

Algoritmo de precificação de odds para um mercado de previsões P2P

Contexto

O seu projeto é um **mercado de previsões** em que os usuários apostam uns contra os outros. A plataforma apenas **intermedia** as transações, registra as apostas e resolve os eventos, sem tomar o outro lado das operações (ou seja, vocês **não são a casa**). Esse modelo P2P é semelhante a uma **bolsa de previsões** e tem duas características principais:

- O preço da ação "YES" ou "NO" deve refletir a *probabilidade* implícita de o evento ocorrer, e deve sempre estar entre 0 e 1. Uma previsão com preço 0,65, por exemplo, indica 65 % de chance de o evento acontecer.
- O preço precisa se ajustar de forma sensata à medida que novos usuários apostam. Para evitar manipulações, é desejável partir de um **prior** ("probabilidade a priori") baseado em dados externos e ir atualizando esse valor conforme entram novas apostas.

Nota: Em mercados com baixa liquidez, uma simples bolsa de ofertas (CDA) pode apresentar grandes spreads e baixa participação. Nesses casos, é comum utilizar um **market maker automatizado** (por exemplo, a regra de pontuação logarítmica de Hanson) para garantir liquidez¹. A LMSR atribui um preço de acordo com a função de custo $C(\vec{q}) = b \log \sum_i e^{q_i/b}$ e define o preço de cada resultado como $p_i(\vec{q}) = \frac{e^{q_i/b}}{\sum_j e^{q_j/b}}$ ². O modelo abaixo utiliza um mecanismo mais simples de agregação de apostas, mas é possível substituir ou complementar por LMSR.

Fluxo do algoritmo

1. **Fonte de probabilidade de referência** (`p_ref`): escolha uma probabilidade inicial para cada evento. Pode ser uma média ponderada de mercados externos (Polymarket, Kalshi) e de modelos internos. Uma forma simples é

```
p_{\text{text{ref}}} = w_1 p_{\text{text{poly}}} + w_2 p_{\text{text{kalshi}}} + w_3  
p_{\text{text{modelo}}}, \quad w_1+w_2+w_3=1.
```

1. **Peso do prior** (`priorWeight`): determine um peso N_0 que representa a "confiança" no prior. Valores mais altos tornam o preço inicial mais estável e evitam que pequenas apostas movam o mercado rapidamente. Por exemplo, se $N_0 = 100$, o prior equivale a 100 BRL de apostas iniciais.

2. **Estado de mercado por evento**: para cada evento mantenha as seguintes variáveis:

3. `stakeYes` : soma do valor (em BRL ou em unidades internas) apostado em "YES".

4. `stakeNo` : soma do valor apostado em "NO".

5. `p_ref` e `priorWeight` definidos no início.

6. Atualização após cada aposta:

7. Quando um usuário aposta v unidades em "YES", incremente `stakeYes` em v . Se apostar em "NO", incremente `stakeNo`.

8. Calcule a probabilidade atual (odds) usando um **agregador Beta**:

$$p_{\text{yes}} = \frac{p_{\text{ref}}}{N_0} + \frac{\text{stakeYes}}{\text{stakeYes} + \text{stakeNo}},$$
$$\quad p_{\text{no}} = 1 - p_{\text{yes}}.$$

Essa fórmula é a média de uma distribuição Beta; ela garante que o preço sempre se mantenha entre 0 e 1 e que a soma das probabilidades seja 1 ³.

9. Cálculo do preço do contrato:

10. Cada ação "YES" paga 1 BRL se o evento ocorrer e 0 BRL se não ocorrer. Assim, o **preço justo** de uma ação é p_{yes} . De forma análoga, o preço da ação "NO" é p_{no} .

11. **Opcional – LMSR para garantir liquidez:** se desejar oferecer liquidez constante como um market maker, substitua o passo 4 pelo cálculo de preço da **LMSR**. Nesse caso, registre q_{yes} e q_{no} como o número de ações vendidas de cada lado e escolha um parâmetro b que controla a sensibilidade do preço. O preço de "YES" seria

$$p_{\text{yes}} = \frac{e^{q_{\text{yes}}/b}}{e^{q_{\text{yes}}/b} + e^{q_{\text{no}}/b}},$$

e a mudança de preço ao vender Δq ações é dada pela diferença na função de custo $C(\vec{q}) = b \ln(e^{q_{\text{yes}}/b} + e^{q_{\text{no}}/b})$ ⁴ ₂.

Exemplo numérico

Suponha um evento com prior $p_{\text{ref}} = 0,60$, `priorWeight = 50`. Até o momento, os usuários apostaram 80 BRL no "YES" e 20 BRL no "NO". A probabilidade atual é

$$p_{\text{YES}} = \frac{50 + 80}{50 + 80 + 20} \approx 0.62.$$

Logo, o contrato "YES" sai por 0,62 BRL (62 centavos) e o contrato "NO" por 0,38 BRL.

Estrutura sugerida em TypeScript (Node)

Abaixo está um esboço de código que pode ser colado no Replit para implementar o algoritmo. Ele define duas classes: `PriceRef`, que fornece a probabilidade inicial a partir de fontes externas, e `EventState`, que mantém o estado das apostas e calcula as odds. A lógica está documentada e pode ser ajustada conforme necessário.

```
// PriceRef.ts
// Módulo responsável por buscar e combinar probabilidades externas.
export class PriceRef {
    /**
     * Obtém a probabilidade inicial de um evento combinando diferentes fontes.
     * Os pesos podem ser ajustados dinamicamente conforme a profundidade e a
     * "frescura" dos dados.
     */
    static async getInitialProbability(eventId: string): Promise<number> {
        const pPoly = await
this.fetchFromPolymarket(eventId); // preço implícito na Polymarket
        const pKalshi = await
this.fetchFromKalshi(eventId); // preço implícito na Kalshi
        const pModel = await this.internalModel(eventId); // estimativa do
seu modelo interno
        const w1 = 0.5, w2 = 0.3, w3 = 0.2;
        return w1 * pPoly + w2 * pKalshi + w3 * pModel;
    }

    private static async fetchFromPolymarket(eventId: string): Promise<number> {
        // TODO: implementar chamada à API/raspagem de dados da Polymarket
        return 0.5;
    }
    private static async fetchFromKalshi(eventId: string): Promise<number> {
        // TODO: implementar chamada à API/raspagem de dados da Kalshi
        return 0.5;
    }
    private static async internalModel(eventId: string): Promise<number> {
        // TODO: implementar modelo interno de probabilidade (por exemplo, baseado
em pesquisas)
        return 0.5;
    }
}
```

```

// EventState.ts
// Mantém o estado das apostas em um evento e calcula as odds.
export class EventState {
    private stakeYes: number = 0;
    private stakeNo: number = 0;
    private pRef: number;
    private priorWeight: number;

    constructor(pRef: number, priorWeight: number) {
        this.pRef = pRef;
        this.priorWeight = priorWeight;
    }

    /**
     * Registra uma nova aposta. O parâmetro side deve ser "YES" ou "NO";
     * stake é o valor apostado em unidades internas (por exemplo, BRL3 ou
     * tokens).
     */
    addBet(side: 'YES' | 'NO', stake: number): void {
        if (side === 'YES') {
            this.stakeYes += stake;
        } else {
            this.stakeNo += stake;
        }
    }

    /**
     * Calcula a probabilidade atual usando o agregador Beta.
     */
    getCurrentProbabilities(): { yes: number; no: number } {
        const weightedPriorYes = this.pRef * this.priorWeight;
        const totalYes = weightedPriorYes + this.stakeYes;
        const total = this.priorWeight + this.stakeYes + this.stakeNo;
        const pYes = totalYes / total;
        return { yes: pYes, no: 1 - pYes };
    }

    /**
     * Retorna o preço de negociação (odds) do contrato YES e NO.
     */
    getQuote(): { yesPrice: number; noPrice: number } {
        const probs = this.getCurrentProbabilities();
        return { yesPrice: probs.yes, noPrice: probs.no };
    }
}

// Exemplo de uso:
(async () => {

```

```

const eventId = 'eleicao-2026';
const pRef = await PriceRef.getInitialProbability(eventId);
const priorWeight = 100;
const event = new EventState(pRef, priorWeight);

// Usuário A aposta 50 no YES
event.addBet('YES', 50);
console.log(event.getQuote()); // { yesPrice: ~0.58, noPrice: ~0.42 }

// Usuário B aposta 30 no NO
event.addBet('NO', 30);
console.log(event.getQuote()); // { yesPrice: ~0.54, noPrice: ~0.46 }
})();

```

Possíveis extensões

- **Reputação de usuários:** cada aposta pode ser ponderada por um fator de reputação baseado no histórico de acertos. Isso permite que previsões de usuários mais assertivos tenham maior peso na probabilidade agregada.
- **LMSR como fallback:** se o volume de apostas for muito baixo, utilize o mecanismo LMSR descrito acima para oferecer cotações contínuas e garantir liquidez ⁴ ².
- **Gestão de taxas:** a plataforma pode cobrar uma taxa fixa ou percentual sobre cada aposta para cobrir custos operacionais e incentivar a participação.

Este algoritmo atende ao cenário em que a plataforma *não* é a casa: as probabilidades são atualizadas a partir das apostas dos usuários, agregadas a um prior obtido de fontes externas. A implementação em TypeScript facilita a integração em um aplicativo Node.js hospedado no Replit.

¹ predmarkets tutorial.ppt

<https://www.probabilityandfinance.com/GTP2014/Slides/Abernethy1.pdf>

² ³ marketsml.dvi

<https://www.jennw.com/papers/marketsml.pdf>

⁴ Cultivate Labs | Collective intelligence solutions using crowdsourced forecasting

<https://www.cultivatelabs.com/crowdsourced-forecasting-guide/how-does-logarithmic-market-scoring-rule-lmsr-work>