Utilização de Algoritmos Random Forest e XGBoost para Predição de Resultados Esportivos

Com Base de Dados dos Resultados e Estatísticas por Jogo da Temporada 2018/2019 da Premiere League.

Lucas Peres de Medeiros
Instituto federal Sul-rio-gramdense
Câmpus Pelotas
Pelotas – RS, Brasil
lucasperesm@gmail.com

I. Introdução

Nos últimos anos o mercado de apostas esportivas sofreu uma aceleração que modificou o mercado de investimentos. Por mais que esse mercado não seja uma novidade no mundo a tecnologia proporcionou uma alavancagem de patamar com sites especializados na área, estimando-se, apenas no ano de 2018, uma movimentação de mais de sete bilhões de reais. [1]

Ao contrário do que se pensa, o "jogo de azar' desse mercado pode ser otimizado com estratégias de redução de risco, que abrem espaço para a aplicação de algoritmos de aprendizado de máquina e inteligência artificial.

Nesse artigo apresentaremos o algoritmo "Randon Forest" aplicado com esse intuito, discutindo seus resultados e apresentando uma introdução à um algoritmo mais avançado, o *Extreme Gradient Boosting*, ou apenas *XGBoost*.

II. Introdução à Floresta Aleatória

A Floresta Aleatória – do inglês *Random Forest* – é um algoritmo de aprendizagem de máquina flexível e fácil de usar que produz excelentes resultados mesmo sem ajuste dos chamados hiper parâmetros [2]. É também um dos algoritmos mais utilizados no universo de classificação e "ranqueamento" devido à sua simplicidade e o fato de que pode ser utilizado também para tarefas de regressão.

Como o próprio nome diz o algoritmo cria uma floresta de um modo aleatório. Essa "floresta" é uma combinação (ensemble) de árvores de decisão, na maioria dos casos treinados com o método de *bagging* [3]. Sendo assim, o algoritmo classifica os dados de entrada de acordo com o seu peso e os distribui através de arvores até encontrar o valor final mais adequado para os parâmetros informados.

III. Introdução Ao XGBoost

Talvez o algoritmo que mais ganha espaço no universo da aprendizagem de máquina e da inteligência artificial, o Extreme *Gradient Boosting* é a "última geração" das árvores de decisão. [4]

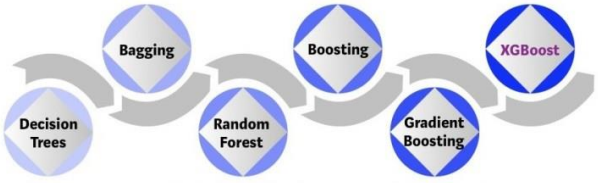

Figura 1: Evolução da Árvore de Decisão até o *XGBoost*. [4]

Apresentado pela Universidade de Washington em 2016 [5], o algoritmo se diferencia dos demais devido às seguintes características:

- Aderente a uma ampla variedade de aplicações: Pode ser usado para resolver problemas de regressão, classificação, ranqueamento, entre outros.

- Portabilidade: Funciona em Windows, Linux e OS X sem problemas de compatibilidade.

- Linguagens de programação: Suporta todas as principais linguagens de programação, incluindo C++, Python, R, Java, Scala e Julia.

- Integração em nuvem: oferece suporte a AWS, Azure, Clusters Yarn e outros.

Enquanto o *Gradient Boosting* puro tem a capacidade de minimizar erro através do gradiente descendente, o *XGBoost* aprimora esse método com a eleição dos melhores hiperparâmetros de forma iterativa e automática em um algoritmo com relativo baixo custo computacional.

IV. A Base de Dados

Antes de se iniciar um trabalho com inteligência artificial é necessário utilizar – ou desenvolver – uma base de dados ou *dataset*. Essa base vai servir para o treinamento e futuros testes dos algoritmos, permitindo a validação do modelo proposto.

Para o objetivo traçado existe uma variedade considerável de material disponível na internet, desde históricos de confrontos específicos de duas equipes, passando por análises de desempenho de jogadores por temporada e indo até análises de ligas específicas. Nesse trabalho utilizamos o *dataset* "*Historical Football Results and Betting Odds Data*" da Premier League 2018/2019 [6], disponível gratuitamente e

que traz resultados e estatísticas de cada confronto do campeonato inglês na temporada, totalizando 106 informações de 380 confrontos.

Como o *dataset* traz informações confronto por confronto foi necessário adaptá-lo, utilizando suas informações para uma nova base de dados.

Foi mensurada a média por equipe até o momento do confronto dos seguintes itens, e nomeados de acordo com a lista abaixo para dados de entrada do algoritmo:

- HT: Equipe mandante do confronto.

AT: Equipe visitante do confronto.

- HPPHPG: Pontos da equipe mandante jogando em casa por jogo – Uma média dos pontos conquistados pela equipe nos jogos anteriores no seu estádio, de zero a três.

- APPAPG: Pontos da equipe visitante jogando fora de casa por jogo – Uma média dos pontos conquistados pela equipe nos jogos anteriores no estádio adversário, de zero a três.

- HPPG: Pontos da equipe mandante independente de estádio por jogo – Quantos pontos a equipe mandante conquistou a cada três até o momento no campeonato.

- APPG: Pontos da equipe visitante independente de estádio por jogo – Quantos pontos a equipe visitante conquistou a cada três até o momento no campeonato.

HGPHPG: Gols por jogo da equipe mandante jogando em casa – média simples.

AGPAPG: Gols por jogo da equipe visitante jogando em estádios adversários – média simples.

HGPG: Gols da equipe mandante por jogo independente de estádio – média simples.

AGPG: Gols da equipe visitante por jogo independente de estádio – média simples.

HSPHPG: Finalizações da equipe mandante jogando em casa por jogo – media simples

HSPHPG: Finalizações da equipe visitante jogando fora de casa por jogo – media simples

HSPG: Finalizações da equipe mandante por jogo independente de estádio – média simples.

ASPG: Finalizações da equipe visitante por jogo independente de estádio – média simples.

HSTPHPG: Finalizações certeiras da equipe mandante jogando no seu estádio por jogo – média simples.

ASTPHPG: Finalizações certeiras da equipe visitante jogando no estádio adversário por jogo – média simples.

HSTPG: Finalizações certeiras da equipe mandante independentemente do local do confronto – média simples.

ASTPG: Finalizações certeiras da equipe visitante independentemente do local do confronto – média simples.

O15HPG: Fração de confrontos onde o time da casa atuou que houve mais de 1,5 gols.

O15APG: Fração de confrontos onde o time visitante atuou que houve mais de 1,5 gols.

O15HPG: Fração de confrontos onde o time da casa atuou que houve mais de 2,5 gols.

O15APG: Fração de confrontos onde o time visitante atuou que houve mais de 2,5 gols.

Além disso, como possíveis dados de saída foram inseridos na base:

– A equipe vencedora do confronto (0 para vitória da casa, 1 para empate e 2 para vitória do time visitante).

- Vitória do time mandante (1 para vitória da casa e 0 para outros resultados.

- Vitória do time visitante (1 para vitória do visitante e 0 para outros resultados.

- Mais de um gol e meio no confronto. (1 para positivo e 0 para negativo).

- Mais de dois gol e meio no confronto. (1 para positivo e 0 para negativo).

Com isso a base de dados original se transformou em um histórico de cada equipe que permitiu analisar ao invés do "nome" das equipes, o que elas fizeram até o momento no campeonato. Partindo dessas informações concentradas em um arquivo .csv pôde-se iniciar o desenvolvimento do algoritmo.

V. DESENVOLVIMENTO DO ALGORITMO DA RANDOM FOREST

O algoritmo da Floresta Aleatória foi desenvolvido em Python sem a utilização de bibliotecas.

Nele foram criadas as funções abaixo com as respectivas funções:

- *crossValidationSplit*: Separa o *dataset* para o método de validação cruzada *"k-fold"*.

- *accuracyMetric*: Calcula a porcentagem da acurácia/ precisão.

- *evaluateAlgorithm*: Avalia o algoritmo através do método de validação cruzada acima.

- *testSplit:* Separa o *dataset* de testes em "parâmetros" e "resultado".

- *giniIndex:* Calcula o index para o split pelo método de Gini.

- *getSplit:* Retorna as posições de split utilizando o método acima.

- *toTerminal*: Retorna o nó terminal.

- *buildTree*: Constrói a Arvore de Decisão.

- *predict*: Realiza uma predição através da árvore e retorna o nó.

- *baggingPredict*: Faz uma predição pelo método de *bagging* e retorna a mesma.

- *randomForest*: Algoritmo da floresta, de acordo com as funções acima.

Com essas funções, o algoritmo criado lê o *dataset* e o separa em n conjuntos para a aplicação do método de

validação cruzada. Os parâmetros da árvore (como *maximum depht* e *minimun size*, por exemplo) são definidos e o algoritmo treina "n-1" dos conjuntos de dados separados pela função *crossValidationSlit* e os testa no conjunto n restante. Em cada árvore e a cada nó os valores de entrada são testados e o algoritmo toma decisões de os "jogar" para a esquerda ou para a direita, conforme a teoria já apresentada.

Para uma melhor compreensão dos métodos aqui apresentados, bem como demais algoritmos e resultados apresentados nos tópicos a seguir, todos os códigos e arquivos utilizados/gerados neste trabalho estão disponíveis no *Github* do autor [7], na sétima referência deste artigo.

VI. Resultados Obtidos com a Random Forest

A. *Casa/Empate/Derrota*

Aplicando a *Randon Forest* no conjunto de dados já citado foi obtido uma precisão de 63,1 % na predição dos resultados. Pela facilidade de implementação do método a seguir validaremos o modelo e encontraremos os pontos positivos e negativos encontrados

VII. Algoritmo XGBoost em Python

O desenvolvimento do algoritmo *XGBoost* também foi feito em *Python* e teve o intuito de comparar os resultados da floresta aleatória. Nele podemos ver um pequeno avanço nas predições. Além disso foram calculadas outras estatísticas dos confrontos para complementar este material, conforme os tópicos abaixo. O algoritmo foi desenvolvido através das bibliotecas *Scikit Learn* [8] e *XGBoost* [5], além de outras para manipulação de tabelas e criação de gráficos como *Matplotlib* [9], *Seaborn* [10] e *Graphviz* [11]. Foi utilizado também o método de validação cruzada *k-fold* com dez subconjuntos.

VIII. Resultados Obtidos com o XGBoost

A. *Casa/Empate/Derrota*

Prevendo três possibilidades de resultado o algoritmo *XGBoost* atingiu 64,74 % de acerto (246/380). Se considerarmos empate como um resultado neutro (time da casa ou empate *versus* time visitante ou empate) a precisão do algoritmo sobe para 86,05 % (327 acertos dos mesmos 380).

Dessa forma, pode-se perceber uma considerável dificuldade do algoritmo para diferenciar os empates dos outros resultados possíveis.

B. *Vitória da casa*

Separando os dados em vitória da casa *versus* empate/derrota o algoritmo foi autocalibrado com os parâmetros:

- colsample_bytree: 0.9943564165441922
- gamma: 0.769957835953689
- learning_rate: 0.10553468874760924
- max_depth: 3
- n_estimators: 52
- subsample: 0.8601756619633539

Com isso, apresentou uma precisão de 125/164 (76,22 %) para o primeiro caso e 160/216 (74,07 %) para o segundo. Com o auxílio das bibliotecas citadas foi possível escalonar a importância das *features* e plota-las para melhor visualização, conforme a imagem abaixo.

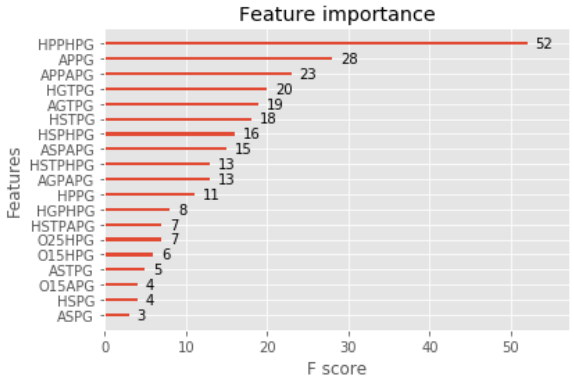

Figura 2: *Feature Importance* do item B

Dessa forma, podemos ver que o elemento mais importante para a classificação foi o número de pontos conquistados pela equipe mandante atuando em casa por jogo, seguido pela média de pontos conquistados pela equipe visitante independente de estádio.

C. *Vitória visitante*

Com os dados de saída separados em vitória visitante *versus* empate/derrota o algoritmo foi autocalibrado com os parâmetros e apresentou uma precisão de 81/109 (74,31 %) para o primeiro caso e 224/271 (82,66 %) para o segundo. Plotando as importâncias encontramos o resultado referido na imagem 3.

- colsample_bytree: 0.879764180288959,
- gamma: 0.34681659006459364,
- learning_rate: 0.25321279289973464,
- max_depth: 11,
- n_estimators: 139,
- subsample: 0.9925296829355851

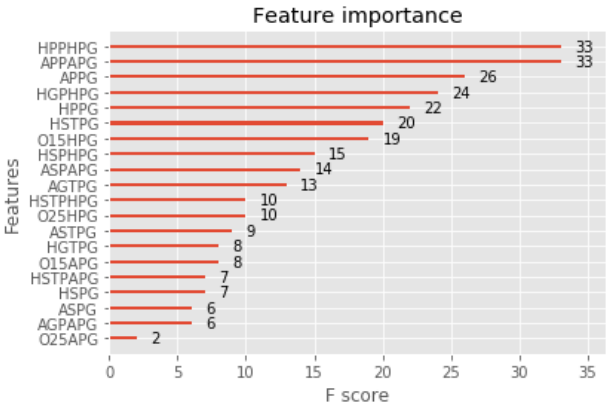

Figura 3: *Feature Importance* do item C

D. *Mais de um gol e meio*

Classificando os resultados naqueles que tiveram dois ou mais gols na partida *versus* um ou menos, com os parâmetros:
- colsample_bytree: 0.97000771555796,
- gamma:0.35956053931056275,
- learning_rate: 0.05862303494712339,
- max_depth: 8,
- n_estimators: 72,
- subsample: 0.9337682505327215

Obtivemos 282/334 (84,43 %) de acerto na primeira hipótese e 24/46 (54,35 %) na segunda. A importância das *features* é mostrada abaixo:

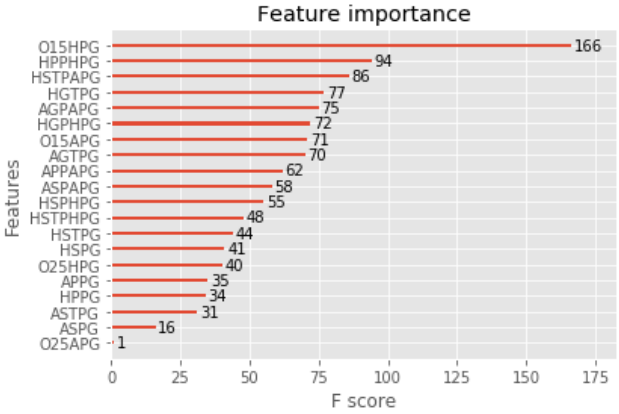

Figura 4: *Feature Importance* do item D

E. *Mais de dois gols e meio*

Nesse item o algoritmo autocalibrou os parâmetros e alcançou 154/204 (75,49 %) de precisão para *over 2.5* e 125/176 (71,59 %) para *under 2.5*.

A importância das *features* e os hiper parâmetros calibrados foram:

- colsample_bytree: 0.9255800778981816,
- gamma:0.06540722555909753,
- learning_rate: 0.29207358723203314,
- max_depth: 14,
- n_estimators: 130,
- subsample: 0.8553755296294407

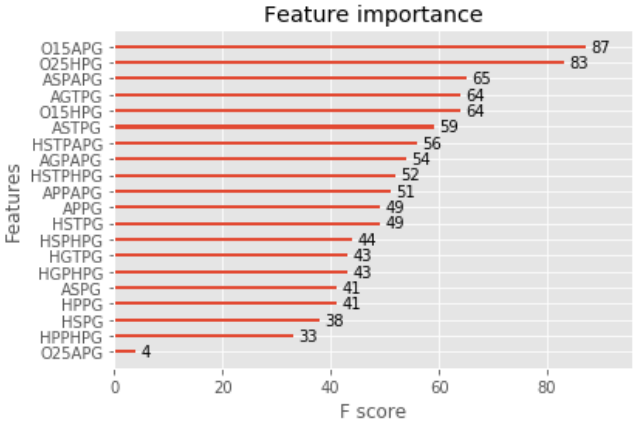

Figura 5: *Feature Importance* do ítem E

IX. Conclusões

Ao final desse trabalho pode-se concluir que a inteligência computacional e o aprendizado de máquina têm muito espaço no mercado de investimentos. Os resultados, para um *dataset* limitado se mostraram interessantes e trouxeram *insights* sobre possibilidades de aprimoramento.

Ao não utilizar o nome das equipes na predição e sim o seu histórico, isso permite a elaboração de um *dataset* maior e mais completo, com informações de diversas ligas. Além disso, acredita-se que os dados iniciais do *dataset* podem ter causado falsos positivos nos resultados pois não havia confrontos anteriores para sua análise. Sendo assim, em um conjunto de dados com maior volume esses em específico poderiam ser facilmente descartados

Uma outra possibilidade é a adição de médias com diferentes intervalos de tempo como parâmetro de entrada para as predições. Por exemplo, os históricos dos últimos três jogos, dos últimos dez e dos últimos quinze em paralelo.

Finalizando, podemos afirmar que estes algoritmos se mostraram muito capazes na maioria dos dados analisados e, com um trabalho intenso em seu aprimoramento podem sim ser aplicados na “vida real”.

REFERÊNCIAS

[1] TERRA. **Mercado de apostas esportivas é tendência no Brasil.** 2019. Disponível em: https://www.terra.com.br/noticias/dino/mercado-de-apostas-esportivas-e-tendencia-no-brasil,cb2c37214a9bcc23f17f8b76a2f8c8209983cshi.html. Acesso em: 15 dez. 2019.

[2] SILVA, Josenildo Costa da. **Aprendendo em uma floresta aleatória.** Machina Sapiens. 2018. Disponível em: https://medium.com/machina-sapiens/o-algoritmo-da-floresta-aleat%C3%B3ria-3545f6babdf8. Acesso em: 14 dez. 2019.

[3] ANICETO, Maria. **Classificadores Ensemble, tipos Bagging e Boosting.** 2017. Disponível em: https://lamfo-unb.github.io/2017/09/27/BaggingVsBoosting/. Acesso em: 15 dez. 2019.

[4] GOMES, Pedro César Tebaldi. **Conheça o algoritmo XGBoost.** 2019. Disponível em: https://www.datageeks.com.br/xgboost/. Acesso em: 03 dez. 2019.

[5] CHEN, Tianqi; GUESTRIN, Carlos. **XGBoost: a scalable tree boosting system.** São Francisco: KDD2016. 2016. Disponível em: https://www.kdd.org/kdd2016/papers/files/rfp0697-chenAemb.pdf. Acesso em: 03 dez. 2019.

[6] FOOTBALL-DATA.CO.UK. **Data files: England.** 2019. Disponível em: http://football-data.co.uk/englandm.php. Acesso em: 03 dez. 2019.

[7] Medeiros, L. P. **RandonForest_XGBoost - Machine Learning**. Github files. Disponível em: https://github.com/lucasgebras/RandonForest_XGBoost---Machine-Learning.git

[8] Pedregosa et al. **Scikit-learn: Machine Learning in Python.** JMLR 12, pp. 2825-2830, 2011.

[9] J. D. Hunter. **Matplotlib: A 2D Graphics Environment.** Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.

[10] M. Waskom et al. **mwaskom/seaborn: v0.8.1 (September 2017).** DOI: 10.5281/zenodo.883859. Disponível em: https://doi.org/10.5281/zenodo.883859

[11] E. R. Gansner, S. C. North. **An open graph visualization system and its applications to software engineering.** Software - Practice And Experience, 2000. Vol 30. Disponível em: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106.5621