Predição de resultado de um jogo de xadrez, na plataforma *Lichess*

Gustavo R. Wanke[1], **Gabriel A. Stipp** [1]

[1]Departamento de Informática – Universidade Estadual de Maringá (Uem)
Maringá – PR – Brazil

ra91671uem.br, ra102919@uem.com

Resumo. *Este trabalho descreve um dataset utilizando-se de análise descritiva de atributos do mesmo, bem como implementa três técnicas de AM(aprendizado de máquina) distintas para tentar prever o vencedor de uma partida ou um empate, na plataforma Lichess.*

1. Introdução

1.1. O jogo

O xadrez é um dos jogos mais antigos da humanidade. Possui como objeto a captura de peças inimigas. Atualmente(depois de algumas variações em peças, tabuleiros, movimentos das peças) um jogador ganha quando captura a peça do Rei do adversário.

Não se sabe ao certo a origem do xadrez; são várias as referências desse jogo milenar. A evidência mais antiga vem do século VII, ao norte da Índia; o jogo era conhecido como Chaturanga, e poderia ser jogado com até quatro oponentes; os exércitos se enfrentavam no tabuleiro, que era composto por quatro grupos de oito peças: Rei (Rajá), Elefante, Cavalo e Barco (ou Carruagem), além da Infantaria. No Brasil, o jogo existe desde 1808, quando D. João VI ofereceu à Biblioteca Nacional, no Rio de Janeiro; atualmente é um jogo bastante praticado no Brasil, inclusive nas escolas.

De acordo com [Castro 1994] , "o xadrez é de extrema complexidade. Jogado num tabuleiro de 64 casas, cada jogador tem inicialmente 32 peças de seis tipos, cada qual com importância, movimentos e possibilidades de captura específicos. Apenas os quatro primeiros lances podem produzir cerca de 72 mil diferentes posições. Os dez primeiros lances podem ser jogados de cerca de 170 seguido de 27 zeros maneiras diferentes. Trata-se, portanto, de um jogo de possibilidades inesgotáveis".

1.2. Técnicas utilizadas

1.2.1. *Support Vector Machine*(SVM):

O *SVM* é um algoritmo de aprendizado de máquina supervisionada, que comumente é utilizado para problemas de classificação ou regressão.

Basicamente, o algoritmo procura encontrar hiperplanos para a separação de dados. Internamente, para a classificação dos diferentes hiperplanos, o algoritmo faz o uso de vetores de suporte, que são os pontos com menor distância entre o hiperplano. Após isto, o melhor hiperplano é um que maximize a distância(ou melhor, a margem) entre o hiperplano e os vetores de suporte. Em outra palavras, é o hiperplano que melhor se

distancie dos vetores de suporte. Em sua forma mais básica, o *SVM* é um modelo para classificação de um conjunto de dados linearmente separáveis, apenas. Entretanto, com a aplicação de diferentes *kernels*, o *SVM* pode realizar tal tarefa.

Parâmetros do SVM:

- C:É o tamanho da margem. O padrão é 1.0. Valores maiores significam um hiperplano de margem menor. Por outro lado, um valor baixo de C faz com que a margem seja grande.
- Kernel: Basicamente é um mapeamento utilizando funções matemáticas(os próprios *kernels*) de um conjunto de dados em outro, alterando assim a curvatura do hiperplano. Os mais comuns são lineares, *sigmoids*, *RBFs*(*Radial Basis Function*) e polinomiais
- Gamma: Define quão longe a influência de um ponto de treinamento chega. Valores altos de gamma intuitivamente significam que a influência não chega muito longe. Por outro lado, valores pequenos, significam o inverso. É usando quando o *kernel* é RBF , polinomial ou *sigmoid*.
- Degree:O Grau do hiperplano, quando o *kernel* é polinomial.

1.2.2. *K-Nearest Neighbor*:

O algoritmo *K-Nearest Neighbors* (KNN) é um algoritmo simples de aprendizado de máquina supervisionado que pode ser usado para resolver problemas de classificação e regressão. O KNN é relativamente fácil de implementar, porém sua velocidade de execução diminui significativamente conforme o tamanho do banco de dados utilizado cresce.

Este algoritmo encontra as distancias entre o objeto a ser analisado e os exemplos presentes no banco de dados disponível, selecionando os *k* elementos mais próximos ao objeto analisado e vota para a classificação mais frequente (se for problema de classificação) ou faz a média entre elas (se for problema de regressão).

Parâmetros do KNN:

- N_neighbors: Numero de vizinhos usados por padrão. O valor padrão é 5.
- Weights: Função de peso utilizada na predição. Os valores possíveis são: '*uniform*'(Todos os pontos em cada vizinhança têm o mesmo peso), '*distance*'(O peso é o inverso de sua distância) ou uma função definida pelo usuário que aceita um *array* de distâncias e retorna um *array* com o mesmo formato contendo os pesos. O valor padrão é '*uniform*'.
- Leaf_size: O tamanho das folhas passadas para *BallTree* ou *KDTree*. O que pode afetar a velocidade de construção e busca, tanto como a memória necessária para armazenar a árvore. O valor padrão é 30.
- P: Parâmetro de potência para a métrica de *Minkowski*. Quando $p = 1$ é equivalente ao uso de *manhattan_distance* e $p = 2$ é equivalente à *euclidean_distance*. Para um valor de P arbitrário, é utilizado *minkowski_distance*. O valor padrão é 2.

1.2.3. *Random Forest*:

Random Forest pe um algoritmo de aprendizado supervisionado. A 'floresta' que o algoritmo cria é um conjunto de árvores de decisão, geralmente treinado com o método de

bagging. A ideia geral deste método é que uma combinação de modelos de aprendizado melhora o resultado total. O *random forest* pode ser utilizado para ambos problemas de classificação e de regressão.

Parâmetros do Random Forest:

- N_estimators: O número de árvores na floresta. O valor padrão é 100.
- Max_features: O número de características a se considerar quando buscar o melhor *split*. Pode ser um valor inteiro, de ponto flutuante, 'auto'($max\_features = sqrt(n\_features)$), 'sqrt'($max\_features = sqrt(n\_features)$, igual 'auto') ou 'log2'($max\_features = log2(n\_features)$). O valor padrão é 'auto'.
- Max_depth: A profundidade máxima da árvore. Se for 'None' os nós são expandidos até que todas as folhas sejam puras ou contenham menos que *min_samples_split* amostras. O valor padrão é 'None'.
- Min_samples_leaf: O valor mínimo de amostras necessárias em um nó folha. Um ponto de *split* em qualquer profundidade só será considerado caso sobre pelo menos min_samples_leaf amostras de treinamento em cada um dos ramos esquerdo e direito. O valor padrão é 1.
- Bootstrap: Se amostras *bootstrap* são usadas na construção de árvores. Se 'False' o *dataset* inteiro é utilizado na construção de cada árvore. O valor padrão é 'True'.

1.2.4. Hiper parametrização, *Cross Validation* e *k - folds*

Primeiramente, *Cross Validation* refere-se a divisão do *dataset* em casos de treino e casos de teste. Tal divisão serve como medida contra o *overfitting*, que é quando um modelo não possui uma boa capacidade de generalização, pois apenas repete dados do *dataset* que estão em sua memória. Um algoritmo que possui acurácia de 100% muitas vezes está com *overfitting*.

Por sua vez, *k-fold* é uma estrategia para se realizar o *Cross Validation*. O método constitui-se na divisão integral do *dataset* em *k* subconjuntos mutuamente exclusivos com o mesmo tamanho e, a partir daí, o primeiro subconjunto é utilizado como conjunto de validação, e o resto(k-1) subconjuntos são utilizados para o ajuste no modelo(*model fitting*), calculando a medida de corretude do modelo após cada ajuste. Na próxima execução, o segundo conjunto é tratado como validação, repetindo novamente o procedimento, até todos os conjunto servirem como validação. No fim, o modelo com maior corretude é mantido.

A hiper parametrização é simplesmente a testagem de vários modelos, até achar um com melhor corretude, mudando apenas os paramêtros. O módulo *scikit* possui um método denominado de *GridSearchCV*, que implementa o *Cross Validation* com o *k-fold*, e, ao mesmo tempo, se responsabiliza de hiper parametrizar o modelo. Isto implica que, para cada conjunto do *k-fold*, o método testa a lista de parâmetros inteira, da hiper parametrização.

1.3. Materiais utilizados

A linguagem de programação utilizada foi *Python*, em sua versão 3.9. Foram utilizados os módulos *scikit*, *seaborn*, *matplotlib*, *pandas*, e *imblearn* . A primeira é comumente utilizada em aplicações que utilizam aprendizado de máquina na linguagem especificada

, a segunda e terceira serviram de suporte para plotar os gráficos/histogramas que serão mostrados em 2.2, a terceira serviu para processar a entrada, que em sua natureza é um arquivo CSV, armazenado em um *Dataframe*, que é uma estrutura de dados que facilita a visualização/ aplicação de técnicas de aprendizado de máquina, por fim, a última serviu para fazer *Upsampling* para distribuir igualmente as entradas na coluna referente aos ganhadores.

2. Modelagem do problema

O *dataset* é [J 2021], no qual o link pode ser acessado pelas referências. Por sua vez, como explicado na especificação do trabalho, necessitamos de um trabalho já desenvolvido por outro autor, no *dataset* em questão, para a comparação de resultados. Foi escolhido [nelver 2021].

2.1. Descrição do *dataset* e *Feature engineering*

O dataset "cru"contém as colunas de: *rated, created_at, last_move_at, turns, victory_status, winner, increment_code, white_id, white_rating, black_id, black_rating, moves, opening_eco, opening_name* e *opening_ply*. A maioria das colunas possui o nome autoexplicativo, porém a coluna *increment_code* é o tempo que é incrementado após cada rodada, para os jogadores. Por exemplo, se o tempo incremental for de 10 segundos, e o tempo restante do jogador ser de 4 minutos após o término de seu movimento, o tempo resultante irá ser de 4 minutos e 10 segundos. Por sua vez, a coluna *opening_eco* refere-se a um código padrão das *openings* mais comuns. Por sua vez, *opening_ply* é o número de jogadas na fase de *opening*.Os valores das colunas serão estudados em 2.2. O *dataset* contém 20.058 entradas, retiradas da api da plataforma de xadrez online *Lichess*.

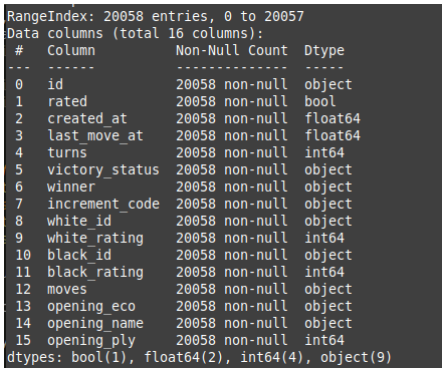
Encontrou-se a necessidade de se retirar as colunas de *created_at* e *last_move_at*, pois continham valores relacionados a data. Além disto, uma das duas colunas continha erro na coleta dos dados, pois em cerca de 40% da totalidade das linhas, se retirasse o valor de *last_move_at* de *created_at*, iria resultar em 0(deveria resultar em uma data, que seria, logicamente, a duração da partida.) Outra coluna que se retirou foi a *moves*. Tecnicamente, a coluna contém uma descrição detalhada do jogo. Uma relação possível de ser feita é com a coluna *victory_status*, que, caso for *mate*, basta apenas pegar a última jogada e ver qual lado que a fez, para se chegar no resultado desejado(previsão do ganhador). O modelo poderia ficar "viciado"nesta relação, aumentando sua acurácia substancialmente, porém ferindo a premissa de possuir dados limitados. Além disto, as últimas deleções feitas no *dataset* referêm se as colunas *white_id* e *black_id*, pois não contribuem em nada no nosso modelo.

Por fim, foi criada uma nova coluna, referente a diferença de *white_rating* e *black_rating*, denominada de *dif_rating*.

2.2. Análise descritivo dos atributos

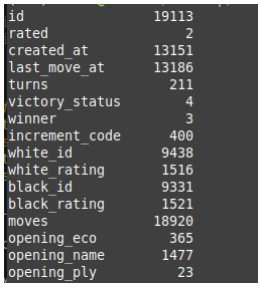
Primeiramente, observamos na figura 1 que possuímos 9 colunas com *dtype* de objeto (que representa uma *string* neste caso. O tratamento destas colunas é explicado em 3.2.

Por sua vez, a figura 2 é o resultado de uma função que conta os valores únicos em cada coluna, logo, valores repetidos não contam. Podemos constatar que temos alguma colunas com alta cardinalidade.



```
RangeIndex: 20058 entries, 0 to 20057
Data columns (total 16 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              20058 non-null  object
 1   rated           20058 non-null  bool
 2   created_at      20058 non-null  float64
 3   last_move_at    20058 non-null  float64
 4   turns           20058 non-null  int64
 5   victory_status  20058 non-null  object
 6   winner          20058 non-null  object
 7   increment_code  20058 non-null  object
 8   white_id        20058 non-null  object
 9   white_rating    20058 non-null  int64
 10  black_id        20058 non-null  object
 11  black_rating    20058 non-null  int64
 12  moves           20058 non-null  object
 13  opening_eco     20058 non-null  object
 14  opening_name    20058 non-null  object
 15  opening_ply     20058 non-null  int64
dtypes: bool(1), float64(2), int64(4), object(9)
```

Figura 1. *dataframe.info()* do *dataset* inicial - Fonte: Própria autoria.



```
id                19113
rated                 2
created_at        13151
last_move_at      13186
turns               211
victory_status        4
winner                3
increment_code      400
white_id           9438
white_rating       1516
black_id           9331
black_rating       1521
moves             18920
opening_eco         365
opening_name       1477
opening_ply          23
```

Figura 2. Valores únicos em cada coluna - Fonte: Própria autoria.

Para as próximas etapas da Análise descritiva, foi realizado o processo descrito em 3.1 e o primeiro parágrafo de 3.2. Ou seja, dividimos o *dataset* em *target* e *feature*, bem como aplicamos o processo de *Label Enconding*. Outra premissa é que os valores da coluna do *target* são: 0 - Peças pretas ganham, 1 - Empate e 2 - Peças brancas ganham. Tal distribuição foi adotada pelo *Label Enconder.*

Primeiramente, foi feito um diagrama de correlação na variável *target*, que atribui valores numéricos entre as variáveis. Podemos observar na figura 3 que as variáveis que possuem maior correlação com a *target* são *white_rating e dif_rating*. Também é possível observar que, logicamente, quanto mais *white_rating* contribui para a *target*, menos a váriavel *black_rating* contribui, justamente por serem opostas. Outra observação que pode se fazer é justamente o porquê da *white_rating* contribuir mais, e a razão é que, por natureza, xadrez é um jogo que possui vantagem para as brancas, por se moverem primeiro, fato que pode ser comprovado no histograma da variável *target*, a figura 4.

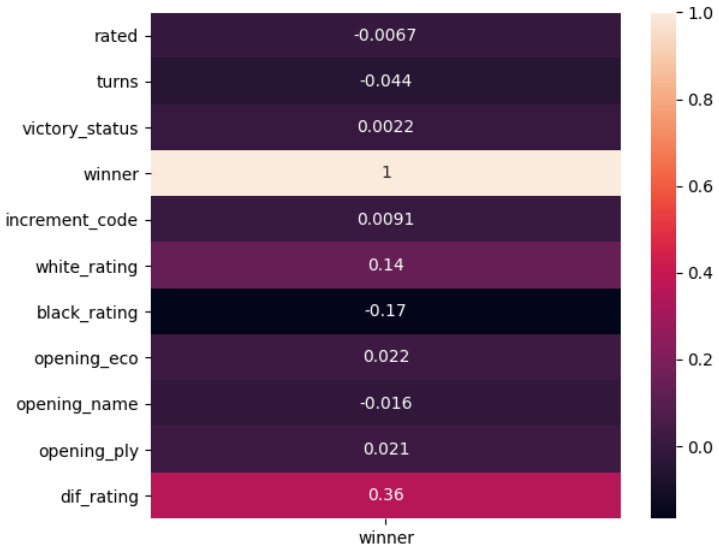


Figura 3. Gráfico de correlação para a variável *winner* - Fonte: Própria autoria.

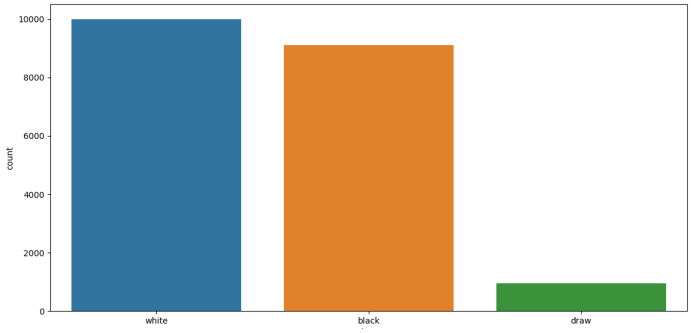


Figura 4. Histograma da variável *winner* - Fonte: Própria autoria.

Com as variáveis *black_rating, white_rating* e *dif_rating*, foram plotados gráficos de pares, para tentar observar visualmente uma relação entre os pontos. O resultado é a figura 5, no qual é possível visualmente constatar que talvez a separação dos pontos baseado em suas distâncias a uma linha talvez seja possível, mas talvez não de uma acurácia tão boa, pois existem muitos pontos que vão ”contra a separação lógica”. A última afirmativa pode ser entendida que existem muitos pontos fora de um padrão lógico, por exemplo, em uma área com um número maior de pontos brancos, significando a vitória das peças pretas, pode ter um ou dois pontos azuis, significando a vitória das peças brancas. Tal constatação é melhor demonstrada na figura 6, no qual é possível observar que existem alguns pontos fora da curva(que é a caixa, nos gráficos).

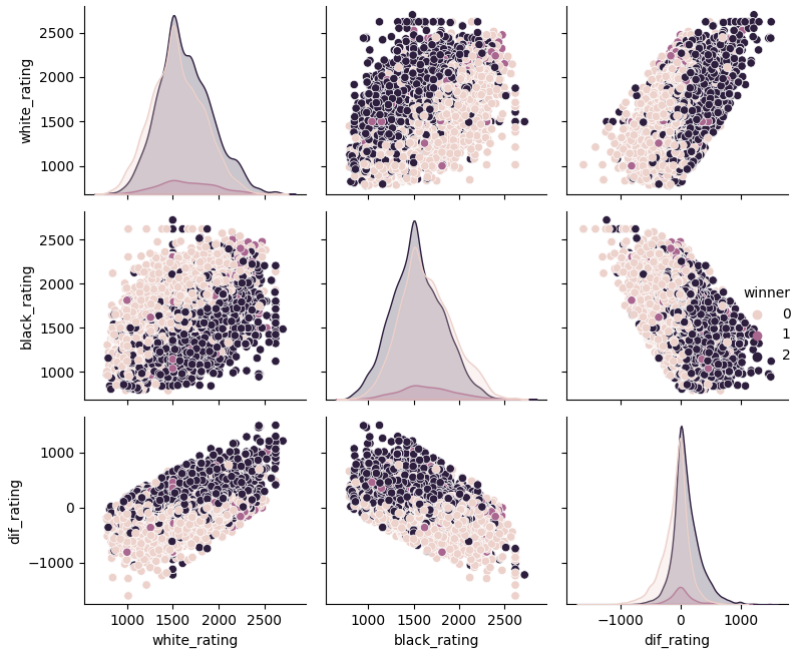


Figura 5. Gráfico de pares com as *features* pré-selecionadas. - Fonte: Própria autoria.

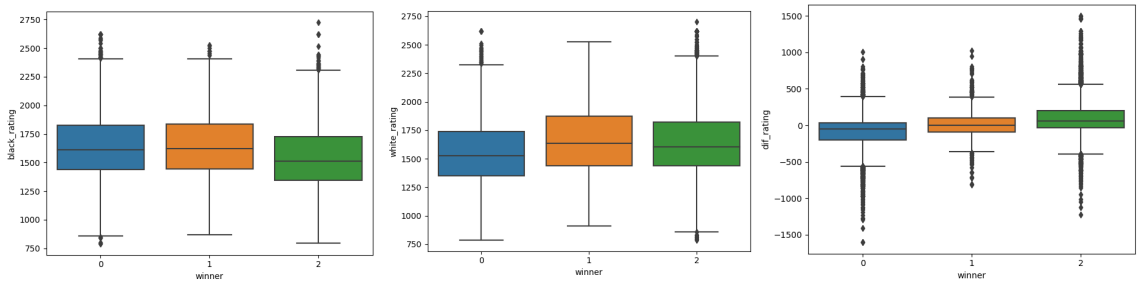


Figura 6. Gráfico de "caixa"(boxplot) das variáveis features pré-selecionadas. - Fonte: Própria autoria.

2.2.1. Histograma das variáveis utilizadas

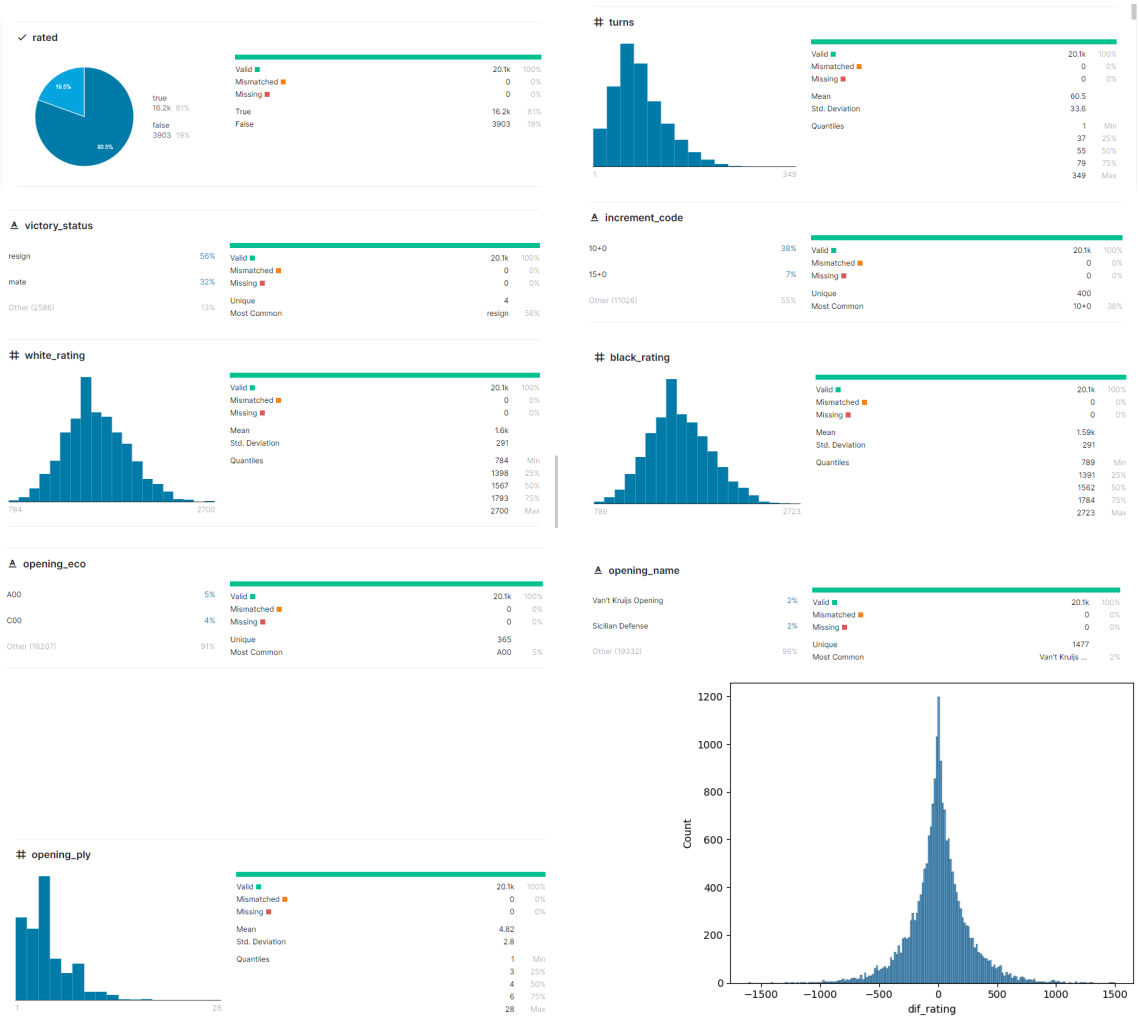


Figura 7. Histograma das variáveis utilizadas. - Fonte: 9 primeiras: [J 2021], última: Própria autoria.

3. Funcionamento do Algoritmo

3.1. Divisão de variáveis

A variável "Y"comumente utilizada em algoritmos de aprendizado de máquina é denominado daqui para frente de *target*. No nosso caso, o *target* é a coluna *winner* do *Dataframe*.

Por sua vez, a variável X é denominada de *features*. No nosso caso, são todas as outras colunas no *Dataframe*, que são *rated, turns, victory_status, winner, increment_code, white_rating, black_rating, opening_eco, opening_name, opening_ply* e *dif_rating*

3.2. Pré processamento

Encontrou-se a necessidade de pré processar os valores das *features*. Primeiramente, as colunas *victory_status, winner, increment_code, opening_eco, opening_name* são valores no formato de *object* no *dataframe*, logo, houve a necessidade de transformar os valores em inteiros. Para isso, foi utilizado o método de *Label Enconder*, do módulo *scikit*. Basicamente, atribui-se *labels* nas *strings*, nunca repetindo uma mesma *label* para valores distintos. Também era possível utilizar o *One Hot Encoded* para isto, mas devido a natureza dos valores(ordem não importa), optou-se pelo primeiro.

Outra mudança necessária foi a utilização do *SMOTE*, do módulo *imblearn* para realizar o *Upsampling* da coluna *target*.(coluna *winner*). O método em questão normaliza a distribuição das classes. Logo, no final, cada classe possui uma distribuição igualitária de valores(33%, já que utilizamos a classe de 0(”*black_win*”), 1(”*draw*”) e 2(”*white_win*”). (nota-se justamente a influência do *Label Enconder* nas classes, pois estão mapeadas devido a sua ocorrência alfabética.)).

Por fim, utilizou-se o *MinMaxScaler*, do módulo *scikit* nas colunas de *features*, para normalizar o valor das *features* para [0,1], a fim de tirar o *bias* no nosso modelo que seria advindo justamente de valores medidos com escalas distintas, não contribuindo igualmente ao ajuste do modelo(*model fitting*). Testou-se outros métodos para isto, sem grandes mudanças significativas na acurácia final, logo, decidiu-se o uso do *MinMaxScaler* pois difere do estudo [nelver 2021], no qual é utlizando o *Standard Scaler*.

3.3. Hiper parametrização das técnicas de AM

O algoritmo foi hiper parametrizado usando *GridSearchCV* do módulo *scikit*, que internamente utiliza *cross-validation* com *k-folds*.

Foram utilizados três modelos para tentar uma maior acurácia no problema. Cada um dos três se utilizou da hiper parametrização. Os paramêtros testados em cada algoritmo podem ser vistos nas figuras 8, 9 e 10

```
param_grid_svm = [
                 {'C': [1, 10, 20], 'gamma': [0.01,0.001,'scale','auto'], 'kernel': ['linear','rbf','sigmoid']},
                 {'C': [1, 10, 20], 'degree': [2,3], 'gamma': [0.01,0.001,'scale','auto'], 'kernel': ['poly']}
]
```

Figura 8. Parâmetros para o *SVM*. - Fonte: Própria autoria.

```
param_grid_knn = [
                {'n_neighbors': [1,5, 10,20,40,1000], 'weights': ['distance','uniform'],'leaf_size': [1,5,10,20,40,1000],
                 'p':[1,2]},
]
```

Figura 9. Parâmetros para o *kNN*. - Fonte: Própria autoria.

```
param_grid_rf = [
                {'n_estimators':[100,200],'max_features':['auto','sqrt'],'max_depth':[10,50,100],
                 'min_samples_split':[2,5,10],'min_samples_leaf': [1, 2, 4],'bootstrap':[True,False]}
]
```

Figura 10. Parâmetros para o *Random Forest*. - Fonte: Própria autoria.

4. Resultados obtidos

4.1. *SVM*:

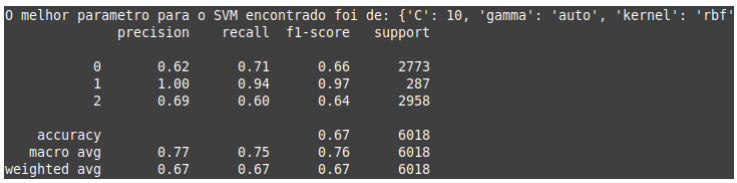


Figura 11. Resultados obtidos com o modelo *SVM*. - Fonte: Própria autoria.

4.2. *kNN*:

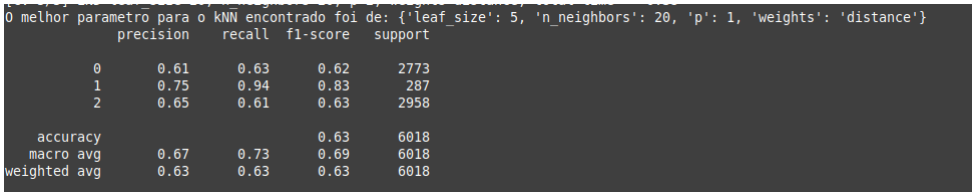


Figura 12. Resultados obtidos com o modelo *kNN*. - Fonte: Própria autoria.

4.3. *Random Forest*:

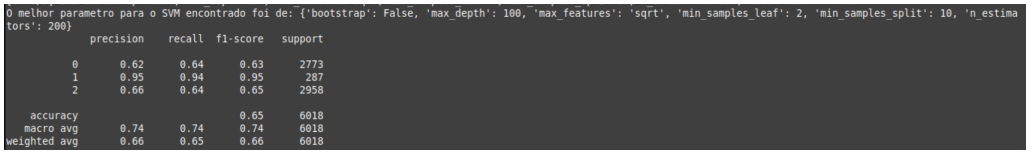


Figura 13. Resultados obtidos com o modelo *Random Forest*. - Fonte: Própria autoria.

Errata: na figura 13, o terminal aponta que o paramêtro foi encontrado para *SVM*. Entretanto, é só um erro no *print*. É possível ver que o algoritmo ainda retorna parâmetros referentes a *Random Forest*, ao invés de *SVM*.

4.4. Comparação com o estudo [nelver 2021]

Os resultados obtidos em [nelver 2021] foram de 64%, utilizando-se de regressão logística, e 73% utilizando árvores de decisão. Entretanto,vale ressaltar que no estudo citado, o autor transformou o problema em um problema com classes binárias, pois ele retirou as entradas referentes aos empates, o que não reflete a realidade, pois é possível existir o empate entre dois jogadores. Além disto, ele não precisou utilizar *Upsampling*, pois ele deixou o problema não balanceado(com vantagem para as peças brancas), e utilizou outro método de *Sampling* para normalizar os valores das *features*.

Apesar disto, os resultados não diferem muito. Ambos apresentam acurácia relativamente baixa, mas entendível, visto a natureza do problema.

Referências

Castro, C. (1994). Uma história cultural do xadrez. In *Cadernos de Teoria da Comunicação*, pages 3–12.

J, M. (2021). Chess game dataset (lichess). `https://www.kaggle.com/datasnaek/chess`. Dataset - Acessado em 25/02/2021.

nelver (2021). Lichess: predicting a winner. `https://www.kaggle.com/nelver/lichess-predicting-a-winner`. *Notebook* - Acessado em 26/02/2021.