1. 简介

1.1 技术架构

$$f(\text{用户信息，资源信息，上下文信息，用户行为序列}) = \text{推荐结果}$$

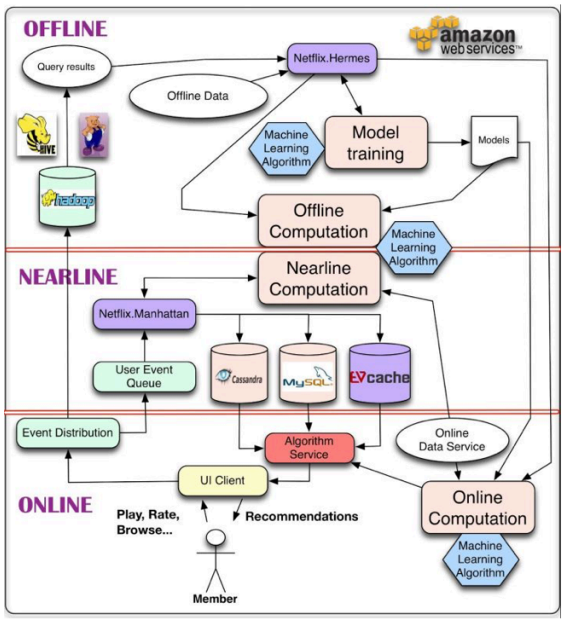
1.2 搜广推

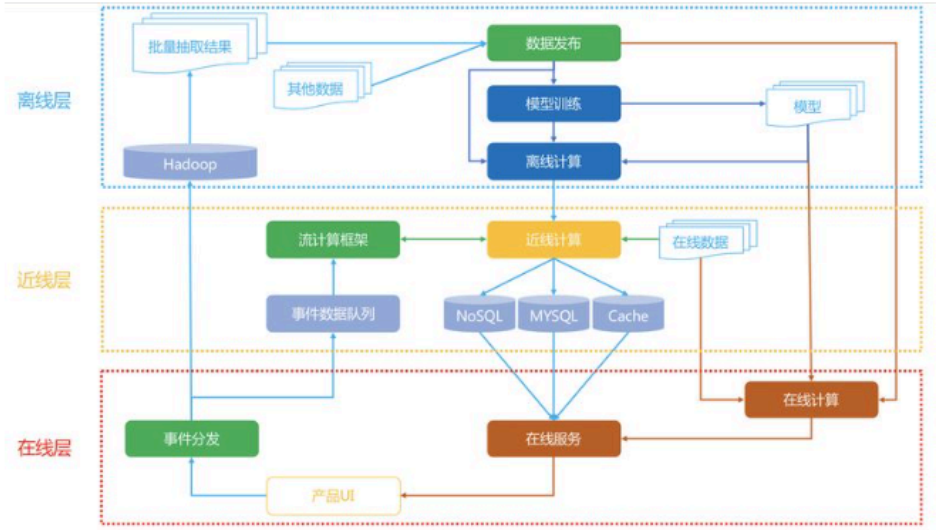
- 搜索：围绕着搜索词的信息高效获取问题
- 广告：直接增加公司收入
- 推荐：提高用户留存和活跃度
  具体区别-知乎-王喆 ↗

1.3 架构

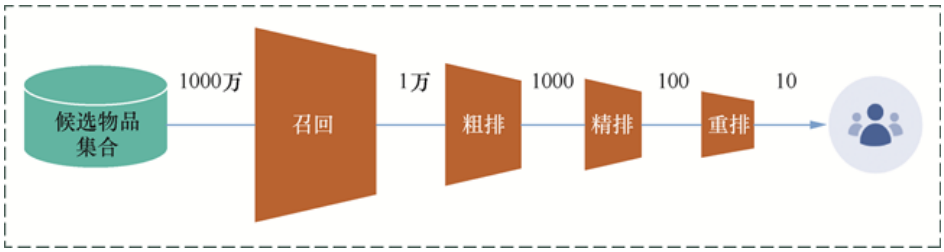
1.3.1 系统架构

1. **离线层**：不用实时数据，不提供实时响应；(大量)
   可以每天更新一次。
2. **近线层**：使用实时数据，不保证实时响应；(几分钟)
   可以在用户访问时更新。
3. **在线层**：使用实时数据，保证实时在线服务。(几十毫秒)
   在用户访问时实时响应，如开屏时的推荐。





1.3.2 算法架构



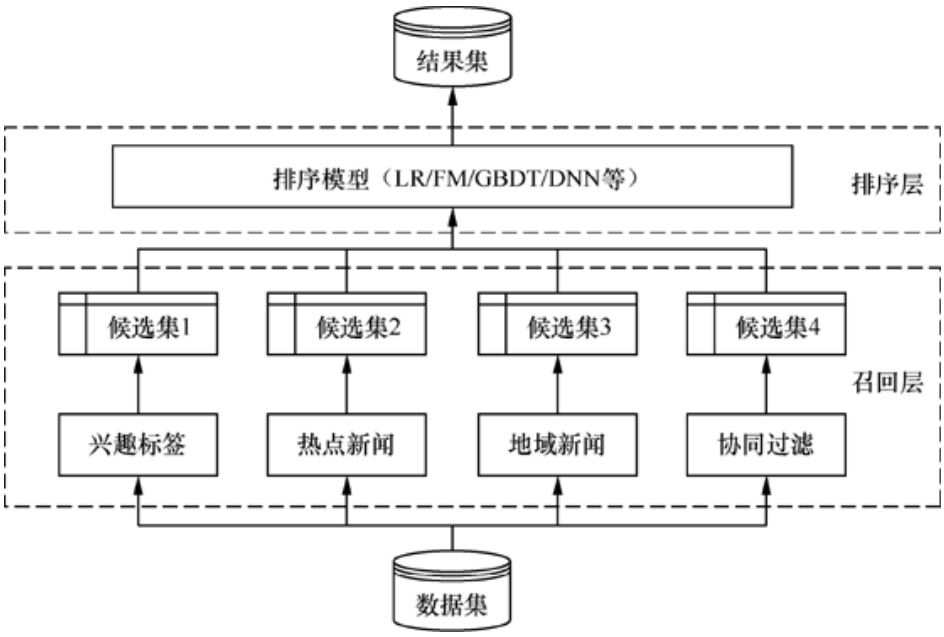
- **召回**：不需要十分准确，但不可遗漏。快速稳定，兴趣多元、内容多样。
  - 非个性化召回、个性化召回
    - 个性化召回：content-based、behavior-based、feature-based
- **粗排**：兼顾精准性和低延迟。
- **精排**：精准性优先，目标ctr、cvr(点击率、转化率)，指标AUC。
  - 样本、特征、模型。
- **重排**：多样性优先，目标Point Wise、Pair Wise、List Wise(点对、对对、列表)，指标NDCG。
  - 基于运营策略、多样性、context上下文。

▶ **点击展开具体算法**

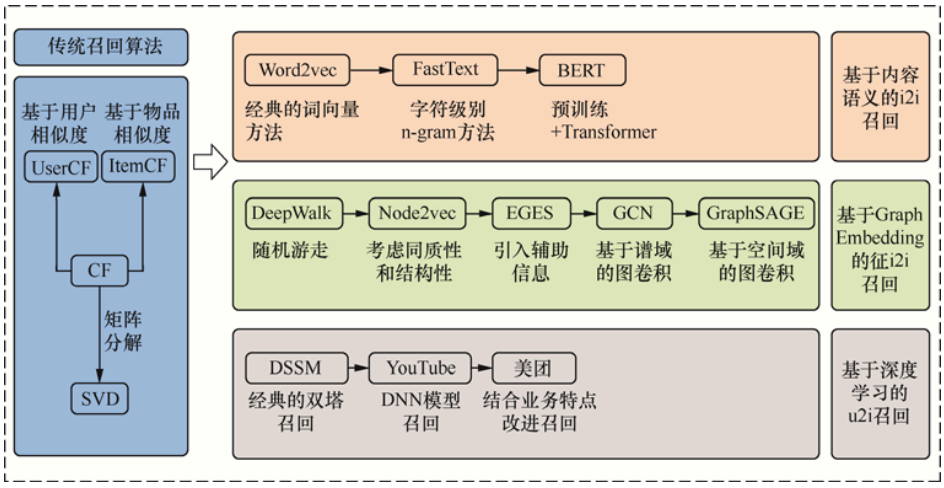
1.4 参考资料

电子书-FunRec ↗
纸质书-推荐系统技术原理与实践 ↗

2. 召回



多路召回



召回技术的演进图谱

2.1 基于协同过滤的召回

2.1.1 协同过滤(CF)

协同过滤(Collaborative Filtering)：基于用户行为数据，找到相似用户或物品， 重点是计算用户/物品之间的相似度。

- **基于用户的协同过滤**(UserCF)：找到相似用户，根据相似用户的行为推荐物品。
- **基于物品的协同过滤**(ItemCF)：找到相似物品，根据该用户的历史行为推荐物品。

2.1.2 相似度计算

记用户为u, v，物品为i, j，用户集合为U。

1. Jaccard相似度

- 反映了两个交互物品交集的数量占这两个用户交互物品并集的数量的比例，适用于隐式反馈的二值数据。
  - **隐式反馈数据**是指从用户行为中隐含地推断出的偏好数据，而不是用户明确表达的偏好或评分。这种类型的数据通常来源于用户在系统中的交互行为，例如点击、浏览、购买等。这些行为可以隐含地表示用户对这些物品有某种程度的兴趣或偏好，但具体的兴趣程度并不明确。隐式反馈数据只记录用户是否与某个物品有过交互(0,1)，而不关心交互的具体程度。

$$sim_{uv} = \frac{|N(u) \cap N(v)|}{|N(u)| \cup |N(v)|}$$

- 其中，$N(u)$,$N(v)$ 分别表示用户u和用户v交互物品的集合。
- 由于杰卡德相似系数一般无法反映具体用户的评分喜好信息，所以常用来评估用户**是否**会对某物品进行打分， 而不是预估用户会对某物品打多少分。
- `from sklearn.metrics.pairwise import jaccard_similarity_score`

2. 余弦相似度

- 衡量了两个向量的夹角，适用于度量向量之间的**相似度**。
- 用户相似度

$$sim_{uv} = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)| \cdot |N(v)|}}$$

- 物品相似度

$$sim_{ij} = \frac{|I(i) \cap I(j)|}{\sqrt{|I(i)| \cdot |I(j)|}}$$

- 其中，$N(u)$,$N(v)$ 分别表示用户u和用户v交互物品的集合，$I(i)$,$I(j)$ 分别表示物品i和物品j被用户交互的集合。
- 设矩阵A为用户-物品交互矩阵，$A_{ij}=0,1$表示用户i对物品j是否交互，$A^{(i)}$表示用户i的评分向量，A_j表示物品j的评分向量，用户之间的相似度计算公式为：$sim_{uv}=\frac{A^{(u)}\cdot A^{(v)}}{\|A^{(u)}\|\cdot\|A^{(v)}\|}=\frac{u\cdot v}{|u|\cdot|v|}=cos(u,v)$。为了节省内存，$A$常使用字典进行存储。
- `from sklearn.metrics.pairwise import cosine_similarity`

3. Pearson相似度

- 衡量了两个向量之间的线性相关性，度量的是两个变量的**变化趋势**是否一致，两个随机变量是不是同增同减。因为先对向量进行中心化处理，所以可以消除用户评分的绝对值差异，适用于评分数据，不适合布尔值向量。
- 用户之间的余弦相似度计算：$sim_{uv}=\frac{r_u\cdot r_v}{|r_u|\cdot|r_v|}=\frac{\sum_i r_{ui}*r_{vi}}{\sqrt{\sum_i r_{ui}^2}\sqrt{\sum_i r_{vi}^2}}$
- 皮尔逊相关系数与余弦相似度的计算公式非常类似：

$$sim(u,v)=\frac{\sum_i(r_{ui}-\bar{r}_u)(r_{vi}-\bar{r}_v)}{\sqrt{\sum_i(r_{ui}-\bar{r}_u)^2}\sqrt{\sum_i(r_{vi}-\bar{r}_v)^2}}\in[-1,1]$$

- 其中，r_{ui},r_{vi}分别表示用户u和用户v对物品i是否有交互(或具体评分值)；$\bar{r}_u,\bar{r}_v$分别表示用户u和用户v交互的所有物品交互数量(或评分的平均值)；
- `from scipy.stats import pearsonr , np.corrcoef()`

4. Tanimoto

$$sim_{uv}=\frac{r_u\cdot r_v}{\|r_u\|^2+\|r_v\|^2-r_u\cdot r_v}$$

2.1.3 基于用户的协同过滤($UserCF$)

1. 计算方法

根据已有的用户向量计算用户y与其他用户X的相似度，得到与用户y最相似的n个用户。根据这 n个用户对物品k的评分情况和与y的相似程度猜测出y对k的评分。如果评分比较高的话， 就把k推荐给y， 否则不推荐。

用户	物品I_1	物品I_2	物品I_3	…	物品k
用户y	1	0	1	...	?
用户X_1	0	1	1	...	0
用户X_2	1	0	0	...	1

- **计算相似度**：计算用户y与其他用户X的相似度。可以采用余弦相似度、皮尔逊相似度等，找到与用户y最相似的n个用户。
- **推荐物品**：根据这n个用户对物品k的评分情况和与y的相似程度猜测出y对k的评分。公式如下：

$$r_{y,k} = \frac{\sum_{X_i \in N(y)} sim(y, X_i) \cdot r_{X_i,k}}{\sum_{X_i \in N(y)} |sim(y, X_i)|}$$

- 式中，$r_{y,k}$表示用户y对物品k的评分，$N(y)$表示与用户y最相似的n个用户，$sim(y, X_i)$表示用户y与用户X_i的相似度，$r_{X_i,k}$表示用户X_i对物品k的评分。
- 考虑到用户的评分主观性强，存在偏置(评分习惯)，有的用户喜欢打高分， 有的用户喜欢打低分的情况。因此公式变为：

$$r_{y,k} = \bar{r}_y + \frac{\sum_{X_i \in N(y)} sim(y, X_i) \cdot (r_{X_i,k} - \bar{r}_{X_i})}{\sum_{X_i \in N(y)} |sim(y, X_i)|}$$

- 式中，$\bar{r}_y$表示用户y的平均评分，$\bar{r}_{X_i}$表示用户X_i的平均评分。因此消除了用户的评分偏置。

- **推荐结果**：实际计算时不止一个物品k，而是所有用户未交互过的物品，所以就根据评分排序，取TopN作为推荐结果。

2. 代码实现

▶ **点击展开查看UserCF完整代码**

3. 优缺点

- **优点**：
  - 简单、易于实现，不需要物品的内容信息，只需要用户的行为数据。
- **缺点**：
  - 商品量大，不同用户之间买的物品重叠性较低，导致相似度计算困难。尤其不适合冷启动、低频购买次数的问题。
  - 用户量大，用户相似度矩阵的存储开销非常大。

4. 算法评估

UserCF和ItemCF结果评估是一致的。

- **召回率**：召回率越高，说明推荐系统越能将用户喜欢的物品推荐给用户。

$$Recall = \frac{\text{推荐成功的物品数}}{\text{用户喜欢的物品数}} = \frac{\text{推荐的}topN \cap \text{用户在测试集上喜欢的物品集合}}{\text{用户在测试集上喜欢的物品集合}}$$

- **准确率**：准确率越高，说明推荐系统推荐的物品越能让用户喜欢。

$$Precision = \frac{\text{推荐成功的物品数}}{\text{推荐出来的物品数}} = \frac{\text{推荐的}topN \cap \text{用户在测试集上喜欢的物品集合}}{\text{推荐的}topN}$$

- **覆盖率**：覆盖率反映了推荐算法发掘长尾的能力。覆盖率越高，说明推荐系统能够将长尾物品推荐给用户。

$$Coverage = \frac{\bigcup_{u \in U} \text{给用户}u\text{推荐的}topN}{\text{总物品数}} = \frac{set(\text{推荐系统给所有用户推荐的商品})}{\text{总物品数}}$$

- **多样性**：多样性是指推荐系统推荐出来的物品之间的差异性。多样性越高，说明推荐系统推荐的物品之间的差异性越大。

$$Diversity = \frac{1}{|U|}\sum_{u \in U} diversity(u)$$

  - 其中，$diversity(u)$表示用户u推荐列表中物品之间的差异性：$diversity(u) = \frac{2}{|I_u|(|I_u|-1)}\sum_{i,j \in I_u, i \neq j} \text{sim}(i,j)$，其中，$I_u$ 是用户 u 的推荐列表，$|I_u|$ 是推荐列表中物品的数量，$\text{sim}(i,j)$ 是物品 i 和物品 j 之间的相似度。
- **新颖度**：用推荐列表中物品的平均流行度度量推荐结果的新颖度。如果推荐出的物品都很热门，说明推荐的新颖度较低。由于物品的流行度分布呈长尾分布，所以为了流行度的平均值更加稳定，在计算平均流行度时对每个物品的流行度取对数。

$$Novelty = \frac{1}{|I_u|}\sum_{i \in I_u} \log(p_i)$$

  - 其中，p_i为第i个物品的流行度(比如点击次数、购买次数等)。Novelty还有其他的度量方法，如$Novelty(u) = \frac{\sum_{i \in I_u}(1-Knows(u,i))}{N}$，其中Knows(u,i)是二值函数，当用户$u$了解物品$i$时取1。

2.1.4 基于物品的协同过滤($ItemCF$)

1. 计算方法

ItemCF算法并不利用物品的内容属性计算物品之间的相似度，而是通过分析用户的行为记录来计算物品之间的相似度。该算法认为，物品i和物品j具有很大的相似度是因为喜欢i的用户也可能喜欢j。

用户	物品I_1	物品I_2	物品I_3	…	物品k
用户y	1	0	1	...	?
用户X_1	0	1	1	...	0
用户X_2	1	0	0	...	1

- **计算相似度**：计算物品k与其他物品I之间的相似度，找出n个物品。
- **推荐物品**：同样考虑物品之间的偏置：

$$P_{y,k} = \bar{r}_k + \frac{\sum_{I_i \in N(y)} sim(k, I_i) \cdot (r_{y,I_i} - \bar{r}_{I_i})}{\sum_{I_i \in N(y)} |sim(k, I_i)|}$$

- **推荐结果**：根据评分排序，取TopN作为推荐结果。

2. 代码实现

▶ **点击展开查看ItemCF完整代码**

3. 优缺点

- **优点**:
  - 适用于物品数量少，用户数量多的场景。
- **缺点**:
  - 物品数量大，计算物品之间的相似度开销大。
  - 物品的流行度分布不均匀，导致推荐出的物品都是热门物品，缺乏多样性。

2.1.5 *Swing*算法

1. 计算方法

若用户u和用户v之间除了购买过i外，还购买过商品j，则认为两件商品是具有某种程度上的相似的(准确说是“**相关度**”或者“**互补性**”)。
为了衡量i,j的相似性，比较同时购买了物品i,j的用户u,v，如果这两个用户共同购买的物品越少，即这两个用户原始兴趣不相似，但仍同时购买了两个相同的物品i,j，则物品i,j的相似性越高。

$$sim(i,j)=\sum_{u\in U_i\cap U_j}\sum_{v\in U_i\cap U_j}\frac{1}{\sqrt{|I_u|}\sqrt{|I_v|}}\cdot\frac{1}{\alpha+|I_u\cap I_v|},u\neq v$$

其中U_i 是点击过商品i的用户集合，I_u 是用户u点击过的商品集合，α是平滑系数,$\frac{1}{\sqrt{|I_u|}}$, $\frac{1}{\sqrt{|I_v|}}$是用户权重参数，来降低活跃用户的影响。

2. 代码实现

▶ **点击展开查看Swing完整代码**

2.1.6 *surprise*算法

1. 类别层面

$$\theta_{i,j}=p(c_{i,j}|c_j)=\frac{N(c_{i,j})}{N(c_j)}$$

- 其中，$N(c_{i,j})$为在购买过i之后购买j类的数量，$N(c_j)$为购买j类的数量。类似kmeans的inertia要看拐点，surprise的inertia要看曲线的拐点来选择相关类的个数。

2. 商品层面

$$s_1(i,j)=\frac{\sum_{u\in U_i\cap U_j}1/(1+|t_{ui}-t_{uj}|)}{\|U_i\|\times\|U_j\|}$$

- 其中，j属于i的相关类，且j的购买时间晚于i。要注意购买顺序(前对后有影响)，购买时间(间隔越短越说明互补)

3. 聚类层面

$$s_2(i,j) = s_1(L(i), L(j))$$

- 其中$L(i)$表示商品i所属的类别。
- 如何聚类？ 传统的聚类算法（k-means、DBSCAN）在数十亿产品规模下的淘宝场景中不可行，所以作者采用了标签传播算法。
- 在哪里标签传播？ Item-item图，其中由Swing计算的排名靠前item为邻居，边的权重就是Swing分数。
- 表现如何？ 快速而有效，15分钟即可对数十亿个项目进行聚类。

4. 线性组合

$$s(i,j) = \omega * s_1(i,j) + (1-\omega) * s_2(i,j)$$

- 其中，$\omega = 0.8$是作者设置的权重超参数。

Surprise算法通过利用类别信息和标签传播技术解决了用户共同购买图上的稀疏性问题。

2.1.7 矩阵分解$MF(matrix\ factorization)$

1. 预测打分

假设n个用户，M部电影有D个特征。我们预测

$$R_{n\times M} = P_{n\times D} \times Q_{D\times M}$$

- 式中R(Rating)是预测n个用户对M部电影的**评分**，P(Preference)是**用户喜好**矩阵，Q(Quality)是**电影特征**矩阵。

例如：
P矩阵：

	特征D_1	**特征D_2**	**特征D_3**
用户p_1	0.1	0.9	0.6
用户p_2	0.8	0.5	0.4

Q矩阵：

	电影Q_1	**电影Q_2**	**电影Q_3**	**电影Q_4**
特征D_1	0.7	0.2	0.3	0.4

	电影Q_1	电影Q_2	电影Q_3	电影Q_4
特征D_2	0.1	0.6	0.9	0.2
特征D_3	0.5	0.8	0.4	0.1

因此：

$$R = \begin{bmatrix} 0.1 & 0.9 & 0.6 \\ 0.8 & 0.5 & 0.4 \end{bmatrix} \times \begin{bmatrix} 0.7 & 0.2 & 0.3 & 0.4 \\ 0.1 & 0.6 & 0.9 & 0.2 \\ 0.5 & 0.8 & 0.4 & 0.1 \end{bmatrix} = \begin{bmatrix} 0.46 & 1.04 & 1.08 & 0.28 \\ 0.81 & 0.78 & 0.85 & 0.46 \end{bmatrix}$$

即：

	电影Q_1	电影Q_2	电影Q_3	电影Q_4
用户p_1	0.46	1.04	**1.08**	0.28
用户p_2	0.81	0.78	**0.85**	0.46

2. 反推用户、物品矩阵

2.1 FunkSVD(Latent Factor Model, LFM)

因为实际获取到的一般是稀疏矩阵$R_{n\times M}$，我们需要反推$P_{n\times D}$,$Q_{D\times M}$。因此损失函数就是预测的R'与真实的R之间的误差，可选MSE。注意正则化的时候需要考虑每个用户对电影打分的数量不同，因此损失函数是：

$$J(P,Q) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{M}\mathbb{I}_{ij}\left((r'_{ij}-r_{ij})^2 + \lambda(\|\boldsymbol{p}^{(i)}\|^2 + \|\boldsymbol{q}_j\|^2)\right)$$

- 其中，用户i对电影j的评分r'_{ij}是用户偏好与电影特征的内积$r'_{ij} = \boldsymbol{p}^{(i)}\boldsymbol{q}_j$，$\lambda$受示性函数$\mathbb{I}$控制，$\mathbb{I}_{ij}$代表用户$i$是否对电影$j$评分，$\boldsymbol{p}^{(i)}$是用户$i$的特征向量，$\boldsymbol{q}_j$是电影$j$的特征向量。
  向量内积是**双线性函数**bilinear model。双线性的含义为，二元函数固定任意一个自变量时，函数关于另一个自变量线性。

2.2 BiasSVD（Bias Latent Factor Model）

在FunkSVD的基础上，加入用户偏好和物品偏好，即：

$$r'_{ij} = \mu + b_u + b_i + \boldsymbol{p}^{(i)}\boldsymbol{q}_j$$

- 其中，μ是全局平均评分，一般使用所有样本评分的均值；b_u是用户偏好，是用户u给出的所有评分的均值；b_i是物品偏好，是物品i得到的所有评分的均值。

▶ **点击展开查看BiasSVD完整代码**

2.3 MF小结

用户和物品都用隐向量的形式存放，空间复杂度从$O(n^2)$降到$O((n+M)\cdot D)$，其中D是隐向量的维度。
MF模型的优点是可以发现用户和物品的隐含特征，但是也有一些缺点：

- 没有考虑到用户特征，物品特征和上下文特征。
- 在缺乏用户历史行为的时候，无法进行有效的推荐。

2.1.8 因子分解机 $FM(factorization\ machines)$

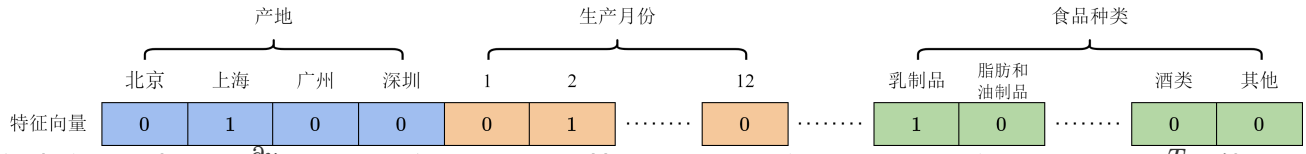
MF的目标是从交互的结果中**计算出用户和物品的特征**。而FM则正好相反，希望通过物品的特征和某个用户点击这些物品的历史记录，**预测该用户点击其他物品的概率**，即点击率(click through rate，CTR)。

因为物品之间可能存在关联，所以也要引入**双线性模型**，记$\boldsymbol{x}$为物品的特征向量，x_i为第i个特征，w_{ij}为第i个特征和第j个特征的权重。FM模型的公式为：

$$y(\boldsymbol{x}) = \sum_{i=0}^{D} \theta_i x_i + \sum_{i=1}^{D-1} \sum_{j=i+1}^{D} w_{ij} x_i x_j$$

即$y(\boldsymbol{x}) = \theta^T \boldsymbol{x} + \frac{1}{2} \boldsymbol{x}^T W \boldsymbol{x}$，其中$x_0 \equiv 1$对应偏置项。时间复杂度$O(D^2)$。

如果将物品离散编码，可以使用**多域独热编码**multi-field one-hot encoding，将物品的多个特征各自独热编码成的向量依次拼接起来。比如：



在此编码方式下，$\frac{\partial y}{\partial w_{ij}} = x_i x_j$在大多数时候等于0，无法更新参数。矩阵分解$W = VV^T$后能解决这一问题，其中$W \in \mathbb{R}^{D \times D}, V \in \mathbb{R}^{D \times k}$，$k$是隐向量的维度。此时算法的时间复杂度为$O(kD^2)$。

算法优化后时间复杂度$O(kD)$：

$$y(x) = \sum_{i=0}^{D} \theta_i x_i + \frac{1}{2} \sum_{f=1}^{k} \left(\left(\sum_{i=1}^{D} v_{if} x_i \right)^2 - \sum_{i=1}^{D} v_{if}^2 x_i^2 \right)$$

其中，v_{if}是第i个特征的第f个隐向量。由于本文章需要将 FM 模型用在召回，故将二阶特征交互项拆分为用户和物品项。记U表示用户相关特征集合，I表示物品相关特征集合。有：

$$\begin{aligned} \text{MatchScore}_{FM} &= V_{item} V_{user}^T \\ &= \sum_{t \in I} w_t x_t + \frac{1}{2} \sum_{f=1}^{k} \left(\left(\sum_{t \in I} v_{t,f} x_t \right)^2 - \sum_{t \in I} v_{t,f}^2 x_t^2 \right) + \sum_{f=1}^{k} \left(\sum_{u \in U} v_{u,f} x_u \sum_{t \in I} v_{t,f} x_t \right) \end{aligned}$$

- 其中，w_t是物品t的权重，$v_{t,f}$是物品t的第f个隐向量，$v_{u,f}$是用户u的第f个隐向量，x_t是物品t的特征向量，x_u是用户u的特征向量。
  用户向量：

$$V_{user} = [1; \quad \sum_{u \in U} v_u x_u]$$

由两项表达式拼接得到，第一项为常数1，第二项是将用户相关的特征向量进行 sum pooling 。
假设用户集合U中有n个用户，每个用户的隐向量v_u的维度为k,则：$v_u = [v_{u1}, v_{u2}, \ldots, v_{uk}]$。对所有用户$u$求和：$\sum_{u \in U} v_u x_u = \sum_{u \in U} [v_{u1} x_u, v_{u2} x_u, \ldots, v_{uk} x_u]$，这意味着我们在每个维度上进行求和，最终结果是一个k维向量。那么，V_{user}的维度为$1 + k$。
物品向量：

$$V_{itcm} = [\sum_{t \in I} w_t x_t + \frac{1}{2} \sum_{f=1}^{k} \left(\left(\sum_{t \in I} v_{t,f} x_t \right)^2 - \sum_{t \in I} v_{t,f}^2 x_t^2 \right); \quad \sum_{t \in I} v_t x_t]$$

第一项表示物品相关特征向量的一阶、二阶特征交互，第二项是将物品相关的特征向量进行 sum pooling 。
同理，第一项的维度为1，假设物品集合I中有m个物品，每个物品的隐向量v_t的维度为k，则：$v_t = [v_{t1}, v_{t2}, \ldots, v_{tk}]$。对所有物品$t$求和：$\sum_{t \in I} v_t x_t = \sum_{t \in I} [v_{t1} x_t, v_{t2} x_t, \ldots, v_{tk} x_t]$，这意味着我们在每个维度上进行求和，最终结果是一个k维向量。那么，V_{itcm}的维度为$1 + k$。

为什么不直接将FM中学习到的User Embedding:$\sum_{u \in U} v_u x_u$和Item Embedding:$\sum_{t \in I} v_t x_t$text的内积做召回呢?
因为用户喜欢的未必一定是与自身最匹配的，也包括一些自身性质极佳的item(e.g.热门item)，所以，非常有必要将 所有Item特征一阶权重之和 和 所有Item特征隐向量两两点积之和 考虑进去。

化简过程见hml第7.3节以及(更推荐)FunRec第2.1.2节

2.1.9 协同过滤小结

1. UserCF和ItemCF的区别

- **UserCF**：适用于用户少，物品多，时效性较强的场合，比较的是人与人之间的相似度，具备更强的社交属性。可以发现用户潜在的尚未发现的兴趣。例如新闻等。
- **ItemCF**：适用于物品少，用户多，物品相对稳定，用户兴趣稳定的场合，比较的是物品与物品之间的相似度，具备更强的个性化推荐属性。例如电影、音乐等。
  - item相比于用户变化的慢，且新item特征比较容易获得，所以ItemCF在实际应用中更加广泛。

2. Swing与MF相较于传统CF的优势

- **Swing**：通过用户共同购买的事件计算物品之间的相似度。
- **MF**：通过矩阵分解的方式，将用户-物品交互矩阵分解为两个低维矩阵的乘积，得到用户和物品的Embedding。

3. 协同过滤算法的权重改进

- **base 公式**

$$w_{ij} = \frac{|N(i) \bigcap N(j)|}{|N(i)|}$$

  - 该公式表示同时喜好物品i和物品j的用户数，占喜爱物品i的比例。
  - 缺点：若物品j为热门物品，那么它与任何物品的相似度都很高。
- **对热门物品进行惩罚**

$$w_{ij} = \frac{|N(i) \cap N(j)|}{\sqrt{|N(i)||N(j)|}}$$

  - 根据 base 公式在的问题，对物品j进行打压。打压的出发点很简单，就是在分母再除以一个物品j被购买的数量
  - 此时，若物品j为热门物品，那么对应的$N(j)$也会很大，受到的惩罚更多。
- **控制对热门物品的惩罚力度**

$$w_{ij} = \frac{|N(i) \cap N(j)|}{|N(i)|^{1-\alpha}|N(j)|^{\alpha}}$$

  - 除了第二点提到的办法，在计算物品之间相似度时可以对热门物品进行惩罚外。
  - 可以在此基础上，进一步引入参数 α ,这样可以通过控制参数 α来决定对热门物品的惩罚力度。
- **对活跃用户的惩罚**

$$w_{ij} = \frac{\sum_{\mathrm{u \in N(i) \cap N(j)}} \frac{1}{\log 1+|N(u)|}}{|N(i)|^{1-\alpha}|N(j)|^{\alpha}}$$

  - 在计算物品之间的相似度时，可以进一步将用户的活跃度考虑进来。
  - 对于异常活跃的用户，在计算物品之间的相似度时，他的贡献应该小于非活跃用户。

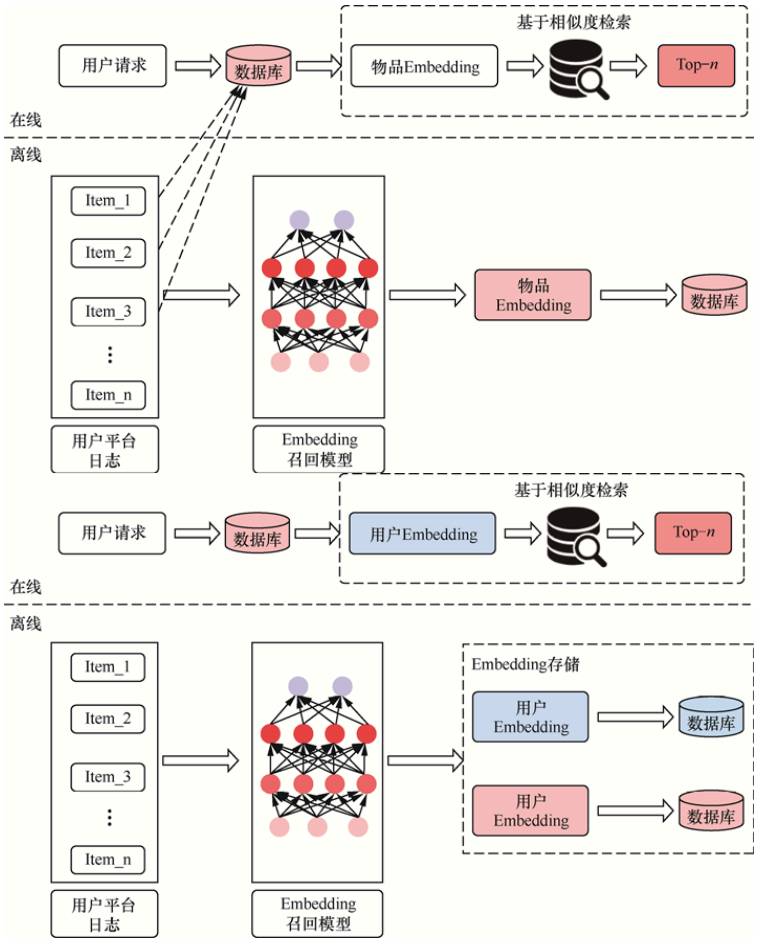
3. 协同过滤算法的问题

- **泛化能力弱**
  - 无法将两个物品相似的信息推广到其他物品的相似性上。
- **头部效应明显， 处理稀疏向量的能力弱**
  - 热门物品具有很强的头部效应，容易跟大量物品产生相似，而尾部物品由于特征向量稀疏，导致很少被推荐。

2.2 基于向量的召回

Embedding就是用一个低维、稠密的向量表示一个对象，相当于对One-hot编码做了平滑处理。它将用户和物品映射到低维空间，通过内积计算相似度。

主要分为**i2i**和**u2u**两种召回，i2i得到的是物品的embedding，u2u得到的是用户和物品的embedding。

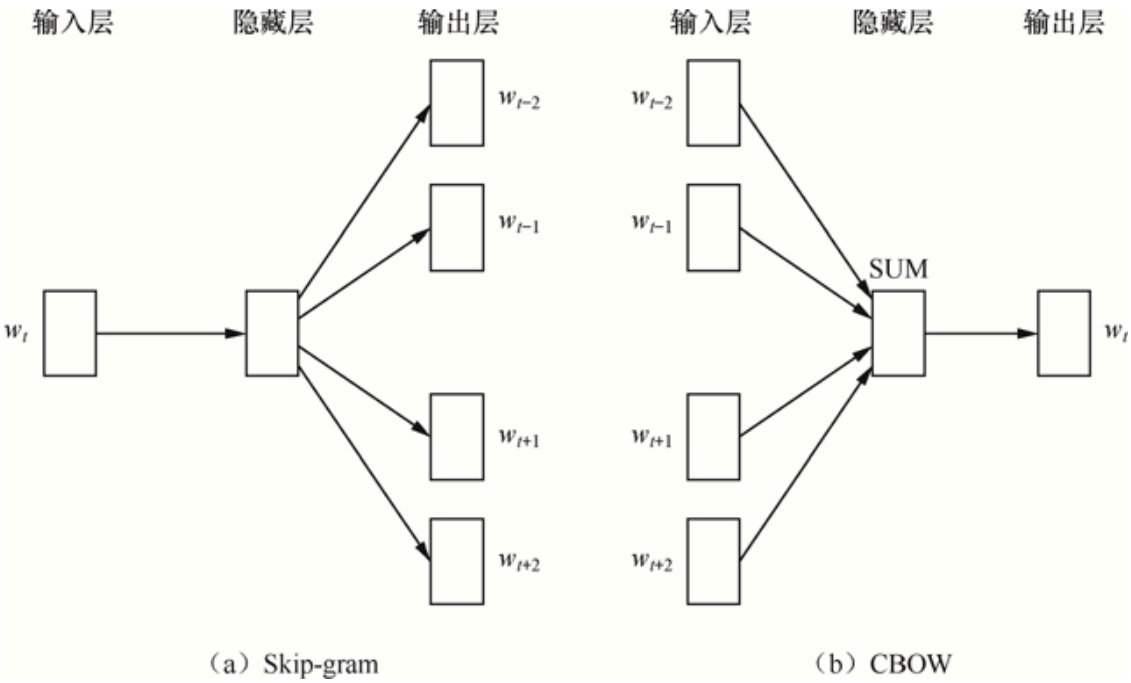


2.2.1 $Word2Vec$

> "You shall know a word by the company it keeps" (J. R. Firth 1957: 11)

分为Skip-gram和CBOW两种模型。

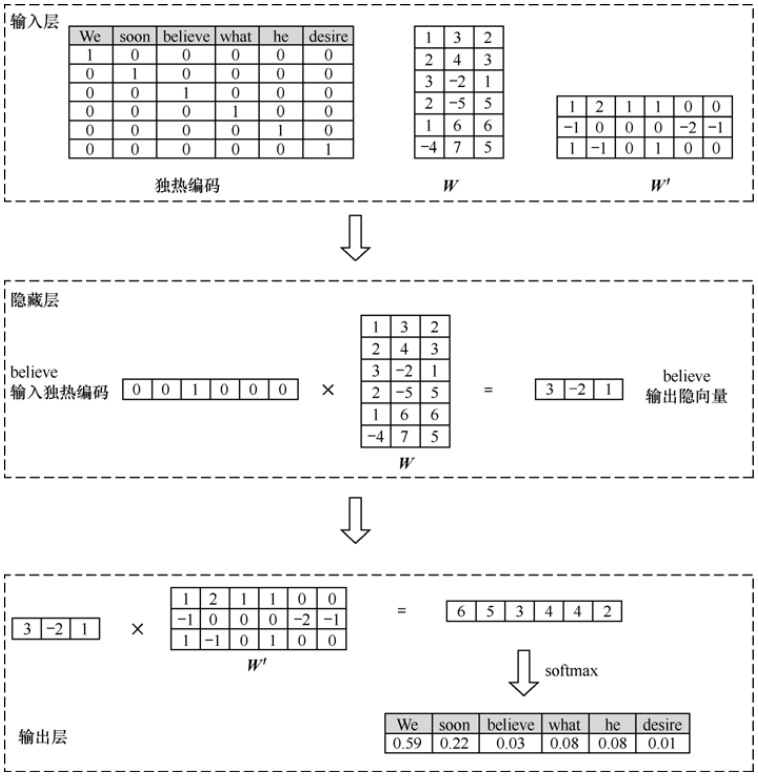
- **Skip-gram**：通过中心词预测上下文
- **CBOW**：通过上下文预测中心词

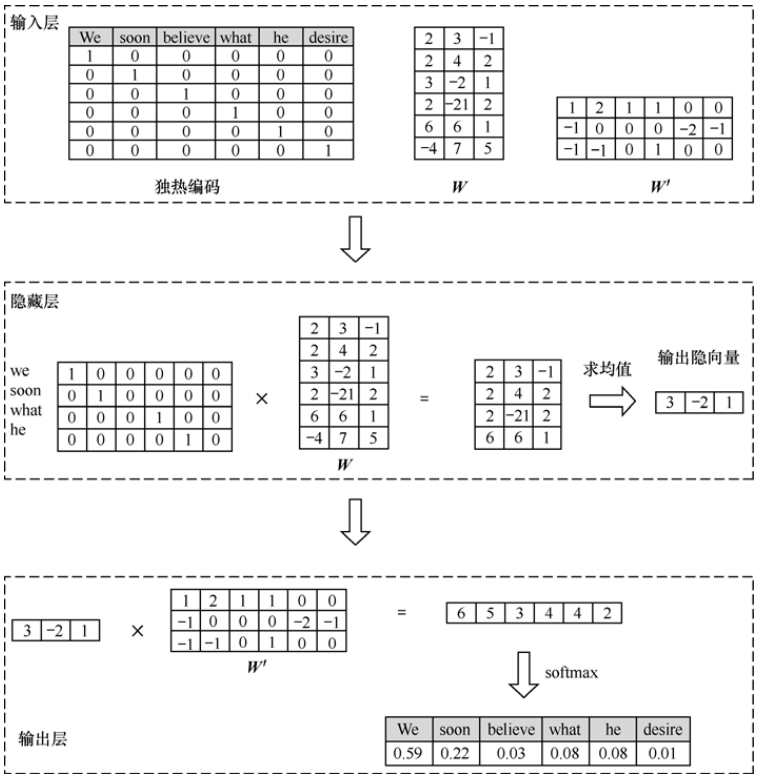


计算过程对比：

左图是根据中心词 `believe` 预测上下文的过程，右图是根据上下文预测中心词 `believe` 的过程。

中心词w_t维度变化是：$(1,V) \rightarrow (1,d) \rightarrow (1,V)$，式中$d$是$W$的列数。第一步是通过$W_{V\times d}$将中心词的one-hot编码转换为低维向量，第二步是通过上下文矩阵$W'_{d\times V}$与softmax将低维向量转换为上下文词的概率分布。





设独热编码矩阵为V，中心词向量为v_i，上下文词向量为u_j，W,W'分别是隐编码和输出概率的权重矩阵。独热编码里的第i行的低维编码就是W的第i行，因此维度从$(1,|V|)$降到$(1,W_D)$，其中W_D是W的列数。
T是文本长度，c是上下文窗口大小，w_t是中心词，w_{t+j}是上下文词，$j \in C$具体而言是$-c \leq j \leq c, j \neq 0$。
使用中心词和上下文词的相似性来计算概率p，最后得到损失函数J：

1. Skip-gram

Skip-gram的损失函数为：

$$J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j\in C}\log p(w_{t+j}|w_t)$$

其中，$p(w_{t+j}|w_t) = \frac{exp(u_j^T v_t)}{\sum_{i=1}^{|V|} exp(u_i^T v_t)}$，$u_j$是上下文词$w_{t+j}$的向量, v_t是中心词w_t的向量，$|V|$是词汇表的大小。

2. CBOW

CBOW的损失函数为：

$$J = -\frac{1}{T}\sum_{t=1}^{T}\log p(w_t|w_{t-j},\cdots,w_{t+j})$$

其中，$\bar{u} = \frac{\sum_{j\in C} u_{t+j}}{|C|}$，$p(w_t|w_{t-c},\cdots,w_{t+c}) = \frac{exp(v_t^T \bar{u})}{\sum_{i=1}^{|V|} exp(v_i^T \bar{u})}$，$u_{t+j}$是上下文词的向量, v_t是中心词的向量，$\bar{u}$是上下文词的平均向量，$|C|$是上下文词的个数。

3. 代码实现

▶ **点击展开查看Word2Vec完整代码**

4. Word2Vec的问题

- 无法处理新词、多义词、词序、长文本
- 计算p的时候分母是所有词的概率，计算量大。可以采用**负采样**，也就是以如下的采样概率选择词w来更新，类似于dropout：

$$P(w) = \frac{f(w)^{\frac{3}{4}}}{\sum_{j=0}^{n}[f(w)^{\frac{3}{4}}]}$$

式中$f(w)$表现其出现频次，n表示词典大小。另外，概率变为$p\left(w_{t+j} \mid w_t\right) = \sigma\left(u_j^T v_t\right) \prod_{i=1}^{|N|} \sigma\left(-u_i^T v_t\right)$，$\sigma(x) = \frac{1}{1+e^{-x}}$，$|N|$是以$P(w)$的概率采样后的negative words的个数。对于小规模数据集，选择5-20个negative words会比较好，对于大规模数据集可以仅选择2-5个negative words。
上面计算的是某词成为负样本的概率，还有一种方法是计算某词w成为正样本的概率$P(w) = \left(\sqrt{\frac{Z(w)}{0.001}} + 1\right) \times \frac{0.001}{Z(w)}$，其中$Z(w)$是词$w$的出现概率，这样能降低热门词被当做正样本的概率。

2.2.2 *Item2Vec*

Item2Vec是Word2Vec的一个变种，用于学习物品的Embedding。它假设对于一个集合的物品，它们之间是相似的，与用户购买它们的顺序、时间无关。

损失函数：

$$\frac{1}{K}\sum_{i=1}^{K}\sum_{j\neq i}^{K}\log p(w_j \mid w_i)$$

- 式中，$p(w_j \mid w_i) = \sigma(u_i^T v_j)\prod_{k=1}^{N}\sigma(-u_i^T v_k)$，$u_i$是物品$i$的向量，$v_j$是物品$j$的向量，$\sigma(x) = \frac{1}{1+e^{-x}}$，$N$是负采样的个数。

在Skip-Gram模型中，每个单词w_i有2个特征表示。Item2Vec同样如此，论文中是将物品的中心词向量u_i作为物品的特征向量。作者还提到了其他两种方式来表示物品向量：add: $u_i + v_i$，concat: $[u_i^T v_i^T]^T$。

2.2.3 *Aribnb*

Airbnb的推荐系统是基于Item2Vec的，它描述了两种 Embedding 的构建方法，分别为：

- 用于描述短期实时性的个性化特征 Embedding：**listing Embeddings**
  `listing` 表示房源的意思。
- 用于描述长期的个性化特征 Embedding：**user-type & listing type Embeddings**

1. Listing Embeddings

假设用户u最近的行为序列为$[l_1, l_2, \ldots, l_n]$,其中l_i表示房源i。使用 Skip-gram 模型，损失函数：

$$\sum_{i=1}^{n} \sum_{-c \le j \le c, j \neq 0} \log P(l_{i+j}|l_i)$$

- 其中，$P(l_{i+j}|l_i) = \frac{exp(v_{l_i}^T v_{l_{i+j}})}{\sum_{k=1}^{|V|} exp(v_{l_i}^T v_k)}$，$v_{l_i}$是房源$i$的向量，$|V|$是房源的数量。

负采样后的损失函数：

$$-\left(\sum_{(l,c)\in\mathcal{D}_p} \log \frac{1}{1+e^{-v_c^T v_l}} + \sum_{(l,c)\in\mathcal{D}_n} \log \frac{1}{1+e^{v_c^T v_l}} \right)$$

- 其中，$\mathcal{D}_p$是正样本集合(是共现对，c在l的一个上下文窗口中出现)，$\mathcal{D}_n$是负样本集合(不是上下文词)，v_c是词的上下文向量，v_l是词向量(词l本身的特征)。

改进--**正负样本集构建**的改进，增加了这两项：

$$\log \frac{1}{1+e^{-v_c^T v_{l_b}}} + \sum_{(l,m_n)\in\mathcal{D}_{m_n}} \log \frac{1}{1+e^{v_{m_n}^T v_l}}$$

- 其中，第一项里的l_b是 `booked listing` ，是用户在session中最终预定的房源，一般只会出现在结束位置，表示用户的最终选择，因此作为全局的上下文(正样本)。
  第二项里的m_n与滑窗中的中心 `listing` 位于同一区域的负样本集，因为这些房源可能是用户可能选择的，但是用户没有选择。如果随机采集负样本，可能会有一些不合理的负样本，例如离用户选择的房源位置 `market` 太远的房源。
- 损失函数变为

$$-\left(\sum_{(l,c)\in\mathcal{D}_p} \log \frac{1}{1+e^{-v_c^T v_l}} + \sum_{(l,c)\in\mathcal{D}_n} \log \frac{1}{1+e^{v_c^T v_l}} + \log \frac{1}{1+e^{-v_c^T v_{l_b}}} + \sum_{(l,m_n)\in\mathcal{D}_{m_n}} \log \frac{1}{1+e^{v_{m_n}^T v_l}} \right)$$

改进--**冷启动问题**的解决:

- 在新的 `listing` 被创建后，房主需要提供如位置、价格、类型等在内的信息。然后利用房主提供的房源信息，为其查找3个相似的 `listing` ，并将它们embedding的均值作为新 `listing` 的embedding表示。

2. User-type & Listing-type Embeddings

Listing Embedding是基于用户的点击sessions学习得到的。同一个session内的点击时间间隔低于30分钟，所以它们更适合短期，session内的**个性化**需求。
长期兴趣的探索是基于 booking session (用户的历史预定序列)，Embedding的构建方法与Listing Embeddings类似，只是在构建时，将用户的行为序列替换为用户的类型序列，房源的行为序列替换为房源的类型序列。

问题：

- booking sessions S_b 数据量的大小远远小于 click sessions S_c
- 用户类型的冷启动，长度为1的 session 无法学习到用户的兴趣
- 用户兴趣的变化， booking 间隔时间较长
- 房源类型的冷启动，平台上大多数 listing 被预定的次数低于5-10次。

因此，Airbnb提出了基于 booking session 来学习用户和房源的Type Embedding。给定一个 booking sessions 集合S_b ,其中包含了M个用户的 booking session：

- 每个 booking session 表示为：$s_b = (l_{b1}, \ldots, l_{bM})$，这里$l_{b1}$表示 listing , 学习到Embedding记作$v_{l_{id}}$

Listing Embedding，与相应的 lisitng 是一一对应的，只包含了 listing 的信息。
而Type Embedding包含了房源和用户的类型信息(User-type Embedding和Listing-type Embedding)。
对于不同的 listing ，它们的Type Embedding可能是相同的(User 同样如此)。比如： listing 可能是apartment_5stars_villa， User 可能是西藏人_男_学生_21岁_工作山东_南翔_挖掘机。这样避免过于稀疏的问题，使得冷启动、动态变化的问题得到解决。

User-type的目标函数：

$$\underset{\theta}{\operatorname{argmax}} \sum_{(u_t,c)\in\mathcal{D}_{book}} \log\frac{1}{1+e^{-v_c^T v_{u_t}}} + \sum_{(u_t,c)\in\mathcal{D}_{neg}} \log\frac{1}{1+e^{v_c^T v_{u_t}}} + \sum_{(u_t,l_t)\in\mathcal{D}_{\text{reject}}} \log\frac{1}{1+e^{v_{l_t}^T v_{u_t}}}$$

Listing-type的目标函数：

$$\underset{\theta}{\operatorname{argmax}} \sum_{(l_t,c)\in\mathcal{D}_{book}} \log\frac{1}{1+e^{-v_c^T v_{l_t}}} + \sum_{(l_t,c)\in\mathcal{D}_{neg}} \log\frac{1}{1+e^{v_c^T v_{l_t}}} + \sum_{(l_t,u_t)\in\mathcal{D}_{\text{reject}}} \log\frac{1}{1+e^{v_{u_t}^T v_{l_t}}}$$

- 其中，$\mathcal{D}_{book}$是正样本集合，$\mathcal{D}_{neg}$是负样本集合，$\mathcal{D}_{reject}$是拒绝样本集合。
- 首先，我们对比与Listing Embedding的区别，在前两项(正负样本)的损失函数中，Listing Embedding是对 listing v_l和 context v_c的内积，而User-type Embedding是对 user v_{u_t}和 context v_c的内积，Listing-type Embedding是对 user v_{l_t}和 listing v_l的内积。
- 其次，为了提高用户预定房源以后，被主人接受的概率。同时，降低房源主人拒绝客人的概率。对于 reject 样本，User-type Embedding是对 user v_{u_t}和 listing v_{l_t}的内积，Listing-type Embedding是对 listing v_{l_t}和 user v_{u_t}的内积。

3. Airbnb检索策略

在给定学习到的Listing Embedding，通过计算其向量v_l和来自同一区域的所有 `listing` 的向量v_j之间的余弦相似度，可以找到给定房源l的相似房源。

- 这些相似房源可在同一日期被预定（如果入住-离开时间已确定）。
- 相似度最高的K个房源被检索为相似房源。
- 基于排序模型GBDT，对预测结果进行排序。

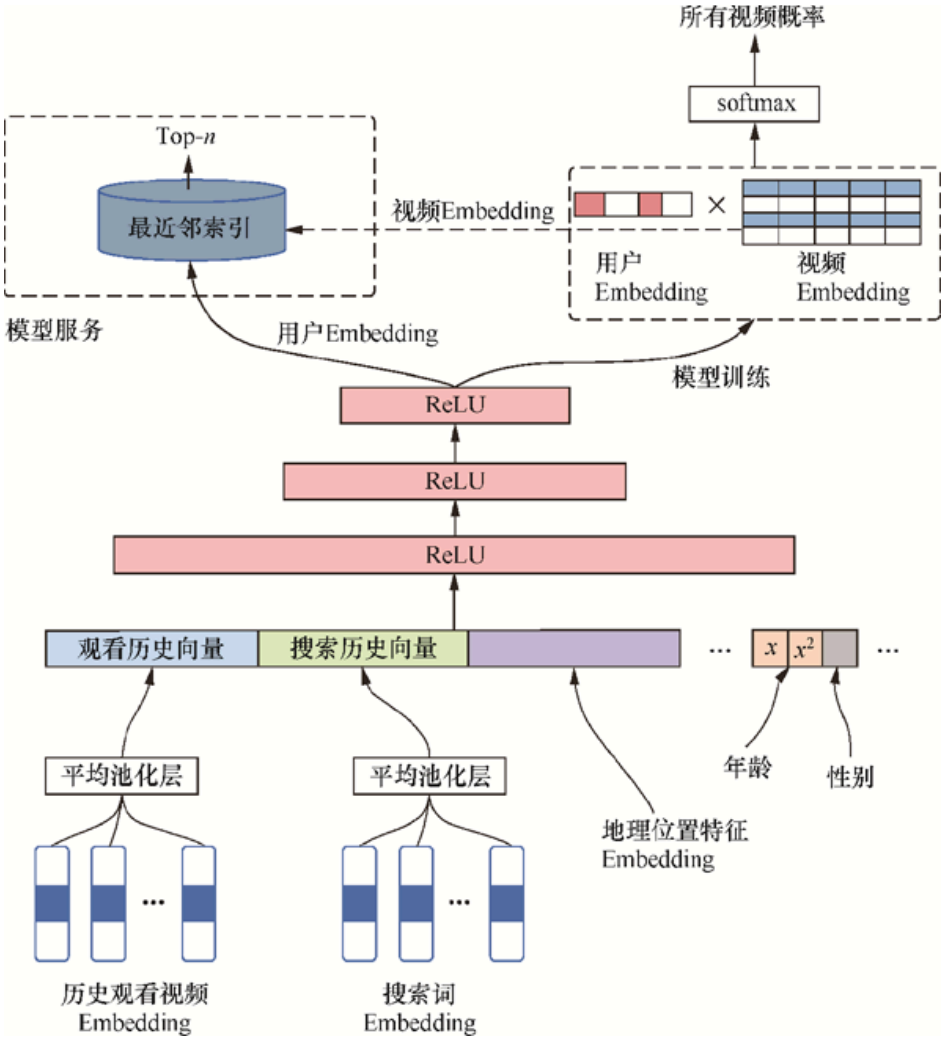
2.2.4 $YoutubeDNN$

YouTubeDNN是一个深度学习模型，用于生成用户和视频的Embedding，目标是最大化观看时间，它的核心是一个深度神经网络，用于预测用户对视频的点击率。
召回模型的目的是在大量YouTube视频中检索出数百个和用户相关的视频来，也就是一个多分类的问题，即用户在某一个时刻点击了某个视频，可以建模成输入一个用户向量u，从海量视频v中预测出被点击的那个视频的概率。

$$P\left(w_t=i \mid U, C\right)=\frac{e^{v_i u}}{\sum_{j \in V} e^{v_j u}}$$

- 式中，w_t是用户在时刻t点击的视频，U是用户向量，C是上下文向量，v_i是视频i的向量，V是视频集合。

1. YoutubeDNN的网络结构(召回模型)



图中模型的输出(ReLU上面)是就是用户u的embedding，图中箭头指向不是很好。。

输入主要是用户侧的特征，包括用户观看的历史video序列，用户搜索的历史tokens，用户的人文特征，如位置、性别、年龄。通过embedding层，将这些特征映射到低维空间，然后将这些特征拼接在一起，输入到DNN中进行降维，最后输出一个用户向量u，而视频向量v_i是模型本身的参数。

- 用户历史、搜索序列，一般是 `item_1,item_2,...,item_n` ，通过embedding层映射到低维空间，然后使用average pooling融合(每一维求平均得到一个最终向量来表示用户的历史兴趣或搜索兴趣)。论文中是每个item事先通过w2v方式算好了的embedding，直接作为了输入，然后进行pooling融合。
- 用户人文特征，也是通过embedding层映射到低维空间，然后拼接在一起，输入到DNN中。对新用户的推荐会比较有帮助。

2. 训练数据的选取和生成