设计架构文档

核心算法的描述

1. **算法名称与概述**

   名称：协同过滤算法（基于相似度的推荐算法）

   概 gaichanpin 述：该算法以用户历史购买记录为依据，通过计算历史购买产品与产品仓库中产品的相似度，为用户推荐与其历史购买产品相似度较高的产品。此处使用归一化后的产品属性计算欧氏距离来衡量相似度。

2. **输入与输出**

   输入：

   1. “userTxData”：包含用户历史交易信息的数组，其中主要包含历史产品及其相关属性（“risk”、“rate”、“minInvest”）。
   2. “productData”：包含产品仓库中的产品及其相关属性。

   输出：

   更新后的“productData”，根据计算出的相似度从高到低排列。

3. **算法步骤**

   1. 提取用户购买的产品列表“userPurchase”；
   2. 计算所有产品属性的最大值和最小值；
   3. 定义归一化函数，将产品属性归一化到[0，1]范围；
   4. 对产品仓库中的每个产品，计算其与每个用户历史购买产品的欧氏距离（相似度）并相加，以此作为该产品与用户购买习惯的符合程度度量；
   5. 根据相似度总和对产品仓库中的产品进行排序；
   6. 更新“productData”，呈现更新排序后的产品列表。

4. **复杂度分析**

   1. 时间复杂度：
      a. 归一化最小值和最大值：O(m)（m 为“productData”的大小）。
      b. 计算相似度：O(m*n)（n 为“userPurchase”的大小）。
      c. 总时间复杂度：O(m+m*n)。
   2. 空间复杂度：O(m+n)。

5. 具体实现

```
useEffect( effect: () : void  => {
    const userPurchases : {...}[]  = userTxData.map(tx :  => tx.item);
    const similarityScores : any[]  = [];

    const minRisk : number  = Math.min(...productData.map(product : {...}  => product.risk));
    const maxRisk : number  = Math.max(...productData.map(product : {...}  => product.risk));
    const minRate : number  = Math.min(...productData.map(product : {...}  => product.rate));
    const maxRate : number  = Math.max(...productData.map(product : {...}  => product.rate));
    const minInvest : number  = Math.min(...productData.map(product : {...}  => product.minInvest));
    const maxInvest : number  = Math.max(...productData.map(product : {...}  => product.minInvest));

    5+ 用法  ♣ Colmeh
    const normalize = (value, min, max) => (value - min) / (max - min);

    productData.forEach(product : {...}  => {
        let totalSimilarity : number  = 0;

        userPurchases.forEach(userProduct => {
            const userRisk = userProduct.risk;
            const userRate = userProduct.rate;
            const userMinInvest = userProduct.minInvest;

            const normalizedRisk = normalize(product.risk, minRisk, maxRisk);
            const normalizedRate = normalize(product.rate, minRate, maxRate);
            const normalizedMinInvest = normalize(product.minInvest, minInvest, maxInvest);

            const normalizedUserRisk = normalize(userRisk, minRisk, maxRisk);
            const normalizedUserRate = normalize(userRate, minRate, maxRate);
            const normalizedUserMinInvest = normalize(userMinInvest, minInvest, maxInvest);

            const distance : number  = Math.sqrt(
                x: Math.pow( x: normalizedRisk - normalizedUserRisk,  y: 2) +
                Math.pow( x: normalizedRate - normalizedUserRate,  y: 2) +
                Math.pow( x: normalizedMinInvest - normalizedUserMinInvest,  y: 2)
            );
            const similarity : number  = 1 / (1 + distance); // 相似度越高，距离越近
            totalSimilarity += similarity;
        });

        similarityScores.push({ product, totalSimilarity });
    });

    similarityScores.sort( compareFn: (a, b) => b.totalSimilarity - a.totalSimilarity);

    const sortedProducts : any[]  = similarityScores.map(score => score.product);
    setProductData(sortedProducts);
}, deps: [userTxData,productData]);
```

技术架构图