Gabriel Goes Braga Takayanagi, Guilherme Rodrigues Bastos, Tomas Gorescu Caldeira

Modelo de predição de resultados de partidas de futebol

São Paulo, SP

2023

Gabriel Goes Braga Takayanagi, Guilherme Rodrigues Bastos, Tomas Gorescu Caldeira

Modelo de predição de resultados de partidas de futebol

Trabalho de conclusão de curso apresentado ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro.

Universidade de São Paulo – USP

Escola Politécnica

Departamento de Engenharia de Computação e Sistemas Digitais (PCS)

Orientador: Prof. Dr. Jorge Rady de Almeida Junior

São Paulo, SP

2023

Este trabalho é dedicado ao futebol, esporte maravilhoso que nos proporciona muita emoção.

Resumo

Nessa projeto se procurou criar uma plataforma integrada de predição de resultado de partidas, focando em duas frentes: a criação de um modelo preditor feito por aprendizado supervisionado e a criação de um aplicativo que seria utilizado para a transmissão dessas informações para o usuário. Além da da predição dos resultados, fizemos uma simulação de apostas com o resultado do modelo, obtendo resultados que sinalizam que com mais tempo investido a aplicação do modelo nesse contexto é promissora.

Palavras-chave: Aprendizado de máquina. Predição de partidas de futebol. Repositório de partidas de futebol e predições.

Abstract

With this project, we wanted to create an integrated soccer match predictor. It was built using machine learning techniques, that predicts the matches results and sends it to a mobile application, through an API, where the data is displayed. We thought of the mobile application as an easy, fast way to access the predictions. Also, we explored application in a betting enviromnent showing promising potencial with further investimen.

Keywords: Machine Learning. Soccer matches predictions. Repository of soccer matches predictions and outcomes.

Lista de ilustrações

Lista de quadros

Lista de tabelas

Sumário

1 Introdução

Este trabalho foi realizado com o intuito de preparar um aplicativo móvel que seja um preditor de resultados de partidas de futebol. Ele consiste na construção de um modelo, e um aplicativo móvel para disponibilizar os resultados. Além disso, simulamos o desempenho deste modelo como apostador, para comprar a acurácia obtida e estudar uma possível exploração financeira.

1.1 Motivação

O mercado de apostas esportivas tem crescido em todo o mundo de maneira assombrosa. Em matéria recente, o Poder360 afirma que o mercado brasileiro corresponde a 100 bilhões de reais por ano. Segundo matéria da Veja, o mercado de apostas americano cresceu 98% em 2022, totalizando mais de 79,6 bilhões de dólares em transações.

Com esse boom, as casas de apostas se apoiam em modelos construídos às pressas para suprir a demanda crescente.

Muito desse grande volume de apostas é concentrado no esporte mais popular do mundo, o futebol, que contribui com 30% do volume total das receitas esportivas globais, segundo o mesmo artigo da Veja. Pensamos que há uma grande possibilidade de exploração econômica no estudo e aprimoramento em modelos de predição de resultados de partidas esportivas, principalmente no futebol.

Com o grande avanço da Inteligência Artificial das últimas décadas, aumentou-se também a disponibilidade de dados detalhados de partidas de futebol. As grandes ligas do mundo tem feito parcerias com Big Techs para fazer a colheita automática desses dados, sem a necessidade de um observador humano para coletá-los.

A La Liga- primeira divisão da liga de futebol da Espanha, por exemplo, fez uma parceria com a Microsoft, formando a Beyond Stats, que capta dados bastante complexos- como mapas de calor do espaço ocupado por cada jogador durante uma partida.

Quando falamos da criação de modelos preditivos, o resultado obtido é altamente acoplado à quantidade e qualidade dos dados. Nosso trabalho pretende partir desse grande volume de dados captados de maneira automatizada, criando um algoritmo simples, mas com grande acurácia.

1.2 Objetivos

O objetivo deste projeto é desenvolver um algoritmo de previsão de resultados de partidas de futebol. É este um esporte complexo e imprevisível, mas com o uso adequado de técnicas de aprendizado de máquina, podemos buscar *insights* valiosos e aumentar a precisão das previsões.

Neste projeto, utilizaremos a técnica de Gradient Boosted Decision Trees (GBDT), um algoritmo de aprendizado de máquina que combina múltiplas árvores de decisão fracas para criar um modelo mais robusto e preciso. O GBDT é especialmente adequado para lidar com dados heterogêneos e complexos, como aqueles encontrados em partidas de futebol, onde vários fatores podem influenciar o resultado final.

1.3 Justificativa

Resumimos as causas proeminentes do projeto em alguns itens:

- Tomada de Decisões Informadas: A previsão de resultados de partidas de futebol pode fornecer informações valiosas para equipes, treinadores e analistas esportivos. Com base nessas previsões, eles podem tomar decisões informadas sobre estratégias de jogo, escalações, substituições e outras táticas que podem aumentar as chances de vitória. O peso final das regras utilizadas pode dizer muito sobre o que é mais relevante para o resultado final de uma partida.

- Análise do esporte: Esse tipo de estudo pode gerar um estudo, um espectro da mudança do futebol ao longo dos anos, por conter a suma do que é mais relevante para o desempenho de uma equipe.

- Apostas Esportivas: A indústria de apostas esportivas movimenta bilhões de dólares em todo o mundo. Os modelos de previsão de resultados de partidas de futebol são amplamente utilizados por apostadores profissionais para tomar decisões mais fundamentadas ao fazer suas apostas. Esses modelos podem ajudar a identificar oportunidades de apostas com maior probabilidade de sucesso, melhorando a lucratividade dos apostadores, ou regulando as chances e pagamentos oferecidos pelas casas de apostas.

- Entretenimento e Engajamento: O futebol é um esporte apaixonante e imprevisível, e as previsões de resultados de partidas podem adicionar um elemento extra de emoção e engajamento para os torcedores. Algoritmos de previsão podem ser usados para criar jogos, competições e aplicativos interativos que permitem aos torcedores testar suas habilidades de previsão e competir entre si.

- Avanço da Tecnologia e Aprendizado de Máquina: O desenvolvimento de algoritmos de previsão de resultados de partidas de futebol é um campo de pesquisa e aplicação em constante evolução. Ao explorar técnicas avançadas de aprendizado de máquina, como o Gradient Boosted Decision Trees, é possível impulsionar a inovação tecnológica e contribuir para o avanço da área de inteligência artificial aplicada ao esporte. .

1.4 Organização do Trabalho

Nos capítulos seguintes entraremos em detalhe na concepção do projeto, na teoria envolvida nos modelos, sua implementação e detalhes do aplicativo.

No Capítulo 2 apresentaremos todos os detalhes sobre o modelo preditor de resultados e seu embasamento teórico.

No Capítulo 3 falaremos sobre a metodoliga utilizada na coleta, tratamento, modelagem e avaliação dos dados.

No Capítulo 4 detalhamos um pouco mais do que buscamos alcançar com o projeto e detalhamos os requisitos do modelo e do aplicativo.

No Capítulo 5 documentamos passo a passo o desenvolvimento do trabalho, explicando as tecnologias utilizadas, detalhes do banco de dados, da construção do aplicativo e do treinamento do modelo.

2 Aspectos Conceituais

Aprendizado de máquina (machine learning) é um termo geral utilizado para definir uma série de algoritmos que extraem informação a partir de um conjunto de dados, sem ser necessário definir um modelo matemático específico. A partir de um conjunto de dados de treinamento, estes algoritmos buscam um padrão relacionando entradas e saídas, permitindo utilizar este padrão para realizar predições.

2.1 Aprendizado Supervisionado

Os algoritmos de aprendizagem supervisionada relacionam uma saída com uma entrada com base em dados rotulados. Neste caso, o usuário alimenta ao algoritmo pares de entradas e saídas conhecidos, normalmente na forma de vetores. Para cada saída é atribuido um rótulo, que pode ser um valor numérico ou uma classe. O algoritmo determina uma forma de prever qual o rótulo de saída com base em uma entrada informada Entre esses algoritmos, os mais comuns são os algoritmos de classificação, onde a saída pode assumir somente um conjunto de rótulos pré-definidos (e não um valor qualquer) e os de regressão que, por sua vez, possui uma saída que pode assumir qualquer valor real. Algoritmos de classificação são algoritmos de aprendizagem supervisionada onde o objetivo é prever uma classe ou rótulo associado com uma variável de entrada contendo determinados atributos. Inicialmente, o algoritmo é treinado com um conjunto de dados com classes conhecidas, podendo estes dados estar divididos em somente duas (classificação binária) ou em várias classes (classificação multiclasse).

2.2 Decision Tree

Uma ‘decision tree’ é um algoritmo de aprendizado supervisionado não paramétrico que utiliza tanto classificação quanto regressão. É uma estrutura de árvore hierárquica composta por um nó de raiz, galhos, nós internos e nós folha. Iniciada pelo nó raiz, conectam-se os nós internos (também conhecidos como nós de decisão) e esses nós ligam nos nós folhas (nós terminais). Nesse processo é conduzido processamentos para formar subsets homogêneos no fim. Os nós folhas representam todas as possíveis saídas do dataset. Esse tipo de estrutura ‘flowchart’ cria uma representação de fácil entendimento que auxilia na tomada de decisão. Para montar essa árvore, é necessário identificar os melhores pontos para fazer as separações. Assim, utilizando um método recursivo, a maioria dos dados são classificados nas suas respectivas categorias. Quanto mais complexa a árvore, mais difícil é ter nós folhas puros e homogêneos. Para evitar “overfitting” e fragmentação dos

dados, geralmente são utilizadas árvores menores e mais consistentes. Esse assunto pode ser explorado com mais detalhes no artigo "Prediction performance of improved decision tree-based algorithms: a review"(MIENYE, 2019)

2.3 Entropia/Information Gain

É difícil explicar o ganho de informação sem discutir primeiro a entropia. A entropia é um conceito que deriva da teoria da informação, que mede a impureza dos valores da amostra. É definida pela seguinte fórmula:

$$\text{Entropy}(S) = -\sum_{c \in C} p(c)\log_2 p(c)$$

Figura 1 – Fomula da entropia

S representa o conjunto de dados para o qual a entropia é calculada c representa as classes do conjunto, S p(c) representa a proporção de pontos de dados que pertencem à classe c em relação ao número total de pontos de dados do conjunto, S

Os valores de entropia podem variar entre 0 e 1. Se todas as amostras do conjunto de dados S pertencerem a uma classe, a entropia será igual a zero. Se metade das amostras forem classificadas numa classe e a outra metade noutra classe, a entropia será igual a 1. Para selecionar a melhor caraterística para dividir e encontrar a árvore de decisão ideal, deve ser utilizado o atributo com a menor quantidade de entropia. O ganho de informação representa a diferença na entropia antes e depois de uma divisão num determinado atributo. O atributo com o maior ganho de informação produzirá a melhor divisão, pois está a fazer o melhor trabalho na classificação dos dados de treino de acordo com a sua classificação alvo. O ganho de informação é geralmente representado pela seguinte fórmula:

$$\text{Information Gain}(S,a) = \text{Entropy}(S) - \sum_{v \in values(a)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

Figura 2 – Ganho de informação

2.4 Random Forests

Os métodos de aprendizagem de conjuntos são constituídos por um conjunto de classificadores - por exemplo, árvores de decisão - e as suas previsões são agregadas

para identificar o resultado mais popular. Os métodos de conjunto mais conhecidos são o bagging, também conhecido como agregação bootstrap, e o boosting. Em 1996, Leo Breiman introduziu o método de ensacamento; neste método, uma amostra aleatória de dados num conjunto de treino é selecionada com substituição, o que significa que os pontos de dados individuais podem ser escolhidos mais do que uma vez. Depois de geradas várias amostras de dados, estes modelos são então treinados independentemente e, dependendo do tipo de tarefa - ou seja, regressão ou classificação - a média ou a maioria dessas previsões produzem uma estimativa mais exacta. Esta abordagem é normalmente utilizada para reduzir a variância num conjunto de dados com ruído. O algoritmo de floresta aleatória é uma extensão do método de ensacamento, uma vez que utiliza tanto o ensacamento como a aleatoriedade das características para criar uma floresta não correlacionada de árvores de decisão. A aleatoriedade das características, também conhecida como ensacamento de características ou "o método do subespaço aleatório", gera um subconjunto aleatório de características, o que garante uma baixa correlação entre as árvores de decisão. Esta é uma diferença fundamental entre as árvores de decisão e as florestas aleatórias. Enquanto as árvores de decisão consideram todas as divisões de características possíveis, as florestas aleatórias selecionam apenas um subconjunto dessas características. Os algoritmos de floresta aleatória têm três hiperparâmetros principais, que têm de ser definidos antes da formação. Estes incluem o tamanho do nó, o número de árvores e o número de características amostradas. A partir daí, o classificador de floresta aleatória pode ser utilizado para resolver problemas de regressão ou classificação. O algoritmo de floresta aleatória é composto por uma coleção de árvores de decisão e cada árvore do conjunto é composta por uma amostra de dados retirada de um conjunto de treino com substituição, denominada amostra de bootstrap. Dessa amostra de treinamento, um terço é reservado como dados de teste, conhecido como amostra fora do saco (oob), ao qual voltaremos mais tarde. Outra instância de aleatoriedade é então injectada através do ensacamento de características, acrescentando mais diversidade ao conjunto de dados e reduzindo a correlação entre as árvores de decisão. Dependendo do tipo de problema, a determinação da previsão varia. Para uma tarefa de regressão, será calculada a média das árvores de decisão individuais e, para uma tarefa de classificação, um voto maioritário - ou seja, a variável categórica mais frequente - produzirá a classe prevista. Finalmente, a amostra oob é usada para validação cruzada, finalizando a previsão.

2.4.1 Key Benefits

Redução do risco de sobreajuste: As árvores de decisão correm o risco de sobreajuste, uma vez que tendem a ajustar com rigor todas as amostras nos dados de treino. No entanto, quando existe um número robusto de árvores de decisão numa floresta aleatória, o classificador não se ajusta excessivamente ao modelo, uma vez que a média das árvores não correlacionadas reduz a variância global e o erro de previsão. Oferece flexibilidade:

Uma vez que a floresta aleatória pode lidar com tarefas de regressão e classificação com um elevado grau de precisão, é um método popular entre os cientistas de dados. O ensacamento de características também torna o classificador de floresta aleatória uma ferramenta eficaz para estimar valores em falta, uma vez que mantém a precisão quando uma parte dos dados está em falta. Fácil de determinar a importância das características: A floresta aleatória facilita a avaliação da importância da variável, ou contribuição, para o modelo. Existem algumas formas de avaliar a importância das características. A importância de Gini e a diminuição média da impureza (MDI) são geralmente usadas para medir o quanto a precisão do modelo diminui quando uma determinada variável é excluída. No entanto, a importância da permutação, também conhecida como precisão da diminuição média (MDA), é outra medida de importância. A MDA identifica a diminuição média da precisão através da permutação aleatória dos valores das características em amostras "out of bag".

2.4.2 Key Challenges

Processo moroso: Uma vez que os algoritmos de floresta aleatória podem lidar com grandes conjuntos de dados, podem fornecer previsões mais exatas, mas podem ser lentos a processar dados, uma vez que estão a calcular dados para cada árvore de decisão individual. Requer mais recursos: Uma vez que as florestas aleatórias processam conjuntos de dados maiores, necessitam de mais recursos para armazenar esses dados. Mais complexa: a previsão de uma única árvore de decisão é mais fácil de interpretar quando comparada com uma floresta de árvores.

2.5 AdaBoost

Um algoritmo que pode ser usado em combinação com árvores de decisão para obter bons resultados é chamado de AdaBoost. AdaBoost é uma técnica que geralmente combina vários algoritmos simples também conhecidos como weak learners de tal forma a cada algoritmo prevenir os erros de classificação dos outros. E é conhecido por obter bons resultados, principalmente em problemas de classificação binários. O resultado final do AdaBoost é uma somatória de n algoritmos, cada um com peso específico calculado.

$$E_t = \sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)]$$

O funcionamento AdaBoost pode ser descrito com três principais objetivos:

- Determinar um conjunto de algoritmos fracos para classificar o problemas
- Determinar o peso que cada um desses algoritmos terá na solução

- Garantir que casos classificados erroneamente por algum dos algoritmo fraco sejam compensadas em outros

No contexto de árvores de decisão, os algoritmos simples usados são um nó com duas folhas, também conhecido como stumps, um exemplo de stump está ilustrado na figura n.

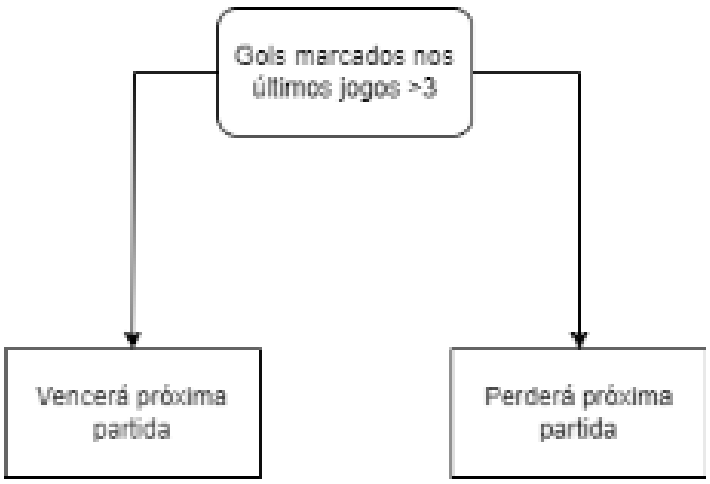

Figura 3 – Exemplo de Stump

Dado uma floresta de stumps podemos definir os stumps que mais ajudam na classificação a partir de métodos como Information Gain, descrita acima. Uma vez definido o stump que será utilizado é necessário calcular o efeito que ele terá no algoritmo. Para o cálculo desse peso é derivado a função de erro com o intuito de minimizar a somatória da perda exponencial, que na fórmula do peso alpha de:

$$\alpha_m = \frac{1}{2} \ln\left(\frac{1 - \epsilon_m}{\epsilon_m}\right)$$

Figura 4 – Calculo do peso alpha

Uma vez completado esse ciclo teremos um stump definido e um peso relativo a ele. Assim precisamos definir os próximos stumps a serem iterados e seus respectivos pesos. Para isso é necessário escalar os casos em que o modelo não é um bom preditor, para isso usamos o mesmo peso w. O peso define, em iterações consequentes, qual o impacto é um

caso específico terá na escolha do próximo stump e seu peso. Priorizando os casos em que o modelo pior prediz, usando a função de peso alpha que não é nada mais que uma forma de calcular o inverso do erro.

$$w_{i,t+1} = w_{i,t} e^{-y_i \alpha_t h_t(x_i)}$$

Figura 5 – Calculo do peso w

2.6 Gradient Boosted Decision Trees

Similarmente ao Adaboost, Gradient Boosted Decision Trees (GBDT) inicia com a criação de uma árvore de decisão inicial, ou árvore fraca, para prever a saída desejada. A diferença chave reside no método de aprimoramento, pois o GBDT utiliza o gradiente descendente para minimizar uma função de perda diferenciável, em vez de focar exclusivamente na atualização de pesos das instâncias de treinamento. Essa abordagem iterativa envolve a construção de árvores subsequentes, cada uma corrigindo os resíduos do modelo anterior. Em geral, GBDT costumam ser mais populares e de uso mais comum que Adaboost. Mais detalhes podem ser encontrados na publicação "AdaBoost, Gradient Boosting, XG Boost: Similarities and Differences"presente no site Medium (BEAST", 2023).

3 Método do trabalho

3.1 Escolha da metodologia

Como base para a o entendimento do CRISP-DM foram utilizados dois artigos "CRISP-DM Twenty years Later: From data mining processes to data science trajectories" e “CRISP-DM: Towards a Standard Process Model for Data Mining”. Para o escopo desse projeto, o intuito será aplicar a metodologia referente a mineração e ciência de dados CRISP-DM, com adaptações necessárias para a inclusão de algumas e alteração de outras etapas que possam ser consideradas úteis para o desenvolvimento do processo. O CRISP-DM (Cross Industry Standard Process for Data Mining) surgiu em 1999 e procurava, em um contexto de cada vez maior demanda de extração de valor de dados sem necessariamente uma prática global da indústria, para criar uma metodologia de forma a deixar esse projetos mais consistentes e menos dependentes de apenas do projetista. Com isso, foi criado um processo em que se acentua as boas práticas, enfatizando a comunicação entre os desenvolvedores do projeto e os clientes, a transparência e acompanhamento contínuo do projeto, possibilitando entender a trajetória do projeto antes de ele ser finalizado. Essa metodologia foi recebida com sucesso e se tornou uma prática um dos mais proeminentes métodos de desenvolvimento para mineração de dados, até hoje em dia é considerado uma das metodologias mais utilizadas na indústria. No cenário atual, existem novas metodologias de mineração de dados, muitas delas inspiradas no CRISP-DM, como por exemplo a Foundational Methodology for Data Science da IBM, mas devido a grande influência nas metodologias mais modernas e do uso difundido do CRISP-DM foi decidido utilizá-lo.

3.2 Metodo

O CRISP-DM é separado em 6 fases iterativas, as quais, dependendo das descobertas feitas, é possível retornar e ajustar as fases anteriores. As fases são subdivididas em diversas etapas com entregáveis sugeridos. Para o escopo desse projeto, devido ao contexto de contato com o orientados, os entregáveis vão ser adaptados de forma a condizer com a maior interatividade que ocorre entre o orientador e orientandos, em oposição aos clientes e projetistas, que é o contexto da indústria.

Figura 6 – As fases do CRISP-DM

3.2.1 Fase de entendimento do negócio

Durante essa fase será elaborado o escopo do projeto, levantando possíveis ferramentas que podem ser utilizadas, direcionamento da solução, além da infraestrutura de entrega, definindo exatamente o que e como será entregue. Para o realizamento dessa fase serão feitas reuniões com o professor orientador e exploração de projetos similares. O entregável dessa etapa será um relatório contendo motivações, objetivos, os aspectos conceituais e a metodologia.

3.2.2 Fase de entendimento dos dados

A fase de entendimento de dados se trata da etapa inicial, durante essa etapa serão coletados os dados dos sites com as estatísticas e informações dos jogadores e times. Será listado exatamente que dados foram obtidos e a estrutura das tabelas desses dados e finalmente será feito um exploratório desses dados, ganhando familiaridade com esses dados, procurando entendê-los de forma mais profunda que inicialmente. Finalmente, será registrado possíveis falhas nos dados e como possivelmente se pode corrigi-las.

As etapas dessa fase são:

- Coleta dos dados iniciais
- Descrição dos dados

- Exploração dos dados
- Verificação da qualidade dos dados

A entrega dessa fase são os dados extraídos, suas tabelas e as observações feitas em cima dele.

3.2.3 Preparação dos dados

A preparação de dados se trata do refinamento final para os dados que serão usados no modelo. Ou seja, durante essa fase será tanto limpado os dados e transformados como serão separados os dados mais relevantes para o modelo.

- Seleção dos dados
- Limpeza dos dados
- Construção dos dados
- Integração dos dados
- Formatação dos dados

A entrega dessa fase são os dados já em uma base de dados, tabelas com processamentos mais complexos e observações de dados mais relevantes para o uso nos modelos.

3.2.4 Modelagem

A Etapa de Modelagem é onde serão criados diversos modelos e esses modelos serão treinados de acordo com os dados preparados na fase de preparação dos dados. Cada problema de Data Mining tem suas peculiaridades e pode ser necessário mais de uma técnica para conseguir explorar os dados completamente. Essa etapa está diretamente ligada com a etapa de Preparação dos Dados já que determinados modelos podem exigir dados em diferentes formatos, sendo possível retornar para a etapa anterior se necessário. As etapas dessa fase são:

- Selecionar a técnica de modelagem
- Gerar o design de Teste
- Construir o Modelo
- Aferir o Modelo

A entrega dessa fase são as especificações dos modelos testados e seus resultados.

3.2.5 Avaliação

Durante essa fase será aferido os resultados obtidos pelos modelos, acertos e erros do processo. Avaliando o resultado do modelo final, os insights obtidos e os aprendizados do processo e serão avaliados os possíveis próximos passos, sejam eles de retorno ao se alguma fase deveria ser refeita ou se deverá ser feito o deploy. As etapas dessa fase são:

- Avaliar resultados
- Revisar o processo
- Determinar os próximos passos

A entrega dessa fase é um relatório detalhando os aprendizados e as especificações do modelo.

3.2.6 Deployment

A última etapa é o deployment, onde será finalmente entregue o modelo de forma funcional e automatizada com seu funcionamento. Assim como o planejamento de como mantê-lo e finalmente o relatório final explicitando os últimos detalhes, aprendizados e funcionamentos que não foram cobertos na etapa anterior. Nota-se que nesse caso não ocorrerá manutenção ou planejamento da mesma devido ao fato de que o produto final não será realmente utilizado. As etapas dessa fase são:

- Planejamento
- Planejamento de monitoramento e manutenção
- Produção do relatório final
- Revisão do projeto

A entrega dessa fase é o relatório final e o produto completo.

4 Especificação de Requisitos

A especificação do projeto está composta inicialmente no presente documento pelos requisitos funcionais e não funcionais descritos abaixo. Além deles, foi adicionado a arquitetura da base de dados e detalhes do aplicativo.

4.1 Requisitos

A especificação abaixo tem o intuito de mapear e direcionar o desenvolvimento do projeto. Os requisitos foram divididos em funcionais e não funcionais, sendo os funcionais aqueles que necessitam que algo seja realizado ativamente enquanto os não funcionais focam mais na implementação.

4.1.1 Requisitos Funcionais

As tarefas essenciais que a solução cumpre foram listadas abaixo:

- O software permite a consulta de resultados passados.
- O software permite que o usuário escolha o campeonato, liga ou torneio de seu interesse.
- O software exibe ao usuário valores calculados em seu modelo com a intenção de auxílio à apostas.

4.1.2 Requisitos Não Funcionais

Para garantir a execução dos itens listados anteriormente, se mostram necessários, os seguintes requisitos não funcionais:

- Dados atualizados referente ao(s) campeonato(s) disponível(is).
- Alta disponibilidade do sistema.
- Interface simples e intuitiva
- Portabilidade para diversos sistemas operacionais

4.1.3 Arquitetura da Base de Dados

A base de dados consiste em dados extraídos via *webscrapping* processados com um código Python e armazenado em tabelas no Supabase. Foram criadas quatro tabelas principais descritas abaixo:

4.1.3.1 Tabela Match

Essa é a tabela mais crua que servirá como base para a solução, nela estão presentes informações como ID das partidas, Data e informações do jogo em questão.

4.1.3.2 Tabela Treated-Match

Com o objetivo de separar a tabela crua por times e filtrar as métricas que serão utilizadas foi criada a tabela Treated-Match.

4.1.3.3 Tabela Round-Stats

A etapa seguinte foi agregar as *features* por time no round adicionando a média de seus oponentes. Para isso foi criada a tabela Round-Stats que consiste na média das métricas definidas anteriormente, tanto no campeonato todo quanto apenas nos jogos mais recentes e a média das mesmas métricas dos adversários.

4.1.3.4 Tabela Features

Essa tabela é responsável por agregar os valores presentes na tabela Round-Stats respectivos à cada um dos dois times em questão.

4.1.4 Design do Aplicativo

O aplicativo é dividido em um grupo pequeno de telas, que se associa diretamente a exibição de nossos dados. Consideramos como requisitos para o aplicativo:

- Permite que o usuário veja dados históricos de partidas do campeonato brasileiro
- Permite que o usuário filtre as partidas por temporada, para agilizar a busca dos dados
- Facilite a visualização das partidas mais recentes, visto que estas são as mais relevantes
- Exiba os resultados da previsão do modelo, mesmo para partidas passadas

5 Desenvolvimento do Trabalho

5.1 Tecnologias Utilizadas

5.1.1 Supabase

Supabase é uma plataforma de gerenciamento de banco de dados, que ao mesmo tempo oferece uma série de recursos *"out-of-the-box"* que encurtam o processo de desenvolvimento. O principal desses recursos, que é também a razão pela qual optamos por essa plataforma, é o oferecimento direto de um API a partir do banco de dados. Dessa forma, a nossa aplicação móvel pode consumir diretamente os dados da API do Supabase, sem a necessidade de criar um *backend* customizado.

Essa API, segundo a própria *documentação do Supabase* (SUPABASE, 2022), é criada diretamente a partir das tabelas PostgreSQL contidas na plataforma, usando a tecnologia PostgREST. Está é uma ferramente open-source já bem conceituada no mercado, que conecta diretamente ao banco de dados, fornecendo as operações básicas de um banco de dados, chamadas de *CRUD (Create, Read, Update, Delete.*

A grande vantagem dessa ferramenta, já integrada ao Supabase, é, segundo sua própria *documentação* (POSTGREST, 2023), é ter o "banco de dados como única fonte de verdade", eliminando a necessidade de uma API customizada, construída do 0. Mais que isso, afirma: "A filosofia do PostgREST estabelece uma única fonte declarativa da verdade: ocs próprios dados". Como a nossa aplicação e projeto são inteiramente focados na análise, manipulação e apresentação dos dados, optamos por seguir esta filosofia.

Junto com a API, o Supabase oferece um cliente dedicado a integração com o Flutter, que transforma o acesso ao API e por conseguinte ao banco de dados, em algo muito parecido com uma *query* em SQL. Para realizar esta conexão, precisamos apenas da *url* do banco de dados e de uma chave privada. Essa conexão é feita logo na inicialização da aplicação, segundo representado na Figura 7

Também é importante notar que, como se trata de um projeto *open-source*, a variável *supabaseKey*, que é a chave privada, não pode ficar visível no código, do contrário qualquer poderia acessar nosso banco de dados. Para tal, usamos as chamadas variáveis de ambiente, que permitem manter a chave sob sigilo.

Além disso, o Supabase oferece armazenamento de arquivos. Isso também é importante para que o aplicativo móvel exiba as imagens referentes as ligas e aos clubes, sem a necessidade de salvar do usuário fazer o *download* de todo o acervo de imagens da aplicação em seu dispositivo móvel, que costuma conter capacidade de armazenamento

```dart
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Supabase.initialize(
    url: const String.fromEnvironment('supabaseUrl'),
    anonKey: const String.fromEnvironment('supabaseKey'),
  );
  runApp(MaterialApp(
    title: 'Flutter Demo',
    theme: ThemeData(
      primarySwatch: Colors.blue,
    ), // ThemeData
    home: const HomePage(),
    routes: {
      matchesRoute: (context) => const MatchesView(),
      seasonsRoute: (context) => const SeasonView(),
    },
  )); // MaterialApp
}
```

Figura 7 – Conexão com Supabase no Flutter

reduzida.

A *documentação do Supabase* (STORAGE, 2023) nos mostra a seguinte arquitetura para o armazenamento de arquivos:

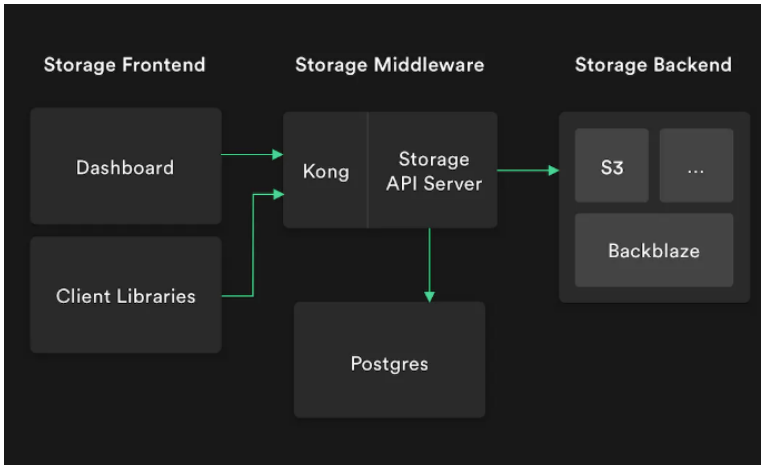

Figura 8 – Conexão com Supabase no Flutter

Na figura, o nosso aplicativo é representado pelo *Client Libraries*, que se comunica com a *middleware*. Também podemos ver que o Supabase delega a terceiros o armazenamento a outros provedores, como o S3, da Amazon.

Depois de adicionar todas as imagens referentes a cada liga e clube incluídos em nossos dados, também precisamos adicionar as *url's* referentes a cada uma das imagens as tabelas de ligas e clubes, respectivamente. Desta forma, a API gerada pelo Supabase, já detalhada nesta seção, retorna também estas imagens, permitindo que o aplicativo as exiba, sem a necessidade de mantê-las em armazenamento local.

5.1.2 Flutter

O Flutter é uma plataforma de desenvolvimento de aplicativos móveis open source criada pelo Google. É amplamente utilizado para a criação de aplicativos móveis multi-plataforma, o que significa que você pode desenvolver um único aplicativo Flutter que funcione em múltiplas plataformas, incluindo Android e iOS. O Flutter se destaca por diversas características e benefícios:

- Linguagem de Programação Dart: O Flutter utiliza a linguagem de programação Dart, que é conhecida por sua eficiência e desempenho. Ela oferece suporte a recursos modernos de programação, como programação assíncrona, que é essencial para aplicativos móveis.
- Interface do Usuário Rica: O Flutter oferece uma ampla variedade de widgets personalizáveis para criar interfaces de usuário ricas e atraentes. A estrutura de interface do Flutter é altamente flexível, permitindo a criação de designs criativos.
- Hot Reload: O recurso Hot Reload do Flutter é especialmente útil para desenvolvedores, pois permite fazer alterações no código e ver imediatamente o impacto nas interfaces de usuário, tornando o processo de desenvolvimento mais ágil e eficiente.
- Desenvolvimento Multiplataforma: Com o Flutter, um único código-fonte pode ser usado para criar aplicativos que funcionam tanto em dispositivos Android quanto iOS, reduzindo significativamente o esforço de desenvolvimento e manutenção.
- Comunidade Ativa: O Flutter possui uma comunidade de desenvolvedores ativa e crescente, o que significa que você pode encontrar recursos, bibliotecas e soluções para uma ampla variedade de desafios de desenvolvimento.

Outra grande vantagem é que podemos utilizar o mesmo código-fonte para a aplicação móvel, voltada ao usuário final, e para o site do projeto, destinado as informações referente a pesquisa e desenvolvimento do projeto.

No geral, o Flutter é uma tecnologia robusta e moderna para o desenvolvimento de aplicativos móveis e se encaixa bem em projetos que exigem uma experiência do usuário rica e a entrega de aplicativos para dispositivos Android e iOS, além de ter uma curva de aprendizado mais leve do que outras tecnologias de desenvolvimento de interfaces de usuário.

Além disso, segundo o artigo *A Comparison of Performance and Looks Between Flutter and Native Applications: When to prefer Flutter over native in mobile application development* (OLSSON, 2020), o Flutter não tem perdas de desempenho significativas em relação as linguagens nativas (*Swift e iOS*), e portanto é adequado para aplicações multi-plataformas.

5.1.3 Google Colab

Google Colab é um serviço que hosteia e permite o compartilhamento de Jupyter Notebooks, permitindo a possibilidade de rodar códigos de python remotamente e de forma de fácil compartilhamento. Essa ferramenta é ótima para fases iniciais de como o desenvolvimento do modelo e a exploratória de dados devido ao seu carácter de notebook, possibilitando visualizações de resultados e gráficos de forma iterativa ao longo do desenvolvimento.

5.2 Projeto e Implementação

Descrever os resultados do projeto e da implementação do trabalho, com justificativas das decisões tomadas.

5.2.1 Frontend

Dado o possível interesse público nos resultados deste trabalho, pela importância do futebol no país e pelo crescente número de apostas esportivas, pensamos que seria mais vantajoso apresentar nossos resultados na forma de uma aplicação móvel. Como mencionado na sessão sobre Flutter, esta é uma tecnologia versátil e feita para com aplicações multi-plataformas em mente. O aplicativo que desenvolvemos pode ser utilizado em dispositivos iOS, Android e na Web, versão que corrsponde ao site do projeto, mas também é possível gerar *builds* binários para Linux, MacOS e Windows. A aplicação compila tanto os resultados do modelo, com detalhes da implementação e dos resultados obtidos, dados de partidas passadas, que compõe a base de dados do algoritmo, as previsões geradas para as partidas futuras e uma série de dados sobre apostas, funcionando tanto como um agregador - para que o usuário possa escolher a as apostas melhores *odds*, como sugerindo as apostas que, segundo nosso modelo, tem maior potencial lucrativo ou maior chance de sucesso.

Um dos grandes princípios do funcionamento do Flutter são os chamados *Widgets*. Cada Widget representa uma classe, segundo o conceito da Orientação a Objetos, mas também representa um componente visual. Segundo a própria documentação do flutter, "Os Widgets do Flutter são construídos usando um framework moderno que recebeu inspiração do React. A ideia central é que você constrói sua UI com Widgets."(FLUTTER, 2023b)

Os Widgets se dividem em duas grandes famílias, os *Stateful Widgets* e os *Statless Widgets*. A diferença entre eles fica clara a partir da nomenclatura: Os Stateless Widgets não tem estado, são estáticos; já o Stateful Widgets tem estado. Quando o estado muda, todo os *widgets* filhos do alterado são reconstruídos e redesenhados na tela.

Aproveitaremos agora esta seção para detalhar implementação do aplicativo, passando por cada uma das telas.

5.2.1.1 Sobre nós

Figura 9 – Captura de tela Sobre nós

Esta tela contém informações sobre o projeto, como as tecnologias utilizadas, links para este documento e para o código-fonte da própria aplicação, mantido no *GitHub*.

A ideia é que ela funcione como uma página institucional do projeto, apresentado tanto a aplicação como seus autores ao público geral.

A Figura 10 representa a chamada *Widget Tree*, ou árvore de Widgets, que é uma representação da hierarquia dos Widgets como components da tela.

Podemos observar que o Widget "pai", ou seja, o Widget de maior hierarquia na tela é o Scaffold. Seu papel é dividir a tela e preparar 3 components imporatntes para a

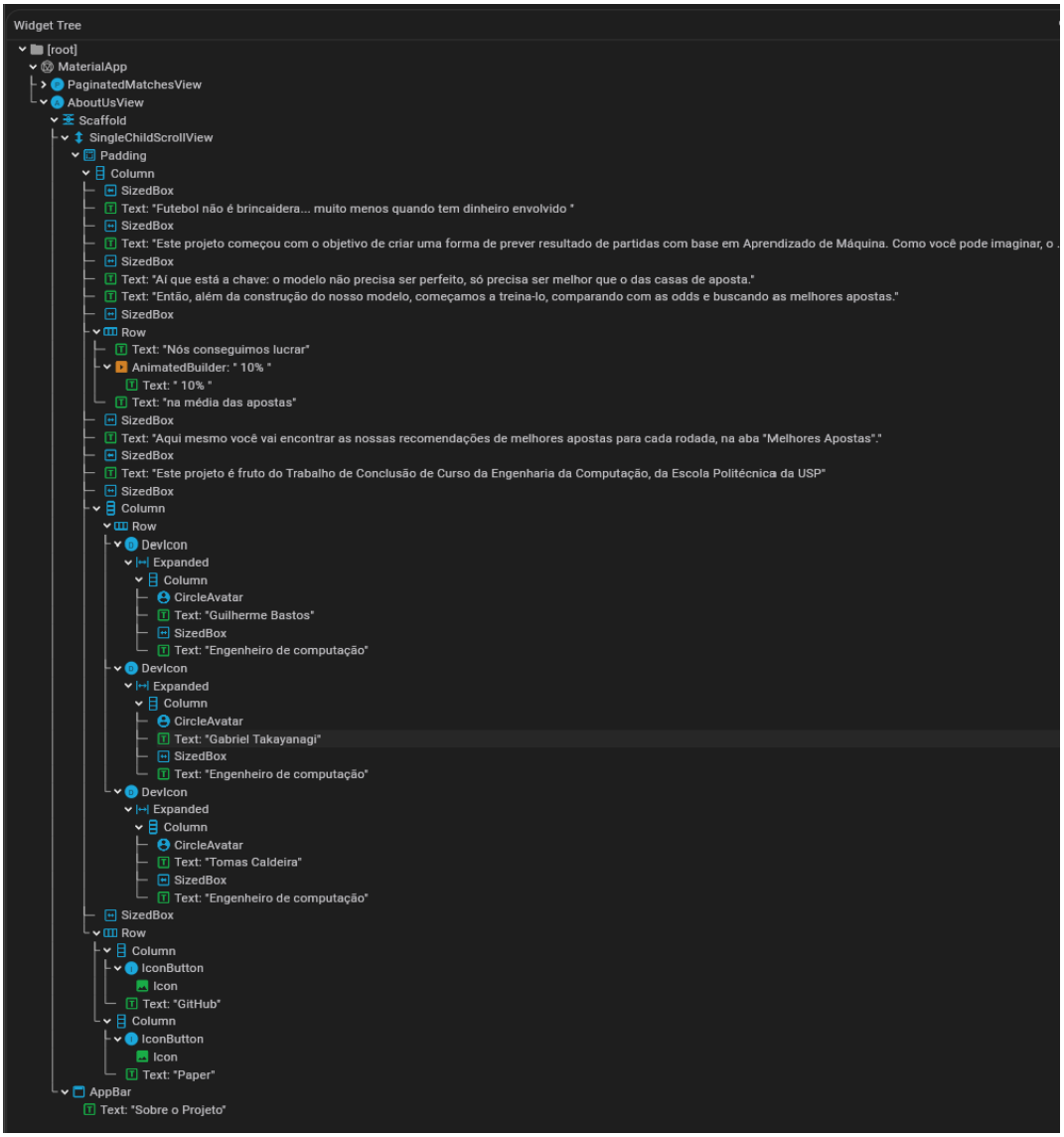

Figura 10 – Widget Tree da sessão sobre-nós

arquitetura da aplicação:

- A AppBar, barra no topo da tela, aonde encontramos o link da página e o botão que abre o menu lateral
- O Drawer, ou gaveta, que a barra lateral, que permite navegar com facilidade para qualquer outra página da aplicação
- O Body, que representa a página em si.

Logo abaixo do Scaffold, temos o SingleChildScrollView, que é apenas um Widget que controla a rolagem da tela, permitindo que, no caso do conteúdo exceder o espaço físico da tela, que o usuário possa deslizar o dedo sobre a tela para "descer"o conteúdo.

Como tudo em Flutter são Widgets, o "padding", ou espaçamento entre a borda e o início do conteúdo, também vem na forma de um Widget. Semelhante ao SingleChildScrollView, ele tem um filho único, mas permite controlar esse espaçamento. No caso, estamos usando o espaçamento horizontal, dando uma margem de 10 pixels a esquerda e a direita da tela.

Depois, temos o Widget *Column*, que faz com que os Widgets que ele contém sejam ordenados de maneira vertical, como uma coluna. Esta primeira coluna não é mais que um conjunto de fragmentos de texto, com as informações da página.

Também há na página um detalhe sobre a animação que há na porcentagem de lucro obtida pelo modelo com as apostas esportivas. Está é uma técnica comum da interface de usuário que convida o usuário a prestar maior atenção ao número.

```
15  class _AboutUsViewState extends State<AboutUsView>
16      with TickerProviderStateMixin {
17    late AnimationController _accuracyController;
18    late Animation<double> _accuracyAnimation;    The value of the field '_acc
19
20    late AnimationController _profitController;
21    late Animation<double> _profitAnimation;
22
23    @override
24    void initState() {
25      super.initState();
26
27      // Set up the animation controller
28      _accuracyController = AnimationController(
29        vsync: this,
30        duration: const Duration(seconds: 4),
31      ); // AnimationController
32
33      _profitController = AnimationController(
34        vsync: this,
35        duration: const Duration(seconds: 2),
36      ); // AnimationController
37
38      _accuracyAnimation =
39          Tween<double>(begin: 0, end: 85).animate(_accuracyController);
40      _profitAnimation =
41          Tween<double>(begin: 0, end: 10).animate(_profitController);
42
43      // Start the animation
44      _accuracyController.forward();
45      _profitController.forward();
46    }
```

Figura 11 – Definição do estado inicial da sessão Sobre nós

Para conseguir esse efeito, usamos um *Controller* e uma *Animation*. O primeiro passo, é, declarar essas variáveis. Depois, elas são inicializadas no método *InitState*, que atribui os valores do estado inicial do Widget. Para o *Controller*, podemos definir a duração da animação. Depois, ao inicializar o valor da Animação, definimos o número em que a animação começa e o número em que ela termina, como podemos observar na Figura 11.

Com tudo definido, basta fazer uso das variáveis usando o Widget *AnimatedBuilder*, conforme observamos em Figura 12.

Neste caso, usamos o Widget *Row* que tem um efeito inverso ao da *Column*, fazendo

```
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    const Text(
      'Nós conseguimos lucrar',
      style: TextStyle(fontSize: 20),
    ), // Text
    AnimatedBuilder(
      animation: _profitAnimation,
      builder: (context, child) {
        // Display the animated value
        return Text(
          ' ${_profitAnimation.value.toInt()}% ',
          style: const TextStyle(fontSize: 25),
        ); // Text
      },
    ), // AnimatedBuilder
    const Text(
      'na média das apostas',
      style: TextStyle(fontSize: 20),
    ), // Text
  ],
), // Row
```

Figura 12 – Uso do Animated Builder

que seus Widgets filhos sejam exibidos em uma sequência horizontal.

Criamos também um Widget chamado *DevIcon*, feito para representar os alunos autores deste projeto e da aplicação. Como podemos observar na da Figura 10, este Widget usa o *Expanded*, que é um Widget chave para aplicações multi-plataformas. Segundo a documentação da Classe, "é um Widget que expande um filho de Row, Column ou Flex, para que o filho ocupe todo o espaço disponível"(FLUTTER, 2023a). Como o espaço horizontal da tela varia muito entre um dispositivo móvel e um computador, esse útil ajuda a tornar o espaço mais bem adaptado ao tamanho da tela. Neste caso, os ícones com as fotos dos autores do projeto mudam de tamanho e distância dinamicamente, conforme a largura da tela.

5.2.1.2 Tela de partidas

A tela de partidas já abriga muito mais lógica do que as anteriores e ele pode ser usada para exibir dados de ligas ou rodadas diferentes, dependendo dos parâmetros passados a ela. Outra caractérisitca importante, é o que chamamos de *scroll infinito*, isso é, conforme se desliza a página para baixo, uma nova chamada a API é feita e carregam-se mais partidas.

Por conta dessa lógica mais complexa, ela é organizada de uma maneira diferente das demais, com maior encapsulação das responsabilidades e conceitos mais próximas da orientação objeto. Podemos começar a análise mais profunda dos componentes da tela a

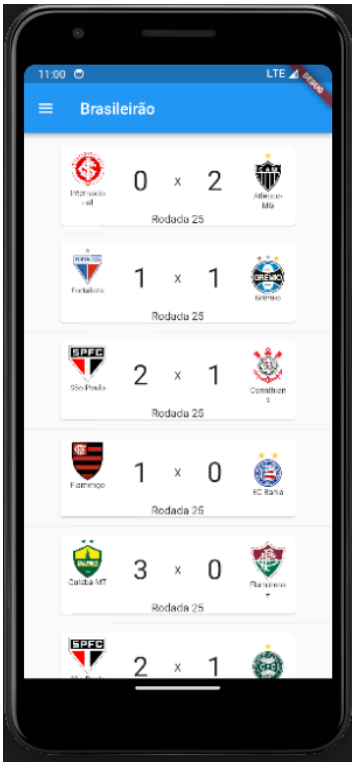

Figura 13 – Captura de tela de partidas

partir da Figura 14

A tela contém os mesmos elementos básicos mencionados na subseção 5.2.1.1, isto é, o *Scaffold*, composto por um *AppBar* com o título da página, um *Drawer* com o menu lateral e o *Body*, que é o conteúdo da tela. Pode-se observar que nesse caso, todo o conteúdo do *Body* foi extraído para um *Widget* personalizado de autoria nossa, chamado MatchesList.

Como sempre, os *Stateless Widgets* são separados na definição do Widget em si e de seu estado. Podemos ver que em sua definição temos um único atributo, um inteiro opcional denominado *seasonId*, que se refere ao Id da temporada para qual se buscará as partidas.

Na definição do estado, observadas na Figura 15 também definimos alguns atributos importantes:

- *currentSeasonId*, que é inicializada com o valor definido no *Widget*.
- *matchRepo*, que é do tipo *MatchRepository*, que é a classe que usamos para abstrair o retorno das partidas
- Aproveitamento de pontos na competição do time, ou seja, quantidade de pontos obtidos dividido por número de jogos. O late indica que o valor da variável não está disponível na inicialização do Widget.

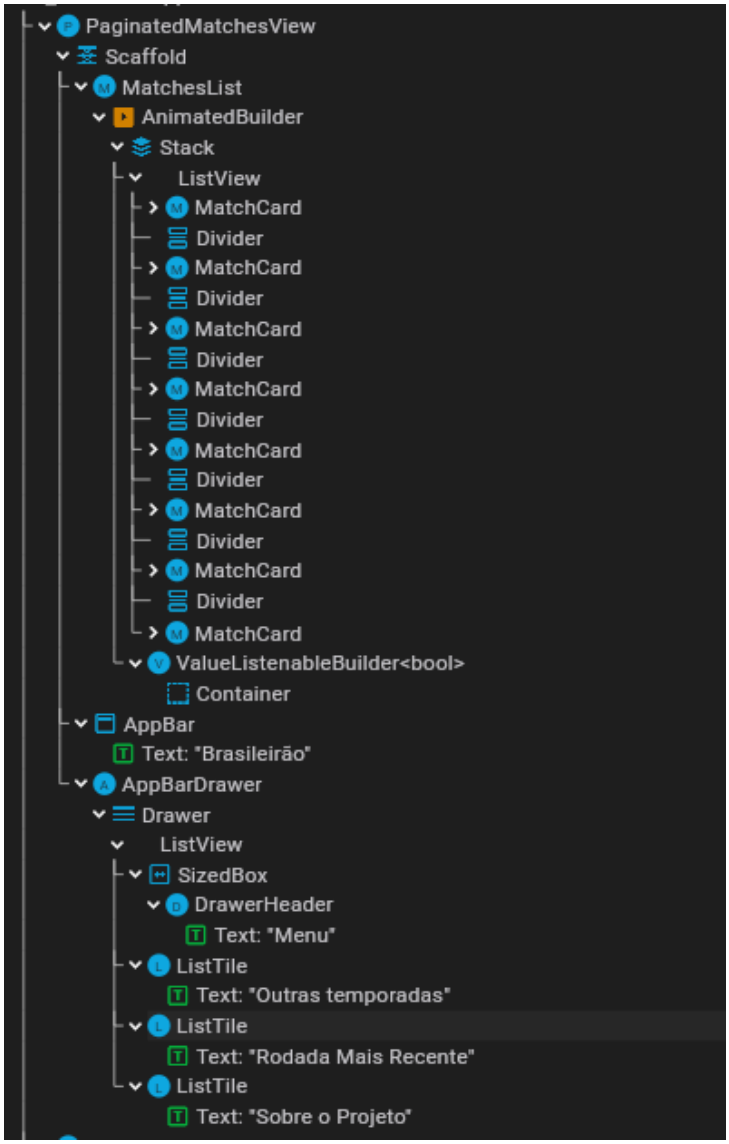

Figura 14 – Widget Tree da tela de partidas

- *loading*, que recebe um *ValueNotifier*, que será usado para exibir ou não um indicador de carregamento, qunado a página estiver carregando.

- *_scrollController*, do tipo *ScrollController*, que usamos para monitorar a posição relativa da tela. Será uma medição de quanto o usuário deslizou sobre a área disponível na tela.

O método *initState* é chamado assim que o *Widget* é criado, e serve para configurar o seu estado inicial. O ponto que mais merece nossa atenção em sua implementação é a chamada do *addListener* ao *_scrollControler*, que faz com que passamos a escutar o *infiniteScrolling*, situação em que a posição do *scroll* é igual ao tamanho da tela e o valor do *loading* é falso. É fácil notar que essa será a condição que usamos para fazer uma nova chamada a API e exibir mais partidas.

```dart
class MatchesList extends StatefulWidget {
  const MatchesList({super.key, this.seasonId});
  final int? seasonId;

  @override
  State<MatchesList> createState() => _MatchesListState();
}

class _MatchesListState extends State<MatchesList> {
  int? _currentSeasonId;
  late MatchRepository matchRepo;
  final loading = ValueNotifier(true);
  late final ScrollController _scrollController;

  @override
  void initState() {
    super.initState();
    _scrollController = ScrollController();
    _scrollController.addListener(infiniteScrolling);
    _currentSeasonId = widget.seasonId;
    matchRepo = MatchRepository();
    loadMatches();
  }

  infiniteScrolling() {
    if (_scrollController.position.pixels ==
            _scrollController.position.maxScrollExtent &&
        !loading.value) {
      loadMatches();
    }
  }

  @override
  void dispose() {
    super.dispose();
    _scrollController.dispose();
  }

  loadMatches() async {
    loading.value = true;
    await matchRepo.getMatches(_currentSeasonId ?? 2023);
    loading.value = false;
  }
```

Figura 15 – Definições do MatchesList

O método *loadMatches* é definido de maneira assíncrona, com a palavra-chave *await*, na busca por mais partidas, que indica que deve-se esperar um evento para dar continuidade a execução.

Com a lógica dos estados definida, podemos partir agora para a implementação do *layout*, definida na Figura 16 ,que também está bem encapsulada em elementos de responsabilidade única.

Nesta tela, o *AnimatedBuilder* cumpre um papel diferente do que na Figura 12. Aqui, seu papel é de reconstruir a tela com as mudanças no *matchRepo*, que acontecem a cada chamada da API.

Logo abaixo dele na *WidgetTree*, temos uma *Stack*, que é a sobreposição de dois elementos. Neste caso, a utilização é uma maneira criativa de exibir ou não o indicador de carregamento, sem atrapalhar a visibilidade. O *Widget loadingIndicatorWidget* apenas retorna o indicador de carregamento quando a variável definida *loading* é verdadeira, e vazio quando é falsa.

Já o *ListView* é utilizado para construir elementos a partir de uma lista. Nesse caso, a lista é a lista de partidas retornas pelo *matchRepo*. Para cada partida é criado um *MatchCard*, que exibe os dados da partida da maneira observada na Figura 13, com a imagem, placar e nome dos dois times da partida, bem como o número da rodada.

Como podemos ver na Figura 17, o MatchCard tem um *Gesture Detector*, que é

```dart
@override
Widget build(BuildContext context) {
  return AnimatedBuilder(
      animation: matchRepo,
      builder: (context, snapshot) {
        return Stack(
          children: [
            ListView.separated(
                controller: _scrollController,
                itemBuilder: ((context, index) {
                  final match = matchRepo.matches[index];
                  return MatchCard(
                    matchId: match.matchId,
                    awayTeamImage: Image.asset(
                      match.awayTeamImage,
                      errorBuilder: (context, error, stackTrace) {
                        return Image.asset(
                          "assets/images/clubs/placeholder.jpeg",
                        ); // Image.asset
                      },
                    ), // Image.asset
                    awayTeamScore: match.awayTeamScore,
                    awayTeamName: match.awayTeamName,
                    homeTeamImage: Image.asset(
                      match.homeTeamImage,
                      errorBuilder: (context, error, stackTrace) {
                        return Image.asset(
                          "assets/images/clubs/placeholder.jpeg",
                        ); // Image.asset
                      },
                    ), // Image.asset
                    homeTeamScore: match.homeTeamScore,
                    homeTeamName: match.homeTeamName,
                    round: match.round,
                    stadium: match.stadium,
                    referee: match.referee,
                    audience: match.audience,
                    date: match.date,
                  ); // MatchCard
                }),
                separatorBuilder: (_, __) => const Divider(),
                itemCount: matchRepo.matches.length), // ListView.separated
            loadingIndicatorWidget(),
          ],
        ); // Stack
      }); // AnimatedBuilder
}
```

Figura 16 – Definições do MatchesList

uma maneira de gerar uma interação a partir de um componente visual. Nesse caso, usamos uma interação de clique, para que, ao clicar em um *card*, o usuário seja redirecionado para a visão detalhada da partida.

Para a exibição das informações, temos uma coluna de informações para cada time, contendo o logo e nome do time, inseridos dentro de uma linha, que contém as colunas e o resultado do jogo.

Voltando a tratar da lógica da paginação, vamos agora descrever o MatchRepo, isso é, o repositório de partidas. Como mencionado, buscamos fazer componentes de responsabilidade única e, portanto, essa é uma classe que não tem componente visual, apenas lidando com a busca das partidas.

A classe possui alguns atributos importantes:

- resultsFetched, que é usado para fazer a lógica da paginação

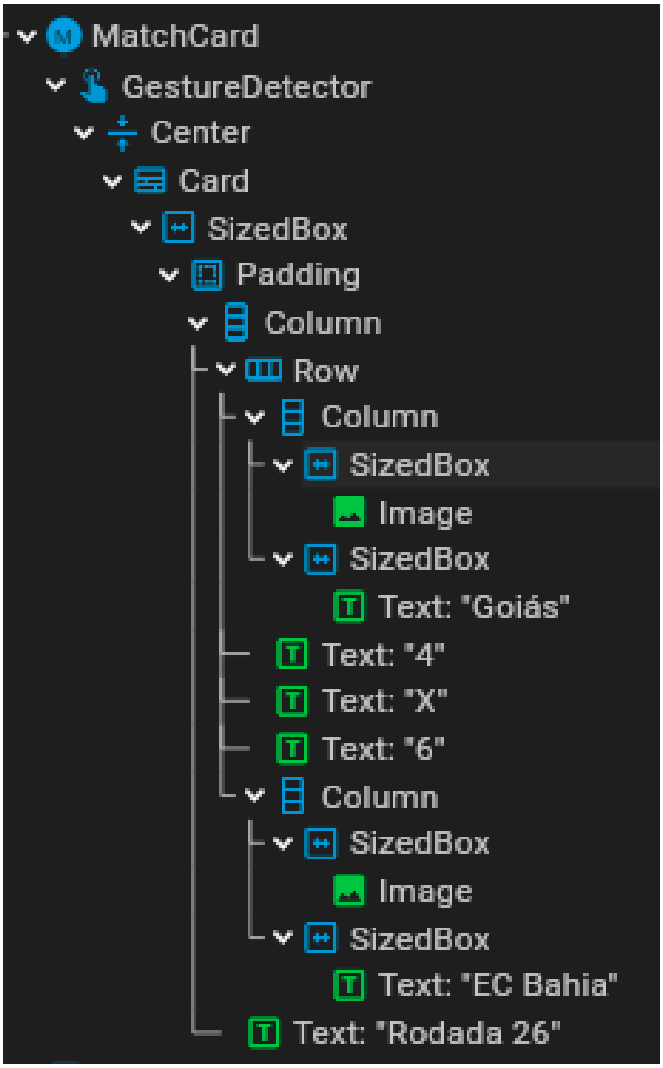

Figura 17 – Widget Tree do MatchCard

- perPage, que é definido como uma constante estática e dita quantas resultados são carregados por vez

- matches, que contém a lista de partidas.

Além dos atributos, definimos o *getter* matches, para expor essa lista para fora da classe.

Também é importante notar que, ao definir uma Lista em *Dart*, podemos definir que tipo de objetos essa lista contém. Nesse caso, temos uma lista de *Matches*, que também é uma classe que criamos.

O método *getMatches* é o que de fato vai em busca dos resultados, usando a API do *Supabase*, que se parece muito com uma *query* de SQL. Instanciamos o cliente do *Supabase*, selecionando todas as colunas da tabela *br_fat_matches* que tenham o anoCampeonato que estamos buscando, aplicando a paginação através do range e ordenando o resultados

```dart
class MatchRepository with ChangeNotifier {
  int resultsFetched = 0;
  static const perPage = 30;
  final List<Match> _matches = [];

  List<Match> get matches => _matches;

  getMatches(int? seasonId) async {
    final anoCampeonato = seasonId ?? 2023;
    final result = await Supabase.instance.client
        .from('br_fat_matches')
        .select<List<Map<String, dynamic>>>('*')
        .eq('ano_campeonato', anoCampeonato)
        .range(resultsFetched, resultsFetched + perPage)
        .order('data', ascending: false);
    for (var i = 0; i < result.length; i++) {
      _matches.add(Match.fromMap(result[i]));
    }
    resultsFetched += result.length;
    notifyListeners();
  }
}
```

Figura 18 – Implementação do repositório de partidas

pela data da partida.

Para cada linha retornada da tabela, criamos uma nova instância da classe partida e a adicionamos a lista matches. Também soma-se o número de resultados obtidos a variável *resultsFetched* como preparação para a nova busca.

5.2.1.3 Tela da partida detalhada

A tela da partida detalhada é acessada a partir do *Gesture Detector* contido em cada *MatchCard*, conforme exibido na Figura 20. Como pode-se ver, os elementos passados para a exibição da partida na listagem são transmitidos no momento de clique e usadas para a exibição nesta tela mais detalhada.

Essas informações são exibidas de maneira muito similar a listagem de partidas, mas sem a margem ao redor. Isso pode ser observado pela *Widget Tree* da tela representada na imagem.

A exibição é composta de uma coluna que contém uma linha com a exibição dos times e resultado e depois uma série de textos representando dados a mais da partida, como data, estádio, árbitro e público.

A linha que exibe os dados dos times é composta de 2 colunas, cada uma contendo o nome e logo do time, depois o placar e o "x".

5.2.1.4 Tela das temporadas

A tela das temporadas é o caminho para o usuário ver os dados de outros anos do campeonato brasileiro. Ela funcionava de maneira quase que idêntica a tela de partidas,

Figura 19 – Implementação do repositório de partidas

```dart
@override
Widget build(BuildContext context) {
  return GestureDetector(
    onTap: () async {
      Navigator.of(context).pushNamed(
        matchRoute,
        arguments: {
          'matchId': matchId,
          'homeName': homeTeamName,
          'awayName': awayTeamName,
          'awayTeamImage': awayTeamImage,
          'homeTeamImage': homeTeamImage,
          'homeTeamScore': homeTeamScore,
          'awayTeamScore': awayTeamScore,
          'round': round,
          'date': date,
          'stadium': stadium,
          'referee': referee,
          'audience': audience,
        },
      );
    },
    child: Center( // Center ···
  ); // GestureDetector
}
}
```

Figura 20 – Método build do MatchCard

mas apenas exibe o *nome* da temporada. Originalmente a tela foi projetada para receber um identificador de uma liga, mas como alteramos o projeto para exibir apenas dados da primeira divisão do futebol brasileiro, não precisamos mais disso.

A requisição para buscar as opções de ligas disponíveis é feita através do *Supabase*, de uma maneira muito parecida com SQL. Estamos filtrando os dados da tabela *seasons*

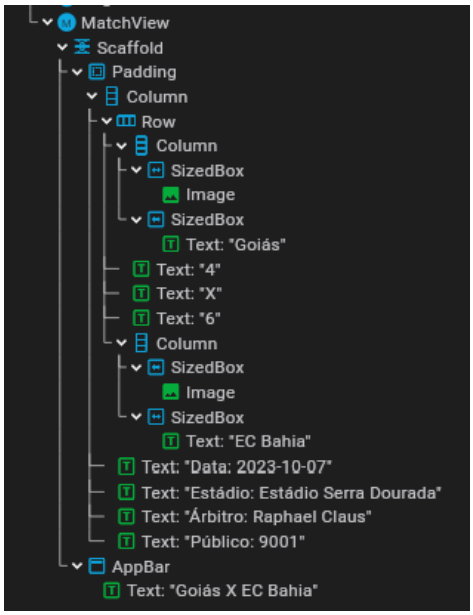

Figura 21 – Widget tree da tela de partidas

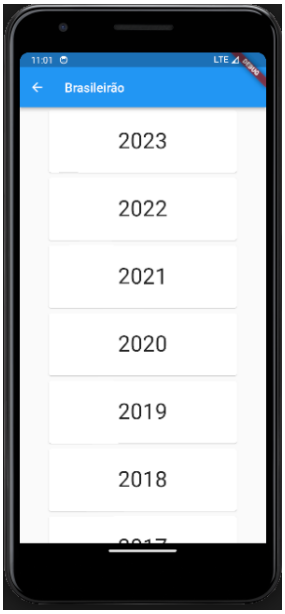

Figura 22 – Captura da tela de temporadas

```dart
class _SeasonViewState extends State<SeasonView> {
  @override
  Widget build(BuildContext context) {
    //final arguments = (ModalRoute.of(context)?.settings.arguments ??
    const int leagueId = 999;
    const String leagueName = 'Brasileirão';
    final _future = Supabase.instance.client    The local variable '_fu
        .from('seasons')
        .select<List<Map<String, dynamic>>>()
        .eq('competition_id', leagueId)
        .order('season_name');

    return Scaffold(
      appBar: AppBar(
        title: Text(leagueName),    Use 'const' with the constructor to
        toolbarHeight: height,
      ), // AppBar
      body: FutureBuilder<List<Map<String, dynamic>>>(
          future: _future,
          builder: (context, snapshot) {
            if (!snapshot.hasData) {
              return const Center(child: CircularProgressIndicator());
            }
            final seasons = snapshot.data!;
            return ListView.builder(
              itemCount: seasons.length,
              itemBuilder: ((context, index) {
                final season = seasons[index];
                return SeasonCard(
                  seasonId: season['season_id'],
                  seasonName: season['season_name'],
                  competitionId: season['competition_id'].toString(),
                  leagueName: leagueName,
                ); // SeasonCard
              }),
            ); // ListView.builder
          }), // FutureBuilder
    ); // Scaffold
  }
}
```

Figura 23 – Implementação do build da tela de temporadas

em que o *competion id* seja igual ao *id* do "Brasileirão".

Também é importante notar como na implementação do layout existe uma checagem para ver se a requisição já foi completa antes de exibir os dados. Quando eles ainda estão sendo carregados, apenas exibe-se um indicador de carregamento.

5.2.2 Criação do modelo

Para a criação do modelo de predição do resultado das partidas, adotando a metodologia do CRISP-DM(SCHRöER, 2021), foi decidido a criação de duas iterações de modelo.

A primeira iteração teria como o objetivo principal apenas o desenvolvimento do grupo, para o entendimento das nuancias da criação de um modelo de predição para futebol, além disso, os resultados dessa iteração inicial poderiam servir como comparação para entender melhor o desempenho da segunda iteração.

A segunda e última iteração do modelo teria um foco maior na aplicabilidade no contexto de apostar de jogos do brasileirão. Dessa forma, acredita-se que é possível avaliar, além da qualidade do modelo sua utilidade em um contexto mais prático.

5.2.2.1 Primeira iteração de desenvolvimento do modelo

5.2.2.1.1 Coleta e entendimento dos dados

Como o intuito da primeira iteração seria apenas para ganhar experiência e entender melhor o problema, não se foi colocado grande ênfase na coleta de dados. Nessa iteração é

foi definido como preferível encontrar uma base de dados pronta ou quase pronta do que passar tempo coletando dados que podem ou não ser descartados no final da iteração.

O dataset escolhido foi uma base de dados não tratados de competições de futebol européias diponibilizados pela Stats Bomb Open Data que foca em disponibilizar dados de eventos esportivos. Essa base de dados possui diversos dados de partidas de futebol, jogadores e técnicos de diversas competições, dessa forma, apesar de ainda não podermos aplicar o modelo para o contexto brasileiro podemos entender os básicos com esses dados mais facilmente disponíveis e aplicar o aprendizado em uma nova base de dados.

Avaliando a base de dados, apesar de ela disponibilizar uma gama muito variada de informações incluindo informações detalhadas de todos os jogadores, técnicos e posições de chutes da partida, foi decidido operar apenas com o volume de gols pois não há um volume suficiente de dados dessas informações para utilizar um modelo complexo o suficiente as quais elas seriam úteis.

Como os dados se encontravam em formato json foi então limpado e separado apenas as informações relevantes.

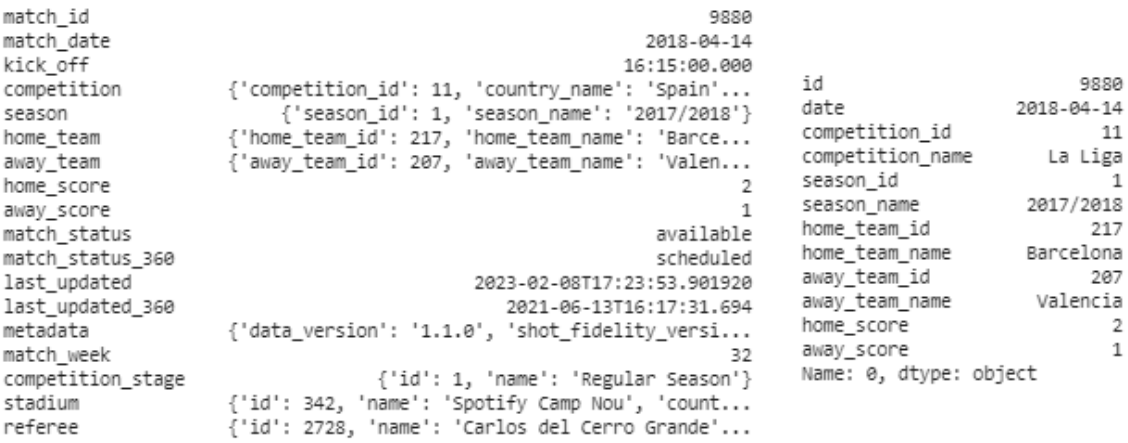

Figura 24 – Um exemplo da mesma linha pré tratamento a esquerda e pós tratamento a direita.

Além disso, um fator muito importante para os parâmetros utilizados no modelo é a completude de uma temporada. Para a parte de tratamento de dados o desempenho recente de um time é muito importante para prevermos os resultados de um jogo. Isso significa que ter uma temporada completa ou quase completa se torna essencial para os dados coletados, uma vez que com quanto maior a quantidade de jogos faltantes, menor será a qualidade dos parâmetros de treinamento que teremos. Uma temporada com muitos dados faltantes no dataset muitas vezes pode ser prejudicial para o treinamento do modelo. Dessa forma, apesar do dataset disponibilizar 66 temporadas foram apenas escolhidas quatro temporadas cujos os dados estavam completos ou quase completos.

5.2.2.1.2 Preparação dos dados

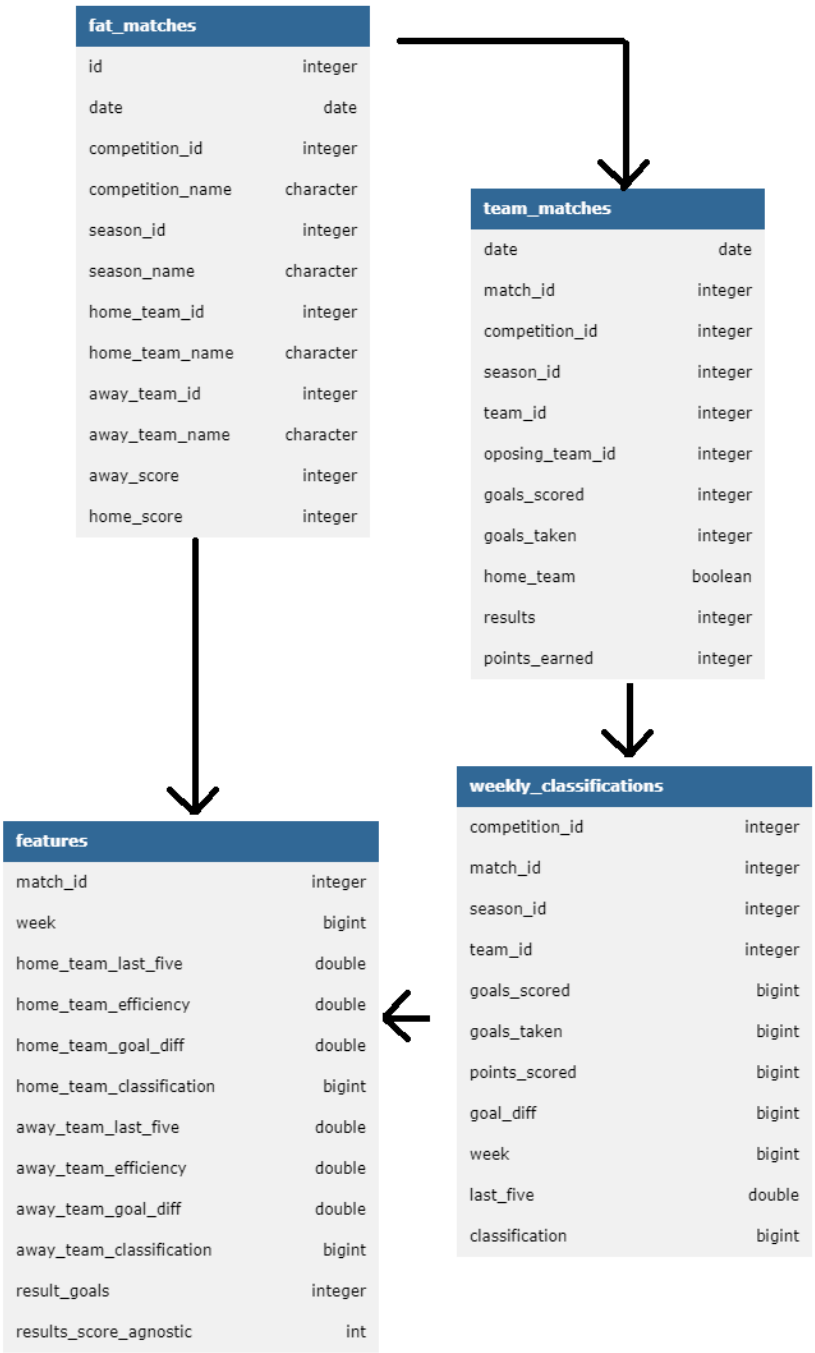

Figura 25 – Tabelas criadas para primeira iteração e sua dependências.

Uma vez com os dados limpos e prontos para serem tratados começou-se a preparação dos dados e a construção de parâmetros para treinamento do modelo.

O principal objetivo nessa extração de parâmetros seria a adição do contexto dos dados de um time em um campeonato antes da partida que seria predita. Apesar do tratamento dos dados ser linear, foi decidido a criação de duas tabelas intermediárias de forma que, caso necessário, seria possível ajustar os parâmetros com maior facilidade.

A primeira tabela criada foi a *teamMatches*, essa tabela tem como objetivo separar os times visitantes e mandantes, sendo única no par de chaves teamId e matchId além disso ela já faz alguns pequenos tratamentos como a quantidade de pontos obtidos e o resultado do jogo para o time. A segunda tabela intermediária foi a *weeklyClassifications*, onde é separado, por rodada, ou *week*, o desempenho do time ao longo do campeonato, incluindo a média de gols realizados por jogos, gols sofridos, saldo de gols, classificação e desempenho recente.

Finalmente a tabela *features* separa as informações imediatamente anteriores ao jogo dos times mandantes e visitantes e junta com as informações identificadoras da partida para o treinamento. Sumarizando os parâmetros separados para o treinamento do modelo inicial foram:

- *week* - Rodada da competição.
- *lastFive* - Média dos resultados dos últimos cinco jogos, sendo uma vitória considerada como 2, empate como 1 e derrota como 0.
- *efficiency* - Aproveitamento de pontos na competição do time, ou seja, quantidade de pontos obtidos dividido por número de jogos
- *goalDiff* - Média da diferença de gols do time por partida
- *classification* - Classificação do time

5.2.2.1.3 Treinamento do modelo e resultados iniciais

Para o treinamento do modelo foi usado o dataset criado anteriormente, que totalizou 1227 jogos. Esses jogos foram separados em dataset de treinamento e dataset de teste, de forma a se obter os resultados evitando o máximo possível a influencia de overfitting. Inicialmente foram utilizados três modelos diferentes, uma árvore de decisão simples, AdaBoost e Gradient Boosting. Observou-se resultados similares nos três modelos inicialmente, com acurácia de entorno de 0.65 e performando pior em casos de empate.

Observando os resultados pode-se fazer algumas observações de pontos de melhoria do modelo nas novas iterações. Primeiramente observa-se a performance pior do modelo em casos de empate, isso pode ser causado por perda de informação nos parâmetros. Por exemplo, nos resultados recentes, como um empate possui o valor de 0.5 isso significa que

Quadro 1 – Resultados arvores de classificação

Classificação	Precisão	recall	f1-score
derrota	0.70	0.63	0.67
empate	0.51	0.61	0.56
vitória	0.72	0.68	0.70

Fonte: Autor.

Quadro 2 – Resultados AdaBoost

Classificação	Precisão	recall	f1-score
derrota	0.64	0.78	0.70
empate	0.51	0.32	0.39
vitória	0.67	0.74	0.70

Fonte: Autor.

Quadro 3 – Resultados Gradient Boosting

Classificação	Precisão	recall	f1-score
derrota	0.65	0.73	0.69
empate	0.55	0.42	0.48
vitória	0.71	0.75	0.73

Fonte: Autor.

o modelo não tem como distinguir entre dois empates e uma vitória e derrota. Dessa forma uma separação mais direta de valores desse tipo pode auxiliar.

Outro fator de interessante que pode mostrar um ponto de melhora é a questão de modelos mais complexos exibirem resultados muito similares a simples árvore de decisão. Isso pode ter acontecido devido a pouca experiência com esses modelos, podendo ser que com maior proficiência de uso possa se obter resultados melhores. Outra causa pode ser devido a pouca variedade de parâmetros para o modelo, sendo eles derivados essencialmente do desempenho recente. É possível que com uma base de dados com mais variáveis independentes e uma modelagem de parâmetros adequadas aumentem consideravelmente a diferença de acurácia entre os modelos.

5.2.2.2 Segunda iteração de desenvolvimento do modelo

Como iterado acima, os dois pontos focais para o desenvolvimento da segunda iteração de desenvolvimento do modelo foram dois.

- Obter uma base de dados com uma diversidade maior de métricas por partida com o intuito de obter uma utilizar recursos de modelos mais complexos.

- Reformular os parâmetros de forma a perder menos informações na sua sumarização,

principalmente em empates.

Além disso, considerando o objetivo da segunda iteração de pode ser testado em um ambiente mais práticos foi procurado tentar prever os resultados da série A do Brasileirão do ano atual.

5.2.2.3 Coleta e entendimento dos dados

Para a coleta de dados foram considerados quatro principais fatores:

- A diversidade de métricas disponíveis, em que em algumas bases de dados há uma grande variedade de possíveis estatísticas coletadas para cada partida, havendo uma variância muito grande em tipos e quantidade de métricas em diferentes base de dados.
- A completude da base, como diferentemente de muitos outros modelos, para obtenção dos parâmetros necessários para o treinamento na predição de partidas de de futebol são altamente dependentes do tempo e, consequentemente, é necessário ter uma alta consistência de dados dentro de um período. Partidas faltantes, e em menor grau, estatísticas faltantes dentro de partidas presentes podem ter um grande impacto em o que seriam várias colunas de dados dentro de um modelo.
- A região e o período em dos dados coletados por que com a evolução do jogo, as métricas podem influenciar diferentemente o resultado de uma partida.
- Além disso, para a avaliação de desempenho do modelo foi decidido coletar os dados de apostas da temporada atual do brasileirão.

Para atender esses requisitos, identificaram-se sites que buscam fornecer ao apostador diversas estatísticas e detalhes de partidas de futebol, uma vez que, devido ao mercado de apostas em partidas esportivas, há uma grande variedade desses recursos disponíveis. No entanto, esses sites geralmente não disponibilizam suas bases de dados, concentrando-se em um público que prefere analisar os dados manualmente navegando pelo site. Isso, aliado às altas exigências de atualização dos dados e ao fato de que a escolha dos dados do Brasileirão, que possui menos bases de dados do que as ligas da Europa, tornou difícil encontrar uma base de dados de outra fonte. Diante desse cenário, foi decidida a construção de uma base de dados própria, utilizando informações de sites que detalham estatísticas de partidas de futebol.

Dessa forma foram explorados três sites principais para a coleta de dados:

- https://www.transfermarkt.co.uk/ - Transfersmarkt é um site europeu que possuí estatísticas de ligas ao redor do mundo, incluindo para o Brasil. Esse site é famoso

por ser utilizado para a construção de datasets de futebol, possuindo alta resistência a tráfego e é de fácil navegação. Porém ele possui a desvantagem de algumas partidas não terem todas as estatísticas, principalmente as mais antigas.

- https://www.sofascore.com/ - Sofascore é focado em fornecer informação a apostadores, tendo um gama de esportes diferentes. Esse site é um dos maiores sites de estatísticas de futebol para jogos de futebol brasileiro, possuindo a maior diversidade de métricas por partida, além de dados de aposta.

- https://oddspedia.com/ - Similarmente ao Sofascore, oddspedia tem uma variedade de métricas por partida mas aonde ela se destaca é na variedade de informações que o site tem sobre as casas de apostas. Esse site possuí detalhes de vários tipos de aposta para mais de 30 sites diferentes incluindo variantes como número de gols e resultado do primeiro tempo.

O primeiro site o qual foi coletado informações foi o *sofascore*. Esse site utiliza a técnica de *lazy loading*, a qual uma página principal possuí a habilidade de disponibilizar uma grande quantidade de informações contudo elas só são realmente enviadas quando o usuário requisita elas. Neste caso específico, todos os jogos estão disponíveis em uma mesma *url* porém, dados como jogos da rodada e suas métricas e dados de apostas só são carregados quando se clica em alguma rodada ou algum jogo. Para coletar esses dados foi utilizado uma ferramenta de controle de *webdriver* chamado *selenium* que basicamente funciona como um navegador o qual é possível programar que comando ele executará. Dessa forma é possível simular um usuário que está explorando as informações da partida manualmente. Contudo, apesar dessa técnica ser difícil de distinguir de um usuário real, devido aos comandos repetitivos executados durante o entendimento da arquitetura do site, eventualmente a ferramente de coleta de dados teve o IP banido e, apesar de existir formas de se contornar um banimento de IP, foi julgado que o investimento de combater um software de detecção de coleta de dados não valeria a pena, logo foi decidido abrir mão da coleta nesse site.

Dado o banimento no primeiro site, foi decidido coletar dados do site *transfermarkt* que é famoso por ser a fonte de vários datasets de ligas de futebol construídos a partir de *web scraping* assim foi pressuposto que o site possuí maior capacidade de lidar com o tráfego e permite essa coleta de dados, além disso antes de qualquer requisição feita ao site foi colocado um intervalo aleatório com distribuição normal para diminuir o impacto que se tem no site e mascarar a natureza da ferramenta. Os dados desse site são de muito mais simples coleta, possuindo uma pagina individual por jogo e uma página por rodada cada campeonato. Dessa forma, foi possível automatizar a coleta de dado com muito maior facilidade com um *loop* que, com o id de um campeonato, a rodada e o ano, coletava os *ids* das partidas e navegava para suas respectivas páginas de métricas. Além disso,

devido a simplicidade do processo, pode-se utilizar apenas as bibliotecas mais simples em *requests* para coletar o conteúdo de uma página e *Beaultiful Soup* para separar os dados importantes. Dessa forma foi possível coletar as métricas de partidas do Brasleirão dos anos de 2020, 2021, 2022 e 2023 além de dados da Bundesliga que poderiam ser usados caso nescessário.

Finalmente para a coleta de dados referentes a apostas foi coletado os dados do site *Oddspedia*, esse site similarmente ao *sofascore* possuí *lazy loading*, porém dessa vez os dados foram coletados utilizando uma nova técnica. Os sites que utilizam *lazy loading* funcionam de forma que quando o usuário realiza um certo comando, como selecionar um jogo, o site manda uma requisições para o servidor e esse servidor envia as informações para o site carregar na página. Dessa forma, foi mapeado essas requisições para o servidor que esse site realiza e, copiando os cookies e headers que o site envia, foi possível automatizar a coleta de dados alterando os parâmetros enviados das requisições *getMatchList* e *getMatchOdds* que coletam os dados de *ids* de partidas e dados das apostas de um dado id de partida repectivamente. Abaixo se encontra um exemplo dos parâmetros utilizados para se coletar dados de aposta de 2021 em python, notando-se que a varíavel *round* é flexível possibilitando coletar os *ids* de partidas de todos as rodadas.

```
"params ="{
   "excludeSpecialStatus":"0",
   "sortBy":"default",
   "perPageDefault":"100",
   "startDate":"2021-01-01T03:00:00Z",
   "endDate":"2021-12-31T02:59:59Z",
   "geoCode":"BR",
   "status":"all",
   "sport":"football",
   "popularLeaguesOnly":"0",
   "category":"brazil",
   "league":"serie-a",
   "seasonId":"82604",
   "round":"f""Round {round}",
   "page":"1",
   "perPage":"100",
   "language":"en"
}
```

Nota-se também que depois dessa coleta de dados, os mesmos foram limpos e tratados

para uma *dataframe* e posteriormente armazenados no banco de dados. Abaixo se encontra um exemplo de limpeza de dados para os dados de aposta, onde se separa os dados apenas das chances do resultado final da partida e se traduz o de que resultado essas chances são.

```
def true_odd_name(result, oddsnames)->None:
  import re
  if not result:
    return None
  try:
    index_result=int(re.findall('([\d]+)', result )[0])-1
    return oddsnames[index_result]
  except:
    print(f'Unrecognized result as {result}')
    return None
for index, row in bets_data.iterrows():
  #pass match id
  match_id=row['match_id']
  bet_match_info=row['prematch']
  bets_list = eval(bet_match_info)
  #making sure index 0 is the odds of the result of the match
  assert(bets_list[0]['name']=='Full Time Result')
  #adding the results names, in this case would be win, tie or defeat
  odds_names=bets_list[0]['oddsnames']
  #going through array to only collect the results of the full match
  for i in bets_list[0]['periods']:
    if i['name']=='Full Time':
      result = i['winning_odd']
      list_rows=list(i['odds'][0].keys())
      list_rows.append('match_id')
      list_rows.append('result')
      #add odds to a row
      df_row=pd.DataFrame(data=list(i['odds']), columns=list_rows)
  #true_odd_name translates the name to the
  #actual result for example o1 (odd 1) would be translated to draw
  df_row['result']=true_odd_name(result, odds_names)
  df_row['match_id']=match_id
  renamed_columns={}
  #also renaming the columns to the actual result
  for column_name in df_row.columns:
```

```
    column_to_replace=re.findall('(o[\d]+)', column_name )
    if column_to_replace:
      renamed_columns[column_to_replace[0]]=true_odd_name(column_to_replace[0],odds_name
  df_row=df_row.rename(columns=renamed_columns)
  try:
    bet_df=pd.concat([bet_df, df_row])
  except:
    bet_df=df_row
```

5.2.2.3.1 Preparação dos dados

Para a segunda iteração, o tratamento dos dados sofreu algumas alterações, analisando alguns vencedores de competições do *kaggle*, como o código feito por um usuário chamado @radar para a competição de March Madness foi observado algumas táticas que podem ajudar o modelo para melhorar seu desempenho. Sendo a principal influência desse estudo foi considerar as métricas dos oponentes dos dois times participantes da partida. Em conjunto com o que foi observado da primeira iteração as principais mudanças na preparação de dados seriam:

- Adição de dados dos oponentes do time mandante e participante
- Separação de dados de empate dos parâmetros de vitória e derrota
- Adição de novas métricas de dados coletados
- Utilização dos dados coletados do Brasileirão

Com isso em mente, a estrutura de dados foi similar a anterior, com um processamento linear de dados. Os parâmetros finais elaborados foram:

Além disso, para a avaliação dos resultados foi feito uma conversão, utilizando o time, ano e rodada do id da partida do site de *oddspedia* para o id da partida do site *Transfersmarkt* e foi separado, por partida, as melhores apostas, ou seja, os maiores pagamentos para cada um dos resultados de cada partida.

5.2.2.3.2 Modelagem

Para a modelagem, foi aplicado a biblioteca *xgboost*, que é uma biblioteca especializada em *gradient boosted trees* e é comumente utilizado para problemas de classificação nesse contexto.

Quadro 4 – Parâmetros finais

	Time Mandante	**Time Visitante**
Informações do time	Idade média do time, Colocação, Valor da equipe titular.	Idade média do time, Colocação, Valor da equipe titular.
Médias por partida	Derrotas, empates e vitórias, gols totais, saldo de gols, chutes, chutes fora, defesas, escanteios e faltas.	Derrotas, empates e vitórias, gols totais, saldo de gols, chutes, chutes fora, defesas, escanteios e faltas.
Médias nas últimas quatro partidas	Gols totais e saldo de gols.	Gols totais e saldo de gols.
Informações de Oponentes	Colocação, Média de saldo de gols e gols na partida.	Colocação, Média de saldo de gols e gols na partida.

Fonte: Autor.

Além disso, foi separado os dados em grupos de treinamento, desenvolvimento e teste. Com o intuito do dataset de desenvolvimento ser utilizado para o *tuning* do modelo. De forma a deixar os resultados mais realistas, essa separação não foi feita aleatoriamente e sim por períodos de tempo.

Quadro 5 – Separação da base de dados

Base	**Intervalo de tempo**	**Total de jogos**
Treinamento	2020-2022	1187
Desenvolvimento	2023 | rodadas 1-10	100
Teste	2023 | rodadas 11-24	142

Fonte: Autor.

No *tuning* do modelo foi nescessário escolher entre os n melhores parâmetros e a profundidade da árvore, de forma a prevenir o máximo possível *overfitting*. Para isso foi testado a acurácia do modelo no dataset de desenvolvimento. Analisando o resultado foi definido que seria utilizado uma profundidade máxima de 4 e um número máximo de parâmetros de 8.

Assim uma vez feito *tuning* conseguimos assim avaliar a performance no *dataset* de teste, e foi obtido um resultado de 0.51 acurácia e as seguintes métricas de resultado:

5.2.2.4 Avaliação do modelo

Analisando os resultado fica evidente que o grande número de parâmetros projetados não beneficiou o modelo, como a quantidade de dados coletados o número de estimadores selecionado foram 6. Isso significa que a grande maioria dos parâmetros de treinamento do modelo foram descartados para evitar *overfitting*. Dessa forma, o procedimento para

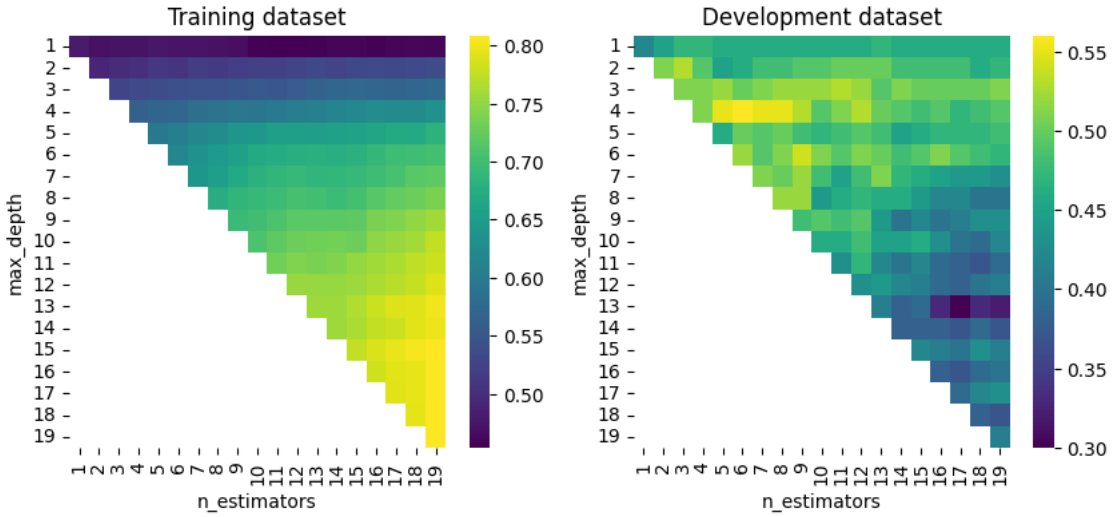

Figura 26 – Definição dos parâmetros de treinamento

Quadro 6 – Resultados modelo final xgboost

Classificação	**Precisão**	**recall**	**f1-score**
derrota	0.67	0.25	0.36
empate	0.26	0.12	0.17
vitória	0.54	0.86	0.66

Fonte: Autor.

a melhoria do modelo poderia seguir alguns possíveis caminhos. A sumarização dos parâmetros de treinamento de menor influência poderiam ajudar a obter um resultado melhor, por exemplo, poderiam-se juntar informações ofensivas como o número de defesas do time inimigo, escanteios e chutes para se obter um parâmetro que ajudaria a indicar um caracter ofensivo de um time e dar um contexto melhor para o jogo.

Outra resposta natural para esse problema seria uma base maior de treinamento. Apesar de realmente existir um contexto para os dados e existir um ponto em que os dados de outros períodos e outras regiões podem prejudicar o resultado do modelo, com a adição de mais dados é possível utilizar mais parâmetros de treinamento com menor impacto no *overfitting*. Quando foram adicionados os dois anos completos da Bundesliga no treinamento do modelo observou-se um melhora considerável tanto na base de desenvolvimento e treino, possivelmente se fossem coletadas mais ligas completas o desempenho poderia melhorar ainda mais. Infelizmente, no site utilizado para a coleta de dados não se encontrou mais ligas com estatísticas completas.

Comparando o modelo da primeira iteração com o da segunda com uma acurácia de 0.65, superior a 0.51. Esse resultado pode parecer surpreendente, mas pode ser explicado por alguns fatores. Primeiramente, e mais importante foi a construção da base de dados de treinamento e avaliação do primeiro modelo. Foram separados aleatoriamente valores

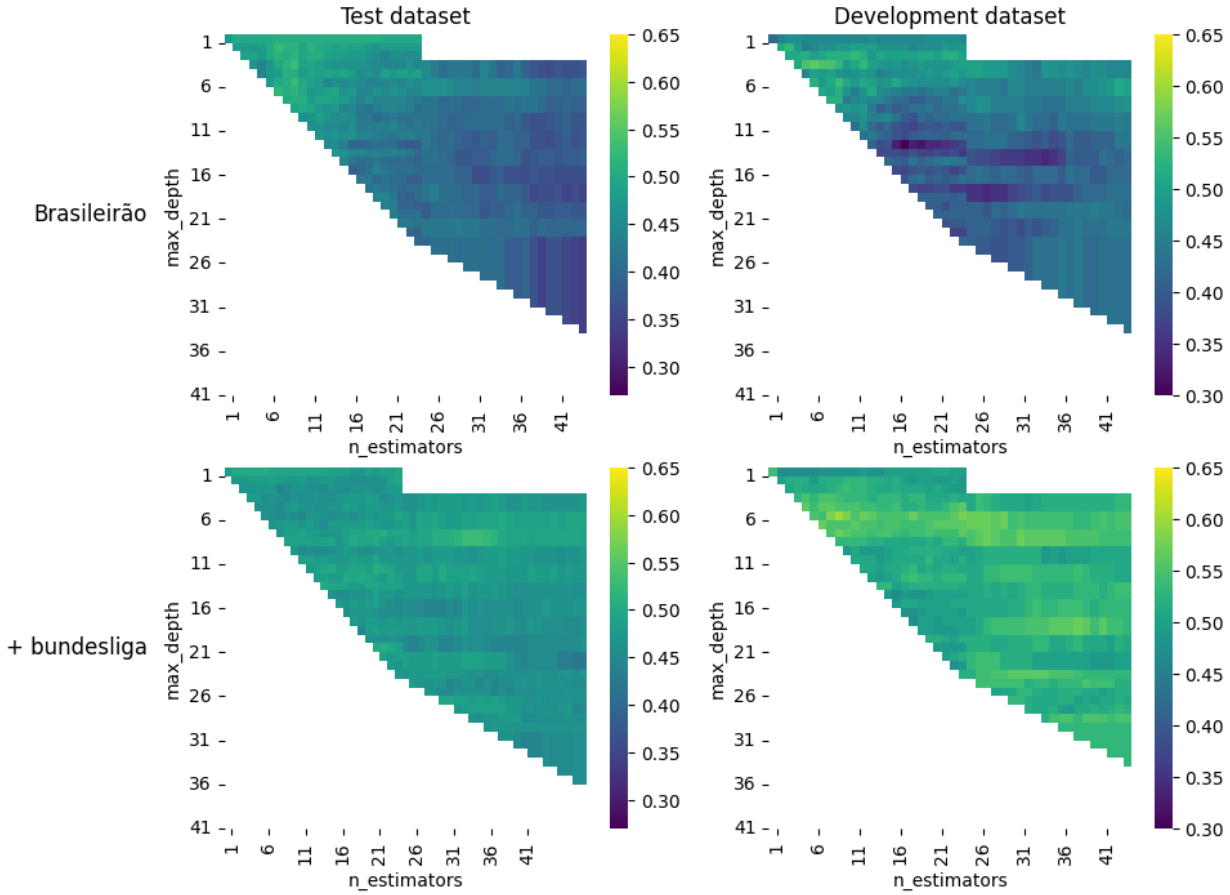

Figura 27 – Acurácia do modelo caso fosse adicionados dados da Bundesliga

de para treinamento e avaliação. Porém, como há uma dependência de dados entre as duas informações pode ter ocorrido uma interferência nos dados de avaliação que produziu um resultado enganoso. Quando se separou o dataset de forma mais similar a iteração final, utilizando a metade final de um campeonato como teste, observou-se uma queda de desempenho para uma acurácia de 0.55 que é mais similar à segunda iteração.

Além disso, pode-se destacar a semelhança maior entre a base de treino da primeira iteração com a segunda. Como a base de treinamento e de teste foi feitas com dados de ligas europeias do mesmo ano é teorizado que isso pode ter ajudado na semelhança dos resultados.

Uma ótica importante quando se olha para a visão geral do resultado desse modelos é aplicar sobre a visão do contexto de precisão de jogos de futebol. Partidas de futebol, em oposição a outros esportes como basquete ou basebol, são notoriamente difíceis de prever o resultado devido ao pequeno número de eventos determinantes de resultados, os gols. O fato de gols serem tão raros prejudica a predição em dois fatores cruciais, primeiramente é a variância de um resultado de uma partida. Um time pode ter tido mais oportunidade de gols, controle do jogo e em média, ter muito mais oportunidade de ter ganhado o jogo porém ter perdido mesmo assim. Segundamento, é a menor eficácia como parâmetro de

treino em comparação a pontuação de outros esportes.

No artigo *Incorporating domain knowledge in machine learning for soccer outcome prediction*(BERRAR; LOPES; DUBITZKY, 2018), foram analisados resultados de uma competição previsão de partidas de futebol e coloca enfase na dificuldade de criação de um modelo de predição devido a pequena margem de diferença de gols que determinam o resultado de uma partida e sugere uma cuidado modelagem de parâmetros de treino para a criação de um modelo.

Rank	Team	RPS_{avg}	Accuracy	Method
1	Team DBL*	0.2054	0.5194	*k*-NN and rating features
2	Team OH	0.2063	0.5243	Hubáček et al. (2018)
3	Team ACC	0.2083	0.5146	Constantinou (2018)
4	Team FK	0.2087	0.5388	Tsokos et al. (2018)
5	Team DBL*	0.2149	0.5049	XGBoost and recency features
6	Team HEM	0.2177	0.4660	N/A
7	League Priors	0.2255	0.4515	Prior information based on leagues
8	Team EB	0.2258	0.4854	N/A
9	Global Priors	0.2261	0.4515	Global Priors of win, draw, lose

Figura 28 – Resultados da competição de predição de partidas de futebol de 2017, com base de dados limitado a gols.

Dessa forma, é concluído que para a melhoria do modelo deveria-se utilizar uma base de dados maior e focar menos na criação de diversos parâmetros de treino e mais no desenvolvimento de poucos parâmetros mais eficazes.

5.2.2.5 Testes de aplicabilidade do modelo no mercado de apostas

Para examinar o modelo em um cenário de apostas, foi calculado, entre os diversos sites de aposta, quais eram os melhores pagamentos para cada um dos cenários de aposta. Ou seja, em qual site dado uma predição do modelo, teria o melhor pagamento caso essa predição estivesse correta. Utilizando esses dados então, foi calculado a média de lucro que o modelo teria utilizando.

Para isso foi suposto que o modelo apostava em todos os casos uma quantia fixa de dinheiro e então foi calculado o total ganho dividido pelo total apostado . Utilizando essa medida, foi obtido lucro de 5%, esse valor pode ser comparado com a técnica comumente empregada comumente de sempre apostar em vitória de casa que forneceu um lucro de 7%. Assim, pode-se concluir que o modelo atual não apresenta um bom resultado para apostas.

Com o intuito de testar o potencial do uso de modelos em apostas foi testado a criação de um modelo de predição diferença de gols esperados em uma partida. A ideia por trás dessa técnica seria que é possível utilizar apenas casos os quais os modelo tem maior assertividade do resultado.

Utilizando novamente o *xgboost* foi criado um modelo de predição de gols de um modelo criado inspirado no código feito por um usuário chamado @radar para a competição de March Madness, o qual são treinados 10 instancias de *gradient boosted trees*, feito a

média de previsão da previsão de gols delas e finalmente é interpolado um *spline* a partir do modelo para fazer a predição. Com isso se obteve um protótipo do que seria um modelo com sensibilidade de assertividade para apostas.

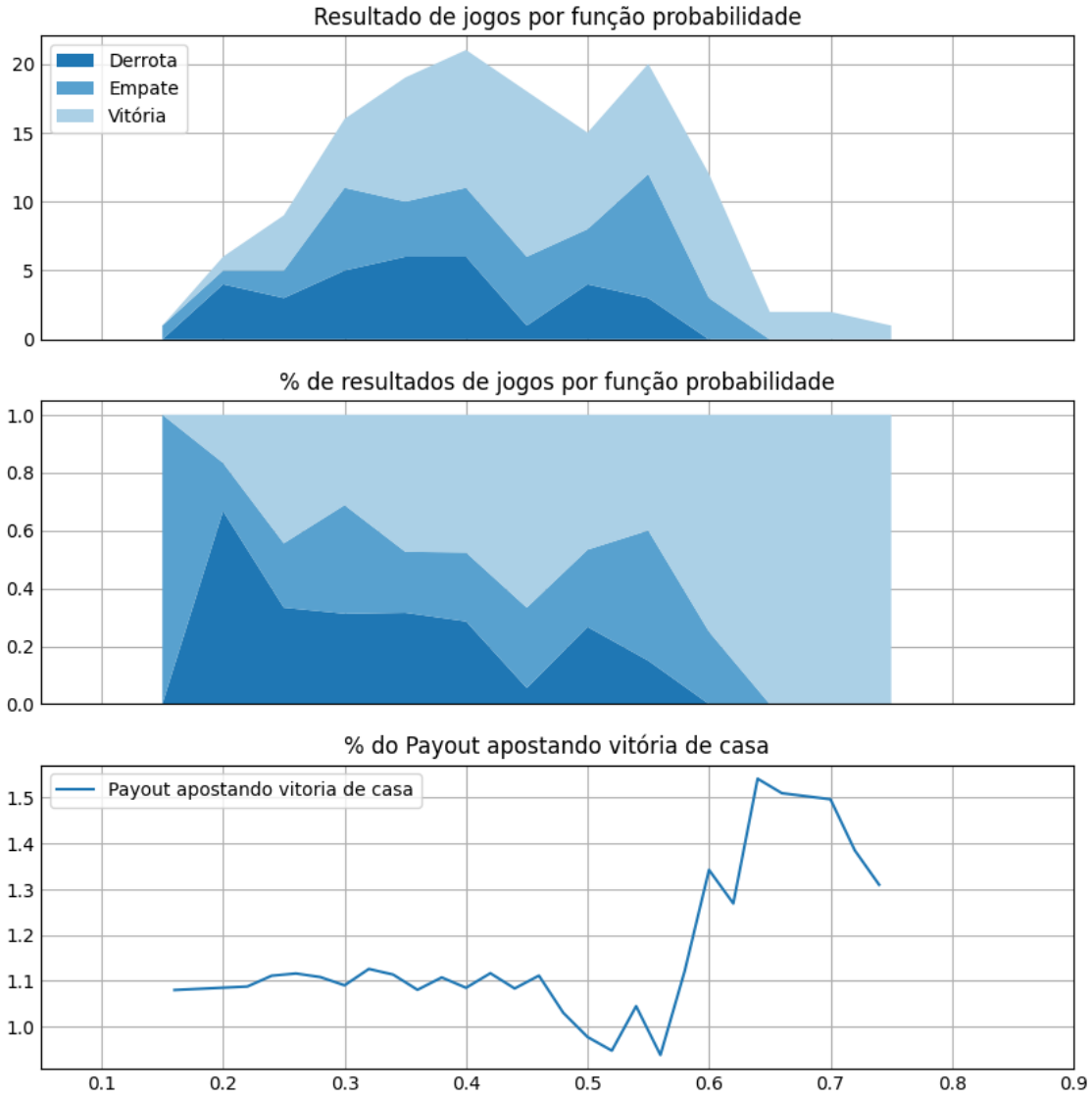

Figura 29 – Resultados protótipo de modelo com sensibilidade para assertividade.

Observando o resultado, pode-se dizer que o conceito inspira promessa pois foi obtido uma forte correlação na base de dados de teste entre o resultado do jogo e as predições com maior assertividade, porém considerando o pouco volume de jogos desse tipo, é certo que o modelo precisaria de mais testes e refinamento para realmente poder ser utilizado com segurança na prática.

6 Considerações Finais

6.1 Conclusões do Projeto de Formatura

Analisando alguns dos resultados primordiais do projeto, podemos chegar a algumas conclusões importantes.

O Quadro 6, que mostra a precisão do nosso modelo preditor, pode parecer decepcionante a primeira vista. A precisão do modelo, que não passa dos 70%, faz parecer que o modelo não é aplicável ao contexto apostas de partidas de futebol. Contudo, apesar de não ser recomendável utilizar o modelo atualmente, a análise da Figura 29, mostra um potencial a ser explorado com maior refinamento do uso de modelos preditivos nesse contexto, apesar de ser necessário enfatizar que um monitoramento de várias casas de apostas e uso outras técnicas de apostas em conjunto com o modelo podem ser essências para o funcionamento de um sistema completo.

6.2 Contribuições

Esse projeto explorou técnicas preditivas de partidas de futebol no campeonato Brasileiro que é menos explorado que o Europeu, além disso foi construída uma base de dados não só de partidas com métricas mas também de apostas nesse campeonato.

6.3 Perspectivas de Continuidade

Esse projeto de conclusão de curso pode ser muito mais considerado como um protótipo, em questões de continuidade o foco seria provavelmente na aplicabilidade do modelo. O a frente do modelo em si, poderia focar nos pontos levantados na avaliação da segunda iteração e nos testes de aplicabilidade do modelo ou seja, para o modelo poderia-se focar em:https://www.overleaf.com/project/647e43ad24123b1bf5349363

- Coleta de um dataset detalhado com um número maior de dados
- Refinamento e diminuição de parâmetros de treino
- Criação de um modelo de spline com sensibilidade para assertividade do modelo
- Integração do modelo com técnicas comuns de aposta

Se realmente fosse tomado esse foco em aplicabilidade no mercado de apostas, aplicativo poderia adicionar funcionalidades para integrar esse foco com dados como:

- Recomendação de apostas do modelo
- Integração com diferentes sites de apostas

Referências

BEAST", T. D. *AdaBoost, Gradient Boosting, XG Boost:: Similarities Differences*. [S.l.], 2023. Citado na página 24.

BERRAR, D.; LOPES, P.; DUBITZKY, W. Incorporating domain knowledge in machine learning for soccer outcome prediction. *Machine Learning*, v. 108, p. 97–126, 2018. Citado na página 60.

FLUTTER. *ExpandedWidget*. [S.l.], 2023. Citado na página 38.

FLUTTER. *Flutter Documentation*. [S.l.], 2023. Citado na página 34.

MIENYE, I. D. Prediction performance of improved decision tree-based algorithms: a review. *Procedia Manufacturing*, 2019. Citado na página 20.

OLSSON, M. A comparison of performance and looks between flutter and native applications. 2020. Citado na página 33.

POSTGREST. *postgREST Documentation*. [S.l.], 2023. Citado na página 31.

SCHRöER, C. A systematic literature review on applying crisp-dm process model. *Procedia Computer Science*, 2021. Citado na página 47.

STORAGE, S. *Supabase Storage Documentation*. [S.l.], 2023. Citado na página 32.

SUPABASE. *Supabase Architeture*. [S.l.], 2022. Citado na página 31.