



PUC Minas

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

NÚCLEO DE EDUCAÇÃO A DISTÂNCIA

Pós-graduação *Lato Sensu* em Analytics e Business Intelligence

RELATÓRIO TÉCNICO

UMA SOLUÇÃO DE ANALYTICS E BUSINESS INTELLIGENCE PARA ANÁLISE
DE DADOS DO CAMPEONATO BRASILEIRO DE FUTEBOL DA ERA DE PONTOS
CORRIDOS

Módulo C – Análises Avançadas

Salomão Fernandes de Freitas Júnior

Belo Horizonte

2022

SUMÁRIO

1. Introdução	3
2. Análises Avançadas	3
2.1. Descrição Geral da Solução.....	3
2.2. Ambiente Tecnológico	5
2.3. Apresentação do Modelo de Análise.....	5
2.3.1. Importação das Bibliotecas e Leitura dos <i>Datasets</i>	5
2.3.2. Preparação da Base de Dados e Seleção de Atributos.....	6
2.3.3. Codificação dos Dados Categóricos	8
2.3.4. Particionamento da Base para Treinamento e Testes do Modelo....	9
2.3.5. Testes de Variações nos Valores dos Parâmetros da Árvore	11
2.3.6. Exibição da Árvore de Decisão.....	12
3. Conclusões	15
3.1. Análise Visual - Dashboards.....	15
3.2. Análises Avançadas – Machine Learning	16
4. Links	17
REFERÊNCIAS.....	18

1. Introdução

Durante as visualizações dos *dashboards* produzidos no [Módulo B do Relatório Técnico](#), já foi possível realizar uma análise prévia, de forma visual, dos dados de nosso projeto, conforme descrito no tópico 2 do referido módulo.

Neste Módulo C do Relatório Técnico vamos tratar de duas entregas muito relevantes para os gestores que demandam soluções de análise de dados. Inicialmente vamos abordar as análises avançadas a partir da base de dados estruturada nos módulos anteriores identificando novos conhecimentos no contexto do projeto, através da aplicação de técnicas de aprendizado de máquina (*machine learning*). Posteriormente, vamos fazer uma análise de tudo que foi obtido com o projeto para indicar possíveis intervenções que podem alavancar diferenciais competitivos para a organização.

2. Análises Avançadas

2.1. Descrição Geral da Solução

A partir da base de dados do *Data Warehouse* desenvolvido no [Módulo A do Relatório Técnico](#), realizamos análises de dados através da aplicação de técnicas de aprendizado de máquina. Nossas análises são independentes do *dashboard* criado no Módulo B do Relatório Técnico.

Para realização das análises, utilizamos a estratégia de gerar 2 arquivos **csv** a partir do conteúdo de 2 *views* criadas sobre dados do *data warehouse* do projeto. Os arquivos utilizados em nossa análise de dados são o *vw_completo_raio_x2.csv* e *vw_dim_clube_adversario.csv*, os quais possuem informações sobre os jogos e sobre os clubes, respectivamente. Ambos podem ser obtidos no repositório do projeto, no link [Arquivos Base – Módulo C](#).

Assim, pelas características dos atributos da base de dados a serem utilizados em nossa análise, optamos por utilizar uma técnica de aprendizado de máquina supervisionado. Algoritmos de aprendizado supervisionado, basicamente, tentam modelar relacionamentos entre valores de atributos de entrada e suas saídas (atributos-alvo ou atributos-classe) correspondentes, a partir de dados de

treinamento. Assim, tornam-se capazes de prever valores de saída para novos dados de entrada. [1.Sarkar, Dipanjan]

Existem dois principais tipos de métodos de aprendizado supervisionado: **Classificação**, no qual o objetivo é prever valores de saída para um atributo categórico (conhecidos como classes); e **Regressão**, que por outro lado, objetivam a estimação de valores de saída para atributos numéricos contínuos. [1.Sarkar, Dipanjan], [2.Guido, Sarah]

Na nossa análise, optamos por utilizar a técnica de **Classificação**, através de um algoritmo de Árvore de Decisão, uma vez que nossa solução tem por objetivo prever o valor do atributo categórico RESULTADO (VITÓRIA, EMPATE, DERROTA), a partir dos valores conhecidos dos atributos ESQUEMA_TATICO, REGIAO_ADVERSARIO, TURNO_DIA e CONDICAÇÃO.

Árvores de decisão são largamente utilizadas para construção de modelos tanto de classificação quanto de regressão. Essencialmente, elas se constituem em uma hierarquia de perguntas SIM/NÃO, conduzindo a uma decisão. [2.Guido, Sarah]

Uma árvore de decisão usa a estratégia de dividir para conquistar para resolver um problema de decisão. Um problema complexo é dividido em problemas mais simples, aos quais, recursivamente, é aplicada a mesma estratégia. Essa é a ideia básica por trás de algoritmos baseados em árvore de decisão, tais como ID3, ASSISTANT, CART e C4.5. Formalmente, uma árvore de decisão é um grafo acíclico direcionado em que cada nó ou é um nó de divisão, com dois ou mais sucessores, ou um nó folha [3.Faceli, Katti]

Na figura 1, a seguir (extraída de [2.Guido, Sarah]), ilustramos, a título de exemplo, uma árvore de decisão que analisa os valores de determinados atributos de entrada, para prever a classe do atributo-alvo (tipo de animal).

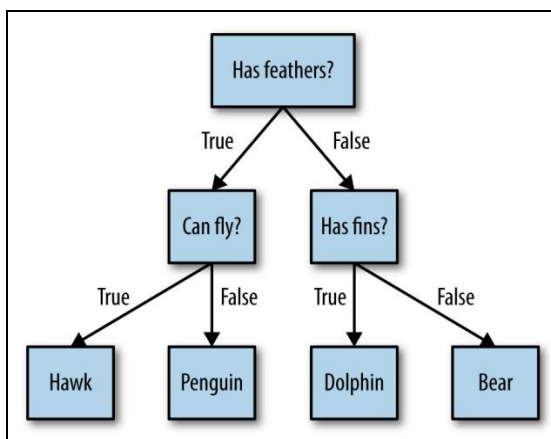


FIG. 1 – Representação gráfica de uma árvore de decisão [2.Guido, Sarah]

2.2. Ambiente Tecnológico

Desenvolvemos nosso modelo utilizando a plataforma de nuvem Google Colab [<https://colab.research.google.com/>] (mostrada na figura 2), em linguagem Python, fazendo uso de bibliotecas próprias para análise de dados e *machine learning*, como *pandas*, *numpy*, *matplotlib*, *sklearning* e *seaborn*, conforme será apresentado no item 2.3, a seguir.

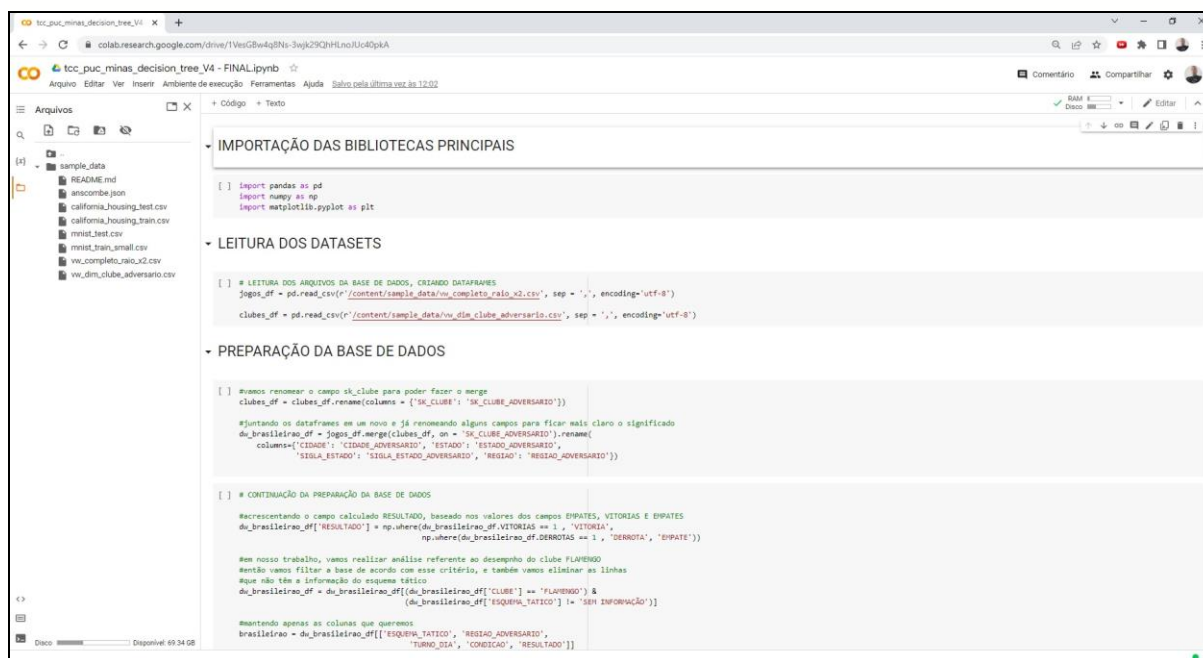


FIG. 2 – Ambiente Google Colab

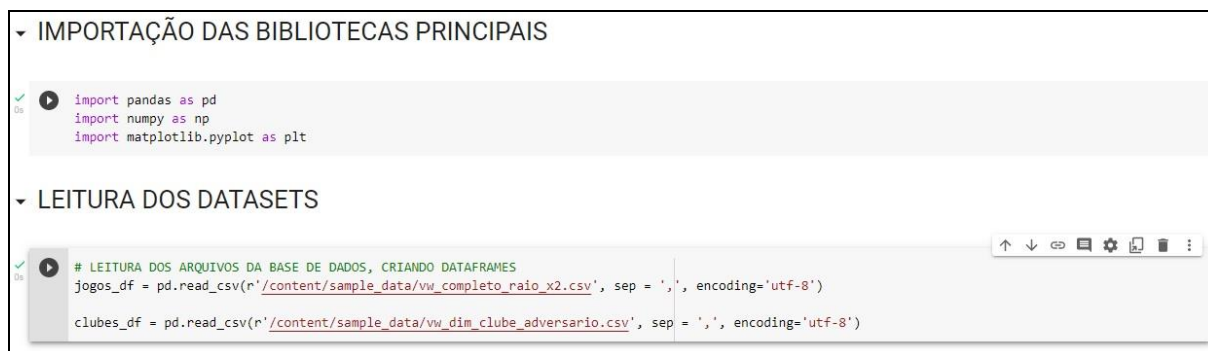
2.3. Apresentação do Modelo de Análise

O arquivo completo do notebook Python descrito neste tópico, está disponível no repositório do projeto, em [Projeto Python – Módulo C](#). O código Python desenvolvido neste trabalho foi baseado nos modelos apresentados em [6.De Paula, Hugo] e [7.Alves, Pedro].

2.3.1. Importação das Bibliotecas e Leitura dos *Datasets*

Inicialmente, fazemos a importação das bibliotecas principais, para uso no nosso modelo, quais sejam: *pandas*, *numpy* e *matplotlib*. Em seguida, fazemos a leitura dos arquivos csv (citados no tópico 2.1), criando os *dataframes* **jogos_df** e **clubes_df**, conforme mostrado na figura 3. Nesse ponto, os *dataframes* contêm o

conjunto de dados, extraído do *Data Warehouse* do projeto, os quais utilizaremos para nossa análise.



```
▼ IMPORTAÇÃO DAS BIBLIOTECAS PRINCIPAIS

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

▼ LEITURA DOS DATASETS

# LEITURA DOS ARQUIVOS DA BASE DE DADOS, CRIANDO DATAFRAMES
jogos_df = pd.read_csv(r'/content/sample_data/vw_completo_raio_x2.csv', sep = ',', encoding='utf-8')

clubes_df = pd.read_csv(r'/content/sample_data/vw_dim_clube_adversario.csv', sep = ',', encoding='utf-8')
```

FIG. 3 – Leitura dos Datasets

2.3.2. Preparação da Base de Dados e Seleção de Atributos

Em seguida, trabalhamos sobre os *dataframes*, com o objetivo de preparar os dados antes da execução da análise. Conforme mostra o código Python da figura 4, inicialmente juntamos os 2 *dataframes* originais (**clubes_df** e **jogos_df**), em um só *dataframe* (**dw_brasileirao_df**). Para isso, utilizamos o comando *merge* da classe *dataframe*, fazendo a junção das bases pelo campo SK_CLUBE_ADVERSARIO, presente em ambos.

Em seguida, já trabalhando sobre o *dataframe* **dw_brasileirao_df**, acrescentamos o campo calculado RESULTADO, baseado no valor dos campos VITORIA, EMPATE e DERROTA, com o uso do comando *where* da biblioteca *pandas*. Na próxima linha de código mostrada na figura 4, fazemos uma filtragem dos dados do *dataframe* **dw_brasileirao_df**, selecionando somente as linhas relativas ao clube FLAMENGO, e que possuem informação no campo ESQUEMA_TATICO.

Para finalizar a preparação dos dados, eliminamos do *dataframe* as colunas desnecessárias, mantendo apenas os campos ESQUEMA_TATICO, REGIAO_ADVERSARIO, TURNO_DIA, CONDICAO e RESULTADO.

Esse processo de preparação e ajustes dos dados, e de seleção dos atributos (*features*) para a análise, denomina-se **feature selection** ou **feature engineering**, devendo se seguir alguns princípios na sua realização, conforme descrito em [6.De Paula, Hugo].

PREPARAÇÃO DA BASE DE DADOS

```
#vamos renomear o campo sk_clube para poder fazer o merge
clubes_df = clubes_df.rename(columns = {'SK_CLUBE': 'SK_CLUBE_ADVERSARIO'})

#juntando os dataframes em um novo e já renomeando alguns campos para ficar mais claro o significado
dw_brasileirao_df = jogos_df.merge(clubes_df, on = 'SK_CLUBE_ADVERSARIO').rename(
    columns={'CIDADE': 'CIDADE_ADVERSARIO', 'ESTADO': 'ESTADO_ADVERSARIO',
            'SIGLA_ESTADO': 'SIGLA_ESTADO_ADVERSARIO', 'REGIAO': 'REGIAO_ADVERSARIO'})

[ ] # CONTINUAÇÃO DA PREPARAÇÃO DA BASE DE DADOS

#acrescentando o campo calculado RESULTADO, baseado nos valores dos campos EMPATES, VITORIAS E EMPATES
dw_brasileirao_df['RESULTADO'] = np.where(dw_brasileirao_df.VITORIAS == 1, 'VITORIA',
    np.where(dw_brasileirao_df.DERROTAS == 1, 'DERROTA', 'EMPATE'))

#em nosso trabalho, vamos realizar análise referente ao desempenho do clube FLAMENGO
#então vamos filtrar a base de acordo com esse critério, e também vamos eliminar as linhas
#que não têm a informação do esquema tático
dw_brasileirao_df = dw_brasileirao_df[(dw_brasileirao_df['CLUBE'] == 'FLAMENGO') &
    (dw_brasileirao_df['ESQUEMA_TATICO'] != 'SEM INFORMAÇÃO')]

#mantendo apenas as colunas que queremos
brasileirao = dw_brasileirao_df[['ESQUEMA_TATICO', 'REGIAO_ADVERSARIO',
    'TURNIO_DIA', 'CONDICAO', 'RESULTADO']]

#Após tratamento da base, vamos resetar os índices para ficarem contíguos
brasileirao.reset_index(inplace=True, drop=True)
```

FIG. 4 – Preparação da Base de Dados

Nas células seguintes do notebook Python, mostradas na figura 5, fazemos uma visualização da estrutura e de uma parte do conjunto de dados contido no *dataframe*, bem como de um resumo estatístico dos mesmos, que exibe para cada coluna, o total de linhas que possuem o valor preenchido (**count**), o total de valores distintos (**unique**), o valor com maior ocorrência (**top**) e o número de ocorrências deste (**freq**).

Visualizando o dataframe

brasileirao

	ESQUEMA_TATICO	REGIAO_ADVERSARIO	TURNIO_DIA	CONDICAO	RESULTADO
0	4-2-3-1	SUDESTE	TARDE	VISITANTE	DERROTA
1	4-2-3-1	SUDESTE	TARDE	MANDANTE	DERROTA
2	4-2-3-1	SUDESTE	MANHÃ	MANDANTE	VITORIA
3	4-2-3-1	SUDESTE	TARDE	VISITANTE	EMPATE
4	4-2-3-1	SUDESTE	TARDE	VISITANTE	DERROTA
...
263	4-4-2	SUDESTE	NOITE	VISITANTE	EMPATE
264	4-2-3-1	SUDESTE	NOITE	MANDANTE	DERROTA
265	4-2-3-1	SUDESTE	NOITE	VISITANTE	EMPATE
266	4-4-2	CENTRO-OESTE	NOITE	VISITANTE	VITORIA
267	4-2-3-1	CENTRO-OESTE	NOITE	MANDANTE	EMPATE

268 rows x 5 columns

[] brasileirao.describe()

	ESQUEMA_TATICO	REGIAO_ADVERSARIO	TURNIO_DIA	CONDICAO	RESULTADO
count	268	268	268	268	268
unique	9	4	3	2	3
top	4-2-3-1	SUDESTE	NOITE	VISITANTE	VITORIA
freq	185	130	135	135	141

FIG. 5 – Visualização da Estrutura do *dataframe* e descrição estatística dos dados

Nas células mostradas na figura 6, colocamos na variável *class_names* uma lista com os valores possíveis para a coluna RESULTADO, que representa o atributo-alvo (ou atributo-classe) da nossa análise, e em seguida já exibimos a lista para conferência. Depois, através do comando *info()* do *dataframe*, exibimos informação para verificar o tipo de dados de cada coluna, que no caso é *object*, indicando que não se tratam de atributos numéricos, e sim categóricos. No caso, já sabemos que estes atributos são do tipo *String*.

```
[7] # pegando os valores possíveis do campo RESULTADO, que será nosso atributo alvo
# será usado mais tarde na visualização da árvore
class_names = list(brasileirao.RESULTADO.unique())
class_names

['DERROTA', 'VITORIA', 'EMPATE']

[8] brasileirao.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 268 entries, 0 to 267
Data columns (total 5 columns):
#   Column                Non-Null Count  Dtype
---  -
0   ESQUEMA_TATICO         268 non-null    object
1   REGIAO_ADVERSARIO     268 non-null    object
2   TURNO_DIA              268 non-null    object
3   CONDICAO               268 non-null    object
4   RESULTADO              268 non-null    object
dtypes: object(5)
memory usage: 10.6+ KB
```

FIG. 6 – Visualização de informações do *dataframe*

2.3.3. Codificação dos Dados Categóricos

Até o passo anterior, preparamos a base deixando somente os dados a serem úteis na análise. No entanto, para aplicação do algoritmo de Classificação (Árvore de Decisão), precisamos ainda codificar os dados categóricos, transformando-os em numéricos. Fazemos isso conforme mostrado na figura 7, utilizando o método *fit_transform()* da classe *LabelEncoder* da biblioteca *sklearn.preprocessing*. A fim de preservar os dados com os valores originais, criamos um novo *dataframe* (*brasileirao_enc*), o qual receberá os dados codificados para formato numérico.

```
APLICANDO O LABEL ENCODER

[9] from sklearn.preprocessing import LabelEncoder

#Atribuição apenas para manter os valores do DF brasileiro
#Vamos usar outro DF com os dados codificados
brasileirao_enc = brasileiro

# APLICANDO O LABEL ENCODER PARA CODIFICAÇÃO DOS ATRIBUTOS CATEGÓRICOS PARA NUMÉRICOS, PARA PODER APLICAR O ALGORITMO DE CLASSIFICAÇÃO
le = LabelEncoder()

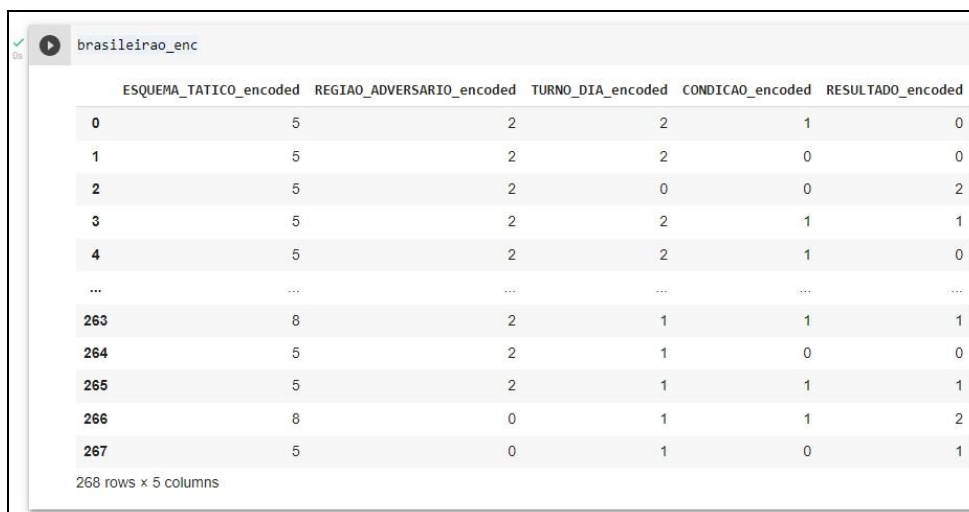
colunas = brasileiro_enc.dtypes.reset_index()

categ_cols = colunas[colunas[0] == 'object']['index'].to_list()

# Aplicando o encode
for i in categ_cols:
    brasileiro_enc[str(i) + '_encoded'] = le.fit_transform(brasileirao_enc[i])
    brasileiro_enc = brasileiro_enc.drop(i, axis = 1)
```

FIG. 7 – Codificação dos valores categóricos para numéricos

Em seguida, visualizamos o *dataframe* com os dados numéricos codificados, conforme mostrado na figura 8. Para comparação, pode-se observar os dados originais ilustrado na figura 5.



The screenshot shows a Jupyter Notebook interface with a play button icon and the text 'brasileirao_enc'. Below it is a table representing the dataframe. The table has 5 columns: 'ESQUEMA_TATICO_encoded', 'REGIAO_ADVERSARIO_encoded', 'TURNO_DIA_encoded', 'CONDICAO_encoded', and 'RESULTADO_encoded'. The rows are indexed from 0 to 267. The first five rows (0-4) show values 5, 2, 2, 1, 0. Rows 263-267 show values 8, 2, 1, 1, 1; 5, 2, 1, 0, 0; 5, 2, 1, 1, 1; 8, 0, 1, 1, 2; and 5, 0, 1, 0, 1 respectively. At the bottom, it says '268 rows x 5 columns'.

	ESQUEMA_TATICO_encoded	REGIAO_ADVERSARIO_encoded	TURNO_DIA_encoded	CONDICAO_encoded	RESULTADO_encoded
0	5	2	2	1	0
1	5	2	2	0	0
2	5	2	0	0	2
3	5	2	2	1	1
4	5	2	2	1	0
...
263	8	2	1	1	1
264	5	2	1	0	0
265	5	2	1	1	1
266	8	0	1	1	2
267	5	0	1	0	1

268 rows x 5 columns

FIG. 8 – Visualização dos valores codificados no *dataframe* *brasileirao_enc*

2.3.4. Particionamento da Base para Treinamento e Testes do Modelo


Nas células exibidas na figura 9, fazemos efetivamente a aplicação do algoritmo de árvore de decisão. Primeiramente, particionamos a nossa base (presente no *dataframe* *brasileirao_enc*), em base de treinamento e de teste. À variável *x*, atribuímos um *dataframe* somente com os atributos de entrada (*features*) da análise (ESQUEMA_TATICO, REGIAO_ADVERSARIO, TURNO_DIA, CONDICAO), e na variável *y*, atribuímos um *dataframe* somente com o atributo-alvo (RESULTADO).

Em seguida, aplicamos o método *train_test_split()* da biblioteca *sklearn.model_selection*, para efetivamente fazer a divisão da base. Na chamada do método, passamos as variáveis *x* e *y*, e alguns parâmetros de configuração, como *test_size*, que indica o percentual do tamanho da base de teste, no caso 25%. Com isso o método particionará a nossa base, reservando 75% para treinamento e 25% para testes, atribuindo-os às variáveis *x_train*, *x_test*, *y_train* e *y_test*.

Com as bases de treinamento e testes já definidas, podemos carregar e treinar nosso modelo. Aplicamos o método *DecisionTreeClassifier()* para criar a nossa árvore de decisão (variável *brasileirao_tree*). Na chamada do método passamos alguns parâmetros de configuração, como *max_depth* e *max_leaf_nodes*,

conforme mostrado na figura 9. Pode-se experimentar diferentes valores desses parâmetros, para encontrar um melhor desempenho do algoritmo. Para de fato aplicar o treinamento no nosso modelo, chamamos o método *fit()* do objeto que representa a nossa árvore de decisão, *brasileirao_tree* (da classe *DecisionTreeClassifier*), passando como parâmetro a nossa base de treinamento (*x_train* e *y_train*).

No treinamento, o algoritmo tenta aprender a prever o valor do atributo-alvo, a partir dos valores dos atributos de entrada (*features*). Após o treinamento do modelo, aplicamos o método *predict()* do objeto *brasileirao_tree* nas nossas bases de treinamento (*x_train*) e de testes (*x_test*). Os valores das predições são atribuídos às variáveis *y_pred_train* e *y_pred_test*, respectivamente.



```
[13] from sklearn.model_selection import train_test_split

x = brasileiro_enc.drop('RESULTADO_encoded', axis = 1)
y = brasileiro_enc['RESULTADO_encoded']

x_train,x_test,y_train,y_test = train_test_split(x,y, test_size=0.25, random_state=42)

PARTICIONANDO A BASE DE DADOS (TREINO E TESTE)

[14] from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

brasileirao_tree = DecisionTreeClassifier(max_depth=10, max_leaf_nodes=20, criterion='gini')

brasileirao_tree = brasileiro_tree.fit(x_train, y_train)

y_pred_train = brasileiro_tree.predict(x_train)
y_pred_test = brasileiro_tree.predict(x_test)

CARREGANDO E TREINANDO O MODELO
```

FIG. 9 – Particionamento da base, Carga e Treinamento do Modelo

Nas células seguintes, mostradas na figura 10, exibimos o resultado dos testes, comparando os valores do atributo-alvo na base de treinamento e testes (*y_train* e *y_test*) com os valores gerados na predição (*y_pred_train* e *y_pred_test*). Com a função *accuracy_score()*, calculamos a acurácia do treinamento e do teste. Podemos notar que, com os parâmetros utilizados, não foi possível obter uma ótima acurácia no nosso modelo, pois a mesma ficou em 59,7% no treinamento e 50,7% no teste. Também exibimos a matriz de confusão (com a função *confusion_matrix()*).

Na matriz de confusão podemos visualizar os comparativos para cada valor do atributo-alvo, RESULTADO (DERROTA, EMPATE, VITORIA), entre os valores reais da base de testes e os valores da predição. Exibimos a visualização da matriz de confusão tanto em formato texto, como em um gráfico de mapa de calor, com a função `heatmap()` da biblioteca `seaborn`.

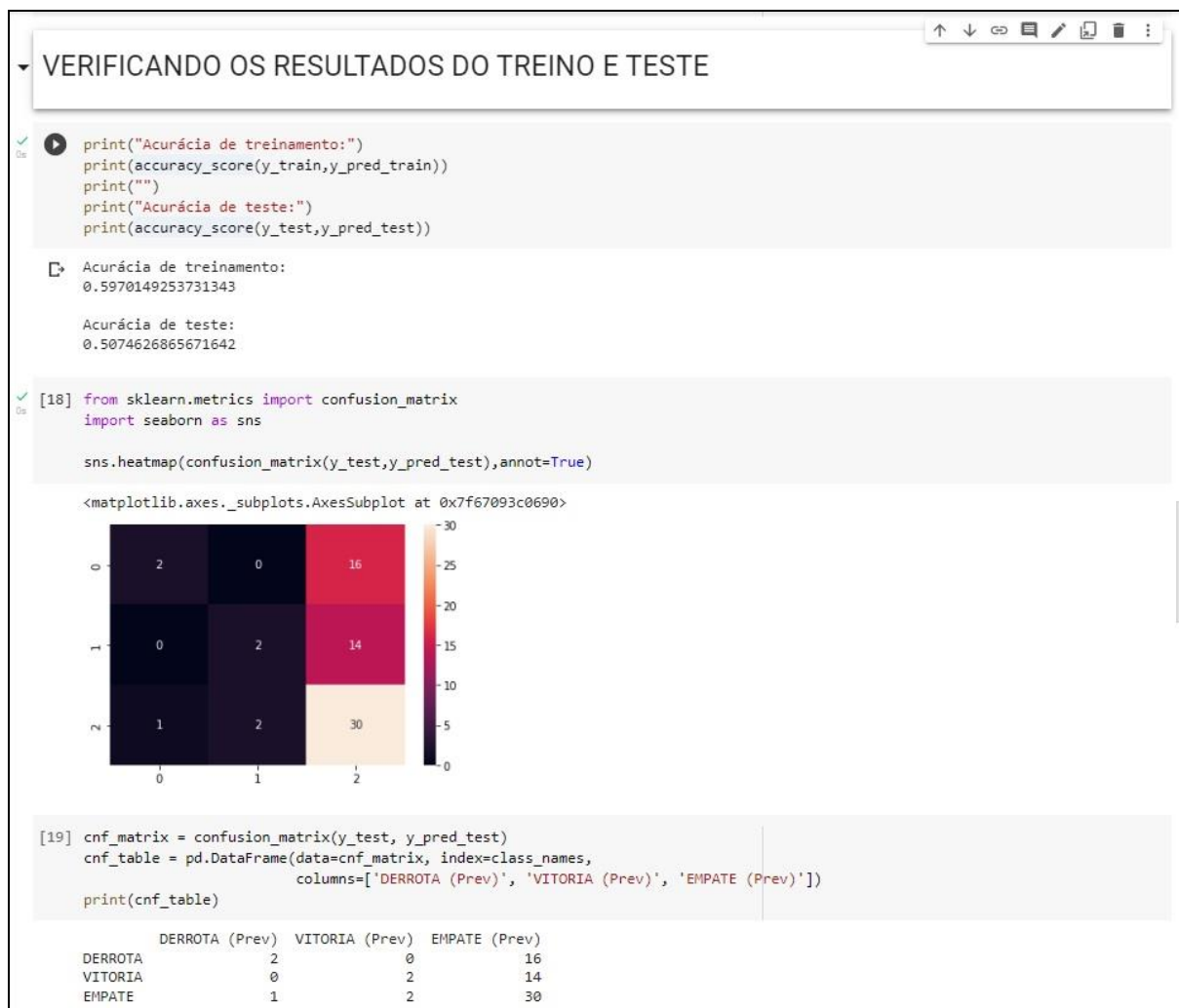
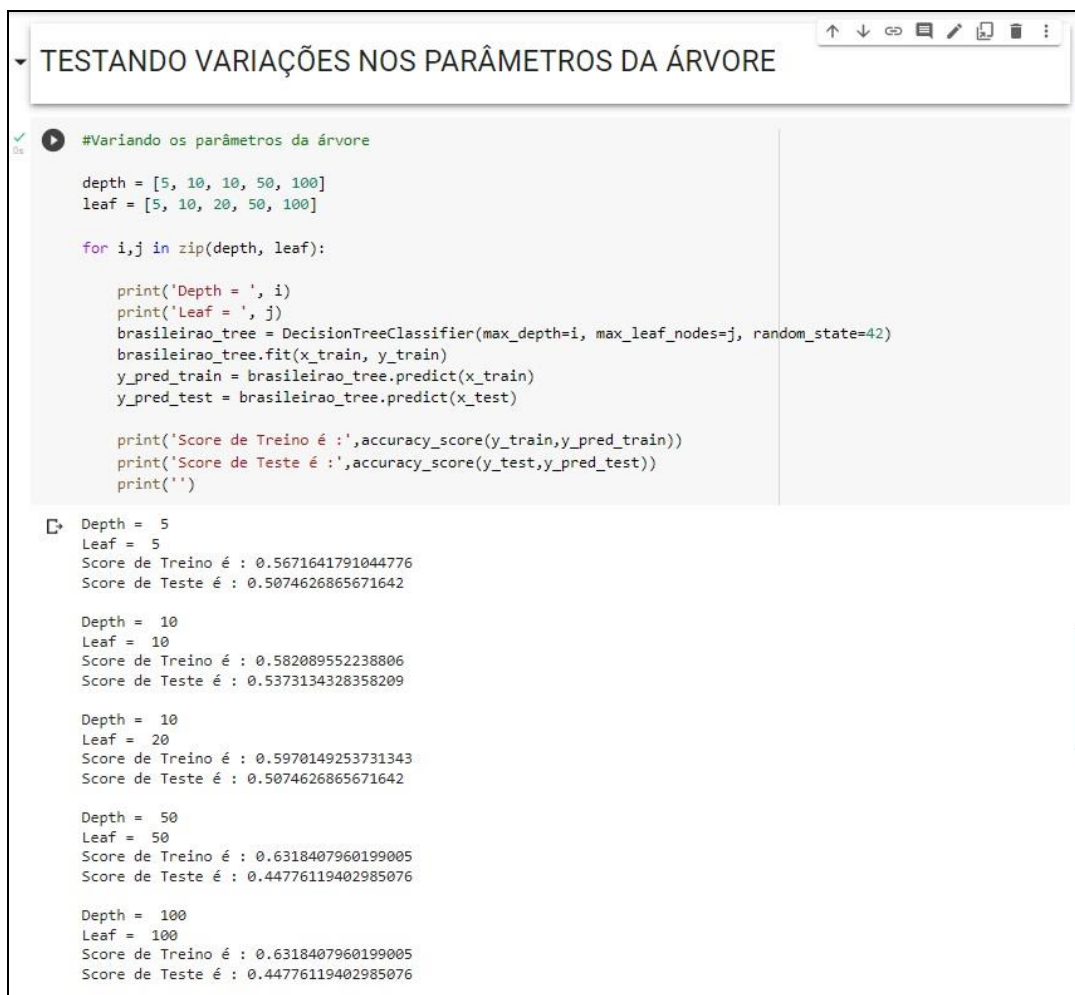


FIG. 10 – Verificação dos Resultados e Acurácia do Modelo

2.3.5. Testes de Variações nos Valores dos Parâmetros da Árvore

Uma boa prática na construção de modelos de *machine learning* é realizar o ajuste dos parâmetros para tentar encontrar um melhor desempenho do algoritmo [6.De Paula, Hugo] e [7.Alves, Pedro]. Dessa forma, mostramos na célula seguinte do notebook Python do nosso modelo (figura 11), uma tentativa de variação dos valores dos parâmetros `max_depth` e `max_leaf_nodes`, do método `DecisionTreeClassifier()`, para tentar obter um melhor desempenho, e exibimos os

valores da acurácia com os diversos valores desses parâmetros, para fins de comparação.



```

TESTANDO VARIAÇÕES NOS PARÂMETROS DA ÁRVORE

#Variando os parâmetros da árvore

depth = [5, 10, 10, 50, 100]
leaf = [5, 10, 20, 50, 100]

for i,j in zip(depth, leaf):

    print('Depth = ', i)
    print('Leaf = ', j)
    brasileiro_tree = DecisionTreeClassifier(max_depth=i, max_leaf_nodes=j, random_state=42)
    brasileiro_tree.fit(x_train, y_train)
    y_pred_train = brasileiro_tree.predict(x_train)
    y_pred_test = brasileiro_tree.predict(x_test)

    print('Score de Treino é :',accuracy_score(y_train,y_pred_train))
    print('Score de Teste é :',accuracy_score(y_test,y_pred_test))
    print('')

Depth = 5
Leaf = 5
Score de Treino é : 0.5671641791044776
Score de Teste é : 0.5074626865671642

Depth = 10
Leaf = 10
Score de Treino é : 0.582089552238806
Score de Teste é : 0.5373134328358209

Depth = 10
Leaf = 20
Score de Treino é : 0.5970149253731343
Score de Teste é : 0.5074626865671642

Depth = 50
Leaf = 50
Score de Treino é : 0.6318407960199005
Score de Teste é : 0.44776119402985076

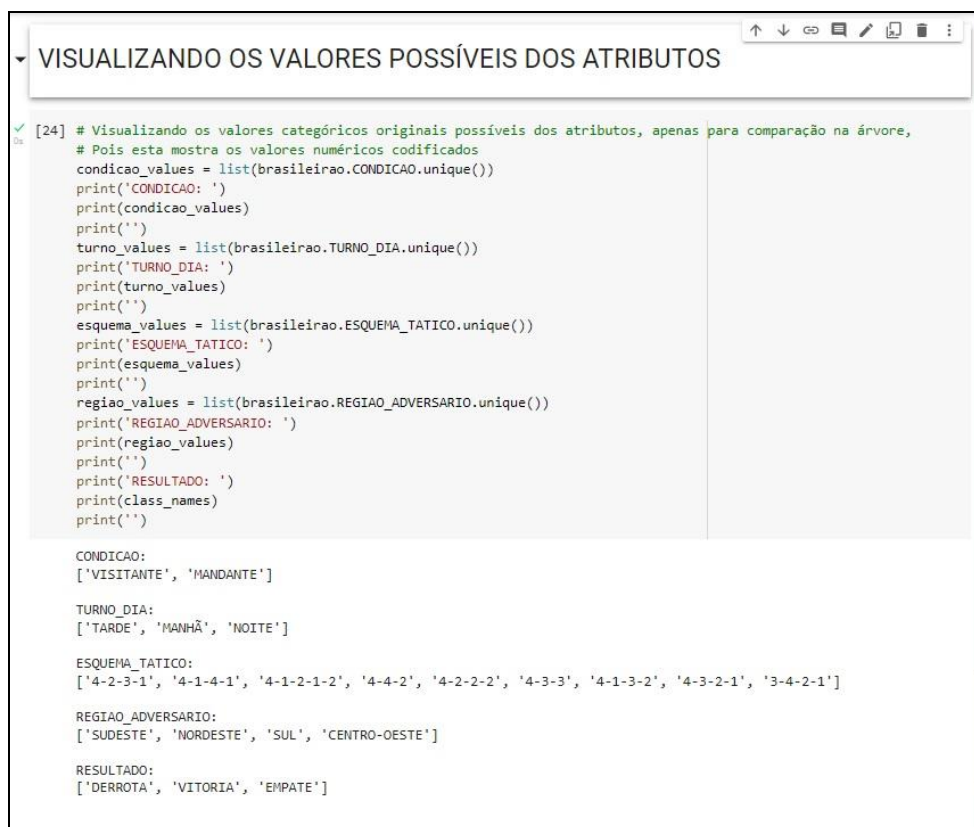
Depth = 100
Leaf = 100
Score de Treino é : 0.6318407960199005
Score de Teste é : 0.44776119402985076

```

FIG. 11 – Variação dos parâmetros da árvore para procura do melhor desempenho do Modelo

2.3.6. Exibição da Árvore de Decisão

Na próxima célula, mostrada na figura 12, apenas exibimos os possíveis valores originais de cada atributo (no *dataframe brasileiro*), para posteriormente facilitar a interpretação da nossa árvore de decisão, uma vez que na construção e no gráfico da árvore são trabalhados os valores numéricos codificados (no *dataframe brasileiro_enc*).



```

[24] # Visualizando os valores categóricos originais possíveis dos atributos, apenas para comparação na árvore,
# Pois esta mostra os valores numéricos codificados
condicao_values = list(brasileirao.CONDICAQ.unique())
print('CONDICAQ: ')
print(condicao_values)
print('')
turno_values = list(brasileirao.TURNO_DIA.unique())
print('TURNO_DIA: ')
print(turno_values)
print('')
esquema_values = list(brasileirao.ESQUEMA_TATICO.unique())
print('ESQUEMA_TATICO: ')
print(esquema_values)
print('')
regiao_values = list(brasileirao.REGIAQ_ADVERSARIO.unique())
print('REGIAQ_ADVERSARIO: ')
print(regiao_values)
print('')
print('RESULTADO: ')
print(class_names)
print('')

CONDICAQ:
['VISITANTE', 'MANDANTE']

TURNO_DIA:
['TARDE', 'MANHÃ', 'NOITE']

ESQUEMA_TATICO:
['4-2-3-1', '4-1-4-1', '4-1-2-1-2', '4-4-2', '4-2-2-2', '4-3-3', '4-1-3-2', '4-3-2-1', '3-4-2-1']

REGIAQ_ADVERSARIO:
['SUDESTE', 'NORDESTE', 'SUL', 'CENTRO-OESTE']

RESULTADO:
['DERROTA', 'VITORIA', 'EMPATE']

```

FIG. 12 – Visualização dos valores possíveis originais (categóricos) dos atributos

Finalmente, na célula mostrada na figura 13, executamos novamente todo o código de construção, treinamento e predição de dados do nosso modelo, utilizando os melhores valores observados durante os nossos estudos, para os parâmetros do algoritmo da árvore de decisão. Após isso, fazemos a chamada do método *plot_tree()*, da biblioteca *sklearn.tree*, para construção do gráfico da árvore de decisão.

O gráfico gerado é mostrado na figura 14. Podemos observar que os valores dos atributos de entrada (ESQUEMA_TATICO, REGIAQ_ADVERSARIO, TURNO_DIA e CONDICAQ), são testados em vários níveis, até se chegar aos nós-folha, que representam os valores da previsão do atributo alvo (RESULTADO) para os diversos caminhos de testes (ramos da árvore).

EXIBIÇÃO DA ÁRVORE DE DECISÃO

```
[25] # Exibição da Árvore de Decisão

from sklearn.tree import plot_tree

brasileirao_tree = DecisionTreeClassifier(max_depth=10, max_leaf_nodes=10, random_state=42)
brasileirao_tree.fit(x_train, y_train)
y_pred_train = brasileirao_tree.predict(x_train)
y_pred_test = brasileirao_tree.predict(x_test)

print('Score de Treino é:', accuracy_score(y_train, y_pred_train))
print('Score de Teste é:', accuracy_score(y_test, y_pred_test))

plt.figure(figsize=(20,20))

plot_tree(brasileirao_tree, feature_names = x.columns, rounded = True, filled=True, class_names = class_names)

Score de Treino é : 0.582089552238806
Score de Teste é : 0.5373134328358209
```

FIG. 13 – Código para Exibição da Árvore de Decisão

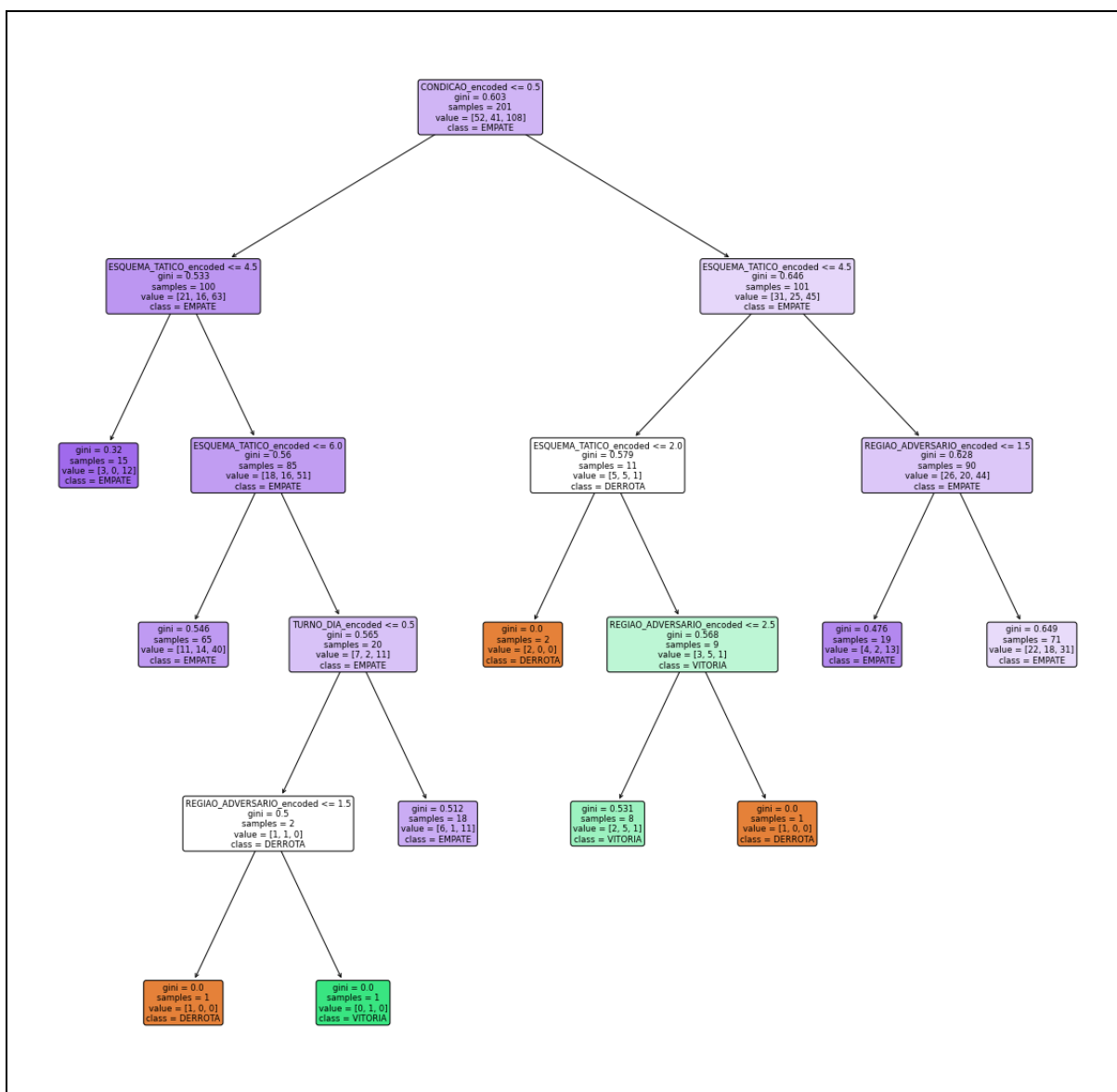


FIG. 14 – Gráfico da Árvore de Decisão

O arquivo completo do notebook Python descrito neste tópico, está disponível no repositório do projeto, em [Projeto Python – Módulo C](#). O mesmo pode ser executado tanto no ambiente Google Colab [<https://colab.research.google.com/>], utilizado neste trabalho, como em outras ferramentas disponíveis no mercado, como Jupyter Notebook [<https://jupyter.org/>] ou Visual Studio Code [<https://code.visualstudio.com/>], dentre outras.

3. Conclusões

3.1. Análise Visual - Dashboards

Ainda na análise visual, realizada através dos *dashboards* produzidos no [Módulo B do Relatório Técnico](#), já foi possível a combinação de filtros e a percepção nos gráficos disponibilizados, de algumas características dos dados, conforme descrito no tópico 2 do referido módulo, dentre as quais, podemos citar como exemplo:

- Média de gols por jogo do FLAMENGO, vem tendo crescimento substancial a partir do ano de 2017, coincidindo com as melhores classificações do clube nos campeonatos no mesmo período;
- Considerando o período a partir do ano de 2017, o número total de vitórias e pontos ganhos do FLAMENGO, é bastante superior aos números dos demais clubes;
- Considerando o período de 2003 a 2016, o FLAMENGO aparece apenas na 7ª posição em número de vitórias e na 8ª posição em número de pontos ganhos;
- Com os 3 achados citados anteriormente, evidencia-se as consequências positivas do trabalho de gestão realizado no clube FLAMENGO, cujos resultados começaram a se confirmar a partir do ano de 2017;
- Considerando somente o melhor período do clube FLAMENGO (2017 a 2021), podemos perceber que o clube, ao jogar como visitante, possui um maior número de vitórias e pontos ganhos nos estádios Arena Castelão, Mané Garrincha e Neo Química Arena, e por outro lado tem um maior número de derrotas nos estádios Arena da Baixada, Arena do Grêmio, Beira-Rio e Mineirão (mesmo assim, apenas 2

derrotas nesses estádios durante o período), o que sugere um maior cuidado/preparação do clube ao jogar contra os clubes mandantes desses estádios;

- Por outro lado, considerando esse mesmo período (2017 a 2021), o FLAMENGO, jogando como mandante possui uma maior dificuldade para jogar contra clubes do estado de São Paulo, para os quais obteve um maior número de derrotas nessa condição, o que também inspira um maior cuidado na preparação para os jogos contra esses clubes, mesmo na condição de mandante;

3.2. Análises Avançadas – Machine Learning

Por outro lado, neste Módulo C, apresentamos um modelo de análise com aplicação de técnicas de aprendizado de máquina (*machine learning*), mais especificamente classificação, através de árvore de decisão.

Conforme mostramos ao longo do item 2.3 deste trabalho, nosso modelo apresentado não atingiu uma ótima acurácia, ficando a mesma em torno de 54% no código final de montagem da árvore de decisão (figura 13), tendendo a prever muitos resultados de EMPATE, conforme pode-se perceber na matriz de confusão (figura 10). Dessa forma, não se consegue perceber no gráfico da árvore (figura 14) uma regra bem definida entre os valores dos atributos de entrada e o valor do atributo de saída.

Em bases comumente utilizadas para fins didáticos, como íris, sonar e titanic, por exemplo, utilizadas durante as aulas da disciplina *Machine Learning* [6.De Paula, Hugo] e em diversas outros textos sobre o assunto, sabemos que podemos obter uma ótima acurácia dos modelos de árvore de decisão. No entanto, em um conjunto de dados real, e não necessariamente preparado para fins didáticos, como o do DW Brasileiro, utilizado neste trabalho, valores ótimos de acurácia dos modelos podem não ser obtidos tão facilmente. Uma ação a ser tomada, em caso de resultado insatisfatório, seria realizar tentativas de se trabalhar sobre outros atributos de entrada, ou até mesmo, um outro conjunto de dados.

Também, em [7.Alves, Pedro], o autor cita que muitas vezes as árvores de decisão são usadas como modelos de comparação, ou seja, modelos que têm um desempenho aceitável na maioria dos casos e podem ser usados como *benchmark* para construção de outros modelos. O autor afirma ainda que é muito comum um

cientista de dados, começar com um modelo de árvore de decisão, entender o resultado que o algoritmo atingiu naquele problema, e aí sim começar a testar outros algoritmos para comparação de desempenho. No entanto, a árvore de decisão normalmente já estabelece um *baseline* de resultado para se usar como referência.

Por fim, podemos concluir que o trabalho realizado, nos seus 3 módulos, foi de grande valia para consolidação e demonstração do conhecimento adquirido ao longo do Curso de *Pós-Graduação em Analytics e Business Intelligence*. Considerando os três módulos do Relatório Técnico desenvolvido, foi possível aplicar diretamente o conhecimento adquirido em várias das principais disciplinas do curso, conforme referências bibliográficas listadas nos mesmos, e tendo desenvolvido atividades de todas as fases de um real projeto de BI e Analytics.

4. Links

Os artefatos produzidos no desenvolvimento deste trabalho, bem como os *Datasets* e demais arquivos relevantes, estão disponíveis para acesso no repositório do projeto, no link [Relatório Técnico – Módulo C](#).

Um vídeo de apresentação do funcionamento do nosso modelo de análise, em código Python, no ambiente Google Colab, está disponível para download no repositório do projeto, no link [Vídeo de Apresentação - Módulo C](#).

REFERÊNCIAS

[1.Sarkar, Dipanjan] DIPANJAN, Sarkar; BALI, Raghav; SHARMA, Tushar. **Practical Machine Learning with Python**. Bangalore, Karnataka, India: Apress, 2018.

[2.Guido, Sarah] GUIDO, Sarah; MÜLLER, Andreas. **Introduction to Machine Learning with Python**. Sebastopol, CA, USA: O'Reilly, 2017.

[3.Faceli, Katti] FACELI, Katti; LORENA, Ana Carolina; GAMA, João; CARVALHO, André Carlos. **Inteligência Artificial, Uma Abordagem de Aprendizado de Máquina**. Rio de Janeiro : LTC, 2011.

[4.Gomes, Rodrigo] GOMES, Rodrigo. **Notas de aula da Disciplina Introdução à Linguagem Python, do Curso de Pós-Graduação em Analytics e Business Intelligence da PUC Minas**. Belo Horizonte, 2021.

[5.Gomes, Rodrigo] GOMES, Rodrigo. **Notas de aula da Disciplina Programação para Ciência de Dados, do Curso de Pós-Graduação em Analytics e Business Intelligence da PUC Minas**. Belo Horizonte, 2021.

[6.De Paula, Hugo] DE PAULA, Hugo. **Notas de aula da Disciplina Machine Learning, do Curso de Pós-Graduação em Analytics e Business Intelligence da PUC Minas**. Belo Horizonte, 2021.

[7.Alves, Pedro] ALVES, Pedro. **Apostila do Curso Python para Data Science e Analytics – Módulo Avançado, Aula 3 – Árvore de Decisão** [<https://python-para-datascience-analytics.club.hotmart.com/>]. PA Analytics [<https://paanalytics.net/>], 2022.