算法核心逻辑

输入数据结构

ratings: Map[String, Map[Int, Double]]

外层键：用户ID（String）

内层键：电影ID（Int）

值：用户对电影的评分（Double）

表示所有用户的历史评分记录。

余弦相似度计算

公式：

实现细节：

仅计算共同评分电影的相似度，避免无效计算。

处理分母为零的情况，返回0.0。

推荐流程

步骤1：筛选相似用户

排除目标用户自身，计算与其他用户的余弦相似度。

取相似度最高的前k个用户（默认k=20）。

步骤2：收集候选电影

合并相似用户评分过的电影，过滤掉目标用户已评分的电影。

步骤3：预测评分

对每个候选电影，基于相似用户的评分加权平均计算预测分：

pred(u,m)=∑v∈Ssim(u,v)⋅rv,m∑v∈S∣sim(u,v)∣

pred(u,m)=∑v∈S​∣sim(u,v)∣∑v∈S​sim(u,v)⋅rv,m​​

按预测分排序，返回前n个电影（默认n=10）。

潜在问题与优化建议

1. 性能瓶颈

问题：

计算目标用户与所有用户的相似度时，时间复杂度为O(N)（N为用户数），若用户量大（如百万级），效率极低。

优化方案：

预计算相似度矩阵：离线计算用户相似度并存储，推荐时直接查询。

局部敏感哈希（LSH）：近似快速查找相似用户。

采样策略：仅计算部分活跃用户的相似度。

2. 候选电影评分预测的稀疏性

问题：

若候选电影的评分数据稀疏（如多数相似用户未评分），预测分可信度低。

优化方案：

设置最小阈值：仅保留至少被t个相似用户评分的电影（如t=3）。

引入全局平均分：对稀疏电影使用全局平均分补全。

3. 负相似度的处理

问题：

负相似度用户可能引入噪声（如反向兴趣），降低推荐质量。

优化方案：

过滤负值：仅保留相似度大于0的用户：

scala

.filter { case (\_, sim) => sim > 0 }

4. 冷启动问题

问题：

新用户（targetRatings为空）或无相似用户时，无法生成推荐。

优化方案：

兜底策略：返回热门电影、随机推荐或基于内容的推荐。

混合推荐：结合用户属性（如年龄、地域）初始化推荐。

5. 实时性不足

问题：

未考虑用户近期行为的重要性（如最近评分权重更高）。

优化方案：

时间衰减因子：在相似度计算中引入时间衰减函数（如指数衰减）。

6. 代码可维护性

问题：

相似度计算与推荐逻辑耦合，难以扩展其他算法（如基于物品的CF）。

优化方案：

抽象相似度接口：

scala

trait SimilarityCalculator {

def compute(u1: Map[Int, Double], u2: Map[Int, Double]): Double

}

依赖注入：通过构造函数传入不同的相似度实现（如皮尔逊系数）。