UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ARIANN MICHAEL MARTINS DE ANDRADE FARIAS

RECOMENDAÇÃO DE PROJETOS DO GITHUB POR MEIO DE ALGORITMOS DE LEARNING TO RANK

CAMPINA GRANDE - PB

2021

ARIANN MICHAEL MARTINS DE ANDRADE FARIAS

RECOMENDAÇÃO DE PROJETOS DO GITHUB POR MEIO DE ALGORITMOS DE LEARNING TO RANK

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Franklin Ramalho.

CAMPINA GRANDE - PB

2021

F224r Farias, Ariann Michael Martins de Andrade.
Recomendação de projetos do GitHub por meio de Algoritmos de Learning to Rank. / Ariann Michael Martins de Andrade Farias. – 2021.

11 f.

Orientador: Professor Dr. Franklin de Souza Ramalho.

Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Plataforma GitHub. 2. Learning to Rank. 3. Algoritmos de Learning to Rank. 4. RankLib. 5. RankNet. I. Ramalho, Franklin de Souza. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

ARIANN MICHAEL MARTINS DE ANDRADE FARIAS

RECOMENDAÇÃO DE PROJETOS DO GITHUB POR MEIO DE ALGORITMOS DE LEARNING TO RANK

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Franklin Ramalho

Orientador – UASC/CEEI/UFCG

Professor Dr. Maxwell Guimarães de Oliveira

Examinador – UASC/CEEI/UFCG

Professor Tiago Lima Massoni

Professor da Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 24 de Maio de 2021.

CAMPINA GRANDE - PB

ABSTRACT

GitHub, as currently the biggest platform for hosting software development and version control, handles on a daily basis, a massive stream of interactions between users and repositories. With millions of repositories hosted on the platform, some projects that could be interesting for some users ended up being unnoticed, same as other projects which are searching for developers ended up staying on a limbo. In these situations, it becomes obvious the need for some mechanism that could help the user on choosing projects. In the literature, there are other studies on the same context, recommending projects using different approaches. Although, still there is space for new studies, using new aspects, in an attempt to verify and validate other results. Therefore, this study focuses on finding relevant projects for the users based on their interest, on the GitHub, using a set of features with the learning to rank algorithms support. Analysing the effectiveness of learning to rank, on the recommending projects context, using the algorithms RankNet, AdaRank and ListNet, in the sample space of 826 repositories and 3464 users on GitHub. The results present the target variable's relevancy and there is still much space for exploration on learning to rank approach for projects recommendation.

Recomendação de projetos do GitHub por meio de algoritmos de Learning to Rank

Ariann Michael M. de A. Farias
Universidade Federal de Campina Grande
Campina Grande, Brasil
ariann.farias@ccc.ufcg.edu.br

Franklin Ramalho
Universidade Federal de Campina Grande
Campina Grande, Brasil
franklin@computacao.ufcg.edu.br

RESUMO

O GitHub, atualmente a maior plataforma para hospedagem de código e controle de versionamento, possui um enorme fluxo diário de interações entre usuários e repositórios. Com o número de repositórios hospedados na casa dos milhões, alguns projetos que poderiam ser do interesse de alguns usuários acabam passando despercebidos, assim como projetos que necessitam de desenvolvedores, acabam ficando no ostracismo. Para esses casos, surge a necessidade de algum mecanismo que possa facilitar a escolha de projetos, pelo usuário. Na literatura outros trabalhos, já realizaram estudos sobre esse contexto, recomendando projetos com diferentes abordagens. Entretanto, ainda há espaço para novos estudos, utilizando novos aspectos, na tentativa de verificar e validar outros resultados. Por isso, esse trabalho busca encontrar projetos relevantes para o usuário, baseando-se nos interesses do mesmo, na plataforma GitHub, utilizando um conjunto de features com o auxílio de algoritmos de learning to rank. Analisamos a efetividade learning to rank, no contexto de recomendação de projetos, utilizando os algoritmos RankNet, AdaRank e ListNet, usando como espaço amostral 826 repositórios e 3464 usuários do GitHub. Os resultados mostram, a relevância da variável resposta e que a abordagem de learning to rank para recomendação de projetos oferece muito espaço para exploração.

PALAVRAS-CHAVE

Recomendação de projetos, learning to rank, GitHub.

1. INTRODUÇÃO

Uma das maiores plataformas de controle e hospedagem de código fonte, o GitHub [1], também disponibiliza para os desenvolvedores um ambiente para interações sociais, colaborações em projetos privados e *open source*. Muitos projetos são criados e mantidos seguindo a abordagem *open source*, ou seja, são abertos para que desenvolvedores não contratados possam modificar, copiar, contribuir e participar ativamente na tomada de decisões. Alguns projetos de sucesso no GitHub que podemos nomear são o Visual Studio Code[1], Blender[2] e Docker[3].

Para os desenvolvedores, contribuir em projetos *open source* é uma forma de criar *networking*, construir uma reputação e adquirir experiência. A plataforma GitHub disponibiliza diversas funcionalidades, como, por exemplo, favoritar repositórios por meio de *stars*, acompanhá-los sendo um *watcher*, visualizar tarefas em aberto como *issues* e contribuir por meio de *commits* e *pull requests.*

Atualmente o GitHub possui mais de 190 milhões de repositórios (28 milhões públicos) e mais de 40 milhões de usuários [2]. Com toda essa gama de informações, um *overflow* de possibilidades acaba sendo gerado para os desenvolvedores, desta forma dificultando na escolha de um projeto para contribuir. Uma vez que esse processo de escolha muitas vezes é realizado de forma manual, não é raro encontrar artigos no Medium [3] e frequentes buscas no Stackoverflow [4] questionando como encontrar um bom projeto *open source* para contribuir. Outra problemática nessa abordagem manual é a escolha inadequada do projeto, pela natural demora no processo de contribuição a um projeto, sendo realizado vários *commits*, a espera da aprovação de cada *commit* e *pull request* pela equipe *core* do projeto. Com a escolha incorreta de projetos, pode se tornar um processo tedioso e acarretar na desistência por parte do contribuidor.

Desta forma, uma abordagem para facilitar a escolha de projetos se faz necessária. A recomendação de repositórios com base nos projetos que o usuário demonstrou interesse no passado, favoritando, seguindo ou contribuindo, é uma maneira de ajudar a tratar este problema.

Alguns trabalhos seguindo essa perspectiva já existem, como por exemplo o Yang et al. [5] que apresenta um modelo de *learning to rank* para recomendação de projetos que o usuário possa vir a ser colaborador, utilizando das relações entre o colaborador e o dono do repositório, e também das diferença de tempo de *onboarding* do desenvolvedor no projeto. No trabalho de Cerqueira et al. [6], o problema da recomendação de projetos foi experimentado na forma de classificação, não utilizando *learning to rank*, tomando como *target* se o usuário irá realizar *fork* do projeto (em caso positivo é realizada a recomendação).

Um trabalho que devemos destacar foi o realizado por Dayvson et al. [7], tendo em vista que daremos continuidade ao mesmo. No trabalho de Dayvson et al., foram aplicadas 08 (oito) features de projetos e usuários: *stars, issues, forks,* linguagem de programação do projeto, linguagem de programação do usuário, *commits*, *watchers* e *followers*. Neste trabalho, o Dayvson et al. utilizaram o modelo de *learning to rank* para recomendação de projetos na plataforma do GitHub, com a biblioteca RankLib [8] em cima dos algoritmos de LambdaMART [9], Random Forest [10], Coordinate Ascent [11].

Nosso trabalho propõe realizar um estudo e avaliação de recomendações também por meio de técnicas de *learning to rank*

[1] https://github.com/microsoft/vscode
[2] https://github.com/blender/blender
[3] https://github.com/docker/docker.github.io

utilizando *features* que possam apresentar uma relevância na escolha dos projetos. Para a realização da recomendação de projetos, o foco foi a utilização de novos algoritmos (não explorador por Dayvson et al.), assim como novas *features* sendo uma união entre 04 (quatro) *features* utilizadas pelo trabalho de Yang et al. e 08 (oitos) utilizadas por Dayvson et al. ou seja *features* que envolvem dados brutos, e outras derivadas das relações entre o projeto e seus contribuidores. Em nosso estudo, avaliamos a relevância de 10 *features*: stars, issues, forks, linguagem de programação do projeto, linguagem de programação do usuário, commits, watchers, followers, social tie e user profile, utilizando a biblioteca RankLib, e aplicando 3 algoritmos de *learning to rank*: RankNet [9], AdaRank [12], ListNet [13]. O espaço amostral dos dados foi de 826 projetos e 3464 usuários do GitHub. As métricas utilizadas foram: NDGG (*Normalize Discounted Gain*) [14], MAP (*Mean Average Precision*) [15] e *Precision* [16].

A organização do artigo está estruturada da seguinte forma. Na seção 2 é apresentada a fundamentação teórica, detalhando os algoritmos utilizados. Na seção 3 é descrita a técnica proposta, contendo as especificações das *features* utilizadas. A seção 4 detalha o experimento realizado, descrevendo a coleta de dados, modelos treinados e métricas utilizadas. Na seção 5 são expostos e analisados os resultados obtidos no experimento. A seção 6 apresenta trabalhos relacionados. Por fim, a seção 7 descreve as conclusões e possíveis trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

Nesta seção, introduzimos a classe de técnicas *learning to rank* (Seção 2.1), assim como sua utilização no contexto de recomendação (Seção 2.2). Apresentamos a biblioteca que utilizamos RankLib (Seção 2.3) e detalhamos os algoritmos de *learning to rank* que foram utilizados neste trabalho (Seção 2.4).

2.1 Learning to Rank

De um modo geral, *learning to rank* [17] (LTR) pode ser definido como uma classe de técnicas de *machine learning* que foca em solucionar problemas de ranqueamento. O uso mais comum de LTR é para ranqueamento nos motores de busca como Google[4] e Bing[5], todavia pode ser utilizado em qualquer contexto que produza uma lista de itens ranqueados.

O treinamento de um modelo de LTR consiste em uma lista de itens e uma pontuação de *ground truth* para cada item da lista. No aspecto mais comum de busca, cada item possui um par de consulta-documento. Por meio do treinamento, o modelo vai gerando uma função de pontuação que podemos chamar de *score*, e para cada par de consulta-documento é associado um *score*. O ranqueamento ocorre ao ordenar a lista a partir dos *scores* de cada item. Em sequência, as listas ranqueadas são comparadas a uma lista de *ground truth*. Neste trabalho utilizamos como ground truth uma lista gerada pelas técnicas de Regressão Linear [18] e Ordenação Randômica.

Learning to rank pode ser categorizado em 3 tipos de abordagens: *pointwise*, *listwise* e *pairwise* [17].

Na abordagem *pointwise*, cada par de consulta-documento é tratado individualmente, aplicando métodos de classificação, classificação ordinal, ou regressão a fim de descobrir o melhor ranqueamento para cada resultado. Na classificação é realizado o agrupamento dos itens similares por classe. Já na regressão, cada par consulta-documento similares recebe uma função valor, que será utilizada para realizar o ranqueamento na lista. Independentemente de qual técnica *pointwise* for utilizada, cada consulta-documento receberá a predição do *score* e será realizado o ranqueamento por ordenação da relevância.

A abordagem *pairwise* se assemelha à *pointwise* ao aplicar métodos de classificação e regressão, todavia se diferencia ao invés de aplicar individualmente, aplica em pares de cada par consulta-documento. Nessa abordagem, o foco é identificar uma relação sequencial entre as duas instâncias, sempre comparando os pares ao *ground truth*, e por fim realizando o ranqueamento.

Na abordagem *listwise*, será encontrada a ordenação ótima por meio de modelos probabilísticos para toda a lista de pares consulta-documento. Cada lista é tomada como uma instância, são identificadas listas *ground truth* e as utiliza para o processo de comparação e ranqueamento com as demais, a fim de minimizar a função de custo, e por fim, gerando uma função de classificação. Um exemplo prático seria uma lista de documentos e uma consulta, onde, com a abordagem *listwise,* busca-se encontrar uma ordenação ótima para esses documentos em relação à consulta.

2.2 Learning to Rank e Recomendação

A utilização de *learning to rank* para recomendações, se dá através da transformação dos problemas de recomendação em problemas de ranqueamento. Ou seja, a lista de itens iniciais a serem recomendados passa por um modelo que utiliza de pontuações para ranqueá-los. Essas pontuações indicam a relevância dos itens para o contexto inserido. Em sequência, é gerada uma lista ranqueada de itens. Por fim, com a lista devidamente ranqueada, são selecionados os *top-n* itens a serem recomendados.

2.3 RankLib

RankLib, é uma biblioteca desenvolvida na linguagem Java para criação, treinamento e validação de modelos utilizando algoritmos de *learning to rank*. Atualmente, a biblioteca disponibiliza 08 (oito) algoritmos: LambdaMART, Random Forests, Coordinate Ascent, MART, RankNet, RankBoost, AdaRank e ListNet.

A biblioteca foi escolhida de maneira a dar continuidade a pesquisa desenvolvida por Dayvson et al. Neste projeto utilizamos os algoritmos RankNet, AdaRank e ListNet..

2.4 Algoritmos de Learning to Rank

Os algoritmos de *learning to rank* selecionados foram tomados com base da biblioteca RankLib, já utilizada em trabalhos anteriores, e que ainda permite espaço para novos estudos com outros algoritmos não analisados. Os algoritmos escolhidos a fim de diferir aos algoritmos utilizados no trabalho de Dayvson et al., foram RankNet, AdaRank e ListNet. A abordagem dos algoritmos RankNet e AdaRank é do tipo *pairwise,* enquanto a do ListNet é do tipo *listwise*.

[4] https://www.google.com/

[5] https://www.bing.com/

2.4.1 RankNet

RankNet [9], propõe encontrar um modelo de ranqueamento, otimizando a função de custo da abordagem *pairwise*, utilizando redes neurais e gradiente descendente estocástico. Essencialmente, o algoritmo de RankNet tenta minimizar o número de inversões realizadas durante as iterações. Uma inversão é realizada quando é identificado em uma lista de ranqueamento, um par consulta-documento com pontuação menor acima de outro par consulta-documento com uma pontuação maior, e realiza a troca entre ambos.

2.4.2 AdaRank

Inspirada no algoritmo de classificação AdaBoost [19], o algoritmo AdaRank [12] propõe encontrar um modelo de ranqueamento f de *learning to rank* por meio da abordagem *listwise*, enquanto o AdaBoost utiliza *pairwise*. AdaRank trabalha em T iterações e em cada iteração cria um *weak ranker* $h_t(t = 1, ..., T)$ seguindo uma distribuição por peso m. Ao término, a função f resulta na combinação linear dos *weak rankers*. Outra diferença entre AdaBoost e AdaRank, é que AdaBoost avalia a cada iteração o peso *m* e o *weak ranker*; Já o AdaRank avalia a cada iteração o peso *m*, o *weak ranker* e o modelo como um todo.

2.4.3 ListNet

Inspirada no algoritmo de *learning to rank* RankNet, o algoritmo ListNet [13], propõe otimizar a função de custo da abordagem *listwise*, enquanto o RankNet utiliza *pairwise*. ListNet trabalha utilizando de *Cross Entropy* [20] como função de custo e a otimização do gradiente descendente para o treinamento de um rede neural.

3. TÉCNICA PROPOSTA

Com o intuito de recomendar projetos relevantes para os usuários, se torna necessário focarmos em *features* que evoquem a relação entre os usuários e projetos que os mesmos apresentaram algum interesse. Essa relação de interesse pode ser verificada ao estudarmos o histórico dos usuários e projetos, em *features* diretas ou derivadas.

Desta forma, para realizarmos a recomendação dos top-n projetos dentro da lista ranqueada, utilizamos a biblioteca Ranklib e os 3 algoritmos: RankNet, AdaRank, ListNet.

Foram selecionadas e extraídas 10 *features*, sendo um conjunto de 08 (oito) *features* previamente apresentadas no trabalho de Dayvson et al. e 02 (duas) no trabalho de Yang et al. As mesmas foram obtidas de projetos e usuários do GitHub, e foram escolhidas a fim de representar a popularidade, atividade, oportunidade de colaboração, assim como a relação entre os usuários e os projetos. Como variáveis de respostas, foram utilizadas *commit* e *fork*, indicando se o usuário realizou essas ações ou não naquele projeto.

As *features* selecionadas foram:

- *Stars*: permite ao usuário favoritar um projeto, ou seja, sinalizar interesse sobre o projeto. Desta forma o GitHub mantém o usuário recebendo informações sobre o projeto. *Stars* é um fator de relevância para a pesquisa na própria plataforma, pois podemos visualizar a pesquisa ranqueada pelo número de *stars*. Abordamos essa *feature* de forma quantitativa, por cada projeto verificamos o número de *stars*.
- *Issues*: tarefas que estão no contexto do projeto, disponíveis para algum usuário contribuir, seja em correções ou em novas funcionalidades. Tem sua importância, pois reflete se o projeto ainda possui margem para futuras colaborações. Utilizamos a quantidade de *issues* com status de disponível, desconsiderando as com status de fechadas.
- *Forks*: São cópias do projeto alvo, realizadas por usuários, com o pretexto de realizar contribuições. De tal forma, ao utilizarmos a quantidade de *forks,* temos uma representação de quantos usuários têm interesse em contribuir para o projeto. Utilizamos essa *feature* de forma quantitativa para cada projeto.
- *Linguagem de programação do projeto*: Cada projeto foi desenvolvido com uma ou mais linguagens de programação. Fator relevante para entrada e mantimento de contribuidores, pois quanto maior a comunidade de uma linguagem, maiores as chances de desenvolvedores à procura de projetos para contribuir. Utilizamos a linguagem principal do projeto alvo.
- *Linguagem de programação do usuário*: também identificada como *Technical Ability*. Sendo a quantidade de projetos que o usuário contribuiu, que tinha como principal linguagem, a mesma do projeto alvo. Por exemplo, considere um usuário que tenha contribuído em 06 (seis) projetos que utilizam como linguagem principal Javascript, e outros 03 (três) projetos com linguagem principal Ruby; Sendo o projeto candidato tendo como principal linguagem Javascript, o valor dessa *feature* será 06 (seis). Abordamos essa *feature* de forma quantitativa.
- *Commits*: também identificada como *Project Growth*. Podemos definir *commit* como a representação de uma alteração realizada no projeto. A quantidade de *commits* pode ser vista como um indicativo de crescimento, estabilidade e relevância de um projeto. Lidamos com a quantidade de *commits* de um projeto.
- *Watchers*: permite ao usuário observar um projeto, sendo notificado sobre novidades referentes ao projeto, como *commits* e *issues*. Sendo mais uma forma de demonstrar interesse em contribuir para o projeto. Lidamos com a quantidade de *watchers* de um projeto.
- *Followers*: uma forma de interação entre usuários se dá através do ato de seguir outro usuário. Usuários que possuem muitos *followers*, tendem a ser ativos e relevantes na plataforma, realizando mais contribuições ou trazendo novos contribuidores. Utilizamos a contagem de *followers* por usuário.
- *Social Tie*: Sendo uma *feature* derivada da relação entre o dono do projeto e os colaboradores, em que verificamos em quantos projetos ambos contribuíram juntos. Por exemplo, se o dono do projeto alvo participou em outros 04 (quatro) projetos, e o usuário participou em 03 (três) desses mesmos projetos, o valor dessa *feature* será 03. Abordamos de forma quantitativa, contabilizando a quantidade de projetos.
- *User Profile*: Sendo uma *feature* derivada da relação entre usuários, assumindo que usuários teriam uma maior afinidade de se unir, em projetos cujos membros

integrem a mesma companhia que o usuário. Verificamos essa relação contabilizando por cada projeto, o número de colaboradores que compartilham da mesma companhia.

4. EXPERIMENTO

4.1 Coleta de Dados

O dataset que utilizamos, consiste dos dados de usuários e projetos da plataforma GitHub, dos anos 2012-2013. Os dados foram primeiramente coletados na base de dados do GH Archive[6], utilizando a ferramenta BigQuery[7]. Obtivemos os dados sobre os eventos, usuários e projetos. Já para informações mais detalhadas e *features* derivadas, fizemos uso da API do GitHub[8]. Este dataset foi o mesmo utilizado pelo trabalho de Dayvson et al., de tal forma a escolha feita pelos anos de 2012-2013, pois são uns dos primeiros anos disponíveis no GH Archive, e têm uma quantidade acessível de dados para analisar e extrair fatores que representam o interesse de usuários em projeto, tornando o experimento viável.

O espaço amostral que utilizamos foi de 826 projetos e 3464 usuários do GitHub.

4.1.1 Coleta de Repositórios

Ainda durante a coleta, como forma de pré-processamento dos dados de repositórios, realizamos uma filtragem sobre os projetos, optando por utilizar projetos com pelo menos 05 desenvolvedores, coletando assim 9854 repositórios. Após a coleta dos dados, realizamos o pré-processamento dos mesmos, removendo repositórios excluídos, que são *forks* de outros repositórios, que possuem menos de 2 *watchers*, não possuem descrição, privados e que não possuem informações sobre as linguagens de programação que foram utilizadas, resultando, assim, em 826 repositórios para utilização no experimento.

4.1.2 Coleta de Usuários

Na coleta dos usuários, obtivemos as informações dos contribuidores, usuários que fizeram pelo menos 01 *commit*, nos 826 repositórios que escolhemos, resultando em 12751 usuários. Após a coleta dos dados destes contribuidores, realizamos o pré-processamento, removendo os usuários excluídos, os determinados como *bots* pela plataforma GitHub, os que não possuíam informações sobre sua localização e os que não tinham repositórios ou que nunca apresentaram atividade, resultando, assim, em 3464 usuários.

4.2 Métricas

Utilizamos as seguintes métricas: NDCG@k, MAP e *Precision*@k para avaliação dos modelos, métricas essas, que são amplamente utilizadas para testes de performance de modelos de *learning to rank*. A aplicação foi através da biblioteca RankLib.

Normalized Discounted Cumulative Gain (NDCG): NDCG [14] é a normalização do DCG, que é representada pela soma ponderada dos graus de relevância de itens ranqueados. Pela natureza variante da métrica DCG, que pode variar junto com outros fatores por usuário. NDCG normaliza o DCG utilizando o DCG ideal (IDCG), que é a medida do DCG para o melhor resultado de ranqueamento possível. Logo NDCG normaliza seu valor para variar de 0 a 1, onde 1 indica uma perfeita ordenação dos itens. Para a validação do modelo, utilizamos NDCG@k, que é o NDCG para top-k de itens, sua fórmula é representada por:

$$NDCG@k = \frac{DCG@k}{IDCG@k}$$

Mean Average Precision (MAP): É uma métrica amplamente utilizada para Recuperação da Informação [21] e Detecção de Objetos [22]. MAP [15] corresponde a média de valores de *Average Precision* (AP) [22] para cada consulta, sua fórmula é representada por:

$$MAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q}$$

Precision: No contexto de recomendação, Precisão [16] é a proporção de documentos relevantes entre os recomendados. Especificamente em *precision@k* para avaliação de modelos é a proporção de itens recomendados entre os top-k documentos relevantes. Sua fórmula é representada por:

$$precision@k = \frac{\{relevant\ documents\} \cap \{retrieved\ documents\}}{\{retrieved\ documents\ k\}}$$

4.3 Configurações

4.3.1 Parâmetros dos algoritmos

Diante do uso dos algoritmos de *learning to rank*, disponibilizados pela biblioteca RankLib, os parâmetros que foram utilizados foram os padrões da própria biblioteca. Não realizamos ajustes e testes exaustivos sobre os parâmetros. A seguir ilustramos os valores utilizados para cada algoritmo:

Parâmetros RankNet	Valor do Parâmetro
Epochs (No. de ciclos no dataset)	100
No. of hidden layers (No. de camadas internas)	1
No. of hidden node per layer (No. de nós em cada camada interna)	10
Learning rate (Taxa de aprendizado)	0.00005

Tabela 4.1: Parâmetros do RankNet.

[6] https://www.gharchive.org/

[7] https://cloud.google.com/bigquery

[8] https://docs.github.com/en/rest

Parâmetros AdaRank	Valor do Parâmetro
Iteration (No. de iterações no dataset)	500
Tolerance (Tolerância entre 02 (duas) iterações consecutivas)	0.002
Max consecutive selection count (No. máximo que uma variável pode ser selecionada até modificar a performance)	5

Tabela 4.2: Parâmetros do AdaRank

Parâmetros ListNet	Valor do Parâmetro
Epochs (No. de ciclos no dataset)	500
Learning rate (Taxa de aprendizado)	0.0001

Tabela 4.3: Parâmetros do ListNet

4.3.2 Divisão dos conjuntos de treino e teste

A distribuição feita sobre o dataset foi de 80/20, *i.e.*, 80% dos dados foram separados para treinamento do modelo e 20% para teste. A escolha dessa proporção foi feita para seguir os padrões utilizados na área de Inteligência Artificial. A divisão dos conjuntos de treino e teste foi feita de forma aleatória. Foi utilizado o mesmo conjunto de treino e teste com todos os algoritmos.

Os conjuntos que utilizamos de teste e treino, assim como os modelos treinados e resultados, estão disponíveis em um repositório[9] no GitHub.

4.4 Baseline

Foram utilizadas duas propostas como *baseline*:

- *Regressão Linear*: usada como algoritmo de *learning to rank*, com abordagem *pointwise*. Escolhida para apresentar o comparativo entre um algoritmo simples e algoritmos construídos especificamente para ranqueamento, com a proposta *learning to rank*.
- *Ordenação Randômica*: uma ordenação aleatória das listas. Escolhida para permitir visualizar de forma intuitiva a relevância de algoritmos de *learning to rank*, em comparativo com uma ordenação simples.

Essas propostas foram escolhidas, a fim de termos a mesma base comparativa utilizada no trabalho de Dayvson et al.

5. RESULTADOS

Calculamos as métricas NDCG@10, NDCG@20, MAP, P@10 e P@20, para os 3 modelos treinados por meio dos algoritmos escolhemos: RankNet, AdaRank e ListNet; e os 3 modelos treinados com os algoritmos selecionados por Dayvson et al.: Coordinate Ascent, LambdaMART, RandomForest. Utilizamos as 10 *features*: *stars, issues, forks,* linguagem de programação do projeto, linguagem de programação do usuário, *commits*, *watchers, followers*, *social tie* e *user profile*. Os valores de 10 e 20 para k, representa avaliar a recomendação dos top-10 e 20 projetos para os usuários. Valores maiores que 20 itens seriam irrelevantes em um cenário real, assim como menores que 10 itens seriam muito restritos.

Apresentamos resultados obtidos pelos modelos, na Tabela 5.1.

Modelo	**NDCG @10**	**NDCG @20**	**MAP**	**P@10**	**P@20**
Coordinate Ascent	0.0432	0.0503	0.0296	0.0111	0.007
Lambda MART	0.0139	0.0113	0.0127	0.0033	0.0028
Random Forest	0.009	0.0158	0.0111	0.0023	0.0027
Regressão Linear	0.0082	0.0189	0.0104	0.0025	0.0035
AdaRank	0.0047	0.006	0.061	0.0015	0.001
ListNet	0.0033	0.0056	0.0052	0.001	0.0011
RankNet	0.0	0.0	7.0E-4	0.0	0.0
Randômico	0.0	0.0	7.0E-4	0.0	0.0

Tabela 5.1: Métricas de avaliação para cada um dos algoritmos.

5.1 Análise

Analisando os algoritmos de *learning to rank* que utilizamos, desconsiderando os *baselines*, os que tiveram melhores resultados foram AdaRank e LisNet. O RankNet foi o algoritmo que teve o desempenho mais baixo, sendo valor tão baixo que resultou próximo de 0.0 (zero) em todas as métricas. Desta forma, mostrando que RankNet, nesse contexto, possui pouca eficiência para colocar os projetos mais relevantes em um *rank* maior na nossa base de dados.

Comparando com os *baselines*, vemos que o melhor algoritmo de *learning to rank* que escolhemos foi o AdaRank, todavia os resultados foram inferiores aos de regressão linear. Logo, indicando que todos os 3 modelos treinados com os algoritmos que selecionamos, não foram eficientes na recomendação de projetos relevantes.

[9] https://github.com/ariannmichael/tcc_github_ltr

Analisando os resultados da Tabela 5.1 e fazendo o comparativo com o trabalho de Dayvson et al., por compartilharmos o uso da mesma base de dados e as mesmas 08 *features*, e testarmos os mesmo 3 algoritmos: Coordinate Ascent, LambdaMART, RandomForest. Verificamos que os resultados foram provenientes da variável de resposta escolhida, visto que neste trabalho utilizamos se um usuário faria ou não *fork* e no trabalho de Dayvson et al. utilizou se um usuário contribuiria ou não.

Ao realizarmos o cálculo do peso das *features* obtivemos resultado apresentado na Figura 1.

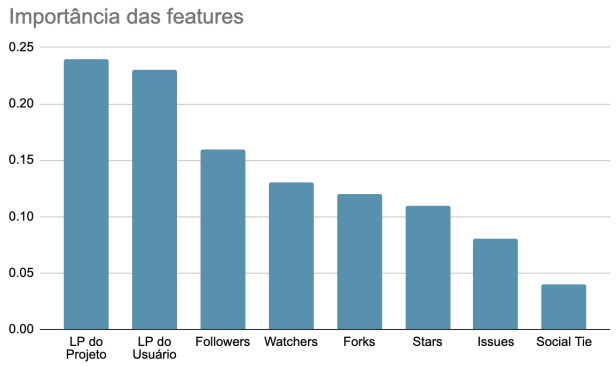

Figura 1: Importância das *features*

As *features* relacionadas às linguagens tanto de projeto, quanto de usuário, tiveram um peso maior que as demais. Logo, que essas *features* representam a relação de habilidade técnica entre usuário e projeto, deixando evidente como um fator decisivo na escolha do usuário. Já as outras *features*: *followers, watchers, forks* e *issues*, apresentam um peso parecido por estarem relacionados com a quantidade de usuários. Além disso, podemos verificar que as *features* derivadas: *social tie* e *user profile* não tiveram tanta relevância para capturar o interesse dos usuários.

6. TRABALHOS RELACIONADOS

O trabalho de Cerqueira et al. [6] analisou a recomendação de projetos no GitHub, utilizando o método de classificação, sobre os dados obtidos do GitHub de 2010 até abril de 2018. Foram utilizadas 23 *features* e os algoritmos de classificação: XGBoost [23], Item-KNN [24], Random-Forests [10] e Máquina de Fatoração [25]. A *feature* alvo foi *fork*, os modelos foram treinados a fim de prever se o usuário faria ou não *fork*. Dentre as 23 features diretas e derivadas observadas, 05 foram de comum uso no nosso estudo: *issues*, *commits*, *watchers*, *followers* e linguagem principal do projeto; e outras 18 diferem: quantidade de eventos de um projeto, ano de criação do projeto, peso da linguagem principal do usuário, média, desvio padrão e quantidade máxima de *commits*, *watchers*, *pull requests* por projeto, quantidade de *pull requests* de um projeto, quantidade máxima, média de *issues*, variação da periodicidade do projeto, contribuição diária em um projeto e atividade do projeto.

Em consideração às métricas de qualidade e aos resultados obtidos no trabalho de Cerqueira et al., foi constatado margem para melhoria. De tal forma, foi realizado um novo treinamento com um subconjunto referente às 23 *features* iniciais, utilizando como métrica de escolha do subconjunto, o contexto da primeira fase de treinamento. Os resultados obtidos dos modelos treinados com esse novo subconjunto, apresentou uma melhora nas métricas de qualidade dos modelos, tendo destaque o algoritmo XGBoost, apresentando um aumento de 0.01270 para 0.0389 na métrica de p@10 e de 0.00492 para 0.00669 na métrica de f1-score@10.

O trabalho de Yang et al. [5] focou o estudo na relação entre os projetos e o processo de entrada dos contribuidores em projetos, resultando no desenvolvimento de um modelo de *learning to rank* chamado Neural Network for List-wise Ranking (NNLRank), modelo esse que procura recomendar projetos no GitHub, em que o usuário seja mais favorável a contribuir. Foram utilizadas 09 *features* derivadas, obtidas dos dados do GitHub de 2012-2013, envolvendo aspectos como relacionamento entre contribuidores, tempo de vida do projeto e espaço para crescimento no projeto. Entre as 09 (nove) *features*, 02 (duas) são comuns com nosso estudo: *technical ability*, e *commits*; outras 02 (duas) escolhemos utilizar no nosso estudo: *social tie*, e *user profile*; 05 são diferentes: *project creation time, first membership time*, *last membership time*, *first commit time*, *last commit time*.

Relativo às métricas de qualidade e resultados obtidos no trabalho de Yang et al., o modelo de NNLRank foi comparado com outros modelos de *learning to rank* sendo estes: SVMRank [26], SVM [27] e BPNet [28]; sendo estes de abordagens *pairwise* e *pointwise*. Em termos comparativos de métricas de qualidade, NNLRank superou os demais modelos, em relação ao segundo melhor SVMRank; houve um aumento de 0.351 para 0.454 em MAP@10, e 0.354 para 0.457 em MAP@20.

O trabalho de Dayvson et al. [7] realizou um estudo de *learning to rank* para recomendação de projetos na plataforma GitHub sobre os dados obtidos do GitHub de 2012-2013. Foi utilizada a biblioteca RankLib, para realizar os treinamentos e validações dos modelos desenvolvidos, obtidos através dos algoritmos LambdaMART, Random Forest e Coordinate Ascent. A *feature* alvo foi *commit*, ou seja, os modelos foram treinados a fim de prever se o usuário contribuiria ou não para o projeto. Foram utilizadas 08 *features*: *stars, issues, forks,* linguagem de programação do projeto, linguagem de programação do usuário,, *commits, watchers, followers;* sendo estas todas em comum com nosso trabalho, tendo em vista que nosso trabalho procura dar continuidade ao estudo de Dayvson et al. O algoritmo que obteve melhores resultados aos demais, foi o LambdaMART: em relação ao segundo melhor Random Forest; houve um aumento de 0.1246 para 0.1289 em NDCG@10, 0.1541 para 0.1604 em NDCG@20, 0.1081 para 0.1108 em MAP, 0.0247 para 0.0254 em P@10, 0.0185 para 0.0193 em P@20. Não obstante, mesmo com bons resultados, ainda há espaço para novos estudos. Podendo ser utilizadas novas possibilidades, como nova variável de resposta, *features* tanto diretas, como derivadas e explorar uma quantidade maior de algoritmos que a própria biblioteca.

Em comparativo com os trabalhos de Cerqueira et al., que utilizou de algoritmos de classificação, e Yang et al que focou na entrada de contribuidores, este presente trabalho utilizou de algoritmos de *learning to rank* e focou na recomendação pelos interesses dos usuários. Através da análise e estudo dos trabalhos citados previamente, este presente trabalho propõe dar continuidade ao estudo de Dayvson et al., adicionando novas *features* e a utilização de novos algoritmos, sobre o mesmo espaço

amostral, desta forma, verificando a existência de diferentes resultados na abordagem *learning to rank*.

7. CONCLUSÕES

O estudo de recomendação de projetos na plataforma GitHub, se apresentou como uma tarefa desafiadora, todavia bastante importante, levando em consideração a enorme gama de projetos e usuários na plataforma, assim como o número de estudos na literatura. O presente trabalho teve o objetivo de analisar novos algoritmos de *learning to rank*, utilizando um novo conjunto de *features* e variável resposta, a fim de encontrar projetos relevantes para o usuário, baseando-se nos interesses do mesmo, assim como nas relações entre usuários, na plataforma GitHub. Analisamos os modelos treinados e fizemos o comparativo com abordagens mais simples, desta forma, verificamos a eficácia das técnicas de *learning to rank*, por meio das métricas (NDCG, MAP e *Precision*), no contexto de recomendação de projetos para usuários. Os resultados mostram, que a abordagem de learning to rank para recomendação de projetos oferece muito espaço para exploração, estudando novas *features*, algoritmos de *learning to rank*, variáveis de resposta e ambientes na tentativa de obter melhores resultados de recomendação.

Por questões de restrição de tempo e recursos computacionais, não conseguimos utilizar mais algoritmos, assim como utilizar uma base de dados maior. Adicionalmente, não exploramos uma quantidade maior de features por limitações de tempo. Além do mais, não utilizamos mais variáveis de resposta, por limitação da biblioteca RankLib [8]. Assim como tivemos dificuldades no experimento, pela documentação defasada e pouco explicativa. Logo, melhores resultados poderiam ser obtidos utilizando *features* que não foram exploradas neste trabalho, e utilizando uma ambiente com melhores recursos computacionais.

Em trabalhos futuros, pretendemos analisar uma quantidade maior de *features*, adicionando mais *features* derivadas que não foram utilizadas neste trabalho, com foco na característica do tempo em relação aos projetos e usuários. Além disso, há margem para exploração de novos algoritmos de *learning to rank*, bem como outros algoritmos que possuem relevância no contexto de recomendação de projeto. Também imaginamos que há relevância para expandir o foco em novas variáveis de resposta, por exemplo, uma junção de *stars* e *contributed* (se o usuário contribuiu ou não), pretendemos utilizar um espaço amostral que englobe anos mais recentes, de forma que possamos verificar outros fatores que sejam relevantes para o interesse do usuário, que não puderam ser analisados no espaço estudado. Assim como exploração de novas variáveis resposta, visto a relevância que a mesma teve nos resultados deste trabalho. Por meio da exploração de uma base de dados maior e uma nova variável de resposta, acreditamos poder encontrar novos resultados e validar os fatores já analisados.

8. AGRADECIMENTOS

Agradeço a Deus, o maior matemático de todos, por me dar a benção de estar aqui hoje.

Ao meu orientador, Prof. Dr. Franklin Ramalho, pela orientação e ter estendido a mão quando eu mais precisei. Agradeço aos meus pais Agenor Filho e Maria Magnólia, e meu irmão Agenor Ashley por todo o suporte e incentivo. A minha sogra por todas as suas orações. Em especial agradeço a minha esposa Nathalia Maia, por todo apoio e companheirismo.

Por fim, agradeço aos meus amigos Agnaldo, Damião, Dayvson, João Pedro, Luan, Matheus Maia, Matheus Melo, Nilton, Ricardo Almeida, Rubens, Samuel, Tiago, Victor, Wesley e Yovany; Por compartilharem comigo o bom humor e ensinamentos, durante essa jornada chamada graduação.

9. REFERÊNCIAS

[1] GitHub - Where the world builds software. Disponível em: <https://github.com/>. Acesso em: 06 abril 2021.

[2] https://en.wikipedia.org/wiki/GitHub#:~:text=As%20of%20January%202020%2C%20GitHub,source%20code%20in%20the%20world.

[3] Medium – Where good ideas find you. Disponível em: <https://medium.com/>. Acesso em: 06 abril 2021.

[4] Stack Overflow - Where Developers Learn, Share, & Build. Disponível em: <https://stackoverflow.com/>. Acesso em: 06 abril 2021.

[5] CHAO LIU, DAN YANG, XIAOHONG ZHANG , BAISHAKHI RAY , AND MD MASUDUR RAHMAN: Recommending GitHub Projects for Developer Onboarding - IEEE Access (Volume: 6), 10 September 2018.

[6] Cerqueira, T.: Explorando Características Sociais e de Colaboração e de Colaboração na Recomendação de Projetos no GitHub. 2020. Tese de Doutorado em Ciência da Computação – Pós-Graduação em Ciência da Computação, Centro de Engenharia Elétrica e Informática, Universidade Federal de Campina Grande, Paraíba, Brasil, 2019.

[7] Nascimento, Dayvson.: Aplicando Algoritmos de Learning to Rank sobre Features no GitHub para Recomendação de Projetos. 2020. Trabalho de Conclusão de Curso - Graduação em Ciência da Computação, Centro de Engenharia Elétrica e Informática, Universidade Federal de Campina Grande, Paraíba, Brasil, 2020.

[8] The Lemur Project - RankLib. Disponível em <https://sourceforge.net/p/lemur/wiki/RankLib/>. Acesso em 08 abril 2021.

[9] Intuitive explanation of Learning to Rank (and RankNet, LambdaRank and LambdaMART). Disponível em: <https://medium.com/@nikhilbd/intuitive-explanation-of-learning-to-rank-and-ranknet-lambdarank-and-lambdamart-fe1e17fac418>. Acesso em: 08 abril 2021.

[10] Understanding Random Forest. Disponível em: <https://towardsdatascience.com/understanding-random-forest-58381e0602d2#:~:text=The%20random%20forest%20is%

20a,that%20of%20any%20individual%20tree>. Acesso em: 08 abril 2021.

[11] Metzler, D., Croft, W.: Linear Feature-Based Models for Information Retrieval - 2006 Kluwer Academic Publishers.

[12] AdaRank: A Boosting Algorithm for Information Retrieval. Disponível em: <http://www.bigdatalab.ac.cn/~junxu/publications/SIGIR2007_AdaRank.pdf>. Acesso em: 09 abril 2021.

[13] ListNet (Listwise Learning to Rank) | by Pere Urbon-Bayes. Disponível em: <https://medium.com/@purbon/listnet-48f56cb80bb2>. Acesso em: 09 abril 2021.

[14] Evaluate your Recommendation Engine using NDCG | by Pranay Chandekar. Disponível em: <https://towardsdatascience.com/evaluate-your-recommendation-engine-using-ndcg-759a851452d1>. Acesso em: 12 abril 2021.

[15] mAP (mean Average Precision) might confuse you! | by Shivy Yohanandan. Disponível em: <https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2#:~:text=The%20mean%20Average%20Precision%20or,an%20IoU%20threshold%20of%200.5>. Acesso em: 15 abril 2021.

[16] Recall and Precision at k for Recommender Systems. Disponível em: <https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54#:~:text=Precision%20at%20k%20is%20the,are%20relevant%20to%20the%20user>. Acesso em: 15 abril 2021.

[17] The ABCs of Learning to Rank. Disponível em: <https://lucidworks.com/post/abcs-learning-to-rank/>. Acesso em: 15 abril 2021.

[18] Como funciona uma Regressão Linear? | by Carlos Alberto Bonfim | Data Hackers. Disponível em: <https://medium.com/data-hackers/como-funciona-uma-regress%C3%A3o-linear-f7208fa6c662>. Acesso em: 17 abril 2021.

[19] Understanding AdaBoost. Anyone starting to learn Boosting… | by Akash Desarda. Disponível em: <https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe> Acesso em: 18 abril 2021.

[20] 49 Responses to A Gentle Introduction to Cross-Entropy for Machine Learning. Disponível em: <https://machinelearningmastery.com/cross-entropy-for-machine-learning/>. Acesso em: 25 abril 2021.

[21] Hang Li: A Short Introduction to Learning to Rank - IEICE TRANS. INF. & SYST., VOL.E94–D, NO.10 October 2011.

[22] Breaking Down Mean Average Precision (mAP) | by Ren Jie Tan. Disponível em: <https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52#1a59>. Acesso em: 25 abril 2021.

[23] A Gentle Introduction to XGBoost for Applied Machine Learning. Disponível em: <https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/>. Acesso em: 03 maio 2021.

[24] Prototyping a Recommender System Step by Step Part 1: KNN Item-Based Collaborative Filtering. Disponível em: <https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea>. Acesso em: 03 maio 2021.

[25] Factorization Machines for Item Recommendation with Implicit Feedback Data. Disponível em: <https://towardsdatascience.com/factorization-machines-for-item-recommendation-with-implicit-feedback-data-5655a7c749db>. Acesso em: 03 maio 2021.

[26] Support Vector Machine for Ranking. Disponível em: <https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html>. Acesso em: 06 maio 2021.

[27] SVM-Struct Support Vector Machine for Complex Outputs. Disponível em: <https://www.cs.cornell.edu/people/tj/svm_light/svm_struct.html>. Acesso em: 06 maio 2021.

[28] Kyungchul Park and Youngmin Yi. 2019. BPNet: Branch-pruned conditional neural network for systematic time-accuracy tradeoff in DNN inference: work-in-progress. In Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis Companion (CODES/ISSS '19). Association for Computing Machinery, New York, NY, USA, Article 2, 1–2.

[29] Mean reciprocal rank. Disponível em: <https://en.wikipedia.org/wiki/Mean_reciprocal_rank>. Acesso em: 08 maio 2021.