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

DAYVSON WESLLEY CANTALICE DO NASCIMENTO

APLICANDO ALGORITMOS DE *LEARNING TO RANK* SOBRE *FEATURES* NO *GITHUB* PARA RECOMENDAÇÃO DE PROJETOS

CAMPINA GRANDE - PB

2020

DAYVSON WESLLEY CANTALICE DO NASCIMENTO

APLICANDO ALGORITMOS DE *LEARNING TO RANK* SOBRE *FEATURES* NO *GITHUB* PARA RECOMENDAÇÃO DE PROJETOS

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 de Souza Ramalho.

CAMPINA GRANDE - PB

2020

N244a Nascimento, Dayvson Weslley Cantalice do.
Aplicando algoritmos de learning to rank sobre features no GitHub para recomendação de projetos. / Dayvson Weslley Cantalice do Nascimento. – 2020.

10 f.

Orientador: Prof. 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. Recomendação de projetos. 2. Algoritmos de learning to rank. 3. GitHub. 4. Algoritmo LambdaMART. 5. Algoritmo Random Forest. 6. Algoritmo Coordinate Ascent. I. Ramalho, Franklin de Souza. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

DAYVSON WESLLEY CANTALICE DO NASCIMENTO

APLICANDO ALGORITMOS DE *LEARNING TO RANK* SOBRE *FEATURES* NO *GITHUB* PARA RECOMENDAÇÃO DE PROJETOS

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 de Souza Ramalho
Orientador – UASC/CEEI/UFCG

Professora Dra. Patrícia Duarte de Lima Machado
Examinador – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni
Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 2020.

CAMPINA GRANDE - PB

Aplicando Algoritmos de Learning to Rank sobre Features no GitHub para Recomendação de Projetos

Dayvson Weslley Cantalice do Nascimento
Universidade Federal de Campina Grande
Campina Grande, Brasil
dayvson.nascimento@ccc.ufcg.edu.br

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

RESUMO

O GitHub é a plataforma de hospedagem de código e controle de versão mais utilizada atualmente. Diariamente, inúmeros projetos são criados, estendidos e modificados por diferentes usuários. Entretanto, muitos projetos que possivelmente seriam do interesse de determinados usuários, acabam por passar despercebidos diante da grande quantidade de projetos disponíveis. Neste contexto, surge a necessidade de algum mecanismo que possa auxiliar o usuário a encontrar projetos que podem ser de seu interesse. Já existe na literatura trabalhos que buscam analisar fatores de interesse com o objetivo de recomendar projetos, entretanto ainda há margem para utilização de outros fatores e critérios na tentativa de obter resultados melhores. Para tanto, o presente trabalho busca utilizar features, algumas já propostas na literatura e outras ainda não utilizadas nesse contexto, disponíveis em projetos do GitHub, com o auxílio de algoritmos de learning to rank, para encontrar relações de interesse em projetos e assim recomendá-los para o usuário. Verificamos a efetividade de learning to rank para recomendação de projetos usando os algoritmos LambdaMART, Random Forest e Coordinate Ascent, utilizando como base 826 repositórios e 3464 usuários do GitHub. Os resultados mostram que a abordagem de learning to rank para recomendação de projetos é promissora e efetiva, ao mesmo tempo que oferece muito espaço para aprimoramento.

PALAVRAS-CHAVE

Recomendação de projetos, learning to rank, GitHub

1. INTRODUÇÃO

A colaboração em projetos *open source* é algo muito comum nos dias atuais. Muitas aplicações foram desenvolvidas e se mantêm graças a esforços coletivos. Como exemplo de projetos de sucesso podemos citar Visual Studio Code[1], React[2] e Tensorflow[3].

O GitHub [1], além de ser uma plataforma para hospedagem e controle de versão colaborativa, também oferece *features* que ajudam usuários a se conectarem entre si e também a se conectarem com repositórios de projetos. É possível, por exemplo, seguir outros usuários e receber informações sobre sua(s) atividade(s) em projetos/repositórios. Além disso, existe muita informação sobre os repositórios públicos desta plataforma, como quais usuários acompanham aquele projeto, quais contribuíram (através de *commits* ou reportando *bug*/*issues*), linguagens de programação utilizadas, data de criação, histórico de *commits*, entre outros.

Contudo, existe o problema de que diante de tanta informação (atualmente existem mais de 100 milhões de projetos e mais de 30 milhões de usuários no GitHub [2]), acontece de muitos usuários que seriam potenciais contribuidores em projetos relacionados aos que estes acessam, não terem conhecimento da existência desses projetos "similares" onde poderiam contribuir.

Diante disso, a existência de um sistema que pudesse recomendar projetos a um usuário com base nos projetos que este acessou no passado, por exemplo, poderia aumentar significativamente a contribuição colaborativa, além de aumentar o engajamento e participação de membros no projetos. Para tanto, é necessário conseguir relacionar projetos que o usuário acessou ou demonstrou interesse.

Já existem trabalhos neste sentido. Yang et al. [3] propõe um modelo de *learning to rank* para recomendação de projetos que o usuário possa virar contribuidor. Já no trabalho de Cerqueira et al. [4], o problema de recomendação de projetos é transformado em um problema de classificação, prevendo se um usuário irá realizar *fork* ou não de um projeto, e assim realizando a recomendação. Entretanto, ainda há margem para utilização de outros critérios e métricas na tentativa de conseguir melhores resultados de recomendação de projetos para usuários.

Neste trabalho, usamos algumas *features* de projetos e usuários do GitHub, com o intuito de capturar fatores que representam o interesse de usuários de contribuir em projetos, utilizando destas como fonte para algoritmos de *learning to rank,* e assim poder realizar a recomendação de projetos relevantes. Em nossa abordagem, extraímos 8 *features* que representam fatores de interesse e utilizamos três algoritmos de learning to rank: LambdaMART [5], Random Forest [6] e Coordinate Ascent [7] para treinar modelos de *learning to rank* com as *features* selecionadas. Verificamos a efetividade desses algoritmos utilizando dados de 826 projetos e 3464 usuários do GitHub, avaliando-os através de métricas amplamente usadas em Recuperação da Informação: NDCG (*Normalized Discounted Gain*), MAP (*Mean Average Precision*) e *Precision*. Nos nossos testes, os algoritmos de *learning to rank* utilizados tiveram desempenho similar para recomendação de projetos, e em especial o LambdaMART alcançou os melhores resultados. Os resultados são encorajadores e indicam que a abordagem de *learning to rank*

[1] https://github.com/microsoft/vscode
[2] https://github.com/facebook/react
[3] https://github.com/tensorflow/tensorflow

para recomendação de projetos no GitHub é promissora e que ainda existe muito espaço para aprimoramento.

A recomendação de projetos no GitHub é uma tarefa relevante que ainda têm muito espaço para discussão e pesquisa. Neste trabalho buscamos contribuir ao analisar a importância e usar *features* que possam refletir relações de interesse, utilizando técnicas de *learning to rank* para a recomendação de projetos e avaliando essas recomendações.

O restante do documento está organizado como segue. A seção 2 apresenta a fundamentação teórica, enquanto a seção 3 descreve a técnica proposta, enfatizando as *features* que foram selecionadas. A seção 4 apresenta como foi realizado o experimento, incluindo a coleta de dados, avaliação dos modelos treinados, *baselines*, ameaças e métricas utilizadas. Na seção 5 é feita a discussão e análise dos resultados obtidos no experimento. A seção 6 apresenta alguns trabalhos relacionados, enquanto que a seção 7 descreve as conclusões, limitações e trabalhos futuros.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Learning to Rank

Learning to rank [8] (LTR) refere-se a uma classe de técnicas de machine learning para treinamento de modelos em uma tarefa de rankeamento, sendo muito útil para aplicações de Recuperação da Informação, Processamento de Linguagem Natural e Mineração de Dados [9]. Porém, pode ser aplicada em qualquer contexto que precise produzir uma lista ranqueada de itens.

O treinamento de um modelo de LTR é feito com uma lista de itens, cada item consistindo de um par consulta-documento, com a relevância do documento de acordo com a consulta. O modelo tenta aprender uma função de *score* e assim determinar um *score* para um par consulta-documento, onde itens com *score* maior recebem um *rank* maior do que aqueles com *score* menor. O teste de um modelo de LTR é feito com a comparação entre as listas ranqueadas geradas pelo modelo e as listas tidas como *ground truth,* ou seja, as listas com a ordenação observada.

De acordo com a função de custo, algoritmos de *learning to rank* são categorizados em 3 grupos: *pointwise*, *pairwise* e *listwise* [8].

Na abordagem *pointwise* o problema de ranqueamento é transformado em classificação, regressão, ou classificação ordinal, e os métodos existentes de classificação, regressão ou classificação ordinal são aplicados [9]. A abordagem *pointwise* trata cada par consulta-documento de maneira separada, tentando predizer o rótulo de relevância correto para cada par [3]. Considerando uma consulta e um conjunto de documentos, nesta abordagem é feita a predição do *score* de cada uma desses documentos em relação à consulta, sendo os maiores *scores* colocados em *ranks* superiores.

Na abordagem *pairwise*, o ranqueamento é transformado em classificação de pares ou regressão de pares. No primeiro caso, um classificador para classificar as ordens de classificação de pares de documentos é criado e empregado na classificação de documentos [9]. Nesta abordagem, o objetivo é aprender uma relação sequencial entre instâncias [3]. Considerando, por exemplo, os documentos A, B e C e uma dada consulta, é verificado entre o pares de documentos qual o mais relevante para a consulta, ou seja, entre A e B qual o mais relevante, entre B e C o mais relevante, e entre A e C o mais relevante.

Já na abordagem *listwise* o problema de ranqueamento é resolvido de maneira mais direta [9]. Nesta abordagem, uma função de classificação é apreendida tomando listas individuais como instâncias e minimizando uma função de custo definida na lista prevista em relação a lista utilizada como *ground truth* [10]. Esta abordagem busca aprender uma relação sequencial entre uma lista de instâncias uma só vez [3]. Considerando uma lista de documentos e uma consulta, por exemplo, nesta abordagem tenta-se encontrar uma ordenação ótima para esses documentos em relação a esta consulta.

2.2 Learning to rank e Recomendação

O problema de recomendação pode ser transformado em uma tarefa de ranqueamento, onde para cada lista ranqueada, podemos extrair o top-n e utilizá-lo como recomendação. Como mostrado na Figura 1, a partir de uma lista de itens, podemos gerar uma lista ranqueada através da aplicação de um modelo de *learning to rank*. Com a lista ranqueada, podemos obter a recomendação extraindo o top-n itens dessa lista.

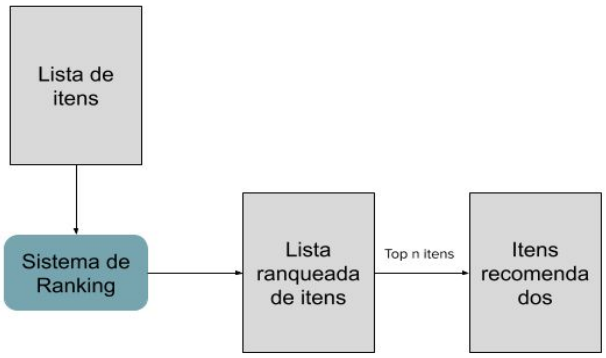

Figura 1: *Learning to rank* utilizado para recomendação.

2.3 Algoritmos de Learning to Rank

Selecionamos três algoritmos de *learning to rank* para serem utilizados em nosso experimento: LambdaMART, Random Forest e Coordinate Ascent, um para cada abordagem diferente de *learning to rank*, *pairwise*, *pointwise* e *listwise*, respectivamente. Foi utilizada a implementação dos algoritmos da biblioteca RankLib [11], escrita em Java com implementação de vários algoritmos de LTR, e que também oferece diversas ferramentas para validação de modelos treinados utilizando métricas comumente utilizadas em LTR.

2.3.1 LambdaMART

LambadaMART é um algoritmo de LTR que combina MART (*Multiple Additive Regression Trees*) [12] e LambdaRank [13]. Enquanto o MART usa árvores de decisão com aumento de gradiente para tarefas de previsão, LambdaMART usa árvores de decisão com aumento de gradiente usando uma função de custo derivada do LambdaRank para resolver uma tarefa de ranqueamento [5].

2.3.2 Random Forest

Random Forest consiste de grande número de árvores de decisão individuais que operam como um conjunto. Cada árvore é criada utilizando *bagging* e seleção randômica de *features,* formando uma floresta não correlacionada de árvores cuja predição é melhor

do que qualquer árvore individual [6]. Para a tarefa de ranqueamento, a árvore com menor erro é selecionado como modelo de ranqueamento [14].

2.3.3 Coordinate Ascent

Coordinate Ascent é descrito como um método de otimização. O algoritmo otimiza iterativamente uma função objetivo multivariada resolvendo uma série de pesquisas. Ele percorre repetidamente cada parâmetro, mantendo todos os outros parâmetros fixos e otimiza sobre o parâmetro livre [7].

3. TÉCNICA PROPOSTA

Com o objetivo de capturar o interesse de desenvolvedores do GitHub em colaborar em projetos, buscamos analisar e selecionar *features* de projetos e usuários da plataforma de forma que pudéssemos, com base nessas *features* e no histórico de colaboração de usuários, recomendar projetos de possível interesse do usuário. Para tanto, utilizamos três algoritmos de *learning to rank* (enumerados na Seção 2.3) para gerar uma lista ordenada de possíveis projetos candidatos e, a partir desta lista, recomendar o top-n projetos para o usuário.

A seguir, apresentamos as *features* selecionadas, bem como de qual maneira elas podem ser usadas de forma a refletir o interesse de usuário em projetos.

Foram extraídas 8 *features* de projetos e usuários do GitHub. Selecionamos *features* que indicam a popularidade, atividade e oportunidade de colaboração em um projeto, bem como *features* que indicam um *match* entre habilidade técnica requerida pelo projeto e a do usuário. Como variável de resposta, utilizamos se o usuário virou ou não contribuidor naquele projeto, ou seja, se fez pelo menos um *commit.*

- *Stars*: são utilizadas no GitHub para acompanhar e manter-se informado sobre um projeto. As *stars* são utilizadas como sinal de apreciação do conteúdo de um repositório. O número de *stars* de um repositório também é utilizado como um filtro de busca dentro da própria plataforma, da forma que é possível pesquisar repositórios pela popularidade com base no números de *stars.* Utilizamos o número de *stars* de um projeto.
- *Issues*: utilizada dentro da plataforma como forma de apontar um erro, uma possibilidade de aprimoramento ou fazer outras solicitações, como pedir uma nova *feature*, por exemplo, em um projeto. A quantidade de *issues* abertas de um projeto pode indicar que existe espaço para colaboração, sendo assim é esperado que projetos com issues disponíveis possa atrair novos colaboradores. Portanto, utilizamos a quantidade *issues* disponíveis (*open issues*) no projeto.
- *Forks*: Um *fork* é criado quando um usuário faz uma cópia de um projeto, sendo utilizado como forma de um usuário contribuir em um projeto no qual ainda não é colaborador. Sendo assim, a quantidade de *forks* de um projeto pode indicar a quantidade de diferentes usuários que têm interesse naquele projeto. Utilizamos a quantidade de *forks* do projeto.
- *Linguagem de Programação do projeto*: Projetos podem ser escritos em uma ou mais linguagem de programação. Geralmente, a linguagem de programação utilizada em um projeto pode ser um fator que influencia um usuário a ter interesse em virar um contribuidor daquele projeto. Utilizamos a principal linguagem de programação de um repositório.
- *Linguagem de Programação do usuário*: Para esta feature utilizamos a quantidade de projetos que um usuário participou anteriormente com a mesma linguagem de programação que a do projeto. Por exemplo, se um usuário participou de 5 projetos tendo como principal linguagem Python e outros 3 tendo como principal linguagem Java, e o projeto candidato tem como principal linguagem Python, o valor dessa *feature* será 5.
- *Commits*: Um *commit* é um registro de alteração em um ou mais arquivos em um projeto. A quantidade de *commits* de um projeto pode ser utilizada como indicador de atividade daquele projeto, onde projetos que tem uma grande quantidade e frequência de *commits* podem ser considerados projetos ativos, de forma que podem atrair mais colaboradores. Utilizamos a quantidade de *commits* de um repositório.
- *Watchers*: *Watchers* são usuários que decidiram "observar" um projeto, passando a ser notificados de qualquer atividade que ocorra no projeto, como por exemplo *commits* e novas *issues*. Geralmente, o usuário vira *watcher* de projetos que tem interesse de saber de suas atividades e, possivelmente, virar um contribuidor. Utilizamos o número de *watchers* do projeto.
- *Followers*: Usuários no GitHub podem ter seguidores (*followers*), bem como seguir outros usuários. Usuários com muitos seguidores tendem a ser bastante ativos na plataforma, com contribuições relevantes e que tem mais chances de se interessar e colaborar em novos projetos. Utilizamos o número de *followers* do usuário.

4. EXPERIMENTO

4.1 Coleta de dados

Coletamos dados de projetos e usuários do GitHub dos anos de 2012-2013 disponíveis no GH Archive[4] através do BigQuery[5]. Também utilizamos API do GitHub[6] para coletar informações adicionais de repositórios e usuários. Decidimos utilizar estes anos pois são um dos primeiros disponíveis na base de dados do GH Archive, e não têm uma quantidade massiva de dados como os anos mais recentes, tornando o experimento viável. De modo que a quantidade de dados utilizada é suficiente para podermos analisar e extrair fatores que representam o interesse de usuários em projeto. Devido à grande quantidade de repositórios disponíveis, decidimos aplicar um filtro, coletando apenas projetos com pelo menos 05 (cinco) desenvolvedores.

Ao todo, foram coletados dados de 9854 repositórios, dos quais removemos repositórios que foram excluídos, que são *fork* de um outro repositório, que não tem pelo menos 2 *watchers*, não possuem descrição ou que não possuem informação sobre a linguagem de programação utilizada. Dessa forma, ao todo, foram selecionados 826 repositórios para serem utilizados no experimento.

Coletamos informações sobre todos os contribuidores (usuários que fizeram pelo menos um *commit* no projeto) dos 826

[4] https://www.gharchive.org/
[5] https://cloud.google.com/bigquery
[6] https://developer.github.com/v3/

repositórios selecionados, totalizando 12571 usuários. Removemos desse conjunto, os usuários excluídos, os considerados pelo GitHub como *bots,* os que não tinham nenhum repositório e os que não tem informação de localização em seu perfil, restando assim 3464 usuários.

4.2 Métricas

Para a avaliação dos modelos, utilizamos as métricas: NDCG [15], MAP [16] e Precision [16], que são métricas de qualidade amplamente utilizadas para teste de performance de modelos de LTR . Estas métricas são apresentadas a seguir.

Normalized Discounted Cumulative Gain (NDCG): NDCG é a versão normalizada do DCG, que indica a soma ponderada dos graus de relevância de itens ranqueados. NDCG normaliza o DCG utilizando o DCG ideal (IDCG), que é a medida do DCG para o melhor resultado de ranqueamento possível [15]. Seu valor varia 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, conforme mostrado na fórmula abaixo:

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

Mean Average Precision (MAP): MAP é uma métrica amplamente utilizada em Recuperação da Informação [9], correspondendo à média dos valores de *Average Precision* (AP) [16] para cada consulta. Sua fórmula é dada por:

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

Precision: Precision é a proporção de documentos relevantes recuperados. No contexto de recomendação, é a proporção de documentos relevantes entre os recomendados. Para avaliação dos modelos utilizamos *Precision@k,* que é a proporção de documentos relevantes entre o top-k [16], sendo definido por:

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

4.3 Configurações

4.3.1 Parâmetros dos algoritmos

Não foi feita uma testagem exaustiva com as diversas variações de parâmetros dos algoritmos de LTR a fim de encontrar a melhor combinação. Assim, decidimos utilizar os parâmetros *default* do próprio RankLib para treinamento dos modelos, conforme ilustrado nas tabelas 6.3.1, 6.3.2 e 6.3.3.

Parâmetros LambdaMART	Valor do Parâmetro
No. of trees	1000
No. of leaves	10
No. of threshold candidates	256
Stop early	100

Tabela 6.3.1: Parâmetros do LambdaMART.

Parâmetros Random Forest	Valor do Parâmetro
No. of bags	300
Feature-sampling	0.3
No. of trees	1
No. of leaves	100
No. of threshold candidates	256
Learning rate	0.1

Tabela 6.3.2: Parâmetros do Random Forest.

Parâmetros Coordinate Ascent	Valor do Parâmetro
Restart	5
MaxIteration	25
StepBase	0.05
StepScale	2.0
Regularized	False
Slack	0.001

Tabela 6.3.3: Parâmetros do Coordinate Ascent.

4.3.2 Divisão dos conjuntos de treino e teste

Utilizamos a proporção 80/20 para separação dos dados, 80% dos dados foram separados para o treinamento dos modelos e 20% para teste. A separação entre os conjuntos de teste e treino foi feita de maneira aleatória, e, para todos os algoritmos usados no experimento, foi utilizado o mesmo conjunto de treino e teste.

Os conjuntos de treino e teste, bem como os modelos treinados estão disponíveis em um repositório[7] no GitHub.

4.4 Baseline

Como *baselines* utilizamos duas técnicas: (i) o método de Regressão Linear [17], usado como algoritmo de LTR com abordagem *pointwise*, por ser um método simples e mais direto do que os outros algoritmos selecionados [13]. Sendo possível verificar com base nisso quanto uma abordagem mais elaborada para a tarefa de ranqueamento afeta os resultados; e (ii) uma ordenação randômica de listas, permitindo que as métricas para os algoritmos sejam mais intuitivas quando compará-las com uma simples ordenação aleatória de itens [3] [14].

[7] https://github.com/DayvsonNascimento/ltr_github

4.5 Ameaças

Não foi feito um *tunning* dos parâmetros utilizados nos algoritmos, optamos por usar os parâmetros *default.* Sendo assim, poderíamos ter resultados diferentes se tivéssemos variados os parâmetros dos algoritmos. Além disso, não utilizamos a base de dados mais recente disponível, de forma que os fatores que podem indicar o interesse de usuários em colaborar em projetos podem ter se alterado com o passar dos anos.

5. RESULTADOS

5.1 Resultados

Utilizamos as métricas propostas para avaliar os 3 modelos treinados utilizando os algoritmos LambdaMART, Random Forest e Coordinate Ascent no conjunto de testes. Calculamos NDCG@10, NDCG@20, MAP, P@10 e P@20 para cada um dos modelos. Os valores de 10 e 20 para k indicam o interesse em avaliar a recomendação dos top-10 e 20 projetos para os usuários, respectivamente. Valores menores que 10 seriam muito restritos, enquanto que uma recomendação com mais de 20 itens seria pouco útil em um cenário prático [3].

Variamos o uso das *features* nos algoritmos por meio da utilização de subconjuntos diferentes das *features* propostas. Adicionalmente, usamos ainda uma outra *feature*, a quantidade de *pull requests*, que são requisições para alteração em um projeto, porém não obtivemos melhoras significativas nos resultados, mantendo assim as oito *features* propostas inicialmente. Os resultados obtidos pelos modelos é mostrado na tabela 6.6.

Modelo	**NDCG @10**	**NDCG @20**	**MAP**	**P@10**	**P@20**
Lambda MART	0.1289	0.1604	0.1108	0.0254	0.0193
Random Forest	0.1246	0.1541	0.1081	0.0247	0.0185
Coordinate Ascent	0.1006	0.1441	0.0972	0.024	0.0181
Regressão Linear	0.0201	0.0327	0.028	0.0046	0.0049
Randômico	0.0078	0.0102	0.0102	0.0017	0.0014

Tabela 6.6: Métricas de avaliação para cada um dos algoritmos utilizados.

5.2 Análise

Dos algoritmos utilizados, o LambdaMART e o Random Forest foram os que obtiveram os melhores resultados para as métricas adotadas. O LambdaMART ainda apresentou performance levemente superior ao Random Forest. O Coordinate Ascent foi o algoritmo de LTR que teve o desempenho mais baixo dos utilizados, desconsiderando os *baselines,* principalmente em termos de NDCG@10, NDCG@20 e MAP, o que pode indicar que sua abordagem não foi tão eficiente para colocar os projetos mais relevantes em um *rank* maior na nossa base de dados.

Para os valores de *precision*, não houve diferença significante entre os três algoritmos, indicando que eles tiveram desempenhos similares na tentativa de recomendar projetos relevantes considerando o top-10 e top-20. Aumentando a quantidade do top-n itens considerados, a precisão diminui, o que indica que os projetos mais relevantes para o usuários estão sendo majoritariamente retornados entre os top-10. Para o NDCG@k, aumentando top-k de 10 para 20, conseguimos um aumento significante nessa métrica, principalmente utilizando o LambdaMART, onde há um aumento de 0.1289 para 0.1604, indicando, assim, que aumentando a quantidade de projetos considerados, a ordenação dos itens mais relevantes melhora. Já para MAP, a variação entre os 3 algoritmos de LTR selecionados é de cerca de 1%, o que mostra que estes tiveram desempenhos similares ao tentar recomendar projetos relevantes para os usuários.

Comparando com os baselines, vemos que houve um ganho considerável nas métricas, especialmente em NDCG@10, NDCG@20 e MAP, indicando que os algoritmos de LTR selecionados foram mais efetivos ao recomendar projetos relevantes.

Selecionamos o modelo Random Forest para calcular o peso das *features,* pois este permite o cálculo dos pesos de maneira eficiente e teve desempenho muito próximo do melhor modelo (LambdaMART) nos testes. Especificamente, utilizamos a quantidade de vezes que a *feature* é usada pelo modelo para este cálculo. Os resultados ordenados por peso são mostrados na Figura 2.

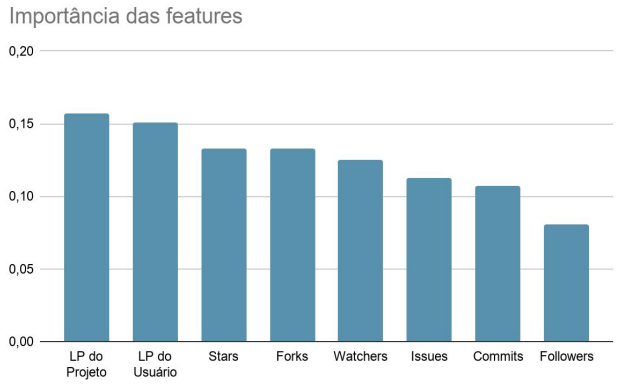

Figura 2: Importância das *features* para o treinamento do modelo Random Forest

No geral, as *features* tiveram pesos similares para o modelo. Em especial as *features* de Linguagem de Programação do projeto e do usuário (LP do projeto e LP do Usuário) foram as que tiveram maiores pesos. Tendo em vista que essas *features* representam o *match* de habilidade técnica entre projeto e usuário, podemos concluir que este é possivelmente um fator decisivo quando um usuário deseja contribuir em um projeto. As *features* que estão mais relacionadas com a quantidade de usuários que têm interesse no projeto (*Stars, Forks, Watchers*), apresentaram um peso bem similar. As *issues* e *commits,* que estão mais relacionadas com a atividade dentro do projeto, também tiveram peso similar. A quantidade de *followers* do usuário foi a *feature* que teve menor peso, indicando que esta *feature* não é um fator tão útil quanto os demais para capturar o interesse de usuários.

6. TRABALHOS RELACIONADOS

Neural Network for List-wise Ranking [3] (NNLRank) é um modelo de LTR desenvolvido para recomendar projetos no GitHub que um usuário é propenso a contribuir. NNLRank aprende um função de ranqueamento para dar um *score* para uma lista de projetos candidatos, recomendando o top-n projetos dessa lista para os desenvolvedores. O NNLRank utiliza 9 *features* derivadas de projetos e usuários que buscam capturar o padrão de interesse de desenvolvedores em ingressarem em projetos. Duas dessas *features* são comuns com nosso trabalho: quantidade de *commits* e linguagem de programação do usuário; e sete são diferentes: vínculo social do dono do projeto e usuário, data do primeiro *commit* no projeto, data do último commit no projeto, data de criação do projeto, qual empresa o usuário pertence e variação de tempo de ingresso de colaboradores em um projeto. O modelo é então avaliado e comparado com os algoritmos de LTR SVMRank [18], BPNet [19], e SVM[20], que utilizam diferentes abordagens (*listwise*, *pairwise* e *pointwise*). Também é feita a comparação utilizando como *baseline* listas de projetos candidatos ordenados randomicamente. O NNLRank teve uma performance superior aos outros algoritmos de LTR que foi comparado, particularmente com o segundo melhor modelo nos testes, SVMRank, apresentou um aumento de 0.359 para 0.462 na métrica de MRR [21], 0.351 para 0.454 na métrica de MAP@10, e 0.354 para 0.457 na métrica de MAP@20. Além disso, também apresentou tempo de execução bastante eficiente [3].

No trabalho de Cerqueira et al. [4], o problema de recomendação é tratado utilizando classificação. Foram utilizados algoritmos de classificação populares para prever se um usuário faria *fork* ou não de um determinado projeto, sendo este o critério considerado para a recomendação. Foram propostas inicialmente 23 feature de projetos e usuários, que foram utilizadas para treinamento de modelos usando os algoritmos de classificação. Das 23 features, 05 foram utilizadas em comum no nosso trabalho: *watchers*, *followers*, *commits*, *issues* 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* e *pull requests* por projeto; quantidade de *pull requests* de um projeto; quantidade máxima e média de *issues*; variação da periodicidade do projeto, contribuição diária em um projeto e atividade do projeto.

Entretanto, os resultados obtidos por Cerqueira et al. para as métricas de qualidade não foram tão satisfatórios, sendo assim foi selecionado um subconjunto de *features* dessas 23 considerando a importância que cada uma teve na primeira fase de treinamento. Foram treinados novos modelos com esse subconjunto e foi observada uma melhora nas métricas de qualidade dos modelos, para o XGBoost [22], que foi o modelo que obteve os melhores resultados (houve 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). Contudo, ainda existe margem para considerar outras *features* e métricas para treinamento e teste de modelos na tentativa de conseguir resultados de recomendação melhores.

7. CONCLUSÕES

A recomendação de projetos no GitHub mostra-se como uma tarefa bastante relevante e ao mesmo tempo complexa, tendo em vista a grande quantidade de projetos e usuários da plataforma, e a grande variedade que existe entre eles. Neste trabalho extraímos algumas *features* de projetos e usuários com a intenção de capturar fatores de interesse dos usuários, e utilizamos estas *features* em algoritmos de *learning to rank* com o objetivo de recomendar projetos relevantes aos usuário. Avaliamos os modelos treinados e comparamos com abordagens mais simples, verificando a efetividade de *learning to rank* no contexto de recomendação de projetos no GitHub. Os resultados obtidos são encorajadores no sentido que mostram que ainda há muito espaço para explorar outras *features*, algoritmos de LTR e configurações na tentativa de obter melhores resultados de recomendação.

Devido às limitações de tempo e recursos computacionais, não conseguimos utilizar uma base de dados maior. Além disso, não exploramos uma quantidade maior de *features* devido às limitações de tempo. Assim, resultados melhores possivelmente pudessem ser atingidos utilizando *features* que não foram exploradas neste trabalho.

Em trabalhos futuros, pretendemos explorar uma quantidade maior de *features*, com foco especial para *features derivadas*, que mostraram ter um efeito significativo na tentativa de capturar o interesse de usuários nos projetos [3] [4]. Além disso, existe margem para explorar uma quantidade maior de algoritmos de LTR, bem como algoritmos que já mostraram bons resultados no contexto de recomendação de projetos no GitHub [3]. Também avaliamos a possibilidade de expandir a variável de resposta para também considerar uma junção de *star* e *fork*.

Por fim, pretendemos utilizar uma base de dados maior e com os anos mais recentes disponíveis, de forma que possamos verificar fatores de interesse em projetos que não puderam ser verificados com uma base de dados menor. Através do uso dessa base de dados maior, imaginamos poder também validar os fatores já verificados.

8. AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Franklin Ramalho, pela orientação, apoio e confiança durante todo processo. Ao colaborador, Prof. Dr. Leandro Marinho, pelas sugestões e discussões no andamento do trabalho. Aos meus amigos e familiares por todo apoio e incentivo, e em especial a minha avó, Lenilda Silva, por sempre me encorajar e apoiar.

9. REFERÊNCIAS

[1] GitHub - The world's leading software development platform. Disponível em: <https://github.com/>. Acesso em: 25 nov 2019.

[2] The GitHub Blog. Disponível em: <https://github.blog/2018-11-08-100m-repos/>. Acesso em: 25 nov 2019.

[3] 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.

[4] Cerqueira, T., Marinho, L., Ramalho, F.: Feature Evaluation for Project Preferences Representation in GitHub. 2019. Dissertação (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.

[5] 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: 20 julho 2020.

[6] Understanding Random Forest. Dispoível em: <https://towardsdatascience.com/understanding-random-forest-58381e0602d2#:~:text=The%20random%20forest%20is%20a,that%20of%20any%20individual%20tree.>. Acesso em: 03 nov 2020.

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

[8] Learning to rank. Disponível em :<https://en.wikipedia.org/wiki/Learning_to_rank> Acesso em 20 jun 2020.

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

[10] Xia, F., Liu, T., Wang, J., Hang, L.: Listwise Approach to Learning to Rank - Theory and Algorithm - Microsoft Research Asia, Sigma Center, No.49 Zhichun Road, Haidian District, Beijing, 100190, P. R. China.

[11] The Lemur Project - RankLib. Disponível em <https://sourceforge.net/p/lemur/wiki/RankLib/>. Acesso em 28 jun 2020.

[12] Multiple Additive Regression Trees. Disponível em :<http://statweb.stanford.edu/~jhf/MART.html>. Acesso em 03 nov 2020.

[13] Burges, C.: From RankNet to LambdaRank to LambdaMART: An Overview. Microsoft Research Technical Report MSR-TR-2010-82.

[14] Tapper K.: Learning to rank, a supervised approach for ranking of documents - Master Thesis in Computer Science - Algorithms, Languages and Logic, Chalmers University of Technology, University of Gothenburg Department of Computer Science and Engineering Göteborg, Sweden, June 2015.

[15] Wang, Y., Wang, L., Li, Y., He, D., Liu, T., Chen, W.: A Theoretical Analysis of NDCG Type Ranking Measures. Journal of Machine Learning Research. 30, 2013.

[16] Evaluation measures (information retrieval). Disponível em: <https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval)>. Acesso em: 04 nov 2020.

[17] Linear Regression. Disponível em: <http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm>. Acesso em: 07 nov 2020.

[18] Support Vector Machine for Ranking. Disponível em: <http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html>. Acesso em: 09 nov 2020.

[19] Hecht-Nielse, R.: Theory of the backpropagation neural network in Neural Networks for Perception. Washington, DC, USA, 1989.

[20] Support Vector Machine for Complex Outputs. Disponível em: <http://www.cs.cornell.edu/people/tj/svm_light/svm_struct.html>. Acesso em: 09 nov 2020.

[21] Mean reciprocal rank. Disponível em: <https://en.wikipedia.org/wiki/Mean_reciprocal_rank>. Acesso em: 07 nov 2020.

[22] A Gentle Introduction to XGBoost for Applied Machine Learning. Disponível em: <https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/>. Acesso em: 09 nov 2020.