第六部分

重排

第 1 章 多样性

工业界的实践经验表明多样性对推荐系统的业务指标有非常显著的影响，多样性做得好，则用户使用推荐的时长、活跃用户数、用户留存等指标都会显著提升。如果你是推荐系统的典型用户，你一定会发现这条经验非常符合你的直觉。举个例子，即便你非常热爱篮球，满屏的 NBA 视频也会让你感到枯燥和无趣，刷 APP 的时间会变短，用户粘性会降低，甚至会卸载 APP。

本章介绍工业界推荐系统中多样性的标准做法。1.1 节介绍一些基础知识，包括物品相似性的度量和提升多样性的主要思路。1.2 节讲解 MMR 多样性算法，以及它与滑动窗口、规则约束的结合。1.3 节讲解 DPP 多样性算法，这节的数学很多，仅适合熟悉矩阵的读者阅读。1.4 节介绍 MGS 算法，它求解的问题与 DPP 完全相同，但是算法实现更简单。

1.1 背景

本节重点讨论两个问题。第一，相似性的度量。在推荐系统中，如果曝光给用户的物品之间有较高的相似性，则说明多样性差。想要研究和提升多样性，我们首先需要定义和量化物品两两之间的相似性。第二，提升多样性的方法。在推荐系统的链路上，应该在哪些环节上做特殊处理，提升多样性?

1.1.1 度量物品相似性

度量相似的方法有很多种，其中最简单、计算速度最快的方法是使用物品的属性标签，比如一级类目、二级类目、品牌、等等。如果两个物品的属性标签相同，则它们的相似度为 1，否则为 0。举个例子，物品 i 和 j 的（一级类目、二级类目、品牌）分别是（美妆，彩妆，香奈儿）和（美妆，香水，香奈儿），则它们的三种相似度分数为

$$\mathrm{sim}_1(i,j) = 1, \qquad \mathrm{sim}_2(i,j) = 0, \qquad \mathrm{sim}_3(i,j) = 1.$$

对三个分数求加权和即可得到相似度总分，其中的权重需要根据经验设置。值得注意的是，类目、品牌等标签通常是由 NLP 算法根据物品描述推断出的，未必准确。

工业界更主流的方法是基于向量表征计算相似度。设物品 i 和 j 的向量表征分别为 $\boldsymbol{v}_i$ 和 $\boldsymbol{v}_j$，如果 $|\langle \boldsymbol{v}_i, \boldsymbol{v}_j \rangle|$ 越大，则说明两个物品越相似。其中值得研究和深挖的问题是如何学习物品的向量表征。首先介绍一种错误的做法，即使用召回的双塔模型，利用其中的物品塔将物品映射为向量表征。双塔模型基于用户与物品交互学习的物品向量表征，但学到的物品向量表征不适用于重排多样性问题，主要原因有两点。第一，物品的曝光次数有严重的头部效应，绝大部分的物品曝光次数很少，模型学不好它们的向量表征。第二，这样学到的向量不是物品内容的表征，而是物品兴趣点的表征。两个物品封面图、标题相似，但向量表征未必相似。

当前最先进的方法是用 CLIP[5] 预训练方法学习图文内容的向量表征。CLIP 的好处是无需人工标注，可以用小红书的海量数据做预训练，学到的模型无需 fine-tune 即可用于提取特征向量。如图 1.1 所示，用 CV 模型（比如卷积神经网络）提取图片的向量表征 $\boldsymbol{a}_i$，用 NLP 模型（比如 BERT）提取文本的向量表征 $\boldsymbol{b}_i$。在小红书中，同一篇笔记的图片和文字往往是相关的，因此 $(\boldsymbol{a}_i, \boldsymbol{b}_i)$ 组成一对正样本。如果从全体样本中随机抽取笔记 i 和 j，那么 $(\boldsymbol{a}_i, \boldsymbol{b}_j)$ 组成一对负样本。在做训练时，应当最大化 $|\langle \boldsymbol{a}_i, \boldsymbol{b}_i \rangle|$，最小化 $|\langle \boldsymbol{a}_i, \boldsymbol{b}_j \rangle|$。CLIP 正是基于这种思想，用 batch 内负采样的方式预训练 CV 模型和 NLP 模型。在做完推理之后，可以把 $\boldsymbol{a}_i$、$\boldsymbol{b}_i$、$[\boldsymbol{a}_i; \boldsymbol{b}_i]$、或者 $\boldsymbol{a}_i + \boldsymbol{b}_i$ 作为笔记 i 的向量表征，用于计算笔记两两之间的相似度。

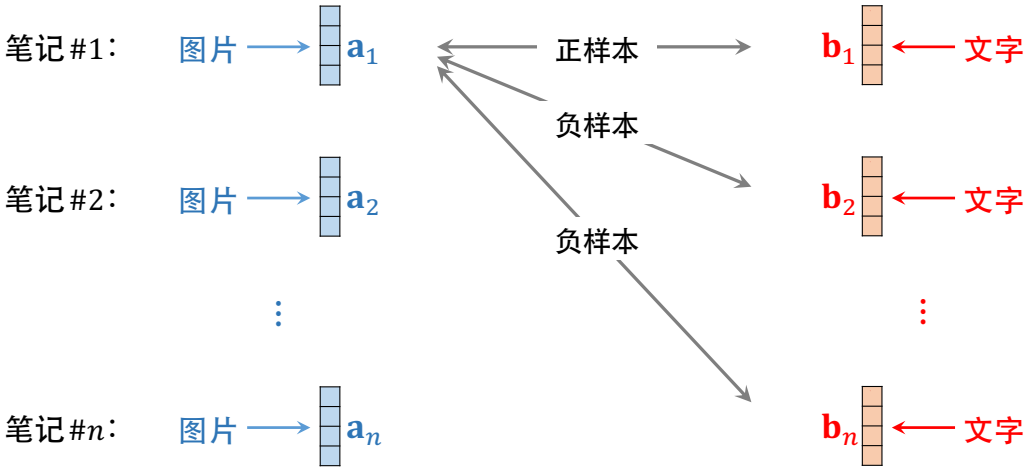


图 1.1: CLIP 方法如图所示，每篇笔记 i 取一张图片和一段文字，分别映射为向量 $\boldsymbol{a}_i$ 和 $\boldsymbol{b}_i$

1.1.2 提升多样性的方法

我们的目标是让曝光给用户的物品具有多样性，即物品之间的相似度较低。为了达到这个目标，工业界普遍的做法是在粗排和精排之后，用后处理的方式提升物品的多样性，如图 1.2 所示。在推荐的链路上，粗排的精排的唯一任务就是准确地做 pointwise 打分，尽量准确地预估点击、交互、时长。pointwise 的意思是把每个物品作为独立的个体，不考虑物品之间的关联。

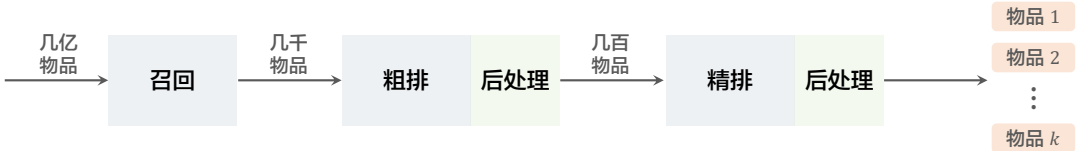


图 1.2: 推荐系统的链路如图所示，精排的后处理通常被称作重排，而粗排的后处理通常不被视作一个独立的环节

具体来说，召回通道返回数千物品，进入粗排环节。对于每个物品 i，粗排模型预估点击率、交互率等指标，并将预估指标融合成一个分数，记作 reward_i，它反映用户对物

品 i 的兴趣。此时有数千物品，每个物品 i 带有一个兴趣分数 reward_i。假如不考虑多样性，那么只需根据 reward_i 对数千物品做排序，选出分数最高的数百物品送入精排。精排亦是如此，给数百物品做 pointwise 打分，假如不考虑多样性，则选出分数最高的数十物品曝光给用户。

如果考虑到多样性，在粗排、精排模型打分之后，该如何从候选物品中选出一个子集呢？设每个候选物品 i 带有一个模型打的分数 reward_i，这个分数是用户对物品的兴趣，可以视作物品本身的价值。做选择时，我们不能把每个物品看做一个独立的个体，而是要考虑到物品之间的相似度，结合 reward_i 与相似度分数，得到物品的总分。后面两节介绍的 MMR 和 DPP 正是基于这样的思想。MMR 与 DPP 的时间复杂度较高，可以用于精排的后处理，但不能用于粗排的后处理。在精排之后从数百物品中选出数十个不需要很大计算量，但是在粗排之后从数千物品中选出数百个则需要很大的计算量。粗排后处理阶段的多样性算法与 MMR、DPP 有相似之处，但此处的算法更简单、粗糙、速度更快。

1.2 Maximal Marginal Relevance（MMR）

设精排返回 n 个物品，它们的精排分数记作 $\text{reward}_1, \cdots, \text{reward}_n$。把物品 i 和 j 的相似度记作 $\text{sim}(i,j)$，数值越大表示两个物品越相似。如图 1.3 所示，用集合 $\mathcal{S}$ 表示已经选中的物品，用集合 $\mathcal{R}$ 表示未选中的物品。对于集合 $\mathcal{R}$ 中的物品 i，定义 marginal relevance（MR）[1]：

$$\text{MR}_i = \underbrace{\theta \cdot \text{reward}_i}_{\text{物品本身价值}} - \underbrace{(1-\theta) \cdot \max_{j \in \mathcal{S}} \text{sim}(i,j)}_{\text{多样性分数}}. \tag{1.1}$$

公式右边第一项 reward_i 的意思是物品 i 的精排分数，比如预估点击率、点赞率等指标的加权和，它的值越大对物品 i 越有利。第二项 $\max_{j \in \mathcal{S}} \text{sim}(i,j)$ 表示候选物品 i 与所有被选中的物品之间的相似度。假如 i 与某个被选中的物品 $j \in \mathcal{S}$ 相似，则 $\max_{j \in \mathcal{S}} \text{sim}(i,j)$ 较大，对物品 i 起抑制作用，不利于物品 i 被选中。公式中的 θ 是介于 0 和 1 之间的超参数，平衡精排分数与多样性。MMR 计算候选物品集合 $\mathcal{R}$ 中每个物品的分数 MR_i，并选出分数最高的物品：

$$\underset{i \in \mathcal{R}}{\text{argmax}} \ \text{MR}_i. \tag{1.2}$$

选中的物品 i 既要有高精排分数 reward_i，也要不能与集合 $\mathcal{S}$ 中的物品相似。

图 1.3: 集合 $\mathcal{S}$ 包含已选中的物品，集合 $\mathcal{R}$ 包含未选中的物品

1.2.1 算法描述与分析

MMR 算法概括如下。初始时，选中的物品集合 $\mathcal{S}$ 为空集，而未选中物品集合 $\mathcal{R}$ 为全集 $[n] = \{1, \cdots, n\}$。首先选取 reward_i 最大的作为第一个物品，将其从集合 $\mathcal{R}$ 移到集合 $\mathcal{S}$。然后用公式 (1.2) 寻找当前最优的物品，将其从集合 $\mathcal{R}$ 移到集合 $\mathcal{S}$。重复这个步骤 $k-1$ 次，则 $\mathcal{S}$ 中包含 k 个物品，它们是最终曝光给用户的物品。

我们定义 n 为候选物品的总数，k 为选出的物品数量，设物品用 d 维向量表征，则算法的时间复杂度为 $\mathcal{O}(nk^2 + nkd)$。具体是这样分析的。当一个物品 i 被选中时，我们计算它与 $\mathcal{R}$ 中所有物品的相似度，并将算出的 $|\mathcal{R}|$ 个相似度存储。算法一共重复这个步骤 $k-1$ 次，总共计算

$$(n-1) + (n-2) + \cdots + (n-k) \;=\; \mathcal{O}(nk)$$

个相似度分数。计算每个相似度分数的时间开销为 $\mathcal{O}(d)$，因此计算相似度的总时间开销为 $\mathcal{O}(nkd)$。每一步还需要计算 $|\mathcal{R}|$ 个分数 MR_i，计算每个 MR_i 的时间开销为 $|\mathcal{S}|$，因此每一步的时间开销为 $\mathcal{O}(|\mathcal{R}| \cdot |\mathcal{S}|) = \mathcal{O}(nk)$。算法运行 $k-1$ 步，因此花在 (1.2) 上的总时间为 $\mathcal{O}(nk^2)$。

1.2.2 滑动窗口

上述的标准 MMR 算法在实践中有个缺点：当集合 $\mathcal{S}$ 较大时，几乎不可能从 $\mathcal{R}$ 中找出一个物品 i，使得 i 与所有选中的物品 $j \in \mathcal{S}$ 都不相似。已经选中的物品越多，即 $\mathcal{S}$ 越大，则越难从 $\mathcal{R}$ 中找出与 $\mathcal{S}$ 不相似的物品。为了解决这个问题，实际实现 MMR 的时候可以滑动窗口（sliding window）来解决上述问题。如图 1.4 所示，设定一个大小固定的滑动窗口 $\mathcal{W}$，其中包含若干最新选中的物品。计算多样性分数的时候，用 $\max_{j \in \mathcal{W}} \text{sim}(i,j)$ 代替 $\max_{j \in \mathcal{S}} \text{sim}(i,j)$，也就是说最新选出的物品 i 只需要跟最近选中的 $|\mathcal{W}|$ 个物品不相似，而不需要跟 $\mathcal{S}$ 中所有物品都不相似。用滑动窗口，则式 (1.1) 变成

$$\underset{i \in \mathcal{R}}{\text{argmax}} \left\{ \theta \cdot \text{reward}_i \;+\; (1-\theta) \cdot \max_{j \in \mathcal{W}} \text{sim}(i,j) \right\}.$$

与式 (1.1) 相比，唯一的区别就是把 $\mathcal{S}$ 替换成 $\mathcal{W}$。

图 1.4: 滑动窗口的示意图

设 $w = |\mathcal{W}|$ 为滑动窗口的大小，它满足 $w \le k$。使用滑动窗口，则总时间复杂度为 $\mathcal{O}(nkd + nkw)$，这说明滑动窗口可以减小计算量。具体是这样分析的。与之前相同，总共需要计算 $\mathcal{O}(nk)$ 个相似度分数，时间复杂度为 $\mathcal{O}(nkd)$。每一步需要计算 $|\mathcal{R}|$ 个分数 MR_i，计算每个 MR_i 所需时间开销为 $\mathcal{O}(|\mathcal{W}|)$。因此每一步需要花费 $\mathcal{O}(nw)$ 时间计算

MR 分数，算法运行 $k-1$ 步，花费在计算 MR 分数上的总时间为 $\mathcal{O}(nkw)$。

1.2.3 规则约束

实际重排的过程中，除了考虑 reward_i 和多样性分数 MR_i，还需要加很多业务规则。这些业务规则大多是为提升用户体验而制定的，它们是硬性的规则，必须要满足。下面举几个例子。[1]

- 小红书推荐的笔记分为图文笔记和视频笔记，最多连续出现 5 篇图文笔记，最多连续出现 5 篇视频笔记。举个例子，如果排名 i 到 $i+4$ 的全都是图文笔记，那么排在 $i+5$ 的必须是视频笔记。这类规则叫做“最多出连续出现 k 篇某种笔记”，这个例子中 $k=5$。
- 推荐的笔记中有运营推广笔记，这些笔记的 reward_i 不是模型真实的打分，而是在模型打分的基础上乘以了大于 1 的系数，以此让运营推广的笔记获得更多流量。为了限制运营推广的笔记曝光过多影响体验，限制条件为每 9 篇最多出 1 篇。也就是说，如果排名 i 的笔记是运营推广的，那么 $i+1$ 到 $i+8$ 都不能是运营推广的笔记。这类规则叫做“每 k 篇笔记最多出 1 篇某种类型的笔记”，这个例子中 $k=9$。
- 排名最高的 t 篇笔记是用户最容易看到的，而且有很大比例的用户不往下翻，因此顶部的笔记质量要有保障。小红书推荐的笔记中有电商笔记，即在笔记图片或视频下方有一个电商卡片。为了限制电商笔记在顶部曝光过多影响体验，限制条件为前 1 篇最多出 0 篇（即第一篇笔记不能是电商笔记），前 4 篇最多出 1 篇。这类规则叫做“前 t 篇笔记最多有 k 篇某种类型的笔记”，两个例子分别设置 $(t=1,k=0)$ 和 $(t=4,k=1)$。

重排需要将 MMR 与规则相结合，在满足规则的前提下最大化 MR_i。MMR 每一轮都求解 $\text{argmax}_{i\in\mathcal{R}}\,\text{MR}_i$，此处的 $\mathcal{R}$ 是未被选中的物品的集合。由于有多种规则约束，每一步都需要先用规则做排除，从 $\mathcal{R}$ 中剔除不符合规则的，得到的子集记作 $\mathcal{R}'$。求解 $\text{argmax}_{i\in\mathcal{R}'}\,\text{MR}_i$ 得到既符合规则，也具有高价值和多样性的物品。

1.3 Determinantal Point Process（DPP）

精排输出 n 个候选物品，已知它们的向量表征 $\boldsymbol{v}_1,\cdots,\boldsymbol{v}_n\subset\mathbb{R}^d$，并且有精排的打分 $\text{reward}_1,\cdots,\text{reward}_n$。我们的目标是从中选出 k 个物品，记作集合 $\mathcal{S}$，它是 $[n]=\{1,\cdots,n\}$ 的子集。与 MMR 类似，DPP 的目标是让集合 $\mathcal{S}$ 中的物品既有很高的价值，也具有多样性。两种方法的区别在于如何衡量多样性，MMR 用物品两两之间的相似度衡量多样性，而 DPP 用行列式衡量整个集合 $\mathcal{S}$ 的多样性。

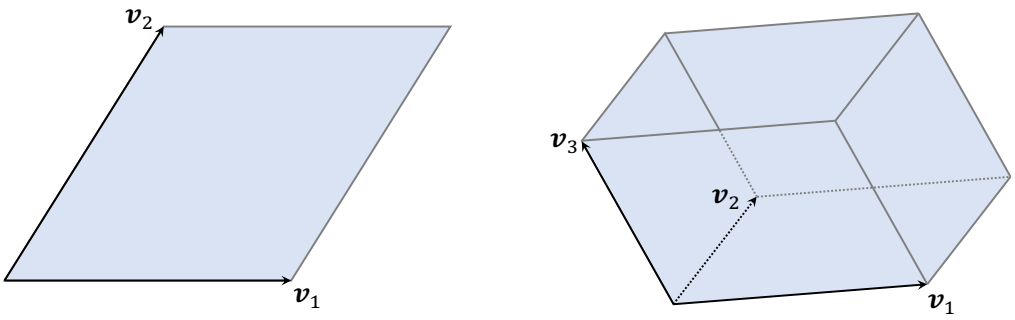
[1]例子中的数字不是小红书业务中的真实数据

1.3.1 超平行体（parallelotope）

给定一组向量 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k \in \mathbb{R}^d$，而且满足 $k \leq d$。那么 $\mathbb{R}^d$ 空间中的 k-维超平行体定义为：

$$\mathcal{P}(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k) = \{\alpha_1 \boldsymbol{v}_1 + \cdots + \alpha_k \boldsymbol{v}_k \mid 0 \leq \alpha_1, \cdots, \alpha_k \leq 1\}.$$

向量 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k \in \mathbb{R}^d$ 被称作超平形体的边，它们唯一确定一个超平形体。超平行体中的点是向量 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k \in \mathbb{R}^d$ 的线性组合，且系数取值为 $\alpha_1, \cdots, \alpha_k \in [0, 1]$。如图 1.5 所示，平行四边形（parallelograms）平行六面体（parallelepiped）分别是 $k = 2$ 和 $k = 3$ 时的超平行体。



(a) 平行四边形　　(b) 平行六面体

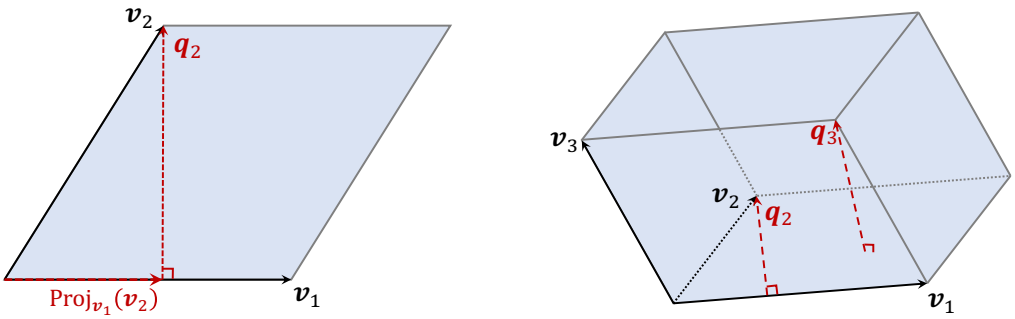
图 1.5: 平行四边形和平行六面体都是特殊的超平行体

计算超平行体的体积，需要对向量 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k$ 做正交化。以图 1.6 中的平行四边形为例，它的面积等于底和高的长度相乘。如果我们以 $\boldsymbol{v}_1$ 为底，那么高就是

$$\boldsymbol{q}_2 = \boldsymbol{v}_2 - \mathrm{Proj}_{\boldsymbol{v}_1}(\boldsymbol{v}_2), \qquad \text{其中} \quad \mathrm{Proj}_{\boldsymbol{v}_1}(\boldsymbol{v}_2) = \frac{\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle}{\|\boldsymbol{v}_1\|_2^2} \boldsymbol{v}_1.$$

上式中的 $\mathrm{Proj}_{\boldsymbol{v}_1}(\boldsymbol{v}_2)$ 表示将 $\boldsymbol{v}_2$ 投影到 $\boldsymbol{v}_1$ 的方向上。不难证明，$\boldsymbol{v}_1$ 与 $\boldsymbol{q}_2$ 正交，即两个向量的内积等于零。平行四边形 $\mathcal{P}$ 的面积等于

$$\mathrm{vol}(\mathcal{P}) = \|\boldsymbol{v}_1\|_2 \cdot \|\boldsymbol{q}_2\|_2.$$



(a) 平行四边形　　(b) 平行六面体

图 1.6: 超平行体的体积示意图

以图 1.6 中的平行六面体为例，它的体积等于底面积乘以高。前面推导过，底面积等于 $\|\boldsymbol{v}_1\|_2 \cdot \|\boldsymbol{q}_2\|_2$。平行六面体的高等于

$$\boldsymbol{q}_3 \;=\; \boldsymbol{v}_3 \;-\; \mathrm{Proj}_{\boldsymbol{v}_1}(\boldsymbol{v}_3) \;-\; \mathrm{Proj}_{\boldsymbol{q}_2}(\boldsymbol{v}_3).$$

不难证明，$\boldsymbol{q}_3$ 与 $\boldsymbol{v}_1$ 正交，而且 $\boldsymbol{q}_3$ 也与 $\boldsymbol{q}_2$ 正交。平行六面体 $\mathcal{P}$ 的体积等于

$$\mathrm{vol}(\mathcal{P}) \;=\; \big\|\boldsymbol{v}_1\big\|_2 \cdot \big\|\boldsymbol{q}_2\big\|_2 \cdot \big\|\boldsymbol{q}_3\big\|_2.$$

上面超平形体的定义要求 $k \leq d$。举个例子，当 $k=2$、$d=3$ 时，3 维空间中的向量 $\boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathbb{R}^3$ 可以确定一个平行四边形，即 3 维空间中的 2 维超平形体。但是当 $k=3$、$d=2$ 时，2 维空间中的向量 $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3 \in \mathbb{R}^2$ 不可能确定一个平行六面体。此外，$\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k \in \mathbb{R}^d$ 线性无关时，超平形体才是有意义的。否则，如果它们线性相关，则它们会落到一个 $k-1$ 维的超平面上，导致它们组成的 k 维超平形体的体积等于 0。

1.3.2 多样性的度量

接下来，我们讨论如何度量向量的多样性。给定 k 个向量 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k \in \mathbb{R}^d$，且它们满足 $\|\boldsymbol{v}_i\|_2 = 1$，$\forall i$。我们做如下假设：如果向量 $\boldsymbol{v}_i$ 和 $\boldsymbol{v}_j$ 接近于平行，即内积 $\boldsymbol{v}_i^\top \boldsymbol{v}_j$ 接近 1 或 -1，则认为物品 i 与 j 相似；如果向量 $\boldsymbol{v}_i$ 和 $\boldsymbol{v}_j$ 接近于正交，即内积 $\boldsymbol{v}_i^\top \boldsymbol{v}_j$ 接近 0，则认为物品 i 与 j 不相似。

基于以上假设，我们可以用超平行体 $\mathcal{P}(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k)$ 的体积衡量衡量 k 个物品的多样性，它的体积介于 0 和 1 之间。当向量 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k$ 两两正交时，物品的多样性最大化，此时的超平行体是一个正方体，它的体积等于 1。假如存在两个向量 $\boldsymbol{v}_i = \pm \boldsymbol{v}_j$，这说明物品的多样性不足，此时的超平行体的体积最小化，等于 0。

超平行体的体积与行列式之间存在某些数学上的等价性。把向量 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k \in \mathbb{R}^d$ 作为矩阵 $\boldsymbol{V}$ 的列，即

$$\boldsymbol{V} \;=\; [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_k] \;\in\; \mathbb{R}^{d \times k}.$$

当 $k=d$ 时，$\boldsymbol{V}$ 是方阵，行列式与体积有这样的关系：

$$\big|\det\big(\boldsymbol{V}\big)\big| \;=\; \mathrm{vol}\Big(\mathcal{P}\big(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k\big)\Big). \tag{1.3}$$

$\boldsymbol{V}^\top \boldsymbol{V}$ 是大小为 $k \times k$ 的对称半正定矩阵，它的行列式非负。当 $k \leq d$ 时，$\boldsymbol{V}^\top \boldsymbol{V}$ 的行列式与体积有这样的关系：

$$\det\big(\boldsymbol{V}^\top \boldsymbol{V}\big) \;=\; \mathrm{vol}\Big(\mathcal{P}\big(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k\big)\Big)^2. \tag{1.4}$$

式 (1.3) 是数学中一个众所周知的结论，它的的证明在很多书中都有，比如[4]。式 (1.4) 并不是一个很显然的结论，因此本书对其做严格的证明。如果不感兴趣，或者对矩阵不熟悉，可以跳过下面的证明。

证明 由于 $k \leq d$，对于任何一个 d 维空间中的 k 维子空间 $\mathcal{H}$，我们可以找到一个 $d \times d$ 的正交矩阵 $\boldsymbol{R}$，对向量 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k \in \mathbb{R}^d$ 做旋转，使得 $\boldsymbol{R}\boldsymbol{v}_1, \cdots, \boldsymbol{R}\boldsymbol{v}_k \in \mathbb{R}^d$ 全部落在子空间 $\mathcal{H}$ 上。设子空间 $\mathcal{H}$ 由前 k 个标准正交基组成，那么 $\mathcal{H}$ 上所有点的后 $d-k$ 个元素

恒等于零：

$$\boldsymbol{R}\boldsymbol{v}_i = \begin{bmatrix} \boldsymbol{u}_i \\ \boldsymbol{0} \end{bmatrix}.$$

上式中的 $\boldsymbol{u}_i$ 是 k 维向量，$\boldsymbol{0}$ 是 $d-k$ 维的全零向量。把 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k \in \mathbb{R}^d$ 作为矩阵 $\boldsymbol{V} \in \mathbb{R}^{d\times k}$ 的列，把 $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_k \in \mathbb{R}^k$ 作为矩阵 $\boldsymbol{U} \in \mathbb{R}^{k\times k}$ 的列，那么有

$$\boldsymbol{R}\boldsymbol{V} = \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{0} \end{bmatrix}. \tag{1.5}$$

很显然，$\{\boldsymbol{u}_i\}$ 组成的超平行体与 $\{[\boldsymbol{u}_i; \boldsymbol{0}]\}$ 组成的超平行体有相同的体积：

$$\mathrm{vol}\Big(\mathcal{P}(\boldsymbol{u}_1, \cdots, \boldsymbol{u}_k)\Big) = \mathrm{vol}\left(\mathcal{P}\left(\begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{0} \end{bmatrix}, \cdots, \begin{bmatrix} \boldsymbol{u}_k \\ \boldsymbol{0} \end{bmatrix}\right)\right).$$

根据 $\boldsymbol{u}_i$ 的定义有 $\boldsymbol{R}\boldsymbol{v}_i = [\boldsymbol{u}_i; \boldsymbol{0}]$，结合上式可得

$$\mathrm{vol}\Big(\mathcal{P}(\boldsymbol{u}_1, \cdots, \boldsymbol{u}_k)\Big) = \mathrm{vol}\Big(\mathcal{P}(\boldsymbol{R}\boldsymbol{v}_1, \cdots, \boldsymbol{R}\boldsymbol{v}_k)\Big). \tag{1.6}$$

$\boldsymbol{R}\boldsymbol{v}_i$ 相当于对 $\boldsymbol{v}_i$ 做旋转，而对超平行体做旋转不影响体积，因此有

$$\mathrm{vol}\Big(\mathcal{P}(\boldsymbol{R}\boldsymbol{v}_1, \cdots, \boldsymbol{R}\boldsymbol{v}_k)\Big) = \mathrm{vol}\Big(\mathcal{P}(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k)\Big). \tag{1.7}$$

结合式 (1.6) 和 (1.7) 可得

$$\mathrm{vol}\Big(\mathcal{P}(\boldsymbol{u}_1, \cdots, \boldsymbol{u}_k)\Big) = \mathrm{vol}\Big(\mathcal{P}(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k)\Big). \tag{1.8}$$

$\boldsymbol{R}$ 是 $d \times d$ 的正交矩阵，由正交矩阵的性质可得 $\boldsymbol{R}^\top\boldsymbol{R} = \boldsymbol{I}_d$（即 $d \times d$ 的单位矩阵）。因此有 $(\boldsymbol{R}\boldsymbol{V})^\top(\boldsymbol{R}\boldsymbol{V}) = \boldsymbol{V}^\top(\boldsymbol{R}^\top\boldsymbol{R})\boldsymbol{V} = \boldsymbol{V}^\top\boldsymbol{V}$。结合式 (1.5) 可得

$$\boldsymbol{V}^\top\boldsymbol{V} = (\boldsymbol{R}\boldsymbol{V})^\top(\boldsymbol{R}\boldsymbol{V}) = \begin{bmatrix} \boldsymbol{U}^\top & \boldsymbol{0} \end{bmatrix}\begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{0} \end{bmatrix} = \boldsymbol{U}^\top\boldsymbol{U}.$$

由于 $\boldsymbol{U}$ 是 $k \times k$ 的方阵，由行列式的性质可得 $\det(\boldsymbol{U}^\top\boldsymbol{U}) = \det(\boldsymbol{U})^2$。结合上式可得

$$\det\left(\boldsymbol{V}^\top\boldsymbol{V}\right) = \det\left(\boldsymbol{U}^\top\boldsymbol{U}\right) = \det\left(\boldsymbol{U}\right)^2. \tag{1.9}$$

由于 $\boldsymbol{U}$ 是方阵，由式 (1.3) 可得

$$\det\left(\boldsymbol{U}\right)^2 = \mathrm{vol}\Big(\mathcal{P}(\boldsymbol{u}_1, \cdots, \boldsymbol{u}_k)\Big)^2. \tag{1.10}$$

结合式 (1.8)、(1.9)、(1.10) 可得

$$\det\left(\boldsymbol{V}^\top\boldsymbol{V}\right) = \det\left(\boldsymbol{U}\right)^2 = \mathrm{vol}\Big(\mathcal{P}(\boldsymbol{u}_1, \cdots, \boldsymbol{u}_k)\Big)^2 = \mathrm{vol}\Big(\mathcal{P}(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k)\Big)^2.$$

由上式可证 (1.4)。 □

1.3.3 k-DPP

前面介绍了超平行体、体积、行列式，接下来我们回到本节的主题——用 DPP 解决重排多样性问题。精排给 n 个物品打分，它们的向量表征为 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n \in \mathbb{R}^d$。我们希望选出 k 个物品，使得它们组成的超平行体 $\mathcal{P}(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k)$ 的体积较大。计算一个 $n \times n$ 的相似度矩阵 $\boldsymbol{A}$，它是一个 $n \times n$ 的对称半正定矩阵，它的第 (i, j) 个元素等于 $a_{ij} = \boldsymbol{v}_i^\top\boldsymbol{v}_j$。设集合 $\mathcal{S}$ 是 $[n]$ 的子集，它的大小为 k。定义 $\boldsymbol{A}_\mathcal{S}$ 是 $\boldsymbol{A}$ 的一个子矩阵，大小为 $k \times k$。如

果 $i, j \in \mathcal{S}$，则 a_{ij} 是 $\boldsymbol{A}_{\mathcal{S}}$ 的一个元素。

我们根据集合 $\mathcal{S}$（$|\mathcal{S}| = k$）从 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n \subset \mathbb{R}^d$ 中选出 k 个向量，作为矩阵 $\boldsymbol{V}_{\mathcal{S}} \in \mathbb{R}^{d \times k}$ 的列。那么 $k \times k$ 的矩阵 $\boldsymbol{A}_{\mathcal{S}}$ 可以写成：

$$\boldsymbol{A}_{\mathcal{S}} \ = \ \boldsymbol{V}_{\mathcal{S}}^{\top} \boldsymbol{V}_{\mathcal{S}}.$$

设 $\mathcal{P}(\boldsymbol{V}_{\mathcal{S}})$ 为选出的 k 个向量张成的超平行体。如果 $k \leq d$，那么由式 (1.4) 可得：

$$\det\left(\boldsymbol{A}_{\mathcal{S}}\right) \ = \ \det\left(\boldsymbol{V}_{\mathcal{S}}^{\top} \boldsymbol{V}_{\mathcal{S}}\right) \ = \ \mathrm{vol}\left(\mathcal{P}\left(\boldsymbol{V}_{\mathcal{S}}\right)\right)^2.$$

对等式两边取对数，可得

$$\log\det\left(\boldsymbol{A}_{\mathcal{S}}\right) \ = \ 2 \cdot \log\mathrm{vol}\left(\mathcal{P}\left(\boldsymbol{V}_{\mathcal{S}}\right)\right). \tag{1.11}$$

前面已经讨论过，可以用超平行体的体积度量集合 $\mathcal{S}$ 中物品的多样性，因此也可以用 $\log\det\left(\boldsymbol{A}_{\mathcal{S}}\right)$ 度量多样性。k-DPP 是一种传统的统计机器学习方法，它的目标函数是选出 k 个物品组成集合 $\mathcal{S}$，使得 $\log\det(\boldsymbol{A}_{\mathcal{S}})$ 最大化。

Hulu 2018 年的论文[2] 将 k-DPP 应用于推荐系统重排，具体为求解这个优化问题：

$$\underset{\mathcal{S}:|\mathcal{S}|=k}{\mathrm{argmax}} \ \theta \cdot \left(\sum_{i \in \mathcal{S}} \mathrm{reward}_i\right) \ + \ (1-\theta) \cdot \log\det(\boldsymbol{A}_{\mathcal{S}}). \tag{1.12}$$

这是个组合优化问题，从 n 个物品中选出 k 个，目前没有高效的方法精确求解这个组合优化问题。Hulu 的论文使用一种贪心算法求解 (1.12)。用 $\mathcal{S}$ 表示已选中的物品，用 $\mathcal{R}$ 表示未选中的物品。算法的每一步求解这样一个问题：

$$\underset{i \in \mathcal{R}}{\mathrm{argmax}} \ \left\{\theta \cdot \mathrm{reward}_i \ + \ (1-\theta) \cdot \log\det\left(\boldsymbol{A}_{\mathcal{S} \cup \{i\}}\right)\right\}. \tag{1.13}$$

暴力求解 (1.13) 是可行的，但是计算量比较大。想要计算行列式 $\det\left(\boldsymbol{A}_{\mathcal{S} \cup \{i\}}\right)$，需要对矩阵做特征分解，代价是 $\mathcal{O}\left(|\mathcal{S}|^3\right)$。对于每一个 $i \in \mathcal{R}$，都需要做特征分解，因此暴力求解 (1.13) 的代价是 $\mathcal{O}\left(|\mathcal{S}|^3 \cdot |\mathcal{R}|\right) = \mathcal{O}(nk^3)$。如果是从 n 个物品中选出 k 个物品，即让集合 $\mathcal{S}$ 的大小从 1 增长到 k，那么需要求解 (1.13) 一共 k 次，总的代价是 $\mathcal{O}(nk^4)$。Hulu 的论文提出了一种巧妙的数值算法求解 (1.13)，原理是已知 $\boldsymbol{A}_{\mathcal{S}}$ 的矩阵分解，可以快速算出 $\boldsymbol{A}_{\mathcal{S} \cup \{i\}}$ 的矩阵分解，从而让行列式的计算变得更快。

1.3.4 数值算法推导

接下来我们推导 Hulu 论文中求解 (1.13) 的数值算法，对数学不感兴趣的读者可以跳过。给定矩阵 $\boldsymbol{A}$，算法总共花费 $\mathcal{O}(nk^2)$ 时间选出 k 个物品。因为 $\boldsymbol{A}_{\mathcal{S}}$ 是对称正定矩阵，所以它存在 Cholesky 分解 $\boldsymbol{A}_{\mathcal{S}} = \boldsymbol{L}\boldsymbol{L}^{\top}$，这里的 $\boldsymbol{L}$ 是大小为 $|\mathcal{S}| \times |\mathcal{S}|$ 的下三角矩阵。下三角矩阵的意思是对角线以上的元素都为零。矩阵 $\boldsymbol{A}_{\mathcal{S} \cup \{i\}}$ 比 $\boldsymbol{A}_{\mathcal{S}}$ 多了一行和一列，记作：

$$\boldsymbol{A}_{\mathcal{S} \cup \{i\}} \ = \ \begin{bmatrix} \boldsymbol{A}_{\mathcal{S}} & \boldsymbol{a}_i \\ \boldsymbol{a}_i^{\top} & a_{ii} \end{bmatrix}. \tag{1.14}$$

上式中的 $\boldsymbol{a}_i$ 的元素是 $\boldsymbol{v}_i^\top \boldsymbol{v}_j$，$\forall j \in \mathcal{S}$，而 $a_{ii} = \boldsymbol{v}_i^\top \boldsymbol{v}_i = 1$。矩阵 $\boldsymbol{A}_{\mathcal{S}\cup\{i\}}$ 的 Cholesky 分解可以写作：

$$\boldsymbol{A}_{\mathcal{S}\cup\{i\}} = \begin{bmatrix} \boldsymbol{L} & \boldsymbol{0} \\ \boldsymbol{c}_i^\top & d_i \end{bmatrix} \begin{bmatrix} \boldsymbol{L} & \boldsymbol{0} \\ \boldsymbol{c}_i^\top & d_i \end{bmatrix}^\top, \tag{1.15}$$

其中 $\boldsymbol{c}_i$ 和 d_i 是未知的。由公式 (1.16) 和 (1.15) 可得：

$$\boldsymbol{A}_{\mathcal{S}\cup\{i\}} = \begin{bmatrix} \boldsymbol{A}_{\mathcal{S}} & \boldsymbol{a}_i \\ \boldsymbol{a}_i^\top & 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{L}\boldsymbol{L}^\top & \boldsymbol{L}\boldsymbol{c}_i \\ \boldsymbol{c}_i^\top \boldsymbol{L}^\top & \boldsymbol{c}_i^\top \boldsymbol{c}_i + d_i^2 \end{bmatrix}.$$

我们得到两个公式：

$$\boldsymbol{a}_i = \boldsymbol{L}\boldsymbol{c}_i \qquad \text{和} \qquad 1 = \boldsymbol{c}_i^T \boldsymbol{c}_i + d_i^2.$$

$\boldsymbol{L}$ 和 $\boldsymbol{a}_i$ 是已知的，$\boldsymbol{L}$ 是上一轮算出的 Cholesky 分解，$\boldsymbol{a}_i$ 包含矩阵 $\boldsymbol{A}$ 的元素。我们需要求出未知的 $\boldsymbol{c}_i$ 和 d_i。由于 $\boldsymbol{L}$ 是下三角矩阵，只需要 $\mathcal{O}(|\mathcal{S}|^2)$ 的浮点数运算即可求出 $\boldsymbol{c}_i = \boldsymbol{L}^{-1}\boldsymbol{a}_i$。然后就可以算出 $d_i^2 = 1 - \boldsymbol{c}_i^\top \boldsymbol{c}_i$。有了 d_i，我们就能快速求出 $\det(\boldsymbol{A}_{\mathcal{S}\cup\{i\}})$。由下三角矩阵和行列式的定义可知：

$$\det\left(\begin{bmatrix} \boldsymbol{L} & \boldsymbol{0} \\ \boldsymbol{c}_i^\top & d_i \end{bmatrix}\right) = \det(\boldsymbol{L}) \times d_i.$$

由于 $\det(\boldsymbol{X}\boldsymbol{Y}) = \det(\boldsymbol{X})\det(\boldsymbol{Y})$，我们得到

$$\det\left(\boldsymbol{A}_{\mathcal{S}\cup\{i\}}\right) = \det\left(\begin{bmatrix} \boldsymbol{L} & \boldsymbol{0} \\ \boldsymbol{c}_i^T & d_i \end{bmatrix} \begin{bmatrix} \boldsymbol{L} & \boldsymbol{0} \\ \boldsymbol{c}_i^\top & d_i \end{bmatrix}^\top\right) = \det(\boldsymbol{L})^2 \times d_i^2.$$

贪心算法的公式 (1.13) 可以等价写作：

$$\operatorname*{argmax}_{i\in\mathcal{R}} \ \theta \cdot \text{reward}_i + (1-\theta) \cdot \Big(\log\det(\boldsymbol{L})^2 + \log d_i^2\Big).$$

由于 $\boldsymbol{L}$ 与 i 无关，上面的公式可以等价写作

$$\operatorname*{argmax}_{i\in\mathcal{R}} \ \theta \cdot \text{reward}_i + (1-\theta) \cdot \log d_i^2. \tag{1.16}$$

这样我们就推导出了求解 k-DPP 的贪心算法：

1. 输入：n 个物品的向量表征 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n \in \mathbb{R}^d$ 和分数 $\text{reward}_1, \cdots, \text{reward}_n$。
2. 计算 $n \times n$ 的相似度矩阵 $\boldsymbol{A}$，它的第 (i,j) 个元素等于 $a_{ij} = \boldsymbol{v}_i^T \boldsymbol{v}_j$。时间复杂度为 $\mathcal{O}(n^2 d)$。
3. 选中 reward 分数最高的物品，记作 i。初始化集合 $\mathcal{S} = \{i\}$ 和 1×1 的矩阵 $\boldsymbol{L} = [1]$。（由于 $a_{ii} = \boldsymbol{v}_i^\top \boldsymbol{v}_i = 1$，此时 $\boldsymbol{A}_{\mathcal{S}} = [a_{ii}] = \boldsymbol{L}\boldsymbol{L}^\top$。）
4. 做循环，从 $t = 1$ 到 $k-1$：
   (a). 对于每一个 $i \in \mathcal{R}$：
      I. 行向量 $[\boldsymbol{a}_i^T, 1]$ 是矩阵 $\boldsymbol{A}_{\mathcal{S}\cup\{i\}}$ 的最后一行。
      II. 求解线性方程组 $\boldsymbol{a}_i = \boldsymbol{L}\boldsymbol{c}_i$，得到 $\boldsymbol{c}_i$。时间复杂度为 $\mathcal{O}(|\mathcal{S}|^2)$。
      III. 计算 $d_i^2 = 1 - \boldsymbol{c}_i^T \boldsymbol{c}_i$。
   (b). 求解 (1.16)：$i^\star = \operatorname{argmax}_{i\in\mathcal{R}} \ \theta \cdot \text{reward}_i + (1-\theta) \cdot \log d_i^2$。
   (c). 更新集合 $\mathcal{S} \leftarrow \mathcal{S} \cup \{i^\star\}$。

(d). 更新下三角矩阵：

$$\boldsymbol{L} \leftarrow \begin{bmatrix} \boldsymbol{L} & \boldsymbol{0} \\ \boldsymbol{c}_{i^\star}^T & d_{i^\star} \end{bmatrix}.$$

5. 返回集合 $\mathcal{S}$，其中包含 k 个物品。

该算法总时间复杂度为 $\mathcal{O}(n^2d+nk^3)$。如果进一步优化线性方程组 $\boldsymbol{a}_i=\boldsymbol{L}\boldsymbol{c}_i$ 的求解，那么总时间复杂度可以降低到 $\mathcal{O}(n^2d+nk^2)$。原理是在第 t 轮循环中，利用第 $t-1$ 轮对 $\boldsymbol{a}_i=\boldsymbol{L}\boldsymbol{c}_i$ 的求解。这里的数学有点复杂，就不展开介绍了。

1.3.5 DPP 多样性算法的扩展

上节介绍的滑动窗口方法适用于 DPP，而且对 DPP 是非常必要的。设集合 $\mathcal{S}$ 为选中的物品的集合，$\mathcal{R}$ 为未选中的物品的集合。DPP 用 $\boldsymbol{A}_{\mathcal{S}}$ 的行列式衡量集合 $\mathcal{S}$ 的多样性，算法的每一步求解这样一个问题：

$$\underset{i\in\mathcal{R}}{\operatorname{argmax}} \left\{\theta\cdot\text{reward}_i + (1-\theta)\cdot\log\det\left(\boldsymbol{A}_{\mathcal{S}\cup\{i\}}\right)\right\}.$$

DPP 存在一个严重的问题，随着集合 $\mathcal{S}$ 逐渐变大，其中必然会包含很多相似的物品，这会导致行列式 $\det(\boldsymbol{A}_{\mathcal{S}})$ 会坍缩到 0，它的对数会接近负无穷。因此，实践中会用滑动窗口方法解决这个问题。方法跟 1.2 节类似，设置一个较小的滑动窗口 $\mathcal{W}$，把上式中的 $\mathcal{S}$ 替换成 $\mathcal{W}$：

$$\underset{i\in\mathcal{R}}{\operatorname{argmax}} \left\{\theta\cdot\text{reward}_i + (1-\theta)\cdot\log\det\left(\boldsymbol{A}_{\mathcal{W}\cup\{i\}}\right)\right\}. \tag{1.17}$$

求解这个问题的数值算法比较复杂，此处就不详细推导了。

上节介绍的规则约束也适用于 DPP，具体用法几乎一样。DPP 的每一步都求解式 (1.17)，从集合 $\mathcal{R}$ 中选出一个物品。如果应用规则约束，则会排除掉 $\mathcal{R}$ 中的部分物品，得到子集 $\mathcal{R}'\subset\mathcal{R}$。把式 (1.17) 中的 $\mathcal{R}$ 替换成 $\mathcal{R}'$，得到

$$\underset{i\in\mathcal{R}'}{\operatorname{argmax}} \left\{\theta\cdot\text{reward}_i + (1-\theta)\cdot\log\det\left(\boldsymbol{A}_{\mathcal{W}\cup\{i\}}\right)\right\}. \tag{1.18}$$

数值算法几乎没有任何区别。由于候选集 $\mathcal{R}$ 缩小成 $\mathcal{R}'$，实际的计算量会有所减小。

1.4 Modified Gram-Schmidt（MGS）

上节介绍了 DPP，它是重排中最常用的多样性算法，而且算法的时间复杂度不高，可以比较快地从几百个物品中选出几十个。DPP 的一个缺点在于数值算法的编程实现比较复杂，如果实现得不好，算法效率不高。小红书论文[3] 提出用 MGS 算法求解 DPP，算法的实现非常简单。

1.4.1 向量正交化

Gram-Schmidt（GS）正交化。 给定一组线性独立（linearly independent）的向量 $\boldsymbol{v}_1,\cdots,\boldsymbol{v}_k\in\mathbb{R}^d$，而且满足 $k\leq d$。GS 输出一组正交基 $\boldsymbol{q}_1,\cdots,\boldsymbol{q}_k\in\mathbb{R}^d$，对于任意

$i \neq j$，满足 $\boldsymbol{q}_i^\top \boldsymbol{q}_j = 0$。GS 具体这样做计算：

$$\begin{aligned}
\boldsymbol{q}_1 &= \boldsymbol{v}_1, \\
\boldsymbol{q}_2 &= \boldsymbol{v}_2 - \mathrm{Proj}_{\boldsymbol{q}_1}(\boldsymbol{v}_2), \\
\boldsymbol{q}_3 &= \boldsymbol{v}_3 - \mathrm{Proj}_{\boldsymbol{q}_1}(\boldsymbol{v}_3) - \mathrm{Proj}_{\boldsymbol{q}_2}(\boldsymbol{v}_3), \\
\boldsymbol{q}_4 &= \boldsymbol{v}_4 - \mathrm{Proj}_{\boldsymbol{q}_1}(\boldsymbol{v}_4) - \mathrm{Proj}_{\boldsymbol{q}_2}(\boldsymbol{v}_4) - \mathrm{Proj}_{\boldsymbol{q}_3}(\boldsymbol{v}_4), \\
&\vdots \\
\boldsymbol{q}_k &= \boldsymbol{v}_k - \sum_{i=1}^{k-1} \mathrm{Proj}_{\boldsymbol{q}_i}(\boldsymbol{v}_k).
\end{aligned}$$

上式中的投影符号定义为

$$\mathrm{Proj}_{\boldsymbol{u}}(\boldsymbol{v}) = \frac{\langle \boldsymbol{u}, \boldsymbol{v} \rangle}{\|\boldsymbol{u}\|_2^2} \boldsymbol{u}.$$

不难发现，超平行体

$$\mathcal{P}(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k) = \{\alpha_1 \boldsymbol{v}_1 + \cdots + \alpha_k \boldsymbol{v}_k \mid 0 \leq \alpha_1, \cdots, \alpha_k \leq 1\}$$

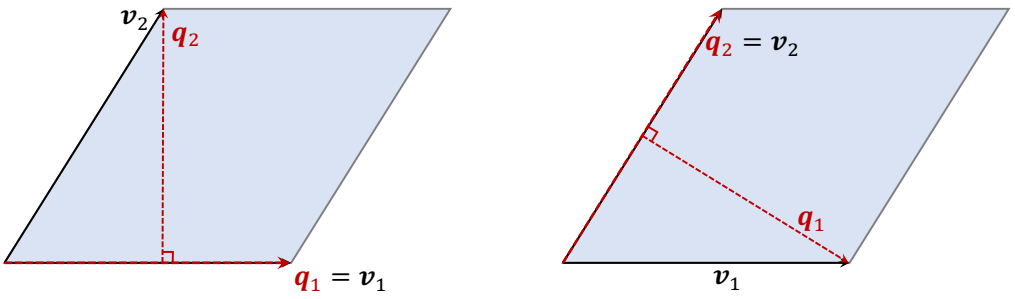
的体积等于

$$\mathrm{vol}\,(\mathcal{P}) = \|\boldsymbol{q}_1\|_2 \cdot \|\boldsymbol{q}_2\|_2 \cdots \|\boldsymbol{q}_k\|_2. \tag{1.19}$$

设 $\boldsymbol{v}_1, \cdots \boldsymbol{v}_k$ 都是单位向量，即二范数等于 1。当 $\boldsymbol{v}_1, \cdots \boldsymbol{v}_k$ 两两正交时，超平行体 $\mathcal{P}$ 是一个正方体，此时它的体积最大化，$\mathrm{vol}(\mathcal{P})$ 等于 1。当 $\boldsymbol{v}_1, \cdots \boldsymbol{v}_k$ 接近线性相关，即存在 i 和 $\beta_1, \cdots, \beta_k$ 使得

$$\boldsymbol{v}_i \approx \sum_{j \neq i} \beta_j \boldsymbol{v}_j,$$

此时超平行体 $\mathcal{P}$ 的体积接近 0。GS 算法得到的正交基不唯一。如果调整 GS 算法中 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k$ 的顺序，那么输出的 $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_k$ 会截然不同，但这并不会影响体积公式 (1.19) 的正确性。如图 1.7 所示，调整 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k$ 的顺序，只是会影响谁是底、谁是高，而算出的体积仍然是相同的。



(a) 以 $\boldsymbol{q}_1$ 为底，以 $\boldsymbol{q}_2$ 为高　　(b) 以 $\boldsymbol{q}_2$ 为底，以 $\boldsymbol{q}_1$ 为高

图 1.7: 如果调整 GS 算法中 $\boldsymbol{v}_1$ 和 $\boldsymbol{v}_2$ 的顺序，那么得到的 $\boldsymbol{q}_1$ 和 $\boldsymbol{q}_2$ 完全不同，但计算面积的公式 $\mathrm{vol}(\mathcal{P}) = \|\boldsymbol{q}_1\|_2 \cdot \|\boldsymbol{q}_2\|_2$ 仍然正确

Modified Gram-Schmidt（MGS）正交化。 GS 存在数值稳定性问题，即在算法运行

的过程中，误差会累加，导致 GS 输出的 $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_k$ 并非严格的正交基。实践中更常用 MGS，它与 GS 在数学上完全等价，但是数值更稳定。

在上文的讨论中，算法的输入是一组线性独立的向量 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_k \in \mathbb{R}^d$，且有 $k \leq d$。而在重排的场景下，算法的输入是向量 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n \in \mathbb{R}^d$，我们需要从 n 个物品中选出 k 个，且有 $n \gg d \geq k$。MGS 首先做初始化：

$$\boldsymbol{q}_1 \longleftarrow \boldsymbol{v}_1, \quad \boldsymbol{q}_2 \longleftarrow \boldsymbol{v}_2, \quad \cdots, \quad \boldsymbol{q}_n \longleftarrow \boldsymbol{v}_n.$$

MGS 算法运行 $k-1$ 步：

$$\begin{aligned}
&\text{第 2 步：} && \boldsymbol{q}_2 \longleftarrow \boldsymbol{q}_2 - \mathrm{Proj}_{\boldsymbol{q}_1}(\boldsymbol{q}_2), \quad \cdots, \quad \boldsymbol{q}_n \longleftarrow \boldsymbol{q}_n - \mathrm{Proj}_{\boldsymbol{q}_1}(\boldsymbol{q}_n);\\
&\text{第 3 步：} && \boldsymbol{q}_3 \longleftarrow \boldsymbol{q}_3 - \mathrm{Proj}_{\boldsymbol{q}_2}(\boldsymbol{q}_3), \quad \cdots, \quad \boldsymbol{q}_n \longleftarrow \boldsymbol{q}_n - \mathrm{Proj}_{\boldsymbol{q}_2}(\boldsymbol{q}_n);\\
&\text{第 4 步：} && \boldsymbol{q}_4 \longleftarrow \boldsymbol{q}_4 - \mathrm{Proj}_{\boldsymbol{q}_3}(\boldsymbol{q}_4), \quad \cdots, \quad \boldsymbol{q}_n \longleftarrow \boldsymbol{q}_n - \mathrm{Proj}_{\boldsymbol{q}_3}(\boldsymbol{q}_n);\\
&\quad\vdots\\
&\text{第 } k \text{ 步：} && \boldsymbol{q}_k \longleftarrow \boldsymbol{q}_k - \mathrm{Proj}_{\boldsymbol{q}_{k-1}}(\boldsymbol{q}_k), \quad \cdots, \quad \boldsymbol{q}_n \longleftarrow \boldsymbol{q}_n - \mathrm{Proj}_{\boldsymbol{q}_{k-1}}(\boldsymbol{q}_n);
\end{aligned}$$

对于任意的 $t \leq k$，在第 t 步之后，向量 $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_t$ 是一组正交基。此外，对于任意 $i > t$ 和 $j \leq t$，有 $\boldsymbol{q}_i^\top \boldsymbol{q}_j = 0$。如图 1.8 中例子所示，在 MGS 算法运行 2 步之后，算出的 $\boldsymbol{q}_3$ 和 $\boldsymbol{q}_4$ 均正交于 $\boldsymbol{v}_1$ 和 $\boldsymbol{v}_2$ 张成的平面。

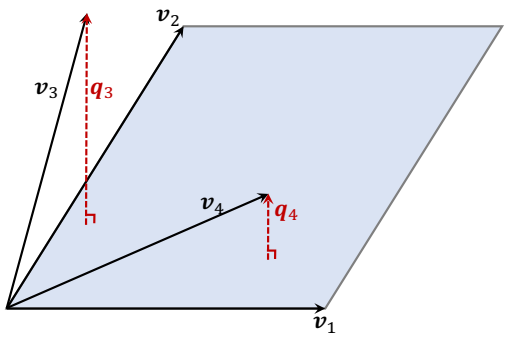


图 1.8: 在 MGS 运行 2 步之后，算出的 $\boldsymbol{q}_3$ 和 $\boldsymbol{q}_4$ 正交于 $\boldsymbol{v}_1$ 和 $\boldsymbol{v}_2$ 张成的平面

1.4.2 MGS 求解 DPP

跟上一节的设定相同，精排输出 n 个物品，它们的向量表征为 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n \subset \mathbb{R}^d$，且它们满足 $\|\boldsymbol{v}_1\|_2 = \cdots = \|\boldsymbol{v}_n\|_2 = 1$。用集合 $\mathcal{S}$ 表示已经选中的物品，用集合 $\mathcal{R}$ 表示未选中的物品。为了记号方便，设 $\mathrm{vol}(\mathcal{S})$ 为集合 $\mathcal{S}$ 中物品向量组成的超平行体的体积。由于体积与行列式的等价性，DPP 多样性算法可以等价写作：

$$\underset{i \in \mathcal{R}}{\mathrm{argmax}} \left\{ \theta \cdot \mathrm{reward}_i + (1-\theta) \cdot \log \mathrm{vol}\left(\mathcal{S} \cup \{i\}\right) \right\}. \tag{1.20}$$

下面我们利用 MGS 多样性算法求解式 (1.20)。

为了符号方便，我们假设在 MGS 的第 t 步结束后，选中的物品为 $\mathcal{S} = \{1, \cdots, t\}$，未选中的物品为 $\mathcal{R} = \{t+1, \cdots, n\}$。MGS 输出的向量为 $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_n$。MGS 算法保证：

- 向量 $\boldsymbol{q}_1, \cdots, \boldsymbol{q}_t$ 是一组正交基，即它们两两正交。
- 对于任意的 $i > t$ 和 $j \leq t$，有 $\boldsymbol{q}_i^\top \boldsymbol{q}_j = 0$。

因此，根据体积公式可得：

$$\mathrm{vol}\left(\mathcal{S}\right) \;=\; \|\boldsymbol{q}_1\|_2 \cdot \|\boldsymbol{q}_2\|_2 \cdots \|\boldsymbol{q}_t\|_2, \qquad \mathrm{vol}\left(\mathcal{S} \cup \{i\}\right) \;=\; \mathrm{vol}\left(\mathcal{S}\right) \cdot \|\boldsymbol{q}_i\|_2.$$

由上可得：

$$\log \mathrm{vol}\left(\mathcal{S} \cup \{i\}\right) \;=\; \log \mathrm{vol}\left(\mathcal{S}\right) \;+\; \log\left(\|\boldsymbol{q}_i\|_2\right).$$

由于 $\log \mathrm{vol}(\mathcal{S})$ 与物品 i 无关，式 (1.20) 可以等价写作：

$$\underset{i \in \mathcal{R}}{\mathrm{argmax}} \; \Big\{ \theta \cdot \mathrm{reward}_i \;+\; (1-\theta) \cdot \log\left(\|\boldsymbol{q}_i\|_2\right) \Big\}. \tag{1.21}$$

MGS 多样性算法概括如下：

1. 初始时，选择 reward 最大的物品加入集合 $\mathcal{S}$。为了记号方便，我们假设选中了 1 号物品，那么此时 $\mathcal{S} = \{1\}$，$\mathcal{R} = \{2, \cdots, n\}$。
2. 做循环，从 $t = 2$ 到 k：
   (a). 做正交化，时间复杂度为 $\mathcal{O}(|\mathcal{R}| \cdot d)$。

   $$\boldsymbol{q}_t \longleftarrow \boldsymbol{q}_t - \mathrm{Proj}_{\boldsymbol{q}_{t-1}}(\boldsymbol{q}_t), \quad \cdots, \quad \boldsymbol{q}_n \longleftarrow \boldsymbol{q}_n - \mathrm{Proj}_{\boldsymbol{q}_{t-1}}(\boldsymbol{q}_n).$$

   (b). 求解 (1.21)，从 $\mathcal{R}$ 中选出物品 i。交换物品 i 与 t 的位置，然后把物品 t 从集合 $\mathcal{R}$ 移到 $\mathcal{S}$。此时 $\mathcal{S} = \{1, \cdots, t\}$，$\mathcal{R} = \{t+1, \cdots, n\}$。

算法循环 $k-1$ 步，每步的时间复杂度为 $\mathcal{O}(|\mathcal{R}| \cdot d) = \mathcal{O}(nd)$，因此算法总的时间复杂度为 $\mathcal{O}(ndk)$。

参考文献

[1] J. Carbonell, J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. International ACM SIGIR Conference on Research and Development in Information Retrieval. 1998 335–336

[2] L. Chen, G. Zhang, E. Zhou. Fast greedy map inference for determinantal point process to improve recommendation diversity. Advances in Neural Information Processing Systems (NIPS), 2018. 31

[3] Y. Huang, W. Wang, L. Zhang, R. Xu. Sliding spectrum decomposition for diversified recommendation. the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD). 2021 3041–3049

[4] D. Margalit, J. Rabinoff, L. Rolen. Interactive linear algebra. Georgia Institute of Technology, 2017

[5] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. International Conference on Machine Learning (ICML). 2021 8748–8763