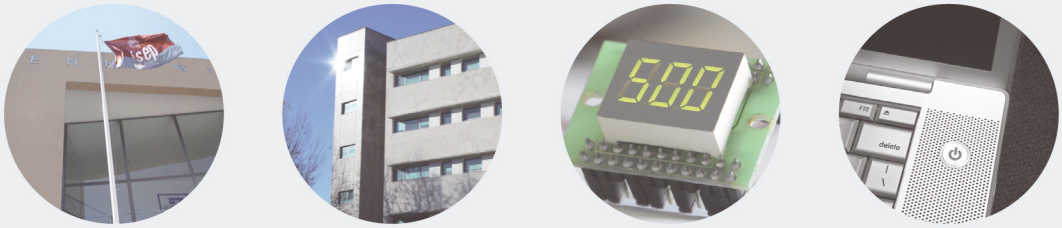
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Utilização de Inteligência Artificial no Futebol: Recolha de dados por Web scraping e previsão de resultados na Liga Portuguesa

NUNO FILIPE LIMA MARINHO
novembro de 2024

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Utilização de Inteligência Artificial no Futebol: Recolha de dados por *Web Scraping* e previsão de resultados na Liga Portuguesa

Nuno Filipe Lima Marinho

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização em Automação e Sistemas

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

Novembro, 2024

Esta dissertação satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores, Área de Especialização em Automação e Sistemas.

Candidato: Nuno Filipe Lima Marinho, Nº 1190915, 1190915@isep.ipp.pt

Orientação Científica: Veríssimo Manuel Brandão Lima Santos (PhD), vms@isep.ipp.pt

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

Novembro, 2024

Agradecimentos

Com a conclusão desta Dissertação, não posso deixar de agradecer a algumas pessoas que, direta ou indiretamente, me ajudaram nesta caminhada. Agradeço em particular ao Prof. Veríssimo Santos, pela orientação e apoio prestado durante todo este trabalho e a todos os docentes e colegas que compartilharam este percurso comigo.

Um agradecimento especial aos meus pais pelo constante suporto emocional e financeiro, que durante todo este meu percurso académico me proporcionaram.

Resumo

Nos últimos anos, o avanço tecnológico tem impulsionado significativamente a transformação de diversas indústrias. Um dos pilares desse progresso é o uso crescente de algoritmos de previsão baseados em *Machine Learning*. Esses algoritmos têm sido aplicados numa ampla gama de setores, desde finanças, saúde, *marketing*, desporto, passando pelas diversas áreas da engenharia, o que tem impulsionando a capacidade das organizações anteciparem eventos futuros e tomarem decisões estratégicas fundamentadamente.

O *Machine Learning*, como método de análise de dados, permite a automatização de construção de modelos analíticos. Sendo um ramo da inteligência artificial que se baseia na capacidade dos sistemas de aprenderem através dos dados, permitindo identificar padrões e tomar decisões com uma reduzida intervenção humana.

Esta dissertação, propõe-se a explorar detalhadamente o desenvolvimento de um algoritmo de previsão de resultados de futebol da Liga Portuguesa, baseado em *Machine Learning*. Para tal, serão estudados vários algoritmos com o objetivo de fornecer uma compreensão mais aprofundada dos métodos utilizados nesta área de investigação.

Palavras-Chave: Algoritmos de previsão, *Machine Learning*, Análise de dados, Modelos analíticos, Inteligência artificial, Identificação de padrões, Tomada de decisões.

Abstract

In recent years, technological advances have significantly driven the transformation of several industries. One of the pillars of this progress is the increasing use of predictive algorithms based on Machine Learning. These algorithms have been applied in a wide range of sectors, from finance and health to marketing and sports and also in different areas of engineering, boosting organizations' ability to anticipate future events and make informed strategic decisions. Machine Learning, as a method of analyzing data and automating the construction of analytical models, is a branch of artificial intelligence based on the ability of systems to learn from data, identify patterns and make decisions with little human intervention. This dissertation aims to explore in detail the development of an algorithm for predicting Portuguese league soccer results, based on Machine Learning. To this end, various algorithms will be studied with the aim of providing a more in-depth understanding of the methods used in this area of research.

Keywords: Predictive algorithms, Machine Learning, Analyzing data, Analytical models, Artificial intelligence, Identifying patterns, Making decisions.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Glossário

Explainability

Conjunto de processos e métodos que permitem aos utilizadores compreenderem e confiarem nos resultados criados pelos algoritmos de *Machine Learning*.

F1 score

Métrica de avaliação de *Machine Learning* que combina precisão e *recall*.

Recall

Métrica que mede a frequência com que um modelo de *Machine Learning* identifica corretamente verdadeiros positivos, de todas as amostras positivas reais no conjunto de dados.

Centróide

Ponto ou coordenada de uma forma geométrica que estabelece o seu centro geométrico.

Exatidão

Métrica que mede a frequência com que um modelo de *Machine Learning* prevê corretamente o resultado.

Precisão

Métrica que mede a frequência com que um modelo de *Machine Learning* prevê corretamente a classe positiva.

Lista de Acrónimos

AFL	*Australian Football League*
ANN	*Artificial Neural Network*
CNN	*Convolutional Neural Network*
CSV	*Comma Separated Values*
DT	*Decision Trees*
EPL	*English Premier League*
FIFA	*Fédération Internationale de Football Association*
GBDT	*Gradient-Boosted Decision Trees*
IA	Inteligência Artificial
ICA	*Independent Component Analysis*
JSON	*JavaScript Object Notation*
K-Means	*K-Means Clustering*
KNN	*K-Nearest Neighbors*
LOGREG	*Logistic Regression*
MEEC	Mestrado em Engenharia Electrotécnica e Computadores
NB	*Naive Bayes*
NCAAB	*National Collegiate Athletic Association Basketball*
NN	*Neural Network*
NRL	*Australian National Rugby League*
PCA	*Principal Component Analysis*
RF	*Random Forest*
SFS	*Sequential Feature Selector*

SQL *Structured Query Language*

SVM *Support Vector Machine*

TEDI Tese / Dissertação

XGBoost *Extreme Gradient Boosting*

Capítulo 1

Introdução

Realizado no âmbito da Unidade Curricular de Tese / Dissertação (TEDI) do 2º ano do Mestrado em Engenharia Electrotécnica e Computadores (MEEC) no ramo de Automação e Sistemas, esta dissertação aborda toda a pesquisa e trabalho desenvolvidos no projeto realizado neste âmbito. Neste capítulo é feita uma breve contextualização do tema em estudo e são abordados todos os objetivos e tarefas a realizar, sendo também apresentada a calendarização e a estrutura da dissertação.

1.1 Contextualização

O futebol, como desporto mais jogado e assistido mundialmente, exerce uma influência inigualável, cativando cerca de 4,5 biliões de pessoas em todo o mundo, o que equivale a aproximadamente dois terços da população global [1]. Esta paixão pelo futebol transcende fronteiras culturais e geográficas, tornando-o não apenas um jogo, mas uma parte intrínseca da identidade e da vida quotidiana de milhões de pessoas. Com a crescente popularidade do futebol, a procura por ferramentas de previsão de resultados tem aumentado exponencialmente. Não se trata apenas de uma busca por entretenimento para os fãs, mas também de uma necessidade profissional para equipas técnicas, jornalistas desportivos e até mesmo apostadores que buscam *insights* para orientar as suas decisões [2].

A previsão de resultados no futebol é um desafio fascinante, amplamente reconhecido pela sua complexidade. Os resultados de uma partida dependem de uma

infinidade de fatores, incluindo o estado de ânimo de uma equipa ou jogador, competências individuais, lesões, estratégias de treino e muito mais. Este cenário multifacetado torna a previsão de resultados uma tarefa exigente e, ao mesmo tempo, emocionante.

Para enfrentar esse desafio, a aplicação de algoritmos baseados em *Machine Learning* emerge como uma solução promissora. O *Machine Learning*, sendo um método de análise de dados, oferece a capacidade de automatizar a construção de modelos analíticos. Este ramo da inteligência artificial fundamenta-se na capacidade dos sistemas de aprender com dados históricos, identificar padrões complexos e, consequentemente, tomar decisões com uma intervenção humana mínima.

1.2 Definição do Problema

A previsão de resultados de jogos de futebol é uma tarefa complexa com grande potencial de aplicação. Este projeto visa desenvolver um algoritmo que consiga prever com precisão os resultados de jogos de futebol da Liga Portuguesa, o que pode trazer diversos benefícios para diversos públicos. O algoritmo deve ser capaz de analisar dados históricos de jogos, estatísticas de equipas, e outras informações relevantes para fazer previsões precisas do resultado de um jogo.

1.2.1 Objetivos

Como principais objetivos do projeto, o algoritmo deve ser capaz de prever os resultados de jogos de futebol da primeira liga de futebol portuguesa. O algoritmo deve ser capaz de prever os seguintes resultados: vitória da equipa da casa, vitória da equipa visitante ou empate.

1.3 Calendarização

Nesta secção é apresentada a calendarização, Tabela 1.1, correspondente às várias tarefas realizadas para a conceção deste trabalho. Como referido, no decorrer deste trabalho foi efetuada uma pesquisa dos conceitos teóricos sobre a temática do *Machine Learning*, tendo sido também pesquisados trabalhos existentes na mesma área cientifica, mais direcionados para o desporto, em particular para o futebol.

Desta pesquisa resultou o capítulo do estado de arte. De seguida, foram recolhidos por *web scraping* os dados de fontes na internet, desenvolvido o algoritmo de *Machine Learning* aplicado aos mesmos, constituindo este o projeto desenvolvido. Por fim, surge a elaboração da dissertação, a qual abrange todo o estudo e trabalho desenvolvidos, bem como os respetivos resultados e conclusões. Os períodos de duração de cada tarefa estão evidenciados na tabela.

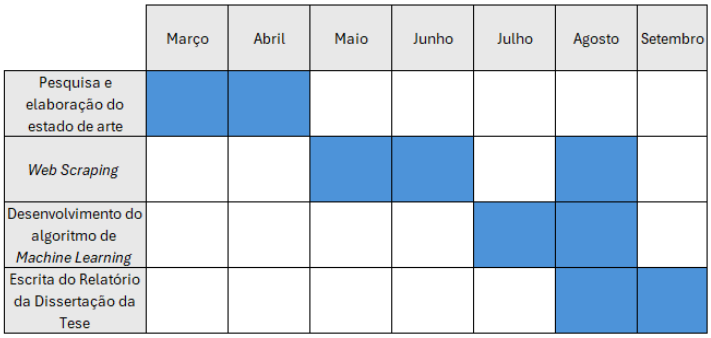


Figura 1.1: Calendarização

1.4 Organização da Dissertação

No capítulo 1 é feita a contextualização, são apresentados os objetivos deste trabalho, a sua calendarização e a organização desta dissertação. O capítulo seguinte, o 2, é dedicado ao Estado da arte e à apresentação dos fundamentos teóricos por detrás deste projeto, sendo também apresentadas algumas das soluções existentes nesta área. No 3.º Capítulo, é feita uma descrição do projeto. O Capítulo 4 está reservado para a explanação da parte de código do projeto e para a apresentação das opções tomadas. No Capítulo 5 são apresentados e discutidos os resultados obtidos para os diversos algoritmos utilizados. Finalmente, no Capítulo 6, são apresentadas as conclusões resultantes deste trabalho e tecidas as perspetivas de desenvolvimentos futuros.

Capítulo 2

Estado da Arte

Neste capítulo são apresentados conceitos-chave sobre este projeto, sendo para tal realizada uma introdução ao futebol, o qual é o caso de estudo descrito nesta dissertação, é feita uma introdução à Inteligência Artificial (IA) e ao *Machine Learning* e por fim, a apresentação de alguns dos algoritmos mais utilizados em *Machine Learning*.

2.1 Futebol

O Futebol é considerado o desporto mais popular do mundo, com cerca de 4,5 biliões de apoiantes em todo o mundo [1]. Os chineses foram os primeiros a chutar uma bola para uma baliza no século III a.C, conhecido como cuju, era bastante diferente do futebol moderno. Inicialmente era utilizado como treino militar competitivo, que visava a aprimorar as habilidades físicas dos cavaleiros e também uma forma de entretenimento nas cortes reais e cidades ricas. Foi na Mesoamérica, a vasta região histórica que se estende do México à Costa Rica, que várias civilizações jogavam um desporto que envolvia uma bola pesada feita de uma substância derivada de resina de árvores. A origem do jogo é incerta, mas era popular entre culturas mesoamericanas como osteotihuacanos, astecas e maias desde há cerca de três mil anos. O nome do desporto variava entre culturas, ullamaliztli em asteca pok-ta-pok ou pitz em maia e as regras também se alteravam.

O desporto atualmente conhecido como futebol nasceu nas escolas britânicas. Embora algumas variantes do jogo já fossem jogadas informalmente há séculos, o

desporto foi formalizado em Inglaterra no século XIX. Na década de 1840, algumas escolas britânicas criaram as suas próprias regras de jogo, o que possibilitou a organização de torneios entre jogadores que conheciam as mesmas regras. Dois conjuntos de regras diferentes começaram a dominar o desporto. O *Sheffield Football Club* oferecia às equipas um pontapé de compensação caso o adversário desobedecesse às regras do jogo, e o Cambridge University proibia os jogadores de transportarem a bola com as mãos. À medida que a popularidade do desporto aumentou, os jogadores juntaram-se, formando a *London Football Association*. Em 1904, o desporto era tão popular que se tornara internacional. A *Fédération Internationale de Football Association* (FIFA) foi criada nesse mesmo ano. Após a estreia do desporto nos Jogos Olímpicos de 1908 e do primeiro campeonato mundial da FIFA em 1930, o futebol profissional disparou [3].

Atualmente, a análise de dados no futebol desempenha um papel crucial na melhoria do desempenho das equipas. Com o avanço das tecnologias e o crescente acesso a informações detalhadas sobre cada aspeto do jogo, treinadores, analistas e gestores desportivos podem tomar decisões mais assertadas e estratégicas. Os dados recolhidos no futebol são de vários tipos. Existem dados sobre estatísticas básicas de classificações e resultados da liga, dados sobre detalhes e características dos jogadores, informações sobre os seus pontos fortes e fracos, qualidades de ataque, velocidade, resistência, qualidades defensivas, ritmo de trabalho, golos marcados, assistências, equipas pelas quais jogaram, dados do histórico salarial, entre outros. São analisados também dados sobre as estatísticas básicas do jogo como jogadores, assistência, golos marcados, cartões, substituições, dados pormenorizados, normalmente numa base fotograma, dos jogadores de ambas as equipas, as suas localizações, localizações da bola, detalhes da ação, como desarmes e passes, a sua direção e, em alguns casos, nível de intensidade ou velocidade. Também são analisados dados sobre a saúde e o desempenho em situações de jogo e de treino, para medir e compreender a sua prontidão para o jogo e pormenorizar as áreas de melhoria.

A análise de dados no futebol desempenha um papel crucial no aprimoramento do desempenho das equipas e dos jogadores por várias razões. Primeiramente, contribui significativamente para a melhoria do desempenho ao identificar forças e fraquezas tanto das equipas quanto dos jogadores, permitindo a elaboração de estratégias mais eficazes e direcionadas. Além disso, a saúde dos jogadores é monitorizada de maneira contínua por via de dados que avaliam a condição física e mental, otimizando o desempenho em campo e prevenindo possíveis lesões. A análise de dados também é fundamental no *scouting* de talentos, pois auxilia na avaliação de futuros jogadores e na verificação da adequação deles às necessidades específicas da equipa. Por último, mas não menos importante, contribui para a eficiência operacional do clube, abrangendo desde a gestão financeira até à compreensão da demografia dos fãs, passando pela implementação de estratégias de *marketing* mais precisas. Dessa forma,

a integração da análise de dados tornou-se indispensável para qualquer clube que visa competir nos níveis mais elevados do cenário futebolístico moderno [4].

2.2 Inteligência Artificial

Existem fortes ligações entre o desenvolvimento dos computadores e o surgimento da Inteligência Artificial (IA). No entanto, as sementes da IA foram plantadas muito antes do desenvolvimento dos computadores modernos. As raízes imediatas mais fortes, provavelmente remontam ao trabalho de McCulloch e Pitts, que, em 1943, os quais descreveram modelos matemáticos (chamados perceptrons) de neurónios no cérebro (células cerebrais) com base numa análise detalhada dos originais biológicos. Eles não apenas indicaram como os neurónios disparam ou não (estão "ligados" ou "desligados"), operando assim de maneira binária comutada, mas também mostraram como tais neurónios poderiam aprender e, portanto, mudar a sua ação em relação ao tempo [5].

O termo inteligência artificial é um conceito com várias definições possíveis, consoante o contexto, a época e as aplicações. Uma definição bastante geral é "a inteligência demonstrada pelas máquinas, em contraste com a inteligência natural demonstrada pelos seres humanos e outros animais" [6]. Desde a avaliação de conjuntos de dados bastante grandes em tempo quase real, carros de condução autónoma e recomendações de vídeos influenciadas pelo histórico de transmissões, até recomendações de compras, anúncios e deteção de fraudes, a IA tornou-se fundamentalmente enraizada em muitas facetas da sociedade e funciona frequentemente de forma invisível no fundo dos nossos dispositivos eletrónicos pessoais [7]. A figura 2.1 é uma ilustração das várias camadas da Inteligência Artificial.

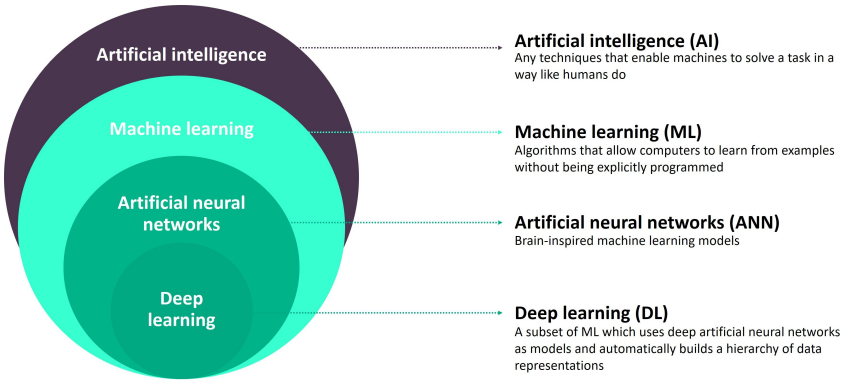


Figura 2.1: Camadas da inteligência artificial [8]

2.3 *Machine Learning*

Machine Learning é um ramo em evolução dos algoritmos computacionais concebidos para imitar a inteligência humana, no qual estes aprendem com dados originários do ambiente circundante. Baseia-se em diferentes algoritmos para resolver problemas de dados. Os cientistas de dados gostam de salientar que não existe um tipo de algoritmo único que seja o melhor para resolver um problema. O tipo de algoritmo utilizado depende do tipo de problema que se pretende resolver, do número de variáveis, do tipo de modelo que melhor se adequa ao problema, etc. [9].

A programação tradicional é caracterizada por um processo manual, no qual os programadores não só desenvolvem a lógica do programa, mas também determinam manualmente as regras e o código necessário para a sua execução. Neste paradigma, os dados de entrada são processados por um programa desenvolvido pelo programador, resultando na saída desejada. No caso do *Machine Learning*, há uma inversão no processo tradicional de programação (Figura 2.2). Aqui, os dados e as saídas/respostas são utilizados como entrada, enquanto as regras de aprendizagem são produzidas como saída. Esse paradigma é particularmente importante, por possibilitar que o computador aprenda novas regras em ambientes complexos e avançados, os quais muitas vezes são desafiadores para a compreensão humana [10].

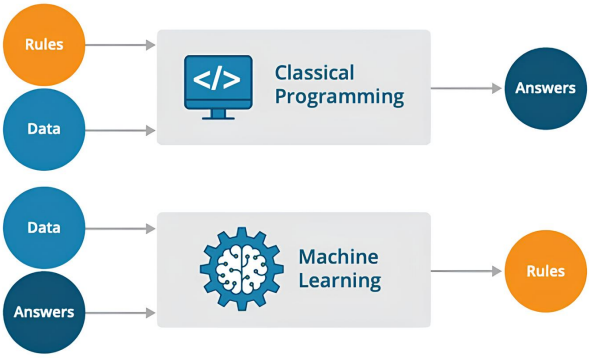


Figura 2.2: Programação clássica vs *Machine Learning* [11]

2.3.1 *Supervised Learning*

A *Supervised Learning*, é um método de *Machine Learning* que utiliza dados rotulados para prever ou classificar resultados. O objetivo é criar um modelo que possa fazer previsões precisas sobre novos dados, com base em aprendizagens de um conjunto de dados de treino. Envolve o uso de um conjunto de dados de entrada (*features*) e saída desejada (*target*) para treinar um algoritmo. O modelo aprende a mapear as entradas para as saídas corretas e, após o treino, pode prever a saída para novas entradas [12].

No domínio do *Supervised Learning*, existe uma ampla gama de algoritmos incluindo, *Support Vector Machine* (SVM), *Artificial Neural Network* (ANN), *Logistic Regression* (LOGREG), *Random Forest* (RF), e *Decision Trees* (DT). Nas próximas subsecções é apresentada uma análise de cada algoritmo, destacando-se as suas características distintivas e explorando as suas respetivas aplicações.

2.3.1.1 *Support Vector Machine*

Support Vector Machine (SVM) é uma das técnicas clássicas de *Machine Learning* que ajuda a resolver problemas de classificação de grandes volumes de dados (Figura 2.3). No entanto, o SVM é matematicamente complexo e computacionalmente dispendioso. O SVM recebe um conjunto de dados como entrada, e prevê, para cada entrada, qual das duas possíveis classes a entrada faz parte, o que faz do SVM um classificador linear binário não probabilístico [13].

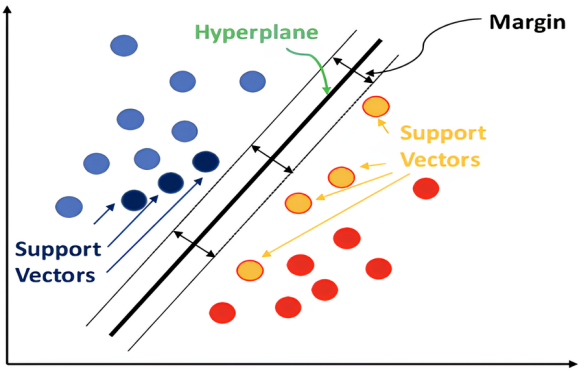


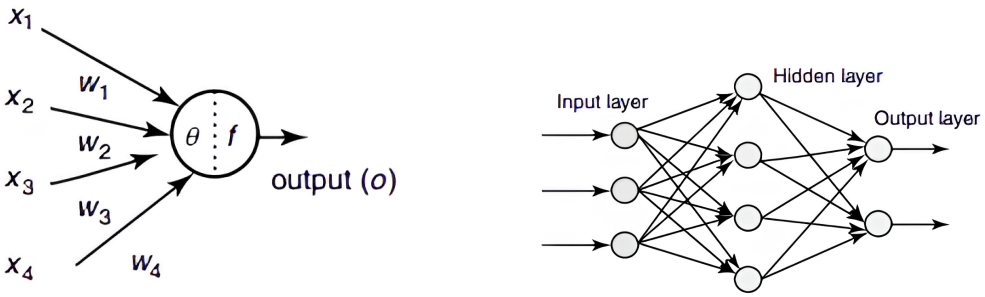
Figura 2.3: Ilustração esquemática do modelo *Support Vector Machine* [14]

2.3.1.2 *Artificial Neural Network*

As *Artificial Neural Network* (ANN) são um subconjunto de *Neural Network* (NN), especificamente concebidas para imitar o comportamento do cérebro humano. O cérebro humano é composto por um conjunto de mais de 10 mil milhões de neurónios interligados e é a prova da existência de redes neuronais maciças que podem ser bem sucedidas nas tarefas cognitivas, percetivas e de controlo, em que os humanos são tão bem sucedidos. O cérebro é capaz de realizar atos percetivos (por exemplo, reconhecimento de rostos, discurso) e atividades de controlo (por exemplo, movimentos e funções corporais) [15].

A arquitetura básica das ANN consiste em três tipos de camadas de neurónios: de entrada, ocultas e de saída. Nas redes de alimentação, o fluxo de sinal vai das unidades de entrada para as unidades de saída, estritamente numa direção de

alimentação. A Figura 2.4 ilustra um neurónio artificial típico e a modelação de uma rede neuronal com várias camadas.



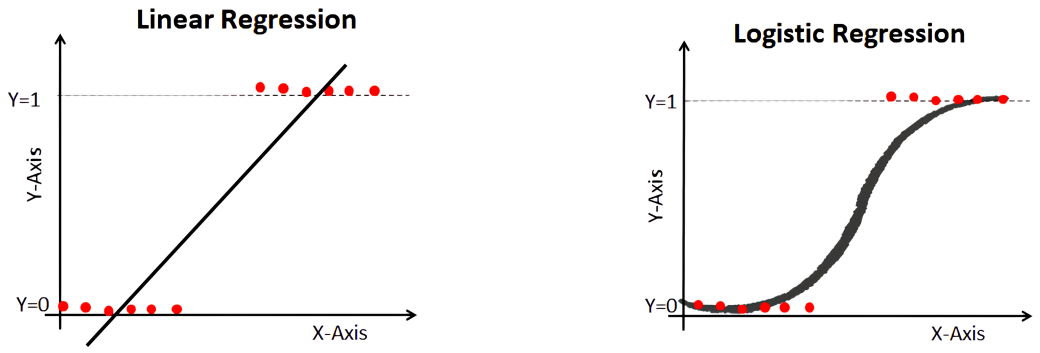
(a) (b)

Figura 2.4: (a) Arquitetura de um neurónio artificial e (b) rede neuronal artificial multicamadas [15]

2.3.1.3 *Logistic Regression*

A *Logistic Regression* (LOGREG) é uma ferramenta de previsão que avalia a relação entre a variável dependente categórica e uma ou mais variáveis independentes, estimando probabilidades por meio de uma função logística. A regressão logística pode ser de natureza binomial ou multinomial. Na regressão logística binomial ou binária, o resultado está restrito a dois possíveis valores (por exemplo, "Sim" ou "Não"). Já na regressão logística multinomial, o resultado pode assumir três ou mais valores distintos (por exemplo, "Baixo", "Médio", "Alto").

Normalmente, os resultados na LOGREG são representados como "0" e "1". Enquanto numa regressão linear ajustamos uma linha reta às nossas observações (Figura 2.5a), na regressão logística ajustamos uma curva em forma de "S" chamada função sigmoide (Figura 2.5b). Esta função sigmoide tem como limite superior o valor 1 e como limite inferior o valor 0 [16].



(a) (b)

Figura 2.5: (a) Regressão linear vs (b) Regressão logística [17]

2.3.1.4 *Decision Trees*

As *Decision Trees* são uma técnica popular de *Machine Learning* para descobrir padrões a partir de dados existentes. Uma *Decision Trees* é um método baseado numa árvore, em que cada caminho a partir da raiz, representa uma sequência de divisão de dados até se chegar a um resultado booleano no nó folha. Na prática, cada caminho da *Decision Trees* é uma regra de decisão que pode ser facilmente traduzida em linguagem humana ou de programação [18]. As *Decision Trees*, começam a sua construção com uma pergunta inicial simples, e prosseguem com uma série de questões subsequentes relacionadas entre si. Estas questões são representadas pelos nós da *Decision Trees*, funcionando como meios de dividir os dados em grupos mais homogéneos. As observações que satisfazem os critérios estabelecidos seguem o ramo "Sim", enquanto aquelas que não se enquadram prosseguem pelo caminho alternativo (Figura 2.6). O objetivo das *Decision Trees* é encontrar as divisões mais eficientes para segmentar os dados.

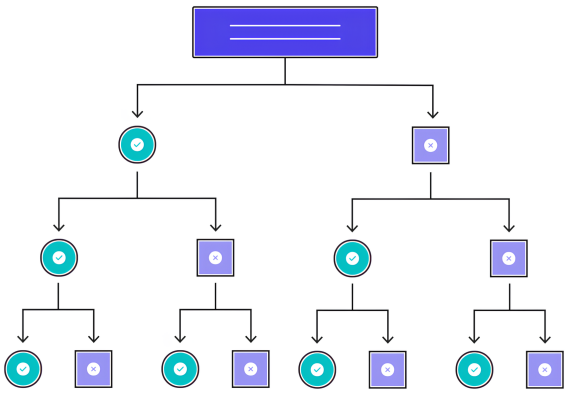


Figura 2.6: Ilustração esquemática de uma *Decision Trees* [19]

2.3.1.5 *Random Forest*

A *Random Forest* é uma das técnicas de *Machine Learning* mais bem sucedida, que se tem revelado muito popular e poderosa no reconhecimento de padrões e, na classificação de conjuntos de dados de elevada dimensão e com problemas de distorção [20]. Devido à sua precisão e robustez superiores, bem como à sua capacidade de fornecer informações através da classificação das suas características, o *Random Forest* tem sido efetivamente aplicado a várias aplicações de *Machine Learning*. O *Random Forest* consiste num conjunto de *Decision Trees*, formando uma "floresta" de classificadores, que votam numa determinada classe (Figura 2.7). Para treinar um *Random Forest*, é necessário fornecer dois parâmetros, o número de árvores na floresta e o número de características selecionadas aleatoriamente para avaliar em cada árvore, bem como uma base de dados de treino com rótulos de classe verdadeiros [21].

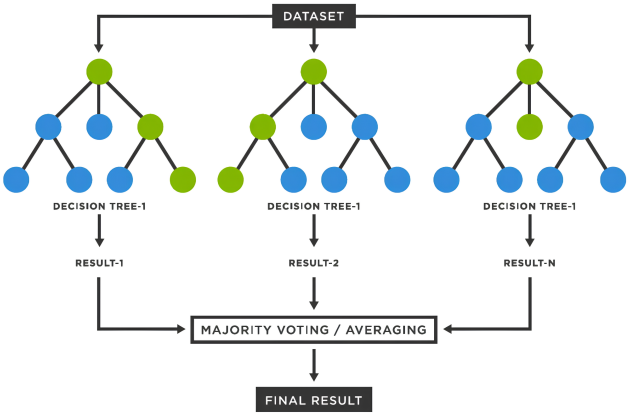


Figura 2.7: Ilustração esquemática do modelo *Random Forest* [22]

2.3.1.6 *Extreme Gradient Boosting*

O *Extreme Gradient Boosting* (XGBoost) é um algoritmo baseado numa *gradient boosting tree*, que pode desempenhar um papel poderoso na melhoria do gradiente, sendo um método muito eficaz para problemas de regressão e classificação. O XGBoost simboliza um algoritmo mais otimizado e avançado do algoritmo *Gradient-Boosted Decision Trees* (GBDT) [23].

O XGBoost tem as vantagens da computação paralela, tal como o *Random Forest*, da utilização otimizada da memória e do tratamento eficiente de dados esparsos. Tende a ter uma melhor precisão do que os modelos lineares ao fazer previsões, mas não tem a capacidade de *Explainability* dos modelos lineares [24]. Na Figura 2.8 é possível ver um esquema do modelo *Extreme Gradient Boosting*.

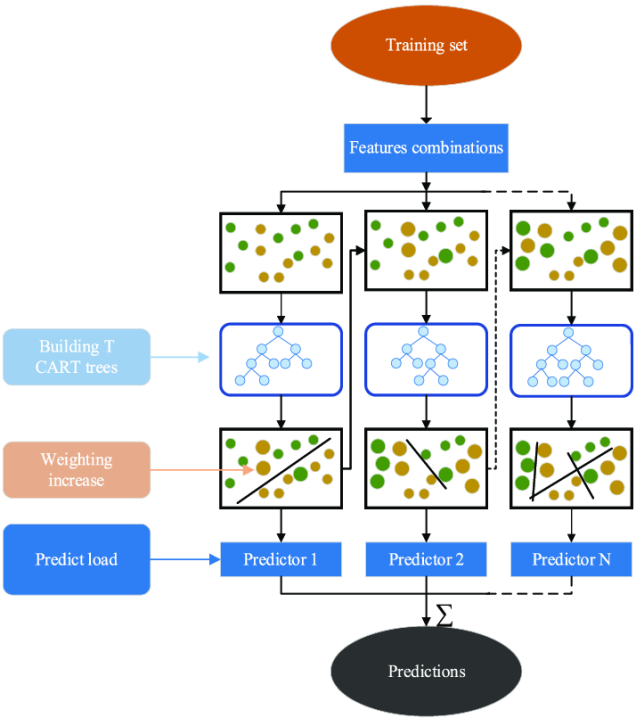


Figura 2.8: Ilustração esquemática do modelo XGBoost [25]

2.3.2 *Unsupervised Learning*

Unsupervised Learning, é um tipo de *Machine Learning* onde os algoritmos analisam e agrupam conjuntos de dados não rotulados. Estes algoritmos descobrem padrões ocultos sem a necessidade de intervenção humana. Diferente do *supervised learning*, que trabalha com dados rotulados, o *unsupervised learning* opera com dados sem rótulos pré-definidos. O modelo visa identificar estruturas e padrões interessantes nos dados, por conta própria [12].

No campo do *Unsupervised Learning*, alguns dos algoritmos mais comuns são, o *K-Means Clustering* (K-Means), o *Hierarchical Clustering*, o *Principal Component Analysis* (PCA), o *Independent Component Analysis* (ICA) e o *Autoencoders*. Cada um destes algoritmos possui características distintas que os tornam adequados para diferentes tipos de problemas e cenários de aplicação, os quais vão ser apresentados nas subsecções seguintes.

2.3.2.1 *K-Means Clustering*

Clustering é uma forma de classificar razoavelmente os dados brutos e de procurar os padrões ocultos que possam existir nos conjuntos de dados. É um processo de agrupamento de dados em *clusters* desarticulados, de modo que os dados do mesmo *cluster* sejam semelhantes, mas os dados pertencentes a *clusters* diferentes sejam

diferentes [26]. O *K-Means Clustering* (K-Means) é um dos algoritmos mais popular para o *clustering*. É um método iterativo, numérico, não supervisionado e não determinístico. É simples e muito rápido, pelo que, em muitas aplicações práticas, o método provou ser uma forma muito eficaz de produzir bons resultados de agrupamento, sendo muito adequado para a produção de aglomerados globulares. Ele agrupa os dados em K *clusters*, onde K é um parâmetro especificado pelo utilizador. Utilizando a noção de Centróide, o K-Means atribui iterativamente pontos de dados ao *cluster* cujo Centróide é o mais próximo, refinando esses Centróide até que a convergência seja alcançada (Figura 2.9). Este algoritmo é frequentemente utilizado em segmentação de mercado, análise de imagem e agrupamento de documentos [27].

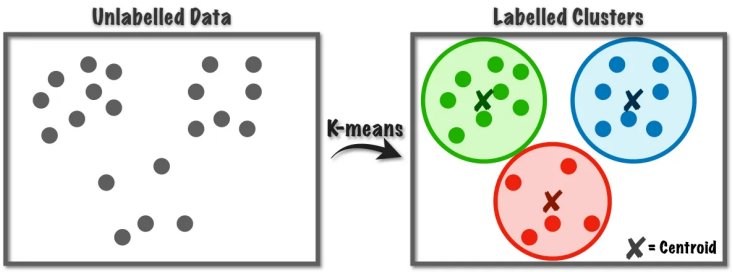
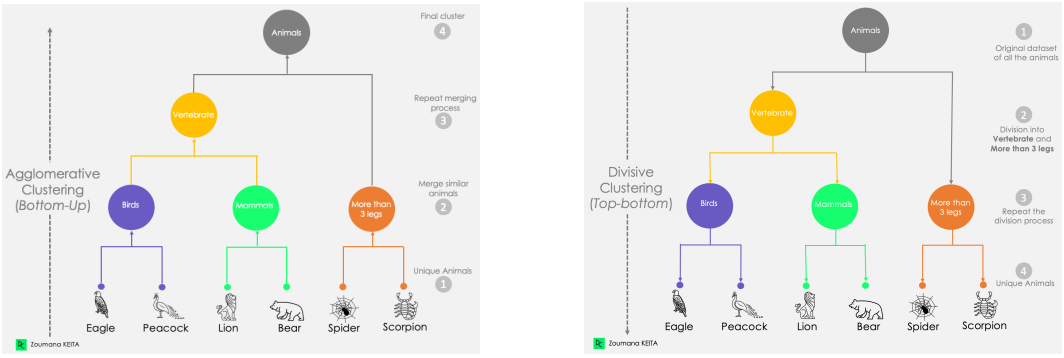


Figura 2.9: Exemplo de uma classificação realizada com *K-Means Clustering* [28]

2.3.2.2 *Hierarchical Clustering*

Hierarchical Clustering é um método comum utilizado para determinar grupos de pontos de dados semelhantes em espaços multidimensionais [29]. Existem duas abordagens principais para o *Hierarchical Clustering*. São elas a abordagem descendente (divisivo) e a abordagem ascendente (aglomerativo). No agrupamento aglomerativo, cada ponto de dados é inicialmente tratado como um *cluster* individual, a cada iteração os dois *clusters* mais similares são fundidos, o que resulta num grande número de camadas hierárquicas (Figura 2.10a). O método de agrupamento divisivo, ao contrário do método aglomerativo, começa com todos os pontos num único *cluster*, em seguida, ele divide recursivamente esse *cluster* em subgrupos menores com base na dissimilaridade dos pontos (Figura 2.10b), tal como faz o k-Means hierárquico, que divide cada nó em K nós. Isto requer um conhecimento prévio do valor de K, que representa o número de *clusters* [30].



(a) (b)

Figura 2.10: (a) Agrupamento hierárquico vs (b) Agrupamento divisivo [31]

2.3.2.3 *Principal Component Analysis*

A *Principal Component Analysis* (PCA) é uma técnica popular para a redução da dimensionalidade, e tem sido amplamente utilizada em muitos domínios. Permite definir um espaço de dimensões reduzidas a qual preserva a informação relevante dos dados originais, permite a visualização de objetos e variáveis. A PCA requer dados multivariados, ou seja, muitas variáveis medidas em muitos objetos [32]. Esta técnica projeta os dados num novo espaço de características, onde as dimensões são ordenadas segundo a variância dos dados (Figura 2.11). Isso permite identificar os principais padrões e tendências nos dados, facilitando a visualização e a compreensão da sua estrutura subjacente. A PCA é aplicado, em diversas áreas, como reconhecimento de padrões, compressão de imagens e análise de séries temporais.

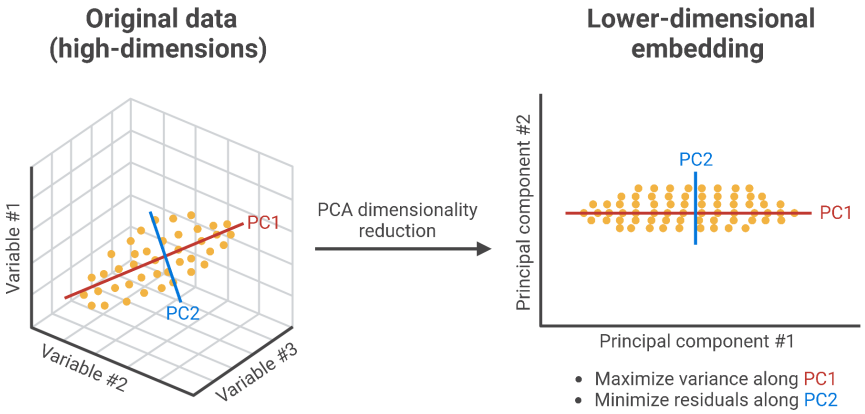


Figura 2.11: Exemplo de uma classificação realizada com *Principal Component Analysis* [33]

2.3.2.4 *Independent Component Analysis*

O *Independent Component Analysis* (ICA) é um método probabilístico que aprende uma transformação linear de um vetor aleatório. O objetivo é encontrar componentes que sejam independentes e não-Gaussianos. A diferença fundamental do ICA para métodos estatísticos multivariados clássicos, reside na suposição de não-Gaussianidade, o que permite identificar os componentes originais subjacentes, ao contrário dos métodos clássicos. O ICA é considerado um método não supervisionado, exploratório e orientado por dados, útil para investigar estruturas de dados quando hipóteses adequadas não estão disponíveis ou são consideradas muito restritivas, ou simplistas [34]. O ICA é um método de decomposição de sinais, semelhante ao PCA, mas que visa identificar componentes que sejam estatisticamente independentes um do outro. É especialmente útil quando os dados são uma mistura de sinais provenientes de diferentes fontes. Na Figura 2.12 é possível ver uma aplicação do método ICA.

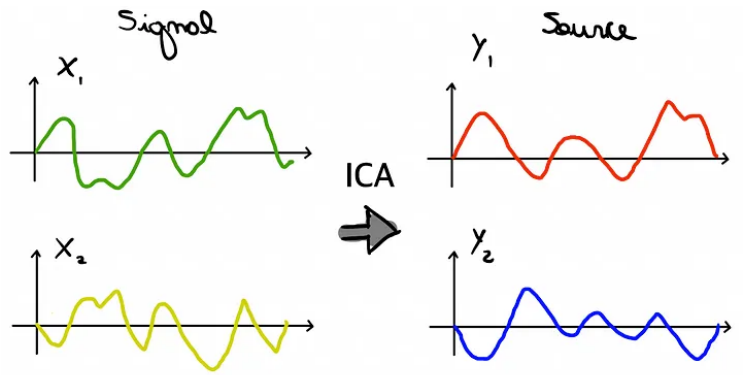


Figura 2.12: Exemplo de utilização de *Independent Component Analysis* [35]

2.3.2.5 *Autoencoders*

Sendo os *Autoencoders* um algoritmo de *Artificial Neural Network* baseado na *Unsupervised Learning*, visa reconstruir os seus vetores de entrada. Desde a sua invenção, os *Autoencoders* têm sido utilizados numa série de aplicações, incluindo a deteção de anomalias [36]. A principal diferença entre os *Autoencoders* e a *Principal Component Analysis* é que, enquanto a PCA encontra as direções ao longo das quais é possível projetar os dados com a máxima variância, os *Autoencoders* reconstroem a entrada original, apenas com uma versão comprimida da mesma. *Autoencoders*, por definição, reduzem as dimensões dos dados ao aprender a ignorar o ruído nos dados. Os *Autoencoders* podem ser divididos em quatro partes principais. Primeiro, temos o *Encoder*, onde o modelo aprende a reduzir as dimensões de entrada e a comprimir os dados. Em seguida, temos o *Bottleneck*, que é a camada que contém essa representação comprimida dos dados de entrada. Depois, vem o *Decoder*, onde

o modelo aprende a reconstruir os dados a partir da representação codificada. Por fim, há a *Reconstruction Loss*, que é o método usado para avaliar o desempenho do descodificador e medir a proximidade da saída em relação à entrada original [37]. Na Figura 2.13 é possível ver uma aplicação do método *Autoencoders*.

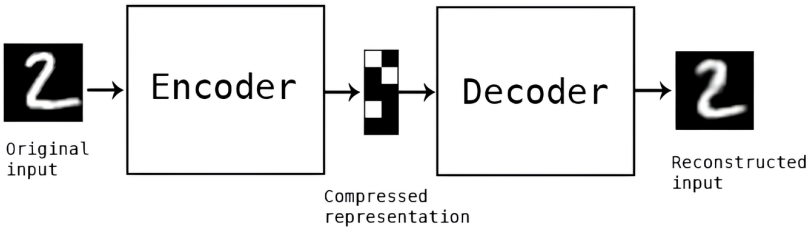


Figura 2.13: Exemplo de utilização de *Autoencoders* [38]

2.4 *Web Scraping*

Web scraping refere-se ao procedimento de extração automática de dados de *website* por meio de *software* [39]. Este método tem duas vertentes: o *crawler* e o *scraper*. O *crawler*, e um algoritmo de inteligência artificial, que segue as hiperligações da Internet para procurar dados específicos na Web. O *scraper* extrai dados do *website* e a sua criação afeta a precisão e a rapidez do seu funcionamento [40]. Um *Web scraping* engloba uma grande variedade de técnicas e tecnologias de programação, como a análise de dados, a análise de linguagem natural e a segurança da informação, e envolve normalmente a escrita de um *script* que acede a uma página *web*, analisa o seu conteúdo e extrai os dados relevantes [41].

Na Figura 2.14, pode-se observar um esquema exemplo do processo de *Web scraping*. Esse processo inicia com a identificação do *website* alvo, seguida pela recolha dos URLs das páginas das quais os dados são extraídos. Em seguida, realiza-se uma *request* para o URL de modo a obter o HTML da página. Após isso, procede-se à identificação dos dados de interesse no HTML, finalizando com o armazenamento desses dados no formato desejado, como *Comma Separated Values* (CSV), bases de dados *Structured Query Language* (SQL), *JavaScript Object Notation* (JSON), entre outros.

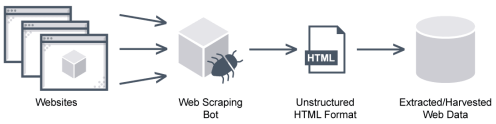


Figura 2.14: Esquema do processo de *Web scraping* [40]

Python é uma das linguagens de programação mais populares para *Web scraping* devido à sua simplicidade e variedade de bibliotecas. As bibliotecas mais utilizadas

são a *BeautifulSoup*, a *Scrappy* e a *Selenium*. A *BeautifulSoup* é uma biblioteca focada na análise de HTML, que se destaca pela sua memória média e baixo uso de CPU, o que a torna uma escolha eficiente para tarefas de *Web scraping* que priorizam a velocidade e a eficiência de recursos. A *Scrapy*, é uma poderosa biblioteca que oferece uma utilização moderada da memória e da CPU, o que resulta num tempo de trabalho equilibrado. Fornece uma interface de alto nível para lidar com tarefas de *scraping* complexas, e é particularmente útil para projetos que requerem funcionalidades avançadas ou o suporte para execução de *JavaScript* [42]. A *Selenium* é uma poderosa ferramenta para tarefas de *Web scraping*. Ao contrário de outras bibliotecas mais leves, o *Selenium* interage diretamente com o navegador, permitindo executar *scripts* em tempo real e manipular elementos complexos das páginas. Essa capacidade torna o *Selenium* ideal para projetos onde a execução de *JavaScript* é essencial, ou onde é necessário replicar a interação humana de maneira precisa.

2.5 Estudos Relacionados com o Projeto Desenvolvido

Nesta secção irão ser apresentados alguns trabalhos, que abordam o tema de modelos de previsão. Para começar, o primeiro trabalho abordado é intitulado *"Artificial Intelligence in Sports Prediction"* escrito por Alan Mccabe Jarrod Trevathan [43]. Este artigo explora a utilização de redes neuronais, nomeadamente *Multilayer perceptron*, na previsão de resultados desportivos. O modelo utiliza um conjunto de características para capturar a qualidade das equipas desportivas como pontos a favor, pontos contra, desempenho em casa e fora dela, performance nos jogos anteriores, local onde os jogos ocorreram entre outros. Este algoritmo foi testado em quatro grandes ligas desportivas: a *Australian National Rugby League* (NRL), a *Australian Football League* (AFL), o *Super Rugby* e a *English Premier League* (EPL). O sistema tem um bom desempenho e compara-se favoravelmente com os apostadores humanos em vários ambientes. Os resultados dos testes ao vivo mostram que o sistema atingiu uma precisão média de 65,1% na AFL, 63,2% na NRL, 67,5% no *Super Rugby* e 54,6% na EPL. Os autores também experimentaram um algoritmo simples de disponibilidade de jogadores, mas abandonaram esta abordagem devido ao significativo esforço necessário para a manter e ao seu impacto mínimo na precisão da previsão. O artigo conclui que os *Multilayer perceptron* utilizados foram capazes de se adaptar rapidamente e ter um bom desempenho, apesar das informações limitadas e das influências externas não incluídas no conjunto de características.

Outro estudo na área de previsão de resultados é intitulado de *"Predictive analysis and modelling football results using Machine Learning approach for English Premier League"* escrito por Rahul Baboota e Harleen Kaur [44]. Neste estudo é utilizado *Machine Learning* para construir um modelo preditivo para os resultados da *Premier League*. O objetivo é prever o resultado de um jogo de futebol entre três

classes: vitória em casa, vitória fora ou empate. Para construir o modelo preditivo, os autores usaram dados de 11 temporadas da *English Premier League*, de 2005 a 2016. Eles projetaram e extraíram recursos significativos usando engenharia de recursos, dividindo-os em duas classes: Classe A e Classe B. A Classe A inclui recursos individuais para as equipas da casa e de fora, enquanto a Classe B inclui recursos diferenciais. Os autores usaram vários algoritmos de *Machine Learning* para construir o modelo preditivo, incluindo *Naive Bayes*, *Support Vector Machine*, *Random Forest* e *Gradient Boosting*. O modelo de melhor desempenho foi o modelo de *Gradient Boosting*, seguido pela *Random Forest* e depois os dois modelos de *Support Vector Machine* com um *kernel* RBF e um *kernel* linear, respetivamente. O modelo com a precisão mais baixa foi o *Naive Bayes*. Embora o modelo de previsão não tenha conseguido superar as previsões dos apostadores, as previsões do modelo de melhor desempenho foram superadas por uma margem de apenas 0,014 a 0,015 aproximadamente. Isso demonstra a natureza promissora dos modelos, considerando as limitadas estatísticas disponíveis para os modelos, em relação às casas de apostas. Os autores concluem que a disponibilidade de recursos mais detalhados e sofisticados, como informações sobre lesões, a presença de um jogador-chave e efeitos psicológicos, melhoraria a precisão do modelo.

Outro estudo importante a abordar é o *"Predicting football scores using machine learning techniques"* escrito por Josip Hucaljuk e Alen Rakipović [45]. Este estudo tem como objetivo prever os resultados de jogos de futebol da Liga dos Campeões usando técnicas de *Machine Learning*. Para atingir este objetivo, os autores desenvolveram um sistema de *software* que testa diferentes combinações de recursos. O sistema usa um conjunto de dados composto por 20 características, como a forma atual das equipas, o resultado do encontro anterior entre as equipas, a posição atual no *ranking*, o número de jogadores lesionados e a média de golos marcados e sofridos por jogo. Os autores testaram um conjunto básico de características e um conjunto construído por especialistas, que incluía uma avaliação subjetiva da qualidade de cada equipa. Os algoritmos de *Machine Learning* usados para prever os resultados dos jogos foram: *Naive Bayes*, *Bayesian network*, *LogitBoost*, *K-nearest neighbors*, *Random Forest* e *Artificial Neural Network*. Os resultados mostraram que a rede neuronal artificial atingiu a melhor precisão de previsão, até 68%. O estudo concluiu que a previsão de resultados de jogos de futebol é uma tarefa desafiadora, mas que o sistema desenvolvido apresenta uma capacidade de previsão satisfatória, superior à do método de referência. Os autores também observaram que um conjunto de dados maior e a modelização da forma de cada jogador podem melhorar a precisão da previsão.

Outro estudo na área de previsão de resultados de futebol é o *"Predicting soccer matches with complex networks and machine learning"* escrito por Eduardo Alves

Baratela, Felipe Jordão Xavier, Thomas Peron, Paulino Ribeiro Villas-Boas e Francisco Aparecido Rodrigues [46]. Este artigo aborda a aplicação de *Machine Learning* na previsão de resultados de jogos de futebol. Os autores utilizam redes de passes para entender a dinâmica e estratégias das equipas e estatísticas de jogos anteriores. Esses dados foram então colocados para serem testados em algoritmos como *Logistic Regression*, *Random Forest* e *Extreme Gradient Boosting*. Inicialmente o modelo foi testado para o teste exclusivo de vitórias e derrotas. Segundo os autores, a *English Premier League* surgiu como a mais previsível, com uma exatidão de 80%, enquanto as outras ligas variavam entre 65% e 69%. Outro teste realizado pelos autores foi a introdução da previsão dos empates, o que segundo eles tiveram um impacto significativo no desempenho dos modelos de previsão. Sem a previsão de empates os autores obtiveram um máximo de 71% de exatidão, já com a introdução dos empates na previsão obtiveram um máximo de 55% de exatidão. O estudo concluiu, que combinar redes de passes e estatísticas de jogos, é a melhor abordagem para a previsão e que a inclusão dos empates aumenta a complexidade da tarefa de previsão, embora os valores sejam positivos, por superarem a previsão aleatória.

Outro estudo importante a abordar é o *"Neural Network Algorithm in Predicting Football Match Outcome Based on Player Ability Index"* escrito por Hengzhi Chen [47]. Este artigo aborda a utilização de diferentes algoritmos de *Machine Learning* para prever os resultados de jogos de futebol. Os dados utilizados neste projeto são referentes as épocas de 2008 até 2016, e são dados sobre os vinte e dois jogadores iniciais e sobre índices de habilidade dos jogadores. Estes dados são testados em algoritmos como *Support Vector Machine*, *Random Forest* e *Convolutional Neural Network* (CNN). Os autores obtiveram resultados de exatidão entre os 54% e os 58%, embora tenham destacado o desafio da previsão de resultados devido à influência de fatores intangíveis, como a moral da equipa, a capacidade do treinador, entre outros. O autor destaca também a dificuldade de prever empates, que são raros, comparados com as vitórias e as derrotas, pelo que o algoritmo não consegui prever corretamente nenhum empate. O estudo concluiu que os algoritmos de *Machine Learning* mostram potencial na previsão de resultados, embora sejam necessários estudos adicionais que tenham em conta a moral da equipa, a capacidade do treinador e as condições do jogo.

O último estudo a ser salientado é o *"Predicting NCAAB match outcomes using ML techniques – some results and lessons learned"* escrito por Albrecht Zimmermann, Sruthi Moorthy e Zifan Shi [48]. Este estudo teve como objetivo avaliar a utilidade da *Machine Learning* para prever o resultado de jogos da *National Collegiate Athletic Association Basketball* (NCAAB). Os autores procuraram entender a importância dos atributos das equipas e os limites da qualidade preditiva dos modelos de *Machine Learning*. Os autores descobriram que os atributos são mais determinantes do que os modelos para prever os resultados dos jogos. A precisão

preditiva depende significativamente dos atributos usados e de como eles são calculados. As Redes *Multilayer perceptron* são uma técnica de *Machine Learning* que atualmente não é amplamente utilizada, mas que se revelam mais eficazes nas configurações exploradas durante o estudo. A modelização explícita das diferenças entre os atributos das equipas não resultou em melhorias na precisão preditiva. Isso sugere que, métodos mais sofisticados de comparação direta entre atributos das equipas, podem não ser necessários. Uma das descobertas mais interessantes do estudo, é a existência de um "teto de vidro" de aproximadamente 74% de precisão preditiva. Os autores concluíram que, independentemente da técnica estatística ou de *Machine Learning* empregada, a precisão preditiva não ultrapassa este valor. Esta descoberta está alinhada com a precisão dos sistemas preditivos existentes. O estudo concluiu ainda que a importância dos atributos na previsão de resultados de jogos da NCAAB, revela um limite máximo de precisão preditiva alcançável com técnicas atuais de *Machine Learning*. Estas descobertas fornecem uma base sólida para futuras pesquisas e melhoramentos na área de previsão de resultados desportivos.

Capítulo 3

Arquitetura do Projeto

Neste capítulo, são apresentados a análise e o processamento de dados realizados no projeto, bem como a descrição detalhada do modelo utilizado.

3.1 Análise e Processamento de Dados

Ao longo desta secção, serão apresentados detalhadamente todos os dados utilizados no projeto, bem como o processamento necessário para a sua preparação.

3.1.1 Análise de Dados

Os dados que serão utilizados no presente estudo, correspondem a um total de 1836 jogos de futebol, que abrangem 6 épocas, de 2018/2019 a 2023/2024. Os jogos são relativos à primeira divisão de futebol portuguesa, cuja designação oficial é Liga Portugal. Nos 1836 jogos recolhidos, a equipa da casa venceu em 800 (43,57%), houve 435 empates (23,69%) e a equipa visitante venceu em 601 (32,73%). Na Figura 3.1 é apresentado um gráfico com a análise percentual dos resultados dos jogos.

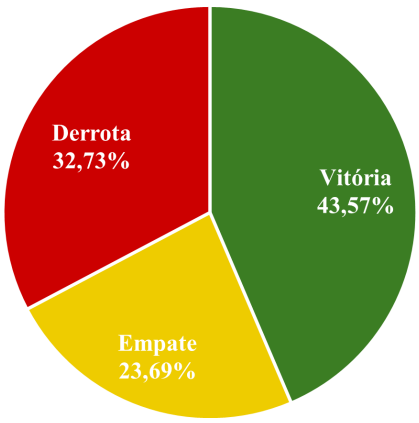


Figura 3.1: Gráfico de análise dos dados da Primeira Liga entre 2018 e 2024

Estes dados foram obtidos através do *website* FBref (Figura 3.2), uma plataforma *online* amplamente reconhecida por fornecer estatísticas detalhadas sobre o futebol mundial. Lançado como parte da família de *website* do *Sports Reference*, o FBref reúne dados e análises abrangentes de jogos, ligas, clubes e jogadores, atendendo tanto a entusiastas do desporto quanto a profissionais do setor.

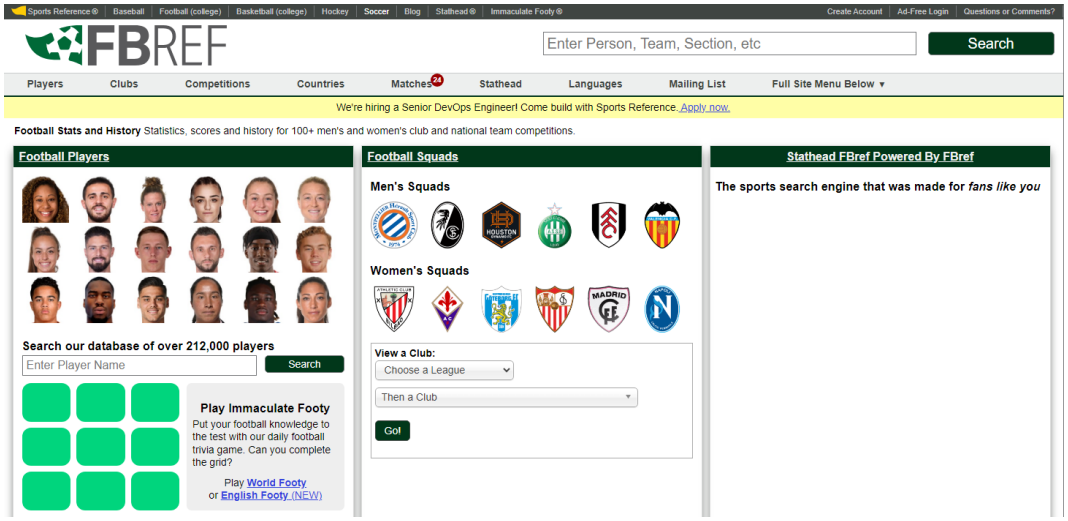


Figura 3.2: *website* FBref

Esta base de dados contém diversas características (*features*) relacionadas aos jogos analisados, obtidas a partir de diferentes tabelas. Da primeira tabela, foram extraídos dados como a data do jogo (”date”), a hora do jogo (”time”), o dia da semana em que o jogo foi disputado (day”), a localização do jogo (se foi disputado em casa ou fora, representada pela *feature* ”venue”), o resultado do jogo (”result”),

golos a favor ("gf"), golos contra ("ga"), o adversário ("opponent"), golos esperados ("xg"), golos esperados contra ("xga"), posse de bola ("poss"), número de espetadores ("attendance"), o capitão de equipa ("captain"), a formação tática utilizada pela equipa da casa ("formation") e pelo adversário ("opp_formation") e o árbitro ("referee"). Da segunda tabela, foram extraídos dados sobre a atuação do guarda-redes, incluindo remates da equipa contrária à baliza ("sota"), defesas realizadas ("saves") e se o guarda-redes manteve um *clean sheet* ("cs") ou não. A terceira tabela forneceu dados sobre os remates, como o total de remates ("sh"), remates à baliza ("sot"), número total de penáltis ("pkatt"), penáltis convertidos ("pk"), penáltis concedidos ("pkcon") e penáltis ganhos ("pkwon"). Finalmente, a última tabela incluiu diversas estatísticas adicionais, como cartões amarelos ("crdy"), cartões vermelhos ("crdr"), segundos cartões amarelos ("2crdy"), faltas cometidas ("fls"), faltas sofridas ("fld"), cruzamentos ("crs"), interseções ("int"), interseções bem-sucedidas ("tklw") e auto golos ("og").

Na Tabela 3.1 é apresentado um excerto do conjunto de dados relativo aos jogos de futebol utilizados.

Tabela 3.1: Excerto da base de dados

	date	**ga_x**	**sota**	**saves**	**cs**	**psxg**	**psxg+/-**	...
0	2018-08-10	3.0	11.0	7.0	0	2.9	-0.1	...
1	2018-08-10	2.0	5.0	3.0	0	1.3	-0.7	...
2	2018-08-11	2.0	5.0	3.0	0	2.2	0.2	...
3	2018-08-11	0.0	0.0	0.0	1	0.0	0.0	...
...	...	...	...	...	...	...	...	...
3401	2024-01-31	1.0	8.0	7.0	0	1.8	0.8	...

Na Tabela 3.2 são apresentadas todas as *features* selecionadas para este projeto.

Tabela 3.2: Descrição das *features*

Nome	**Descrição**
2crdy	segundos cartões amarelos
attendance	número de espetadores
captain	capitão de equipa
crdr	cartões vermelhos
crdy	cartões amarelos
crs	cruzamentos
cs	*clean sheet*
day	dia do jogo
date	data do jogo
fld	faltas sofridas
fls	faltas cometidas
formation	formação tática
ga	golos contra
gf	golos a favor
int	interseções
og	auto golos
opponent	adversário
opp_formation	formação tática do adversário
pk	penáltis convertidos
pkatt	número total de penáltis
pkcon	penáltis concedidos
pkwon	penáltis ganhos
poss	posse de bola
referee	árbitro
result	resultado do jogo (V/E/D)
saves	defesas realizadas
sh	remates totais
sot	remates à baliza
sota	remates da equipa contrária à baliza
time	hora do jogo
tklw	interseções bem-sucedidas
venue	jogo em casa ou fora
xg	golos esperados
xga	golos esperados contra

3.1.2 Processamento de Dados

Os dados que já se encontravam em formato numérico, estavam prontos para serem interpretados por um algoritmo. No entanto, algumas variáveis estavam em formato de *string* e precisaram ser convertidas. A primeira variável a ser convertida foi a ”date”, que foi transformada no formato *datetime64*. A variável ”venue” foi convertida numa nova variável chamada ”venue_code”, que é um valor binário que: quando o jogo foi disputado em casa, esta variável assume o valor 1, e quando o jogo foi fora de casa, assume o valor 0. A variável ”opponent” foi convertida numa nova variável denominada ”opp_code”, que consiste num valor único atribuído a cada equipa adversária. A variável ”hour” foi transformada num valor numérico, assim como a variável ”day”, que passou a representar numericamente cada dia da semana em que o jogo foi disputado. Por fim, a variável ”result” foi convertida numa nova variável chamada ”target”, onde uma vitória é representada pelo valor 1, um empate pelo valor 0 e uma derrota pelo valor 2.

O próximo passo era realizar testes às *features* para se detetar as que tem mais impacto para esta previsão. Para isto foi utilizado uma técnica chamada *Sequential Feature Selector* (SFS).

O *Sequential Feature Selector* é um algoritmo que adiciona ou remove iterativamente caraterísticas de um conjunto de dados, para melhorar o desempenho de um modelo preditivo, que pode ser uma seleção direta ou uma seleção inversa. O processo do algoritmo é o seguinte, é inicialmente definido o número de caraterísticas a selecionar, a direção e a métrica de avaliação. Após isto, o seletor ajusta o modelo de previsão ao conjunto completo de caraterísticas, e avalia o modelo no conjunto de treino utilizando a métrica de pontuação. Por fim, a caraterística que melhorar mais a pontuação de validação cruzada do modelo, é adicionada ao conjunto de caraterísticas selecionadas (*Forward selection*), ou a caraterística que menos reduzir a pontuação de validação cruzada do modelo, é removida do conjunto de caraterísticas selecionadas (*Backward selection*) [49]. Na Figura 3.3 está ilustrado um exemplo de *Sequential Feature Selector* com a direção *Backward selection*.

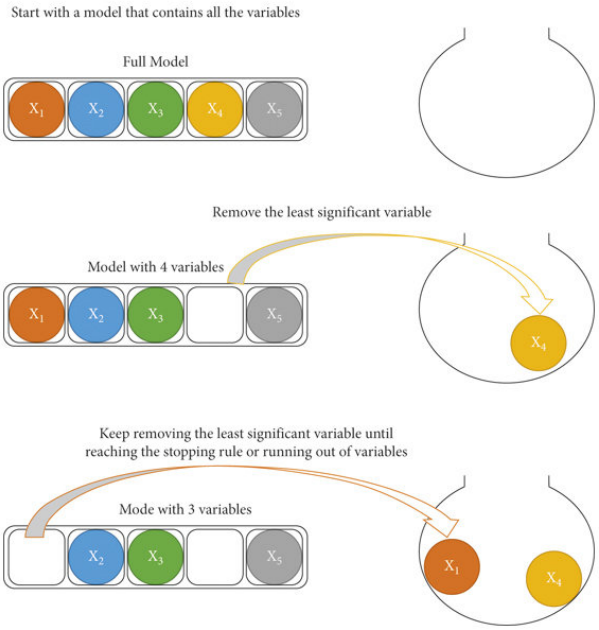


Figura 3.3: SFS com *Backward selection* [50]

3.2 Modelo

Do conjunto de algoritmos apresentados na subsecção 2.3 desta dissertação, foram selecionados para implementação neste projeto os algoritmos *Logistic Regression*, *Neural Network*, *Random Forest*, *Support Vector Machine*, *Extreme Gradient Boosting*, *Bagging Classifier* e *K-Nearest Neighbors* (KNN). A escolha destes algoritmos deve-se ao facto de serem métodos de *Supervised Learning* amplamente utilizados em problemas de previsão, conforme evidenciado nos estudos abordados na subsecção 2.4 desta dissertação.

Como os dados obtidos estão no formato *Comma Separated Values* (CSV), é necessário importá-los utilizando funções de leitura em Python. Após a importação dos dados, é possível aplicar as técnicas de processamento descritas na subsecção 3.1.2. Concluídas as alterações nos dados, procedeu-se à etapa de *feature selection* para todos os algoritmos testados. Com as *features* selecionadas, definem-se os melhores parâmetros a serem utilizados em cada algoritmo designados de hiperparâmetros, visando otimizar o desempenho de todos os modelos. Para a descoberta desses melhores parâmetros foi utilizado o algoritmo *Grid Search CV*. Este algoritmo é um processo de afinação de hiperparâmetros para determinar os valores ótimos para um determinado modelo. O desempenho de um modelo depende significativamente do valor desses hiperparâmetros [51].

Para realizar uma avaliação dos resultados obtidos pelo modelo, foram utilizadas métricas de avaliação como *F1 score*, Precisão, Exatidão e *Recall*. Na Equação (3.1)

é apresentada a fórmula utilizada no cálculo da Exatidão, onde é realizado, uma divisão entre o número de casos em que o modelo previu corretamente o resultado e o número total de previsões do modelo.

$$\text{Exatidão} = \frac{\text{Número de previsões corretas}}{\text{Número total de previsões}} \tag{3.1}$$

A Precisão em problemas de classificação multiclasse é calculada individualmente para cada classe, a qual consiste numa divisão entre verdadeiros positivos (VP) e a soma entre verdadeiros positivos e falsos positivos (FP). Na Equação (3.2) é apresentada a fórmula de cálculo da precisão onde "i" representa cada classe. Esta métrica pode ser apresentada de três formas, a *Macro-Averaging* que consiste em calcular a precisão de cada classe individualmente; e depois faz-se a média aritmética; a *Weighted-Averaging* que consiste no cálculo da precisão para cada classe, ponderando cada uma conforme o número de previsões dessa classe, fazendo com que as classes maiores tenham mais peso no cálculo da precisão geral; e por fim *Micro-Averaging* que consiste em combinar os valores de verdadeiros positivos e falsos positivos de todas as classes antes de calcular a precisão.

$$\text{Precisão}_i = \frac{\text{VP}_i}{\text{VP}_i + \text{FP}_i} \tag{3.2}$$

O *Recall* é a métrica que mede a proporção de exemplos de uma classe específica que foram corretamente classificados como tal, em relação ao número total de exemplos que pertencem a essa classe. Tal como a Precisão, em problemas multiclasse pode ser apresentado em *Macro-Averaging*, *Weighted-Averaging* e *Micro-Averaging*. Na Equação (3.3) é apresentada a fórmula de cálculo do *Recall* que consiste na divisão entre os verdadeiros positivos (VP) e a soma entre verdadeiros positivos (VP) e falsos negativos (FN).

$$\textit{Recall}_i = \frac{\text{VP}_i}{\text{VP}_i + \text{FN}_i} \tag{3.3}$$

O *F1 score* é uma métrica de desempenho que considera tanto a Precisão como o *Recall*. É útil para encontrar um equilíbrio entre estes dois aspetos, especialmente quando existe um desequilíbrio entre as classes. Tal como o *Recall* e a Precisão, o *F1 score* pode ser apresentado nas três formas *Macro-Averaging*, *Weighted-Averaging* e *Micro-Averaging*. Na Equação (3.4) é apresentada a fórmula de cálculo do *F1 score*.

$$\textit{F1 score}_i = 2 * \frac{\text{Precisão}_i * \text{Recall}_i}{\text{Precisão}_i + \text{Recall}_i} \tag{3.4}$$

Na Figura 3.4 é apresentado um fluxograma resumo de todo o processo de desenvolvimento do modelo, agora descrito.

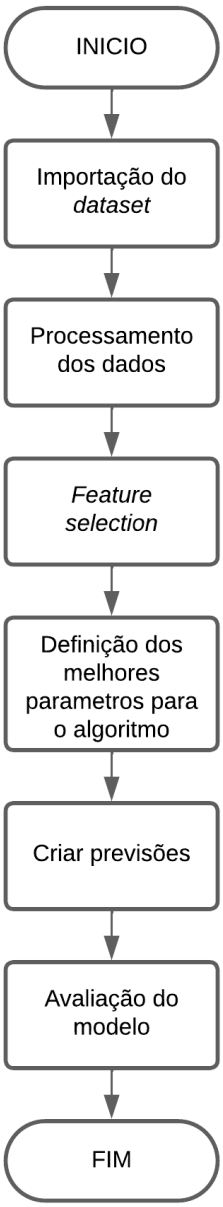


Figura 3.4: Fluxograma do Modelo

Capítulo 4

Implementação

Neste capítulo, descreve-se detalhadamente todo o processo de desenvolvimento deste projeto, desde a obtenção dos dados, passando pela *feature selection*, até à elaboração do algoritmo e, por fim, as métricas utilizadas na sua avaliação.

4.1 *Web Scraping*

Esta subsecção explica como o dataset foi obtido para treinar e testar o modelo, através da recolha de dados do *website* FBref. Na Figura 4.1 é possível ver um exemplo de uma tabela de onde a informação foi retirada.

Date	Time	Comp	Round	Day	Venue	Result	GF	GA	Opponent	xG	xGA	Poss	Attendance	Captain	Formation	Opp Formation	Referee
2024-04-07	20:30	Primeira Liga	Matchweek 28	Sun	Home	L	1	2	Vitória	1.2	0.2	69	34,582	Pepe	4-2-3-1	3-5-2	Fábio Veríssimo
2024-04-13	18:00	Primeira Liga	Matchweek 29	Sat	Home	D	2	2	Famalicão	2.8	2.1	69	30,510	Diogo Costa	4-2-3-1	4-2-3-1	Gustavo Correia
2024-04-21	18:00	Primeira Liga	Matchweek 30	Sun	Away	W	2	1	Casa Pia	1.7	0.9	55	3,387	Pepe	4-2-3-1	3-4-3	Manuel Oliveira
2024-04-28	20:30	Primeira Liga	Matchweek 31	Sun	Home	D	2	2	Sporting CP	1.5	1.6	43	45,230	Diogo Costa	4-2-3-1	3-4-3	Nuno Almeida
2024-05-04	20:30	Primeira Liga	Matchweek 32	Sat	Away	W	3	0	Chaves	2.6	0.3	73	3,439	Diogo Costa	4-2-3-1	4-3-3	José Armando Torres Bessa
2024-05-12	20:30	Primeira Liga	Matchweek 33	Sun	Home	W	2	1	Boavista	2.4	0.2	76	33,031	Diogo Costa	4-2-3-1	4-1-4-1	Artur Soares Dias
2024-05-18	20:30	Primeira Liga	Matchweek 34	Sat	Away	W	1	0	Braga	2.1	0.4	68	23,284	Diogo Costa	4-2-3-1	4-2-3-1	Nuno Almeida

Figura 4.1: Tabela FBref

Num algoritmo de *Web scraping* o primeiro passo a ser realizado é enviar uma solicitação HTML à página *web* onde se encontra o conteúdo desejado, para se obter o conteúdo HTML da mesma. Com o auxílio de bibliotecas, o próximo passo é identificar e extrair elementos de interesse, como textos, tabelas ou imagens. Por fim, os dados são estruturados e armazenados no formato pretendido.

Inicialmente foi escolhida qual a informação a ser retirada. Do *website* foram escolhidas quatro tabelas, *Scores & Fixtures*, *Shooting*, *Goalkeeping* e *Miscellaneous Stats*. Para realizar a obtenção destes dados, foram desenvolvidos três *scripts* iguais de onde, no primeiro foram obtidos os dados de *Scores & Fixtures* e *Shooting*, no segundo os dados de *Scores & Fixtures* e *Goalkeeping* e no terceiro os dados de *Scores & Fixtures* e *Miscellaneous Stats*. Posteriormente, esses três *datasets* foram fundidos num único onde está presente toda a informação necessária para o modelo deste projeto.

Para realizar a obtenção do conteúdo HTML foi utilizada a biblioteca de Python *Requests*, que fornece uma interface fácil de utilizar, a qual simplifica o processo de envio e receção de dados de *website* e fornece uma *interface* uniforme para os métodos GET e POST [52]. Para realizar a análise do conteúdo HTML recebido, foi utilizada a biblioteca *Beautiful Soup* que permite interagir com o HTML de uma forma semelhante à forma como se interage com uma página Web, utilizando ferramentas de programação. A biblioteca expõe um par de funções intuitivas que podem ser utilizadas para explorar o conteúdo HTML [53]. Para converter o HTML para um *dataset* foi utilizada a biblioteca Pandas que, de uma maneira simples e intuitiva permite trabalhar com dados relacionais.

No algoritmo desenvolvido, inicialmente é necessário realizar a importação das três bibliotecas de Python que vão ser utilizadas, e definir o URL onde se encontra a informação pretendida. Este algoritmo foi utilizado para obter dados de seis épocas. Para garantir o bom funcionamento e diminuir a probabilidade de erro, obteve-se dados de duas épocas de cada vez. Para obter os dados de duas épocas foi necessário criar um vetor com os anos das duas épocas. Com o objetivo do código correr duas vezes, uma para cada época, foi criado um *loop*.

Com a ajuda da biblioteca *Beautiful Soup*, obtiveram-se os dados HTML da tabela *stats_table*, onde se encontram os URL de cada equipa, para a seguir realizar a obtenção dos dados de cada uma delas. No bloco de código 4.1 é possível ver o código utilizado para a obtenção dos dados HTML da tabela *stats_table*.

```
1 data = requests.get(standings_url)
2 soup = BeautifulSoup(data.text)
3 standings_table = soup.select('table.stats_table')[0]
```

Código 4.1: Código para obtenção dos dados HTML da tabela *stats_table*

Com os dados HTML obtidos, é necessário filtrá-los para se obter os URL relativos das equipas, com o objetivo de através dos mesmos se adquirir os dados individuais dos jogos de cada equipa. Com estes URL, o último passo consistiu em

acrescentar 'https://fbref.com' aos URL da tabela *stats_table*, convertendo-os assim em URL absolutos (Código 4.2).

```
links = [l.get("href") for l in standings_table.
    find_all('a')]
links = [l for l in links if '/squads/' in l]
team_urls = [f"https://fbref.com{l}" for l in links]
```

Código 4.2: Código para obter e converter os URL das equipas

Visto que o *script* visa a obtenção de duas épocas, é necessário recolher o URL do botão que apresenta os dados na época anterior (Figura 4.2). Para esse fim, foi utilizada a biblioteca *Beautiful Soup* (Código 4.3).

Figura 4.2: Botão *Previous season*

```
previous_season = soup.select("a.prev")[0].get("href")
```

Código 4.3: Código para se obter o URL da época anterior

Para obter os dados de todas as equipas, foi criado um *loop* que utilizará um de cada vez, todos os URL das equipas. Para obter o nome da equipa foi utilizado o URL. A estratégia passou por eliminar do URL tudo o que estava para trás da última barra, ficando apenas com o nome da equipa e com um "-Stats". O último passo seria então retirar o "-Stats". Algumas equipas nas quais o seu nome não seja só uma palavra como, por exemplo, Rio Ave, foi também retirado o hífen que dividia as palavras. No Código 4.4 é possível ver os resultados do processo de obtenção dos nomes das equipas.

```
https://fbref.com/en/squads/eea856da/2023-2024/Rio-Ave-Stats
Rio-Ave-Stats
Rio Ave
```

Código 4.4: Extração dos nomes das equipas apartir dos URL

Seguidamente foi obtida a informação pretendida do *website*. Os primeiros dados a serem retirados foram da tabela *Scores* & *Fixtures* (Código 4.5). Nesta tabela estão presentes dados como golos esperados (”xg”), golos esperados contra (”xga”), posse de bola (”poss”), a formação tática utilizada pela equipa da casa (”formation”) e pelo adversário (”opp_formation”), entre outros.

```
data = requests.get(team_url)
matches = pd.read_html(data.text, match="Scores & Fixtures")[0]
```

Código 4.5: Código para obter a informação da tabela *Scores* & *Fixtures*

O processo até este ponto é comum aos três *scripts*. O próximo passo no primeiro *script*, passa por obter a *hiperligação* da tabela *Shooting* e convertê-lo num URL absoluto. No segundo *script* o objetivo é o mesmo, mas a tabela alvo é a *Goalkeeping*. Por fim, no último *script*, obteremos o URL da tabela *Miscellaneous Stats*. No excerto de código 4.6 é apresentado o processo de obtenção e conversão do URL da tabela *Shooting*.

```
links = [l.get("href") for l in soup.find_all('a')]
links = [l for l in links if l and 'all_comps/shooting/' in l]
data = requests.get(f"https://fbref.com{links[0]}")
```

Código 4.6: Código para obter e converter o URL da tabela *Shooting*

A próxima tarefa a realizar é obter os dados das novas tabelas. Para esse efeito a estratégia é a mesma anteriormente apresentada, onde com a ajuda da biblioteca Pandas é realizada a leitura dos dados. O código necessário para realizar a leitura é apresentado no excerto de código 4.7, onde o nome da tabela alvo varia nos três *scripts*.

```
shooting = pd.read_html(data.text, match="Shooting")
    [0]
shooting.columns = shooting.columns.droplevel()
```

Código 4.7: Código para obter os dados da tabela *Shooting*

Com os dados de duas tabelas por *script* é necessário fundir os dois *datasets* num só. Foram selecionadas as colunas a serem adicionadas ao *dataset* da tabela *Scores & Fixtures*. Para garantir que a biblioteca Pandas não desse erro, devido à falta de algumas estatísticas para algumas equipas, este passo foi colocado num ciclo *try*, onde as equipas que não tem esses dados são passadas à frente. No código 4.8 é apresentado o exemplo para os dados relativos à tabela de *Shooting*. Para as duas restantes tabelas é alterado o nome da mesma e as colunas que se vão fundir com a tabela *Scores & Fixtures*.

```
try:
    team_data = matches.merge(shooting[["Date", "Sh",
        "SoT", "PK", "PKatt"]], on="Date")
except ValueError:
    continue
```

Código 4.8: Código para fundir os dados da tabela *Scores & Fixtures* com os dados da tabela *Shooting*

Neste ponto, na variável "team_data" já temos todos os dados da equipa. Como o foco deste projeto é a Primeira Liga, é necessário selecionar só os jogos dessa competição. Para isso, foram selecionados apenas os jogos nos quais a coluna "Comp" tinha como valor "Primeira Liga". As últimas informações necessárias que não estavam presentes na "team_data" era a época a que os jogos se referem e a equipa que os disputou. Para armazenarmos a época, foi criada uma coluna, chamada "Season", e foi-lhe atribuído o valor presente na variável "year". O mesmo foi feito para a equipa, em que uma nova coluna chamada "Team" foi criada e atribuído o valor presente na variável "team_name". Por fim esses dados são colocados numa variável chamada "all_matches", que a cada iteração do *loop* vai ficar com mais um *dataset* dos dados de cada equipa, até ficar com todos os dados pretendidos (Código 4.9).

```
team_data = team_data[team_data["Comp"] == "Primeira
    Liga"]
team_data["Season"] = year
team_data["Team"] = team_name
all_matches.append(team_data)
```

Código 4.9: Processamento dos dados obtidos

No fim de cada ciclo de código, é introduzido um tempo de espera de cinco segundos. Esta paragem no código tem como objetivo garantir que o *website* continua a permitir obter informação. A maioria dos *website* tem este mecanismo de defesa, porque se o *Web scraping* for realizado muito rápido pode originar uma quebra de desempenho do *website*. A última tarefa a realizar, é converter toda a informação presente na variável ”all_matches” num único *dataset* e exportá-lo para um ficheiro CSV para poder ser utilizado (Código 4.10). A função utilizada para concatenar os *datasets* num só, foi a *concat* da biblioteca *Pandas*.

```
match_df = pd.concat(all_matches)
match_df.to_csv("shot.csv")
```

Código 4.10: Concatenar e exportar o *dataset* como CSV

Com os dados obtidos dos três *scripts*, os *datasets* resultantes são fundidos e as colunas repetidas apagadas, ficando apenas com um *dataset* completo com os dados referentes às diferentes tabelas. Na Figura 4.3 é apresentado um fluxograma resumo de todo o processo realizado para a obtenção dos dados por *Web scraping*.

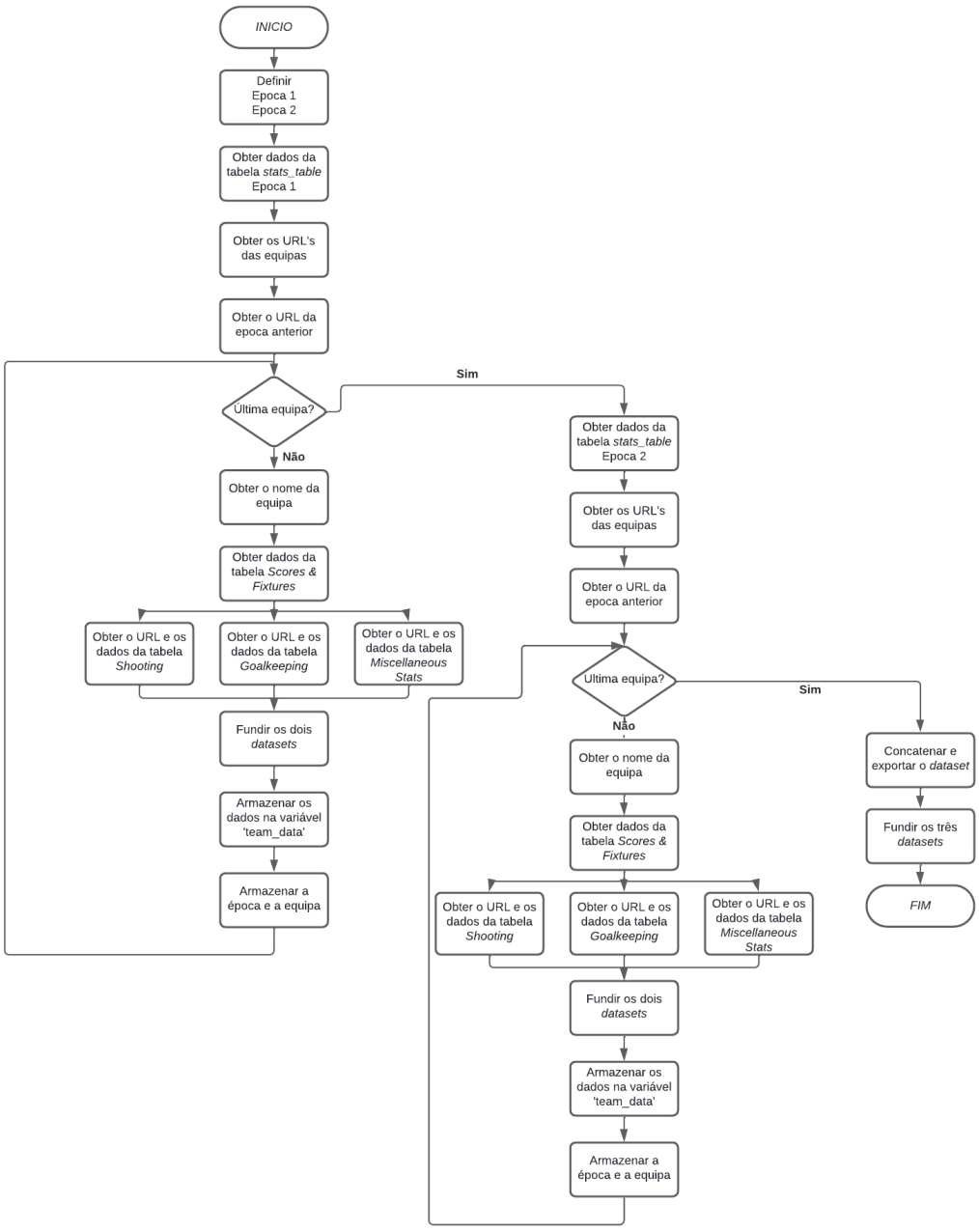


Figura 4.3: Fluxograma do processo de *Web scraping*

4.2 *Feature Selection*

Na tentativa de obter os melhores resultados possíveis, foi utilizado, uma técnica chamada *Feature selection*. Neste processo usou-se um método, o *Sequential Feature Selector* (SFS) no modo *Backward selection*. No algoritmo desenvolvido inicialmente, é necessário realizar a importação das bibliotecas nomeadamente do algoritmo que queremos utilizar, o *StratifiedKFold*, o *SequentialFeatureSelector* e o *matplotlib*. Com as bibliotecas carregadas, o *dataset* é importado. Para se fazer a divisão entre dados de teste e dados de treino, é necessário converter a coluna ”date” para *datetime64*. Os dados de treino e teste vão ser divididos no dia um de agosto de dois mil e vinte e três, ficando assim os dados de treino com todos os jogos ocorridos antes dessa data e os dados de teste com os jogos ocorridos após essa data (Código 4.11).

```
1 cutoff_date = pd.to_datetime('2023-08-01')
2 train_df = df[df['date'] < cutoff_date]
3 test_df = df[df['date'] >= cutoff_date]
```

Código 4.11: Divisão do *dataset* em treino e teste

Com o *dataset* dividido, é agora tempo de definir as *features* a serem testadas. O próximo passo é configurar o algoritmo que testaremos e fazer a validação cruzada. Para isso, foi utilizada neste projeto a validação cruzada estratificada, pois o *dataset* é dividido em K grupos, mantendo a proporção das classes do conjunto de dados original. Esse valor de K, varia com o tamanho do *dataset*, ou seja, quanto maior o número de dados maior deve ser esse valor. Neste projeto o valor utilizado foi o 5, que é o valor usualmente utilizado. No Código 4.12 a definição da validação cruzada é guardada numa variável 'cv' que mais tarde será utilizada para a definição do *Sequential Feature Selector* (SFS).

```
1 cv = StratifiedKFold(n_splits=5)
```

Código 4.12: Definição da validação cruzada

Com o objetivo de imprimir os resultados e no fim criar um gráfico que relacione o valor da exatidão em função do número de *features* selecionadas, foi necessário criar três vetores, o primeiro para guardar o número de *features*, o segundo para guardar os valores de exatidão e o terceiro para guardar as *features* selecionadas (Código 4.13).

```
n_features_list = []
accuracy_list = []
selected_features_list = []
```

Código 4.13: Criação dos vetores para armazenar os dados do SFS

Para testar e obter os valores de todas as exatidões e o número de *features*, foi criado um *loop* onde o número de *features* desejadas incrementa a cada iteração, sendo o valor armazenado na variável 'n_features'. Os valores utilizados foram no mínimo de 10 e com o valor máximo o de todas as *features* disponíveis. Dentro deste *loop*, é necessário definir os parâmetros a utilizar no SFS. Os parâmetros que são necessários fornecer ao SFS são o estimador, que representa o algoritmo que será utilizado para as previsões, o número de *features* que como já foi dito a cada iteração aumentará, a direção que neste caso será *Backward selection*, e o número da validação cruzada que se encontra armazenado na variável 'cv'. No Código 4.14 é apresentada a definição do SFS com os parâmetros desejados.

```
sfs = SequentialFeatureSelector(bagging,
    n_features_to_select=n_features, direction="
    backward", cv=cv)
```

Código 4.14: Parâmetros do SFS

Após o algoritmo correr e obter as melhores *features* para o número atual, é necessário guardar as *features* para o algoritmo ser treinado, testado e mais tarde ser apresentada a sua exatidão. No Código 4.15 é demonstrado o processo para obter as *features* do SFS e como armazená-las no vetor 'selected_features_list'.

```
selected_features = np.array(common_features)[sfs.
    get_support()]
selected_features_list.append(selected_features)
```

Código 4.15: Obter e armazenar as *features*

Com as *features* no vetor 'selected_features_list' é então o momento para treinar e avaliar o modelo. Para treinar o modelo vão ser utilizados os dados de treino previamente divididos, e para a avaliação vão ser utilizados os dados de teste. O Código utilizado para o treino e a avaliação do modelo com as *features* selecionadas encontra-se no excerto de código 4.16.

```
bagging.fit(X_train[selected_features], y_train)
score = bagging.score(X_test[selected_features],
    y_test)
```

Código 4.16: Treino e avaliação do algoritmo

Com os resultados já obtidos é altura de armazenar o número de *features* e a exatidão obtida nos vetores previamente criados, 'n_features_list' e 'accuracy_list' respetivamente. Após o armazenamento, os dados referentes ao número de *features*, as *features* selecionadas e a exatidão vão ser apresentados no ecrã. Na Figura 4.4 é possível ver um exemplo da impressão das *features* ao longo das iterações.

```
Número de features: 10, Score: 0.4382
Features selecionadas: ['ga_x_rolling' 'sota_rolling' 'saves_rolling' 'fk_rolling' 'gf_rolling'
 'poss_rolling' 'fls_rolling' 'fld_rolling' 'crs_rolling' 'int_rolling']
Número de features: 11, Score: 0.4382
Features selecionadas: ['ga_x_rolling' 'sota_rolling' 'saves_rolling' 'fk_rolling' 'gf_rolling'
 'poss_rolling' 'fls_rolling' 'fld_rolling' 'crs_rolling' 'int_rolling'
 'og_rolling']
Número de features: 12, Score: 0.4119
Features selecionadas: ['ga_x_rolling' 'sota_rolling' 'saves_rolling' 'fk_rolling' 'gf_rolling'
 'xg_rolling' 'poss_rolling' 'fls_rolling' 'fld_rolling' 'crs_rolling'
 'int_rolling' 'og_rolling']
```

Figura 4.4: Impressão dos resultados do *Sequential Feature Selector*

Após a impressão dos resultados obtidos, o código volta a correr, realizando assim mais uma iteração até alcançar o máximo de *features* definido. Foi utilizada validação cruzada para validar a eficácia de diferentes conjuntos, ou seja, vários conjuntos de *features* são testados, sendo selecionado aquele que obtiver melhor exatidão. Quando o código já correu as vezes desejadas, é então altura para criar um gráfico que relacione o valor de exatidão em função do número de *features* selecionadas, o conjunto de features com maior exatidão é o escolhido. Na Figura 4.5 esta representado um exemplo do gráfico obtido no final do procedimento do *Sequential Feature Selector*. Neste processo de *feature selection* não foram colocadas *features* referentes aos dados do jogo atual, ou seja, o oponente ("opp_code"), se o jogo é em casa ou fora dela ("venue_code"), a hora ("hour") e o dia ("day_code"). A sua introdução foi testada mais tarde.

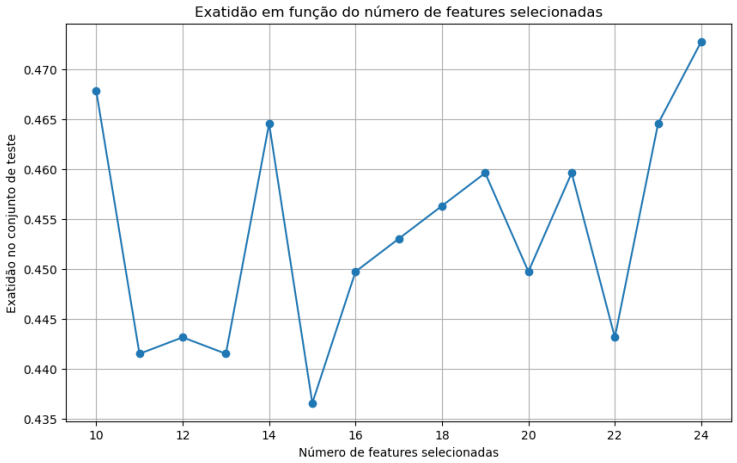


Figura 4.5: Gráfico da exatidão em função do número de *features* selecionadas

4.3 Melhores Hiperparâmetros

Com as *features* selecionadas, inicia-se a busca pelos melhores hiperparâmetros a serem utilizados em cada algoritmo. Para essa tarefa foi selecionado o algoritmo *Grid Search CV*. No algoritmo desenvolvido é necessário importar inicialmente as bibliotecas, nomeadamente o algoritmo que será usado para as previsões e também a biblioteca *GridSearchCV*. Neste método é necessário definir os hiperparâmetros a serem testados, variando estes de algoritmo para algoritmo. No Código 4.17 é apresentado um exemplo de definição dos hiperparâmetros a serem testados, neste caso para o algoritmo *Bagging Classifier*.

```
param_grid = {
    'n_estimators': [10, 50, 100, 200],
    'max_samples': [0.5, 0.7, 1.0],
    'max_features': [0.5, 0.7, 1.0],
    'bootstrap': [True, False],
    'bootstrap_features': [True, False],
    'base_estimator__max_depth': [3, 5, 10, None],
    'base_estimator__min_samples_split': [2, 5, 10]
}
```

Código 4.17: Definição dos hiperparâmetros para teste *Bagging Classifier*

Com a definição dos hiperparâmetros concluída, é altura de definir os parâmetros que queremos utilizar no *Grid Search*. O primeiro parâmetro a ser definido é o estimador, o qual é o algoritmo que vai realizar as previsões, depois vamos definir os hiperparâmetros de teste que neste caso estão guardados em "param_grid", o número da validação cruzada que irá ser 5, e por fim a métrica que queremos ter como comparação, que neste caso foi a precisão na forma *Weighted-Averaging*, por ser a mais indicada para problemas de multiclasse onde as classes não são igualmente divididas. No Código 4.18 são apresentados os parâmetros utilizados no *Grid Search* no caso do *Bagging Classifier*.

```
grid_search = GridSearchCV(estimator=bagging,
    param_grid=param_grid, cv=5, scoring='
    precision_weighted', n_jobs=-1)
```

Código 4.18: Parâmetros utilizados no *Grid Search*

Com todos os parâmetros definidos, o próximo passo é ”correr” o algoritmo e obter os melhores hiperparâmetros. O Código utilizado para obter os melhores hiperparâmetros encontra-se no excerto de código 4.19.

```
best_params = grid_search.best_params_
best_score = grid_search.best_score_
```

Código 4.19: Obtenção dos melhores hiperparâmetros

O último passo é imprimir esses parâmetros no ecrã para poderem ser utilizados no algoritmo que faz as previsões. Na Figura 4.6 está representado um exemplo do *output* obtido no final do procedimento do *Grid Search*.

```
Melhores parâmetros encontrados para Bagging: {'base_estimator__max_depth': 5, 'base_estimator__min_samples_split': 5, 'bootstr
ap': False, 'bootstrap_features': True, 'max_features': 0.7, 'max_samples': 1.0, 'n_estimators': 100}
Melhor pontuação para Bagging: 0.5809636845999531
```

Figura 4.6: *Output* com os melhores hiperparâmetros do *Bagging Classifier*

Com estes resultados concluímos que para o *Bagging Classifier*, os melhores parâmetros são os apresentados na Tabela 4.1

Tabela 4.1: Melhores hiperparâmetros *Bagging Classifier*

Parâmetro	Valor
base_estimator_max_depth	5
base_estimator_min_samples_split	5
bootstrap	*False*
bootstrap_features	*True*
max_features	0.7
max_samples	1.0
n_estimators	100

4.4 Modelo

O primeiro passo no desenvolvimento do modelo é carregar os dados a partir de um ficheiro CSV. Para realizar a leitura do *dataset* foi utilizada a biblioteca *Pandas*. Com os dados já importados, é altura de realizar algumas verificações nos dados. A primeira verificação a ser feita foi o número de jogos que o *dataset* tinha para cada época (Figura 4.7). O valor esperado era de 612, porque são 18 equipas e 34 jornadas, o que dá um total de 612 jogos devido ao facto de cada jogo ser representado no *dataset* duas vezes, uma para a equipa da casa e outra para o visitante. Outra verificação realizada foi as equipas terem o mesmo número de jogos na coluna ”team” e na coluna ”opponent” o que garante que todos os jogos se encontram no *dataset*, tanto para a equipa da casa como para o visitante. Por fim, foi também verificado se existia o mesmo número de jogos em todas as jornadas.

```
matches["season"].value_counts()

2024    612
2023    612
2022    612
2021    612
2020    612
2019    612
Name: season, dtype: int64
```

Figura 4.7: Número de jogos por época

Algumas variáveis estavam em formato de *string* e precisaram ser convertidas. As variáveis convertidas foram a ”date”, que foi transformada no formato *datetime64*; a ”venue” foi convertida numa variável ”venue_code”, que é um valor binário que: quando o jogo foi disputado em casa, esta variável assume o valor 1, e quando o jogo foi fora de casa, assume o valor 0; a variável ”opponent” foi convertida numa variável ”opp_code”, que consiste num valor único atribuído a cada equipa adversária; a variável ”hour” foi transformada num valor numérico, assim como a variável ”day”,

que passou a representar numericamente cada dia da semana em que o jogo foi disputado, e a variável ”result” que foi convertida numa variável ”target”, onde uma vitória (W) é representada pelo valor 1, um empate (D) pelo valor 0 e uma derrota (L) pelo valor 2. O código para a conversão das variáveis está apresentado no excerto de código 4.20.

```
matches["date"] = pd.to_datetime(matches["date"])
matches["venue_code"] = matches["venue"].astype("
    category").cat.codes
matches["opp_code"] = matches["opponent"].astype("
    category").cat.codes
matches["hour"] = matches["time"].str.replace(":.+", "
    ", regex=True).astype("int")
matches["day_code"] = matches["date"].dt.dayofweek
matches["target"] = matches["result"].map({"W": 1, "D"
    : 0, "L": 2})
```

Código 4.20: Conversão das variáveis em valores numéricos

Com o objetivo de o algoritmo compreender o desempenho das equipas nos últimos jogos, uma função chamada ”rolling_averages”, que tem em consideração os últimos 5 jogos, foi desenvolvida. Uma *rolling averages* é uma métrica que calcula tendências durante curtos períodos, utilizando um conjunto de dados. Esta função terá como parâmetros de entrada uma equipa, as colunas das quais queremos extrair a *performance* e as colunas onde guardaremos esses valores. A primeira tarefa a ser feita nesta função é a ordenação dos jogos da equipa por data, para termos os dados dos jogos por ordem de acontecimento. Com a ajuda da função *DataFrame rolling* da biblioteca *Pandas* é criada a *performance* dos últimos jogos da equipa. Os parâmetros colocados nessa função foram o valor "5", que é o número de jogos que queremos utilizar para a média, e o parâmetro ”closed=left” que faz com que o algoritmo não utilize a semana que vamos prever para o cálculo da *performance*. Essa média que foi calculada e armazenada na variável ”rolling_stats”. Após isto, esses valores são colocados nas novas colunas que foram passadas à função. Por último é necessário apagar os jogos onde não foi possível calcular os ”rolling_averages”. No excerto de código 4.21 é apresentada a função ”rolling_averages”.

```
def rolling_averages(group, cols, new_cols):
    group = group.sort_values("date")
    rolling_stats = group[cols].rolling(5, closed='
        left').mean()
    group[new_cols] = rolling_stats
    group = group.dropna(subset=new_cols)
    return group
```

Código 4.21: Função "rolling_averages"

As colunas que vão ser enviadas à função para determinar a *performance*, são as colunas que foram obtidas no processo de *feature selection*. Para isso, foram criadas colunas, com o mesmo nome, mas acrescentado "_rolling" no fim. Por exemplo, a coluna dos jogos a favor tem o nome "gf", a nova coluna vai ter o nome "gf_rolling". Estas colunas são armazenadas na variável "new_cols".

Esta função tem de ser aplicada por equipa, logo o *dataset* foi dividido por equipa, ou seja, um *dataset* individual para cada equipa. Isto foi feito com uma ferramenta da biblioteca *Pandas* chamada *groupby*. Esta ferramenta divide o *dataset* em vários, consoante o valor de uma coluna específica. Para realizar este processo, para todas as equipas foi usada a função *lambda*, sendo estes dados armazenados na variável "matches_rolling". No Código 4.22 é apresentado o código para criar a *performance* por jogo de todas as equipas. Na Figura 4.8 é apresentado um excerto do *dataset* após a aplicação da função "rolling_averages".

```
matches_rolling = matches.groupby("team").apply(lambda
    x: rolling_averages(x, cols, new_cols))
```

Código 4.22: Código para realizar a *performance* para todas as equipas

team	Column1	date	ga_x	sota	saves	cs	psxg	psxg+/-	team	sh	sot	...	fld_rolling	crs_rolling	int_rolling	tklw_rolling	pkwon_rolling	pkcon_roll
Arouca	481	2021-09-18	2.0	4.0	2.0	0	2.1	0.1	Arouca	11.0	5.0	...	16.6	17.6	14.2	11.0	0.2	
	482	2021-09-25	2.0	3.0	1.0	0	1.8	-0.2	Arouca	17.0	5.0	...	15.6	16.0	13.2	12.2	0.2	
	483	2021-10-02	2.0	8.0	6.0	0	2.5	0.5	Arouca	10.0	5.0	...	16.0	22.0	12.6	12.0	0.2	
	484	2021-10-24	0.0	0.0	0.0	1	0.0	0.0	Arouca	15.0	6.0	...	14.6	20.2	12.4	12.6	0.0	
	485	2021-10-29	0.0	6.0	6.0	1	2.0	2.0	Arouca	2.0	1.0	...	13.8	23.8	10.8	11.8	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
Vizela	607	2024-04-20	2.0	5.0	3.0	0	2.2	0.2	Vizela	5.0	2.0	...	15.0	28.0	8.2	7.0	0.0	
	608	2024-04-27	1.0	3.0	2.0	0	0.7	-0.3	Vizela	19.0	4.0	...	15.8	21.6	7.4	7.6	0.0	
	609	2024-05-03	1.0	5.0	4.0	0	2.2	1.2	Vizela	8.0	2.0	...	14.4	25.2	6.6	9.0	0.0	
	610	2024-05-11	0.0	3.0	3.0	1	0.3	0.3	Vizela	16.0	7.0	...	15.2	25.2	6.0	10.2	0.0	
	611	2024-05-18	2.0	4.0	2.0	0	1.2	-0.8	Vizela	13.0	4.0	...	14.0	24.4	6.6	10.4	0.0	

Figura 4.8: *Dataset* após aplicar a função ”rolling_averages”

Com os dados referentes aos jogos passados no *dataset*, as últimas alterações a serem feitas são retirar a coluna ”team”, que foi criada com a aplicação da função ”rolling_averages”, visto que é repetida, e editar a primeira coluna que serve como *index* para colocar os números únicos em cada jogo. Na Figura 4.9 é apresentado um excerto do *dataset* final após as últimas alterações.

	date	ga_x	sota	saves	cs	psxg	psxg+/-	team	sh	sot	...	fld_rolling	crs_rolling	int_rolling	tklw_rolling	pkwon_rolling	pkcon_rolling	og_rollir
0	2021-09-18	2.0	4.0	2.0	0	2.1	0.1	Arouca	11.0	5.0	...	16.6	17.6	14.2	11.0	0.2	0.2	0
1	2021-09-25	2.0	3.0	1.0	0	1.8	-0.2	Arouca	17.0	5.0	...	15.6	16.0	13.2	12.2	0.2	0.2	0
2	2021-10-02	2.0	8.0	6.0	0	2.5	0.5	Arouca	10.0	5.0	...	16.0	22.0	12.6	12.0	0.2	0.2	0
3	2021-10-24	0.0	0.0	0.0	1	0.0	0.0	Arouca	15.0	6.0	...	14.6	20.2	12.4	12.6	0.0	0.2	0
4	2021-10-29	0.0	6.0	6.0	1	2.0	2.0	Arouca	2.0	1.0	...	13.8	23.8	10.8	11.8	0.0	0.2	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
3532	2024-04-20	2.0	5.0	3.0	0	2.2	0.2	Vizela	5.0	2.0	...	15.0	28.0	8.2	7.0	0.0	0.0	0
3533	2024-04-27	1.0	3.0	2.0	0	0.7	-0.3	Vizela	19.0	4.0	...	15.8	21.6	7.4	7.6	0.0	0.0	0
3534	2024-05-03	1.0	5.0	4.0	0	2.2	1.2	Vizela	8.0	2.0	...	14.4	25.2	6.6	9.0	0.0	0.0	0
3535	2024-05-11	0.0	3.0	3.0	1	0.3	0.3	Vizela	16.0	7.0	...	15.2	25.2	6.0	10.2	0.0	0.0	0
3536	2024-05-18	2.0	4.0	2.0	0	1.2	-0.8	Vizela	13.0	4.0	...	14.0	24.4	6.6	10.4	0.0	0.0	0

Figura 4.9: *Dataset* final

Com todos os dados necessários, está na altura de fazer previsões. Primeiramente é necessário importar a biblioteca do algoritmo a utilizar e da biblioteca *Sklearn* o *accuracy_score* e *precision_score*. Para realizar as previsões foi criada uma função chamada ”Prediction” que recebe como parâmetros o *dataset* e as *features* que vão ser utilizadas para previsão. Nessa função, inicialmente, os dados vão ser divididos em dados de treino e de teste. Essa divisão é feita no dia um de agosto de dois mil e vinte e três, onde os dados de treino são todas as épocas antes de 2023/2024, conforme já referido anteriormente e os dados de teste são os da época 2023/2024.

Com os dados divididos é altura de treinar o algoritmo de *Machine Learning* com os dados de treino, e criar as previsões com os dados de teste. Com as previsões criadas, estas são juntas com o resultado real do jogo e por fim calculada a precisão (Código 4.23).

```
def predictions(data, predictors):
    train = data[data["date"] < '2023-08-01']
    test = data[data["date"] > '2023-08-01']
    bagging.fit(train[predictors], train["target"])
    preds = bagging.predict(test[predictors])
    combined = pd.DataFrame(dict(actual=test["target"
        ], predicted=preds), index=test.index)
    error = precision_score(test["target"], preds,
        average='weighted', zero_division=0)
    return combined, error
```

Código 4.23: Função *Predictions*

As previsões que resultam desta função formam um *dataframe* com duas colunas, a "actual" e a "predicted", logo é necessário acrescentar informação para sabermos a que jogos essas previsões se referem. São então acrescentadas através do *index* da linha, dados presentes no *dataset* "matches_rolling" como "date", "team", "opponent" e "result". Na Figura 4.10 é apresentado um excerto das previsões obtidas.

	actual	predicted	date	team	opponent	result
63	1	2	2023-08-13	Arouca	Estoril	W
64	0	2	2023-08-20	Arouca	Vizela	D
65	0	2	2023-08-26	Arouca	Portimonense	D
66	0	2	2023-09-03	Arouca	Porto	D
67	2	2	2023-09-17	Arouca	Casa Pia	L
...	...	...	...	...	...	...
3532	2	2	2024-04-20	Vizela	Braga	L
3533	0	2	2024-04-27	Vizela	Rio Ave	D
3534	2	2	2024-05-03	Vizela	Moreirense	L
3535	1	2	2024-05-11	Vizela	Estrela	W
3536	0	2	2024-05-18	Vizela	Boavista	D

Figura 4.10: Dados das previsões

Para facilitar a visualização da previsão dos jogos, as duas previsões feitas do mesmo jogo vão estar na mesma linha. Para isto, é necessário resolver um problema do *dataset*, que é o facto de a equipa quando joga em casa por vezes não ter um nome igual ao que usa quando joga fora; por exemplo, o Paços de Ferreira quando joga em casa o seu nome está "Pacos de Ferreira" e quando joga fora está só "Paços". Os nomes foram guardados na variável "new_team". Para obtermos um *dataset* com as duas previsões, é necessário fundir o *dataset* com ele mesmo. Para isso será utilizada a função *merge*, onde vão ser juntas as linhas onde a *data* coincida e a coluna "new_team" seja igual à coluna 'opponent' de outra linha. Na Figura 4.11 é apresentado um excerto das previsões por jogo obtidas.

	actual_x	predicted_x	date	team_x	opponent_x	result_x	new_team_x	actual_y	predicted_y	team_y	opponent_y	result_y	new_team_y
0	1	2	2023-08-13	Arouca	Estoril	W	Arouca	2	2	Estoril	Arouca	L	Estoril
1	0	2	2023-08-20	Arouca	Vizela	D	Arouca	0	2	Vizela	Arouca	D	Vizela
2	0	2	2023-08-26	Arouca	Portimonense	D	Arouca	0	2	Portimonense	Arouca	D	Portimonense
3	0	2	2023-09-03	Arouca	Porto	D	Arouca	0	1	Porto	Arouca	D	Porto
4	2	2	2023-09-17	Arouca	Casa Pia	L	Arouca	1	2	Casa Pia	Arouca	W	Casa Pia
...	...	...	...	...	...	...	...	...	...	...	...	...	...
593	2	2	2024-04-20	Vizela	Braga	L	Vizela	1	1	Braga	Vizela	W	Braga
594	0	2	2024-04-27	Vizela	Rio Ave	D	Vizela	0	2	Rio Ave	Vizela	D	Rio Ave
595	2	2	2024-05-03	Vizela	Moreirense	L	Vizela	1	2	Moreirense	Vizela	W	Moreirense
596	1	2	2024-05-11	Vizela	Estrela	W	Vizela	2	1	Estrela	Vizela	L	Estrela
597	0	2	2024-05-18	Vizela	Boavista	D	Vizela	0	2	Boavista	Vizela	D	Boavista

Figura 4.11: Dados das previsões finais

Capítulo 5

Resultados

Neste capítulo, são apresentados os resultados obtidos para os diversos algoritmos utilizados para testar o modelo desenvolvido de previsão de resultados de futebol.

5.1 Algoritmos Utilizados

Os algoritmos de *Machine Learning* utilizados para o teste do modelo foram o *Logistic Regression*, o *Neural Network*, o *Random Forest*, o *Support Vector Machine*, o *Extreme Gradient Boosting*, o *Bagging Classifier* e o *K-Nearest Neighbors* (KNN). Estes algoritmos obtiveram resultados diferentes na etapa de *Feature Selection*. Alguns deles utilizaram mais *features* e outros menos. O algoritmo que utilizou mais *features* para realizar a previsão utilizou vinte e cinco, já o que menos utilizou precisou de catorze *features*.

Na Figura 5.1 é apresentado uma tabela resumo da utilização de cada *feature* por algoritmo. Com a análise da tabela é possível verificar que o algoritmo *Bagging Classifier* utilizou um total de vinte e cinco *features*, o *Neural Network* utiliza vinte e uma, o *Logistic Regression* vinte e quatro, o *Extreme Gradient Boosting* dezasseis, o *Support Vector Machine* quinze, o *Random Forest* dezassete e o *K-Nearest Neighbors* treze *features*.

Tabela 5.1: *Features* utilizadas por algoritmo

Features	**BAG**	**NN**	**LOGREG**	**XGBoost**	**SVM**	**RF**	**KNN**
crdr_rolling	X	X	X	X	X	X	
crdy_rolling	X	X	X				
crs_rolling	X	X	X	X		X	X
cs_rolling	X	X	X	X	X		X
day_code	X		X			X	X
fld_rolling	X	X	X	X		X	X
fls_rolling	X	X		X			X
fk_rolling	X	X	X		X	X	
ga_rolling	X		X		X	X	
gf_rolling	X	X	X	X	X		X
hour	X		X			X	X
int_rolling	X	X	X	X		X	
og_rolling	X	X	X	X		X	
opp_code	X		X			X	X
pk_rolling	X	X	X	X	X		
pkatt_rolling	X	X	X	X	X		
pkcon_rolling	X	X	X		X	X	
pkwon_rolling	X		X	X	X	X	
poss_rolling	X	X	X	X	X	X	X
saves_rolling	X	X	X	X	X		X
sh_rolling	X	X	X	X	X		
sot_rolling		X					
sota_rolling	X	X	X	X	X	X	X
tklw_rolling	X	X	X	X			X
venue_code	X		X			X	X
xg_rolling	X	X	X		X	X	
xga_rolling		X			X	X	

Na Figura 5.1 é apresentado um gráfico com o número de vezes que cada *feature* foi utilizada pelos algoritmos. Pela análise do gráfico é possível, verificar que as *features* mais utilizadas foram a média de remates da equipa contrária à baliza nos últimos jogos ("sota_rolling") e a média de posse de bola nos últimos jogos ("poss_rolling"), as quais foram utilizadas por todos os algoritmos, e ainda a média de golos marcados nos últimos jogos ("gf_rolling"), a média de defesas realizadas nos últimos jogos ("saves_rolling"), a média de *clean sheets* ("cs_rolling"), a média de vermelhos dos últimos jogos ("crdr_rolling"), a média de faltas sofridas dos últimos jogos ("fld_rolling") e a média de cruzamentos dos últimos jogos ("crs_rolling"), que foram todas utilizadas por 6 dos algoritmos.

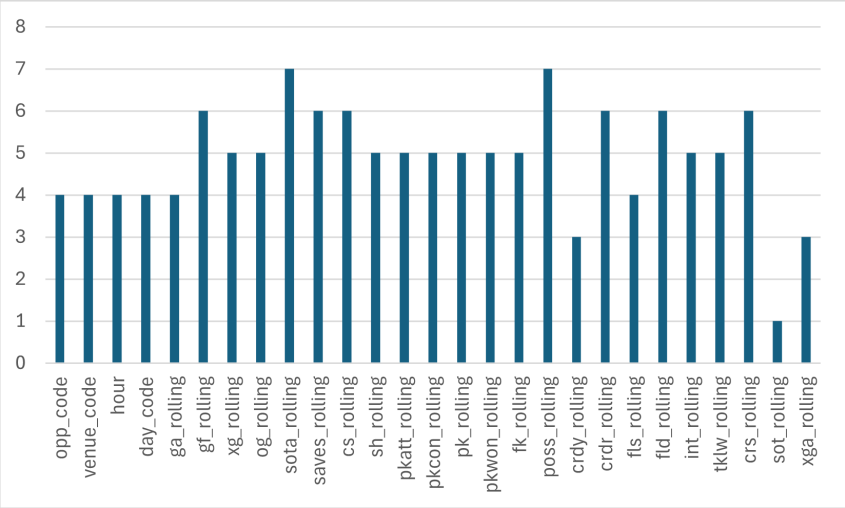


Figura 5.1: Gráfico da utilização das *features*

Os hiperparâmetros dos algoritmos também foram testados através de *Grid Search*. A Tabela 5.2 mostra a gama em que o teste ocorreu, e quais os hiperparâmetros escolhidos para cada algoritmo. A coluna "Variação dos Hiperparâmetros" mostra a faixa de valores em que cada parâmetro foi testado pelo algoritmo, e a coluna "Hiperparâmetros Utilizados" mostra os valores utilizados para cada algoritmo.

Tabela 5.2: Hiperparâmetros de cada um dos algoritmos

Algoritmo	**Variação dos Hiperparâmetros**	**Hiperparâmetros Utilizados**
BAG	$10 \leq n\_estimators \leq 200$	n_estimators = 100
	$0.5 \leq max\_samples \leq 1$	max_samples = 1
	$0.5 \leq max\_samples \leq 1$	max_features = 0.7
	False $\leq$ bootstrap $\leq$ *True*	bootstrap = *False*
	False $\leq$ bootstrap_features $\leq$ *True*	bootstrap_features = *True*
XGBoost	*multi:softprob* $\leq$ *objective* $\leq$ *multi:softmax*	objective = *multi:softprob*
	$50 \leq n\_estimators \leq 200$	n_estimators = 50
	$3 \leq max\_depth \leq 10$	max_depth = 5
	$0.01 \leq learning\_rate \leq 0.2$	learning_rate = 0.01
	$0 \leq \gamma \leq 0.5$	$\gamma = 0$
	$0.6 \leq subsample \leq 1$	subsample = 0.6
	$0.6 \leq colsample\_bytree \leq 1$	colsample_bytree = 0.6
RF	$10 \leq n\_estimators \leq 500$	n_estimators = 200
	$0 \leq max\_depth \leq 30$	max_depth = 10
	$2 \leq min\_samples\_split \leq 10$	min_samples_split = 2
	2$\leq$ min_samples_leaf $\leq$4	min_samples_leaf = 4
	False $\leq$ bootstrap $\leq$ *True*	bootstrap = *True*
SVM	$0.1 \leq C \leq 100$	C = 0.1
	scale $\leq$ *gamma* $\leq$ *auto*	gamma = *auto*
KNN	$3 \leq n\_neighbors \leq 9$	n_neighbors = 9
	uniform $\leq$ *weights* $\leq$ *distance*	weights = *uniform*
NN	$(50,) \leq hidden\_layer\_sizes \leq (100,)$	hidden_layer_sizes = (50,)
	adam $\leq$ *solver* $\leq$ *sgd*	solver = *adam*
	$0.0001 \leq alpha \leq 0.01$	alpha = 0.01
LOGREG	$0.1 \leq C \leq 100$	C = 0.1
	$1000 \leq max_i ter \leq 10000$	max_iter = 2000

5.2 Análise dos Resultados

Na Tabela 5.3 são apresentados os valores de Precisão, Exatidão, *Recall* e *F1 Score* para cada um dos sete algoritmos testados.

Tabela 5.3: Resultados obtidos na previsão

Algoritmo	**Precisão**	**Exatidão**	***Recall***	***F1 Score***
BAG	64,32%	52,00%	52,00%	45,08%
XGBoost	63,39%	48,33%	48,33%	41,60%
RF	55,53%	50,83%	50,83%	44,74%
NN	47,37%	48,66%	48,66%	43,62%
LOGREG	45,83%	49,33%	49,33%	43,11%
SVM	45,62%	49,50%	49,50%	43,73%
KNN	45,13%	45,99%	45,99%	45,08%

Analisando os valores da Tabela 5.3 é possível verificar que o algoritmo *Bagging Classifier* destacou-se dos demais, obtendo melhores resultados em todas as métricas quando comparado com os demais, mostrando-se o mais eficaz na classificação correta dos resultados dos jogos. Comparando agora o XGBoost e RF o primeiro obteve uma melhor precisão o que significa que teve menos falsos positivos. Já o RF obteve melhor exatidão o que indica que fez mais previsões corretas do que o XGBoost, o qual obteve também um maior *recall*, ou seja, detetou mais verdadeiros positivos mesmo que para tal, cria-se mais falsos positivos; tendo-se também verificado que obteve melhor *F1 Score* o que indica um maior equilíbrio entre a precisão e o *recall*. O algoritmo que obteve piores resultados foi o KNN, este fenómeno pode ser explicado pelo facto dos dados terem as classes desequilibradas. O KNN é um classificador baseado em proximidade, o que significa que ele faz previsões com base na classe predominante entre os vizinhos mais próximos. Visto que uma classe é menos representada do que as outras, haverá uma menor probabilidade de que os vizinhos mais próximos pertençam à classe minoritária.

Visto isto, o algoritmo escolhido para ser analisado mais ao pormenor foi o *Bagging Classifier* pois foi o que mais se destacou relativamente aos demais.

Comparando agora os resultados obtidos recorrendo ao algoritmo *Bagging Classifier* com os resultados dos estudos apresentados no estado de arte, é possível afirmar que os resultados obtidos são positivos. No estudo apresentado em [47], este possui dados sobre os jogadores e as suas habilidades, tendo obtido uma exatidão entre 54% e os 58%, não prevendo corretamente nenhum empate. Já no estudo descrito em [46] foram obtidos máximos de exatidão com o valor de 55%. No estudo referido em [43] foram obtidos valores de exatidão para a *English Premier League* de 54,6%. O último estudo que serve de comparação a este trabalho descrito em [45] foram obtidos valores de exatidão de 68% para a liga dos campeões. Este valor relativamente alto pode ser explicado pela abundância de dados que os autores dispunham, tais como a forma atual das equipas com base nos resultados alcançados nos últimos seis jogos, o resultado do encontro anterior entre as equipas envolvidas no jogo, a posição atual no *ranking*, o número de jogadores lesionados das equipas, a média

de golos marcados e sofridos por jogo e a avaliação subjetiva da qualidade de cada equipa por especialistas na área. Analisando a exatidão dos outros estudos e do estudo realizado nesta dissertação, é possível afirmar que os resultados são positivos considerando o reduzido número de dados disponíveis sobre os jogos da Liga Portuguesa e a dificuldade de obtenção de dados avançados, como, por exemplo, a avaliação das equipas e dos jogadores. Na Tabela 5.4 é apresentado um resumo dos vários estudos apresentados e das suas exatidões, bem como do projeto desenvolvido para comparação.

Tabela 5.4: Comparação de resultados

Estudo	**Exatidão**
Artificial intelligence in sports prediction [43]	54,6%
Predicting football scores using machine learning techniques [45]	68,0%
Predicting soccer matches with complex networks and machine learning [46]	55,0%
Neural network algorithm in predicting football match outcome based on player ability index [47]	58,0%
Algoritmo desenvolvido	52,0%

As previsões são realizadas individualmente para cada equipa, o que por vezes cria jogos inválidos, jogos nos quais a previsão indica que ambos ganham, ambos perdem, um ganha, o outro empata ou um perde e o outro empata. Analisando agora as previsões válidas obtidas com o algoritmo *Bagging Classifier*, foram previstas 123 vitórias da equipa da casa, acertando-se em 75 (60,97%) e tendo ocorrido um total de 131; foram previstas 58 vitórias da equipa visitante (derrota da equipa da casa), tendo-se acertando em 38 (60,03%) e ocorrido 95. Não foram previstos corretamente nenhum empate, tendo ocorrido 73. Na Tabela 5.5 são apresentados os dados das vitórias, empates e derrotas previstos e acertados, bem como a percentagem de vitórias e derrotas acertadas face às ocorridas.

Tabela 5.5: Previsões do modelo *Bagging Classifier* para equipa da casa

	Vitória	**Empate**	**Derrota**
Previsto	123	0	58
Acertado	75	0	38
Percentagem	57,25%	-	40,00%

Também aqui se pode comprovar pelas percentagens obtidas que o algoritmo proposto atingiu resultados muito satisfatórios, em particular para as vitórias.

Capítulo 6

Conclusões

Em resumo, esta dissertação demonstrou que é possível realizar previsões de resultados de futebol com algoritmos de *Machine Learning*. Os dados foram obtidos por *Web Scrapping*, visto que, as bases de dados disponíveis não tinham tantos parâmetros de análise como no *website* de onde os dados foram obtidos. Os resultados obtidos são difíceis de comparar com os estudos apresentados na bibliografia, visto que dos estudos referidos nenhum previa resultados sobre a Liga Portuguesa, podendo também os valores de exatidão ser afetados pelas características dos diversos campeonatos. De referir ainda, que conforme documentado, a previsão dos empates demonstrou ser muito difícil, obtendo-se melhores resultados quando apenas se pretende prever vitórias e derrotas. No modelo de classificação desenvolvido no âmbito deste trabalho, procurou-se introduzir este parâmetro, o que se revelou um desafio, tendo originado resultados menos bons, mas estando estes, no entanto, muito próximos dos obtidos em modelos treinados com melhores recursos tais como conjuntos de dados mais completos e robustos. O algoritmo *Bagging Classifier* destacou-se sendo o algoritmo mais eficaz, superando os demais em todas as métricas. O XGBoost obteve maior precisão, enquanto o RF apresentou melhor exatidão e *F1 Score*, indicando um melhor equilíbrio entre precisão e *recall*.

A tarefa de previsão de resultados de futebol provou ser uma tarefa bastante complexa, devido ao facto de não existirem mais dados de épocas anteriores, com estatísticas avançadas como em outros desportos, como, por exemplo, no *Beisebol* onde as estatísticas avançadas tiveram um uso generalizado desde o início do ano 2000 [54].

Em conclusão, o objetivo inicial foi atingido com êxito. Foi desenvolvido um modelo de *Machine Learning* que obteve resultados enquadrados no panorama das previsões de resultados de futebol.

6.1 Trabalho Futuro

Como trabalho futuro, o ideal será obter dados não só sobre os últimos jogos, mas também sobre os jogadores de ambas as equipas, nomeadamente lesões, preparação física, moral, entre outros fatores, que de alguma forma possam influênciar os jogos, os quais não foram tidos em conta neste projeto por falta de dados disponíveis. Outro ponto a explorar seria, não só a *performance* nos últimos jogos, mas também com as *features* disponíveis, criar outras que possam ter impacto na previsão de resultados.

Referências

[1] M. e Melhores, "Quais os 14 tipos de esportes mais populares do mundo?." Available at https://www.maioresemelhores.com/tipos-de-esportes-mais-populares-do-mundo/. [Citado nas páginas 1 e 5]

[2] B.Min, J.Kim, C.Choe, H.Eom, and R.I.(Bob)McKay, "A compound framework for sports results prediction: A football case study," *Knowledge-Based Systems*, vol. 21, no. 7, pp. 551–562, 2008. [Citado na página 1]

[3] E. Blakemore, "Quem inventou o desporto mais popular do mundo?." Available at https://www.nationalgeographic.pt/historia/futebol-inglaterra-popularizou-o-mas-quem-tera-inventado_3906. [Citado na página 6]

[4] C. Kotitschke, "Soccer analytics data: Beginners guide." Available at https://medium.com/the-sports-scientist/soccer-analytics-data-beginners-guide-6cc1b7844792. [Citado na página 7]

[5] K. Warwick, ed., *Artificial intelligence : the basics*. New York: Routledge, 2012. [Citado na página 7]

[6] D.Visvikis, C. L. Rest, V.Jaouen, and M.Hatt, "Artificial intelligence, machine (deep) learning and radio(geno)mics: definitions and nuclear medicine imaging applications," *European Journal of Nuclear Medicine and Molecular Imaging*, vol. 46, no. 13, pp. 2630–2637, 2019. [Citado na página 7]

[7] M. H. et al, "Machine learning and artificial intelligence: Definitions, applications, and future directions," *Current Reviews in Musculoskeletal Medicine*, vol. 13, no. 1, pp. 69–76, 2020. [Citado na página 7]

[8] Mattab, "Artificial intelligence, enough of the hype! what is it?." Available at https://community.hpe.com/t5/hpe-blog-uk-ireland-middle-east/artificial-intelligence-enough-of-the-hype-what-is-it/ba-p/7046672. [Citado nas páginas vii e 7]

[9] I. Naqa and M.J.Murphy, eds., *What Is Machine Learning?* Springer, 2015. [Citado na página 8]

[10] R. Doshi, ed., *MACHINE LEARNING : master supervised and unsupervised learning algorithms with real examples*. Bpb Publications, 2021. [Citado na página 8]

[11] S. Reddysetty, "Traditional programming vs machine learning." Available at `https://sravya-tech-usage.medium.com/traditional-programming-vs-machine-learning-e9bbed5e491c`. [Citado nas páginas vii e 8]

[12] T. J. Jaimie, L. G. Anthony, and J. Rosellini, "Supervised machine learning: A brief primer," *Behavior Therapy*, vol. 51, no. 5, pp. 675–687, 2020. [Citado nas páginas 8 e 13]

[13] S. Suthaharan, ed., *Support Vector Machine, Machine Learning Models and Algorithms for Big Data Classification*. Boston: Springer, pp. 207–235, 2016. [Citado na página 9]

[14] D. Technologies, "What is a support vector machine?." Available at `https://datatron.com/what-is-a-support-vector-machine/`. [Citado nas páginas vii e 9]

[15] A. Abraham, "Meta learning evolutionary artificial neural networks," *Neurocomputing*, vol. 56, pp. 1–38, 2004. [Citado nas páginas vii, 9 e 10]

[16] J. Ijeoma, "A comparative study of support vector machine and logistic regression," *Oriental journal of science & engineering*, vol. 2, pp. 85–91, 2021. [Citado na página 10]

[17] A. Saini, "Logistic regression | what is logistic regression and why do we need it?." Available at `https://acesse.dev/ZpxgS`. [Citado nas páginas vii e 10]

[18] F. J. Yang, "An extended idea about decision trees," *International Conference on Computational Science and Computational Intelligence*, pp. 349–354, 2019. [Citado na página 11]

[19] C. Brooks, "What is a decision tree?." Available at `https://www.businessnewsdaily.com/6147-decision-tree.html`. [Citado nas páginas vii e 11]

[20] A. T. Azar, H. I. Elshazly, A. E. Hassanien, and A. M. Elkorany, "A random forest classifier for lymph diseases," *Computer Methods and Programs in Biomedicine*, vol. 2, pp. 465–473, 2014. [Citado na página 11]

[21] D. Petkovic, R. Altman, M. Wong, and A. Vigil, eds., *Improving the explainability of Random Forest classifier – user centered approach*. Biocomputing, 2017. [Citado na página 11]

[22] D. Gunay, "Random forest." Available at `https://medium.com/@denizgunay/random-forest-af5bde5d7e1e`. [Citado nas páginas vii e 12]

[23] Y. Qiu, J. Zhou, M. Khandelwal, H. Yang, P. Yang, and C. Li, "Performance evaluation of hybrid woa-xgboost, gwo-xgboost and bo-xgboost models to predict blast-induced ground vibration," *Engineering with Computers*, 2021. [Citado na página 12]

[24] J. Zhang, X. Ma, J. Zhang, D. Sun, X. Zhou, C. Mi, and H. Wen, "Insights into geospatial heterogeneity of landslide susceptibility based on the shap-xgboost model," *Journal of Environmental Management*, vol. 332, 2023. [Citado na página 12]

[25] Yao, X. . Fu, X. . Zong, and Chaofei, "Short-term load forecasting method based on feature preference strategy and lightgbm-xgboost," *IEEE Access*, 2022. [Citado nas páginas vii e 13]

[26] Z. Huang, "Extensions to the k-means algorithm for clustering large data sets with categorical values," *Data Mining and Knowledge Discovery*, vol. 2, pp. 283–304, 1998. [Citado na página 14]

[27] S. Na, L. Xumin, and G. Yong, "Research on k-means clustering algorithm: An improved k-means clustering algorithm," *Third International Symposium on Intelligent Information Technology and Security Informatics*, pp. 63–67, 2010. [Citado na página 14]

[28] A. Jeffares, "K-means: A complete introduction." Available at `https://towardsdatascience.com/k-means-a-complete-introduction-1702af9cd8c`. [Citado nas páginas vii e 14]

[29] C. F. Olson, "Parallel algorithms for hierarchical clustering," *Parallel Computing*, vol. 21, pp. 1313–1325, 1995. [Citado na página 14]

[30] J. Han, M. Kamber, and J. Pei, eds., *Data Mining: Concepts and Techniques*. Elsevier, pp. 443-495, 2011. [Citado na página 14]

[31] Z. Keita, "An introduction to hierarchical clustering in python." Available at `https://www.datacamp.com/tutorial/introduction-hierarchical-clustering-python`. [Citado nas páginas vii e 15]

[32] P. Geladi and J. Linderholm, "Principal component analysis," *ScienceDirect*, pp. 17–37, 2020. [Citado na página 15]

[33] M. Nashed, "Principal component analysis (pca) transformation | biorender science templates." Available at https://www.biorender.com/template/principal-component-analysis-pca-transformation. [Citado nas páginas vii e 15]

[34] A. Hyvärinen, "Independent component analysis: recent advances," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 371, 1984. [Citado na página 16]

[35] R. da M. C. Carvalho, "The power of independent component analysis (ica) on real-world applications — egg example." Available at https://encr.pw/VqzvM. [Citado nas páginas vii e 16]

[36] H. Choi, M. Kim, G. Lee, and W. Kim, "Unsupervised learning approach for network intrusion detection system using autoencoders," *The Journal of Supercomputing*, vol. 75, pp. 5597–5621, 2019. [Citado na página 16]

[37] W. Badr, "Auto-encoder: What is it? and what is it used for?." Available at https://towardsdatascience.com/auto-encoder-what-is-it-and-what-is-it-used-for-part-1-3e5c6f017726. [Citado na página 17]

[38] F. Chollet, "Building autoencoders in keras.." Available at https://blog.keras.io/building-autoencoders-in-keras.html. [Citado nas páginas vii e 17]

[39] M. Khder, "Web scraping or web crawling: State of art, techniques, approaches and application," *International Journal of Advances in Soft Computing and its Applications*, vol. 13, pp. 145–168, 2021. [Citado na página 17]

[40] A. Networks, "What is web scraping?." Available at https://avinetworks.com/glossary/web-scraping/. [Citado nas páginas vii e 17]

[41] A. J. Herrera, ed., *Ryan Mitchell Web Scraping with Python COLLECTING MORE DATA FROM THE MODERN WEB*. O'Reilly, 2018. [Citado na página 17]

[42] Y. Dikilitaş, Ç. Çakal, A. C. Okumuş, H. N. Yalçın, E. Yıldırım, Ö. F. Ulusoy, B. Macit, A. E. Kırkaya, Ö. Yalçın, E. Erdoğmuş, and A. Sayar, "Performance Analysis for Web Scraping Tools: Case Studies on Beautifulsoup, Scrapy, Htmlunit and Jsoup," in *Emerging Trends and Applications in Artificial Intelligence*, pp. 471–480, Cham, Switzerland: Springer, Apr. 2024. [Citado na página 18]

[43] A. McCabe and J. Trevathan, "Artificial intelligence in sports prediction," *Fifth International Conference on Information Technology: New Generations*, 2008. [Citado nas páginas 18, 53 e 54]

[44] R. Baboota and H. Kaur, "Predictive analysis and modelling football results using machine learning approach for english premier league," *International Journal of Forecasting*, vol. 35, 2019. [Citado na página 18]

[45] J. Hucaljuk and A. Rakipović, "Predicting football scores using machine learning techniques," *Proceedings of the 34th International Convention MIPRO*, 2011. [Citado nas páginas 19, 53 e 54]

[46] E. A. B. e Felipe Jordão Xavier e Thomas Peron e Paulino Ribeiro Villas-Boas e Francisco Aparecido Rodrigues, "Predicting soccer matches with complex networks and machine learning." Available at `https://arxiv.org/html/2409.13098v1`. [Citado nas páginas 20, 53 e 54]

[47] H. Chen, "Neural network algorithm in predicting football match outcome based on player ability index," *Advances in Physical Education*, vol. 9, 2019. [Citado nas páginas 20, 53 e 54]

[48] A. Zimmermann, S. Moorthy, and Z. Shi, "Predicting ncaab match outcomes using ml techniques - some results and lessons learned." Available at `https://arxiv.org/abs/1310.3607`. [Citado na página 20]

[49] Tapendrakumar, "Sequential feature selection." Available at `https://www.geeksforgeeks.org/sequential-feature-selection/`. [Citado na página 27]

[50] Siddiqi, M. . Alsayat, A. . Alhwaiti, Y. . Azad, M. . Alruwaili, M. . Alanazi, S. . Kamruzzaman, M. . Khan, and Asfandyar, "A precise medical imaging approach for brain mri image classification," *Computational Intelligence and Neuroscience*, pp. 1–15, 2022. [Citado nas páginas vii e 28]

[51] G. L. E. Team, "Hyperparameter tuning with gridsearchcv." Available at `https://www.mygreatlearning.com/blog/gridsearchcv/`. [Citado na página 28]

[52] Simplilearn, "Python requests: Here's everything you should know." Available at `https://www.simplilearn.com/tutorials/python-tutorial/python-requests`. [Citado na página 32]

[53] M. Breuss, "Beautiful soup: Build a web scraper with python." Available at `https://realpython.com/beautiful-soup-web-scraper-python/`. [Citado na página 32]

[54] T.-M. Warning, "Baseball e as estatísticas." Available at `https://two-minutewarning.blogspot.com/2012/08/baseball-e-as-estatisticas.html`. [Citado na página 55]