



Centro Pi

Centro de Projetos
e Inovação IMPA

**RELATÓRIO 7º WORKSHOP DE SOLUÇÕES
MATEMÁTICAS PARA PROBLEMAS
INDUSTRIAIS**

BIG DATA

7º Workshop de Soluções Matemáticas para Problemas Industriais - Projeto BigData

Roberto Andreani, Vilmar Oro Boff, Victor Cunha, Samuel Carlos de Souza Ferreira, Amanda Figur, Lucas Nissenbaum, Thiago Ramos

ABSTRACT

Determinar o preço ótimo a que deve ser vendido um produto é fundamental para qualquer empresa, em particular, em lojas de varejo. Qualquer que seja o segmento, essas lojas costumam atender um grande número de pessoas, que invariavelmente possuem sensibilidades diferentes a preços. A Big Data propôs o sistema Price-O com o objetivo de determinar o preço ótimo de cada item em cada loja, e alcança este objetivo por meio de um sistema de reinforcement learning, balanceando as etapas de exploration-exploitation para a avaliação de preços. Este sistema depende de um número muito grande de amostras para convergir em um tempo viável. Por este motivo, ele não é viável em um contexto de uma rede pequena de lojas. O objetivo deste relatório é considerar algumas avenidas pelas quais o Price-O poderia ser modificado para convergir em um menor número de amostras.

1. PRECIFICAÇÃO

Determinar o preço ótimo a que deve ser vendido um produto é fundamental para qualquer empresa, em particular, em lojas de varejo. Qualquer que seja o segmento, essas lojas costumam atender um grande número de pessoas, que invariavelmente possuem sensibilidades diferentes a preços. Naturalmente a percepção do valor de um item é diferente para cada pessoa. Isso significa que, oferecer um produto a um preço muito acima do valor percebido pelo cliente pode implicar em um número muito baixo de produtos vendidos. Nesse caso, mesmo o maior lucro por produto pode não ser suficiente para compensar a perda em vendas. Por outro lado, oferecer o mesmo produto a um preço muito baixo leva a um lucro abaixo do desejado por produto, por mais que haja um aumento significativo no número de vendas. O segredo então se encontra na diferenciação de preços: cada consumidor é "convidado" a pagar por um determinado item o preço que condiz com sua percepção de valor para o mesmo item, dadas as restrições particulares da empresa em relação àquele item (e.g., preço mínimo, margem desejada, etc.). A otimização deste preço é crucial para maximizar o resultado de qualquer companhia.

Em um contexto de uma rede com um grande número de lojas e produtos, o preço ótimo pode ser encontrado através de experimentação usando um framework de Reinforcement Learning com multi-armed bandits, ou ainda, através da extrapolação de curvas de elasticidade. Estudos mostram que a diferenciação de preço tipicamente traz retornos da ordem de 4% a 10% em lucro bruto. Muitos setores já se valem dessa otimização de preços. Por exemplo, companhias aéreas diferenciam preços por horário do dia, dia da semana e uma série de outras variáveis. Gigantes da internet, como Amazon, praticam essa diferenciação de preço em uma escala altamente granular, chegando a diferenciar o preço do mesmo item várias vezes ao longo do dia.

Por outro lado, algumas áreas de negócios ainda estão para trás na adoção de estratégias similares. Um desafio particular aparece nos cenários com restrição de amostras, seja no número de lojas ou itens. Nestes cenários,

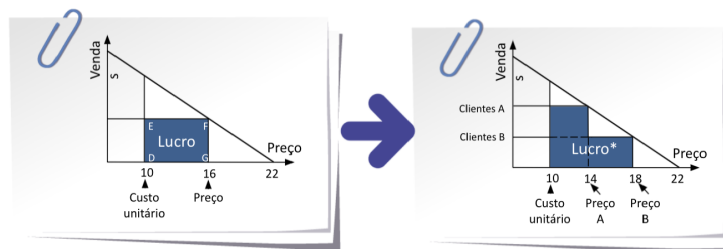


Figure 1. Otimização do lucro com diferenciação de preços

a experimentação é praticamente proibitiva, já que não se tem uma massa relevante de amostras. Por outro lado, essas empresas tipicamente não alteram seus preços com frequência, o que também dificulta o estudo de elasticidades.

A Big Data desenvolveu o seu produto de precificação dinâmica, chamado PriceO, para um cenário em que é possível usar experimentação. Este sistema tem como objetivo determinar o preço ótimo de cada item em cada loja, e alcança este objetivo por meio de um sistema de reinforcement learning, balanceando as etapas de exploration-exploitation para a avaliação de preços. Em linhas gerais, o produto se vale de um grande volume de variáveis explicativas (e.g., renda, fluxo de trânsito, índice de concorrência, clima, criminalidade, entre outros) para segmentar as lojas em diferentes clusters. Em seguida, uma abordagem baseada em Thompson Sampling é usada para a etapa de experimentação de preços. Nesse processo, preços campeões são recompensados, enquanto os preços perdedores vão ficando pelo caminho, até chegar ao subconjunto de preços ótimos.

Embora já executado com sucesso em diversos clientes, o produto PriceO da Big Data não é apropriado para cenários em que consideramos uma rede com um número pequeno de lojas. Como mencionado antes, este cenário deixa a experimentação quase proibitiva, dado o pequeno número de amostras e portanto o PriceO não é indicado para a obtenção dos preços ótimos.

Neste relatório, proporemos caminhos para otimizar este sistema no contexto de um pequeno número de lojas com poucas alterações históricas de preços. Quais modificações podem ser implementadas para alcançarmos uma solução de forma mais eficiente, ainda que esta solução não seja uma solução ótima global? Quais outros artifícios podem ser usados como surrogate nesse caso? É possível criar um bom modelo para o preço ótimo, que não dependa de experimentação, nesse cenário? Este projeto propôs algumas alternativas a serem consideradas neste desafio.

2. THOMPSON SAMPLING E REGRET NO CONTEXTO DE PRICING

Se soubéssemos com qual preço um produto traz mais rendimento, uma estratégia que traria o maior retorno a longo prazo seria sempre vender o produto a esse preço. Na prática não sabemos qual esse valor, que pode, inclusive, flutuar com o tempo e as variações do mercado.

Se, a longo prazo, queremos maximizar o retorno obtido com a venda dos produtos, precisamos fazer algumas experimentações. Não qualquer experimentação, mas uma que busque um equilíbrio entre *exploration* e *exploitation*.

O **Thompson Sampling binário** é um método de aprendizado por reforço (*Reinforcement Learning*) que usa Inferência Bayesiana para executar essa estratégia. Vamos explicar como ele funciona.

2.1 O algoritmo Thompson Sampling

Suponha que temos uma lista finita de preços para um produto e queremos descobrir, a longo prazo, qual preço traz melhores ganhos financeiros.

No início da nossa exploração, podemos escolher qualquer um dos preços para um produto. É como se todos tivessem a mesma taxa de sucesso. Representamos isso com uma função de distribuição *beta* $\beta(1, 1)$ para cada máquina, como visto na Figura 2.

O preço original do produto é separado como grupo de controle. Ao testar o desempenho de um preço, podemos estabelecer um critério para determinar se o preço foi um *sucesso* ou um *fracasso*. Neste caso, comparamos o lucro com o obtido por um grupo de controle. Se o grupo de controle teve um lucro menor que o preço sendo testado, consideramos o preço com bem sucedido para este cluster de lojas. Caso o grupo de controle tenha um lucro maior que o preço sendo testado, consideramos o preço como um fracasso para este cluster de loja. O fato de ser um sucesso ou fracasso é portanto dado por um indicador binário.

Ao fim da primeira rodada de exploração com um preço em um cluster de lojas, dizemos que nos casos de

- **sucesso:** há mais chances desse preço ser o melhor;
- **fracasso:** há menos chances desse preço ser o melhor.

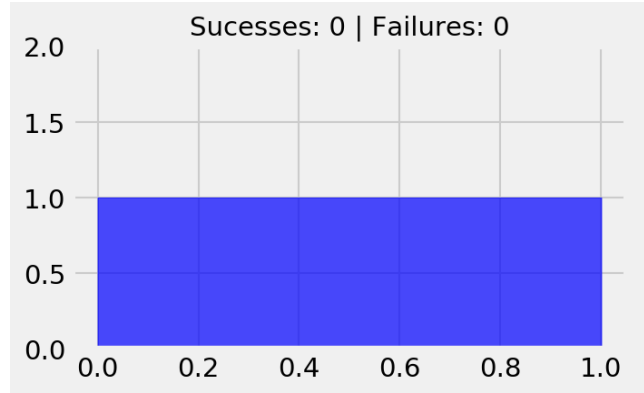


Figure 2. A função de distribuição $\beta(1, 1)$ dos preços. Se um preço não foi testado, a taxa de sucesso dele pode ser qualquer uma. Fonte: [1].

Passamos a representar a distribuição deste preço de acordo com a Figura 3. A função beta atualiza sua distribuição de acordo com os sucessos ou fracassos dos testes realizados. No caso de um sucesso, a distribuição indica maiores chances de acontecer um novo sucesso. No caso de um fracasso, as chances de um sucesso são consideradas menores para as próximas tentativas.

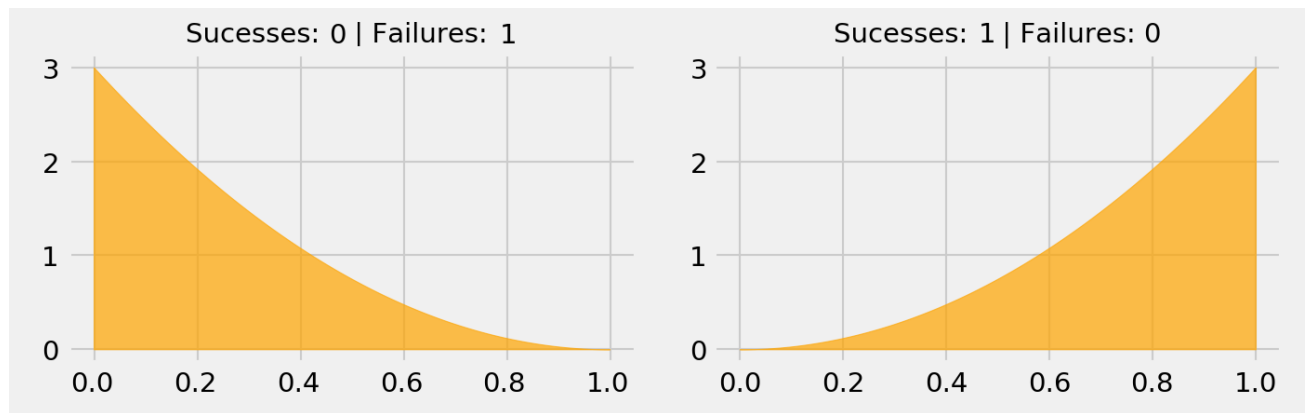


Figure 3. A função β atualiza seus valores de acordo com a marcação de sucesso ou fracasso de um teste realizado. Fonte: [1].

A cada rodada de testes um preço a ser testado é escolhido com base na distribuição β com parâmetro dado por $(n_s + 1, n_f + 1)$, onde n_s é o número de sucessos e n_f é o número de fracassos de cada preço específico. Depois de muitas rodadas, se uma máquina ou preço foi muitas vezes escolhida, a distribuição β dá uma boa ideia da taxa de sucesso esperada naquela máquina ou naquele preço, como na Figura 4.

O que acontece durante a execução do algoritmo é que, com a função β , a cada rodada inferimos a chance de sucesso de um preço:

- A cada rodada, escolhemos um preço baseados na probabilidade que inferimos anteriormente.
- Se um preço mostrar obter sucessos com maior frequência, a chance de o escolhermos numa próxima rodada de testes é maior.

Depois de algumas rodadas de testes, começamos a “ter uma ideia” de qual preço parece retornar melhores resultados e nossa tendência será ir nele mais vezes.

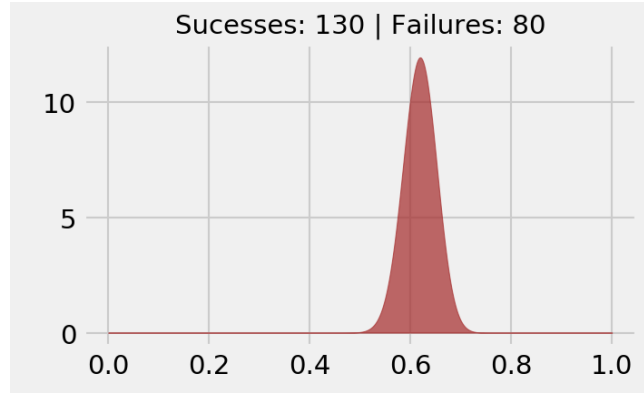


Figure 4. A função $\beta(131, 81)$ indica que esse preço tem uma taxa de sucessos próxima a 60%. Fonte: [1].

2.2 Regret

Podemos medir se esses resultados são realmente os melhores com o conceito de *regret*. Ele mede o arrependimento que temos ao escolher um preço.

Podemos o calcular a cada rodada da seguinte forma:

- Supomos qual seria o resultado do melhor preço caso soubéssemos o melhor preço (sucesso ou fracasso). Em outras palavras, usamos o preço caso tivéssemos o conhecimento da estratégia ideal.
- Comparamos com o resultado do preço escolhido.

Quando o resultado obtido no teste é pior do que o do melhor preço, ficamos arrependidos da escolha (e acumulamos *regret*). Neste sentido, se algoritmo deixar de acumular *regret* sabemos que ele passou que este passou fazer a melhor escolha de preço, ou uma solução equivalente. Vemos um exemplo disto na Figura 5.

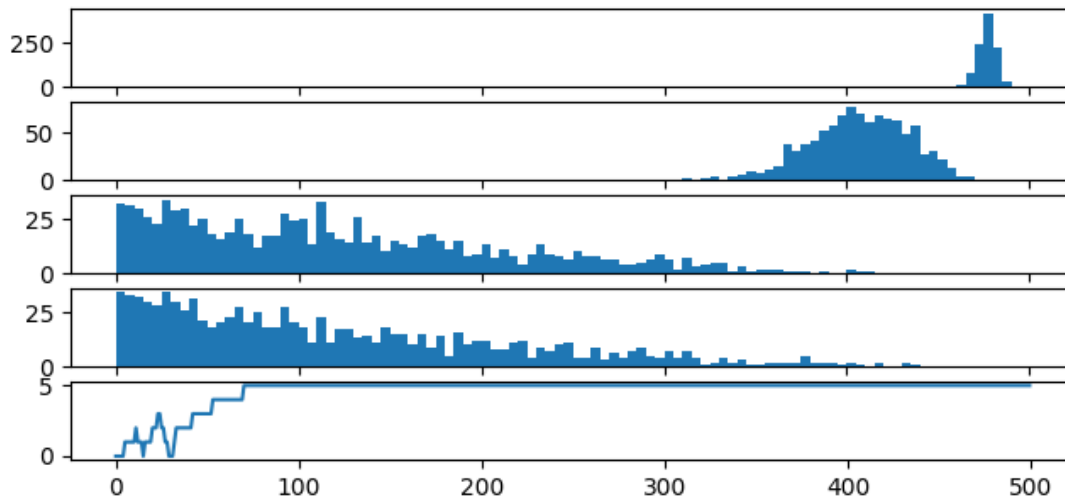


Figure 5. Nessa simulação do Thompson Sampling podemos ver que por volta das 100 iterações parou-se de acumular *regret*. Vemos também que o preço 1 tem uma taxa de sucesso inferida por volta de 90%. Isso significa que a partir do centésimo teste escolheu-se quase sempre o preço 1. A taxa de sucesso dos preços nessa simulação é, respectivamente: 95%, 70%, 10% e 20%.

3. THOMPSON SAMPLING NÃO-BINÁRIO

Nessa seção retrataremos uma variação do algoritmo de Thompson Sampling para precificar determinado produto com um passo de adaptação não-binário (ou seja, para resolver uma variação do Multi-Armed Bandit). Utilizamos de conceitos de economia juntamente com a inferência Bayesiana para obtenção das soluções. O algoritmo e sua implementação baseiam-se num artigo publicado no blog da empresa Grid Dynamics [2].

A adaptação ocorre quando olhamos para o problema Multi-Armed Bandit com uma perspectiva diferente. No caso, trocamos os braços das máquinas caça-níqueis por estimativas de demanda, essa por sua vez atrelada a um preço. Utilizando a Lei da Demanda da economia, temos que

$$\exists D(p), \text{ tal que } D(p) \text{ é decrescente (e convexa) em } p. \quad (1)$$

Ou seja, podemos tentar fazer estimativas iniciais para essa função de demanda e ir atualizando nossas crenças (inferência Bayesiana) para maximizar a receita em cada passo. Dessa forma, pretendemos achar a demanda ótima que, multiplicada com o preço atrelado a ela, maximize a receita.

Na primeira parte dessa subseção explicamos o algoritmo através de um exemplo ilustrativo, definimos o conceito de regret nesse contexto e mostramos resultados mais realistas; na segunda parte, descrevemos algumas sugestões de otimizações para um contexto de baixas amostras.

3.1 O Algoritmo e Regret

Começaremos com um exemplo ilustrativo. Devido a ausência de dados reais, iremos supor que existe uma curva de demanda em que podemos simular as estimativas em cada passo. No caso real da Big Data, os testes ocorreriam em cada subgrupo de estabelecimentos dentro de um mesmo cluster.

Exemplo Ilustrativo Seja um produto de um cluster de lojas previamente selecionadas. Desejamos testar os preços p de 1,99, 2,99, 3,99 e 4,99. Considere que nesse contexto vale a seguinte função de demanda $D(p) = 2 * 10^4 - 2p$, graficada abaixo:

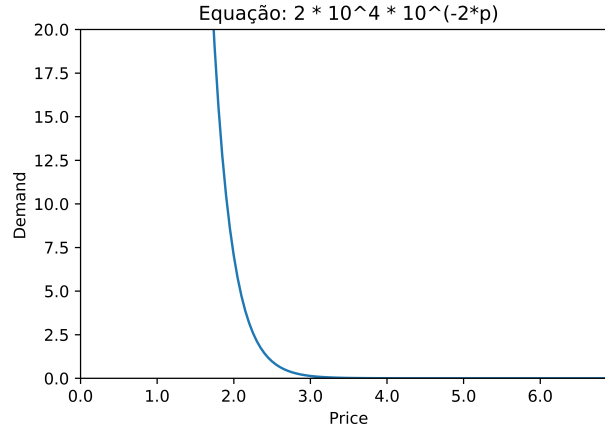


Figure 6. Demanda para o Exemplo Ilustrativo

Analisando previamente o cenário, vemos que a demanda cai drasticamente a partir de $p = 2$. Logo, para maximizar a função receita $R(\text{preço}) = \text{preço} \times \text{demanda}$, o algoritmo deve reconhecer que preços mais altos que 1,99 geram demandas muito baixas e por consequência receitas menores.

Como resolver esse problema considerando que temos acesso limitado a função de demanda (através um teste por iteração)? A solução a ser apresentada baseia-se numa variação do algoritmo de Thompson Sampling, onde uma função de distribuição $\text{Gamma}(\alpha_i, \beta_i)$ é usada como *prior* para estimar a demanda de todos os preços p_i

naquela iteração. Sendo assim, considere um chute em cada α_i e β_i a ser aplicado na função Gamma. Nesse exemplo, começamos com $\alpha_i = 6.00$ e $\beta_i = 1.00, \forall i$. Aplicando o valor esperado, temos

$$\mathbb{E}[\text{Gamma}(\alpha_i, \beta_i)] = \frac{\alpha_i}{\beta_i} = \frac{6.00}{1.00} = 6.00$$

Dessa forma, dentre os preços de 1,99, 2,99, 3,99 e 4,99, selecionamos aquele atrelado a demanda que maximiza a relação preço $\times$ demanda. Nesse exemplo, na primeira iteração, o preço escolhido tende a ser 4.99, que está longe de ser o correto. Assim, testamos esse preço na função de demanda - que representa a realidade - e ela nos revela a verdadeira demanda para esse preço. Como o leitor pode notar através do gráfico 6, na realidade, tem-se uma demanda d_t baixíssima para o preço de 4,99. Com essa demanda d_t em mãos, fazemos o passo de atualização:

$$\begin{aligned}\alpha_i &\leftarrow \alpha_i + d_t \\ \beta_i &\leftarrow \beta_i + 1\end{aligned}$$

Para $d_t \approx 0$, na próxima iteração teremos

$$\mathbb{E}[\text{Gamma}(\alpha_i + d_t, \beta_i + 1)] = \frac{6.00 + d_t}{2.00} \approx \frac{6.00}{2.00} = 3.00$$

e caso o preço de 4,99 seja escolhido em algum passo novamente, seu valor irá diminuir. Portanto, podemos concluir que esse processo leva a uma estimação da demanda real (disponível na simulação através da função $D(p) = 2 * 10^{4-2p}$).

A adaptação das demandas em cada iteração desse algoritmo pode ser verificada nesse [GIF](#). O início e fim do algoritmo para esse exemplo estão na figura abaixo:

Note que a partir de um certo momento somente a demanda atrelada ao preço de 1,99 é escolhida (somente ela "se move"), indicando que o algoritmo acerta mais vezes o valor da demanda conforme passam as iterações. Isso evidencia uma boa relação equilibrada **exploration e exploitation**, sempre presente na inferência Bayesiana.

3.1.1 O Algoritmo

Dado um exemplo explicando o funcionamento do algoritmo, podemos defini-lo através do Algoritmo 1:

Algoritmo 1: Thompson Sampling Não-Binário

- 1: Definir o conjunto de preços $p = p_1, \dots, p_n$, os parâmetros $\alpha_1, \dots, \alpha_n, \beta_1, \dots, \beta_n$, a função de demanda $D(p)$ o número de iterações T .
- 2: Para $t = 1, \dots, T$ faça
 - 3: Simule as demandas $d_i(p_i) = \text{Gamma}(\alpha_i, \beta_i), \quad \forall i$.
 - 4: Ache o preço $p^* = p(k), k \in \{1, \dots, n\}$ que maximiza a receita:

$$p^* = \underset{p_i \in p}{\operatorname{argmax}} \sum_i p_i d_i(p_i)$$

- 5: Ofereça o preço ótimo p^* e observe a demanda real $d_t = \text{Poisson}(D(p^*))$.
- 6: Atualize os valores de α e β no k -ésimo preço

$$\begin{aligned}\alpha_k &\leftarrow \alpha_k + d_t \\ \beta_k &\leftarrow \beta_k + 1\end{aligned}$$

Duas observações podem ser feitas: a primeira diz respeito aos parâmetros α 's e β 's iniciais. Embora seja recomendado começar com $\alpha_1 = \alpha_2 = \dots = \alpha_n$ e $\beta_1 = \beta_2 = \dots = \beta_n$, pode-se *a priori* determinar quais devem ser esses valores iniciais para que as estimações das demandas convirjam mais rápido.

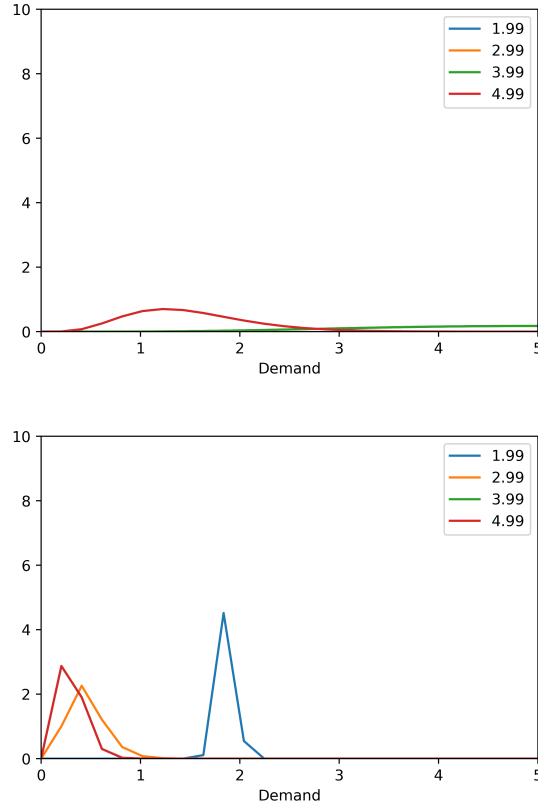


Figure 7. Demandas no exemplo.

Além disso, no passo 5 a demanda é simulada a partir de uma distribuição de Poisson em torno do valor de $D(p^*)$, pois uma distribuição de Poisson modelaria uma sequência de $D(p^*)$ eventos ocorrendo consecutiva e independentemente.

3.1.2 Regret

Uma característica em comum dos algoritmos de *Reinforcement Learning* é a presença de um **Regret**, ou seja, um arrependimento ao não escolher de maneira ótima numa iteração (ou cumulativamente até ela). Para o Thompson Sampling Não-Binário, podemos entender o Regret como **o arrependimento ao não escolher o preço ótimo numa determinada iteração**.

Sendo assim, temos a seguinte definição:

DEFINITION 3.1 (REGRET LOCAL E REGRET ACUMULADO). *Considere o vetor de preços $p = [p_1, \dots, p_n]$ e a função de demanda $D(p)$. Suponha conhecido o preço $p^* \in p$, onde*

$$p^* = \operatorname{argmax}_{p_i \in p} \mathbb{E}[\text{Receita}] = \operatorname{argmax}_{p_i \in p} \mathbb{E}[p * D(p)]$$

*Definimos o **Regret Local** na iteração t como*

$$r_t = p^* * d(p^*) - p_t * d_t,$$

*onde p_t e d_t são o preço e a demanda a ser considerado na iteração t , respectivamente. O **Regret Acumulado** na iteração t é definido como*

$$R_t = \sum_{j=1}^t r_j.$$

Para o exemplo da seção anterior, temos o comportamento da figura 8. Nas primeiras iterações, erramos algumas vezes o preço até convergir para 1,99. Note que a partir do momento em que estamos escolhendo o preço ótimo, a função Regret Acumulado estabiliza. De fato, se a partir de um certo tempo temos $p_t = p^*$, teremos $r_t = 0$ e $R_t = R_{t+k}$, $k = 1, \dots, n$.

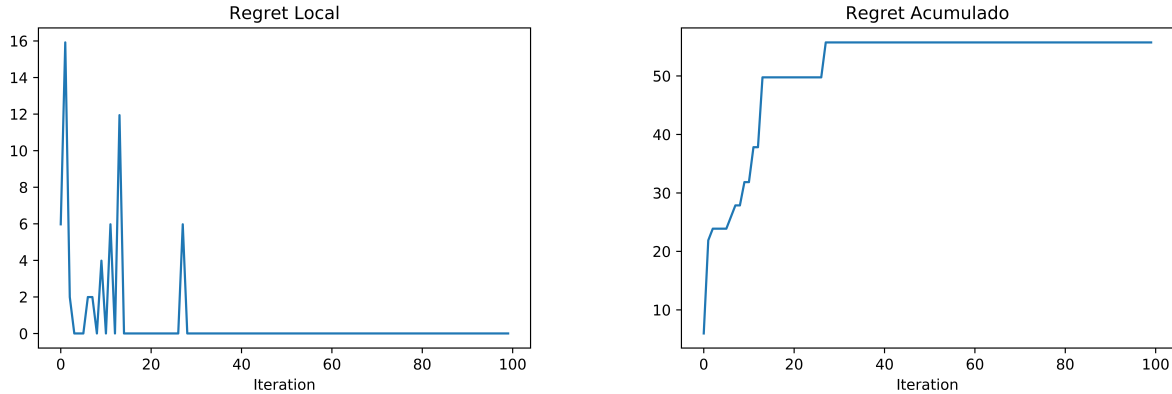


Figure 8. Regret para o exemplo ilustrativo.

Porém, a observação mais marcante fica a respeito do Regret Acumulado. Ele calcula quanto de receita (do contratante) foi deixado de ganhar em face a um contexto onde sabemos o preço ótimo desde o começo. Dessa forma, essa métrica pode ser utilizada não somente em simulações, mas também na realidade, auferindo a eficiência e acurácia do método. Esse fato pode servir como comparação entre cases do produto de precificação oferecido pela Big Data.

Exemplo Realista Seja um produto de um cluster de lojas previamente selecionadas. Desejamos testar os preços de 15, 20, 25, 30 e 40. Após alguns testes empíricos feitos pela equipe do projeto - juntamente com um gráfico de demanda fornecido pela empresa - suponha um contexto com a seguinte função de demanda $D(p) = 2e^{4 - \frac{p}{20}}$, presente na figura 9:

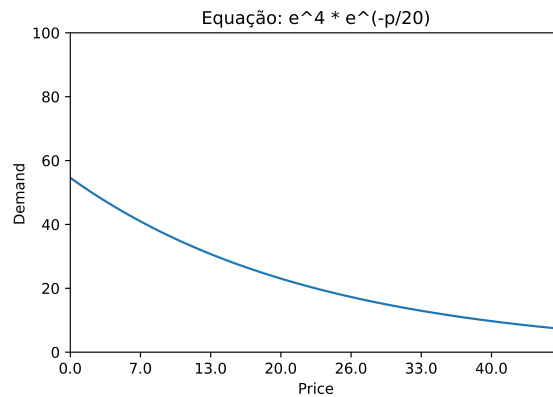


Figure 9. Demanda para o Exemplo Realista.

Nesse contexto, iniciamos com $\alpha_i = 15.00$ e $\beta_i = 1.00$, $\forall i$. Após quase 40 iterações, o preço converge para $p^* = 20.00$ com demanda de 20 unidades. Para esse caso, o regret local e regret acumulado estão na figura 10.

3.2 Sugestão num Contexto de Baixo Número de Amostras

Como verificado nesse último exemplo mais realista, o Thompson Sampling Não-Binário resolve o problema de precificação dinâmica balanceando o descobrimento de preços não testados sem muitas perdas em termos de

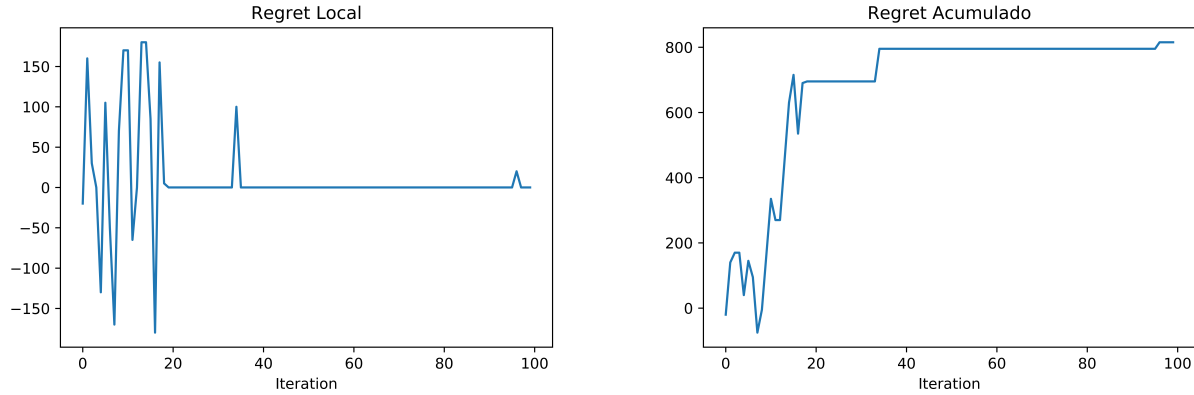


Figure 10. Regret para o exemplo realista.

receita. Nessa última subseção veremos três sugestões para ter melhor rendimento do algoritmo num contexto de poucas amostras:

- Aceleração da Convergência
- Influência em demandas vizinhas
- Elasticidade Preço-cruzada

3.2.1 Aceleração da Convergência

Esta proposta é bem simples. Basta definir um parâmetro $\omega > 1$ e inseri-lo no passo 6 do Algoritmo 1, multiplicando os parâmetros de atualização:

$$\begin{aligned}\alpha_k &\leftarrow \alpha_k + \omega * d_t \\ \beta_k &\leftarrow \beta_k + \omega * d_t\end{aligned}$$

Essa alteração faz com que a distribuição $\text{Gamma}(\alpha_k, \beta_k)$ da próxima demanda seja mais assertiva, requerindo menos passos para se ter uma demanda $d_k(p_k)$ consistente atrelado ao preço p_k .

Porém, um valor muito alto de ω pode levar a casos em que a condição de escolha de qual preço testar numa iteração do Thompson Sampling Não-Binário ($\text{argmax}_{p_i \in p} \text{Receita}$) caia sempre num preço sub-ótimo. Essa situação é semelhante a um caso de mínimo local. É como se afirmássemos com muita convicção que queremos uma demanda que pode ser sub-ótima e deixássemos de testar uma demanda que ainda não temos informação, prejudicando a busca de novas soluções ótimas.

3.2.2 Influência em Demandas Vizinhas

A influência em demandas vizinhas pode ser dividida em duas partes: uma recomenda a calibragem dos parâmetros iniciais e a segunda a atualização do passo. Para essa subseção, supomos que os preços estão ordenados em ordem crescente ($p_i < p_{i+1}$).

A primeira sugestão é aproveitar o comportamento decrescente da demanda em função do preço - dada pela Lei da Demanda - para iniciar os parâmetros $\alpha_1, \dots, \alpha_n$ e $\beta_1, \dots, \beta_n$ também decrescentemente, baseado em estimativas empíricas. A mudança no Algoritmo 1 ocorreria na inicialização, como abaixo:

1: Definir o vetor de preços $p = [p_1 < \dots < p_n]$, os parâmetros $\alpha_1 < \dots < \alpha_n$, $\beta_1 < \dots < \beta_n$, a função de demanda $D(p)$ o número de iterações T .

A segunda ideia seria aproveitar os valores de quantidade demandada em iteração para inferir nas demandas vizinhas de forma que se estamos atualizando $d_k(p_k)$, devemos ter um sétimo passo de atualização da forma:

$$\begin{aligned}\alpha_{k-1} &\leftarrow \alpha_{k-1} + \omega_1 * d_t \\ \beta_{k-1} &\leftarrow \beta_{k-1} + 1\end{aligned}$$

e

$$\begin{aligned}\alpha_{k+1} &\leftarrow \alpha_{k+1} + \omega_2 * d_t \\ \beta_{k+1} &\leftarrow \beta_{k+1} + 1,\end{aligned}$$

onde $\omega_1 > 1 > \omega_2$, visto que a função de demanda é decrescente. Isso implica que as atualizações de α_{k-1} e α_{k+1} devem manter essa proporção.

As duas ideias carregam consigo a desvantagem de demorarem mais iterações para convergir a um valor correto. De fato, como estamos mudando todas as demandas, elas flutuam com mais facilidade. Porém, sua grande vantagem é aproveitar a informação de um par (preço, demanda) noutro, **ideal para um contexto de baixo número de amostras**. Em suma, nas duas estratégias **estamos trocando mais iterações do algoritmo por um fluxo de informações entre demandas vizinhas**.

3.2.3 Elasticidade Preço-Cruzada

A terceira sugestão diz respeito a relação de substituição ou complementariedade entre dois bens num mesmo mercado. Mais especificamente, suponha que você é dono de um *fast-food* que vende um tipo de hambúrguer e um sabor de suco. A pergunta que explica a proposta a seguir é: será que um aumento no preço do hambúrguer implica em menos vendas do suco? Ou o contrário?

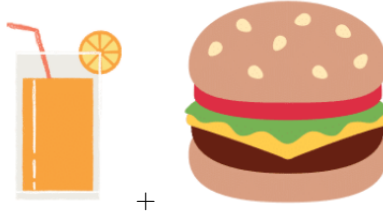


Figure 11. Bens Complementares.

No primeiro caso teremos uma situação de **bens complementares** e no contrário temos **bens substitutos**. Para medir essas relações, utilizamos o conceito de **elasticidade preço-cruzada**, dada pela fórmula

$$e_{d_x, p_y} = \frac{\partial d_x}{\partial p_y} \frac{p_y}{d_x}, \quad (2)$$

onde d_x é a função de demanda por um bem x e p_y é o preço de um bem y .

Em particular, para uma aplicação real da Big Data, vamos supor que estamos investigando lojas num mesmo cluster (mercado) que vendem esses dois produtos: sucos e hambúrgueres. Naturalmente, a informação de elasticidade pode ser verificada empiricamente. Suponha que foi encontrado que sucos e hambúrgueres são bens complementares. Logo, para $x = \text{suco}$ e $y = \text{hambúrguer}$ na equação 2, teremos uma queda na demanda de sucos caso o preço caia, ou seja

$$\frac{\partial d_x}{\partial p_y} < 0 \implies e_{d_x, p_y} < 0,$$

significando que dado um aumento de 1% no preço de y , a demanda em x diminui em $z\%$, $z > 0$. Analogamente, para bens substitutos, teremos $e_{x, p_y} > 0$.

Num cenário de aplicação do Thompson Sampling Não-Binário para os dois produtos, teremos dois algoritmos em execução. Dessa forma, a ideia seria completar uma iteração para o hambúrguer e, por exemplo, ao testar um preço que gerou uma demanda de hambúrgueres menor que o esperado, atualizar a demanda de sucos com uma certa queda também. O inverso valeria para aumentos de demanda nos hambúrgueres.

Outro cenário poderia ocorrer quando temos dois tipos de hambúrgueres, onde temos uma competição entre eles: pois dado que um dos hambúrgueres fica muito caro, os consumidores podem tender a escolher o outro. Portanto, um erro de precificação em um bem pode gerar aumento na demanda do seu "concorrente". Essas observações valem tanto para o Thompson Sampling Não-Binário quanto para a versão executada pela Big Data, de forma que as atualizações em preços dependentes podem ocorrer também quando deflagramos um fracasso no Thompson Sampling Binário.

3.2.4 Relação entre Clusters

Antes de se iniciar o método de precificação via Thompson-Sampling, há um processo de clustering das lojas através de um critério de semelhança de preços (por exemplo, separação entre lojas com preços mais caros e lojas com preços mais baratos).

Acreditamos que esse processo de clusterização cria uma estrutura extra no problema que pode ser utilizada para aproveitar resultados de um cluster para outro. Essa estrutura pode ser imaginada como um dicionário que traduz os preços entre diferentes clusters.

A ideia é modelar essa estrutura e utilizá-la para reduzir a complexidade do problema, que deixará de se comportar como vários métodos independentes de Thompson Sampling funcionando separadamente em cada cluster, para um processo onde o resultado de um cluster influencia nos resultados de um outro cluster, acelerando dessa forma a convergência.

Já existe uma literatura muito interessante sobre como trabalhar com esses problemas de Low-rank Bandits. Por exemplos, deixamos [3, 4] onde são apresentados garantias de convergência e metodologia de implementação, além de uma excelente discussão sobre o problema.

3.2.5 Outros Métodos Bayesianos

Para finalizar, gostaríamos de sugerir dois métodos Bayesianos que acreditamos que possam ser úteis no desenvolvimento e modelagem do problema apresentado.

O primeiro é o método de Otimização Bayesiana [5]. Este é uma abordagem para otimização de funções objetivo que levam muito tempo para serem avaliadas, assim como no nosso problema se considerarmos por exemplo a curva de demanda. Esse método cria uma aproximação da função objetivo, levando em conta uma quantificação de incerteza que podem ser muito útil na prática.

O segundo, é uma poderosa ferramenta chamada de Empirical Bayes [6]. A ideia desse método é usar ferramentas Bayesianas para estimar distribuições a priori através dos nossos dados. Note que isso pode ser muito vantajoso no nosso problema, já que se tivermos uma distribuição a priori melhor calibrada, podemos conseguir uma convergência mais rápida ao aplicar o Thompson Sampling.

REFERENCES

- [1] Marmerola, G. D., “Introduction to thompson sampling: the bernoulli bandit.” <https://gdmarmmerola.github.io/ts-for-bernoulli-bandit/>. Acesso em 10/09/21.
- [2] Katsov, I., “A guide to dynamic pricing algorithms.” <https://blog.griddynamics.com/dynamic-pricing-algorithms>. Acesso em 10/09/21.
- [3] Gopalan, A., Maillard, O.-A., and Zaki, M., “Low-rank bandits with latent mixtures,” (2016).
- [4] Jun, K.-S., Willett, R., Wright, S., and Nowak, R., “Bilinear bandits with low-rank structure,” (2019).
- [5] Frazier, P. I., “A tutorial on bayesian optimization,” (2018).
- [6] Casella, G., “An introduction to empirical bayes data analysis,” *The American Statistician* **39**(2), 83–87 (1985).