UNIVERSIDADE FEDERAL DE MATO GROSSO

CAMPUS DE VÁRZEA GRANDE

Projeto de Inteligência Artificial

	Nome	RGA
1	Eduardo Oliveira Silva	201921901002
4	Matheo Rodrigues Bonucia	201921901015

Disciplina: INTELIGÊNCIA ARTIFICIAL

Prof. Frederico Santos de Oliveira

Cuiabá, junho de 2023

Modelo Preditivo Brasileirão

Projeto de Inteligência Artificial

Documentação do Projeto de Modelo Preditivo apresentado na disciplina de Inteligência Artificial como parte da avaliação do aprendizado na Universidade Federal de Mato Grosso na Faculdade Engenharia.

Cuiabá, junho de 2023

Sumário

Lista de Figuras

Listings

1 Introdução

Esse projeto tem como objetivo principal aplicar conceitos de inteligência artificial aprendidos durante o estudo da disciplina. Optamos por utilizar conceitos de machine learning dentre os vários aprendidos, isso porque, queriamos desenvolver um modelo preditivo de partidas esportivas. Sendo assim, decidimos criar um modelo de previsão especificamente para o Campeonato Brasileiro Série A (Brasileirão), pois moramos no Brasil e acompanhamos o esporte futebol.

Para alcançar nosso objetivo, utilizamos da plataforma *Kaggle*, onde a comunidade disponibiliza bases de dados para estudo de análise de dados, machine learning, ciência de dados e engenharia de dados. Inicialmente, desenvolvemos o modelo com base em um conjunto contendo apenas informações de classificação final, como posição, nome do time, quantidade total de gols feitos, quantidade total de gols sofridos e etc.

Adiante, utilizamos outro conjunto, com informações de cada partida. A fim de melhorar o resultado do modelo, combinamos esses dois conjuntos de dados para obter resultados mais precisos, analisando a chance de um time vencer em relação ao adversário e considerando também o histórico de títulos do time.

É importante ressaltar a dificuldade de prever resultados de partidas no Campeonato Brasileiro Série A, pois o empate é permitido em todas as partidas até o final do campeonato, e o time com mais pontos, considerando vitórias e empates, é classificado como vencedor. Sendo assim, em campeonatos com sistema eliminatório, onde só um dos times pode vencer, o modelo se sairia melhor.

Contudo, iremos apresentar um resultado interessante que demonstrará a possibilidade de criar um modelo de previsão de vitórias em um campeonato como o Brasileirão.

2 Descrição do Problema

Inicialmente, buscamos conjuntos de dados relevantes que pudessem colaborar para o desenvolvimento do projeto e encontramos uma base de dados contendo informações de classificação, incluindo a posição final de cada time no campeonato.

No entanto, essa tabela abrangia o período de 2003 à 2019. Para garantir a atualização dos dados corretamento, utilizamos o site da Confederação Brasileira de Futebol para obter estatísticas atualizadas até o final de 2022. Dessa forma, garantimos a inclusão de dados atualizados na nosso modelo.

Figura 1: Tabela de Ranqueamento

	year	position	team	points	games	victories	draws	losses	goals_scored	goals_against	goals_difference	perc_points_won
0	2003	1	Cruzeiro	100	46	31	7	8	102	47	55	72
1	2003	2	Santos	87	46	25	12	9	93	60	33	63
2	2003	3	Sao Paulo	78	46	22	12	12	81	67	14	56
3	2003	4	São Caetano	74	46	19	14	13	53	37	16	53
4	2003	5	Coritiba	73	46	21	10	15	67	58	9	52
...	...	...	...	...	...	...	...	...	...	...	...	...
405	2022	16	Cuiaba	41	38	10	11	17	31	42	-11	35
406	2022	17	Ceara	37	38	7	16	15	34	41	-7	32
407	2022	18	Atletico-GO	36	38	8	12	18	39	57	-18	31
408	2022	19	Avai	35	38	9	8	21	34	60	-26	30
409	2022	20	Juventude	22	38	3	13	22	29	69	-40	19

410 rows × 12 columns

Além dessa base de dados, utilizamos outra fonte de dados estatísticos de cada partida. Essa base de dados incluía detalhes sobre o time mandante, o time visitante, quem ganhou a partida, os técnicos envolvidos e etc. A informação sobre o time mandante e visitante é especialmente interessante, pois no futebol existe uma pressão maior sobre o time vencer quando joga em casa, ou seja, no seu próprio estádio. Isso leva os jogadores a se dedicarem ainda mais para representar a torcida, resultando às vezes em vitória.

Figura 2: Tabela estatística de cada partida

	ID	rodata	data	hora	mandante	visitante	formacao_mandante	formacao_visitante	tecnico_mandante	tecnico_visitante	vencedor
4729	4741	13	2014-08-03	16:00	Chapecoense	Flamengo	4-4-2	4-2-3-1	C. Rodrigues	V. Luxemburgo da Silva	Chapecoense
4797	4808	20	2014-09-10	19:30	Palmeiras	Criciuma	4-2-2-2	4-2-3-1	D. Silvestre Júnior	G. Dal Pozzo	Palmeiras
4822	4833	22	2014-09-17	22:00	Coritiba	Sao Paulo	4-3-2-1	4-4-2	M. dos Santos Gonçalves	M. Ramalho	Coritiba
4835	4846	23	2014-09-21	18:30	Gremio	Chapecoense	4-1-4-1	4-2-3-1	L. Scolari	J. da Silva	Gremio
4842	4853	24	2014-09-24	22:00	Fluminense	Gremio	4-2-3-1	4-1-4-1	C. Borges dos Santos	L. Scolari	-
...	...	...	...	...	...	...	...	...	...	...	...
8020	8021	38	2022-11-13	16:03	Cuiaba	Coritiba	4-1-4-1	4-1-4-1	A. Cardoso de Oliveira	A. Ferreira	Cuiaba
8021	8022	38	2022-11-13	16:03	Bragantino	Fluminense	4-2-3-1	4-2-3-1	M. Nogueira Barbieri	F. Diniz Silva	Fluminense
8022	8023	38	2022-11-13	16:03	Corinthians	Atletico-MG	4-1-4-1	4-2-3-1	F. J. Monteiro Almeida	A. Stival	Atletico-MG

Inicialmente, iremos desenvolver um modelo de previsão usando os dados da figura 1 e analisar seu desempenho. É interessante para observarmos como o modelo se comportará em relação a um base de dados com um número limitado de características.

Posteriormente, iremos criar um modelo combinando as informações das duas bases de dados, a fim de obter um resultando melhorado. Acreditamos que unir esses dados estatísticos de cada partida, com os resultados referente ao ranqueamento, será possível obter previsões mais precisas.

3 Propostas de Soluções

Para resolver o problema que foi proposto, utilizaremos a biblioteca *Scikit-Learn* que foi desenvolvida especificamente para implementar conceitos de aprendizado de máquina, também utilizaremos um método da biblioteca *XGBoost*.

3.1 Árvore de Decisão

Um dos modelos que empregaremos para previsão é o algoritmo de árvore de decisão. Este método de aprendizagem é adequado para tarefas de classificação e regressão. Pois, permite que possamos construir um modelo que prevê o valor de uma variável com base em regras de decisão simples aplicadas às características do conjunto de dados.

3.2 k-Vizinho mais próximo

Outro modelo que utilizaremos, é o k Nearest Neighbor (k-NN). Este método de aprendizado usa a distância euclidiana entre os pontos de dados para identificar os vizinhos mais próximos do ponto de dados que está sendo analisado. A classificação de um ponto de dados é determinada com base nos rótulos de classe de seus vizinhos mais próximos. Neste método, definimos o número de vizinhos a serem considerados, e esta escolha impacta nos resultados finais de previsão. A equação do cálculo da distância euclidiana é:

$$d = \sqrt[2]{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{1}$$

3.3 Floresta Aleatória

Ademais, utilizaremos um método de aprendizado baseado no algoritmo Random Forest. Esse método aproveita da combinação de várias árvores de decisão para classificação. O algoritmo Random Forest consiste em várias árvores , onde cada árvore do conjunto é treinada usando um subconjunto diferente a partir dos dados de treinamento. Essa diversidade permite que cada ávore capture diferentes aspectos do conjuntos de dados, levando à um melhor desempenho preditivo.

3.4 XGBoost

Outro método utilizado, é o XGBoost, que trata-se de um algoritmo de aprendizado que aproveita da técnica de boosting para contruir modelos eficientes. O boosting é um processo que combina vários modelos para criar um modelo mais preciso e confiável. O XGBoost aplica o gradiente descendente, que é um método para otimizar a perda. Calculando o grandiante de

perda com base em previsões anteriores e ajustando-os para minimizar a perda. Esse processo é realizado a partir do treinamento de um série de ávores de decisão, refinando-as iterativamente. No final o resultado é um conjunto de previsões de todas as ávores treinadas, resultando em um modelo aprimorado.

3.5 SVC

Por fim, o último método aplicado foi o SVC (Support Vector Classifier). Esse tipo de método utiliza de funções de kernel parra transformar dados de entrada em um espaço de dimensão superior. Essa transformação permite que o modelo identifique um hiperplano que de maneira efetiva, realize a separação de diferentes classes. Ao instanciar esse modelo, precisamos definir qual tipo de função de kernel, podendo ser: linear, polinomial e radial (RBF). No nosso caso, decidimos que seria melhor utilizar o radial, pois é adequado para conjuntos de dados não lineares. Ao utilizar o kernel RBF, nosso modelo pode tomar decisões complexas com as quais um kernel linear teria dificuldades.

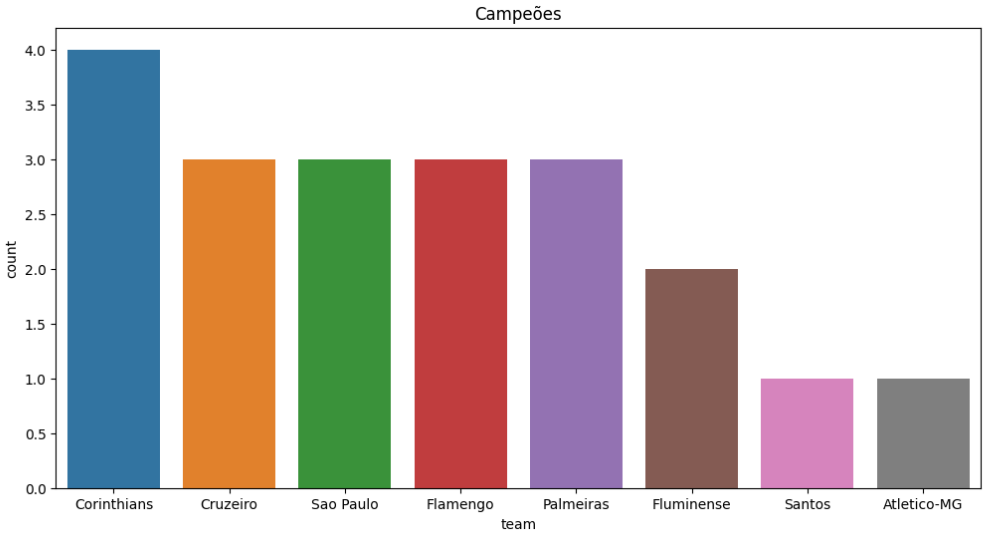
4 Resultados

Para começarmos nosso modelos, foi necessário realizar uma padronização das bases de dados, isso porque, a forma de escrever os nomes dos times estavam diferentes em cada uma.

4.1 Primeiro Modelo (Dados de ranking)

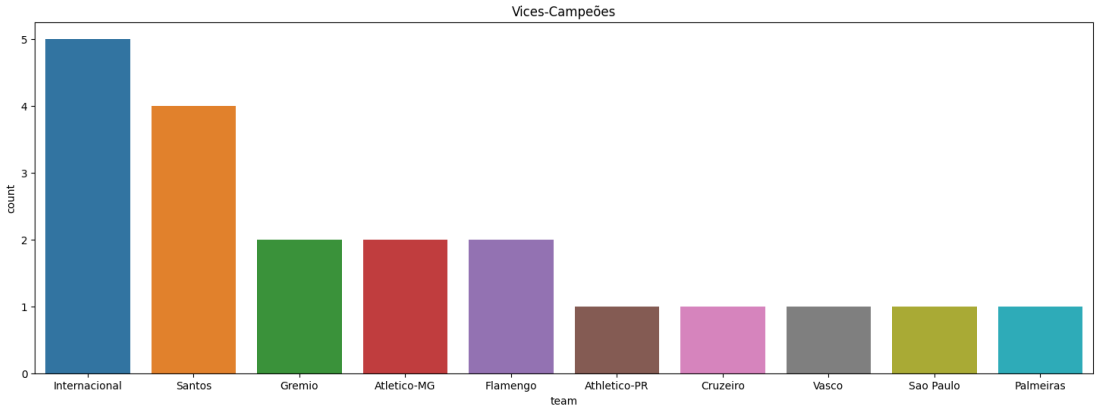
Antes de começarmos a projeção do modelo, fizemos um análise dos dados. Então separamos em duas tabelas os times que foram vencedores, e em outra os times que ficaram em vice.

Figura 3: Times campeões



A partir da figura 3 podemos observar quais foram os times que mais obtiveram títulos no campeonato brasileiro. Sendo assim, esses são os principais candidatos para times mais fortes.

Figura 4: Times vice-campeoões



Os vice-campeões representam fortes candidatos a melhores times do campeonato, então gráfico da figura 4 é essencial.

Dando continuidade, como estamos tratando de um modelo de machine learning é interessante realizarmos a transformação dos dados em String para dados do tipo númerico, isso nos proporciona um melhor tempo de desempenho.

```
equipes = {} # dicionario que cria um numero (index) para cada nome de time
index = 0

for id, row in ranking.iterrows():
  name = row['team']
  if(name not in equipes.keys()):
    equipes[name] = index
    index += 1
def alterar_nome_id(df): # funcao para alterar o nome do time pelo index
   gerado
  df['team'] = equipes[df['team']]
  return df

ranking_id = ranking.apply(alterar_nome_id, axis='columns')
```

Listing 1: Código para alterar o nome em um ID

Com isso, criamos um ID único para cada um dos times que estamos analisando e aplicamos em nossa base de dados. Além disso, é essencial deletarmos as características que não são importantes para a modelagem. Dessa forma, retiramos: ano, diferença de gols e round. Também precisamos retirar a coluna posição, pois a posição é o valor que queremos que nosso modelo realize a previsão.

Após a limpeza, a nossa tabela de análise, ficará assim:

Figura 5: Tabela de ranqueamento limpa

	team	points	victories	draws	losses	goals_scored	goals_against	perc_points_won
0	0	100	31	7	8	102	47	72
1	1	87	25	12	9	93	60	63
2	2	78	22	12	12	81	67	56
3	3	74	19	14	13	53	37	53
4	4	73	21	10	15	67	58	52
...	...	...	...	...	...	...	...	...
405	44	41	10	11	17	31	42	35
406	36	37	7	16	15	34	41	32
407	37	36	8	12	18	39	57	31
408	33	35	9	8	21	34	60	30
409	17	22	3	13	22	29	69	19

410 rows × 8 columns

Adiante, realizamos a separação dos nossos dados, dados de treino e dados de teste. Para isso, utilizamos da biblioteca da *SKlearn train_test_split*. Nesse função, passamos como parâmetros o tamanho da nossa base teste e definimos um valor de aleatoriedade. Utilizamos como seed de aleatoridade 47 e dividimos a base em 80% treino e 20% teste.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=42) # separando o modelo em treino e teste

# carregando os modelos
decision_tree_model = DecisionTreeClassifier()
knn_model = KNeighborsClassifier(n_neighbors = 1)
random_forest_model = RandomForestClassifier()

# treinando os modelos
decision_tree_model.fit(X_train, y_train)
knn_model.fit(X_train, y_train)
random_forest_model.fit(X_train, y_train)

# salvando a predicao de cada modelo
y_pred_tree = decision_tree_model.predict(X_test)
y_pred_knn = knn_model.predict(X_test)
y_pred_forest = random_forest_model.predict(X_test)
```

Listing 2: Separação dos dados e aplicação dos métodos

Para exibir os resultados, achei interessante utilizar a biblioteca *classification_report*, pois com ela é possível analisar a precisão, o recall e o f1-score que o modelo atingiu a partir de suas previsões.

A precisão mede a proporção de exemplos positivos corretamente classificados em relação ao número total de exemplos classificados. Sua equação pode ser definida como:

$$P = \frac{TP}{TP + FP} \tag{2}$$

Enquanto o recall, é uma medida da proporção de exemplos positivos corretamente classificados em relação ao número total de exemplos que são realmente positivos. Sua equação pode ser definida como:

$$r = \frac{TP}{TP + FN} \tag{3}$$

Já o f1-score, é a média harmônica de precisão e recall. Essa medida nos fornece uma única métrica que equilibra a precisão e o recall. Então esse valor é o mais importante para a gente.

Os resultados obtidos foram:

- Método Árvore de Decisão: Precisão = 0.36. Recall = 0.33. f1-score: 0.32
- Método kNN: Precisão = 0.18. Recall = 0.20. f1-score: 0.17
- Método Random Forest: Precisão = 0.32. Recall = 0.29. f1-score: 0.28

Como podemos observar, esse nosso modelo apresentou o melhor resultado de exatidão de 32% a partir do modelo de árvore de decisão. Entretanto, esse valor ainda é muito baixo, signifcando que não é possível dizer qual o vencedor do campeonato com base na pontuação e na quantidade de gols.

4.2 Segundo Modelo (Dados de Partidas)

Dessa forma, iremos desenvolver o segundo modelo, utilizando a base de dados com as estatísticas de cada partida e utlizar da base de ranqueamento.

Conforme feito anteriormente, precisamos alterar os nomes por números, para melhorar o desempenho, entretanto, essa base de partidas possui nome do time mandante e do time visitante. Sendo assim, criaremos uma função que define um ID exclusivo para cada time verificando se esse time já não recebeu um ID anteriormente no laço de repetição.

```
# funcao pra criar um index pro nome do time
def index_nome_team(df):
  equipes = {}
  index = 0

  for idx, row in df.iterrows():
    name = row['mandante']
    if(name not in equipes.keys()):
      equipes[name] = index
      index += 1
    name = row['visitante']
    if(name not in equipes.keys()):
      equipes[name] = index
      index += 1
  return equipes
equipes = index_nome_team(matches)
matches_id = matches_clear.apply(alterar_nome_index, axis='columns')
```

Listing 3: ID exclusivo time mandante e time visitante

Continuando, agora iremos realizar a limpeza da nossa base, retirando dados que iriam atrapalhar o desenvolvimento do nosso modelo. Sendo assim, foi retirado dados de: data, hora,

arena, formação mandante, formação visitante, tecnico mandante, tecnico visitante, mandante estado, visitante estado e rodada.

Agora iremos vincular os dados das duas bases. Iremos pegar o nome do time da tabela de partidas e verificar quantas vezes ele foi campeão na tabela de ranqueamento. Para isso, criamos duas novas colunas para nossa base, uma para títulos do mandante e outra para títulos do visitante.

```
# limpando base
matches_clear = matches.drop(['ID','data','hora','arena',
                              'formacao_mandante', 'formacao_visitante',
                              'tecnico_mandante','tecnico_visitante',
                              'mandante_Estado', 'visitante_Estado','rodata'
    ], axis = 1)

# adicionando colunas de vitoria do mandante de visitante
matches_clear['mandante_vitorias'] = 0
matches_clear['visitante_vitorias'] = 0

# contador de vitorias (titulo do campeonato), verifica na tabela de ranking
    quantos anos o time ficou no primeiro lugar
def count_vitorias(df):
  if(campeoes.get(df['mandante'])!= None ):
    df['mandante_vitorias'] = campeoes.get(df['mandante'])
  if(campeoes.get(df['visitante'])!= None ):
    df['visitante_vitorias'] = campeoes.get(df['visitante'])
  return df
matches_clear = matches_clear.apply(count_vitorias, axis='columns')
```

Listing 4: Definindo a quantidade de títulos para cada time

Ademais, precisamos alterar a coluna de 'vencedor', pois ela traz o nome do time vencedor, é interessante que essa coluna seja númerica, onde 1 represente caso o time mandante for vencedor, 2 caso o time visitante seja o vencedor, e em caso de empate 0.

```python
# altera o nome pelo index
def alterar_nome_index(df):
  # se o mandante venceu = 1
  # se o visitante venceu = 2
  # se empate = 0
  if(df['vencedor'] == df['mandante']):
      df['vencedor'] = 1
  elif(df['vencedor'] == df['visitante']):
      df['vencedor'] = 2
  else:
      df['vencedor'] = 0

  # troca o nome do time pelo indice
  df['mandante'] = equipes[df['mandante']]
  df['visitante'] = equipes[df['visitante']]

  return df
matches_id = matches_clear.apply(alterar_nome_index, axis='columns')
```

Listing 5: Definindo um valor para o vecendor

Após realizar a padronização dos dados e a limpeza da base, obtemos a seguinte tabela:

Figura 6: Tabela de partidas limpa

	mandante	visitante	vencedor	mandante_Placar	visitante_Placar	mandante_vitorias	visitante_vitorias
4729	0	1	1	1	0	0	3
4797	2	3	1	1	0	3	0
4822	4	5	1	3	1	0	3
4835	6	0	1	1	0	0	0
4842	7	6	0	0	0	2	0
...	...	...	...	...	...	...	...
8020	32	4	1	2	1	0	0
8021	31	7	2	0	1	0	2
8022	14	8	2	0	1	4	1
8023	10	2	1	3	0	0	3
8024	12	5	2	0	4	0	3

3048 rows × 7 columns

Dessa vez, para obter o melhor desempenho possível, resolvi utilizar uma tática *flipping*. Essa tática consiste em duplicar sua base de dados e inverter as caractéristicas e os resultados. Isso será útil pois, como a distribuição dos rótulos que estamos tratando é desbalanceada, será interessante utilizar esse método porque nosso conjunto de dados não é tão grande.

```
# convertendo o dataframe em um vetor array do tipo float para aplicar
    funcoes da lib NumPy
X = np.array(X).astype('float64')

_X = X.copy() # copia dos dados
# inverte a primeira e a segunda coluna
_X[:,0] = X[:,1]
_X[:,1] = X[:,0]
# inverte a terceira e a quarta coluna
_X[:,2] = X[:,3]
_X[:,3] = X[:,2]

# cria um copia de y
_y = y.copy()

# inverto a resposta da copia do y
for i in range(len(_y)):
  if(_y[i] == 1):
    _y[i] = 2
  elif(_y[i] == 2):
    _y[i] = 1

    # junto o vetor de dados original + o vetor de dados invertidos
X = np.concatenate((X,_X), axis=0)
y = np.concatenate((y,_y))
```

Listing 6: Método Flipping de dados

Iremos utilizar a biblioteca do *SKlearn Shuffle*. Sua vantagem é garantir uma aleatorização no nosso conjunto de amostra.

```
# embaralho os dados
X,y = shuffle(X,y)

# separo em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.20)
```

Listing 7: Embaralhando os dados e separando treino e teste

Agora iremos instanciar os modelos, realizar os treinos e em sequência os testes.

```
# instancio o modelo SVC
model_svc = SVC(kernel='rbf', class_weight='balanced',probability=True)
# treino o modelo SVC
model_svc.fit(X_train,y_train)

# instancio o modelo XGB
classifier = XGBClassifier()
# treino o modelo XGB
classifier.fit(X_train, y_train)
```

Listing 8: Aplicação dos métodos

Além desses modelos do bloco de código anterior, utilizaremos os outros que usamos no modelo anterior a fim de comparação. Os resultados foram:

Tabela 1: Resultados do segundo modelo

Método	**Precisão**	**Recall**	**F1-score**
SVC	0.42	0.43	0.43
kNN	0.37	0.37	0.37
Ávore de Decisão	0.36	0.36	0.36
XBGClassifier	0.39	0.40	0.40

Como podemos observar o método que apresentou a melhor precisão foi o SVC. Então daremos continuidade com ele. Com o modelo pronto, agora é possível criarmos um função onde poderemos fornecer parâmetros de entrada e o modelo realizar a previsão dos parâmetros preenchidos.

Para isso, utilizaremos uma função do modelo SVC chamada *predict_proba* essa função recebe os parâmetros entrada do modelo e nos retorna a probalidade de cada classificação.

```
def previsao_times(mandante, visitante):
  # pega o id do time de entrada
  id_mandante = equipes[mandante]
  id_visitante = equipes[visitante]

  # verifica se os times de entrada foram campeoes alguma vez, senao retorna 0
  campeao_mandante = campeoes.get(mandante) if campeoes.get(mandante) != None else 0
  campeao_visitante = campeoes.get(visitante) if campeoes.get(visitante) != None else 0

  # crio um array de dados com os dados necessarios para passar pro modelo. Dados do tipo float
  x = np.array([id_mandante, id_visitante, campeao_mandante, campeao_visitante]).astype('float64')
  # redimensiono o array x para transformalo em uma matriz unidimensional com uma linha
  # queremos saber a previsao para sobre um dado
  x = np.reshape(x, (1, -1))

  # a funcao predict_proba retorna a probabilidade para cada uma das classes
  # o [0] quer dizer somente a primeira linha da matriz de entrada x.
  y = model_svc.predict_proba(x)[0] # aplica o modelo na primeira linha da matriz x e retorn a probabilidade para a varial y
  # y[0] = probabilidade de dar empate pois 0 foi classificado como empate
  # y[1] = probabilidade do time mandante ganhar
  # y[2] = probabilidade do time visitante ganhar

  # texto retornando o resultado
  resultado = (mandante+ ' {}\n' +visitante+ ' {}\nEmpate {}').format(round(y[1]*100,2), round(y[2]*100,2), round(y[0]*100,2))
  return resultado
```

Listing 9: Função que retorna a probabilidade de um time ganhar uma partida

Com essa função em mãos é possível testar o modelo na prática e observar os resultados. Sendo assim, fui na base de dados da CBF e peguei uma amostra aleatória, 10 partidas da 6 rodada do Brasileirão do ano de 2023. O resultado foi:

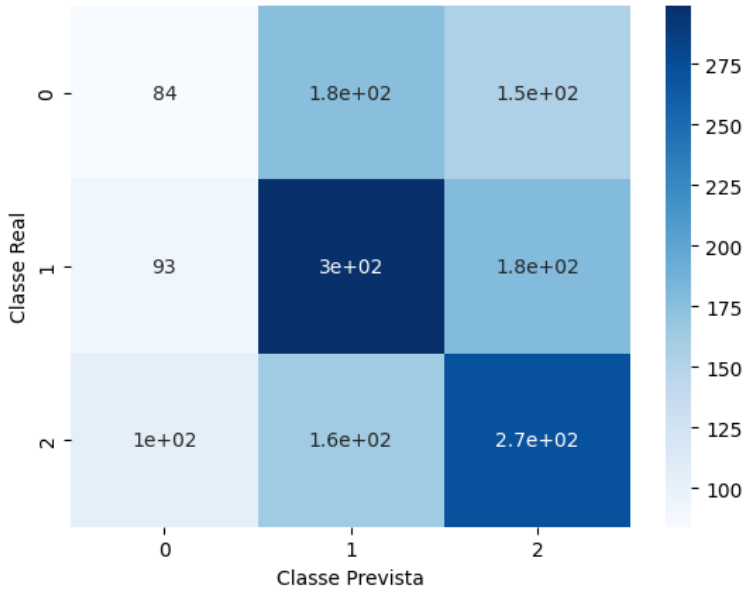
Tabela 2: Amostragem de partidas reais aplicadas ao modelo preditivo

Time mandante	**Time Visitante**	**Previsão**	**Resultado**	**Modelo acertou?**
Bahia	Flamengo	Flamengo	Flamengo	Sim
Fluminense	Cuiabá	Fluminense	Fluminense	Sim
Palmeiras	Bragantino	Palmeiras	Empate	Não
Atlético-MG	Internacional	Atlético-MG	Internacional	Sim
Grêmio	Fortaleza	Grêmio	Empate	Não
Vasco	Santos	Santos	Santos	Sim
Corinthians	São Paulo	Empate	Empate	Sim
Athletico-PR	Coritiba	Coritiba	Athletico-PR	Não
Goias	Botafogo	Goias	Goias	Sim
América-MG	Cruzeiro	Cruzeiro	Cruzeiro	Sim

Como podemos observar em uma amostra aleatória prática, o modelo obteve um resultado de 70% de assertividade. O que é um resultado muito bom. Entretanto, isso é uma pequena amostra, o verdadeiro resultado foi de 43% a partir da base teste. O objetivo era que ficasse acima de 50%, mas não foi possível, e isso se dá pelo resultado de empate, com certeza se não houvesse empate o resultado seria muito superior.

Essa relação do empate interferir muito na analise pode ser observada a partir da matriz de confusão.

Figura 7: Matriz de confusão



Como podemos observar, a previsão de empate que realmente resultou em empate é muito baixa, pouquissímos casos de acertos. Além disso, é possível verificar que há um número de casos relativamente grande de previsão de do time mandante ou time visitante vencedor e acabar resultando em empate.

5 Conclusão

Podemos concluir que o projeto foi concluído com êxito, mesmo que não tenha sido possível obter um modelo com mais de 50% de assertividade. Acredito que conseguimos um bom resultado no modelo proposto. Inicialmente, partimos da ideia de montar um modelo de previsão a partir do ranking e pontuação, ou seja, queríamos ver qual seria a capacidade do nosso modelo prever a posição do time no ranking a partir da quantidade de gols e pontos feitos durante o campeonato. Entretanto, o resultado não foi satisfatório.

Acredito que a junção das duas bases de dados para propor um modelo melhor foi uma ótima ideia. No segundo modelo, conseguimos obter uma certa taxa de assertividade e a função que criamos para utilizar o modelo para um partida específica, definida por parâmetro, foi uma boa ideia, pois permitiu que testassemos o modelo com base em partidas reais do campeonato que está acontecendo esse ano. Esse segundo modelo foi melhor, mas ele tem uma proposta diferente, sua ideia é definir o vencedor de uma partida a partir de confrontos anteriores e a baseado no ranqueamento que o time ficou nos últimos campeonatos.

Uma ideia para aprimorar o projeto seria atualizar a base de dados para incluir o técnico mandante e técnico visitante como catacterísticas de análise para o modelo. Atualmente, essa base de dados possui apenas algumas linhas com essa informação e acredito que esse dado do técnico melhoraria o modelo de forma signficativa, já que no campeonato brasileiro há alguns técnicos que, independente do time, garantem bons resultados.

Por fim, esse projeto foi de extrema importância para nossa carreira profissional. Acabamos nos interessando muito pelo estudo de machine learning. Pretendemos aprimorar esse projeto futuramento para tentar obter resultados melhores.

6 Referências Bibliográficas

Referências

[1] Scikit Learn. *Decision Trees*. Disponível em: <https://scikit-learn.org/stable/modules/tree.html >. Acesso em: 05 de junho de 2023.

[2] XGBoost. *XGBoost Documentation*. Disponível em: <xgboost.readthedocs.io/en/stable/index.html >. Acesso em: 05 de junho de 2023.

[3] Bekhruz Tuychiev. *Using XGBoost in Python Tutorial*. Disponível em: <https://www.datacamp.com/tutorial/xgboost-in-python >. Acesso em: 05 de junho de 2023.

[4] Scikit Learn. *RandomForestClassifier*. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html >. Acesso em: 05 de junho de 2023.

[5] Kylie Ying. *Machine Learning for Everybody*. Disponível em: <https://youtu.be/i_LwzRVP7bg >. Acesso em: 05 de junho de 2023.

[6] Kylie Ying. *Machine Learning for Everybody*. Disponível em: <https://youtu.be/i_LwzRVP7bg >. Acesso em: 05 de junho de 2023.

[7] Adão Duque. *Campeonato Brasileiro de futebol*. Disponível em: <https://www.kaggle.com/datasets/adaoduque/campeonato-brasileiro-de-futebol >. Acesso em: 05 de junho de 2023.

[8] Jose Michelin. *Brazilian Football Championship*. Disponível em: <https://www.kaggle.com/datasets/josevitormichelin/brazilian-football-championship-brasileiro >. Acesso em: 05 de junho de 2023.