Universidade Federal
do Rio de Janeiro
Escola Politécnica

REALRANK: UM ARCABOUÇO PARA RANQUEAMENTO DE REPUTAÇÃO DE USUÁRIOS EM REDES SOCIAIS ONLINE

Caio César Riqueza Ramos

Projeto de Graduação apresentado ao Curso de Engenharia de Computação e Informação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Daniel Ratton Figueiredo

Rio de Janeiro
Março de 2019

REALRANK: UM ARCABOUÇO PARA RANQUEAMENTO DE REPUTAÇÃO DE USUÁRIOS EM REDES SOCIAIS ONLINE

Caio César Riqueza Ramos

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE COMPUTAÇÃO E INFORMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE COMPUTAÇÃO.

Examinado por:

Prof. Daniel Ratton Figueiredo, Dr.

Prof. Alexandre Gonçalves Evsukoff, Dr.

Prof. Heraldo Luís Silveira de Almeida, Dr.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2019

César Riqueza Ramos, Caio

RealRank: Um arcabouço para ranqueamento de reputação de usuários em redes sociais online/Caio César Riqueza Ramos. – Rio de Janeiro: UFRJ/ Escola Politécnica, 2019.

XI, 54 p.: il.; 29, 7cm.

Orientador: Daniel Ratton Figueiredo

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia de Computação e Informação, 2019.

Referências Bibliográficas: p. 52 – 54.

1. Redes Sociais. 2. Network Embedding. 3. Fake. 4. Sybil. 5. Detecção de Fakes. 6. Redes Complexas. I. Ratton Figueiredo, Daniel. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Computação e Informação. III. Título.

Agradecimentos

Gostaria de agradecer aos meus pais por conseguirem me dar uma boa educação e formação, apesar de todas as dificuldades socioeconômicas. Ao meu irmão, por me ajudar e apoiar em importantes e difíceis decisões acadêmicas e profissionais.

Agradeço aos amigos e colegas de curso que me ajudaram e motivaram nos momentos necessários.

Agradeço a todos os professores que tive durante o curso. Em especial ao meu orientador Daniel Figueiredo, que foi um dos meus grandes motivadores para alcançar essa formação. Sua dedicação e forma de ensinar foram e tem sido essenciais na motivação pela minha constante busca pelo conhecimento na área da computação.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Computação.

REALRANK: UM ARCABOUÇO PARA RANQUEAMENTO DE REPUTAÇÃO DE USUÁRIOS EM REDES SOCIAIS ONLINE

Caio César Riqueza Ramos

Março/2019

Orientador: Daniel Ratton Figueiredo

Curso: Engenharia de Computação e Informação

Com a continuada e crescente popularização da Internet, o número de redes sociais online vem se multiplicando assim como o número de usuários destes sistemas, inclusive usuários maliciosos, que exploram tais sistemas para obter algum tipo de vantagem. Conhecidos por "contas *fakes*", estes usuários criam amizades e trocam mensagens com usuários comuns, que desconhecem sua intenção ou reputação. Desta forma, um atual desafio para redes sociais online é a identificação de contas *fake*. Neste sentido, este projeto propõe o RealRank: um arcabouço flexível que realiza um ranqueamento da reputação de usuários em uma rede social online. RealRank se baseia em algoritmos de aprendizado supervisionado para classificação de reputação e algoritmos de passeio aleatório para a propagação de reputação pela rede social. Ambos algoritmos utilizam apenas a similaridade estrutural hierárquica e a relação entre os usuários da rede social. O algoritmo é avaliado na rede social online da plataforma Steam e comparado com outras abordagens que utilizam mais informação dos usuários.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Engineer.

REALRANK: A FRAMEWORK FOR USER REPUTATION RANKING IN ONLINE SOCIAL NETWORKS

Caio César Riqueza Ramos

March/2019

Advisor: Daniel Ratton Figueiredo

Course: Computer Engineering

With the continued Internet's popularization, the number of online social networks has been increasing as well as the number of users of these systems, including malicious users, who exploit such systems to gain some kind of advantage. Known as "fake accounts", these users create friendships and exchange messages with common users who are unaware of their intent or reputation. In this way, a current challenge for online social networking is the identification of fake accounts. In this sense, this project proposes RealRank: a flexible framework that performs a ranking of the user's reputation in an online social network. RealRank relies on supervised learning algorithms for reputation ranking and random walk algorithms for social network reputation propagation. Both algorithms use only the hierarchical structural similarity and the relationship between the users of the social network. The algorithm is evaluated in the online social network of the platform Steam and compared with other approaches that use more user's information.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Algumas redes sociais online chegam a possuir milhões de usuários ativos e milhões de relacionamentos. Parte desses relacionamentos se concretizam na vida real enquanto outros existem apenas na própria rede, de maneira virtual. Dessa forma, dois usuários relacionados podem não ter certeza da identidade um do outro. Isso acontece pois, na maioria das redes sociais online (RSOs), é possível que se configure informações pessoais sem que sejam validadas ou comprovadas por administradores da rede. Sendo assim, a chance de um usuário se disfarçar com o intuito de atacar outros membros em uma RSO é alta. Esses ataques podem ser executados de diferentes maneiras. Uma delas são os ataques aleatórios, onde o usuário adiciona diferentes pessoas com o objetivo de fazer *spam* ou conseguir alguma informação pessoal de qualquer pessoa. A outra forma são os ataques direcionados, onde o usuário *fake* adiciona pessoas, conhecidas por ele, com algum objetivo malicioso, disfarçando sua própria identidade para não ser reconhecido durante o ataque.

Um sistema de ranqueamento de reputação de usuários de uma rede social pode oferecer ao usuário comum uma maior segurança ao se conectar com alguma uma outra conta na rede. Além disso, o sistema poderá oferecer também um maior controle de ataques à rede para seus administradores, que, atualmente, dependem principalmente do sistema de denúncias dos usuários para tal. Para isso, neste projeto foi desenvolvido um arcabouço flexível que calcula o ranqueamento de reputação de usuários, a partir de algoritmos de aprendizado supervisionado e algoritmos de passeios aleatórios, utilizando apenas a similaridade estrutural hierárquica e a relação entre os usuários de uma RSO. O ranqueamento será aplicado e avaliado na rede social online da plataforma Steam e comparado com resultados de outro método de classificação, que utiliza informações específicas de usuários da rede.

Essa seção irá apresentar a motivação para a realização deste projeto, uma introdução ao estudo de caso realizado, o objetivo do projeto, a técnica utilizada, a estruturação deste documento e as contribuições originais oferecida pelo autor.

1.1 Motivação

A motivação para a realização desse trabalho é a grande propagação de usuários maliciosos na rede social da *Steam* nos últimos anos. A criação de conta na plataforma é gratuita e usuários se aproveitam para tentar se aproximar de outras contas para roubar itens de conta ou informações pessoais.

A forma de tentar identificar um usuário não confiável em uma rede social ainda não é tão bem definida. Alguns estudos na área utilizam premissas de comportamento de usuários na rede para tentar identificá-los. Outros utilizam dados de atividades de contas. Nesse trabalho, será possível a identificação de usuários não confiáveis apenas com a estrutura de relacionamentos da rede, a partir de um modelo que oferece diferentes combinações de parâmetros de entrada para abranger qualquer tipo de rede social.

Com a premissa de que usuários não confiáveis tendem a ter um comportamento parecido na rede e, consequentemente, uma estrutura hierárquica de relacionamentos similar, a principal fonte de dados para o ranqueamento será a própria estrutura da rede: os usuários e as relações entre eles.

1.2 Estudo de caso: Steam

Steam é uma plataforma de jogos online, onde um usuário cria uma conta de forma gratuita e adquire jogos por ela. A plataforma também possui uma rede de relacionamento, utilizada para facilitar a relação entre as pessoas nos jogos que a plataforma oferece. A rede da Steam utilizada para o estudo de caso deste trabalho foi obtida em um estudo de algoritmo para recomendação de jogos [3]. A rede contém 114.047 vértices (usuários) com 1.098.684 arestas (amizades). No Capítulo 4 teremos uma melhor visão da estrutura da rede social da Steam, tal qual a execução do *framework* utilizando a rede e a respectiva análise dos resultados obtidos.

1.3 Objetivo

O objetivo desse trabalho é desenvolver um framework que tenha como entrada os relacionamentos de uma rede social qualquer, ou seja, uma lista de arestas, em conjunto com uma lista de classificação de usuários como confiáveis ou não confiáveis. O resultado será o ranqueamento da reputação ou a própria classificação de todos os vértices da rede, dependendo dos parâmetros submetidos ao framework. O ranqueamento final será diretamente influenciado pelo conjunto de treino oferecido como entrada. A tabela 1.1 mostra um exemplo de dados de entrada para o framework e sua possível saída.

Tabela 1.1: Exemplo de entrada e saída do framework. Dados apenas demonstrativos, não demonstram um resultado real do RealRank. Nesse caso, é possível ver que os vértices 5 e 6 possuem uma maior probabilidade de serem *fakes*.

Vértices	Relações	Confiável	Saída
1	2,3,4	Sim	0.93
2	1,3,5	-	0.85
3	1,2,5	-	0.70
4	1,6	Sim	0.50
5	2,3,6	Não	0.33
6	4,5	-	0.11

O framework terá como entrada dois arquivos de texto: o primeiro arquivo terá, em cada linha, uma representação de uma aresta da rede. Já no segundo arquivo, cada linha representará um vértice previamente classificado como confiável. A rede de entrada poderá ser não direcionada, como a rede da *Steam*, ou direcionada, como a rede do *Twitter*.

1.4 Técnica

O principal fundamento do arcabouço *RealRank* combina três etapas importantes para o ranqueamento da reputação ou classificação de usuários em uma rede: definição de *roles* na rede por similaridade estrutural hierárquica, classificação de confiança de usuário por modelos preditivos supervisionados e a disseminação da confiança pela rede através dos relacionamentos da rede utilizando passeios aleatórios. Para isso, utilizaremos como base do projeto a combinação e adaptação de dois modelos importantes: *struc2vec* [1], na primeira etapa, e *SybilRank* [2] na terceira.

Existe uma diferença de conceito importante para *Sybil accounts* neste trabalho. No *Sybil Rank*, usuários não confiáveis geralmente formam comunidades entre si e, além disso, formam o que chamaremos de *arestas de ataques* para comunidades confiáveis. A propagação da confiança pela rede é executada por esse motivo: se conseguirmos identificar usuários confiáveis na rede, a confiança não propagará para as comunidades de usuários não confiáveis. Porém, para este trabalho, foi considerado que dificilmente *Sybil accounts* formarão comunidades entre si, mas sempre formarão arestas de ataque na rede, ou seja, sempre estarão conectados em diferentes comunidades de usuários confiáveis. Isso faz com que se defina um novo conceito importante: usuários não confiáveis em uma rede terão uma média de amigos em comum menor que usuários confiáveis. Ou, pelo menos, terão proporções de vizi-

nhos em comum na rede parecidas com outros *fakes*. Portanto, a ideia base para esse estudo é de que, a partir de um conjunto de usuários previamente classificados, é possível definir a confiança de outras contas a partir da similaridade estrutural de grau e de vizinhos em comum entre tais usuários.

As três etapas do arcabouço são: projeção para um espaço latente de acordo com a similaridade estrutural dos vértices *(struc2vec)*, classificação através de modelo supervisionado tendo o espaço projetado como entrada e a propagação da confiança classificada pela rede *(SybilRank)*.

Diferentemente do *struc2vec* em sua origem, o qual busca a similaridade estrutural entre os vértices da rede apenas a partir da sequência de graus dos vizinhos de cada um (para cada anel de distância a partir do vértice), na primeira parte do framework busca-se similaridades entre vértices também a partir da quantidade de vizinhos em comum com seus vizinhos. Desse modo, vértices (usuários) com uma maior similaridade terão menor distância euclidiana entre si em um novo espaço latente de projeção desejado. A figura 1.1 representa um grafo Barbell(10,10), enquanto na figura 1.2 pode-se ver um exemplo de projeção desse mesmo grafo em um espaço latente considerando somente a similaridade estrutural hierárquica de grau.

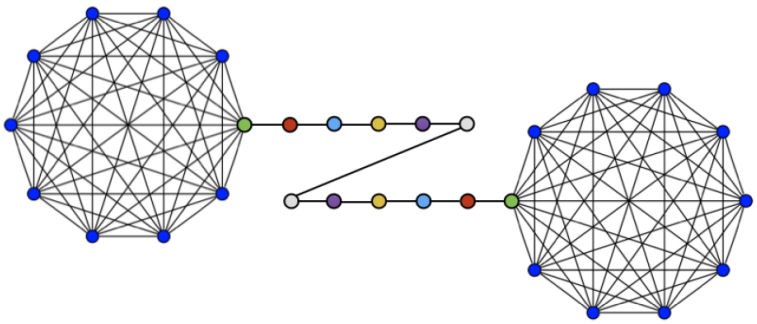


Figura 1.1: Barbell graph B(10x10) Imagem retirada de [1].

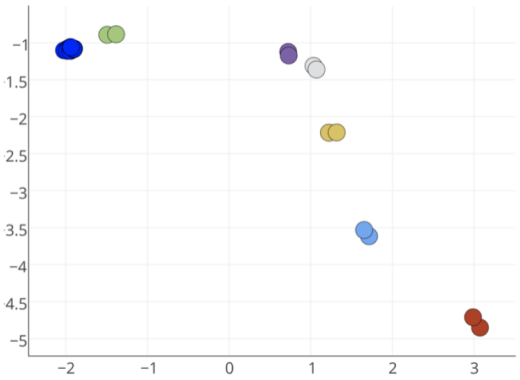


Figura 1.2: Projeção em espaço latente de 2 dimensões pelo struc2vec. Imagem retirada de [1].

Na segunda parte do framework, utilizaremos o espaço de vértices projetados na

primeira parte para treinar e classificar vértices como *confiáveis* ou *não confiáveis* utilizando um modelo de classificação supervisionado. Nessa etapa, o modelo poderá nos retornar, para cada vértice, sua probabilidade de ser confiável ou a sua própria classificação: 0 (zero) ou 1 (um). Isso dependerá do modelo de classificação escolhido e do parâmetro de propagação que será definido na seção 3.3.

Por último, na terceira parte, propaga-se a classificação definida na segunda etapa pela rede utilizando passeios aleatórios modificados, tendo como base uma propriedade importante do *Sybil Rank*: a limitação da quantidade de passeios em $log(n)$ iterações. A figura 1.3 demonstra a propagação em $log(n)$ iterações e também até atingir o estado estacionário. No estado estacionário, a confiança é distribuída pela rede proporcionalmente ao grau dos vértices. Já no passeio aleatório com iterações limitadas, a confiança ficará concentrada no que pode-se chamar de *parte confiável* da rede, visto que a quantidade de arestas entre usuários não confiáveis e usuários confiáveis (chamadas de *attack edges*) são limitadas em sua quantidade quando comparadas à arestas comuns.

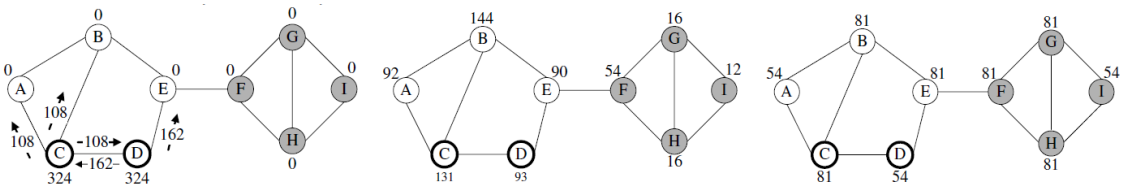


Figura 1.3: Propagação de confiança pela rede por passeios aleatórios. Imagem retirada de [2].

Com as relações de uma rede social e um conjunto de vértices previamente classificados, que são os dados de entrada, o framework RealRank poderá ser utilizado em um problema de otimização para qualquer métrica de classificação, levando em conta seus parâmetros de entrada, definidos no capítulo 3.4.

1.5 Estruturação do documento

No Capítulo 2, são descritos alguns trabalhos relacionados à detecção de *fake accounts*, os quais utilizam dados estruturais da rede e/ou características dos usuários. O Capítulo 3 descreve a solução adotada para o problema, que consiste no framework RealRank. Na Seção 3.4, será tratada a maneira em como o framework foi implementado e como poderá ser utilizado. Já no Capítulo 4, será analisado um estudo de caso que motivou esse trabalho: a detecção de usuários *fakes* na rede social da Steam. Por último, no Capítulo 5 será apresentada a conclusão sobre o trabalho realizado e indicações para possíveis trabalhos futuros.

1.6 Contribuições originais

Os itens a seguir são as contribuições originais oferecida pelo autor. Com isso, serão detalhados os modelo existentes utilizados, as adaptações que foram realizadas nesses modelos e as novas técnicas propostas.

- **Rede de relacionamentos da Steam** Serão disponibilizados pelo autor a sub-rede de relacionamentos (lista de arestas) obtida para a realização do estudo de caso da Steam, incluindo o respectivo conjunto de treino utilizado. A forma de obtenção do conjunto de treino está descrito na Seção 3.4.

- **Generalização do struc2vec** Como detalhado na Seção 3.1, o struc2vec oferece a projeção dos vértices para um espaço latente a partir da estrutura de relacionamentos da rede e do grau dos vértices. O RealRank propõe e implementa uma adaptação do modelo para a projeção da rede considerando também uma nova propriedade, a proporção da média dos amigos em comum pelo grau. Com isso, o RealRank considera uma combinação da similaridade nos dois parâmetros para o cálculo de similaridade final. Além disso, na Seção 5.1 é destacada a possibilidade de uma adaptação no RealRank para o uso de qualquer propriedade para o cálculo de similaridades, generalizando a classificação para compara todos os possíveis comportamentos estruturais de usuários em redes sociais.

- **Adaptação do SybilRank** Originalmente, o SybilRank considera uma propagação de confiança inicialmente definida pela rede. Essa propagação é uniforme entre os vizinhos de cada vértice e toda a confiança de um determinado vértice será considerada na propagação. Como o RealRank propõe um modelo supervisionado para definir uma confiança inicial a partir de uma projeção dos vértices rede para um espaço latente seguindo algumas propriedades, o SybilRank foi adaptado para que se consiga definir um peso desse classificador no resultado final, como definido na Seção 3.3.

- **Classificação supervisionada para ranqueamento de reputação de usuários** O RealRank propõe a aplicação de modelos supervisionados de classificação utilizando apenas a estrutura da rede para conseguir realizar o ranqueamento de reputação de usuários em qualquer rede social. Em combinação com modelos de projeção de vértices para um espaço latente e propagação de uma confiança inicial definida pelos vértices da rede, o RealRank possui diversos parâmetros de entrada, permitindo análises de diversas combinações de parâmetros e a obtenção de melhores resultados.

Capítulo 2

Trabalhos relacionados

Neste capítulo serão citados alguns trabalhos relacionados ao tema, que tiveram uma influência importante na motivação para a proposta de uma nova solução e no desenvolvimento do framework desse trabalho.

2.1 Classificação com modelos preditivos utilizando dados de usuário

Os métodos mais convencionais de detecção de *fake accounts* utilizam dados de usuário e modelos preditivos de classificação. Em algumas RSOs, pode ser possível identificar *clusters* de contas não confiáveis a partir de alguns dados de entrada, por exemplo, analisando a frequência de um mesmo padrão de comportamento na geração de e-mails e nomes. Assim, pode-se verificar a existência de um cluster de contas *fakes* criadas por um mesmo ator [4]. Outra forma de classificar seria treinando um modelo preditivo com dados de criação de cada relacionamento na rede, como por exemplo, utilizando o *timestamp* de criação de cada aresta da rede [5]. Nesse caso, é considerado que *Sybil accounts* não possuem um padrão estrutural de relacionamento na rede, ou seja, não se diferenciam de contas confiáveis somente por sua estrutura e que, dificilmente, contas não confiáveis tendem a se relacionar com outras *Sybil accounts*. Além disso, assume-se que o modo em que usuários não confiáveis se relacionam com os outros usuários segue um padrão parecido quando se observa o tempo de criação dos relacionamentos.

2.2 Propagação de confiança em rede

Existem alguns trabalhos que utilizam apenas a estrutura da rede para o ranqueamento de confiança dos vértices. Para isso, deve ser possível distinguir vértices confiáveis e não confiáveis olhando apenas para suas características estruturais, por

exemplo. Nesse tipo de abordagem, é considerado que *Sybils accounts* se conectam de uma forma diferente na rede quando comparado à usuários normais [2], de modo que fiquem mais separados de uma comunidade de vértices confiáveis, por exemplo. Portanto, se for definida uma confiança inicial para alguns usuários da rede e, depois disso, propagar tal confiança com passeio aleatório de maneira limitada, como por exemplo em log(num. vértices) iteracões, é possível disseminar essa confiança entre vértices realmente confiáveis e impedir que essa confiança chegue aos vértices não confiáveis. Um método para definir a confiança inicial é utilizar um algoritmo de detecção de comunidades como o Louvain Method [6], separando K vértices confiáveis para cada comunidade detectada.

Uma outra maneira de classificar usuários como não confiáveis é utilizando um modelo supervisionado a partir de dados de atividade da conta. Com isso, pode-se transformar a confiança inicial em pesos de arestas na rede [7] e propagar esse valores inicialmente definidos também por passeios aleatórios limitados. Desse modo, arestas entre usuários que são vizinhos de usuários classificados como *não confiáveis* terão pesos menores. O modelo descrito no *Íntegro* se aproxima com a ideia desse trabalho, visto que utiliza a combinação de classificação por modelos preditivos e conceitos de redes.

2.3 Projeção de vértices de uma rede para um espaço latente

Embedding de uma rede é a projeção de seus vértices para um espaço latente, assim obtendo uma nova representação da estrutura da rede. Dessa forma, pode-se utilizar esse espaço latente como entrada para modelos de aprendizado, por exemplo. Existem alguns tipos de algoritmos para realizar o *embedding* da rede: DeepWalk[8], node2vec[9] e struc2vec.

Os três algoritmos utilizam o Word2vec[10] para realizar a projeção de vértices para um novo espaço latente a partir de sequências de vértices, que são extraídas por passeios aleatórios. Para isso, nos três algoritmos, para cada vértice v da rede, são aplicados n_p passeios aleatórios de largura l_p. A largura significa a quantidade de visitas que o passeio aleatório irá realizar em cada passeio.

A diferença entre os algoritmos é o controle das probabilidades de cada visita no passeio aleatório. O DeepWalk aplica passeios aleatórios com probabilidade uniforme, ou seja, dado um vértice inicial v, a probabilidade do passeio aleatório ir para qualquer vizinho u de v é de $\frac{1}{d(v)}$, enquanto a probabilidade de visitar vértices que não são vizinhos é nula. O node2vec considera os parâmetros de entrada p e q para calcular a probabilidade do passeio aleatório visitar o vértice u a partir de um

vértice v, após visitar o vértice t. O cálculo dessas probabilidades é demonstrado no conjunto de equações 2.1. Por último, o struc2vec utiliza o conceito de similaridade estrutural hierárquica de grau para calcular as probabilidades do passeio. Dessa forma, se um vértice v possui vizinhos com o grau parecido com os vizinhos de u, por exemplo, v e u podem ter uma similaridade alta e, portanto, uma probabilidade alta para o passeio aleatório. Na seção 3.1, o cálculo de similaridade estrutural hierárquica será melhor detalhada.

$$
\begin{aligned}
&w(y,z) = \text{peso da aresta entre } y \text{ e } z \\
&d(y,z) = \text{menor distância entre } y \text{ e } z \\
&P(u|x) = \begin{cases} \frac{1}{p} & \text{se } d(t,x) = 0 \\ 1 & \text{se } d(t,x) = 1 \\ \frac{1}{q} & \text{se } d(t,x) = 2 \end{cases}
\end{aligned}
\tag{2.1}
$$

Capítulo 3

Framework proposto: RealRank

O framework proposto neste trabalho contém três partes em sua essência: detecção de similaridade estrutural dos vértices, classificação por modelo preditivo supervisionado utilizando o espaço latente projetado pela detecção de similaridades e a propagação da confiança prevista pelos relacionamentos da rede através de passeios aleatórios limitados. Serão detalhadas cada uma das etapas nas seções a seguir.

A inicialização de confiança no *Sybil Rank* é manual, podendo ser feita através da detecção de comunidades, por exemplo. Além disso, de forma parecida às premissas definidas no *Sybil Rank*, é considerado que *Sybil accounts* podem se conectar de uma maneira comum, de forma que ataquem uma pequena comunidade de vértices confiáveis, formando arestas de ataque pela rede. Porém, a premissa não considera que vértices não confiáveis necessariamente formem uma comunidade entre eles antes de realizar um ataque. A seção 3.1 descreverá como é possível identificar *Sybil Accounts* e usuários confiáveis através apenas da estrutura da rede. Já na seção 3.2, será utilizada a projeção gerada pela primeira etapa para treinar modelos preditivos de classificação. Por último, a seção 3.3 descreverá como será feita a propagação da confiança entre os vértices da rede utilizando passeios aleatórios em cima da estrutura original de relacionamentos da rede social.

A figura 3.1 define o fluxo que o framework seguirá para realizar o ranqueamento ou classificação dos vértices confiáveis.

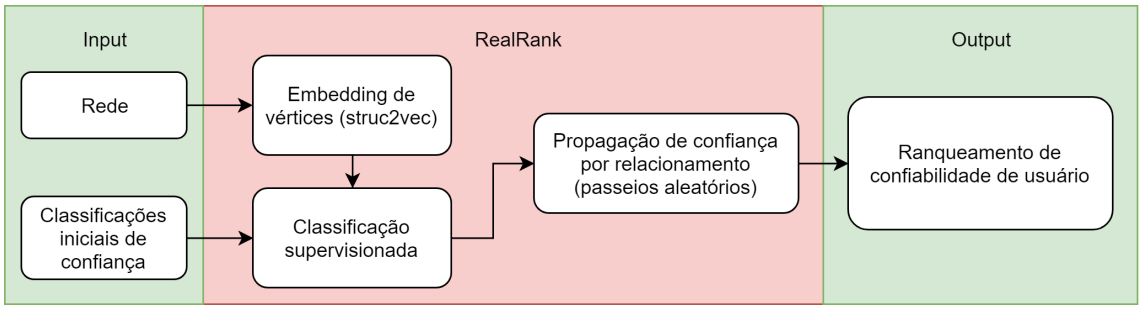
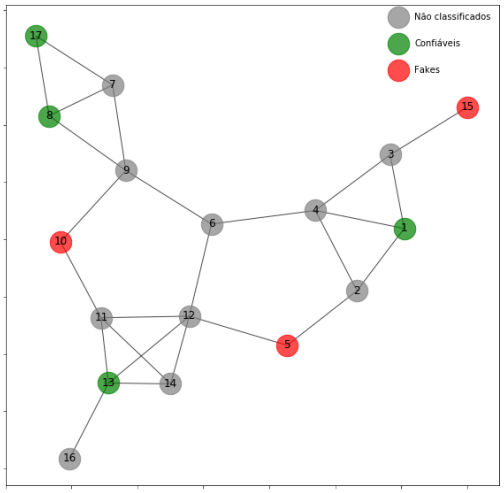


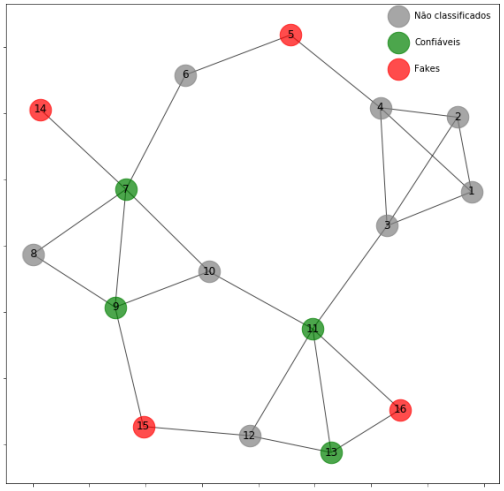
Figura 3.1: Fluxograma demonstrando etapas do RealRank.

3.1 Adaptação do struc2vec para projeção da rede

O objetivo dessa etapa é identificar um padrão hierárquico estrutural dos vértices confiáveis e não confiáveis, utilizando somente a estrutura de uma rede social, não considerando dados externos à ela. Para isso, foi realizada uma análise da estrutura de alguns vértices não confiáveis em duas redes sociais quaisquer. Como pode ser observado na imagem 3.2a, *Sybil accounts* podem ter uma alta similaridade em suas estruturas quando a relacionamos à quantidade de amigos em comum dos vértices em cada anel a partir de seu centro. Isso quer dizer que eles se conectam de forma parecida, como por exemplo *atacando* comunidades de vértices confiáveis. Já na figura 3.2b, é observado que vértices *fakes* também podem ter uma alta similaridade na estrutura hierárquica baseada nos graus dos vértices, onde se conectam de forma parecida quantitativamente, dificilmente estando fortemente conectados à uma comunidade de vértices confiáveis, por exemplo. Logo, dependendo da rede social a ser ranqueada, é necessária uma avaliação da maneira em que vértices não confiáveis criam arestas de ataque para outros usuários, pois podem se introduzir em comunidades confiáveis ou não, podendo ter um baixo ou elevado índice de vizinhos em comum em sua estrutura hierárquica. Devido a isso, o framework estará apto a realizar o cálculo de ambas similaridades hierárquicas estruturais entre os vértices.



(a) Usuários não confiáveis possuem uma hierarquia de vizinhos em comum parecida.



(b) Usuários não confiáveis possuem uma hierarquia de graus parecida.

Figura 3.2: Redes sociais com vértices previamente classificados como confiáveis ou não confiáveis (Sybil).

Como base de conceito, as notações descritas em 3.1 serão utilizadas nas próximas definições. N_v é o conjunto de vizinhos do vértice v, enquanto c(v) retorna a média da quantidade amigos em comum do vértice v. Por último, q(v) será a propriedade utilizada para o cálculo de similaridades: a proporção da média de amigos em comum

de um vértice v relativa ao seu grau. Os limites do valor de cada propriedade também estão presentes.

$$
\begin{aligned}
& d(v) = grau(v) \\
& c(v) = \tfrac{1}{d(v)} \textstyle\sum_{u \in N_v} |N_u \cap N_v| \\
& q(v) = \tfrac{c(v)}{d(v)} \\
& 0 \leq c(v) \leq d(v) - 1 \\
& 0 \leq q(v) \leq \tfrac{d(v)-1}{d(v)}
\end{aligned}
\tag{3.1}
$$

A similaridade estrutural entre cada vértice, no final da etapa, será representada pela distância euclidiana entre eles em um novo espaço latente. Para iniciar essa projeção, é necessário analisar cada um dos vértices da rede: deve-se obter a sequência de graus $d(v)$ e a sequência de média de amigos em comum $q(v)$ para cada anel k a partir do vértice central analisado, onde $k \in [0, k_{max}]$. Ou seja, no primeiro anel de um determinado vértice u ($k = 0$) tem-se seu próprio grau $d(u)$ e sua proporção da média de amigos em comum $q(u)$. Já em $k = 1$, tem-se a sequência $d(v)$ e $q(v)$ de seus vizinhos, para todo vizinho v. No terceiro anel, $k = 2$, irão ser extraídas as sequências dos vizinhos dos vizinhos do vértice raíz em questão. Isso se repete até que o último anel não possua vizinhos no próximo anel ou quando o número do anel ultrapassar o valor de k_{max}, parâmetro de entrada do framework. O número máximo de anéis possíveis é igual ao diâmetro da rede, que equivale ao maior valor das menores distâncias possíveis entre quaisquer dois vértices da rede. Como *Sybil accounts*, em geral, se conectam de forma semelhante até o segundo anel devido às conexões com comunidades de confiáveis, como é possível visualizar na figura 3.3, o valor padrão de k_{max} adotado no framework para o cálculo as sequências de grau e vizinhos em comum é até o segundo anel de cada vértice, ou seja, $k_{max} = 2$. Uma evidência para isso pode ser observada na tabela 3.1, onde é representada a quantidade de vértices considerados por anel $k \in [1, 3]$ para cada vértice raíz $u \in [0, 9]$, junto com a proporção de vértices até $k = 3$ sobre o número total de vértices da rede. Nesse último caso, acaba sendo considerada uma grande parte da rede que não representa o comportamento estrutural do usuário que está sendo analisado.

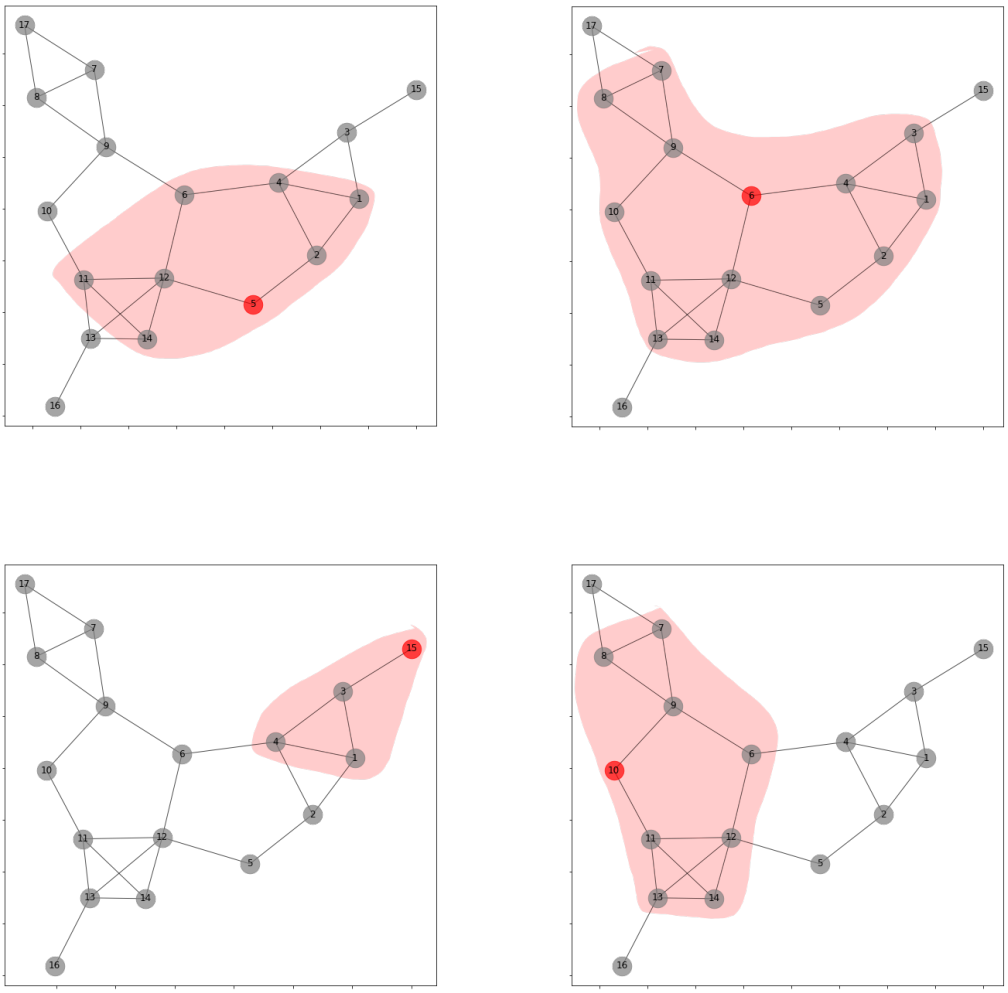


Figura 3.3: Demonstração de estrutura da rede até o segundo anel de usuários não confiáveis (marcados em vermelho).

Tabela 3.1: Quantidade de vértices por anel. Se for considerado $k = 3$, a estrutura da rede obtida dificilmente representará o comportamento estrutural do vértice raíz, pois abrange uma boa proporção de toda a estrutura da rede. N_k representa a quantidade de vértices até o anel k.

Vértice raíz	$N_{k=1}$	$N_{k=2}$	$N_{k=3}$	$\frac{N_{k=3}}{N}$
0	52	2,153	114,046	0.99
1	49	2,469	36,923	0.32
2	112	5,549	52,074	0.45
3	39	1,277	19,658	0.17
4	54	4,389	30,892	0.27
5	126	7,807	60,296	0.52
6	89	6,483	48,675	0.42
7	71	5,828	52,233	0.45
8	40	1,952	33,504	0.29
9	101	7,092	69,792	0.61

As equações de sequência de grau $R_k(v)$ e a sequência de média de amigos em comum $Q_k(v)$ foram definidas no conjunto de equações presentes em 3.2. Dessa forma, para qualquer vértice v, para seu primeiro anel, $k = 0$, $R_0(v)$ será igual ao número de grau do vértice d_v, enquanto $Q_0(v)$ será igual a média de amigos em comum do vértice v q_v, como definido nas equações em 3.3.

$$\begin{aligned} &l = \text{vértices no anel } k \\ &R_k(v) = [d(v_{0,k}), ..., d(v_{l,k})] \quad \forall v \in N, k \in [1, k_{max}] \\ &Q_k(v) = [q(v_{0,k}), ..., q(v_{l,k})] \quad \forall v \in N, k \in [1, k_{max}] \end{aligned} \tag{3.2}$$

$$\begin{aligned} R_0(v) &= [d(v)] \quad \forall v \in N \\ Q_0(v) &= [q(v)] \quad \forall v \in N \end{aligned} \tag{3.3}$$

Diferentemente do *struc2vec*, o cálculo de similaridade entre os vértices u e v será calculado a partir de funções que calculam distâncias entre sequências ordenadas. Portanto, é necessário ter ambas as sequências ordenadas em forma crescente, como definidas em 3.4.

$$
\begin{aligned}
s(R_k(v)) &= sorted(R_k(v)) \\
s(Q_k(v)) &= sorted(Q_k(v))
\end{aligned}
\tag{3.4}
$$

Para calcular ambas as similaridades hierárquicas, seja de grau ou de vizinhos em comum até o anel k entre dois vértices quaisquer na rede u e v, será calculada a distância entre suas respectivas sequências $s(R_k(v))$ e $s(Q_k(v))$. Para tal, foi utilizado o algoritmo *Fast Dynamic Time Warping* [11], o qual encontra o alinhamento ótimo entre qualquer duas sequências numéricas. O *Dynamic Time Warping* originalmente possui complexidade computacional $O(L^2)$, enquanto o FastDTW oferece uma complexidade $O(L)$, onde L é o tamanho da maior sequência de entrada. No cenário atual, no pior caso, uma sequência de graus de um determinado vértice para um determinado anel pode chegar a n. Portanto, se comparar todos os vértices em cada anel, ou seja, calcular a similaridade entre $\binom{n}{2}$ pares, obtêm-se uma complexidade de $O(n^3)$. Como a similaridade entre os vértices será calculada para cada anel, tem-se $O(kn^3)$. Como o valor de k não pode ser maior que o diâmetro da rede social em questão, geralmente não terá grande um impacto final na complexidade computacional.

Calcular a similaridade entre todos os pares de vértices para cada anel não será necessário. É possível reduzir o número de comparações sem alterar o resultado do modelo, reduzindo o número de comparações de n para os $log(n)$ vértices com o grau e média de amigos em comuns mais próximo do vértice em questão. Essa otimização é prevista no *struc2vec*. Assim, para o vértice u, serão selecionados $log(n)$ vizinhos com os números dos graus mais próximos a $d(u)$ e outros $log(n)$ vizinhos com média de amigos em comum mais próximos a $q(u)$. Dessa maneira, a complexidade é reduzida de $O(kn^3)$ para $O(kn^2log(n))$.

A função de custo definida para o cálculo de distância entre duas sequências é definida na equação 3.5. Diferentemente das sequências de amigos em comum, que será composta por números decimais, as sequências de graus serão compostas apenas por números inteiros. Como o número de grau varia entre 0 e $n-1$, muito provavelmente serão obtidas várias sequências de grau com números repetidos em uma rede. É possível reduzir o tamanho das sequências de grau descartando os números repetidos, reduzindo assim a complexidade de uma rede que possui muitos vértices com o mesmo grau. Para isso, foi utilizada a função de custo definida em 3.6, onde x_0 e y_0 são os graus e x_1 e y_1 são os números de ocorrência de cada grau na sequência. Porém, para RSOs com estruturas de relacionamentos mais esparsas que possuem, em sua maioria, vértices com diferentes números de grau, essa otimização não trará grandes resultados. Por outro lado, ao calcular a similaridade de amigos em comum, a sequência dificilmente será reduzida. Portanto, no pior caso, o maior

tamanho possível da lista de entrada para o cálculo do *fastDTW* continua sendo n, mantendo a complexidade anterior.

$$d(x, y) = \frac{max(x, y) + \epsilon}{min(x, y) + \epsilon} - 1 \tag{3.5}$$

$$d(x, y) = \left(\frac{max(x_0, y_0) + \epsilon}{min(x_0, y_0) + \epsilon} - 1\right) max(x_1, y_1) \tag{3.6}$$

Exemplificando, para as redes das imagem 3.2b e 3.2a, algumas propriedades de R e Q são demonstradas na tabela 3.2. A partir disso, é possível observar o cálculo da similaridade hierárquica dos usuários não confiáveis em cada rede.

Tabela 3.2: Sequências de graus e vértices para cada anel utilizado no cálculo de $fr_k(v) e fq_k(v)$ para vértices previamente classificados como não confiáveis

Sequência de média de vizinhos em comum para 3.2a	Sequência de grau para 3.2b
$s(Q_0(5)) = [0.00]$	$s(R_0(15)) = [2]$
$s(Q_0(10)) = [0.00]$	$s(R_0(16)) = [2]$
$s(Q_1(5)) = [0.67, 1.20]$	$s(R_1(15)) = [3, 4]$
$s(Q_1(10)) = [0.50, 1.50]$	$s(R_1(16)) = [3, 5]$
$s(Q_2(5)) = [0.00, 1.00, 1.33, 1.50, 1.50, 2.00]$	$s(R_2(15)) = [2, 3, 3, 5, 5]$
$s(Q_2(10)) = [0.00, 1.20, 1.33, 1, 33, 1.50, 2.00]$	$s(R_2(16)) = [3, 3, 4]$

Os resultados do cálculo de $f_k(v)$ e $h_k(v)$ para cada rede, seguindo as equações definidas em 3.7, podem ser observados na tabela 3.3. Vale lembrar que foram tratadas nos exemplos apenas as distâncias de similaridade em que podemos ter uma maior distinção de usuários não confiáveis dos confiáveis em cada rede. Ou seja, foram calculadas apenas a similaridade hierárquica de vizinhos em comum para a rede que possui usuários não confiáveis com uma maior semelhança nesse quesito (grau ou vizinho em comum). Isso pode ser definido através de um parâmetro de entrada do framework, que será definido na seção 3.4.

$$f_k(u, v) = \frac{1}{exp(fastDTW(s(R_k(u)), s(R_k(v)), d))} \quad \text{onde d é a função de custo definida em 3.6}$$

$$h_k(u, v) = \frac{1}{exp(fastDTW(s(Q_k(u)), s(Q_k(v)), d))} \quad \text{onde d é a função de custo definida em 3.5} \tag{3.7}$$

Tabela 3.3: Cálculo de similaridades de grau e média de vizinhos em comum com fastDTW e suas respectivas funções de custo

Similaridade por média de vizinhos em comum para 3.2a	Similaridade por grau para 3.2b
$h_0(5,10) = 0.00$	$f_0(15,16) = 0.00$
$h_1(5,10) = 0.34$	$f_1(15,16) = 0.22$
$h_2(5,10) = 0.13$	$f_2(15,16) = 1.24$

Com as similaridades calculadas, o próximo passo é transformar esses resultados em pesos de arestas em novas redes de similaridades hierárquicas. Para isso, serão montadas duas novas redes multi-camadas, onde o número de camadas é igual ao número de anéis k_{max}. Cada rede possuirá os respectivos pesos de arestas definidas a partir das similaridades calculadas. Portanto, para cada camada das novas redes, o resultado da similaridade entre os vértices u e v do anel k será transformado no peso da aresta entre u e v da camada k. Desse modo, as novas redes terão kn vértices e aproximadamente $knlog(n)$ arestas. Estas arestas serão unidirecionais e são necessárias para a aplicação dos passeios aleatórios enviesados através dos pesos da rede. Cada peso de uma aresta de saída será a proporção do peso calculado a partir da respectiva similaridade ao peso total de todas as arestas de um determinado vértice u, como definido nas equações em 3.8.

$$\begin{aligned} w_k(u \to v) &= \frac{f_k(u,v)}{\sum\limits_{j \in N_u} f_k(u,j)} \\ y_k(u \to v) &= \frac{h_k(u,v)}{\sum\limits_{j \in N_u} h_k(u,j)} \end{aligned} \tag{3.8}$$

Com a estrutura e pesos das duas redes multi-camadas de similaridades bem definidas, o próximo passo será montar a rede final em que será aplicado o passeio aleatório para a identificação de similaridades. Para qualquer camada k, o passeio aleatório terá a probabilidade de mudar de um vértice u para o vértice v igual a $p_{u,v,k}$, definida na equação 3.9. O valor de p_c será um parâmetro de entrada responsável pela regularização das similaridades de grau e de vizinhos em comum. Por exemplo, se p_c for igual a 0, a probabilidade do passeio aleatório pular de um vértice u para o vértice v será igual ao peso entre u e v calculado a partir da similaridade de grau $w_k(u,v)$. Por outro lado, se o valor de p_c for igual a 1, a probabilidade será de $w_k(u,v)$, que é o peso calculado a partir da similaridade da média de vizinhos em comum.

$$p_{u,v,k} = p_c(w_k(u,v)) + (1 - p_c)(y_k(u,v)) \tag{3.9}$$

As k_{max} camadas da rede multi-camadas final também estarão conectadas entre

si. Isso é necessário para permitir o passeio aleatório navegar entre todas as camadas, considerando assim todos os níveis (anéis) de similaridade. Para isso, todo vértice estará conectado com ele mesmo na camada superior e inferior. Ou seja, para $k_{max} = 2$, haverão duas arestas unidirecionais entre u_0 e u_1 e outras duas arestas entre u_1 e u_2. Para definir os pesos delas, *struc2vec* define um conceito de que a probabilidade de um passeio aleatório chegar a uma camada superior através do vértice u deve ser proporcional à quantidade de arestas que apontam para u que possuam o peso acima da média em relação ao resto da rede, como definido nas equações em 3.10. Ou seja, quanto maior a quantidade de arestas incidentes ao vértice u que possuem peso maior que a média de pesos da rede ($T(u)$), maior o peso da aresta para o mesmo vértice na camada superior, aumentando a probabilidade do passeio aleatório subir para aquela camada enquanto estiver navegando por aquele vértice. Por outro lado, o peso do vértice para ele mesmo na camada inferior será sempre 1. Esse peso ainda não é a probabilidade final para o passeio aleatório, mas será transformado nela. Pode-se observar a probabilidade do passeio aleatório mudar de camadas nas equações presentes em 3.11. Dessa maneira, o passeio aleatório só irá considerar similaridades hierárquicas em anéis superiores caso u tenha uma similaridade maior com outros vértices que a média da rede para aquele anel.

$$\begin{aligned} w_{k,k+1}(u) &= log(T(u) + \epsilon) \\ w_{k,k-1}(u) &= 1 \end{aligned} \tag{3.10}$$

$$\begin{aligned} p_{k,k+1}(u) &= \frac{w_{k,k+1}(u)}{w_{k,k+1}(u)+w_{k,k-1}(u)} \\ p_{k,k-1}(u) &= 1 - p_{k,k+1}(u) \end{aligned} \tag{3.11}$$

Com todas as probabilidades definidas, para cada vértice serão executados n_p passeios aleatórios de tamanho l_p. Ambos serão entradas para o framework *RealRank*, como também são no *struc2vec*. A execução do passeio aleatório será sequencial nos vértices e o resultado será a lista dos vértices visitados a partir de cada vértice. Dessa maneira, teremos um total de Nn_pl_p visitas. Para a rede com os pesos já calculados exibida na figura 3.4, o passeio aleatório será executado em cada vértice e podemos visualizar o resultado na tabela 3.4.

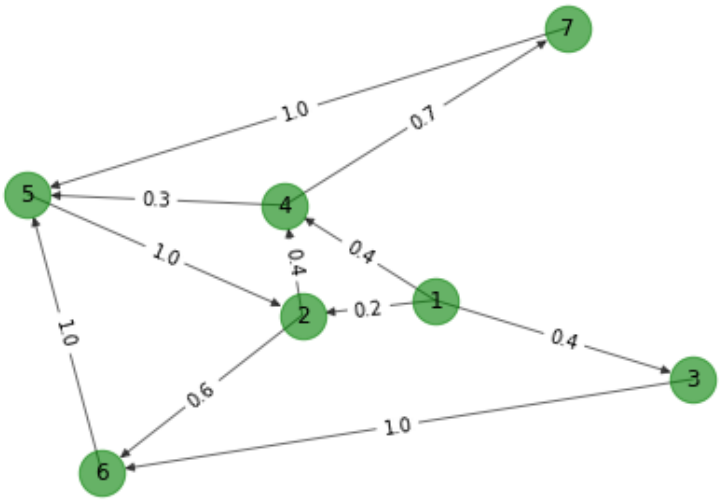


Figura 3.4: Exemplo de rede com pesos para aplicação de passeios aleatórios.

Tabela 3.4: Exemplo de uma aplicação de passeios aleatórios com número de passeios por vértice $n_p = 2$ e largura do passeio $l_p = 2$.

Vértice origem	Sequência 1 de passeios aleatórios	Sequência 2 de passeios aleatórios
1	3, 6	4, 7
2	6, 5	4, 5
3	6, 5	6, 5
4	7, 5	5, 2
5	2, 6	2, 4
6	5, 2	5, 2
7	5, 6	5, 6

Para produzir o *embedding* da rede, ou seja, a projeção dos vértices em um espaço latente onde as distâncias entre eles estão diretamente relacionadas às suas similaridades hierárquicas, é necessária uma representação latente desses vértices e de um modelo que nos retorne uma função de probabilidades de um contexto de vértices, dado um certo vértice. Além disso, as sequências de vértices produzidas pelos passeios aleatórios retorna as relações de similaridades entre os próprios vértices, ou seja, um vértice u que aparece em sequência após o vértice v significa que estes provavelmente possuem um alto valor de similaridade.

A representação de vértices será por *one-hot encoding*, onde cada palavra possui um bit específico em um vetor de zeros. Um exemplo dessa representação pode ser observado na tabela 3.5. Para este trabalho foi utilizado o modelo *Skip-gram*

(word2vec), o mesmo indicado no *struc2vec*. Porém, o framework pode utilizar qualquer outro modelo para realizar esta etapa de projeção, dado uma sequência de vértices/palavras. O *Skip-gram* retorna uma função de distribuição da probabilidade de uma palavra h, que em nosso caso é um vértice, estar no mesmo contexto de outra palavra x, ou seja, $P(h|x)$. O modelo utiliza uma rede neural com apenas uma camada escondida e com a função de ativação *SoftMax* na camada de saída, retornando a equação definida em 3.12. Na equação, tem-se que θ é o vetor de parâmetros do modelo, e_x é o vetor de projeção do vértice x e W é o conjunto de palavras de entrada para o modelo, ou seja, o conjunto de vértices da rede.

Tabela 3.5: One-hot encoding

Vértice	Representação por one-hot encoding
1	[1,0,0,0]
2	[0,1,0,0]
3	[0,0,1,0]
4	[0,0,0,1]

$$p(h|x) = \frac{exp(\theta_h^T e_x)}{\sum\limits_{j \in W} exp(\theta_j^T e_x)} \tag{3.12}$$

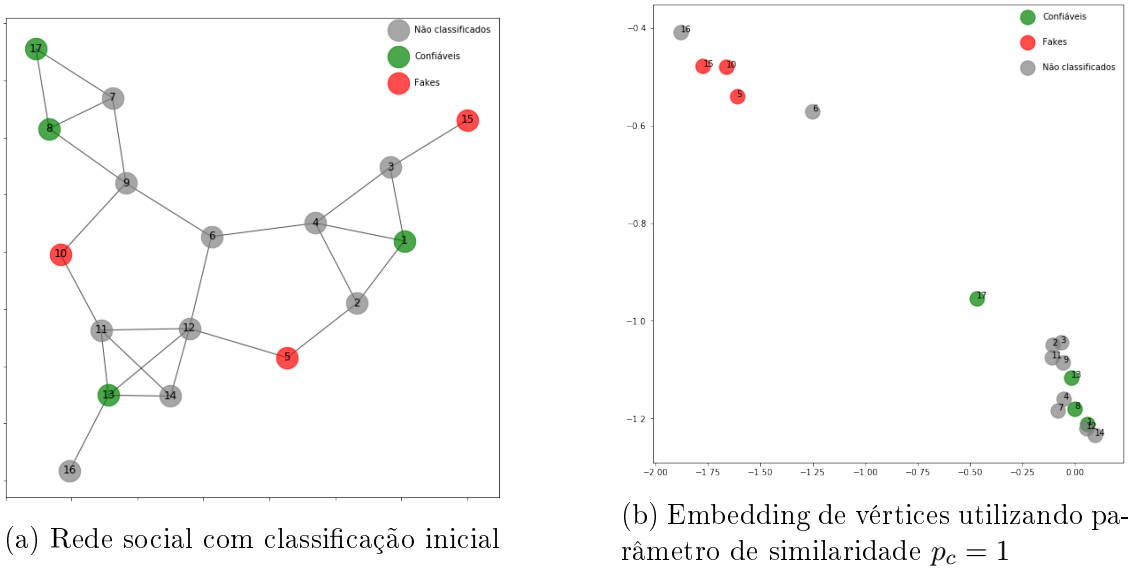
Com o tamanho de janela w definido, é exemplificado na tabela 3.6 o contexto de cada centro a partir de uma sequência de vértices inicial e o respectivo conjunto de treino gerado. A cada iteração realizada sobre o conjunto de treino, a rede minimizará a função de custo definida pela equação 3.13, conhecida como *Cross entropy*. A função de otimização do custo pode ser entendida como a maximização da função de verossimilhança do vetor de probabilidades retornado pela camada de saída do modelo. Desse modo, após o modelo estar treinado, todas as palavras que aparecem com frequência em um mesmo contexto terão uma distância euclidiana menor no espaço latente produzido.

Tabela 3.6: Conjunto de treino para o SkipGram.
Sequência obtida pelo passeio aleatório: 17, 38, 47, 22, 18. Tamanho da janela w=2

Centro	Contexto para janela w	Conjunto de treino gerado
17	38,47	(17,38), (17,47)
38	17,47, 22	(38,17), (38,47), (38,22)
47	17,38,22,18	(47,17), (47,38), (47,22), (47,18)
22	38,47,18	(22,38), (22,47), (22,18)
18	47,22	(18,47), (18,22)

$$C(p, y) = -\sum_{i \in W} y_i log(p_i) \tag{3.13}$$

Voltando ao contexto de RSO, é possível observar uma projeção dos vértices de uma rede social aleatória nesta primeira etapa do framework na figura 3.5. A projeção gerada no espaço latente, o conjunto de treino de classificação (vértices vermelhos e verdes) e o conjunto não classificado (vértices em cinza) serão os dados de entrada para a próxima etapa.



(a) Rede social com classificação inicial

(b) Embedding de vértices utilizando parâmetro de similaridade $p_c = 1$

Figura 3.5: Rede social aleatória e embedding de vértices

3.2 Classificação de espaço latente projetado por modelos preditivos supervisionados

Para a segunda etapa do *framework*, será utilizado um modelo supervisionado de classificação para definir uma classificação inicial de todos os usuários da rede social de entrada, a partir da projeção de vértices produzida na primeira etapa. O espaço latente utilizado como entrada nesta etapa é agnóstico aos rótulos dos vértices, sendo gerado considerando apenas dados extraídos da estrutura da rede. Portanto, o modelo supervisionado irá classificar vértices como confiável ou não confiável de acordo apenas com a estrutura da rede.

Um dos modelos indicados para esta etapa é a Regressão Logística [12], visto que este é utilizado em problemas de classificação binária e sua saída é definida como a probabilidade de um conjunto de variáveis de entrada ser classificado como 1 (um), como definido na equação 3.14. Porém, qualquer outro modelo de classificação supervisionada pode ser utilizado, retornando a classificação ou a sua probabilidade

de classificação. O conjunto de dados de entrada será o *embedding* dos vértices, ou seja, os vértices projetados em um espaço latente de acordo com as propriedades estruturais hierárquicas como definidas na seção 3.1. Tomando como exemplo, para a rede da imagem 3.5a, as variáveis x do conjunto de treino definem a posição no espaço de cada vértice na figura 3.5b, juntamente com suas classificações iniciais y, definidas na tabela 3.7. Com esse conjunto de treino oferecido como entrada do framework (além da rede), como definido na seção 1.3, o modelo preditivo de classificação será treinado e as classificações iniciais serão geradas.

$$P(y = 1|X) = \frac{1}{1+exp(-X\theta^T)} \tag{3.14}$$

Tabela 3.7: Conjunto de treino para a rede social da imagem 3.5a

Vértice	X1	X2	Y
1	-1.21	0.05	1
5	-0.54	-1.61	0
8	-1.17	0.00	1
10	-0.48	-1.66	0
13	-1.11	-0.01	1
15	-0.47	-1.77	0
17	-0.95	-0.46	1

Dependendo dos parâmetros definidos para a projeção da primeira etapa e da qualidade do conjunto de treino, pode ser possível obter um conjunto de dados linearmente separável. Dessa forma, vértices com a mesma classificação poderão ficar bem separados no espaço latente, de forma com que se for aplicada uma análise discriminante linear para o conjunto de treino observado, será possível chegar a um número de separabilidade das duas classes em apenas uma dimensão: número de classes (2) menos 1 (um). Por outro lado, se o conjunto de treino não for linearmente separável, o resultado da classificação retornada pela regressão logística (ou de qualquer outro classificador) pode ser pior. Na figura 3.6 é exibido o resultado da classificação pela regressão logística a partir de uma projeção de rede gerada pela primeira etapa, onde o usuário é classificado como confiável se sua probabilidade de ser confiável for maior que 0.5, valor padrão de *threshold de classificação* para a regressão logística.

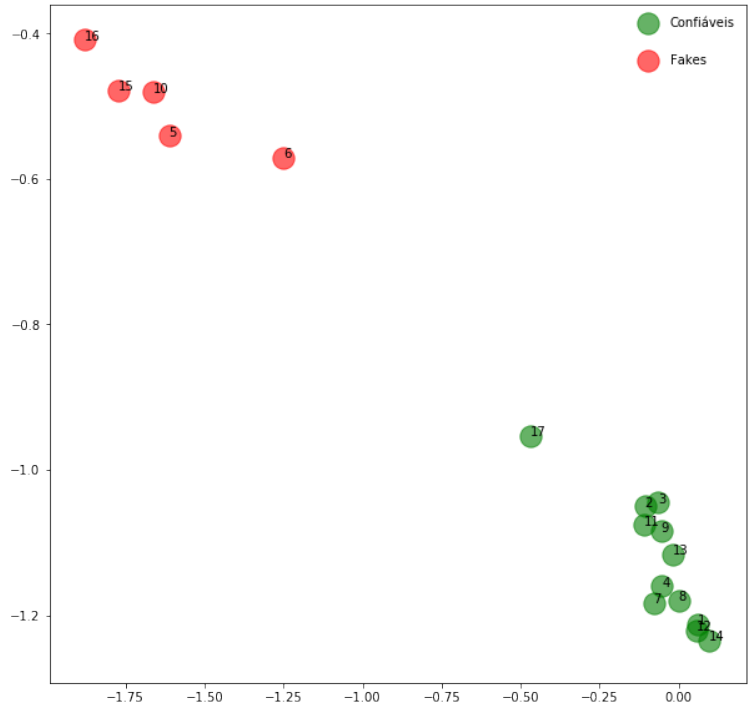
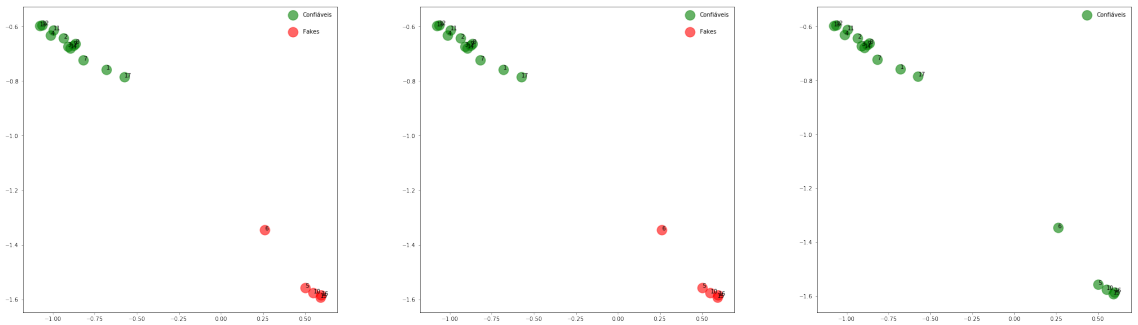


Figura 3.6: Classificação de vértices por regressão logística em um conjunto de treino linearmente separável. Nesse caso, os vértices 1, 7, 13 e 17 foram previamente classificados como confiáveis e os vértices 5, 10 e 15 classificados como não confiáveis

Outro método de classificação que pode ser utilizado nessa etapa é o *K-Nearest-Neighbor*, um modelo de classificação não linear que utiliza a distância entre os vértices no espaço latente, onde a classificação é definida a partir da classificação de K vizinhos mais próximos. Um resultado de exemplo é exibido na figura 3.7. Portanto, o resultado da classificação retornada pelo KNN ou por qualquer outro modelo de classificação também pode ser utilizada como um vetor de classificação inicial para a próxima etapa.



(a) N = 1. (b) N = 4. (c) N = 7.

Figura 3.7: Classificação utilizando KNN para diferentes valores de N

Para as redes onde a projeção não produziu um conjunto de treino linearmente separável, para qualquer parâmetro utilizado, é possível realizar uma clusterização

na rede com todos os vértices projetados para visualizar a posição de vértices do conjunto de treino em um mesmo *cluster*. Em alguns casos, usuários não confiáveis e usuários confiáveis irão pertencer a um mesmo agrupamento. Por isso, a etapa descrita na seção 3.3 é de suma importância.

O resultado final dessa etapa será a probabilidade ou a classificação definida pelo modelo de classificação utilizado. Assim, cada vértice terá uma probabilidade inicial de ser confiável p_i prevista pelo modelo, onde $p_i \in [0, 1]$. A tabela 3.8 mostra um exemplo da probabilidade inicial de classificação de usuários confiáveis para a rede da figura 3.6.

Tabela 3.8: Classificação inicial definida pela regressão logística

Vértice	Probabilidade de ser confiável (Y_proba)	Y >= 0.5
1	0.77	Sim
2	0.75	Sim
3	0.74	Sim
4	0.75	Sim
5	0.26	Não
6	0.28	Não
7	0.78	Sim
8	0.70	Sim
9	0.80	Sim
10	0.29	Não
11	0.77	Sim
12	0.74	Sim
13	0.81	Sim
14	0.75	Sim
15	0.24	Não
16	0.23	Não
17	0.64	Sim

3.3 Propagação de confiança pela rede utilizando passeios aleatórios (Sybil Rank)

O SybilRank utiliza de uma classificação inicial realizada de forma manual para os vértices da rede. Isso pode ser um problema quando redes sociais possuem muitos usuários. O RealRank, até aqui, propôs uma classificação inicial para que a propagação de confiança pela rede possa ser realizada, visto que para a aplicação do *Sybil Rank* precisa-se de uma classificação coesa de vértices de diferentes comunidades, de forma a espalhar confiança por toda a rede.

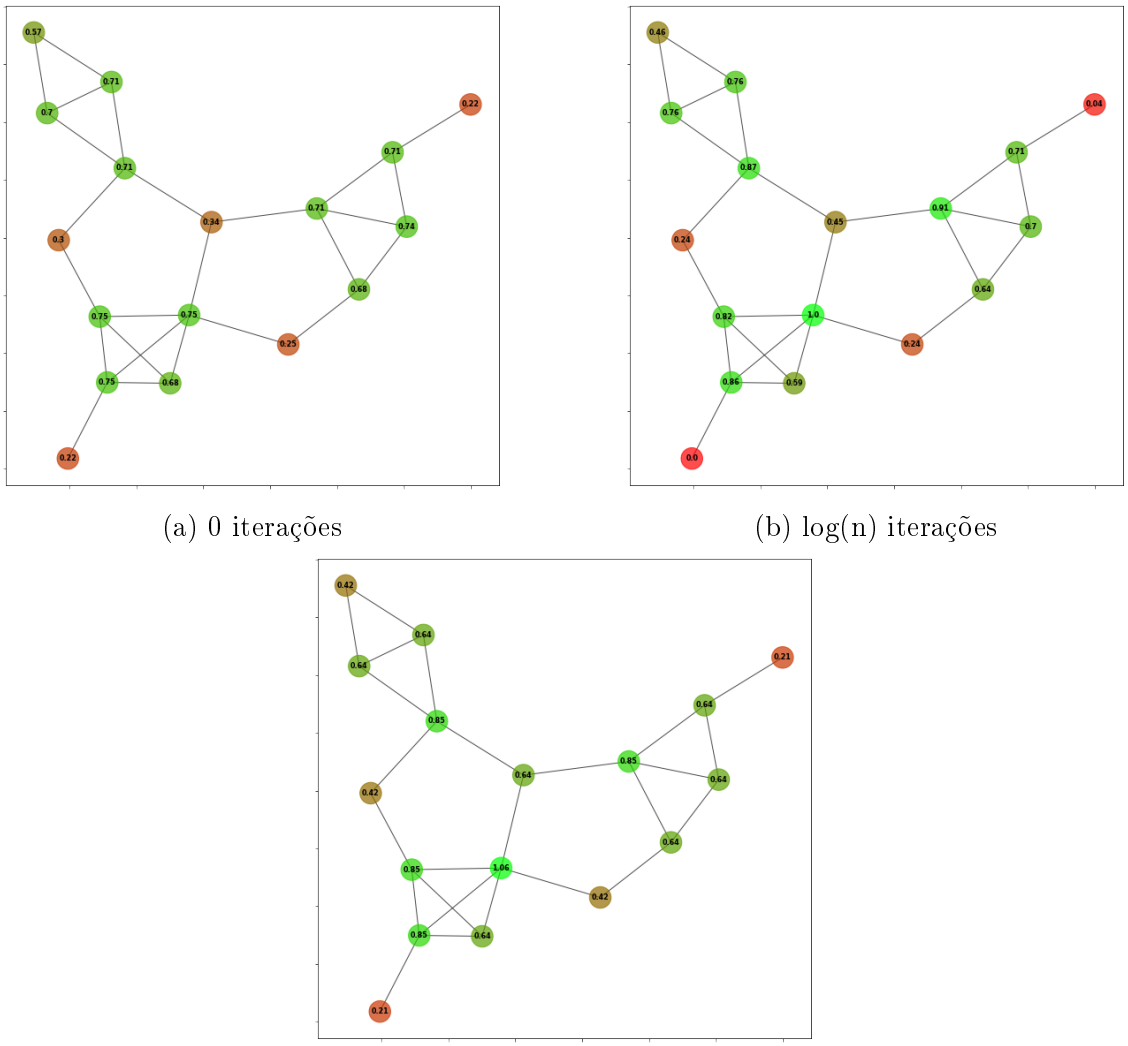
A partir de um vetor inicial de classificação de confiança p_i de todos os vértices da rede social, definidos na execução das tapas descritas nas seções 3.1 e 3.2, essa etapa irá aplicar o conceito de passeios aleatórios com iterações limitadas para a propagação da confiança inicial pela rede. A forma de propagação por passeios aleatórios adotada nesse trabalho foi por *power iteration*, visto que a probabilidade do passeio aleatório andar de um vértice u para um vizinho v é igual à probabilidade de andar para um outro vizinho j. Portanto, a partir de uma lista de adjacência, é possível propagar a confiança para todos os seus vizinhos.

Usuários não confiáveis normalmente formam arestas de ataque para diferentes comunidades confiáveis e ficam, de certa forma, mais isolados da rede do que usuários confiáveis. Para todo caso, dado que a confiança inicial da rede é inicializada através da classificação supervisionada sobre a similaridade estrutural dos vértices pré classificados, será possível definir a confiança final dos vértices após a propagação da mesma por *power iterations*, utilizando a fórmula de propagação dada na equação 3.15. Para realizar essa propagação de confiança pela rede, é definido um parâmetro p_f que determinará o peso da confiança que será disseminada. Se $p_f = 0$, a cada iteração as confianças de cada vértice serão completamente distribuídas entre seus vizinhos. Já para $p_f = 1$, a confiança do vértice não será propagada entre seus vizinhos. Esse parâmetro é utilizado para ter um maior controle do impacto da classificação inicial dos vértices obtido na etapa de classificação supervisionada sobre o resultado final.

$$C_t(u) = (p_f)C_{t-1}(u) + (1 - p_f)\sum_{v \in N_u} \frac{C_{t-1}(v)}{d(v)} \tag{3.15}$$

Como definido na seção 1.4, as iterações são limitadas em $log(n)$ para a confiança não seja completamente distribuída entre toda a rede, considerando a premissa de que usuários não confiáveis tendem a estar mais separados das comunidades onde se concentram vértices confiáveis. A figura 3.8 demonstra uma rede social de exemplo antes da propagação, depois $log(n)$ iterações de propagação e até atingir o estado estacionário.

O último passo desta etapa é a normalização dos resultados para que o ranqueamento final fique entre 0 (zero) e 1 (um). Dessa forma, sempre existirá um vértice com confiança igual a 1 (um), que será o mais confiável, e outro vértice com confiança igual a 0 (zero), o menos confiável. É possível observar na tabela 3.9 um ranqueamento final de confiança para a rede da figura 3.8.



(a) 0 iterações

(b) log(n) iterações

(c) Estado estacionário

Figura 3.8: Propagação de confiança por SybilRank utilizando $p_f = 0$

Tabela 3.9: Ranqueamento de confiança final após a propagação por SybilRank

Vértice	Ranqueamento final
12	1.00
4	0.90
13	0.89
9	0.82
11	0.80
7	0.76
8	0.75
3	0.71
1	0.69
2	0.63
14	0.56
17	0.46
6	0.41
5	0.22
10	0.19
15	0.02
16	0.00

3.4 Implementação

O framework foi implementado utilizando a linguagem Python na versão 2.7.15. A implementação da primeira etapa, a projeção dos vértices da rede, foi realizada a partir de uma modificação da implementação do struc2vec, disponível em seu repositório no GitHub, *https://github.com/leoribeiro/struc2vec*. Modificação para que, além da similaridade estrutural hierárquica por grau, também seja calculada a similaridade estrutural hierárquica de vizinhos em comum. Inicialmente, para cada vértice v da rede é calculada sua propriedade $q(v)$. Dessa forma, ao aplicar uma busca em largura pela rede para cada anel de um vértice, é possível montar uma sequência $s(Q)$ de cada anel. Para a segunda parte, três modelos de classificação supervisionada foram utilizados, que podem ser selecionados a partir dos parâmetros de entrada do framework: Regressão Logística, Support Vector Machine (SVM) [13] e K-Nearest-Neighbors (KNN) [14]. Esses três modelos de classificação, inclusive

os métodos de avaliação, foram utilizados a partir da implementação na biblioteca *scikit-learn* [15]. Já para a manipulação dos dados foram utilizadas a biblioteca *pandas* [16] e *numpy* [17]. Por último, a propagação de confiança por passeios aleatórios foi implementada com multiplicação por lista de adjacência. O pseudo código da implementação para a última etapa pode ser visto abaixo:

```
numIteracoes =  log(qtd.vertices)
i = 0
while (i < numIteracoes):
 para cada vertice V:
   confiancaVizinhos = 0
   para cada vertice U em listaAdjacencia(V):
       confiancaVizinhos = confiancaVizinhos + confianca(i-1)(U)/grau(U)
   confianca(i)(V) = p*confianca(i-1)(U) + (1-p)*confiancaVizinhos
 i++;
```

Os parâmetros de entrada do framework estão representados tabela 3.10, juntamente com seus valores padrão. A forma de execução é por linha de comando e, os parâmetros que forem diferentes de seus valores padrão, deverão ser informados, como representado abaixo:

```
python main.py --input=''rede.edgelist'' --train=''rede.train''
--output-rank=''rede.realrank'' --pc=0.75 --pf=0.5
```

Tabela 3.10: Parâmetros de entrada para o framework RealRank

Parâmetro	Valor Padrão	Forma de Entrada
Rede de entrada	"social_network.edgelist"	–input="rede.edgelist"
Conjunto de treino	"social_network.train"	–train="rede.train"
Saída do embedding	"social_network.struc2vec"	–output="rede.struc2vec"
Saída do ranqueamento	"social_network.realrank"	–output-rank="rede.realrank"
Número de dimensões do embedding	128	–dimensions=32
Largura do RW	80	–walk-length=80
Número de RW's por vértice	10	–num-walks=10
Largura da janela do Word2Vec	5	–window-size=5
Quantidade de anéis para o cálculo das similaridades	2	–until-layer=2
Número de épocas para o treinamento do Word2Vec	5	–iter=5
Quantidade de processos simultâneos para execução	4	–workers=4
Modelo de classificação	lr	–model=knn
Parâmetro de regularização para Regressão Logística e SVM	1	–c=0.1
Kernel utilizado no SVM	rbf	–kernel=linear
Número de vizinhos no KNN	3	–nneigh=5
Peso de cada similaridade no embedding	0.5	–pc=0.5
Influência do resultado do classificador no ranqueamento final	0.5	–pf=0.5

O formato de entrada do framework será por arquivo de texto. Dois arquivos serão necessários: um deve conter a relação entre os vértices da rede e o outro deve conter um conjunto de treino. As tabelas 3.12 e 3.13 exemplificam cada um dos arquivos de entrada. As colunas do arquivo de entrada da rede são as arestas (dois vértices relacionados) e as colunas do arquivo de entrada de treinamento são os vértices e sua classificação.

Tabela 3.11: Arquivos necessários de entrada para execução do framework

Tabela 3.12: Rede

rede.edgelist	
1	3
1	4
1	23
1	735
23	1110

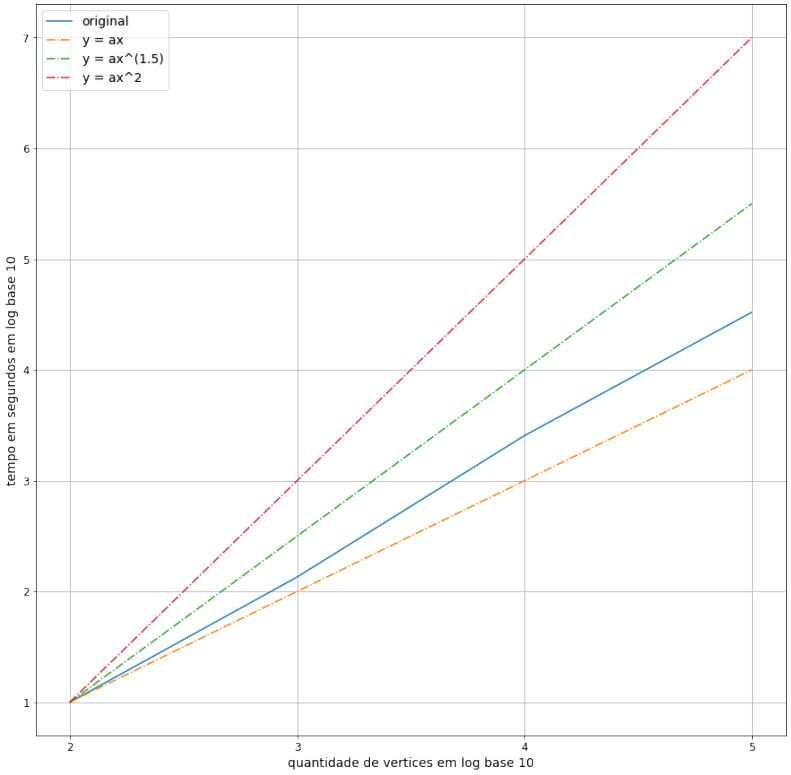
Tabela 3.13: Conjunto de treino

rede.train	
1	0
3	1
4	1
23	0
1110	1

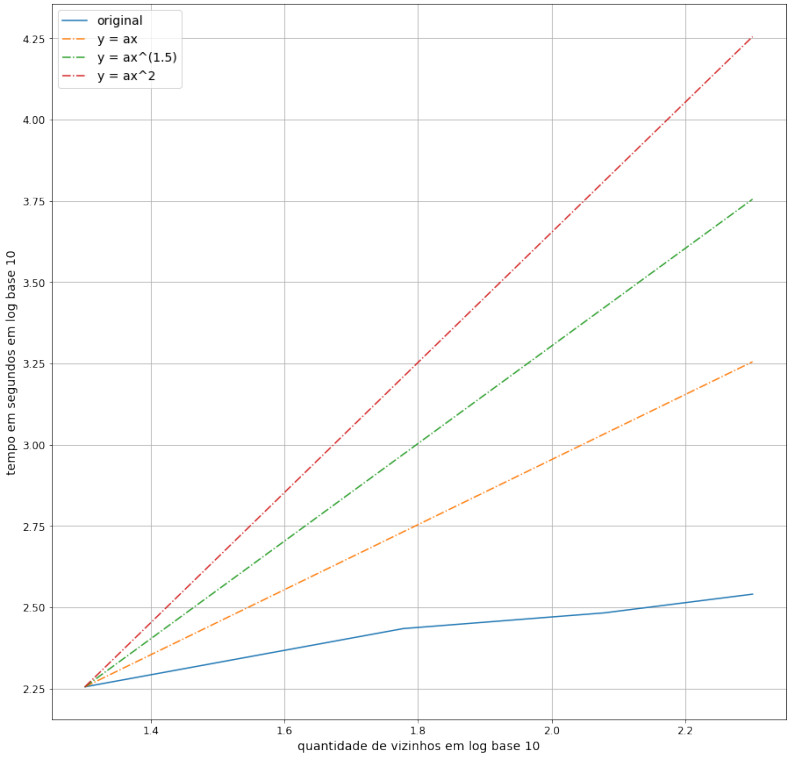
A implementação do framework considera apenas RSOs com relacionamentos bidirecionais. Contudo, o conceito do *RealRank* também se aplica para redes de relacionamentos unidirecionais. A mudança no tratamento dos dois tipos de rede está na forma em que o framework trata os relacionamentos na hora do cálculo de similaridades e na propagação de confiança. Para o caso de redes sociais direcionadas, para todos os cálculos envolvendo o grau do vértice e sequências, o arcabouço deverá considerar seu grau de saída. Isso também se aplica para a média de amigos em comum: a quantidade de vizinhos em comum com outro vizinho será calculada a partir das arestas de saída dos dois vértices. Todo o resto do fluxo do framework será o mesmo para ambos tipos de rede.

Para avaliar a complexidade computacional do framework em seu tempo de execução, foram utilizadas algumas RSOs geradas utilizando o modelo Barabási–Albert (BA) [18]. Esse modelo pode representar um pouco a estrutura de uma rede social, visto que a probabilidade de uma aresta existir está ligada diretamente ao grau do vértice (*Preferential attachment*). A imagem 3.9a mostra os tempos de execução do framework para redes aleatórias com diferentes tamanhos n: 10, 100, 1.000, 10.000 e 100.000. Além disso, o valor de m foi fixado em 3 (três) para todos os valores de n, ou seja, cada vértice novo do modelo aleatório irá se conectar em outros três vértices, seguindo o conceito de *preferential attachment*. Já na figura 3.9b, foram geradas redes aletórias com $n = 1.000$ e diferentes valores de m: 20, 60, 120 e 200. Pode-se observar que o modelo não tem sua complexidade dependente da média de vizinhos em comum da rede, e possui uma relação direta com o tamanho da rede, como esperado na seção 3.1. O tempo de execução t fica entre as retas $y = ax$ e $ax^{1.5}$, para x variando em n. Por outro lado, o tempo de execução fica abaixo da reta de $y = ax$, para x variando em m. Para o conjunto de treino, nos dois casos, foram geradas aleatoriamente $2log^2(n)$ classificações binárias (confiáveis e não confiáveis)

iniciais.



(a) Tempo de execução variando a quantidade de vértices em uma rede aleatória BA com m=3



(b) Tempo de execução variando a quantidade de vizinhos em uma rede aleatória BA com n=1.000

Figura 3.9: Tempos de execução em redes aleatórias BA

Uma versão da implementação está disponível no repositório RealRank no GitHub, *https://github.com/caiocrr/RealRank*.

Capítulo 4

Estudo de caso: Steam

A rede de jogos e de relacionamento da Steam possui mais de 100 milhões de usuários. Dentre eles, podemos ter usuários em suas contas originais, usuários em contas secundárias e usuários maliciosos. Contas originais são aquelas que a sua conta Steam está relacionada diretamente à identidade do usuário. Contas secundárias são aquelas em que o usuário já possui uma conta da Steam original, porém utiliza uma conta secundária por algum motivo, seja para jogar em um nível competitivo diferente de sua conta original (*smurf*), para jogar sem ser reconhecido ou até mesmo para atacar outros usuários, roubando itens de jogo ou a própria conta. Pode-se chamar esse último como usuário malicioso: aquele perfil da Steam que só é utilizado para tirar proveito de outros membros da rede. Não necessariamente um usuário malicioso será um usuário secundário e vice-versa. Além disso, usuários originais também podem ser maliciosos, ou seja, são usuários originais não confiáveis. Em todo caso, finalmente, tem-se duas classificações na rede da Steam: usuários confiáveis e usuários não confiáveis.

O problema se concentra em definir um ranqueamento alto para usuários confiáveis e um ranqueamento baixo para todos os usuários não confiáveis na rede da Steam, que não necessariamente são maliciosos. Esse conjunto inclui usuários secundários que utilizam as contas sem nenhum objetivo maldoso, como é o caso de contas *smurfs*, como também incluirá as contas originais que são maliciosas. Vale lembrar que essas informações de identidade são apenas para definir melhor o domínio de classificação da rede na análise realizada. Portanto, elas não são visíveis para o framework, apenas a estrutura da rede e o conjunto de treino oferecido como entrada.

A sub-rede da Steam obtida (através de sua API) contém 114.047 usuários e só possui uma componente conexa. Ou seja, todos os usuários de nossa sub-rede estão conectados.

Para realizar a classificação inicial (conjunto de treino), foi necessária a análise de um especialista. Isso é, uma pessoa com experiência na rede da Steam para

classificar os usuários como confiável ou não confiável a partir de qualquer dado ou informação que consiga obter de cada usuário. Para tal, foi desenvolvida uma aplicação, evidenciada na figura 4.1, a fim de oferecer a visualização da conta através da própria plataforma da steam (website), no qual o especialista acessa, analisa todos os dados públicos apresentados na plataforma, e decide se o usuário é confiável ou não confiável. Se houver dúvida por parte do especialista, o usuário não é classificado e é descartado do conjunto de treino.

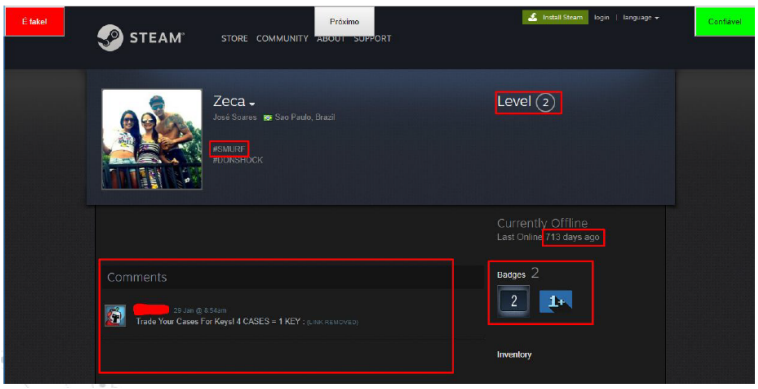


Figura 4.1: Aplicação desenvolvida para análise do especialista.

Foram classificados 654 usuários por especialista (confiáveis e não confiáveis), ou seja, cerca de 0.5% da rede. O conjunto é desbalanceado, sendo 146 (0.1%) usuários não confiáveis e 508 (0.4%) confiáveis. Para realizar essa classificação, foram analisadas cerca de 3.000 contas de usuários, selecionados de maneira aleatória na rede. Nessa seção, serão comparados o resultado obtido pelo framework desse estudo e o resultado obtido por classificação supervisionada utilizando dados específicos de conta, de usuário e da rede.

4.1 Resultados obtidos pelo framework RealRank

Para avaliar o ranqueamento final de confiança dos usuários, foram aplicadas diferentes combinações de parâmetros de entrada: diferentes combinações de modelos de classificação supervisionada e seus hiperparâmetros, com diferentes projeções da rede e taxas de propagação de confiança. Além disso, também foram aplicadas duas formas de análise: resultado no conjunto de treino completo e resultado com validação cruzada. Para cada um dos resultados obtidos, foram verificadas as métricas de acurácia balanceada e *F-Measure*, tanto em validação cruzada quanto no resultado final. A tabela 4.1 descreve alguns exemplos de resultados sobre o conjunto de treino para diferentes parâmetros de entrada do *framework*.

Tabela 4.1: Resultados obtidos pelo RealRank. B.A. é a acurácia balanceada.

p_c	p_f	Modelo	Hiperparâmetros	Threshold	F-measure (classe negativa)	B.A.
0.25	0.75	knn	nneigh = 1	0.006	0.71	0.79
1.00	1.00	svm	C = 1, kernel = poly	0.0	0.66	0.84
1.00	1.00	lr	C = 1	0.405	0.65	0.80
0.75	1.00	knn	nneigh = 3	0.0	0.64	0.75
0.00	1.00	lr	C = 1	0.496	0.63	0.81
0.00	1.00	svm	C = 1, kernel = linear	0.0	0.62	0.81
0.00	1.00	knn	nneigh = 3	0.0	0.62	0.75
0.50	1.00	knn	nneigh = 3	0.0	0.61	0.73
1.00	1.00	svm	C = 1, kernel = RBF	0.0	0.61	0.79
0.25	1.00	knn	nneigh = 3	0.0	0.60	0.73

Para a avaliação da métrica de F-Measure, foi tratada como classe positiva a classe em minoria: a de não confiáveis. Isso foi necessário pois a F-Measure é uma média harmônica entre o *recall* e a *precisão* da classe positiva, como pode-se ver nas equações em 4.1. Portanto, se o conjunto de treino possuir uma quantidade de classificações negativas muito maior que a positiva, a F-measure será uma boa métrica de avaliação. Por outro lado, a acurácia balanceada não se altera conforme trocamos os *labels* de classificação.

$$\begin{aligned} &\text{F-measure} = \left(\frac{recall^{-1}+precision^{-1}}{2}\right)^{-1} = 2\left(\frac{precision*recall}{precision+recall}\right) \\ &\text{Balanced accuracy} = \frac{1}{2}\left(\frac{VP}{P} + \frac{VN}{N}\right) \end{aligned} \tag{4.1}$$

O cálculo de resultados baseado em validação cruzada deverá ser realizado após a última etapa do *framework*, a propagação de confiança. Portanto, a validação cruzada foi calculada da seguinte maneira: na segunda etapa, ao realizar o treino do modelo supervisionado com o conjunto de treino de entrada, são selecionadas, a partir desse conjunto, as devidas proporções para teste e treino. A partir disso, os modelos serão treinados com os conjuntos de treino selecionados e os resultados serão calculados somente após a propagação da terceira etapa, utilizando os conjuntos de testes selecionados na segunda etapa. A figura 4.2 define o fluxo para o cálculo da validação cruzada do *framework*.

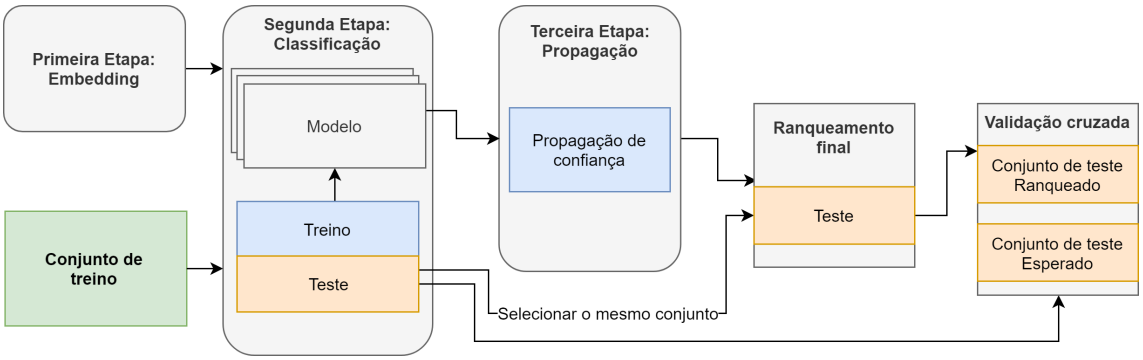


Figura 4.2: Fluxo utilizado para o cálculo da validação cruzada.

A figura 4.3 mostra alguns dos melhores resultados de acurácia balanceada em validação cruzada com 10 (dez) *folds*. Nela estão descritos alguns parâmetros de entrada, tais como o parâmetro do peso das similaridades p_c, parâmetro para propagação de confiança p_f, modelo utilizado para a classificação e alguns de seus hiperparâmetros.

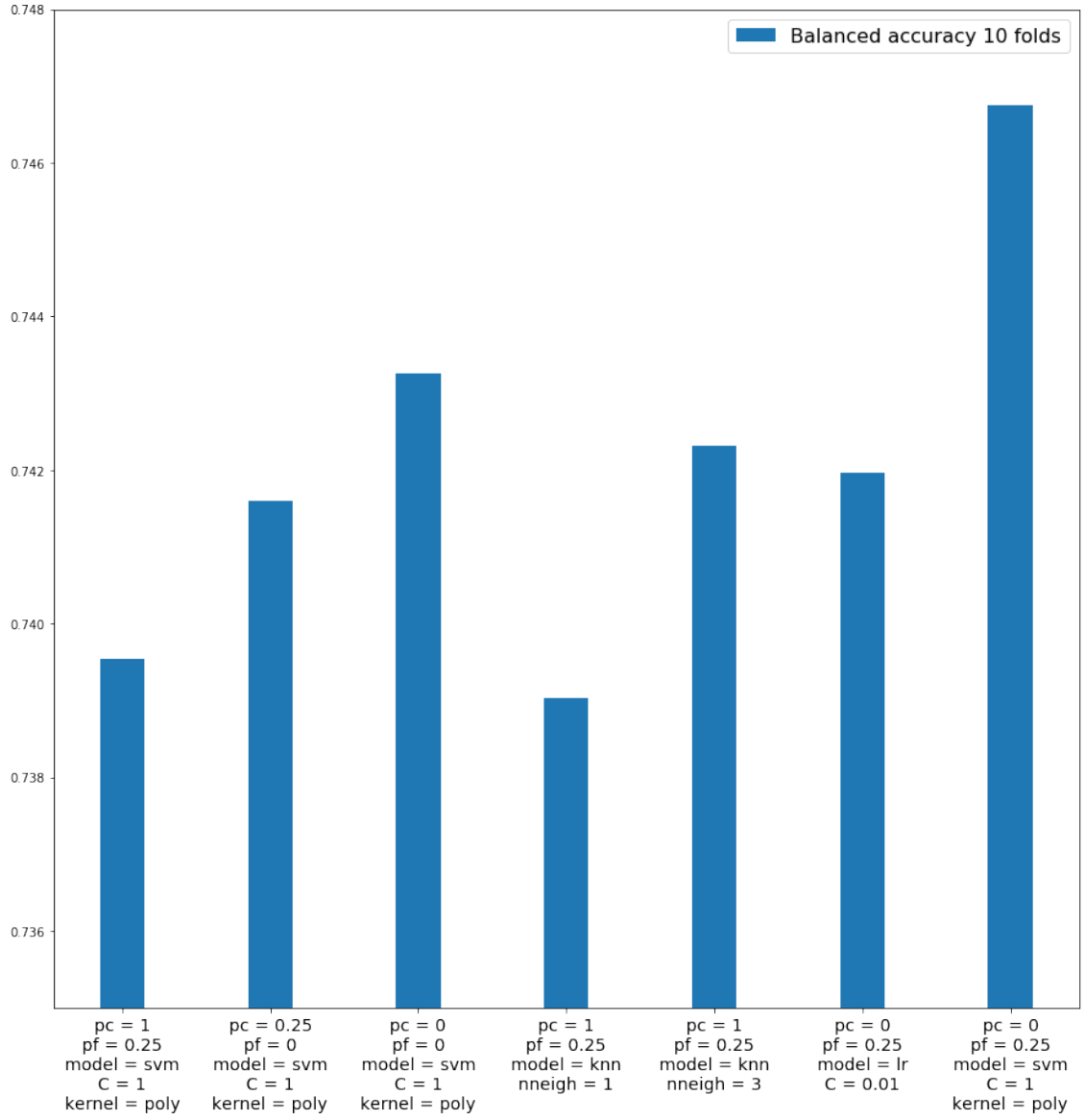


Figura 4.3: Acurácia balanceada em validação cruzada na rede da Steam para diferentes parâmetros de entrada no RealRank.

Dependendo da combinação dos parâmetros de entrada do framework, é possível obter como resultado tanto um ranqueamento final, onde o valor é um número real entre 0 e 1, quanto uma simples classificação, onde a saída se limita em 0 ou 1. Para os casos onde é obtido um ranqueamento, é possível aplicar diferentes *thresholds* para determinar uma classificação final, podendo assim avaliar o resultado utilizando alguma métrica, como nas figuras 4.4, 4.5 e 4.6. Por outro lado, se for obtida diretamente a classificação (0 ou 1), será possível calcular o score da métrica de forma direta. Para a avaliação de resultados, foi aplicada uma sequência de threshold de classificação no ranqueamento previsto para visualizar o melhor valor de cada métrica.

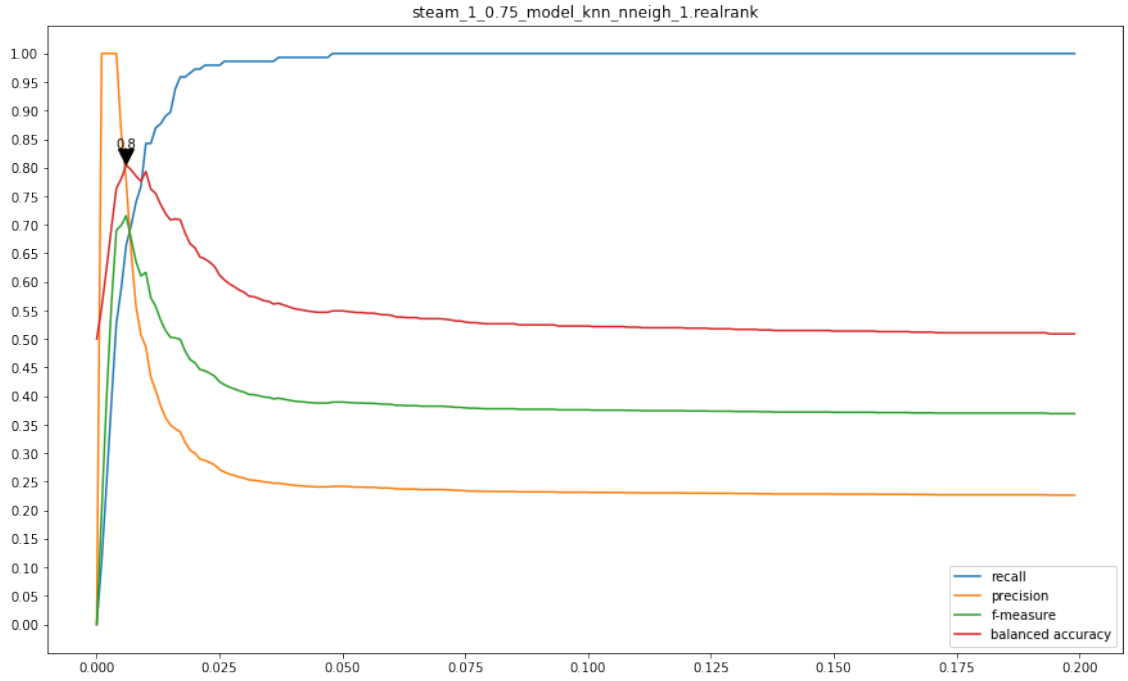


Figura 4.4: Scores (eixo y) no conjunto de treino por diferentes thresholds de classificação (eixo x) em um modelo KNN

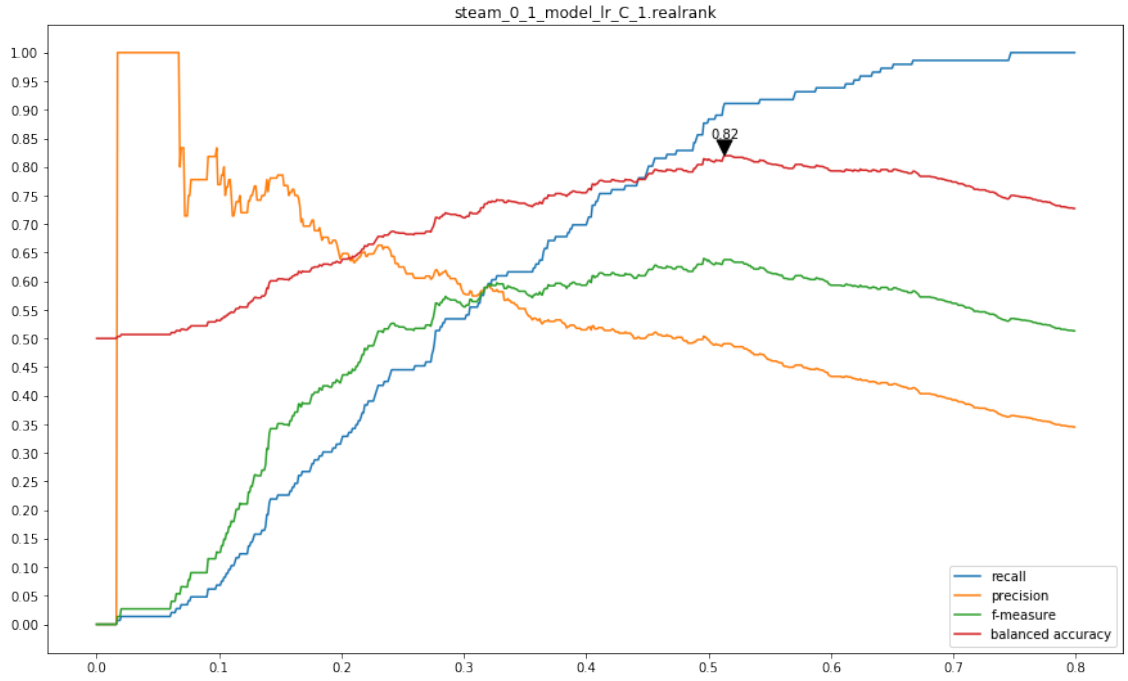


Figura 4.5: Scores (eixo y) no conjunto de treino por diferentes thresholds de classificação (eixo x) em um modelo LR

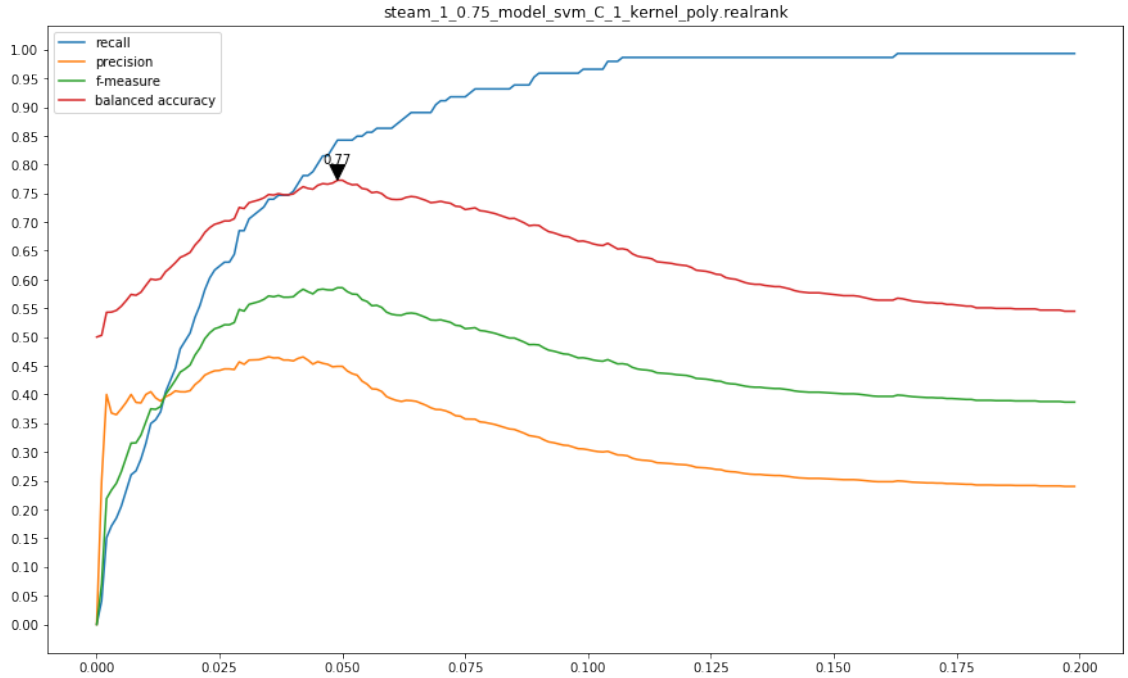


Figura 4.6: Scores (eixo y) no conjunto de treino por diferentes thresholds de classificação (eixo x) em um modelo SVM

Para cada um dos três modelos implementados nesse estudo, foram analisados os melhores resultados a partir da validação cruzada da acurácia balanceada, como é possível ver nas figuras em 4.7, 4.8 e 4.9. Para isso, foram testados diferentes parâmetros de entrada do framework.

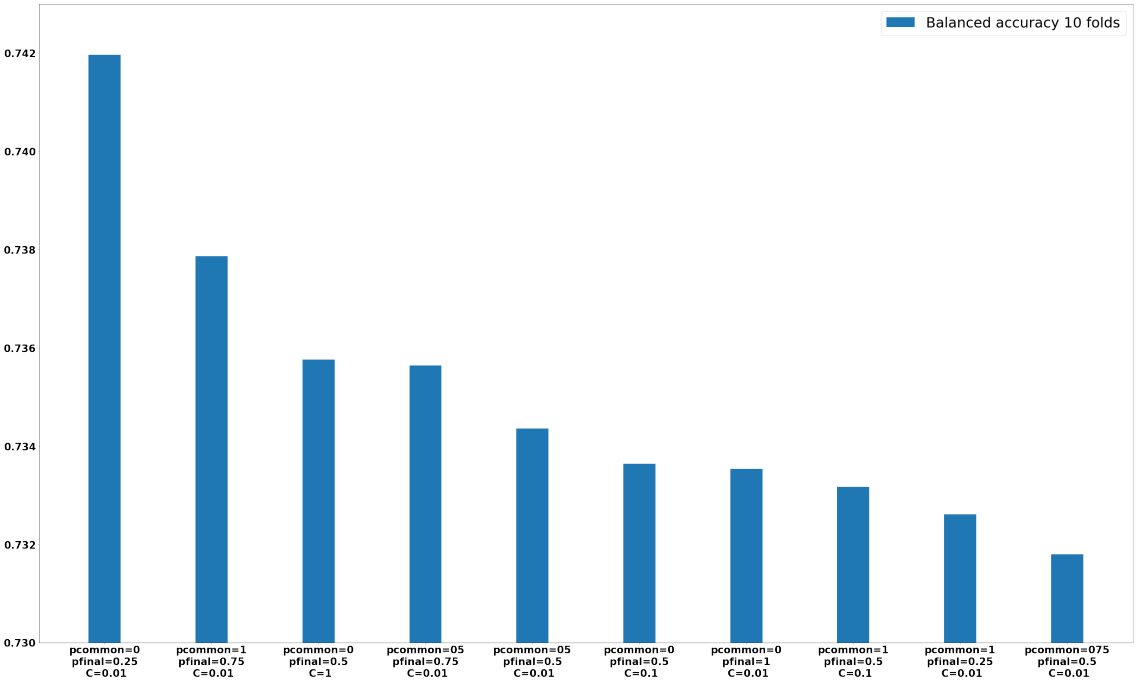


Figura 4.7: Melhores resultados em Regressão Logística

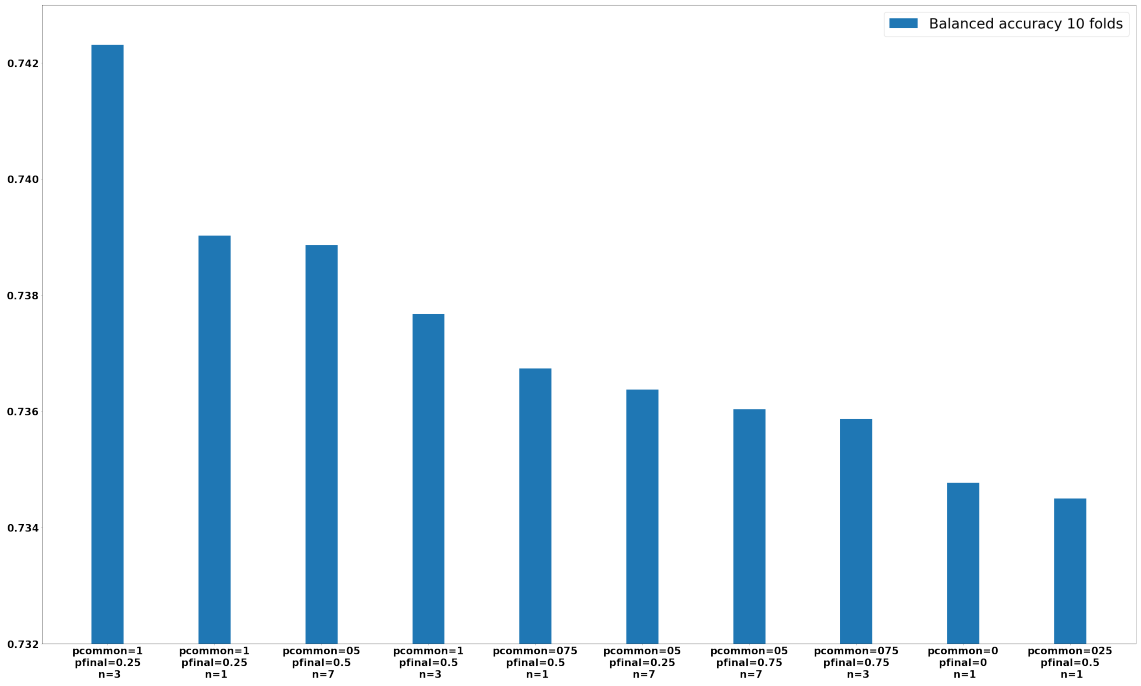


Figura 4.8: Melhores resultados com KNN

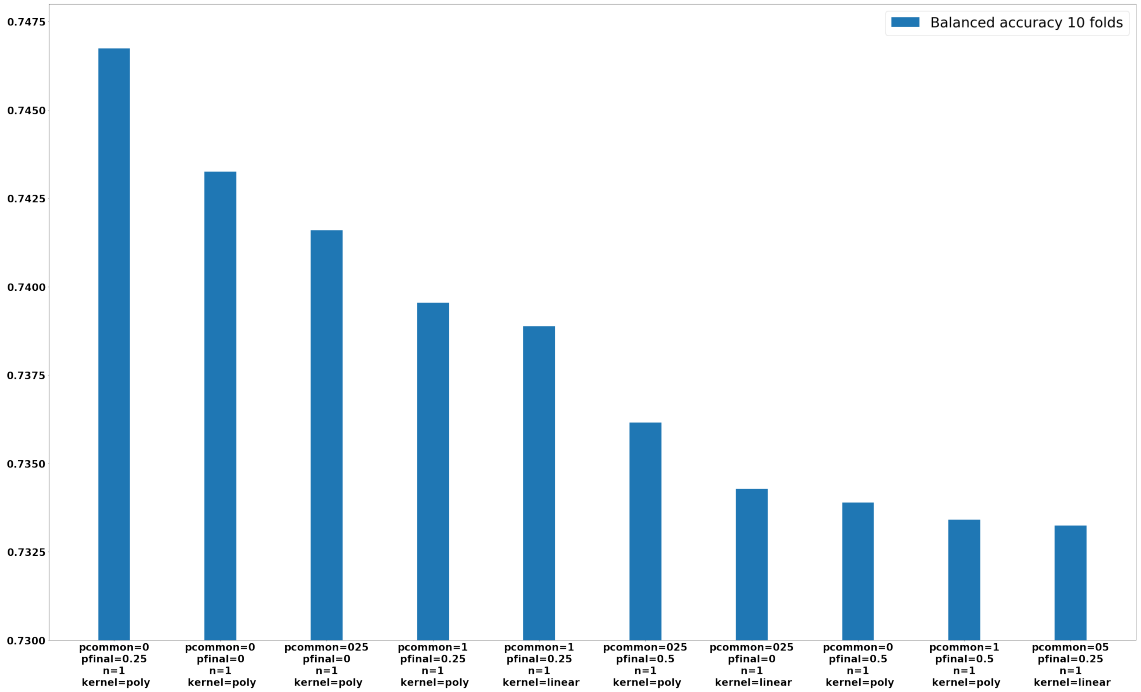


Figura 4.9: Melhores resultados com SVM

Foram comparados os melhores resultados dos três modelos implementados para a acurácia balanceada e *f-measure* sobre o conjunto de treino. A partir dos resultados na figura 4.10, pode-se reparar que uma combinação específica de entradas do framework com o modelo KNN, parâmetro de similaridade $p_c = 0.25$ e propagação $p_f = 0.75$, obteve um melhor resultado sobre o conjunto de treino para as

duas métricas avaliadas. É claro que, para cada estudo de caso, a escolha do melhor resultado dependerá da métrica avaliada e, consequentemente, do critério de avaliação escolhido. Por outro lado, pode-se descartar alguns resultados frutos de combinações de entradas do framework que retornam, por exemplo, uma acurácia balanceada por volta de 50% ou um *recall* igual a 1.

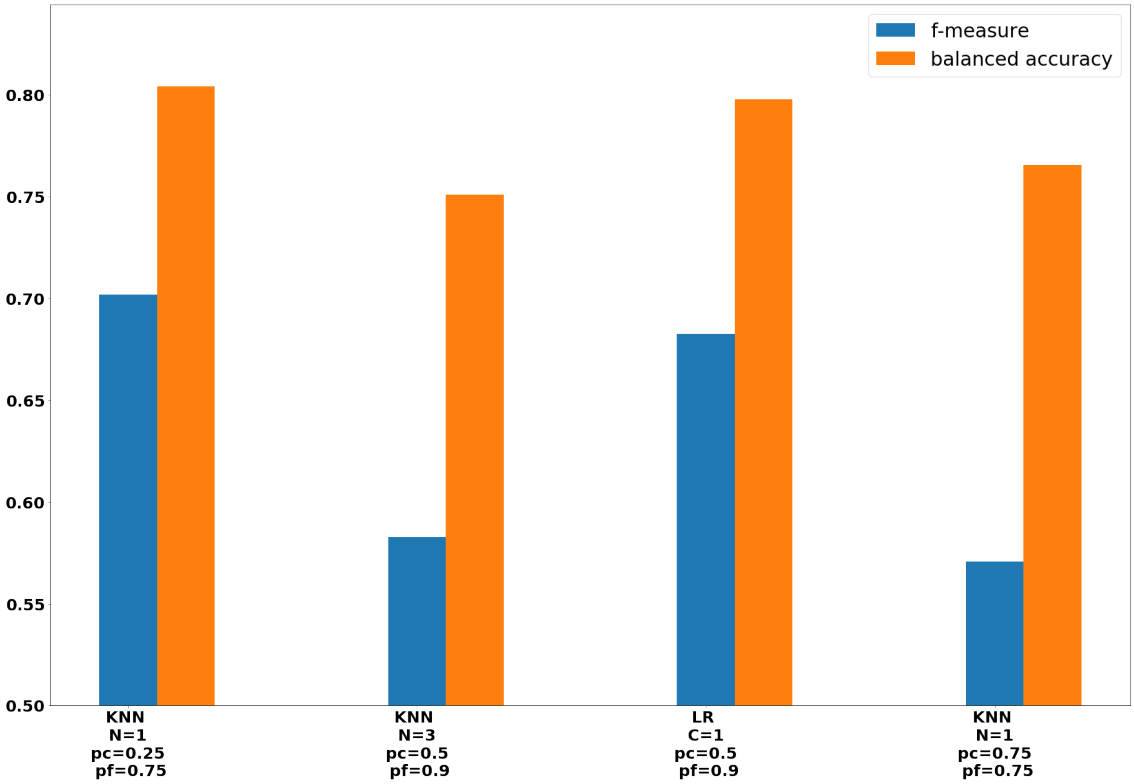
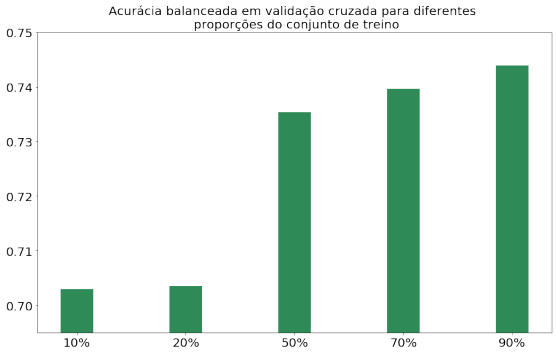
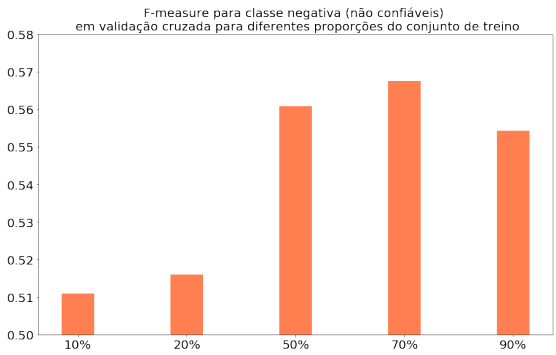


Figura 4.10: Alguns resultados sobre o conjunto de treino. O eixo x representa o modelo e alguns parâmetros para o framework. O eixo y representa a acurácia balanceada avaliada sobre o conjunto de treino.

Foi realizada a comparação do resultado em validação cruzada para diferentes tamanhos do conjunto de treino, com o objetivo de visualizar melhor o tamanho do conjunto necessário para treino do modelo a fim de obter uma boa classificação. Nas figuras 4.11a e 4.11b, pode-se observar a média de resultados em validação cruzada com 10%, 20%, 30%, 50%, 70% e 90% do conjunto de treino. Para todos os casos, a validação cruzada foi realizada em 10 *folds*, utilizando os outros 10% do conjunto de treino, de maneira a não repetir os dados selecionados para o conjunto utilizado no treino. Pode-se observar também que o resultado de acurácia balanceada melhora proporcionalmente ao tamanho do conjunto de treino utilizado. Já o resultado de *f-measure* tem uma leve piora utilizando 90% do conjunto de treino. Isso pode ser consequência dos parâmetros de similaridade p_c e de propagação p_f, visto que o conjunto de treino oferecido pode possuir vértices similares estruturalmente com classificações diferentes.



(a) Acurácia balanceada em validação cruzada para diferentes tamanhos de conjunto de treino.



(b) F-measure em validação cruzada para diferentes tamanhos de conjunto de treino.

Figura 4.11: Avaliação de métricas utilizando diferentes proporções do conjunto de treino.

4.2 Resultados obtidos por modelos de classificação utilizando dados específicos de usuários

A classificação e identificação de usuários não confiáveis em RSO através de modelos supervisionados utilizando dados cadastrais, de conta e de atividades é um dos modelos mais utilizados atualmente. O problema disso é a necessidade de um vasto domínio de dados de usuários, processamento de tais dados e a produção de um modelo supervisionado bem treinado. Para cada rede social, por exemplo, devemos coletar o máximo de informação possível de seus usuários e fazer seu devido pré-processamento, que será específico a cada diferente rede social a ser avaliada. Isso acaba diferindo bastante do fluxo de classificação do *RealRank*, o qual utiliza de uma solução genérica, podendo ser aplicada à qualquer rede social sem o acesso aos dados específicos.

Para fins de comparação, foi realizado um estudo sobre a rede social da Steam para a classificação de usuários não confiáveis a partir de suas informações pessoais, de conta e de jogos. Os dados dos usuários coletados para esse estudo de caso estão descritos na tabela 4.2. Nessa seção iremos descrever de maneira superficial o pré-processamento realizado e os resultados das classificações obtidas.

Tabela 4.2: Dados de usuários coletados para o estudo de caso da Steam. Inicialmente foram coletadas 31 variáveis.

Variável	Descrição
steamid	Identificador de conta. Não será utilizado no processamento.

communityvisibilitystate	Status de visiblidade da conta. Como todos os usuários que foram registrados possuem o mesmo status de visibilidade (Visível), a variável possui variância 0 e não será utilizada no processamento.
profilestate	Indica se o usuário possui o perfil configurado.
commentpermission	Indica o nível de permissão para realizar comentários em sua conta.
timecreated	Data de criação da conta
loccountrycode	Código do país do usuário
locstatecode	Código do estado do usuário
loccityid	Código da cidade do usuário
friendcount	Quantidade de amigos
gamecount	Quantidade de jogos adquiridos
commonfriends	Proporção entre média de amigos em comum e quantidade de amigos
closeness	Métrica que define o quanto o vértice (usuário) é próximo de todos os outros vértices da rede.
betweenness	Métrica que define o quanto o vértice (usuário) se encontra nos mínimos caminhos entre quaisquer dois vértices da rede.
medfriendsince	Média de tempo de amizade para todos os amigos
maxfriendsince	Valor máximo de tempo de amizade entre todos os amigos
stdfriendsince	Devio padrão do tempo de amizade entre todos os amigos
avatarisset	Define se o usuário possui uma foto de avatar definida
sumplaytimeforever	Somatório de minutos jogados para todos os jogos
maxplaytimeforever	Máximo de minutos jogados entre todos os jogos
numgamesplayed	Número de jogos jogados

nameisset	Define se o usuário possui um nome configurado para a conta
medprice	Média de preço de jogos adquiridos
qtdgroups	Quantidade de grupos que o usuário participa
playerlevel	Level da conta Steam do usuário
qtdbadges	Quantidade de badges adquiridos pelo usuário
meanlevelbadges	Level médio das badges adquiridas
maxlevelbadges	Level máximo das badges
CommunityBanned	Define se o usuário foi banido da comunidade
NumberOfVACBans	Quantidade de bans VAC
NumberOfGameBans	Quantidade de game bans
EconomyBan	Define se o usuário foi banido da rede de economia da Steam

4.2.1 Pré-processamento

Para as variáveis de entrada, foram incluídas as seguintes etapas no pré-processamento: verificação de outliers, verificação dos dados ausentes, normalização da distribuição das variáveis contínuas, verificação de correlação entre variáveis, deleção e transformação de variáveis e a discriminação de variáveis categóricas. No final dessa etapa, a quantidade de variáveis transformadas e discretizadas passaram de 31 (trinta e um) para 91 (noventa e um).

A distribuição da maioria das variáveis contínuas possuiam, inicialmente, um alto viés. Portanto, para analisar possíveis outliers, foi necessária a normalização por *log* da distribuição através da Transformação de BoxCox [19]. Após a transformação, é possível visualizar melhor a distribuição dos dados e identificar fortes candidatos à *outliers*. A figura 4.12 representa algumas variáveis antes e depois da tranformação, enquanto a figura 4.13 representa o antes e o depois da remoção de um possível outlier do *dataset* coletado.

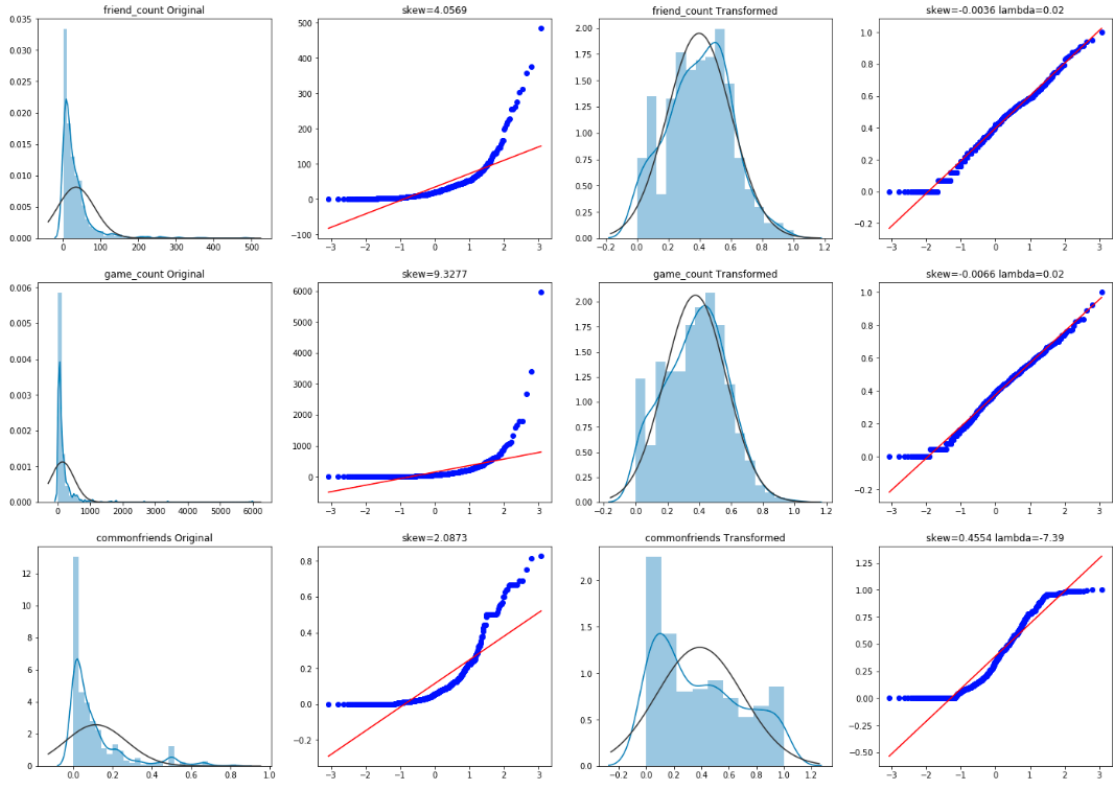


Figura 4.12: Transformação Box-Cox de 4 variáveis contínuas de entrada. As duas primeiras colunas representam a distribuição normal e gráfico Q-Q antes da transformação enquanto as duas últimas demonstram os mesmos gráficos depois da transformação.

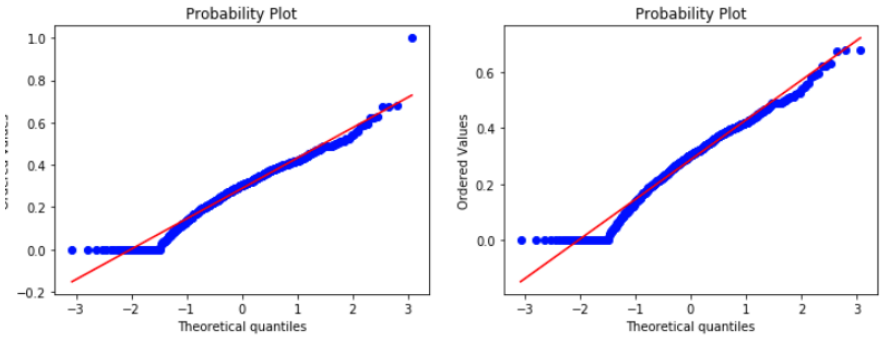


Figura 4.13: Exemplo de candidato à outlier em uma variável contínua transformada.

Como a saída é binária, a correlação entre variáveis foi analisada a partir de histogramas em vez de uma matriz de correlação. Na figura 4.14, pode-se perceber que as variáveis numéricas como *friendcount*, *gamecount*, *maxlevelbadges* e *playerlevel* possuem uma forte correlação com a variável de saída *naoconfiavel*. Para essas variáveis, por exemplo, a região de não confiáveis se instala nas áreas de menor valor das variáveis numéricas citadas.

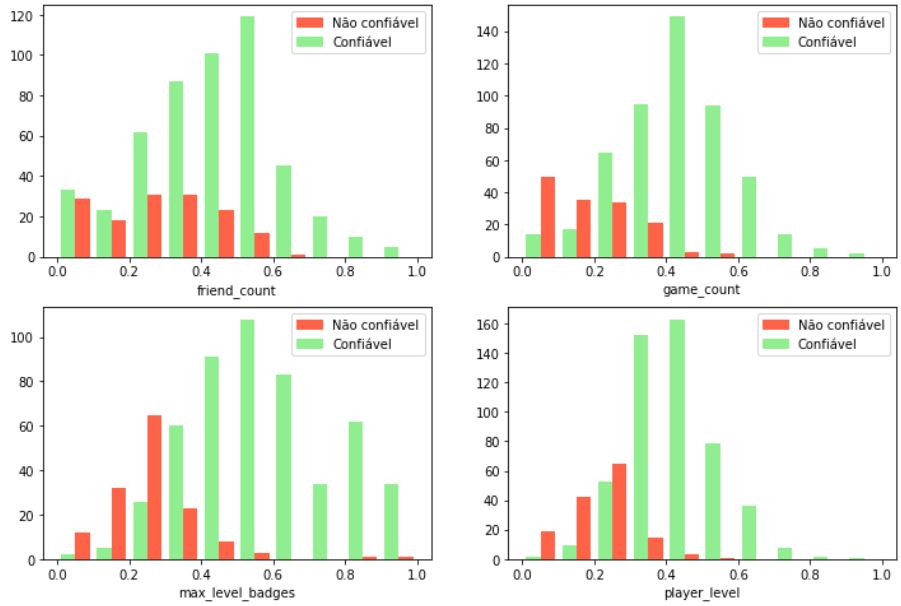


Figura 4.14: Correlação entre variáveis contínuas e saída binária.

Algumas variáveis foram deletadas por não possuir influência na classificação e outras foram transformadas a fim de aumentar tal influência. A variável *communityvisibilitystate*, por exemplo, possui variância nula e foi deletada do conjunto de dados. Já a variável *timecreated*, inicialmente representada por segundos em formato de data UNIX, foi transformada para o mês da data (*mescriacao*). Além disso, foram verificados os dados faltantes do *dataset* para o devido tratamento, como pode ser observado na figura 4.15.

	Missing Ratio
loccityid	56.375
locstatecode	39.939
loccountrycode	21.352
max_level_badges	2.151
mean_level_badges	2.151
qtd_badges	2.151
player_level	1.229
qtd_groups	1.229

Figura 4.15: Porcentagem de dados faltantes do dataset.

Para as variáveis *loccityid*, *locstatecode* e *loccountrycode*, que são categóricas, foi criada uma nova categoria "Vazio" para os dados não preenchidos. Já os dados numéricos faltantes, como os das variáveis *maxlevelbadges*, *meanlevelbadges*, *qtdbadges*, *playerlevel* e *qtdgroups*, foram preenchidos com valor 0 (zero).

As variáveis *loccityid* e *locstatecode* foram descartadas do dataset, pois possuem um alto índice de valores nulos e não agregam positivamente para o resultado da

classificação.

Na última penúltima etapa do pré-processamento, as variáveis categóricas foram transformadas em várias outras variáveis binárias, como é o caso da variável *loccountrycode*. Essa variável possui 41 possíveis categorias distintas. Portanto, foram criadas 41 variáveis binárias, onde cada uma representa de forma binária cada categoria possível da variávei original.

Ao final do pré-processamento, foi realizada uma etapa de remoção de features para um melhor resultado. Essa etapa utiliza um conceito de força bruta sobre testes de resultados de alguma métrica, em validação cruzada. Desse modo, variáveis menos importantes são removidas a fim de aumentar o score final. Para tal, foi utilizado o Recursive Feature Elimination (RFE) [20]: um algoritmo recursivo onde, em cada etapa, é removido um certo número (*step*) de atributos do *dataset* de acordo com o valor do seu parâmetro (importância) no modelo linear definido como parâmetro do RFE. Para a seleção de *features*, foi utilizado o modelo de regressão logística. O modelo nos retorna os coeficientes do modelo treinado através do atributo *coef_*. Além disso, foi utilizado o passo (*step*) igual a 1 (um), de modo que seja removida apenas uma feature por vez, em validação cruzada de 10 *folds* utilizando métrica F-Measure. O melhor score obtido de um dataset reduzido de 83 (oitenta e três) para 40 (quarenta) variáveis, retornando uma melhora de 1% no resultado final. A figura 4.16 uma possível iteração de remoção de variáveis com o modelo de regressão logística. Após a remoção, o *score* de métrica selecionada por validação cruzada é escolhida e o resultado é verificado. Se o score for mais alto que a iteração anterior, a variável é removida.

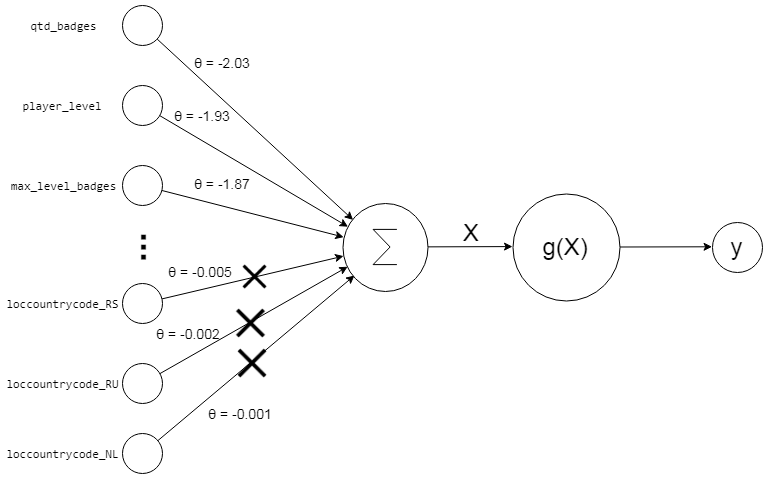


Figura 4.16: Etapa de de remoção de variáveis a partir dos coeficientes de um modelo linear treinado utilizando RFE.

4.2.2 Classificação e Resultado

Na etapa de classificação, alguns modelos lineares e não lineares foram testados, com diferentes hiperparâmetros e analisando diferentes métricas. Dos modelos lineares, foram treinados a Regressão Logística, SVM linear e Perceptron [21]. Já dos não lineares, foram avaliados a árvore de decisão simples, Floresta Aleatória [22], SVM polinomial, SVM RBF e Perceptron com multi-camadas. As combinações de hiperparâmetros que retornaram o melhor resultado em cada modelo foram selecionadas. Após isso, o modelo que obteve a melhor avaliação foi selecionado para posteriormente ser comparado ao RealRank.

Para a comparação dos resultados, foram utilizadas as médias das métricas recall, precisão, acurácia balanceada e F-Measure para a classe negativa (não confiável) em validação cruzada com 10 *folds*. Como a F-Measure é uma média harmônica entre recall e precisão, ela foi utilizada para a escolha do melhor resultado.

A figura 4.17 demonstra o resultado final dos modelos treinados. O modelo Regressão Logística retornou a melhor classificação de acordo com a *F-Measure*, enquanto o SVM com *kernel* RBF retornou o melhor resultado de acurácia balanceada. O modelos Random Forest também retornou bom resultado.

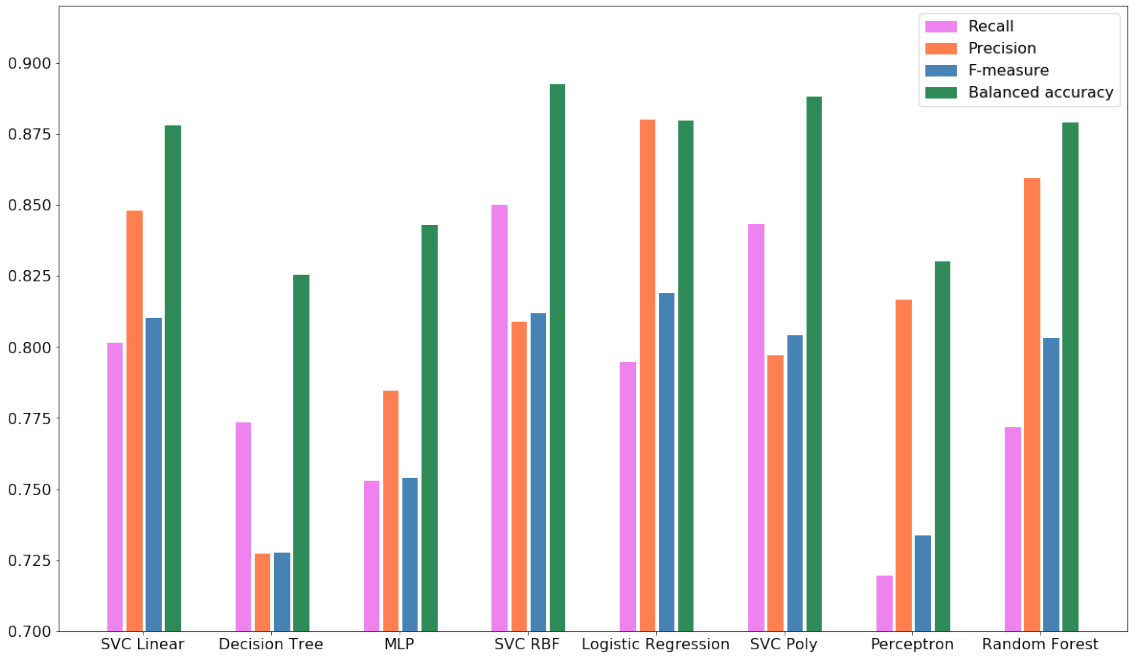


Figura 4.17: Avaliação de métricas para modelos testados em validação cruzada com 10 folds.

4.3 Comparação de resultados

Existem duas interpretações importantes para a avaliação dos resultados: considerar a classe de não confiáveis como mais importante ou considerar a classe de confiáveis

como mais importante. Deve-se lembrar que o conjunto de treino é desbalanceado, onde a classe negativa (não confiáveis) tem a menor proporção. Portanto, se a classe negativa for considerada como mais importante, devemos levar em consideração as métricas *recall*, precisão e, consequentemente, *F-Measure* aplicadas à classe negativa. Dessa maneira, ao obter um bom resultados nessas métricas, pode-se ter a certeza de que o ranqueamento final garante baixos *scores* apenas para aqueles que não são confiáveis. Isso não ocorre para o *RealRank* para a rede analisada, enquanto ocorre para o classificador da seção 4.2. Por outro lado, se a classe de confiáveis for considerada como a mais importante, de modo a garantir um ranqueamento alto para a maioria dos usuários confiáveis e um ranqueamento baixo para a maioria dos usuários não confiáveis, a métrica de acurácia balanceada deverá ser avaliada. *F-measure* para esse caso não se aplica devido ao desbalanceamento do conjunto de treino (classe positiva é maioria). Por essa interpretação, *RealRank* retorna um bom resultado quando comparado ao classificador por dados específicos da rede, como pode-se ver na figura 4.18.

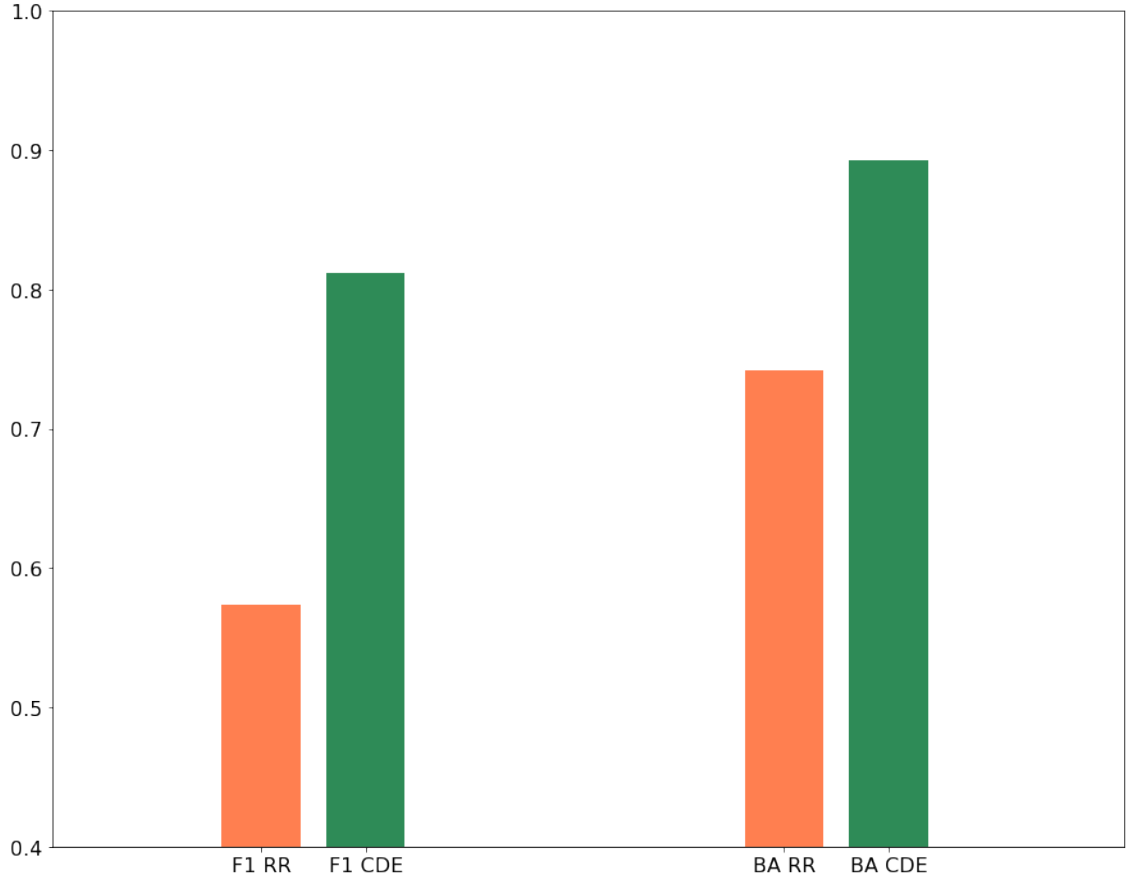


Figura 4.18: Comparação dos resultados dos dois classificadores para F-Measure e Acurácia Balanceada em validação cruzada (10 folds 90/10). BA=Balanced Accuracy. F1 = F-Measure. RR = RealRank. CDE = Classificador com Dados Específicos.

Capítulo 5

Conclusão e Trabalhos Futuros

O diferencial do RealRank é considerar apenas a estrutura da rede para realizar a classificação da reputação dos usuários. A estrutura de uma rede pode oferecer muitas informações, que podem ser extraídas como propriedades da rede. Nesse trabalho, foram utilizadas apenas duas propriedades das inúmeras possíveis. Conseguir identificar e extrair dados a partir estrutura da rede pode ajudar em diversos outros problemas de classificação.

A identificação de usuários não confiáveis ou confiáveis em redes sociais online não possui uma solução genérica. Isso acontece pois cada rede social irá possuir uma estrutura diferente, com usuários que se comportam de maneiras diferentes na rede. Informações de cada usuário e de suas atividades serão completamente distintas em diferentes redes sociais. Além disso, o objetivo de usuários *fakes* em cada rede também diverge, fazendo com o que o usuário tenha um outro comportamento na rede e, consequentemente, uma outra estrutura hierárquica. Sabendo disso, a classificação de usuários utilizando informações específicas de cada rede, ou seja, dados cadastrais de usuário ou atividade, acabam por oferecer melhores resultados para as métricas aqui avaliadas. Por outro lado, o RealRank pode oferecer um bom resultado de uma forma bem mais genérica, podendo ser aplicado em qualquer rede social sem a necessidade de dados externos a rede, utilizando-se apenas da estrutura de relacionamentos da rede social.

No estudo de caso realizado, RealRank obteve um resultado inferior na predição de usuários confiáveis e não confiáveis quando comparado aos modelos de aprendizado que utilizam dados específicos de usuários da rede. Por outro lado, RealRank é mais simples de ser aplicado em qualquer outra rede social online, pois requer um conjunto de dados de entrada menor. Na seção de trabalhos futuros 5.1, serão indicados alguns possíveis pontos de melhoria para o framework, a fim de obter melhores resultados.

5.1 Trabalhos Futuros

O RealRank considera apenas a estrutura da rede para realizar um ranqueamento de reputação dos usuários. Portanto, o cálculo de similaridade estrutural poderia considerar qualquer propriedade que possa ser extraída da estrutura da rede. Este estudo utilizou algumas premissas que se referem ao comportamento de usuários confiáveis e não confiáveis. Por exemplo, foi considerado que usuários não confiáveis ou confiáveis tendem a ter uma similaridade estrutural hierárquica por grau e por vizinhos em comum parecida entre si. Estudar melhor o comportamento de usuários em redes sociais e observar novas premissas relacionadas à estrutura da rede que podem estar diretamente ligadas à esse comportamento certamente irá ajudar a melhorar os resultados obtidos. De fato, existem propriedades de redes sociais não mapeadas nesse estudo que relacionam diretamente usuários confiáveis com outros confiáveis ou não confiáveis com outros usuários não confiáveis. Identificar essas propriedades e aplicá-las no cálculo das similaridades da primeira etapa do *framework*, como foi realizado com a hierarquia de graus e vizinhos em comum nesse trabalho, é um caminho indicado para a continuação desse trabalho.

A propagação da confiança é uma etapa que pode ajudar ou piorar o resultado obtido. Outra possível continuação para este trabalho é propagar a confiança de maneira não-uniforme. Para isso, seria necessário calcular pesos que serão considerados nessa propagação. Esse cálculo poderia ser feito considerando apenas a estrutura da rede, dados do conjunto de treino recebido ou uma combinação de ambos.

O RealRank pode ser alterado para receber outros dados de entrada, além da estrutura da rede. Desse modo, o *framework* irá considerar a estrutura da rede em conjunto com outros dados externos para realizar a classificação inicial de confiança. Esses dados podem ser, por exemplo, dados obtidos por sistemas de *reports* ou dados específicos de usuários da rede, como na seção 4.2. Os dois modelos testados no estudo de caso da Steam poderiam ser combinados. Dessa forma, o classificador inicial do RealRank teria como entrada, além da projeção obtida a partir da similaridade hierárquica estrutural da rede, dados específicos de usuários e de atividades.

Referências Bibliográficas

[1] RIBEIRO, L. F., SAVERESE, P. H., FIGUEIREDO, D. R. "Struc2Vec: Learning Node Representations from Structural Identity". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pp. 385–394, New York, NY, USA, 2017. ACM. ISBN: 978-1-4503-4887-4. doi: 10.1145/3097983.3098061. Disponível em: <http://doi.acm.org/10.1145/3097983.3098061>.

[2] CAO, Q., SIRIVIANOS, M., YANG, X., et al. "Aiding the Detection of Fake Accounts in Large Scale Social Online Services". In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pp. 15–15, Berkeley, CA, USA, 2012. USENIX Association. Disponível em: <http://dl.acm.org/citation.cfm?id=2228298.2228319>.

[3] SAB, G. A. A. "Proposta E Avaliação De Um Algoritmo Para Recomendação De Jogos Baseado Em Difusão Em Redes Sociais". 2017.

[4] XIAO, C., FREEMAN, D. M., HWA, T. "Detecting Clusters of Fake Accounts in Online Social Networks". In: *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, AISec '15, pp. 91–101, New York, NY, USA, 2015. ACM. ISBN: 978-1-4503-3826-4. doi: 10.1145/2808769.2808779. Disponível em: <http://doi.acm.org/10.1145/2808769.2808779>.

[5] YANG, Z., WILSON, C., WANG, X., et al. "Uncovering Social Network Sybils in the Wild". In: *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '11, pp. 259–268, New York, NY, USA, 2011. ACM. ISBN: 978-1-4503-1013-0. doi: 10.1145/2068816.2068841. Disponível em: <http://doi.acm.org/10.1145/2068816.2068841>.

[6] BLONDEL, V. D., GUILLAUME, J.-L., LAMBIOTTE, R., et al. "Fast unfolding of communities in large networks", *Journal of Statistical Mechanics:*

Theory and Experiment, v. 2008, n. 10, pp. P10008, 2008. Disponível em: <http://stacks.iop.org/1742-5468/2008/i=10/a=P10008>.

[7] BOSHMAF, Y., LOGOTHETIS, D., SIGANOS, G., et al. "Íntegro: Leveraging Victim Prediction for Robust Fake Account Detection in Large Scale OSNs", *Computers & Security*, v. 61, 06 2016. doi: 10.1016/j.cose.2016.05.005.

[8] PEROZZI, B., AL-RFOU, R., SKIENA, S. "DeepWalk: Online Learning of Social Representations". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pp. 701–710, New York, NY, USA, 2014. ACM. ISBN: 978-1-4503-2956-9. doi: 10.1145/2623330.2623732. Disponível em: <http://doi.acm.org/10.1145/2623330.2623732>.

[9] GROVER, A., LESKOVEC, J. "Node2Vec: Scalable Feature Learning for Networks". In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 855–864, New York, NY, USA, 2016. ACM. ISBN: 978-1-4503-4232-2. doi: 10.1145/2939672.2939754. Disponível em: <http://doi.acm.org/10.1145/2939672.2939754>.

[10] MIKOLOV, T., CHEN, K., CORRADO, G., et al. "Efficient Estimation of Word Representations in Vector Space", *CoRR*, v. abs/1301.3781, 2013. Disponível em: <http://dblp.uni-trier.de/db/journals/corr/corr1301.html#abs-1301-3781>.

[11] SALVADOR, S., CHAN, P. "Toward Accurate Dynamic Time Warping in Linear Time and Space", *Intell. Data Anal.*, v. 11, n. 5, pp. 561–580, out. 2007. ISSN: 1088-467X. Disponível em: <http://dl.acm.org/citation.cfm?id=1367985.1367993>.

[12] COLLINS, M., SCHAPIRE, R. E., SINGER, Y. "Logistic Regression, AdaBoost and Bregman Distances", *Mach. Learn.*, v. 48, n. 1-3, pp. 253–285, set. 2002. ISSN: 0885-6125. doi: 10.1023/A:1013912006537. Disponível em: <https://doi.org/10.1023/A:1013912006537>.

[13] HEARST, M. A. "Support Vector Machines", *IEEE Intelligent Systems*, v. 13, n. 4, pp. 18–28, jul. 1998. ISSN: 1541-1672. doi: 10.1109/5254.708428. Disponível em: <http://dx.doi.org/10.1109/5254.708428>.

[14] COVER, T., HART, P. "Nearest Neighbor Pattern Classification", *IEEE Trans. Inf. Theor.*, v. 13, n. 1, pp. 21–27, set. 2006. ISSN: 0018-9448. doi: 10.

1109/TIT.1967.1053964. Disponível em: <https://doi.org/10.1109/TIT.1967.1053964>.

[15] BUITINCK, L., LOUPPE, G., BLONDEL, M., et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[16] MCKINNEY, W. "pandas: a Foundational Python Library for Data Analysis and Statistics", .

[17] OLIPHANT, T. "NumPy: A guide to NumPy". USA: Trelgol Publishing, 2006–. Disponível em: <http://www.numpy.org/>. [Online; accessed <today>].

[18] BARABÁSI, A.-L., PÓSFAI, M. *Network science*. Cambridge, Cambridge University Press, 2016. ISBN: 9781107076266 1107076269. Disponível em: <http://barabasi.com/networksciencebook/>.

[19] BOX, G. E. P., COX, D. R. "An Analysis of Transformations", *Journal of the Royal Statistical Society: Series B (Methodological)*, v. 26, n. 2, pp. 211–243, 1964. doi: 10.1111/j.2517-6161.1964.tb00553.x. Disponível em: <https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1964.tb00553.x>.

[20] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., et al. "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, v. 12, pp. 2825–2830, 2011.

[21] ROSENBLATT, F. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain", *Psychological Review*, pp. 65–386, 1958.

[22] BREIMAN, L. "Random Forests", *Machine Learning*, v. 45, n. 1, pp. 5–32, Oct 2001. ISSN: 1573-0565. doi: 10.1023/A:1010933404324. Disponível em: <https://doi.org/10.1023/A:1010933404324>.