Universidade do Minho

Mestrado Integrado em Engenharia Informática

Projeto - Análise de Dados: Dataset Chess Game

Perfil de Ciência de Dados
Aprendizagem Automática II

Diogo Dias - PG42825
Nelson Costa - PG44587

Docente
Miguel Rocha

Braga, Portugal
7 de junho de 2021

Conteúdo

1 Introdução **2**

2 Exploração de Dados **2**
2.1 Dataset 2
2.2 Missing Values 3
2.3 Duplicated Values 4
2.4 Exploração gráfica dos dados 5

3 Pré-Processamento **7**

4 Modelos de Machine Learning **8**
4.1 Divisão do Dataset 8
4.2 Construção de Modelos 8

5 Análise de Resultados **9**
5.1 K-Neighbors 9
5.2 SVM 9
5.3 Random Forest 9
5.4 Adaptive Boosting 10
5.5 MLP Model 10

6 Conclusão **10**

1 Introdução

No âmbito da unidade curricular de Aprendizagem Automática II foi proposta a realização de um trabalho prático, o qual tem como objetivos a análise de uma base de dados no ambiente python, em que as principais etapas são:

- Colheita/preparação de dados;
- Análise exploratória de dados;
- Pré processamento de dados;
- Construção de modelos;
- Otimização de hiperparâmetros;
- Predição através dos dados de teste;
- Avaliação do modelos.

Em Machine Learning, o computador aprende a realizar uma tarefa treinando um conjunto de dados. Em seguida, o computador realiza a mesma tarefa com os dados de teste. A aprendizagem supervisionada é de dois tipos: baseada em classificação e baseada em regressão. O problema apresentado neste projeto é supervisionado, uma vez que pretendemos efetuar uma medição da variável resposta (winner) que fornece a informação do vencedor (peças brancas, peças pretas, empate) num jogo de xadrez. Esta medição é realizada com base nas restantes variáveis (preditores). Portanto estamos a usar a aprendizagem supervisionada baseada em classificação.

A base de dados escolhida contém 20058 exemplos e 16 features. Através do pré processamento será verificada a existência de dados em falta, dados duplicados e posteriormente serão tratados esses mesmos dados. Para treinar a máquina, dividimos o conjunto de dados em duas partes (dados de treino e dados de teste). Esta divisão foi realizada assegurando que cada classe apresenta o mesmo número de exemplos. Assim a máquina irá treinar o modelo com o conjunto de dados de treino e posteriormente irá testar o conjunto de dados de teste. Segue-se a construção de modelos e a otimização de hiperparâmetros. Finalmente, é avaliado o modelo com a apresentação de métricas que permitam concluír se o modelo usado foi eficiente na tarefa de predição.

2 Exploração de Dados

2.1 Dataset

O *dataset Chess Game* foi retirado da plataforma Kaggle. O *dataset* é composto por 16 variáveis, entre as quais:

- *id*: identificação da partida;
- *rated*: identifica se a partida conta para o rating;
- *created_at*: quando o jogo começou;
- *last_move_at*: quando o jogo acabou;

- *turns*: número de turnos que a partida teve;
- *victory_status*: tipo de resultado da partida (empate, check mate, desistiu);
- *winner*: indica o vencedor, ou empate;
- *increment_code*: número de segundos que são adicionados ao tempo depois de uma jogada;
- *white_id*: identificação do jogador das peças brancas;
- *white_rating*: classificação do jogador das peças brancas;
- *black_id*: identificação do jogador das peças pretas;
- *black_rating*: classificação do jogador das peças pretas;
- *moves*: todas as jogadas em notação de xadrex;
- *opening_eco*: código que representa uma opening (movimento de abertura) em xadrex (lista de todas as openings);
- *opening_name*: nome do movimento de abertura;
- *opening_ply*: número de jogadas na fase de opening.

2.2 Missing Values

De modo, a detetar os valores em falta, foi elaborado um algoritmo de forma a percorrer as linhas de cada coluna e a validar se existem missing values. Esta validação apresenta para cada atributo a frequência relativa percentual de dados completos. Pela análise da figura, observamos que todos os campos estão populados, ou seja não existem missing values no *dataset*.

```
Verifica o quanto populados as colunas estão:
id está populado:  100.0 %
rated está populado:  100.0 %
created_at está populado:  100.0 %
last_move_at está populado:  100.0 %
turns está populado:  100.0 %
victory_status está populado:  100.0 %
winner está populado:  100.0 %
increment_code está populado:  100.0 %
white_id está populado:  100.0 %
white_rating está populado:  100.0 %
black_id está populado:  100.0 %
black_rating está populado:  100.0 %
moves está populado:  100.0 %
opening_eco está populado:  100.0 %
opening_name está populado:  100.0 %
opening_ply está populado:  100.0 %
```

Figura 1: Verificação de Missing Values

2.3 Duplicated Values

Através da verificação do número de dados duplicados observamos que existem 945 jogos com o id duplicado no *dataset*.

```
dupe = cd.duplicated(subset=['id'])
sum(dupe)
```

```
945
```

Figura 2: Verificação de jogos com id duplicado

De seguida pretende-se eliminar esses mesmos jogos:

```
cd = cd.drop_duplicates(subset=['id'])
```

Figura 3: Remoção de jogos duplicados através do id

Foi criado um algoritmo para validar a unicidade das features. Este algoritmo produz como resultado a percentagem em que os dados são únicos, isto é não produzem dados repetidos.

```
for column in cd.columns:
      if cd[column].is_unique:
          print(column, "is unique: 100%")
      else:
          comp1 = len(cd[column]) # número de linhas
          comp2 = len(cd[column].drop_duplicates(keep=False))
          print(column, "is unique: ", comp2/comp1 * 100, "%")
```

```
id is unique: 100%
rated is unique:  0.0 %
created_at is unique:  56.908910165855694 %
last_move_at is unique:  57.21760058598859 %
turns is unique:  0.1046408203840318 %
victory_status is unique:  0.0 %
winner is unique:  0.0 %
increment_code is unique:  0.5075079788625543 %
white_id is unique:  39.41296499764558 %
white_rating is unique:  0.7900381938994401 %
black_id is unique:  39.04149008528227 %
black_rating is unique:  0.8475906451106577 %
moves is unique:  98.72861403233402 %
opening_eco is unique:  0.26683409197928115 %
opening_name is unique:  2.0666562025846282 %
opening_ply is unique:  0.010464082038403181 %
```

Figura 4: Unicidade das features

Podemos retirar as seguintes informações da figura:

- 39% dos jogadores só jogaram uma vez de peças brancas ou só uma vez de peças pretas.

- 98% das jogadas ('moves') foram únicas, ou seja, em 2% dos jogos há partidas que tiveram moves iguais a outros.
- 2% de Movimentos de Abertura ('opening_name') são únicos, só foram jogados uma vez no historial de 20k jogos.

2.4 Exploração gráfica dos dados

Com recurso ao gráfico circular averiguamos que a percentagem de jogos competitivos representa uma fatia de 80,92% de todas as partidas. No histograma, verifica-se que a frequência de turnos concentra-se em torno das 50 jogadas.

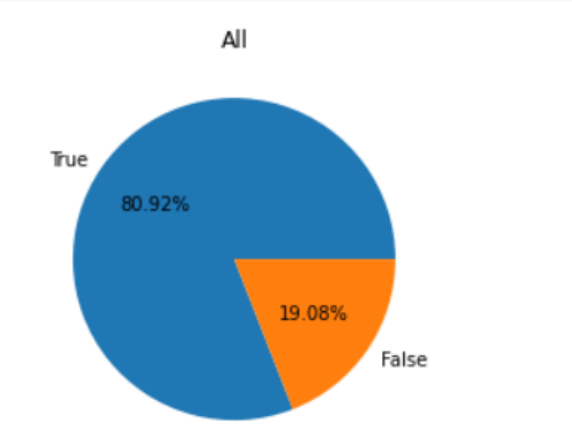

Figura 5: Gráfico circular - Jogos Competitivo (True) vs Jogos Casuais (False)

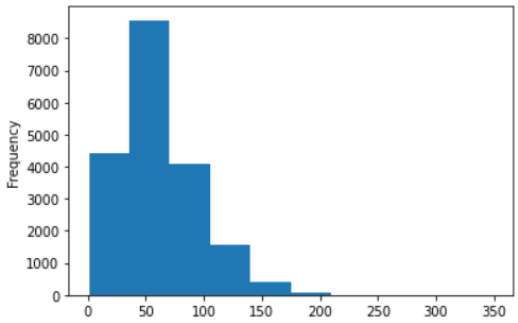

Figura 6: Frequência de turnos

O gráfico da figura 7 representa a percentagem de cada tipo de resultado de uma partida, e o gráfico da figura 8 representa a percentagem de vitórias de cada peça ou se é empate. No gráfico da figura 7 conseguimos observar que em 56% dos jogos eles acabam com um jogador a desistir, apenas 31% é que acabam em xeque mate. No gráfico da figura 8 vemos que no nosso dataset 50% das

partidas acabam com as peças brancas a ganharem, apenas 45% são vitórias para as peças pretas. Esta situação pode estar relacionada com o facto de as peças brancas terem o primeiro movimento numa partida.

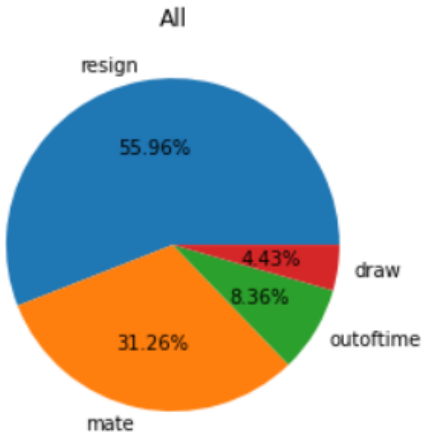

Figura 7: Percentagem de cada tipo de resultado de uma partida

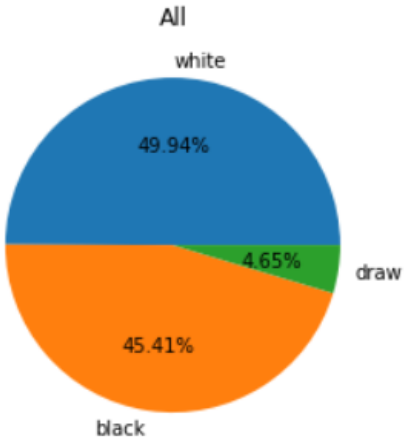

Figura 8: Percentagem de vitórias de cada peça (ou empate)

Aplicando a matriz de correlação aos dados conseguimos analisar que: *Created_at* e *Last_move_at*, features que é suposto representarem a data de inicio e fim do jogo, apresentam correlação muito forte. *White_rating* e *Black_rating* têm correlação forte, o que faz sentido, o site prioriza partidas em que os jogadores têm rating semelhante. *Opening_ply* expõe uma correlação fraca com o rating dos jogadores.

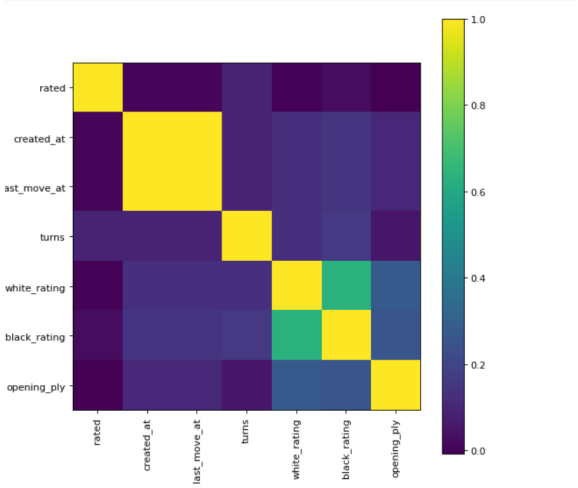

Figura 9: Matriz de Correlação

3 Pré-Processamento

De seguida torna-se fundamental remover as variáveis que não são interessantes para o modelo. O objetivo do trabalho é prever quem é o vencedor de uma partida de xadrex, ou então se a partida resultou em empate, através dos movimentos que os jogadores realizam ao longo do jogo.

As seguintes features foram removidas com o objetivo de desenvolver o melhor modelo de machine learning:

- *id*: Não tem nenhum impacto na partida;
- *rated*: Em príncipio não tem nenhum impacto na partida;
- *created_at* *last_move_at*: Não tem nenhum impacto na partida
- *turns*: Produz um modelo que é contra produtivo aos nossos objetivos, pois pelo número de turnos torna-se bastante fácil prever o resultado, visto que numa partida começa sempre o jogador das peças brancas. Por exemplo, numa partida que acabou em xeque-mate, se o número de turnos for ímpar, 13 por exemplo, foi o jogador das peças brancas o último a jogar, portanto presume-se que haja uma maior probabilidade de ter sido ele a ganhar. Se for um número par de turnos foi o jogador das peças pretas o último a jogar e é ele o presumido winner;
- *victory_status and incrementcode*: Não nos ajuda a descobrir quem é o vencedor;
- *white_id* *black_id*: Não nos interessa saber quem é o jogador;
- *moves*: Pelo mesmo raciocínio que levou a retirar a feature 'turns', também não vai ajudar no modelo.

Copia-se a coluna 'winner' para uma variável y e de seguida realiza-se uma operação de label encoding, que passa os valores de um tipo de variável categórica para um valor númerico. Assim consegue-se usar essa variável como target do modelo de Machine Learning.

```
le = LabelEncoder()
y = le.fit_transform(y)
y = pd.DataFrame({'winner': y})
```

Figura 10: Operação de label encoding

As features $opening_eco$ e $opening_name$ são convertidas para variáveis fictícias (dummy variable) que indicam a presença ou ausência de algum efeito categórico que possa ter impacto na feature 'winner'.

4 Modelos de Machine Learning

Nos capítulos abaixo está exposto o desenvolvimento dos modelos de aprendizagem supervisionada para a predição do vencedor de uma partida de Xadrez.

4.1 Divisão do Dataset

Após a normalização dos dados é realizada uma divisão dos dados, 70% para treino e 30% para teste e de seguida é calculada a frequência das classes da *target* feature. Observa-se que existe uma desproporcionalidade dos valores, mais especificamente o valor de empate, representando apenas 4% dos dados. Assim optou-se por utilizar o método $SMOTE$, método de oversampling, para popular os dados e chegar a um equilíbrio. O motivo para esta tomada de decisão é evitar o overfitting do modelo para a predição de vitória das peças brancas ou pretas (paralelamente, têm má precisão sobre os resultados de empate).

4.2 Construção de Modelos

Os modelos de Machine Learning utilizados foram os seguintes:

- K-Neighbors
- Support Vector Machine
- Random Forest Classifier
- Adaptive Boosting

Numa primeira fase correu-se os modelos com os default parameters de cada modelo, usando $5 - Fold\ CrossValidation$ para a previsão de resultados.

Posteriormente, usou-se a biblioteca GridSearchCV para percorrer através de um fixed value de parameters que definimos, em que o resultado enquadra as configurações ótimas para o *estimator* (modelo de ML) a ser usado. Assim, com esta Otimização de Hiper-Parâmetros conseguiu-se melhorar o valor de precisão dos nossos modelos. Utilizou-se também uma Neural Network, e variou-se os parâmetros de forma a admitir valores de accuracy satisfatórios.

5 Análise de Resultados

Neste capítulo são apresentados os resultados obtidos com os vários modelos para predição da vitória, segundo a métrica de *accuracy*.

5.1 K-Neighbors

Default	ANOVA	Otimizado
0.5983	0.4777	0.5466

Pela análise de resultados, verifica-se que o modelo K-Neighbours, apresentou valores de *accuracy* mais baixos, com a aplicação da seleção de atributos (ANOVA) comparativamente com as restantes versões do modelo. Por *default* obteve-se uma *accuracy* mais alta para o modelo dos KNeighbours. O modelo otimizado teve em conta o número de vizinhos mais próximos, os pesos *(uniform, distance)* e a métrica *(euclidean, manhattan)*.

5.2 SVM

Default	ANOVA	Otimizado
0.5783	0.4851	0.6054

Com o modelo SVM, obteve-se 48,51% de *accuracy* com a aplicação da seleção de atributos (ANOVA), representando o valor de *accuracy* mais baixo entre todos os modelos SVM. Por *default* obteve-se uma *accuracy* 57,83% representando um aumento considerável quando comparado com a versão anterior. A otimização do modelo SVM permitiu um aumento de *accuracy* de forma gradual e significativa.

5.3 Random Forest

Default	ANOVA	Otimizado
0.5814	0.5827	0.5905

Com o modelo Random Forest obtiveram-se valores estáveis de *accuracy* para as três versões do modelo. No entanto, pode ser salientado que a versão que produziu melhor *accuracy* foi o modelo otimizado. Este modelo foi otimizado utilizando o parâmetro *n_estimators* que estima a quantidade de árvores na Random Forest, um número de features para considerar em cada split *max_features*, o máximo número de níveis da árvore *max_depth*. Usou-se como critério *gini* e *entropy*.

5.4 Adaptive Boosting

Default	ANOVA	Otimizado
0.6198	0.6088	0.6321

O modelo Adaptive Boosting foi o que apresentou melhores resultados de *accuracy* para as versões *default*, ANOVA e otimizado. A *accuracy* tomou valores acima dos 60% para os três casos.

5.5 MLP Model

Accuracy	Loss	Epochs	Batch_Size
0.53	0.5925	25	150

Paralelamente, aos modelos mencionados atrás também se utilizou uma Neural Network que produziu resultados de *accuracy* nos 53%. Para esta rede utilizaram-se 25 *epochs* e um *batch_size* de 150. O valor de loss do modelo localiza-se em torno dos 0.5925. Com a variação dos parâmetros para valores demasiado altos ou baixos foi possível observar que a *accuracy* do modelo tomava valores bastante altos causando overfitting no modelo. Assim procurou-se um compromisso entre um número estável de *epochs* e de *batch_size* que garantisse uma performance viável do modelo.

6 Conclusão

Com a realização deste trabalho conseguimos expandir os nossos conhecimentos na área de machine learning, mais especificamente conseguimos aplicar modelos para aprendizagem supervisionada, e conseguimos desenvolver a nossa capacidade de construir modelos. Foi possível melhorar a qualidade dos modelos atráves do uso de técnicas de oversampling, de otimização de hiperparâmetros, mas também melhorar as nossas habilidades em termos de exploração de dados e pré-processamento. Apesar de os resultados de predição dos modelos serem satisfatórios, não é possível melhorar muito os modelos. Uma forma de melhorá-los seria com a inclusão de mais dados. Uma possível forma de realizar o trabalho seria expandindo o pré-processamento da feature *moves*, em que se separa por primeiro move, segundo move... para depois usar num modelo LSTM de Time Series Forecasting. O modelo que apresentou melhores resultados foi o Adaptive Boosting com uma *accuracy* acima dos 60%. No entanto, todos os modelos apresentaram de um modo geral variações de *accuracy* entre 50% e 60%. Com a otimização dos hiperparâmetros dos modelos produziram-se melhorias em relação aos modelos com parâmetros por *default*. A seleção dos valores para otimizar os hiperparâmetros foi realizada de forma a não comprometer a viabilidade do modelo, quer em *overfitting*, quer em *underfitting*. Em suma, o trabalho prático realizado permitiu não só uma maior capacidade de análise de dados, como também inferir conhecimentos na grande área de Ciência de Dados.