Clairvoyance: Predizendo resultados em League of Legends

Joao Francisco B. S. Martins
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
joaofbsm@dcc.ufmg.br

RESUMO

Atualmente League of Legends é o jogo mais jogado do planeta, com mais de 100 milhões de jogadores ativos mensalmente. Além de uma comunidade casual muito grande, o jogo também conta com um cenário competitivo muito desenvolvido, tendo distribuído mais de 5 milhões de dólares em prêmios no último campeonato mundial. Apesar disso, a área de predição de resultados do jogo foi muito pouco explorada, e, portanto, este projeto foi desenvolvido com o objetivo de fornecer à comunidade uma ferramenta de predição em tempo real, nomeada Clairvoyance, que utiliza apenas dados obtidos antes do início de uma partida. Como consequência das análises, também foi possível identificar os fatores mais relevantes para a vitória de um time. Uma previsão precisa pode ser utilizada tanto para apostas no cenário competitivo, como para realização com confiabilidade do chamado *queue dodging*, que é a ação de deixar uma partida antes da mesma se iniciar de fato.

PALAVRAS-CHAVE

aprendizado supervisionado, boosting, bagging, árvores de decisão, league of legends

1 INTRODUÇÃO

League of Legends (LoL) é um jogo do gênero *Multiplayer Online Battle Arena* (MOBA), desenvolvido e publicado pela Riot Games em Outubro de 2009. LoL é atualmente o jogo mais jogado do planeta, contando com mais de 100 milhões de jogadores ativos mensalmente[11] além de ser um grande expoente na comunidade de esportes eletrônicos (*eSports*), oferecendo um *prize pool* total de mais de 5 milhões de dólares no campeonato mundial de 2016. Num contexto mais local, o Campeonato Brasileiro de League of Legends (CBLoL) é uma das maiores competições do gênero no mundo, foi criado em 2012 e teve suas últimas finais protagonizadas em grandes e respeitáveis espaços, como o Ginásio do Ibirapuera, Estádio Allianz Parque e Ginásio do Maracanãzinho. Já foi anunciado que o CBLoL 2017 terá uma premiação total de 200 mil reais[4].

Figura 1: Concept art do mapa Summoner's Rift.

Com uma comunidade tão grande e um cenário profissional que, apesar de novo, já é extremamente relevante, uma ferramenta de predição de resultados pode ser interessante, e até mesmo lucrativa, se usada para prever resultados de jogos competitivos. Além disso, tal ferramenta pode nós dar um discernimento melhor de quais atributos (*features*) são mais relevantes para predição da vitória de um time. Este artigo descreve o design e a construção de Clairvoyance, um preditor de vencedores para partidas de LoL. Seu nome foi inspirado em uma magia de invocadores do jogo, e em português significa clarevidência.

1.1 Visão geral do jogo

No jogo, os jogadores assumem o papel de um invocador, o qual nunca aparece, que controla campeões numa arena de batalha. O objetivo final é destruir uma estrutura chamada Nexus na base do time oponente. LoL possuí diversos modos de jogo e mapas, porém o foco deste trabalho será no modo clássico e no mapa Summoner's Rift, que juntos compõem a maioria das partidas já jogadas. Esse mapa possuí 3 caminhos (*lanes*) pelos quais andam tropas (*minions*) controladas pelo computador. Ao matar uma tropa do time inimigo, o seu

campeão ganha ouro (*gold*), que pode ser usado para comprar itens de diversos tipos, e experiência (XP), que faz com que seu campeão suba de nível, ficando assim mais poderoso. As lanes são: superior (*top*), do meio (*mid*) e inferior (*bottom*). Além das lanes o mapa possuí um rio que o divide diagonalmente e vários campos de monstros espalhados, local chamado de selva (*jungle*). Uma imagem do mapa pode ser vista na figura 1.

Nessa composição de modo e mapa, cada time é um agrupamento de cinco jogadores, cada um controlando seu próprio campeão, os quais se dividem em funções (*roles*) específicas. Existem 5 funções bem definidas: Tanque (*Tank*), *Ability Power Carry* (APC), *Attack Damage Carry* (ADC), Suporte (*Support*), e *Jungler*. Uma descrição breve das funções é a seguinte:

- **Tank**: Joga sozinho (*solo*) no top e apesar de na maioria das vezes ser lento e causar pouco dano, consegue absorver muito dano e tem habilidades de controle de grupo (*crowd control*), sendo essenciais em lutas de time (*teamfights*).
- **APC**: Joga solo no mid e é capaz de dar muito dano, em um curto espaço de tempo, com suas habilidades. Ganha muitos pontos de experiência por estar sozinho e, por causa de sua posição, pode se locomover facilmente entre as lanes para ajudar seus aliados.
- **ADC**: Joga no bot com um suporte, e por conta disso divide a experiência advinda das tropas com ele. Sua força de ataque vem do seu ataque corpo-a-corpo, e por isso o tanto de gold que ele consegue coletar no início da partida é essencial para o desenrolar da mesma. É o campeão que vai ficar mais forte no fim do jogo e literalmente carregar o time para vitória.
- **Support**: Apoia o ADC no bot. Seu objetivo é manter o ADC vivo pela maior quantidade de tempo, mesmo que isso signifique gastar seu próprio gold, ou até sacrificar sua vida. Possuí muitas habilidades de crowd control e tenta dar visão do mapa para o time comprando e posicionando *wards*.
- **Jungler**: Joga sozinho na jungle. O uso da jungle para ganho de gold e XP, processo conhecido como *farm*, é ótimo pois, tanto o campeão no top quanto o campeão na jungle, conseguem ganhar XP integralmente. Assim, nenhum fica para trás na questão de progressão de níveis. Outro motivo para ter um jungler no time é o fato de que ele tem que se movimentar pelo mapa para chegar nos campos de monstros, e, assim pode auxiliar na lane que estiver mais próximo, caso necessário.

Ao longo da partida, essa distribuição de lanes original não necessariamente continuará igual. De fato, o jogo é extremamente dinâmico e os times adequam seu modo de jogo de acordo com a situação na qual se encontram.

Mesmo o jogo tendo sido totalmente traduzido para o português, alguns termos em inglês são preferidos pela comunidade, e portanto, o restante deste trabalho utilizará uma mescla das duas nomenclaturas.

2 TRABALHOS RELACIONADOS

Com uma comunidade tão grande e por consequência uma quantidade de dados massiva gerada pelos jogos, League of Legends é um prato cheio para os mais diversos tipos de análise de dados. Existem diversos sites que utilizam técnicas de *data science* para mapear quais os melhores heróis, melhores itens, melhor escolha de habilidades (*builds*), etc. Entre os mais populares temos LoLKing, Elophant e Champions GG. No entanto, não existe nenhuma grande iniciativa ligada a predição de resultados com o uso de aprendizado de máquina, nenhuma publicação científica e apenas alguns poucos trabalhos na área[2, 5, 10, 12].

3 COLETA DOS DADOS

Inicialmente a proposta era utilizar dados de uma, então recém publicada, base de dados (*dataset*) na plataforma Kaggle[1]. Essa base contém 3803 partidas do cenário competitivo de LoL dos anos 2015, 2016 e 2017. Porém, quando o projeto começou a ser implementado, logo se viu que essa quantidade de instâncias seria demasiadamente pequena para um treino significativo com muito atributos. Sabendo que a Riot Games disponibiliza uma API muito bem estruturada, foi decidido que seriam coletadas mais partidas através de um wrapper da mesma, escrito em Python, chamado Cassiopeia.

Para que o processo de coleta seja entendido melhor, é necessário que expliquemos algumas coisas com relação ao sistema de partidas do jogo. Em League of Legends, novos invocadores começam no nível 1. A medida que eles jogam, vão subindo de nível até atingirem o nível máximo 30, quando então abre-se a opção de jogar partidas ranqueadas, caso o jogador tenha comprado pelo menos 20 campeões na loja do jogo. Partidas ranqueadas são uma alternativa competitiva para partidas comuns, com o formato de escolha de campeões por turnos (*picks*), onde cada time bane 3, totalizando 6 campeões banidos (*bans*). Neste formato um mesmo herói só pode ser escolhido uma vez por partida, e não por time como é o caso nas partidas comuns.

As partidas ranqueadas introduzem o jogador ao sistema de ligas. O sistema de ligas é extremamente complexo, então a explicação que se segue é básica, envolvendo apenas os componentes necessários para o entendimento dos dados. No sistema existem 7 níveis (*tiers*), que em ordem crescente são: Bronze, Prata, Ouro, Platina, Diamante, Mestre e Desafiante. Os 5 primeiros contam com 5 divisões pelas quais os jogadores tem que progredir até chegarem na fronteira entre

[1] https://www.kaggle.com/chuckephron/leagueoflegends

dois tiers, quando então podem passar por uma sequência de partidas para tentar alcançar a promoção para um tier superior. Já os tiers Mestre e Desafiante não possuem divisões, sendo que o primeiro não tem limite de jogadores enquanto o segundo só comporta os 200 melhores de cada região.

As primeiras dez partidas de um jogador servem para medir seu nível de habilidade e posicioná-lo inicialmente num tier concordante. O ranqueamento dos jogadores se dá baseado em seus Pontos de Liga (PdL), obtidos quando vitoriosos em uma partida ranqueada e perdidos quando derrotados. Se um jogador é desconectado durante uma partida ele perde PdL independente do resultado, porém, caso esse jogador saia da partida na tela de carregamento da mesma, ou seja, antes do jogo começar, não perda de PdL mas sim uma penalização em forma de tempo de espera para entrar em outro jogo. Quando essa prática é intencional, ela é apelidada de *Queue Dodging* e é muito mal vista pela comunidade.

No dia 13 de Julho de 2010, quando o sistema de ligas foi introduzido, se iniciou a primeira temporada (*season*) de partidas ranqueadas. Com aproximadamente 10 meses de duração, ao serem iniciadas, as temporadas fazem um *soft reset* dos rankings de cada invocador, reduzindo seus PdL de acordo com sua colocação na última temporada. A temporada corrente é a sétima, que começou no dia 7 de Dezembro de 2016 e é esperado que ela termine no dia 7 de Novembro de 2017.

3.1 API e Cassiopeia

Para garantir acesso à API do LoL e ser possível realizar requisições por dados, foi necessário o cadastramento de uma conta no site de desenvolvedores[2] da Riot para que assim uma chave de desenvolvedor fosse obtida. Essa chave permite que um desenvolvedor faça até 10 requisições por segundo e 500 requisições por minuto.

Foi desenvolvido um script coletor de dados que usa o wrapper Cassiopeia. Esse wrapper não só oferece uma interface de alto nível para que sejam feitas as requisições mas também faz todo o *bookkeeping* dos dados utilizando a biblioteca SQLAlchemy para fazer um mapeamento objeto-relacional dos mesmos em um banco de dados MySQL. Com os dados pré-organizados, foi possível focar mais na criação dos modelos, que é a parte que realmente importa neste projeto.

Para que os dados representassem tendências mais gerais, foram coletados dados de 4 regiões diferentes: América do Norte (NA), Brasil (BR), Europa Ocidental (EUW) e Coreia do Sul (KR). Ao todo 40.228 partidas, todas da temporada atual, foram salvas no banco de dados. A tabela 1 mostra a distribuição de partidas por região.

[2]https://developer.riotgames.com/

Região	Nºde Partidas
América do Norte	9946
Brasil	7244
Europa Ocidental	12985
Coreia do Sul	10034

Tabela 1: Partidas coletadas por região.

O sistema de coleta usado opera em teia, inicialmente pegando todos os invocadores presentes no tier Mestre e guardando-os em uma estrutura de fila para, logo em seguida, usá-los como *seed*. Há, em média, 500 jogadores no tier Mestre por região. Ao retirarmos um invocador da fila, olhamos todo o seu histórico de partidas e coletamos uma por uma. Quando algum dos invocadores na partida não tiver sido visto ainda, o adicionamos à fila para que num futuro próximo suas partidas sejam também coletadas. Dessa forma são necessários apenas poucos jogadores iniciais para que seja coletada uma base de dados praticamente sem restrições de tamanho.

Uma outra vantagem do uso da coleta em teia é o fato de que os invocadores que participam dessas partidas tendem a estar por volta do mesmo nível dos iniciais (Mestre), criando assim uma base de jogadores experientes, o que é necessário para se fazerem predições concisas e sem muita influência da sorte. A distribuição de jogadores por liga pode ser vista na figura 2, onde os invocadores não ranqueados são aqueles que ainda estão em processo de ranqueamento inicial. No total foram coletados 38.181 invocadores diferentes

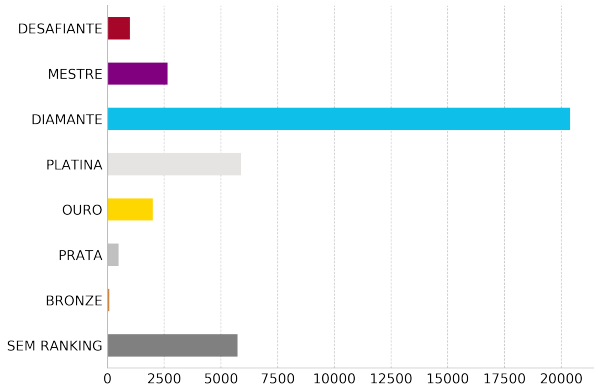

Figura 2: Distribuição de jogadores por tier.

4 METODOLOGIA

A proposta inicial compreendia apenas a predição usando dados do pré-jogo (*pregame*) e esse se mantém ainda como o foco do trabalho. No entanto, pela riqueza dos dados coletados, foram criadas bases de dados também com informações

obtidas no decorrer do jogo (*in-game*) para comparações e análises.

A predição dos resultados é um problema de classificação de instâncias. Sendo assim, definimos o atributo de saída binário como:

$$y_i = \begin{cases} 1, \text{se o time azul for o vencedor} \\ 0, \text{se o time vermelho for o vencedor} \end{cases}$$

É um fato conhecido na comunidade do LoL que o time azul possui uma probabilidade de vencer um pouco maior. Ninguém sabe ao certo o que dita isso, mas muitos especulam que é o fato de a primeira escolha de campeão ser sempre do time azul, dando a ele assim a oportunidade de guiar o restante dos picks. Já outros dizem que, o fato de o time vermelho ficar no canto direito superior do mapa e ter que avançar para o canto esquerdo inferior, causa um atraso cognitivo por não estarmos acostumados com esse tipo de associação. Nicholas Ver Halen fez uma análise gráfica em cima do dataset competitivo do Kaggle tentando encontrar alguma explicação lógica para o fato[8]. Os achados são inconclusivos mas lá também podem ser encontradas outras teorias para explicar a superioridade azul.

A comparação de vitórias entre os times no dataset coletado pode ser encontrada na tabela 2.

Time	Vitórias	Porcentagem
Azul	20381	50.66%
Vermelho	19847	49.34%

Tabela 2: Porcentagem de vitórias por time.

4.1 Seleção de atributos

No jogo existem diversas informações presentes logo antes do início de uma partida e que podem influenciar em seu resultado, como a árvore de talentos de cada invocador, os feitiços e runas escolhidos por ele, sua maestria com o campeão escolhido, etc. Por esse motivo, League of Legends foi a escolha ideal de jogo para realização desse projeto, uma vez que outros MOBAs famosos, como DotA por exemplo, não seriam tão bons por não possuírem tais customizações.

Um módulo foi definido especificamente para tratar a massiva base de dados coletada e extrair os atributos de acordo com a escolha do usuário. Para isso foi feito o uso de estruturas da biblioteca de manipulação de dados **pandas**, com o auxílio de consultas SQL.

4.1.1 Atributos pregame. Os atributos pregame coletados e utilizados em modelos de base de dados foram:

- Campeões escolhidos por cada time.
- Feitiços escolhidos por cada invocador.
- Árvore de talentos de cada invocador.
- Tipos e capacidade de dano infrigidos pelos campeões escolhidos.
- Maestria de cada invocador com o campeão escolhido.
- Soma das maestrias com todos os seus campeões para cada invocador no time.
- Porcentagem e número total de vitórias para cada invocador.
- Porcentagem e número total de vitórias para cada invocador com o campeão escolhido na partida corrente.

Como a informação de campeões, feitiços e talentos escolhidos é categórica, foi necessário o uso de *one-hot encoding* para darmos um significado conciso das escolhas para os modelos de aprendizado. Assim, para cada time, cada um dos 136 possíveis campeões se tornou um atributo binário que indica se aquele campeão foi escolhido pelo time, **1**, ou se não foi, **0**, totalizando 272 novos atributos. A forma como o one-hot foi usado para feitiços e talentos se diferenciou um pouco, pois, com 9 possíveis feitiços escolhidos aos pares e 45 possíveis talentos, o número de atributos seria demasiadamente alto por invocador. Portanto, se optou por uma soma das ocorrências de cada um por time, esperando que uma distribuição ótima dessas características fosse o suficiente para melhorar a predição do vencedor. O uso de runas como um atributo preditivo, seguindo o mesmo formato, foi descartado devido aos baixos resultados para os atributos citados.

Os danos podem ser físico ou mágico e a capacidade de cada campeão infringi-los depende diretamente da função do mesmo no time. Um ADC por exemplo terá um alto dano físico e um baixo dano mágico em geral. Esses atributos são modelados como a soma dos tipos de dano por time e a porcentagem de cada tipo, também por time, e foram escolhidos pois, um time que foca muito em certo tipo de dano, pode ser facilmente anulado (counterado) com uma estratégia que aumenta a defesa contra este.

Uma das características interessantes no LoL é que a medida que um jogador ganha e tem um bom desempenho em uma partida, ele é recompensado com a progressão de um nível extra-jogo, denominado nível de maestria. Níveis de maestria tem recompensas apenas cosméticas, mas podem ser considerados um indicador da habilidade daquele usuário com o campeão em questão. Assim, esses dados foram usados de duas formas: A soma das maestrias de todos os campeões para todos os invocadores por time (e também a diferença delas entre os times) e a soma, por time, das maestrias dos invocadores apenas com os campões escolhidos por eles naquela partida. Dessa forma é pretendido analisar a habilidade geral dos jogadores, assim como sua habilidade no contexto da partida atual, utilizando de marcadores fáceis e rápidos de serem coletados através da API, o que poderia facilmente ser

feito antes da tela de carregamento da partida sumir e o jogo se iniciar de fato.

Por fim foi avaliado o histórico dos jogadores, tanto no geral como com os campeões escolhidos na partida corrente. Esses dados foram agrupados somando o número de vitórias por time, a taxa dessas vitórias (*win rate*), obtida através de uma média harmônica das taxas individuais, e a diferença entre os times. Os mesmos agrupamentos foram feitos com os dados apenas dos campeões correntes. O objetivo aqui é o mesmo que com os níveis de maestria, porém o histórico, além de ser incompleto pela forma de coleta dos dados, também é um dado custoso de se obter, tanto em tempo de processamento quanto em número de requisições na API.

4.1.2 Atributos ingame. Como já dito anteriormente, os atributos ingame não foram o foco e portanto estão em menor número. Eles foram considerados em intervalos de tempo separados de 10 em 10 minutos, para que fosse possível saber a precisão do modelo de predição a medida que o jogo transcorre. São eles:

- Quantidade de *creeps* mortos por minuto por jogador. Creeps são as criaturas controladas pelo computador: tropas e monstros da selva.
- Quantidade de dano recebido por minuto por jogador.
- Ganho de ouro por minuto por jogador.
- Ganho de experiência por minuto por jogador.

Os dados são agrupados dos 0 aos 10 minutos, 0 aos 20 minutos, 0 aos 30 minutos e 0 minutos até o fim do jogo, dependendo da preferência escolhida no seletor de atributos. Todos eles foram somados nos intervalos e por time, sendo também usada a diferença dos mesmos entre os times.

Além dos citados acima, existem diversos outros marcadores que poderiam ser utilizados, como o número de torres e inibidores destruídos por cada time a cada intervalo, o número de monstros especiais (Dragão e Barão) aniquilados e a quantidade de mortes em sequência (*killing sprees*) cada jogador causou no time inimigo, apenas para citar alguns. No entanto, pela forma como a coleta começou a ser feita, muitos desses dados foram armazenados apenas como valores totais e não por períodos de tempo. De qualquer forma, eles provavelmente não teriam um peso muito grande nas predições gerais, por serem dados fracos: muitas vezes iguais entre times e com pouca variância em escala tão grande.

4.2 Modelos de aprendizado

Devido à natureza tabular das bases de dados criadas, o uso de modelos baseados em árvores de decisão é de longe a melhor escolha. Como árvores de decisão não são influenciadas pela falta de *feature scaling* e normalização, os mesmos datasets criados podem ser utilizados em todos os modelos, sem perda de correlação dos resultados.

Foram escolhidos dois métodos de boosting e um de bagging. O processo de boosting é baseado preditores fracos (alto viés, baixa variância) enquanto que o processo de bagging é baseado em preditores fortes (baixo viés, alta variância), como árvores de decisão totalmente crescidas. Assim, é como se os modelos se complementassem, oferecendo uma análise mais completa. Todos os modelos usam conjuntos de árvores de decisão (*tree ensembles*).

4.2.1 AdaBoost. Primeiro algoritmo de boost a surgir, criado por Freund e Schapire em 1997[6], e até hoje amplamente utilizado por sua simplicidade e eficiência. Combina diversos preditores fracos em um único preditor forte, denominado *ensemble*, buscando diminuir o erro principalmente pela diminuição do viés desse preditores. Os preditores fracos podem ser qualquer modelo de aprendizado, no entanto usaremos árvores de decisão por motivos já citados.

A hipótese H do ensemble para uma instância x é dada pela seguinte fórmula:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Onde α_t é o peso da hipótese h_t para instância x gerada pelo melhor preditor fraco da iteração t. O peso do preditor é dado por:

$$\alpha_t = \frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

Onde ϵ_t é o erro desse preditor no dataset de treino.

4.2.2 XGBoost. Desenvolvido por Tianqi Chen e Carlos Guestrin[3], o algoritmo *eXtreme Gradient Boosting* (XGBoost) é extremamente rápido, por ter otimização de sistemas como um de seus conceitos básicos. Ele também é eficiente em predições e, como o próprio nome diz, é baseado no conceito fundamental de Gradiente Boosting, introduzido por Friedman[7].

Assim como todo algoritmo de aprendizado de máquina, ele tenta otimizar uma função objetivo:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

Onde l é uma função de perda qualquer, Ω é o termo de regularização, muito importante para controlar a complexidade do modelo, y_i é a saída esperada para instância i de um total n, e $\hat{y}_i^{(t)}$ é o preditor fraco t do ensemble, dado por:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

Onde K é o número de árvores, f é uma função no espaço funcional $\mathcal{F}$, e $\mathcal{F}$ é o conjunto que contém todas as árvores de classificação e regressão (CARTs) possíveis.

Atributos por base de dados	AdaBoost	XGBoost	Random Forests
Campeões por time	52.41% (0.46%)	**52.89% (0.30%)**	51.11% (0.63%)
Feitiços de invocador por time	**51.24% (0.37%)**	51.14% (0.23%)	50.61% (0.41%)
Talentos de invocador por time	52.37% (0.48%)	**52.53% (0.15%)**	50.61% (0.43%)
Tipos/capacidade de dano por campeão	51.01% (0.38%)	**51.39% (0.25%)**	50.31% (0.55%)
Total, por time, de maestrias de campeão por invocador	**51.38% (0.30%)**	51.37% (0.33%)	50.29% (0.68%)
Total, por time, de maestrias de invocador com campeão escolhido	**58.78% (0.72%)**	58.63% (0.62%)	54.68% (0.58%)
Taxa e total de vitórias por invocador	66.46% (2.52%)	**66.91% (2.59%)**	62.41% (2.53%)
Taxa e total de vitórias para campeão escolhido por invocador	88.88% (1.97%)	**88.96% (1.92%)**	87.43% (2.21%)
Acurácia média	59.07%	**59.23%**	57.18%

Tabela 3: Acurácia de predição de vitória com dados pregame.

4.2.3 Random Forests. O modelo Random Forests, descrito por Samuel Breiman[1], é baseado no método de *Bootstrap Aggregation* (*bagging*) e, ao aplicá-lo ao dataset, visa diminuir a variância de um grande conjunto de árvores complexas, para assim diminuir o erro da predição. A floresta (de árvores de decisão) faz uma média da predição de m árvores cada qual com funções de peso W_j. A fórmula para a predição da floresta é:

$$\hat{y} = \sum_{i=1}^{n} \left(\frac{1}{m} \sum_{j=1}^{m} W_j(x_i, x') \right)$$

5 ANÁLISE EXPERIMENTAL

Pela extensa quantidade de atributos de pregame, foram criados modelos separados para cada um deles, de forma a medirmos a importância de cada conjunto de atributos na predição. Foi utilizada uma validação cruzada com 5 partições (*5-fold cross-validation*) em conjunção com os modelos de predição para obtermos uma acurácia mais próxima da real. A escolha de 5 partições foi empírica, baseada em testes com 5, 10 e 20 partições. A tabela 3 apresenta os resultados obtidos. No geral, o modelo XGBoost foi o melhor e o modelo Random Forests o pior, ficando bem atrás do AdaBoost.

A acurácia obtida foi bem inferior à esperada para todos os casos exceto para o histórico de campeões escolhidos. A comunidade de DotA, que tem bem mais publicações relativas a predição de resultados, já conseguiu obter 73% apenas com a escolha dos heróis[9], que são equivalentes aos campeões de LoL. As muitas combinações advindas do one-hot encoding dos campeões explica o grande erro da predição para o primeiro caso: Considerando que 6 campeões são banidos e que não podemos ter dois do mesmo campeão por partida, temos que cada lado pode ter 5 entre 125 campeões, totalizando $\binom{250}{10} = 2.2 \times 10^{17}$, ou seja 13 ordens de grandeza maior do que o número de partidas coletadas. Por esse motivo nem foi testado um modelo onde o campeão escolhido por cada invocador era representado em one-hot encoding, o que totalizaria 1360 atributos, com certeza dando uma predição pior.

O uso do histórico dos campeões foi muito além das expectativas, provavelmente devido à incompletude dos dados com relação ao histórico total dos jogadores. No entanto o uso desses dados seria inviável para predições em tempo real devido ao custo computacional elevado de coleta e préprocessamento.

Foram então criados dois datasets principais, um com todos os atributos e outro com todos os atributos exceto o histórico, para que fosse possível ver a acurácia da conjunção dos modelos.

5.1 Ajuste de hiperparâmetros

Com ambas as novas bases de dados criadas foi feito um ajuste de hiperparâmetros para os modelos de aprendizado através da execução de um *gridsearch*. Os melhores valores para cada hiperparâmetro e modelo por dataset foram:

- AdaBoost:
  - **Sem histórico**: 50 estimadores com taxa de aprendizado 1.
  - **Todos os atributos**: 300 estimadores com taxa de aprendizado 0.1.
- XGBoost:
  - **Sem histórico**: 300 estimadores, taxa de aprendizado 0.1 e profundidade máximas das árvores igual a 2.
  - **Todos os atributos**: Os parâmetros se mantiveram os mesmos.
- Random Forests:
  - **Sem histórico**: 100 estimadores com profundidade máximas das árvores igual a 2.
  - **Todos os atributos**: Os parâmetros se mantiveram os mesmos.

Os novos parâmetros melhoram a acurácia dos modelos em todos os casos. A baixa taxa de aprendizado (*learning rate*) encontrado para o XGBoost encontrado previne o overfitting.

Dataset	AdaBoost	XGBoost	Random Forests
Exceto histórico(pré-ajuste)	59.44%	**59.77%**	54.58%
Exceto histórico(ajuste)	60.62%	61.42%	**64.80%**
Todos os atributos(pré-ajuste)	89.08%	**89.29%**	87.68%
Todos os atributos(ajuste)	89.41%	**89.50%**	89.10%

Tabela 4: Acurácia para modelos agregados e ajuste de hiperparâmetros.

A tabela 4 apresenta as acurácias obtidas para os datasets com e sem o ajuste dos hiperparâmetros. É interessante notar o ganho do Random Forests no ajuste, aumentando sua acurácia em 10% e se tornando o melhor preditor disparado, uma situação completamente contrária às tendências observadas.

5.2 Pregame vs. in-game

Os resultados para os dados in-game foram dentro do esperado, não chegando nem perto de 100%, algo que realmente era o esperado, uma vez que a certeza de vitória em MOBAs, mesmo para um jogo que parece definido, ainda sim não é garantida. Para fins de comparação, a tabela 5 apresenta as acurácias de predição para atributos in-game. Os resultados com o uso dos históricos de campeões escolhidos são melhores que as predições para dados de até 30 minutos de jogo, o que mostra o poder de predição elevadíssimo do atributo.

Intervalos de tempo	AdaBoost	XGBoost	Random Forests
0 a 10 minutos	71.80% (0.82%)	**71.93%** (0.98%)	71.89% (1.01%)
0 a 20 minutos	81.95% (0.75%)	**82.03%** (0.79%)	81.85% (0.69%)
0 a 30 minutos	87.98% (0.50%)	**88.09%** (0.47%)	87.88% (0.47%)
0 até o fim do jogo	90.37% (0.51%)	**90.57%** (0.46%)	90.35% (0.51%)

Tabela 5: Acurácia para modelos de dados in-game.

5.3 Importância dos atributos

Por fim, analisamos a importância dos atributos nas predições. Os pesos do modelo XGBoost foram escolhidos por este ter se mostrado o mais robusto e eficiente nas predições ao longo do projeto. Uma relação desses valores pode ser vista na figura 3.

A diferença dos win rates por time foi disparadamente a feature mais importante, com peso mais do que três vezes maior que a segunda colocada. As features que se seguem são todas ou do histórico por campeões, ou do histórico total dos invocadores ou das maestrias por campeão, o que faz bastante sentido, uma vez que esses foram os atributos com maior acurácia, e portanto maior poder de predição.

6 CONCLUSÃO

Neste trabalho descrevemos a criação do preditor de resultados Clairvoyance, o qual utiliza apenas dados pregame

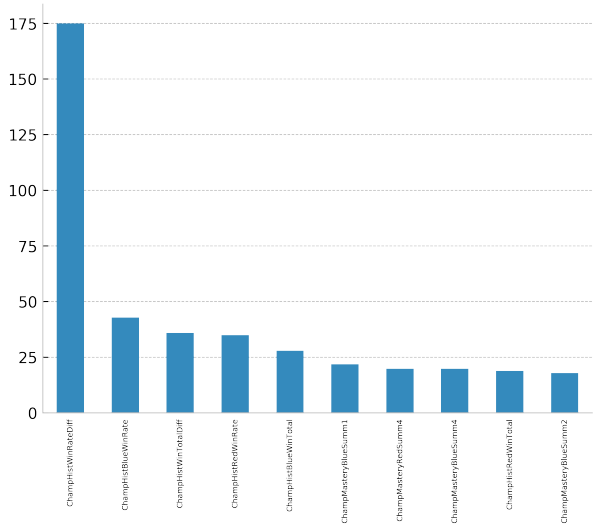

Figura 3: 10 atributos mais importantes para o XGBoost.

para inferir o time vencedor em uma partida de League of Legends. Foram usados 3 modelos preditores baseados em árvores de decisão, sendo dois que operam com boosting e um que realiza bagging. A acurácia dos modelos em cima dos datasets foi feita utilizando 5-fold cross-validation e medida de erro simples. Foi proposta uma combinação de atributos que pode ser obtida em tempo real, ou seja, antes do início de um jogo, para predizer com precisão de 64.80% o resultado do mesmo, ao utilizar a maestria de cada invocador com o seu campeão de escolha como atributo essencial dessa combinação. O uso das features de histórico retornou uma acurácia incrivelmente grande, maior do que o de dados in-game até 30 minutos, porém se mostrou inviável de obter em tempo real.

Para que o processo de predição possa ser realmente utilizado em um tempo viável, uma interface será criada para automatizar a coleta dos dados para os jogadores e campeões presentes em uma partida, da qual o usuário participará.

Também serão coletados dados de jogos profissionais para uma análise das tendências, do mercado de apostas e de uma possível correlação de jogos semi-competitivos (ranqueados) com jogos competitivos.

AGRADECIMENTOS

Gostaria de agradecer ao professor Adriano Veloso, do Departamento de Ciência da Computação da Universidade Federal de Minas Gerais, pelo ótimo conteúdo dado ao longo do curso de Aprendizado de Máquina, o qual foi material base para o desenvolvimento desse projeto, e também por todas as dicas e conselhos dados ao longo do semestre.

REFERÊNCIAS

[1] Leo Breiman. 2001. Random Forests. *Mach. Learn.* 45, 1 (Oct. 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[2] Matías Cabrera. 2016. Predicting the winner of a League of Legends match at the 10-minute mark. (2016). https://www.dropbox.com/s/hb9t1nfc0uhb9ge/Predicting%20the%20Winner%20of%20a%20League%20of%20Legends%20Match.pdf?dl=0 Acessado em 3 de Junho, 2017.

[3] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *KDD 2016*.

[4] ESPN. 2017. 'League of Legends': CBLoL 2017 manterá R$ 200 mil como premiação. (2017). http://espn.uol.com.br/noticia/665415_league-of-legends-cblol-2017-mantera-r-200-mil-como-premiacao Acessado em 3 de Junho, 2017.

[5] Thomas Huang et al. 2015. League of Legends Predictor. (2015). http://thomasythuang.github.io/League-Predictor Acessado em 29 de Maio, 2017.

[6] Yoav Freund and Robert E Schapire. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. System Sci.* 55, 1 (1997), 119 – 139. https://doi.org/10.1006/jcss.1997.1504

[7] Jerome H. Friedman. 2000. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics* 29 (2000), 1189–1232.

[8] Nicholas Ver Halen. 2017. Why is blue a better? (2017). https://www.kaggle.com/verhalenn/why-is-blue-a-better Acessado em 5 de Julho, 2017.

[9] Nicholas Kinkade and Kyung Lim. 2015. DOTA 2 Win Prediction. (2015). https://cseweb.ucsd.edu/~jmcauley/cse255/reports/fa15/018.pdf Acessado em 25 de Junho, 2017.

[10] Lucas Lin. 2016. League of Legends Match Outcome Prediction. (2016). http://cs229.stanford.edu/proj2016/report/Lin-LeagueOfLegendsMatchOutcomePrediction-report.pdf Acessado em 17 de Abril, 2017.

[11] Paul Tassi. 2016. Riot Games Reveals 'League of Legends' Has 100 Million Monthly Players. (2016). https://www.forbes.com/sites/insertcoin/2016/09/13/riot-games-reveals-league-of-legends-has-100-million-monthly-players Acessado em 3 de Junho, 2017.

[12] Keith Trnka. 2015. Predicting League match outcomes: First week of machine learning. (2015). https://kwtrnka.wordpress.com/2015/08/25/predicting-league-match-outcomes-first-week-of-machine-learning Acessado em 29 de Maio, 2017.