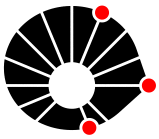

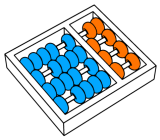

Uma Aplicação Prática de MABs em Sistemas de Recomendação

Matheus Neves de Jesus Oliveira Santos *Helio Pedrini*

Relatório Técnico - IC-PFG-23-33
Projeto Final de Graduação
2023 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Uma Aplicação Prática de MABs em Sistemas de Recomendação

Matheus Neves de Jesus Oliveira Santos* Helio Pedrini†

Resumo

Este trabalho propõe uma técnica de ordenação de itens em lista, baseada em MABs (*Multi-Armed Bandits*), para um inédito aplicativo em dispositivo móvel. O aplicativo consiste em um agregador de eventos universitários que pretende recomendar aqueles que são mais relevantes. O estudo desenvolveu um método de testes baseado em hipóteses relacionadas ao funcionamento do aplicativo, além da aplicação desses testes em dois algoritmos de ordenação. Os algoritmos abordados modelam as listas nos *Click Models*: *Document Based* e *Cascade Based*. Este relatório descreve o algoritmo a utilizado e a arquitetura do sistema.

1 Introdução

Considerando a realidade dos estudantes universitários, encontrar o próximo evento para aproveitar o final de semana pode ser desafiador visto que os mesmos estão distribuídos entre diversas plataformas, como Cheers, Byma, Blacktag, Sympla, entre outros. Essa escolha poderia ser melhor tomada caso houvesse um centralizador de eventos que fosse capaz de ranqueá-los por relevância e apresentá-los aos usuários.

O problema consiste, portanto, em aplicar MABs (*Multi-Armed Bandits*) para sistemas de recomendação de eventos para usuários com gostos heterogêneos e de diferentes localizações.

A proposição resultante do presente trabalho é utilizar o ecossistema Spotted, que é composto por diversas páginas do Instagram, como por exemplo, @spotted_unicamp, @usp_spotted, @spotted_puccamp, @usprp_spotted, entre outras, com o intuito de ser um "correio elegante" acadêmico, sendo utilizado para esclarecer dúvidas sobre a universidade, informações sobre disciplinas oferecidas, divulgação de festas e até mesmo para encontrar alguém, conseguindo ter um alto engajamento e mais de vinte mil seguidores.

1.1 Objetivos

Os principais objetivos deste trabalho são dois. O primeiro consiste na investigação de técnicas para ordenação de eventos mais relevantes em uma região, para um dado usuário.

*Instituto de Computação, Universidade Estadual de Campinas, Campinas, SP, 13083-852
†Instituto de Computação, Universidade Estadual de Campinas, Campinas, SP, 13083-852

Além disso, acerca desse primeiro tema, ampla revisão bibliográfica é realizada para identificar técnicas possíveis. A partir das abordagens identificadas, serão realizadas simulações a fim de que se escolha a mais adequada. O segundo objetivo diz respeito à proposição de uma arquitetura de software que favoreça a aplicação da abordagem escolhida e que possibilite, por exemplo, a atualização do modelo na menor latência possível (angariando os benefícios do aprendizado online). A aplicação será implementada em uma nova aplicação do ecossistema Spotted, sendo que dados públicos e dados próprios serão utilizados.

1.2 Problema

Visando à proposição de uma estratégia adequada à ordenação de eventos, primeiro se faz necessário compreender adequadamente o problema e os dados que estão disponíveis para serem utilizados.

A respeito do problema, tem-se a seguinte configuração (conforme elucidado na Figura 1): para uma dada região, para um dado perfil de usuário, este será servido com uma lista de $(0, 1, 2, \ldots, N)$ eventos, em que $N \lesssim 120$. No entanto, considerando que algumas regiões podem ter mais ou menos eventos, estabeleceu-se, para a proposição dos testes, que o número de eventos em uma dada região pode ser de 20, 60 ou 120.

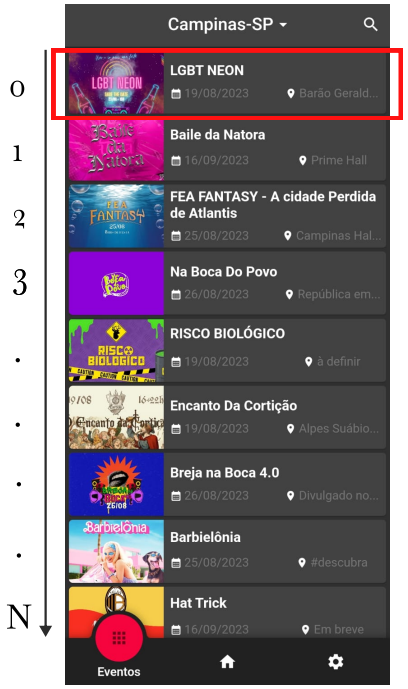

Figura 1: Ilustração dos eventos a serem ordenados.

Além disso, faz-se relevante entender quais são os dados disponíveis na abordagem online do problema, isto é, quais são os dados que se tornam disponíveis ao agente conforme este interage com o ambiente. Para isso, também se faz necessário entender como é o fluxo pelo qual o usuário passa ao acessar o aplicativo. Esse fluxo foi elaborado na Figura 2.

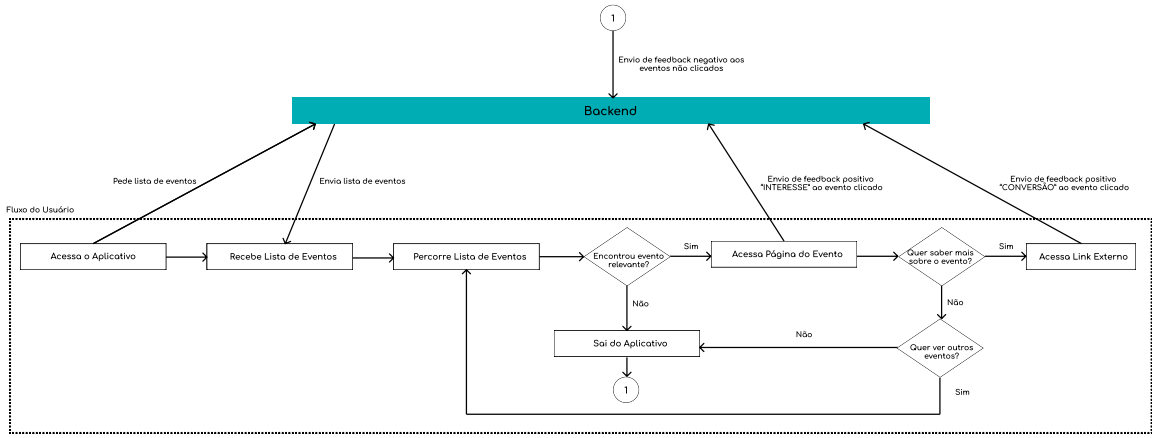

Figura 2: Fluxo do usuário.

Sendo assim, conforme a Figura 2, um usuário que acessa o aplicativo, requisita ao *backend* do sistema uma lista de eventos. Em seguida, percorre a lista e busca por um evento que seja de seu interesse. A forma como se modela a percepção dos usuários acerca dos eventos relevantes é chamado *Click Model* e será abordado a seguir.

Caso encontre um evento na lista que parece interessante, o usuário clica no evento, acessa a página do evento e envia um *feedback* positivo ao *backend* acerca daquele evento (*feedback* positivo 1: "Interesse"). Ao acessar a página do evento no aplicativo, consegue saber mais detalhes sobre aquele evento (como descrição completa, atrações, preços) e decidir observá-lo na bilheteria onde o evento está sendo anunciado, o que implicaria uma saída do aplicativo. Esse evento também envia um *feedback* positivo ao *backend* (*feedback* positivo 2: "Conversão").

Além disso, há a opção de voltar à lista de eventos. Caso nenhum evento chame a atenção do usuário, este pode sair do aplicativo e enviar um *feedback* negativo ao *backend* acerca de todos os eventos que visualizou, mas não demonstrou interesse.

Quatro últimas considerações acerca do problema são que: (i) com o passar do tempo, espera-se que alguns eventos da lista aconteçam e sejam substituídos por eventos novos; (ii) existe a hipótese de que eventos que estão na iminência de acontecer têm acrescida relevância se comparados a eventos que têm algum tempo para acontecer; (iii) o número de vezes que o modelo será atualizado por dia provavelmente impactará no desempenho das estratégias que serão analisadas a seguir; (iv) nenhum evento será deixado de fora da ordenação.

1.3 Estrutura do Texto

O texto está organizado como segue. A Seção 1 refere-se à introdução, onde os objetivos e o problema são apresentados. A Seção 2 descreve os conceitos abordados durante o presente trabalho. A Seção 3 apresenta os desenvolvimentos a respeito dos algoritmos abordados e os testes propostos. Os resultados desses testes são descritos na Seção 4. Finalmente, as

conclusões deste trabalho são apresentadas na Seção 5.

2 Conceitos Relacionados

Esta seção descreve brevemente os principais conceitos relacionados a este projeto.

2.1 Reinforcement Learning

Quando submetidos a um ambiente incerto e dinâmico, pode ser que algoritmos de aprendizado supervisionado ou não supervisionados não sejam a melhor opção. Nesses casos, pode-se fazer uso de algoritmos de aprendizado por reforço (*Reinforcement Learning*), que é uma classe de aprendizado e algoritmos baseados no reforço e refere-se a formas de melhorar a eficácia por meio de tentativa e erro.

Quando formulado matematicamente, ele pode ser considerado como um problema de otimização com o objetivo de encontrar uma ação ou estratégia de produzir ações que seja ótima em um sentido definido. Esse paradigma de aprendizado costuma ocorrer em situações em que ter os dados catalogados para o aprendizado, por exemplo, pode ser cara ou até mesmo impossível [1].

Os MABs são um problema clássico de *Reinforcement Learning* e chamam a atenção pela alta gama de aplicações atuais no mercado como, por exemplo, na escolha de imagens de capa para títulos na Netflix [2] ou na otimização de ranqueamentos de itens relevantes na Expedia [3].

2.2 Multi-Armed Bandits

Os MABs (*Multi Armed Bandits*) são estratégias de tomada de decisão em meio à incerteza. Sendo assim, essas estratégias buscam capturar o melhor resultado possível em uma aplicação, enquanto exploram um conjunto de possibilidades (esse dilema é chamado *explore* $\times$ *exploit trade-off*).

Considere a situação em que um usuário de aplicativo pode receber apenas um de quatro tipos de botões diferentes. Propor uma estratégia visando priorizar o botão que resultará na maior taxa de cliques para quais perfis de clientes é um problema que pode ser endereçado utilizando MABs.

Somado a isso, grandes empresas de tecnologia, como Spotify e Amazon, utilizam *bandits* (como são comumente chamados) para a construção de sistemas de recomendação a seus usuários [4].

2.3 Reward

A recompensa (*reward*) é o *feedback* recebido após uma interação com o ambiente, geralmente aparecendo na forma de sucesso ($reward = 1$) ou não sucesso ($reward = 0$) [5].

2.4 Regret

Faz-se necessário encontrar um modo de medir a eficácia de uma estratégia de tomada de decisão em meio à incerteza e de avaliar a qualidade de suas escolhas.

A forma como se define o *regret* é a diferença entre o *reward* total esperado capturado por uma dada estratégia após n rodadas e o *reward* total esperado capturado por outra estratégia após n rodadas. Sendo assim, pode ser considerada uma métrica que é calculada em relação a "algo", e esse algo - durante o presente trabalho - será uma estratégia que sempre escolhe um *ranking* de forma ótima, isto é, com base em sua atratividade real [5].

Apesar de existirem outras métricas voltadas à avaliação das estratégias, no presente trabalho, utilizaremos uma métrica semelhante ao *regret* como forma de avaliar os algoritmos testados. Dessa forma, além de permitir a avaliação de uma estratégia, essa métrica, assim como o *regret*, permite certa comparação entre a métrica de diferentes estratégias, de modo que se possa escolher a mais adequada.

2.5 Rank-Bandits

O objetivo do algoritmo é recomendar uma lista de itens, e o único *feedback* que recebe é na forma de cliques do usuário, de modo que o algoritmo, por si só, é capaz de avaliar suas próprias recomendações. Sendo assim, o objetivo do algoritmo é maximizar o número esperado de cliques.

Além disso, uma possível abordagem para recomendar os *rankings* poderia ser um arranjo A_N^K, em que N é o número total de itens e K é o número de itens que serão recomendados e encontrar qual desses arranjos maximiza o número de cliques.

No entanto, em uma situação com 20 itens em que 10 serão recomendados ($N = 20$ e $K = 10$), o número de *arms* será grande demais e se tornará impraticável, pois $A_{20,10} = \frac{20!}{10!}$.

Uma forma de simplificar a situação é por meio da forma como a interação do usuário com a lista é modelada, como se chamam os *Click Models* [5].

2.6 Click-Models

A forma como se modela a interação do usuário com a lista recomendada simplifica a tratativa para o problema, assim como define a forma como o *feedback* será percebido e passado para o modelo. Sendo assim, os *Click Models* modelam a probabilidade de clique em um item tendo como base algumas características.

Uma das formas mais simples de modelar é chamada *Document-Based Model*. Nesse *Click Model*, a probabilidade de clique em um item depende unicamente da atratividade do item.

Uma forma consideravelmente mais rebuscada de modelar é chamada *Cascade Model* e consiste em fazer a probabilidade de clique uma função tanto da atratividade do item quanto da posição. Além disso, define que o usuário percorre a lista de forma sequencial e, ao clicar no enésimo item, define que o usuário não demonstrou interesse pelos itens 0 ao enésimo (ou seja, $reward = 0$), enquanto mostrou interesse no n-ésimo ($reward = 1$) [5].

Para os itens posteriores, ou seja, não analisados, não se tem valor para o *reward*.

3 Método

A partir daqui, pretende-se elencar e aplicar dois algoritmos, visando entender o comportamento desses algoritmos, dada a configuração do problema apresentada.

O primeiro é um algoritmo proposto pelo autor (referido aqui por *DocumentBasedByAuthor*), a fim de que sirva como base de comparação. O segundo chama-se *CascadeBanditUCB1*. Em seguida, serão apresentados os fatores de teste, que são heurísticas propostas pelo autor e pelas quais os algoritmos serão comparados. Por fim, dois testes finais abordando todos os fatores também serão aplicados aos algoritmos. A métrica que possibilitará essa comparação entre os algoritmos é a distância da ordenação proposta da ordenação ótima. A formulação matemática está apresentada em seguida.

È importante ressaltar que cada um dos testes aplicados foram submetidos às mesmas condições. Ou seja, tinham que "descobrir"o mesmo valor de conversão ótimo iteração a iteração. Além disso, cada teste foi realizado vinte e cinco vezes, cada uma com diferentes dados gerados de conversão real e *reward* rodada a rodada. O resultado final apresentado será a média observada da métrica após passar pela configuração de teste, em um total de vinte e cinco vezes diferentes.

Os valores de conversão ótimo foram estimados e consistem em números aleatórios gerados no intervalo de 0,5% e 2,5%.

3.1 DocumentBasedFromAuthor

A inteligência por trás dessa forma de ordenar é uma das formas mais simples presentes na literatura. Esse algoritmo modela a lista e simplifica a atratividade de um item para ser função apenas do CTR (*Click Through Rate*) observado. Desse modo, foi proposto pelo autor que a parte de *exploit* do algoritmo consistisse em uma ordenação decrescente de CTR, enquanto a parte de *explore* consistia em uma equação cujo valor decai ao longo do tempo, ou, mais especificamente $e^{-t/10}$.

Por fim, define-se uma métrica atratividade como $\max(\text{CTR}, e^{-t/10})$, sendo que a lista passa a ser sugerida a partir da ordenação decrescente do valor atratividade.

3.2 CascadeBanditUCB1

O CascadeUCB1 aplica o *Click Model* Cascade Model como forma de contabilizar *feedbacks*, somado ao UCB (*Upper Confidence Bound*), como forma de escolher quais itens aparecem antes. Sendo assim, a ordenação ocorre por ordem decrescente de limite superior do intervalo de confiança de cada evento, sendo que esse intervalo é reduzido a cada nova rodada na qual se tem *feedbacks*. Desse modo, a parte *explore* do algoritmo consiste em utilizar os limites superiores do intervalo de confiança, enquanto a parte *exploit* consiste em ordená-los em ordem decrescente de UCB.

Os testes desenvolvidos aqui foram realizados a partir de uma adaptação desse algoritmo em linguagem de programação Python, disponibilizada no GitHub [8].

3.3 Métrica Utilizada: Distância da ordenação ótima

A cada rodada, calcula-se, para cada posição na lista, qual foi o evento recebido vs qual seria a sua posição ideal. As distâncias são somadas e o resultado, por sua vez, é registrado para aquela rodada. É como se fosse uma "perda" observada em cada rodada.

Para cada experimento, apresentamos tanto a distância naquela rodada, no primeiro gráfico, quanto a distância acumulada após n rodadas. Formulando matematicamente:

$$\sum_{i=1}^{K} |pos_{observada} - pos_{verdadeira}|,$$

em que K é o número de eventos mostrados na lista.

3.4 Heurísticas de Teste

Esta subseção descreve informações relevantes sobre as heurísticas de teste.

3.4.1 Número de Eventos a serem Ordenados na Lista

Estima-se que o número de eventos em uma dada região possa variar entre $20 \lesssim K \lesssim 120$, de modo que para essa heurística se escolha um valor K.

3.4.2 Número de Visitas no Aplicativo Diariamente

Logo que começar a ser utilizado em uma região, o aplicativo contará com poucos acessos, até que se mostre útil para a população daquele lugar. Esta heurística é responsável pelo número de cliques disponíveis numa dada região. Pode ser que um valor muito baixo incapacite o modelo de convergir rapidamente em razão do baixo número de cliques disponíveis.

3.4.3 Número de Rventos Ciclados Semana a Semana

Um comportamento esperado do aplicativo é que, com o passar dos dias, alguns eventos aconteçam, fazendo-os deixar de existir na listagem, enquanto outros novos sejam adicionados, passando, então, a integrar a lista. Essa heurística busca reproduzir esse comportamento.

3.4.4 Frequência de Atualização do Modelo

Pode ser que atualizar o modelo a todo momento seja custoso para a solução final. Pode ser que, ao agregar os *rewards* e utilizá-los para atualizar o modelo periodicamente seja uma decisão com custo-benefício adequado. Pode ser que a perda em desempenho seja ínfima perante a complexidade adicionada ao sistema como um todo (ou não).

3.4.5 Acréscimo de Relevância dos Eventos na Iminência de Acontecerem

É esperado também que, na iminência da ocorrência de um evento, este se mostre mais relevante do que anteriormente, isto é, a sua relevância ao usuário seja inversamente proporcional ao número de dias até o evento. Buscou-se simular esse efeito acrescendo 0,5% na relevância real do evento uma semana antes deste acontecer.

3.4.6 Teste Final

O teste final consiste em definir parâmetros para cada uma das heurísticas e testá-los juntos. Para o teste final 01, definiu-se: número de eventos = 60; 1000 visitas no aplicativo diariamente; número de eventos ciclados por semana = 20%; frequência diária de atualização do modelo, com acréscimo de relevância de eventos na iminência de acontecerem.

4 Resultados

As heurísticas supracitadas foram aplicadas e os resultados observados foram os seguintes. Define-se o caso base como: o teste foi realizado durante 60 dias; o número de eventos em uma dada região era 60; o número de visitas diárias é 1000; o modelo é atualizado imediatamente após cada rodada.

4.1 Caso Base

Ao observar a Figura 3, referente ao experimento no caso base, entende-se que ambos os algoritmos convergiram para uma ordenação próxima da ótima rapidamente, mas que um dos algoritmos obteve resultado um pouco melhor do que o outro, apesar de bastante parecidos.

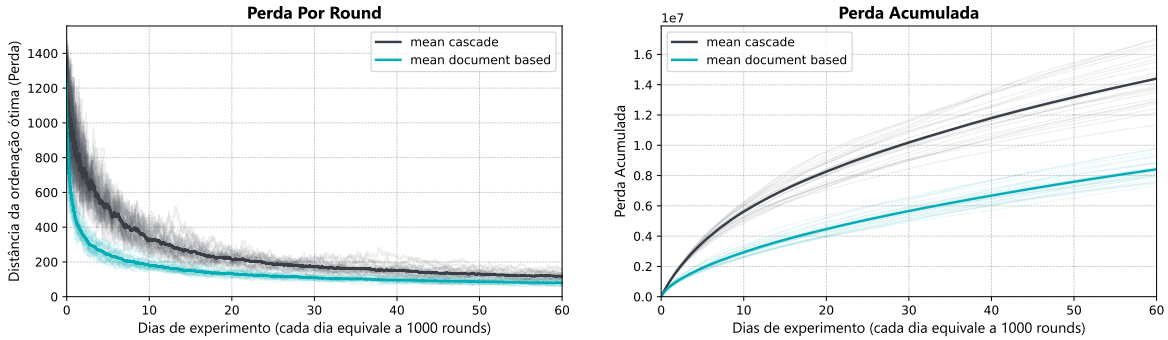

Figura 3: Resultado do caso base.

4.2 Caso Base com Apenas 20 Eventos

Com poucos eventos na lista, os algoritmos convergiram para um ordenação próxima da melhor possível, além de que o desempenho dos dois foi semelhante (ainda mais do que no

caso base). Isso foi observado por meio da Figura 4.

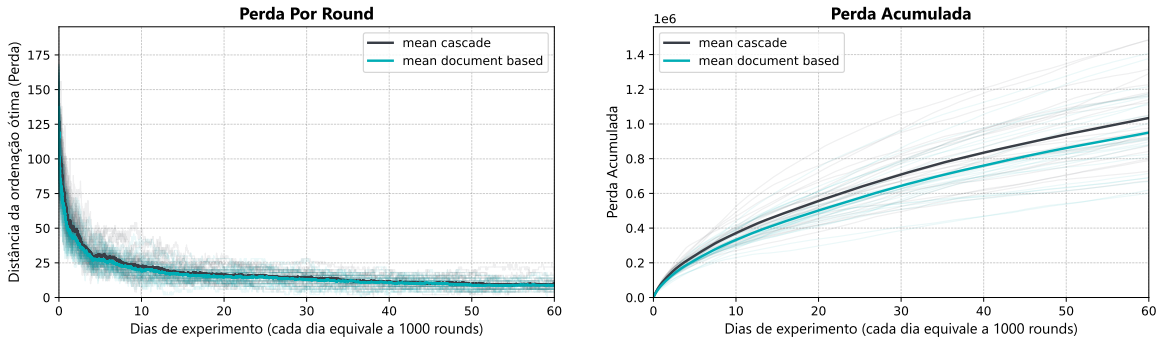

Figura 4: Resultado do Caso 20 eventos na Região.

4.3 Caso Base com 120 Eventos

Por meio da Figura 5, entende-se que ambos os algoritmos chegam a uma ordenação mais próxima da ordenação ideal. No entanto, para o *document based* acontece de forma muito mais veloz. Aqui, percebe-se que a diferença de performance dos casos foi maior se comparada aos casos anteriores.

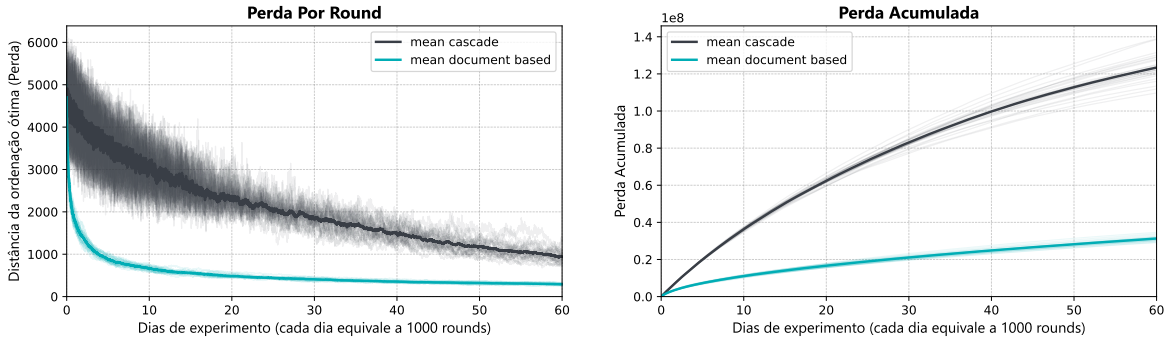

Figura 5: Resultado do Caso 120 eventos na Região.

4.4 Caso Base com 100 Visitas Diárias

Ao observar o resultado com um número menor de visitas diárias (consequentemente, um número menor de *feedbacks* diários), por meio da Figura 6, entende-se que ambos os algoritmos chegaram a uma ordenação mais próxima da ótima.

4.5 Caso Base com 10000 Visitas Diárias

Observado pela Figura 7, ao aumentar o número de visitas diárias (consequentemente, um número maior de *feedbacks* diários), a ordenação se aproxima da ótima em menos dias, o

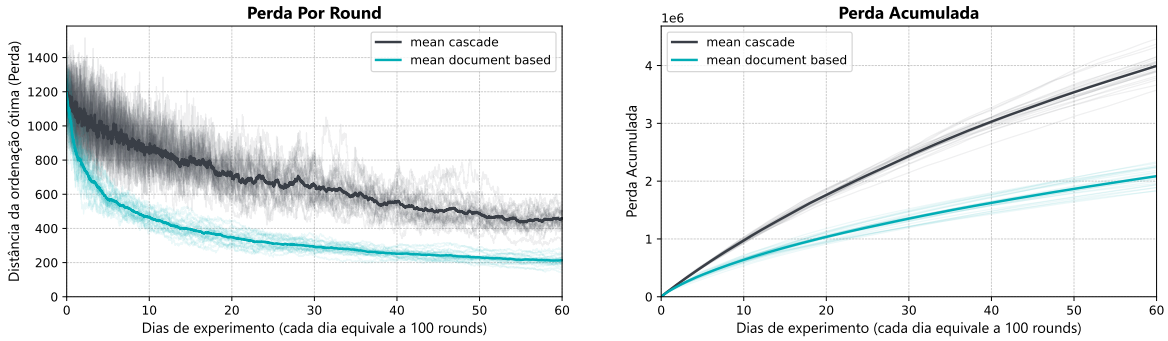

Figura 6: Resultado do caso com 100 visitas diárias no aplicativo.

que condiz com o alto número de *feedbacks* recebidos diariamente.

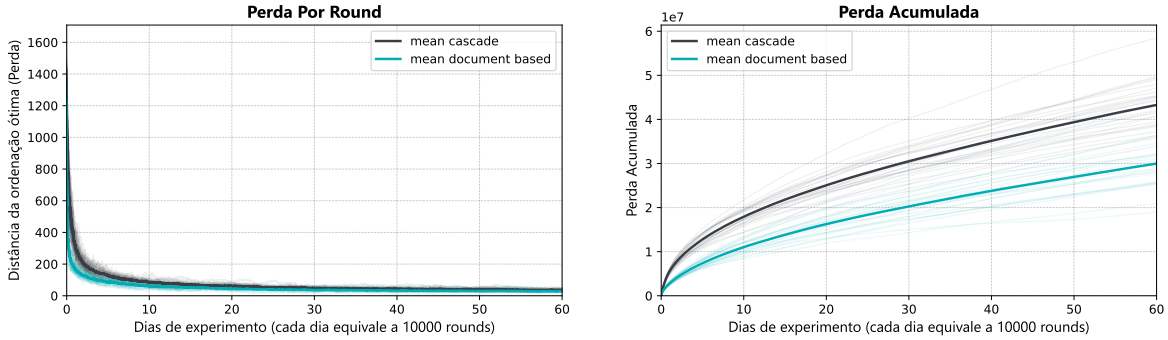

Figura 7: Resultado do caso com 10.000 visitas diárias no aplicativo.

4.6 Caso Base com Ciclagem de 20% por Semana

A Figura 8 exemplifica a situação em que a cada semana, alguns eventos deixam de existir na ordenação, enquanto outros novos eventos assumem esses lugares. Esse teste mostrou que tal comportamento dificulta aos algoritmos encontrar a melhor ordenação possível.

Percebe-se que, ainda assim, o *document based* foi capaz de manter a qualidade de seus resultados, enquanto o *cascade* alcança um resultado bom no início, mas com o passar do tempo perde qualidade em suas decisões.

4.7 Caso Base com Periodicidade de Atualização do Modelo Diária

Por meio da Figura 9, observa-se o caso em que os *feedbacks* só atualizavam o modelo uma vez ao dia. Os dois modelos foram capazes de recomendar uma ordenação próxima à ótima, apesar de que existe uma perda maior diária associada à periodicidade de passada dos *feedbacks* para que atualizem a tomada de decisão dos algoritmos.

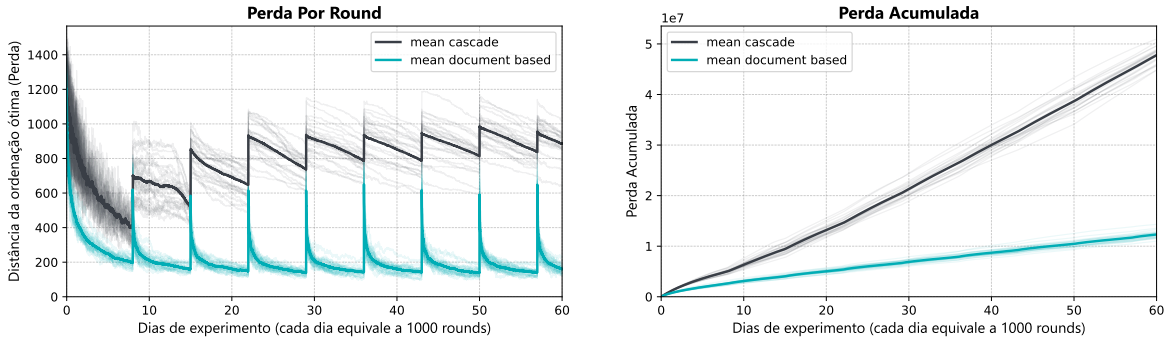

Figura 8: Resultado do Caso base com ciclagem de 20% por semana.

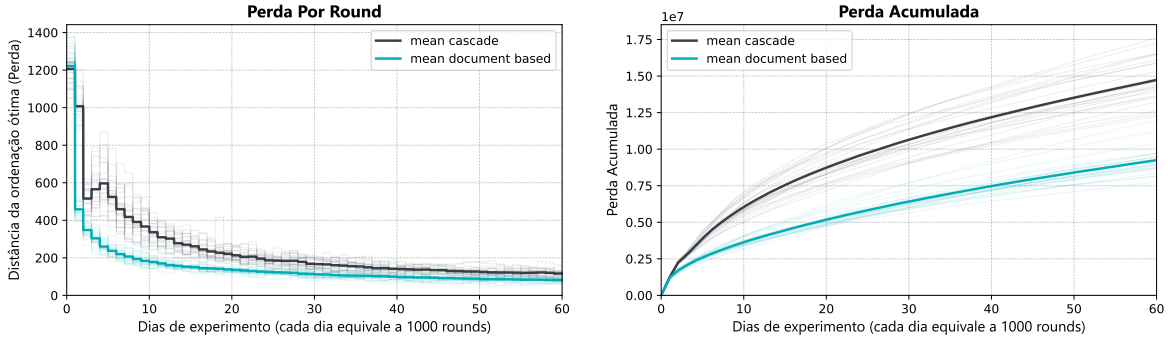

Figura 9: Resultado do caso atualização do modelo diária.

4.8 Caso Base com Acréscimo de 0.5% de Conversão Real de Eventos na Iminência de Acontecerem

Pela Figura 10, observa-se que o aumento de relevância dos eventos com o passar do tempo não é um fator esperado por ambos algoritmos, de modo que a qualidade da decisão de ambos é minimizada dado esse efeito.

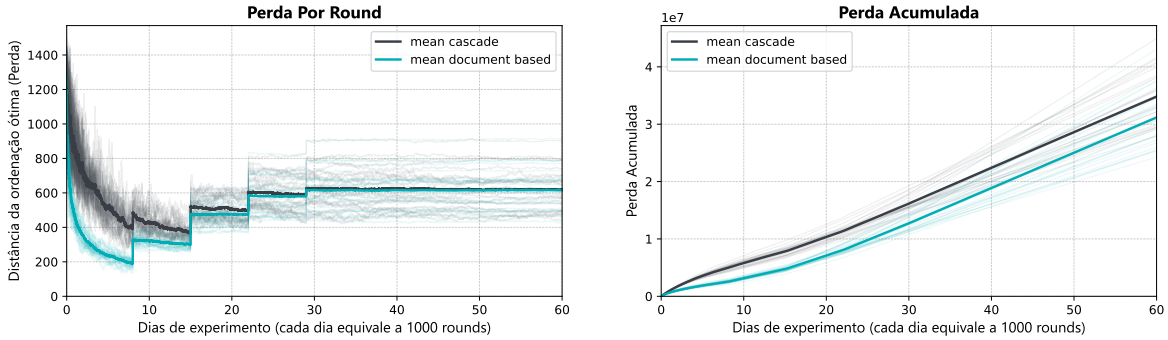

Figura 10: Resultado do caso mudança de gostos.

4.9 Teste Final

O teste final, cujo resultado está apresentado na Figura 11, envolveu todos os fatores em um único teste. Percebe-se que a qualidade de escolha do *cascade*, dada essa configuração, é pouco satisfatória, enquanto a apresentada pelo *document based* é satisfatória, apesar de que talvez pudesse ser melhor.

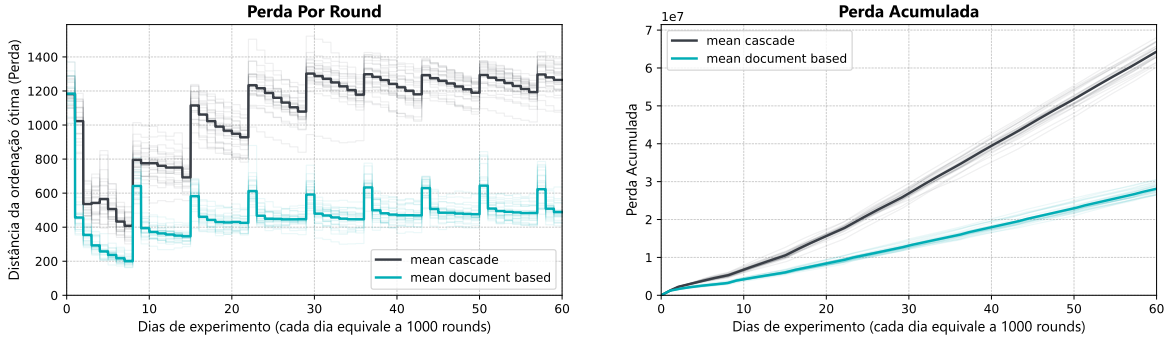

Figura 11: Resultado do caso de teste final.

5 Conclusões

Com a observação dos resultados. percebe-se que os algoritmos foram capazes de encontrar alguma ordem nos testes, sendo que em alguns casos foram mais desafiadores do que outros. Enquanto o *DocumentBasedFromAuthor* foi capaz de desempenhar melhor nas simulações, o *CascadeUCB1* teve dificuldades em alguns casos, como na heurística de ciclagem, assim como na situação final. Isso pode ter ocorrido devido à especificidade do problema.

Após observar os resultados, percebeu-se que o fator intervalo de confiança do *CascadeUCB1* - sendo que a ordenação decresce de CTR_*observado* + *intervalo de confiança* - demorava a se tornar menor do que o intervalo ao qual o CTR foi estimado. Sendo assim, durante a maior parte dos experimentos (principalmente aqueles que envolviam a substituição de eventos), o que mais influenciava no ranqueamento proposto era o fator intervalo de confiança ao invés do CTR observado até então.

O algoritmo que se mostrou melhor após os testes será aplicado na ordenação dos eventos por região no aplicativo futuramente. Vale comentar também, que, sendo a criação de testes uma das grandes realizações do presente trabalho, torna-se mais fácil testar outros algoritmos futuramente e incorporá-los ao aplicativo.

A forma como o algoritmo será aplicado é a seguinte: quando um usuário acessar o aplicativo, uma lista será designada a ele, junto de um identificador único de sessão. Qualquer evento de clique ou de conversão será encaminhado ao *backend* em C# e armazenado. Ao final do dia, os registros diários não lidos e as listas sorteadas naquele dia serão utilizados para atualizar o modelo.

Os códigos utilizados nos experimentos realizados neste trabalhos foram desenvolvidos pelo autor e estão disponíveis no GitHub, por meio do link: `https://github.com/`

mtnevess/TFG/.

Como futuros esforços, pretende-se testar novos algoritmos utilizando esse mesmo mecanismo de testes desenvolvido, além de buscar uma melhoria nas recomendações de listas por também considerar os gostos dos usuários.

Referências

[1] OMIDVAR, Omid; ELLIOTT, David. *Neural Systems for Control*. In: BARTO, Andrew. Chapter 2 - Reinforcement Learning. 1997. Disponível em: https://doi.org/10.1016/B978-012526430-3/50003-9. Acesso em: 15 out. 2023.

[2] MEDIUM. Netflix TechBlog. In: CHANDRASEKHAR, Ashok; AMAT, Fernando; BASILICO, Justin; JEBARA, Tony. *Artwork Personalization at Netflix*. 2022. Disponível em: https://netflixtechblog.com/artwork-personalization-c589f074ad76. Acesso em: 13 nov. 2023.

[3] MEDIUM. Expedia Group Technology. In: MARCHINI, Andrea. *How to Optimise Rankings with Cascade Bandits*. 2022. Disponível em: https://medium.com/expedia-group-tech/how-to-optimise-rankings-with-cascade-bandits-5d92dfa0f16b. Acesso em: 13 nov. 2023.

[4] ZIYOU, Yan. *Bandits for Recommender Systems*. Eugeneyan.com, 2022. Disponível em: https://eugeneyan.com/writing/bandits/. Acesso em: 10 set . 2023.

[5] LATTIMORE, Tor; SZEPESVARI, Csaba. *Bandit Algorithms*. Cambridge University Press, 2020. Disponível em: https://tor-lattimore.com/downloads/book/book.pdf. Acesso em: 10 out. 2023.

[6] KVETON, Branislav; SZEPESVARI, Csaba; WEN, Zheng; ASHKAN, Azin. *Cascading Bandits: Learning to Rank in the Cascade Model*. Proceedings of the 32nd International Conference on Machine Learning, 2015. DOI https://doi.org/10.48550/arXiv.1502.02763. Disponível em: https://arxiv.org/abs/1502.02763. Acesso em: 20 out. 2023.

[7] RADLINSKI, Filip et al. *Learning Diverse Rankings with Multi-Armed Bandits*. Conference on Machine Learning, Cornell University, 2008. DOI https://doi.org/10.1145/1390156.1390255. Disponível em: https://www.cs.cornell.edu/~rdk/papers/icml08.pdf. Acesso em: 22 out. 2023.

[8] AMINEDIRO. *CascadeBandit*. 2023. Disponível em: https://github.com/AmineDiro/CascadeBandit. Acesso em: 22 Oct. 2023.