

Nanodegree Engenheiro de Machine Learning

Projeto Final

Previsão de colocação final dos jogadores em uma partida de Playerunknown's Battlegrounds (PUBG)

Raphael Brito Alencar
25 de Junho de 2019

Definição

Visão geral do projeto

Jogos de vídeo game no estilo *Battle Royale* ganharam muita força nos últimos tempos. 100 *players* são jogados em uma ilha, sem carregar nenhum tipo de itens, precisando explorar, percorrer a ilha e eliminar outros *players* até que apenas um fique vivo, isso tudo enquanto a zona do mapa no jogo vai encolhendo.

Playerunknown's Battlegrounds (PUBG) desfruta de grande popularidade. Com mais de 50 milhões de cópias vendidas, é o quinquagésimo jogo mais vendido de todos os tempos e possui milhões de players ativos por mês.

O time de criadores do PUBG tornou público dados de partidas para que os curiosos e estudantes pudessem explorá-los e analisá-los. Sendo assim, a plataforma Kaggle coletou os dados e criou uma competição onde os participantes devem desenvolver modelos de previsão para a classificação dos *players* de acordo com as *features* fornecidas. São disponibilizados mais de 65000 dados de jogadores anônimos em diversas partidas do jogo, divididos em conjuntos de treinamento e testes.

Com esses dados, podemos descobrir qual a melhor estratégia para vencer uma partida e analisar quais as chances de vitória de uma *player* de acordo com suas estatísticas.

Este documento sumariza todas as etapas do processo de machine learning aplicadas ao trabalho que tem todo seu código de forma detalhada no Jupyter notebook.

Descrição do problema

Até 100 jogadores começam uma partida de PUBG, que possui um identificador (`matchId`). Os jogadores podem estar em equipes ou sozinhos e são classificadas no final do jogo com base em quantas outros *players* ainda estão vivos. Durante a partida, os jogadores podem pegar diferentes armas e munições, reviver companheiros de equipe abatidos, dirigir veículos, nadar, correr e experimentar todas as consequências de suas atitudes.

Munidos de um grande número de estatísticas de partidas e formatados para que cada linha contenha as estatísticas pós-jogo de um jogador, os usuários do Kaggle participantes da competição devem criar um modelo que preveja a colocação final dos *players* com base em suas estatísticas finais, em uma escala de 1 (primeiro lugar) a 0 (último lugar).

No presente trabalho será detalhada a forma escolhida para resolução desse problema, começando com uma análise exploratória dos dados, utilizando agrupamentos e visualizações, seguida de uma engenharia de dados, tratando valores nulos e faltantes, excluindo *outliers* e por fim aplicando técnicas de Machine Learning para seleção de *features* e escolha do melhor modelo e hiperparâmetros para aperfeiçoar a previsão final.

Métricas

Para a competição do Kaggle foi escolhida a métrica *Mean Absolute Error* (MAE):

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| .$$

O erro médio absoluto mede a magnitude média dos erros em um conjunto de previsões, sem considerar sua direção. É a média da amostra de testes, da diferença absoluta entre a previsão e a observação atual em que todas as diferenças individuais têm peso igual.

Os dados fornecidos para o desenvolvimento deste projeto possuem uma grande quantidade de *outliers*, sendo esta uma das justificativas para a escolha do MAE como métrica, dado que ela penaliza erros maiores e, dessa forma, não é tão sensível a valores discrepantes como, por exemplo, o erro quadrático médio.

Análise

Exploração dos dados

Os dados estão disponíveis em dois arquivos do tipo csv, sendo o `train_V2.csv` o conjunto de treinamento e `test_V2.csv` o conjunto de testes. Ao final do projeto é preciso gerar um novo arquivo, `submission.csv`, com os valores das chances de vitória para cada *player*.

O conjunto de treinamento possui 29 colunas e 4446966 linhas, enquanto que o conjunto de testes possui 28 colunas (não possui a variável alvo) e 1934174 linhas. Cada linha corresponde a um registro com os dados estatísticos e informativos de um *player* em uma partida. As colunas são:

1. DBNOs - número de inimigos nocalteados.
2. assists - número de inimigos feridos pelo jogador, mas que foram finalizados por outros jogadores do time, que o mesmo pertence.
3. boosts - quantidade de itens de incremento de atributos que o jogador usou.
4. damageDealt - dano total causado. Nota: Dano infligido contra si próprio é subtraído.
5. headshotKills - número de inimigos abatidos com tiro na cabeça.
6. heals - quantidade de itens de cura utilizados.
7. Id - identificador do jogador.
8. killPlace - rank, na partida, do número de inimigos abatidos.
9. killPoints - rank externo, baseado no número de inimigos abatidos. (Pensar como um elo de rank onde apenas os abates importam). Se houver um valor diferente de -1 em rankPoints, então qualquer valor 0 em killPoints deve ser tratado como *None*.
10. killStreaks - número máximo de inimigos abatidos em um curto espaço de tempo.
11. kills - número de inimigos abatidos.
12. longestKill - maior distância entre o jogador e o inimigo no momento do abate. Esse atributo pode ser enganoso, dado que o jogador pode nocautear o inimigo e se mover para longe até que o inimigo de fato morra.
13. matchDuration - duração da partida em segundos.
14. matchId - identificador da partida. Não há correspondências nos conjuntos de treinamento e de testes.
15. matchType - String que identifica o tipo da partida. Os tipos padrão são “solo”, “duo”, “squad”, “solo-fpp”, “dua-fpp” e “squad-fpp”; outros modos são de eventos ou partidas customizadas.
16. rankPoints - elo de rank dos jogadores. Esse rank é inconsistente e será descontinuado na próxima versão da API, por isso deve ser usado com cautela. Valores -1 são decorrentes de valores *None*.
17. revives - número de vezes que ressuscitou jogadores da sua equipe.
18. rideDistance - distância total percorrida em veículos medida em metros.
19. roadKills - número de abates enquanto estava em um veículo.
20. swimDistance - distância total percorrida nadando medida em metros.

21. teamKills - quantidade de vezes que o jogador abateu parceiros de equipe.
22. vehiclesDestroys - número de veículos destruídos
23. walkDistance - distância total percorrida a pé medida em metros
24. weaponsAcquired - número de armas coletadas
25. winPoints - rank externo baseado em vitórias dos jogador. (Pensar como um elo de rank onde apenas o número de vitórias importa). Se houver um valor diferente de -1 em rankPoints, então qualquer valor 0 em winPoints deve ser tratado como *None*.
26. groupId - Identificador do grupo que o jogador pertence dentro da partida. Se o mesmo grupo de jogadores jogar em partidas diferentes, eles terão groupId diferentes em cada partida.
27. numGroups - número de grupos que se tem dados na partida.
28. maxPlace - pior colocação para a qual tem dados na partida. Este atributo pode não coincidir com numGroups, pois as vezes os dados ignoram as colocações.
29. winPlacePerc - variável target da predição. Este é o percentual de chance de vitória, onde 1 corresponde ao primeiro lugar e 0 corresponde ao último lugar. Ele é calculado a partir de maxPlace, não de numGroups, portanto, é possível ter valores faltantes em uma partida.

Os conjuntos de treinamento e testes ocupam 983.90 mb e 413.18 mb respectivamente. Ambos compostos por dados dos mais diversos tipos como object, int64 e float64. Mais adiante, na seção de metodologia, será mostrado um recurso utilizado para reduzir essa quantidade de memória utilizada, baseado na quantidade de bytes ocupados por cada tipo.

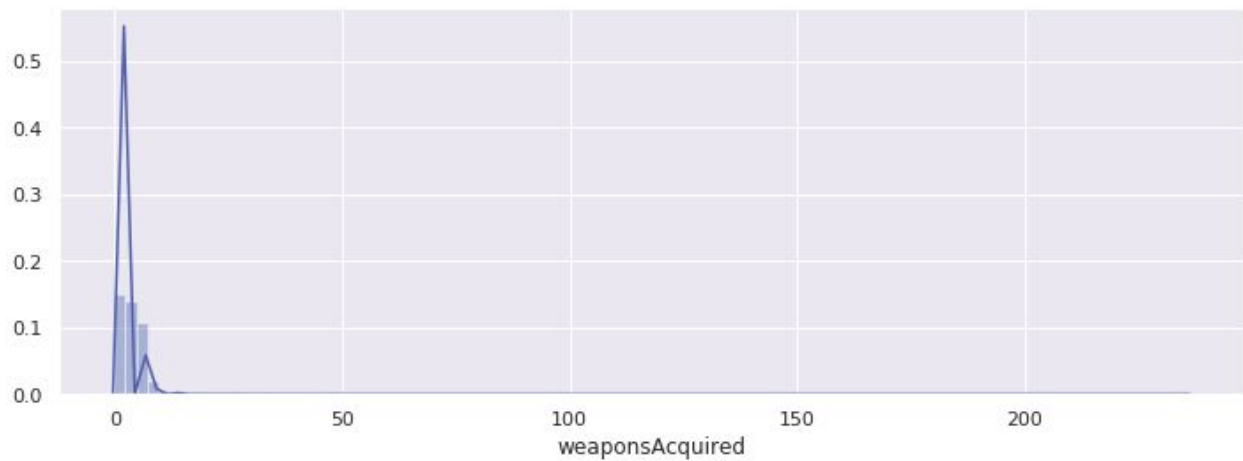
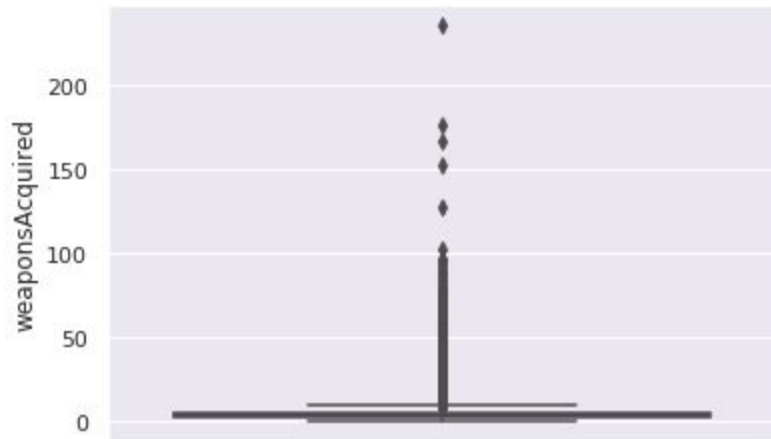
O conjunto de treinamento possui apenas um valor nulo que se encontra exatamente na variável target (winPlacePerc) e dentro do universo de dados disponibilizados, o registro que contém esse valor nulo não terá relevância, por isso o registro pode ser excluído. Além disso, existem algumas variáveis categóricas, como Id, groupId e matchId, que deverão ser tratadas para não influenciar na previsão.

Segue uma breve descrição estatística da distribuição dos dados:

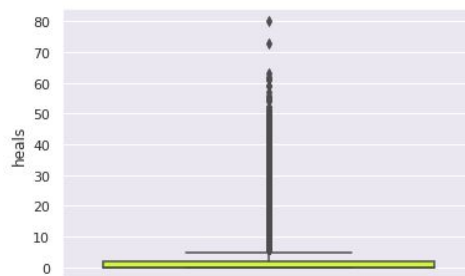
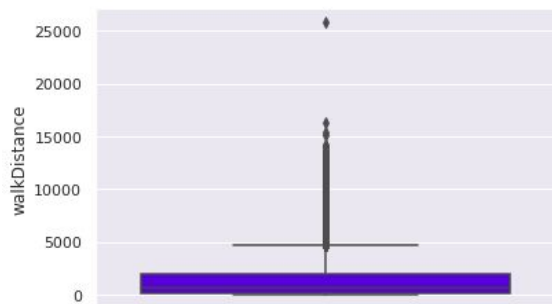
	mean	std	min	25%	50%	75%	max
assists	0.233815	0.588573	0.0	0.000000	0.000000	0.0000	22.0
boosts	1.106908	1.715794	0.0	0.000000	0.000000	2.0000	33.0
damageDealt	130.633148	169.886963	0.0	0.000000	84.239998	186.0000	6616.0
DBNOs	0.657876	1.145743	0.0	0.000000	0.000000	1.0000	53.0
headshotKills	0.226820	0.602155	0.0	0.000000	0.000000	0.0000	64.0
heals	1.370148	2.679982	0.0	0.000000	0.000000	2.0000	80.0
killPlace	47.599361	27.462931	1.0	24.000000	47.000000	71.0000	101.0
killPoints	505.006156	627.504921	0.0	0.000000	0.000000	1172.0000	2170.0
kills	0.924784	1.558445	0.0	0.000000	0.000000	1.0000	72.0
killStreaks	0.543955	0.710972	0.0	0.000000	0.000000	1.0000	20.0
longestKill	22.993486	51.476093	0.0	0.000000	0.000000	21.3200	1094.0
matchDuration	1579.506793	258.738814	133.0	1367.000000	1438.000000	1851.0000	2237.0
maxPlace	44.504680	23.828099	2.0	28.000000	30.000000	49.0000	100.0
numGroups	43.007602	23.289489	1.0	27.000000	30.000000	47.0000	100.0
rankPoints	892.010303	736.647791	-1.0	-1.000000	1443.000000	1500.0000	5910.0
revives	0.164659	0.472167	0.0	0.000000	0.000000	0.0000	39.0
rideDistance	606.092468	1496.470703	0.0	0.000000	0.000000	0.1910	40710.0
roadKills	0.003496	0.073373	0.0	0.000000	0.000000	0.0000	18.0
swimDistance	4.509241	30.237846	0.0	0.000000	0.000000	0.0000	3823.0
teamKills	0.023868	0.167394	0.0	0.000000	0.000000	0.0000	12.0
vehicleDestroys	0.007918	0.092612	0.0	0.000000	0.000000	0.0000	5.0
walkDistance	1148.517212	1180.552734	0.0	155.100006	685.599976	1976.0000	25780.0
weaponsAcquired	3.660488	2.456543	0.0	2.000000	3.000000	5.0000	236.0
winPoints	606.460267	739.700471	0.0	0.000000	0.000000	1495.0000	2013.0
winPlacePerc	0.472814	0.306804	0.0	0.200000	0.458300	0.7407	1.0

Visualização Exploratória

De acordo com a imagem acima, é possível identificar algumas características relevantes, como a presença de outliers, vide a média, a distribuição dos quartis e o valor máximo do atributo weaponsAcquired.

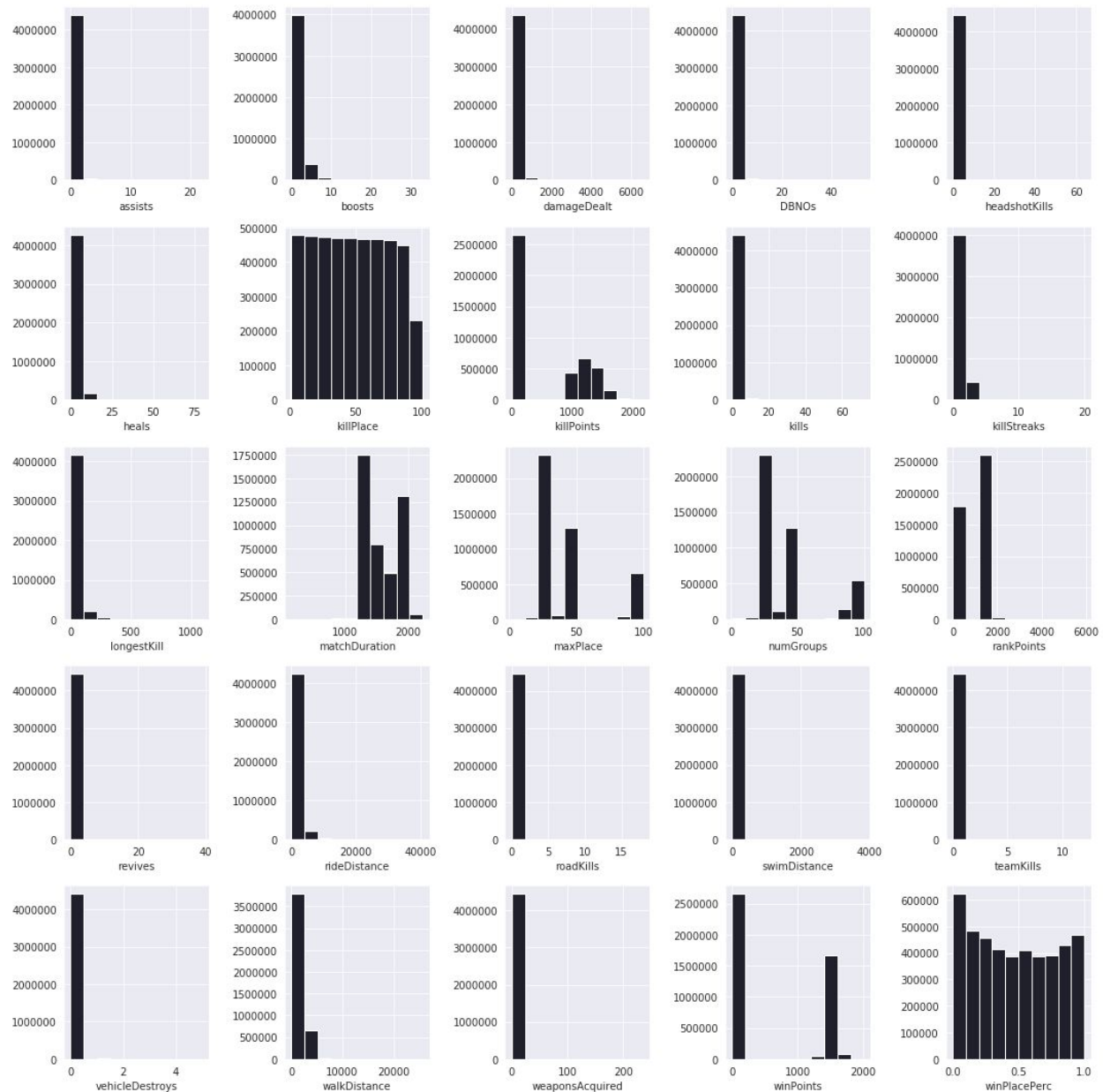


Os outliers também aparecem com frequência em outros atributos, entre eles walkDistance e heals:



O *box plot* é uma boa escolha, pois permite visualizar a distribuição dos valores discrepantes, fornecendo assim um meio complementar para desenvolver uma perspectiva sobre o caráter dos dados.

Um ponto importante a ser verificado, também, é que a maior parte dos atributos possuem uma distribuição assimétrica (*skewed distribution*), o que fica mais claro na imagem abaixo:



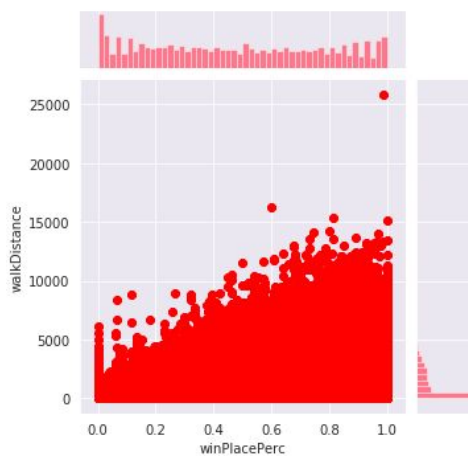
Verificando a matriz de correlação, percebe-se que as cinco variáveis independentes que melhor se correlacionam positivamente com a variável alvo são walkDistance, boosts, weaponsAcquired, damageDealt e heals. Por outro lado, as cinco que melhor se correlacionam negativamente são killPlace, matchDuration, winPoints, killPoints e rankPoints.

```

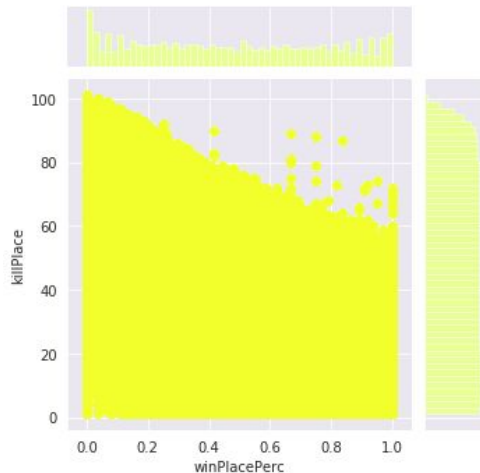
winPlacePerc      1.000000
walkDistance      0.810888
boosts            0.634234
weaponsAcquired   0.583806
damageDealt       0.440507
heals             0.427857
kills             0.419916
longestKill       0.410154
killStreaks       0.377566
rideDistance      0.342915
assists           0.299441
DBNOs            0.279970
headshotKills     0.277722
revives           0.240881
swimDistance      0.149607
vehicleDestroys   0.073436
numGroups         0.039621
maxPlace          0.037377
roadKills         0.034544
teamKills         0.015943
rankPoints        0.013523
killPoints        0.012908
winPoints         0.007061
matchDuration     -0.005171
killPlace         -0.719069
Name: winPlacePerc, dtype: float64

```

Levando em consideração o maior coeficiente de correlação de Pearson, segue a distribuição da variável walkDistance em relação a winPlacePerc, com coeficiente 0.81:



Para o menor coeficiente, killPlace em relação a winPlacePerc:

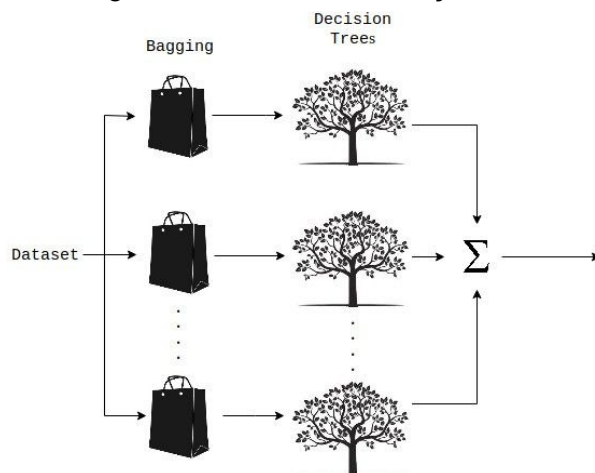


Os gráficos *jointplot* facilitam o entendimento da interação cruzada entre as variáveis traz informações mais reais sobre essas interações.

Algoritmos e técnicas

Baseado na quantidade de dados, pesquisa de outras soluções do Kaggle e limitações dos recursos disponíveis para treinamento, 13GB de RAM, optou-se por utilizar dois algoritmos, RandomForestRegressor e LightGBM.

O RandomForestRegressor é um estimador que ajusta várias árvores de decisão de classificação nas diversas amostras do conjunto de dados e usa a média para melhorar a precisão preditiva e controlar o over-fitting, importante devido a quantidade de dados. Basicamente é uma técnica de *ensemble* capaz de realizar tanto tarefas de regressão quanto de classificação com o uso de múltiplas árvores de decisão e uma técnica chamada *Bootstrap Aggregation*, comumente conhecida como *bagging*. *Bagging*, no *Random Forest*, envolve o treinamento de cada árvore de decisão em uma amostra de dados diferente, onde a amostragem é feita com substituição.

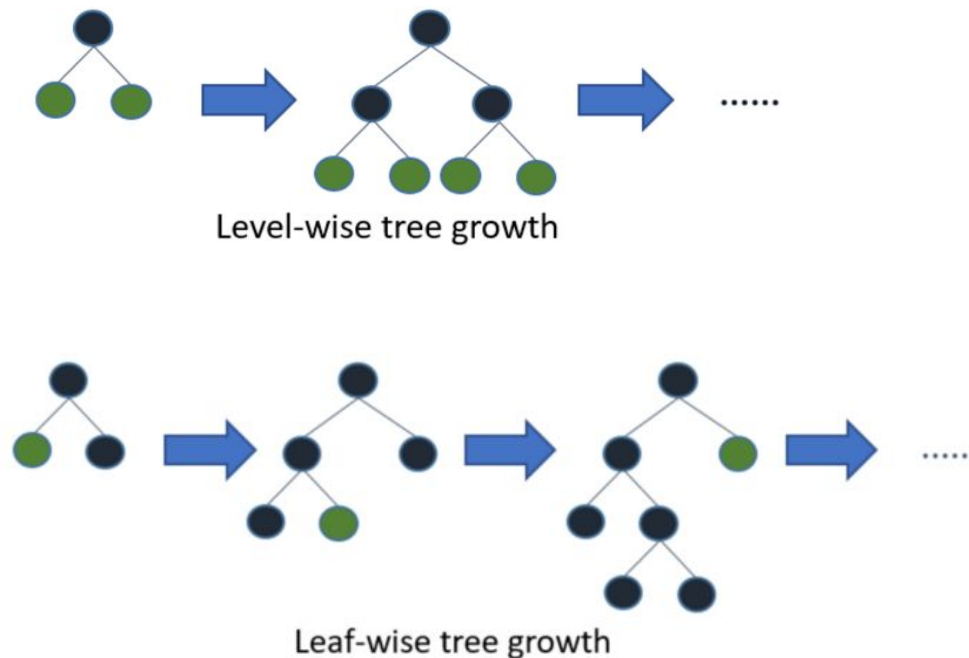


A idéia por trás dessa técnica é combinar múltiplas árvores de decisão para determinar o resultado final, em vez de depender de árvores de decisão individuais. O Random Forest foi escolhido por ser considerado um método altamente preciso, mesmo em uma larga proporção de dados com grande dimensionalidade, e robusto devido ao número de árvores de decisão no processo e porque ele pega a média de todas as previsões, o que elimina o *bias* evitando problemas de super ajuste.

O LightGBM é um *framework* de *gradient boost* que usa algoritmos de aprendizado baseado em árvores de decisão e que é projetado para ser distribuído eficientemente, tendo como principais vantagens:

1. Maior velocidade de treinamento e maior eficiência
2. Menor uso de memória
3. Melhor Precisão
4. Capaz de lidar com dados em larga escala

O lightGBM apresenta uma diferença básica em relação a outros algoritmos baseados em árvore de decisão. Enquanto que os outros algoritmos desenvolvem as árvores horizontalmente, ele aumenta a árvore verticalmente, o que significa que o lightGBM cresce em termos de folha da árvore (*leaf-wise*), enquanto os outros crescem em termos de nível (*level-wise*). Ele irá escolher a folha com a perda máxima de delta para crescer. Ao cultivar a mesma folha, o algoritmo *leaf-wise* pode reduzir mais a perda que um algoritmo *level-wise*. O diagrama abaixo mostra a diferença:



O lightGBM foi escolhido pois ele consegue lidar muito bem com grande massa de dados, que é o caso, além de fazer isso de forma extremamente rápida (por isso o prefixo *light*), usando pouca memória. Além disso, é um algoritmo focado na acurácia dos resultados.

Benchmark

Ao realizar diversos testes com diferentes tamanhos de amostras e diferentes configurações dos algoritmos, percebemos que o lightGBM se comporta melhor que RandomForest, principalmente no quesito velocidade. Para apenas 1% da amostra de dados de treinamento, os dois tiveram resultados de erro médio absoluto parecidos, porém o RandomForest com levou 15s para treinar, enquanto que o lightGBM levou apenas 3s como mostra a imagem abaixo.

```
RandomForestRegressor: 20211 samples
Training MAE: 0.035251453745564765
Training time: 14.988124132156372
Predictions time: 9.626047372817993

LGBMRegressor: 20211 samples
Training MAE: 0.03243436240640361
Training time: 2.9276435375213623
Predictions time: 17.034681797027588
```

Para 10% da amostra de dados, as coisas começam a ficar mais discrepantes. Embora os dois tenham obtido resultados de MAE muito próximos, o lightGBM treinou cerca de 150 vezes mais rápido.

```
RandomForestRegressor: 202114 samples
Training MAE: 0.028846537251161954
Training time: 2107.5815439224243
Predictions time: 122.58906102180481

LGBMRegressor: 202114 samples
Training MAE: 0.0308727910252393
Training time: 14.660771608352661
Predictions time: 15.715221881866455
```

Isso levou a escolha do lightGBM como melhor alternativa.

Metodologia

Pré-processamento de dados

A remoção do valor nulo existente no conjunto de dados foi a primeira tarefa de pré-processamento realizado. Dado que é apenas um registro do conjunto de treinamento, com o valor nulo exatamente na variável alvo, e tendo mais de quatro milhões de outros registros, o mesmo pode ser excluído sem gerar nenhuma perda de informação ou problema para a predição.

Devido a grande de memória ocupada pelos tipos de dados originais, uma função para redução de memória foi construída. Essa função, chamada de `reduce_mem_usage`, que recebe como parâmetro um `DataFrame`, percorre todos os tipos de dados presentes nas variáveis independentes e verifica se existe a possibilidade de transformá-los em outros que ocupem menos espaço na memória, sem prejudicar as manipulações nos dados executadas posteriormente. Os resultados de redução de memória foram bem satisfatórios, diminuindo o espaço ocupado pelo conjunto de treinamento em 65,5% e de teste em 66,1%.

```
Memory usage of dataframe is 983.90 MB --> 339.28 MB (Decreased by 65.5%)
Memory usage of dataframe is 413.18 MB --> 140.19 MB (Decreased by 66.1%)
(4446966, 29) (1934174, 28)
CPU times: user 25.6 s, sys: 8.81 s, total: 34.4 s
Wall time: 34.5 s
```

Para melhor organizar os tipos das partidas, `matchType` teve seus valores reduzidos a apenas três valores possíveis. As partidas do tipo 'solo-fpp', 'normal-solo', 'normal-solo-fpp' tiveram seu `matchType` alterados para 'solo', assim como 'duo-fpp', 'normal-duo', 'normal-duo-fpp', 'crashfpp', 'crashtpp' foram alterados para 'duo' e 'squad', 'squad-fpp', 'normal-squad', 'normal-squad-fpp', 'flarefpp', 'flaretp' para 'squad'. Outros atributos categóricos como `matchId` e `groupId` também passaram por um processo de *encode* para números inteiros, para serem utilizados nos algoritmos escolhidos.

Analisando separadamente em algumas das variáveis, foi possível excluir diversos valores discrepantes (*outliers*). Os valores foram excluídos, pois afetam diretamente no resultado final da predição.

Foram criadas novas variáveis combinando as variáveis originais, ou apenas ranqueando-as em valores percentuais:

```

match = data.groupby('matchId')
data['_killsPerc'] = match['kills'].rank(pct=True).values
data['_damageDealtPerc'] = match['damageDealt'].rank(pct=True).values
data['_walkDistancePerc'] = match['walkDistance'].rank(pct=True).values
data['_walkPerc_killsPerc'] = data['_walkDistancePerc'] / data['_killsPerc']

data['_totalDistance'] = data['walkDistance'] + data['rideDistance'] + data['swimDistance']
data['_items'] = data['heals'] + data['boosts']
data['_damageDealtAndWalkDistance'] = data['damageDealt'] + data['walkDistance']
data['_headshotRate'] = data['headshotKills'] / data['kills']
data['_killPlace_maxPlace'] = data['killPlace'] / data['maxPlace']

```

É possível perceber que outra técnica adotada para trabalhar com os dados foi o agrupamento dos dados em funções de alguns atributos como matchId (ver figura acima), groupId e matchType.

O método rank é utilizado quando se deseja ranquear um dataframe de acordo com os valores em alguma coluna. Aqui, foi realizado um agrupamento a partir do matchId, ou seja, por partida, e posteriormente criado um rank deste agrupamento de partidas, de acordo com algumas das *features* que melhor se correlacionam com o winPlacePerc. Essa técnica, traz a melhoria de sabermos qual foi a classificação de um jogador em uma partida, de acordo, por exemplo, com o número de abates.

Dado que se trata de um jogo com diversos jogadores em uma mesma partida, o agrupamento por matchId e por groupId, é um artefato interessante para que os registros possam ser analisados em cada contexto de uma partida e grupo, sem misturar com os registros que não pertencem a esse mesmo contexto. Até porque, mesmo que um jogador de um grupo não tenha contribuído muito com o seu time, ele pode terminar muito bem classificado sem sequer ter abatido algum inimigo, veja:

matchId	Id	groupId	kills	killPlace	winPlacePerc
0000a43bce5eec	ccb5a67f210a08	c0f2e306380d3c	0	36	0.8889
0000a43bce5eec	b66a690a2c32f9	6620b219ed2ee2	0	37	0.7778
0000a43bce5eec	a638435c730f4e	6620b219ed2ee2	0	38	0.7778
0000a43bce5eec	c63253392e4fbb	b947dfc5cf2d02	0	39	0.7407
0000a43bce5eec	7a5595685078c5	7bd08592bb25e2	0	40	0.6667
0000a43bce5eec	a121348062f67a	7bd08592bb25e2	0	41	0.6667
0000a43bce5eec	02035450b9c95a	7bd08592bb25e2	0	42	0.6667
0000a43bce5eec	486907e7ef0909	7bd08592bb25e2	0	43	0.6667
0000a43bce5eec	a86e30ee9cc26f	7bd08592bb25e2	0	44	0.6667
0000a43bce5eec	731b2318b2adf2	8ebb76fe544c27	0	45	0.6296

Implementação

Todo fluxo de código está detalhado e disponível no Jupyter Notebook, ficando esse espaço reservado para uma visão mais geral da implementação.

Inicialmente, os dados foram tratados de forma a melhorar a acurácia da predição, eliminando os valores faltantes, reduzindo o espaço em memória ocupado pelos dados, removendo os *outliers*, tratando valores categóricos, criando novos atributos e eliminando atributos que eram o se tornaram menos importantes para o resultado final.

Os registros começaram sendo agrupados por partida, isso porque partidas diferentes podem ter diferentes números de jogadores. Esse agrupamento serviu para sumarizar um pouco do que aconteceu em cada partida, como por exemplo, a quantidade de *kills* total em uma partida e também para melhorar a distribuição dos dados em relação a elas. Outro agrupamento importante é foi o realizado por grupo, para entender melhor o comportamento de cada *player* dentro do seu grupo. Dessa forma, sendo a classificação final do grupo igual para todos os participantes dele, é possível entender qual jogador contribuiu mais para o grupo em questão, e no fim gerar um sumário de como cada grupo da partida chegou ao sua colocação final. Isso mostra que, por mais que o jogador não seja dos melhores, ele pode alcançar uma bela posição final, sendo “carregado” pelos seus companheiros de equipe.

```
'''Grouping by match and groups'''
match = data.groupby('matchId')
group = data.groupby(['matchId', 'groupId', 'matchType'])

agg = list(data.columns)
cols_to_exclude = ['Id', 'matchId', 'groupId', 'matchType', 'maxPlace', 'numGroups', 'winPlacePerc']

for c in cols_to_exclude:
    agg.remove(c)

sum_col = ['kills', 'killPlace', 'damageDealt', 'walkDistance', '_items']

matchData = pd.concat([
    match.size().to_frame('m.players'),
    match[sum_col].sum().rename(columns=lambda s: 'm.sum.' + s),
    match[sum_col].max().rename(columns=lambda s: 'm.max.' + s),
    match[sum_col].mean().rename(columns=lambda s: 'm.mean.' + s)
], axis=1).reset_index()

matchData = pd.merge(matchData, group[sum_col].sum().rename(columns=lambda s: 'sum.' + s).reset_index())
matchData = reduce_mem_usage(matchData)
```

Para avaliar o comportamento de cada um dos modelos escolhidos, verificando o valor do erro médio absoluto, dois *samples* da amostra de treinamento, 1% e 10% do total, foram utilizados. Além disso, foram mostrados os tempos de execução que levaram a escolha do lightGBM Regressor.

O lightGBM foi então refinado, onde encontrou-se os melhores parâmetros para rodar a base de dados disponível.

Refinamento

Todas as mudanças relatadas anteriormente foram feitas para melhorar o poder preditivo do modelo. Primeiro o modelo foi rodado apenas tratando os dados faltantes e reduzindo a memória, porém sem a criação de novas features e remoção de *outliers*. Isso gerou resultados pouco satisfatórios com o erro médio absoluto chegando a ser maior que 0.5.

Realizando todas as tratativas mencionadas, porém sem utilização do GridSearchCV, utilizando os parametros default, o lightGBM já atingiu valores mais próximos do esperado:

```
Training MAE: 0.030
```

Por fim, o GridSearchCV do módulo model_selection do scikit-learn foi a técnica escolhida para chegar na melhor configuração do modelo lightGBM, sendo a seguinte:

```
params={'learning_rate': 0.1,
        'objective': 'mae',
        'metric': 'mae',
        'num_leaves': 31,
        'verbose': 1,
        'random_state': 42,
        'bagging_fraction': 0.7,
        'feature_fraction': 0.7
    }

reg = lgb.LGBMRegressor(**params, n_estimators=10000)
```

Os parametros testados com o GridSearchCV foram o num_estimators, num_leaves, bagging_fraction e feature_fraction nos respectivos valores:

- num_estimators : [100, 1000, 10000]
- num_leaves: [11, 21, 31]
- bagging_fraction: [0.5, 0.7, 1.0]
- feature_fraction: [0.5, 0.7, 1.0]

Sendo assim, o GridSearchCV contribuiu positivamente melhorando em cerca de 0.01 o valor do erro médio absoluto final. Mais valores de cada atributo poderiam ser testados e avaliados, porém esse processo é extremamente custoso e demorado, sendo o valor encontrado suficiente para o presente trabalho.

Resultados

Modelo de avaliação e validação

Por se tratar de uma competição do Kaggle, não é possível rastrear a variação do erro médio absoluto durante a predição no conjunto de testes, pois o conjunto de testes não possui valores para a variável alvo. Com o modelo lightGBM escolhido e devidamente ajustado, foi criado o arquivo csv para submissão que só deveria ter apenas duas colunas, Id e winPlacePerc. Após a submissão o Kaggle retorna o resultado final do erro médio absoluto, comparando os valores reais do conjunto de testes com os valores previstos no arquivo de submissão, que no caso foi de 0.02064 como indicam as imagens abaixo.

The screenshot shows the Kaggle interface for the 'PUBG - Finish Placement Prediction - v3.0' competition. At the top, there are tabs for 'Public', 'Your Work', and 'Favorites'. Below these are filters for 'Outputs', 'Languages', 'Types', and 'Tags', along with a 'Search kernels' bar. The main section displays the competition title, a user profile picture, and the score '0.02064' achieved 8 days ago. Below this, there is a table showing submissions for 'Raphael Alencar'. The table has columns for 'Submission and Description', 'Public Score', and 'Use for Final Score'. The submission 'PUBG - Finish Placement Prediction - v3.0 (version 1/1)' is listed with a public score of 0.02064 and a checkbox for 'Use for Final Score'.

Submission and Description	Public Score	Use for Final Score
PUBG - Finish Placement Prediction - v3.0 (version 1/1) 8 days ago by Raphael Alencar From "PUBG - Finish Placement Prediction - v3.0" Script	0.02064	<input type="checkbox"/>

Rodando o modelo RandomForest com o conjunto de treinamento e sob as mesmas condições o valor obtido foi 0.018, porém, o tempo de processamento foi muito maior:

```
Training time: 3566.11
```

Isto levou a escolha do modelo lightGBM como modelo final, pois ele conseguiu superar o RandomForest sobre as mesmas condições, levando uma extensa vantagem no tempo de execução.

Para mostrar como o modelo está generalizando bem, o valor do erro médio absoluto final do conjunto de treinamento foi de 0.0244, como mostra a figura abaixo, valor este que se aproxima do valor final do MAE para o conjunto de testes.

```
print('Training MAE: {}'.format(mean_absolute_error(Y_train, reg.predict(X_train, num_iteration=reg.best_iteration_))))
```

```
Training MAE: 0.0244452939878564
```

Esse valor de MAE coloca este trabalho entre os 350 trabalhos com melhor score público da competição no Kaggle, onde o melhor valor alcançado foi 0.01385 até o momento que esse documento foi desenvolvido.

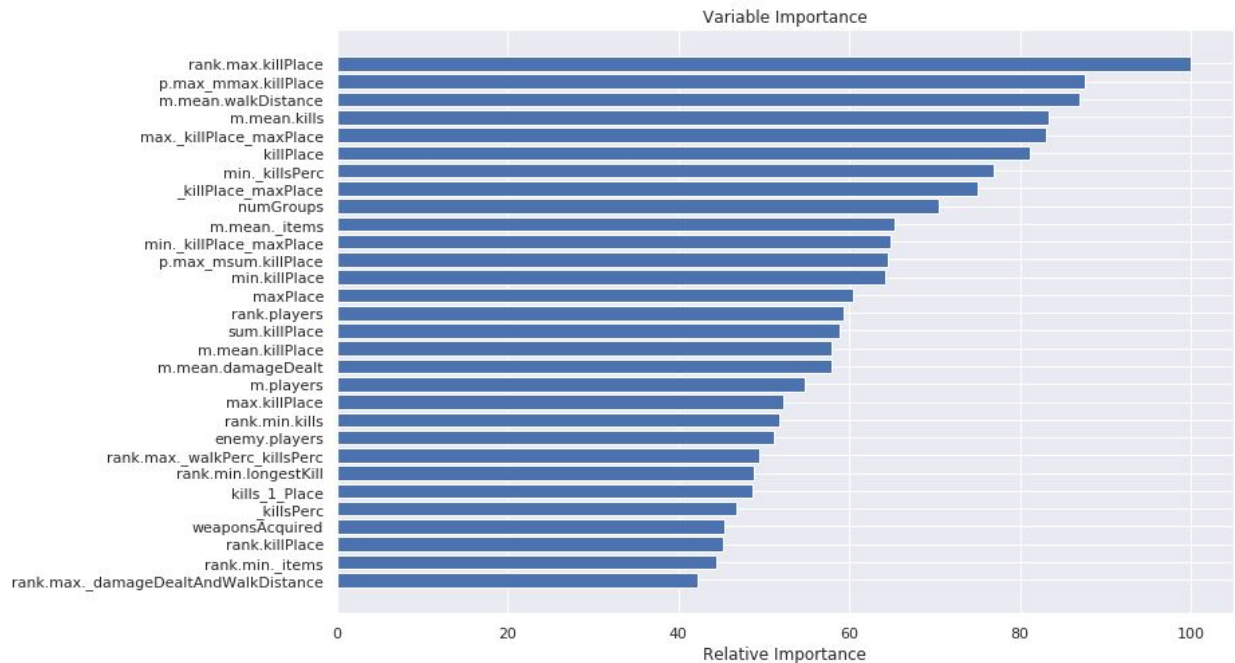
Como foi verificado anteriormente no benchmark, o resultado do modelo final, comparado ao RandomForest, é bem parecido em termos do erro médio absoluto, porém leva muita vantagem por conta do tempo de execução e recurso gasto.

Conclusão

Forma livre de visualização

O presente trabalho conseguiu alcançar bons resultados que foram fruto de uma extensa jornada de preparação e análise dos dados disponibilizados. Por se tratar de uma grande quantidade de dados, mesmo com a redução de memória alcançada com as técnicas utilizadas, foram encontrados problemas com os recursos disponíveis para treinar o modelo e testá-lo, além do enorme tempo despendido para comparar os diversos modelos disponíveis que não foram citados aqui, mas que também foram visitados, visando encontrar a melhor solução.

Ao final do treinamento do modelo que melhor se encaixou aos dados, foi gerado um gráfico mostrando quais seriam as features mais importantes para a predição. Segue o gráfico:



Essa é uma visualização importante para o projeto, pois com ela pode-se reduzir o uso de todos os atributos aos mais importantes e alcançar resultados significativos em menos tempo e com menos recursos.

Melhorias

Agrupar melhor os dados e aprofundar um pouco mais na análise dos dados, podem ser pontos de melhoria para aumentar a acurácia do modelo, algo que poderia ser alcançado com recursos melhores disponíveis. Devido a quantidade de dados, redes neurais (MLPRegressor do scikit-learn) poderiam ser utilizadas para realizar a predição. Além disso, o algoritmo de BaggingRegressor também poderia ser utilizado, pois ele tem a vantagem de ajudar a reduzir a variância e ajudar a evitar o overfitting, porém há uma perda na interpretabilidade do modelo, pode ocorrer problemas de alto bias se não for modelado corretamente e é computacionalmente muito caro.

No mais, o trabalho consegue flutuar no ambiente de machine learning, utilizando desde técnicas de preparação dos dados até a escolha do modelo final, com passos desafiadores que exigiram diversas pesquisas e análises de trabalhos anteriores.

