第八章、评分预测问题

讲师：武永亮

教学目标

- 理解离线实验方法
- 理解评分预测算法

目录

评分预测问题

- 很多推荐系统研究最早关注的却是评分预测问题。
- 评分预测问题都是推荐系统研究的核心。评分预测问题最基本的数据集就是用户评分数据集。该数据集由用户评分记录组成，每一条评分记录是一个三元组(u, i, r)，表示用户u给物品i赋予了评分r，本章用r_{ui}表示用户u对物品i的评分。因为用户不可能对所有物品都评分，因此评分预测问题就是如何通过已知的用户历史评分记录预测未知的用户评分记录。

表8-1　评分预测问题举例

	虎口脱险	变形金刚	唐山大兄	少林足球	大话西游	黑客帝国
A	1	?	5	4	5	?
B	4	2	?	3	?	5
C	?	4	?	3	?	5
D	5	?	5	?	2	?
E	?	5	?	?	4	4

评分预测问题

- 评分预测问题基本都通过离线实验进行研究。在给定用户评分数据集后，研究人员会将数据集按照一定的方式分成训练集和测试集，然后根据测试集建立用户兴趣模型来预测测试集中的用户评分。对于测试集中的一对用户和物品（u, i），用户u对物品i的真实评分是r_{ui}，而推荐算法预测的用户u对物品i的评分为r_{ui}’，那么一般可以用均方根误差RMSE度量预测的精度：

$$\mathrm{RMSE} = \frac{\sqrt{\sum_{(u,i)\in T}(r_{ui} - \hat{r}_{ui})^2}}{|\mathrm{Test}|}$$

- 评分预测的目的就是找到最好的模型最小化测试集的RMSE。

目录

评分预测算法

- 最简单的评分预测算法是利用平均值预测用户对物品的评分的。
- 全局平均值
  - 它的定义为训练集中所有评分记录的评分平均值： $\mu = \frac{\sum_{(u,i)\in \text{Train}} r_{ui}}{\sum_{(u,i)\in \text{Train}} 1}$
  - 最终的预测函数可以直接定义为： $\hat{r}_{ui} = \mu$
- 用户评分平均值
  - 用户u的评分平均值定义为用户u在训练集中所有评分的平均值： $\overline{r_u} = \frac{\sum_{i\in N(u)} r_{ui}}{\sum_{i\in N(u)} 1}$
  - 最终的预测函数可以定义为： $\hat{r}_{ui} = \overline{r_u}$
- 物品评分平均值
  - 物品i的评分平均值定义为物品i在训练集中所有评分的平均值： $\overline{r_i} = \frac{\sum_{u\in N(i)} r_{ui}}{\sum_{u\in N(i)} 1}$
  - 最终的预测函数可以定义为： $\hat{r}_{ui} = \overline{r_i}$

评分预测算法

- 用户分类对物品分类的平均值

假设有两个分类函数，一个是用户分类函数 ϕ，一个是物品分类函数 φ。$\phi(u)$ 定义了用户u所属的类，$\varphi(i)$ 定义了物品i所属的类。那么，我们可以利用训练集中同类用户对同类物品评分的平均值预测用户对物品的评分，即：

$$\hat{r}_{ui} = \frac{\sum_{(v,j)\in \text{Train}, \phi(u)=\phi(v), \varphi(i)=\varphi(j)} r_{vj}}{\sum_{(v,j)\in \text{Train}, \phi(u)=\phi(v), \varphi(i)=\varphi(j)} 1}$$

前面提出的全局平均值，用户评分平均值和物品评分平均值都是类类平均值的一种特例。

- 如果定义 $\phi(u)=0,\ \varphi(i)=0$，那么 $\hat{r}_{ui}$ 就是全局平均值。
- 如果定义 $\phi(u)=u,\ \varphi(i)=0$，那么 $\hat{r}_{ui}$ 就是用户评分平均值。
- 如果定义 $\phi(u)=0,\ \varphi(i)=i$，那么 $\hat{r}_{ui}$ 就是物品评分平均值。

评分预测算法

- 在用户评分数据上还可以定义很多不同的分类函数。
  - 用户和物品的平均分
  - 用户活跃度和物品流行度
- 下面的Python代码给出了类类平均值的计算方法。

```
def PredictAll(records, user_cluster, item_cluster):
    total = dict()
    count = dict()
    for r in records:
        if r.test != 0:
            continue
        gu = user_cluster.GetGroup(r.user)
        gi = item_cluster.GetGroup(r.item)
        basic.AddToMat(total, gu, gi, r.vote)
        basic.AddToMat(count, gu, gi, 1)
    for r in records:
        gu = user_cluster.GetGroup(r.user)
        gi = item_cluster.GetGroup(r.item)
        average = total[gu][gi] / (1.0 * count[gu][gi] + 1.0)
        r.predict = average
```

评分预测算法

表8-2　MovieLens数据集上不同平均值方法的RMSE

UserGroup	ItemGroup	Train RMSE	Test RMSE
Cluster	Cluster	1.1171	1.1167
IdCluster	Cluster	1.0289	1.0351
Cluster	IdCluster	0.9754	0.9779
UserActivityCluster	Cluster	1.1100	1.1093
UserActivityCluster	IdCluster	0.9740	0.9914
Cluster	ItemPopularityCluster	1.0902	1.0891
IdCluster	ItemPopularityCluster	1.0004	1.0258
UserActivityCluster	ItemPopularityCluster	1.0860	1.0847
UserVoteCluster	Cluster	1.0370	1.0425
UserVoteCluster	IdCluster	0.9209	0.9441
Cluser	ItemVoteCluster	0.9841	0.9864
IdCluster	ItemVoteCluster	0.9055	0.9449
UserVoteCluster	ItemVoteCluster	**0.9272**	**0.9342**

评分预测算法

- 基于用户的邻域算法认为预测一个用户对一个物品的评分，需要参考和这个用户兴趣相似的用户对该物品的评分，即:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in S(u,K) \cap N(i)} w_{uv}(r_{vi} - \bar{r}_v)}{\sum_{v \in S(u,K) \cap N(i)} |w_{uv}|}$$

这里，$S(u,K)$是和用户u兴趣最相似的K个用户的集合，$N(i)$是对物品i评过分的用户集合，r_{vi}是用户v对物品i的评分，$\bar{r}_v$是用户v对他评过分的所有物品评分的平均值。用户之间的相似度w_{uv}可以通过皮尔逊系数计算：

$$w_{uv} = \frac{\sum_{i \in I}(r_{ui} - \bar{r}_u) \cdot (r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I}(r_{ui} - \bar{r}_u)^2 \sum_{i \in I}(r_{vi} - \bar{r}_v)^2}}$$

评分预测算法

- 基于物品的邻域算法在预测用户u对物品i的评分时，会参考用户u对和物品i相似的其他物品的评分，即：

$$\hat{r}_{ui} = \overline{r_i} + \frac{\sum_{j \in S(u,K) \cap N(u)} w_{ij} (r_{uj} - \overline{r_i})}{\sum_{j \in S(i,K) \cap N(u)} |w_{ij}|}$$

这里，$S(i, K)$是和i最相似的物品集合，$N(u)$是用户u评过分的物品集合，w_{ij}是物品之间的相似度，$\overline{r_i}$是物品i的平均分。对于如何计算物品的相似度，Badrul Sarwar等在论文[①]里做了详细的

评分预测算法

- 物品相似度的计算方法：

第一种是普通的余弦相似度（cosine similarity）：

$$w_{ij} = \frac{\sum_{u \in U} r_{ui} \cdot r_{uj}}{\sqrt{\sum_{u \in U} r_{ui}^2 \sum_{u \in U} r_{uj}^2}}$$

第二种是皮尔逊系数（pearson correlation）：

$$w_{ij} = \frac{\sum_{u \in U} (r_{ui} - \overline{r}_i) \cdot (r_{uj} - \overline{r}_j)}{\sqrt{\sum_{u \in U} (r_{ui} - \overline{r}_i)^2 \sum_{u \in U} (r_{uj} - \overline{r}_j)^2}}$$

第三种被Sarwar称为修正的余弦相似度（adjust cosine similarity）：

$$w_{ij} = \frac{\sum_{u \in U} (r_{ui} - \overline{r}_u) \cdot (r_{uj} - \overline{r}_u)}{\sqrt{\sum_{u \in U} (r_{ui} - \overline{r}_u)^2 \sum_{u \in U} (r_{uj} - \overline{r}_u)^2}}$$

评分预测算法

- 在推荐系统领域，提的最多的就是潜语义模型和矩阵分解模型。其实，这两个名词说的是一回事，就是如何通过降维的方法将评分矩阵补全。
- 用户的评分行为可以表示成一个评分矩阵R，其中$R[u][i]$就是用户u对物品i的评分。但是，用户不会对所有的物品评分，所以这个矩阵里有很多元素都是空的，这些空的元素称为缺失值（missing value）。因此，评分预测从某种意义上说就是填空，如果一个用户对一个物品没有评过分，那么推荐系统就要预测这个用户是否是否会对这个物品评分以及会评几分。

评分预测算法

- 最早的矩阵分解模型就是SVD（奇异值分解）。
- 给定m个用户和n个物品，和用户对物品的评分矩阵。首先需要对评分矩阵中的缺失值进行简单地补全，比如用全局平均值，或者用户/物品平均值补全，得到补全后的矩阵R'。接着，可以用SVD分解将

$$R' = U^T S V$$

其中 $U \in \mathbb{R}^{k \times m}$ ， $V \in \mathbb{R}^{k \times n}$ 是两个正交矩阵， $S \in \mathbb{R}^{k \times k}$ 是对角阵，对角线上的每一个元素都是矩阵的奇异值。为了对R进行降维，可以取最大的f个奇异值组成对角矩阵S_f，并且找到这f个奇异值中每个值在U、 V矩阵中对应的行和列，得到U_f、 V_f，从而可以得到一个降维后的评分矩阵：

$$R'_f = U_f^T S_f V_f$$

其中， $R'_f(u,i)$ 就是用户u对物品i评分的预测值。

评分预测算法

- SVD分解是早期推荐系统研究常用的矩阵分解方法，不过该方法具有以下缺点，因此很难在实际系统中应用。
  - 该方法首先需要用一个简单的方法补全稀疏评分矩阵。一般来说，推荐系统中的评分矩阵是非常稀疏的，一般都有95%以上的元素是缺失的。而一旦补全，评分矩阵就会变成一个稠密矩阵，从而使评分矩阵的存储需要非常大的空间，这种空间的需求在实际系统中是不可能接受的。
  - 该方法依赖的SVD分解方法的计算复杂度很高，特别是在稠密的大规模矩阵上更是非常慢。一般来说，这里的SVD分解用于1000维以上的矩阵就已经非常慢了，而实际系统动辄是上千万的用户和几百万的物品，所以这一方法无法使用。如果仔细研究关于这一方法的论文可以发现，实验都是在几百个用户、几百个物品的数据集上进行的。

评分预测算法

- Simon Funk的SVD分解

第3章曾经简单介绍过LFM在TopN推荐中的应用，因此这里我们不再详细介绍这一方面背后的思想。从矩阵分解的角度说，如果我们将评分矩阵R分解为两个低维矩阵相乘：

$$\hat{R} = P^T Q$$

其中 $P \in \mathbb{R}^{f \times m}$ 和 $Q \in \mathbb{R}^{f \times n}$ 是两个降维后的矩阵。那么，对于用户u对物品i的评分的预测值 $\hat{R}(u,i) = \hat{r}_{ui}$，可以通过如下公式计算：

$$\hat{r}_{ui} = \sum_f p_{uf} q_{if}$$

其中 $p_{uf} = P(u,f)$，$q_{if} = Q(i,f)$。那么，Simon Funk的思想很简单：可以直接通过训练集中的观察值利用最小化RMSE学习P、Q矩阵。

评分预测算法

- Simon Funk的SVD分解

Simon Funk认为，既然我们用RMSE作为评测指标，那么如果能找到合适的P、Q来最小化训练集的预测误差，那么应该也能最小化测试集的预测误差。因此，Simon Funk定义损失函数为：

$$C(p,q)=\sum_{(u,i)\in \text{Train}}(r_{ui}-\hat{r}_{ui})^2=\sum_{(u,i)\in \text{Train}}\left(r_{ui}-\sum_{f=1}^{F}p_{uf}q_{if}\right)^2$$

直接优化上面的损失函数可能会导致学习的过拟合，因此还需要加入防止过拟合项$\lambda(\|p_u\|^2+\|q_i\|^2)$，其中$\lambda$是正则化参数，从而得到：

$$C(p,q)=\sum_{(u,i)\in \text{Train}}\left(r_{ui}-\sum_{f=1}^{F}p_{uf}q_{if}\right)^2+\lambda\left(\|p_u\|^2+\|q_i\|^2\right)$$

要最小化上面的损失函数，我们可以利用随机梯度下降法[①]。该算法是最优化理论里最基础的优化算法，它首先通过求参数的偏导数找到最速下降方向，然后通过迭代法不断地优化参数。下面我们将介绍优化方法的数学推导。

评分预测算法

- Simon Funk的SVD分解

上面定义的损失函数里有两组参数（p_{uf}和q_{if}），最速下降法需要首先对它们分别求偏导数，可以得到：

$$\frac{\partial C}{\partial p_{uf}} = -2q_{ik} + 2\lambda p_{uk}$$

$$\frac{\partial C}{\partial p_{if}} = -2p_{uk} + 2\lambda q_{ik}$$

然后，根据随机梯度下降法，需要将参数沿着最速下降方向向前推进，因此可以得到如下递推公式：

$$p_{uf} = p_{uf} + \alpha(q_{ik} - \lambda p_{uk})$$

$$q_{if} = q_{if} + \alpha(p_{uk} - \lambda q_{ik})$$

其中，α是学习速率（learning rate），它的取值需要通过反复实验获得。

评分预测算法

- 下面的代码实现了学习LFM模型时的迭代过程。在LearningLFM函数中，输入train是训练集中的用户评分记录， F是隐类的格式， n是迭代次数。

```
def LearningLFM(train, F, n, alpha, lambda):
    [p,q] = InitLFM(train, F)
    for step in range(0, n):
        for u,i,rui in train.items():
            pui = Predict(u, i, p, q)
            eui = rui - pui
            for f in range(0,F):
                p[u][k] += alpha * (q[i][k] * eui - lambda * p[u][k])
                q[i][k] += alpha * (p[u][k] * eui - lambda * q[i][k])
        alpha *= 0.9
    return list(p, q)
```

评分预测算法

- 初始化P、 Q矩阵的方法很多，一般都是将这两个矩阵用随机数填充，但随机数的大小还是有讲究的，根据经验，随机数需要和1/sqrt(F)成正比。下面的代码实现了初始化功能。
- 而预测用户u对物品i的评分可以通过如下代码实现：

```
def InitLFM(train, F):
    p = dict()
    q = dict()
    for u, i, rui in train.items():
        if u not in p:
            p[u] = [random.random()/math.sqrt(F) \
                    for x in range(0,F)]
        if i not in q:
            q[i] = [random.random()/math.sqrt(F) \
                    for x in range(0,F)]
    return list(p, q)
```

```
def Predict(u, i, p, q):
    return sum(p[u][f] * q[i][f] for f
 in range(0,len(p[u]))
```

评分预测算法

- 这个预测公式通过隐类将用户和物品联系在了一起。但是，实际情况下，一个评分系统有些固有属性和用户物品无关，而用户也有些属性和物品无关，物品也有些属性和用户无关。因此，Netflix Prize中提出了另一种LFM，其预测公式如下：

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T \cdot q_i$$

这个预测公式中加入了3项 μ 、b_u 、b_i 。本章将这个模型称为BiasSVD。这个模型中新增加的三项的作用如下。

- μ 训练集中所有记录的评分的全局平均数。在不同网站中，因为网站定位和销售的物品不同，网站的整体评分分布也会显示出一些差异。比如有些网站中的用户就是喜欢打高分，而另一些网站的用户就是喜欢打低分。而全局平均数可以表示网站本身对用户评分的影响。
- b_u 用户偏置（user bias）项。这一项表示了用户的评分习惯中和物品没有关系的那种因素。比如有些用户就是比较苛刻，对什么东西要求都很高，那么他的评分就会偏低，而有些用户比较宽容，对什么东西都觉得不错，那么他的评分就会偏高。
- b_i 物品偏置（item bias）项。这一项表示了物品接受的评分中和用户没有什么关系的因素。比如有些物品本身质量就很高，因此获得的评分相对都比较高，而有些物品本身质量很差，因此获得的评分相对都会比较低。

评分预测算法

- 增加的3个参数中，只有*bu*、*bi* 是要通过机器学习训练出来的。同样可以求导，然后用梯度下降法求解这两个参数，我们对LearningLFM稍做修改，就可以支持BiasLFM模型：

```
def LearningBiasLFM(train, F, n, alpha, lambda, mu):
    [bu, bi, p,q] = InitLFM(train, F)
    for step in range(0, n):
        for u,i,rui in train.items():
            pui = Predict(u, i, p, q, bu, bi, mu)
            eui = rui - pui
            bu[u] += alpha * (eui - lambda * bu[u])
            bi[i] += alpha * (eui - lambda * bi[i])
            for f in range(0,F):
                p[u][k] += alpha * (q[i][k] * eui - lambda * p[u][k])
                q[i][k] += alpha * (p[u][k] * eui - lambda * q[i][k])
        alpha *= 0.9
    return list(bu, bi, p, q)
```

评分预测算法

- 而bu、bi在一开始只要初始化成全0的向量。

```
def InitBiasLFM(train, F):
    p = dict()
    q = dict()
    bu = dict()
    bi = dict()
    for u, i, rui in train.items():
        bu[u] = 0
        bi[i] = 0
        if u not in p:
            p[u] = [random.random()/math.sqrt(F) for x in range(0,F)]
        if i not in q:
            q[i] = [random.random()/math.sqrt(F) for x in range(0,F)]
            return list(p, q)

def Predict(u, i, p, q, bu, bi, mu):
    ret = mu + bu[u] + bi[i]
    ret += sum(p[u][f] * q[i][f] for f in range(0,len(p[u]))
    return ret
```

评分预测算法

- 无论是MovieLens数据集还是Netflix Prize数据集都包含时间信息，对于用户每次的评分行为，都给出了行为发生的时间。因此，很多研究人员提出了利用时间信息降低预测误差的方法。
- 利用时间信息的方法也主要分成两种，
  - 一种是将时间信息应用到基于邻域的模型中
  - 另一种是将时间信息应用到矩阵分解模型中

评分预测算法

- Netflix Prize的参赛队伍BigChaos在技术报告中提到了一种融入时间信息的基于邻域的模型，本节将这个模型称为TItemCF。该算法通过如下公式预测用户在某一个时刻会给物品什么评分：

$$\hat{r}_{uit} = \frac{\sum_{j \in N(u) \cap S(i,K)} f(w_{ij}, \Delta t) r_{uj}}{\sum_{j \in N(u) \cap S(i,K)} f(w_{ij}, \Delta t)}$$

这里，$\Delta t = t_{ui} - t_{uj}$ 是用户u对物品i和物品j评分的时间差，w_{ij} 是物品i和j的相似度，$f(w_{ij}, \Delta t)$ 是一个考虑了时间衰减后的相似度函数，它的主要目的是提高用户最近的评分行为对推荐结果的影响，BigChaos在模型中采用了如下的 f ：

$$f(w_{ij}, \Delta t) = \sigma(\delta \cdot w_{ij} \cdot \exp\left(\frac{-|\Delta t|}{\beta}\right) + \gamma)$$

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

这里，$\sigma(x)$ 是sigmoid函数，它的目的是将相似度压缩到（0，1）区间中。从上面的定义可以发现，随着 Δt 增加，$f(w_{ij}, \Delta t)$ 会越来越小，也就是说用户很久之前的行为对预测用户当前评分的影响越来越小。

评分预测算法

- 在引入时间信息后，用户评分矩阵不再是一个二维矩阵，而是变成了一个三维矩阵。不过，我们可以仿照分解二维矩阵的方式对三维矩阵进行分解。

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T \cdot q_i$$

这里，μ 可以看做对二维矩阵的零维分解，b_u 、b_i 可以看做对二维矩阵的一维分解，而 $p_u^T \cdot q_i$ 可以看做对二维矩阵的二维分解。仿照这种分解，我们可以将用户–物品–时间三维矩阵如下分解：

$$\hat{r}_{uit} = \mu + b_u + b_i + b_t + p_u^T \cdot q_i + x_u^T \cdot y_t + s_i^T z_t + \sum_f g_{u,f} h_{i,f} l_{t,f}$$

这里 b_t 建模了系统整体平均分随时间变化的效应，$x_u^T \cdot y_t$ 建模了用户平均分随时间变化的效应，$s_i^T z_t$ 建模了物品平均分随时间变化的效应，而 $\sum_f g_{u,f} h_{i,f} l_{t,f}$ 建模了用户兴趣随时间影响的效应。这个模型也可以很容易地利用前面提出的随机梯度下降法进行训练。本章将这个模型记为TSVD。

评分预测算法

- Koren在SVD++模型的基础上也引入了时间效应

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \cdot (p_u + \frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} y_j)$$

我们可以对这个模型做如下改进以融合时间信息：

$$\hat{r}_{uit} = \mu + b_u(t) + b_i(t) + q_i^T \cdot (p_u(t) + \frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} y_j)$$

$$b_u(t) = b_u + \alpha_u \cdot \mathrm{dev}_u(t) + b_{ut} + b_{u,\mathrm{period}(t)}$$

$$\mathrm{dev}_u(t) = \mathrm{sign}(t - t_u) \cdot |t - t_u|^{\beta}$$

$$b_i(t) = b_i + b_{it} + b_{i,\mathrm{period}(t)}$$

$$p_{uf}(t) = p_{uf} + p_{utf}$$

这里，t_u 是用户所有评分的平均时间。$\mathrm{period}(t)$ 考虑了季节效应，可以定义为时刻t所在的月份。该模型同样可以通过随机梯度下降法进行优化。

评分预测算法

- Netflix Prize的最终获胜队伍通过融合上百个模型的结果才取得了最终的成功。由此可见模型融合对提高评分预测的精度至关重要。

假设已经有一个预测器 $\hat{r}^{(k)}$，对于每个用户–物品对(u, i)都给出预测值，那么可以在这个预测器的基础上设计下一个预测器 $\hat{r}^{(k+1)}$ 来最小化损失函数：

$$C = \sum_{(u,i)\in \text{Train}} \left(r_{ui} - \hat{r}_{ui}^{(k)} - \hat{r}_{ui}^{(k+1)} \right)^2$$

由上面的描述可以发现，级联融合很像Adaboost算法。和Adaboost算法类似，该方法每次产生一个新模型，按照一定的参数加到旧模型上去，从而使训练集误差最小化。不同的是，这里每次生成新模型时并不对样本集采样，针对那些预测错的样本，而是每次都还是利用全样本集进行预测，但每次使用的模型都有区别。

评分预测算法

- 一般来说，级联融合的方法都用于简单的预测器，比如前面提到的平均值预测器。下面的Python代码实现了利用平均值预测器进行级联融合的方法。

```
def Predict(train, test, alpha):
    total = dict()
    count = dict()
    for record in train:
        gu = GetUserGroup(record.user)
        gi = GetItemGroup(record.item)
        AddToMat(total, gu, gi, record.vote - record.predict)
        AddToMat(count, gu, gi, 1)
    for record in test:
        gu = GetUserGroup(record.user)
        gi = GetUserGroup(record.item)
        average = total[gu][gi] / (1.0 * count[gu][gi] + alpha)
        record.predict += average
```

评分预测算法

表8-3　MovieLens数据集中对平均值方法采用级联融合后的效果

UserGroup	ItemGroup	Train RMSE	Test RMSE
Cluster	Cluster	1.1171	1.1167
IdCluster	Cluster	1.0282	1.0344
Cluster	IdCluster	0.9186	0.9274
UserActivityCluster	Cluster	0.9165	0.9254
Cluster	ItemPopularityCluster	0.9164	0.9253
UserVoteCluster	Cluster	0.9142	0.9222
Cluser	ItemVoteCluster	0.9140	0.9221
UserVoteCluster	ItemVoteCluster	0.9123	0.9205
UserActivityCluster	ItemPopularityCluster	0.9121	0.9202

评分预测算法

- 模型加权融合

假设我们有K个不同的预测器$\{\hat{r}^{(1)}, \hat{r}^{(2)}, \cdots, \hat{r}^{(K)}\}$，本节主要讨论如何将它们融合起来获得最低的预测误差。

最简单的融合算法就是线性融合，即最终的预测器$\hat{r}$是这K个预测器的线性加权：

$$\hat{r} = \sum_{k=1}^{K} \alpha_k \hat{r}^{(k)}$$

评分预测算法

- 一般来说，评分预测问题的解决需要在训练集上训练K个不同的预测器，然后在测试集上作出预测。但是，如果我们继续在训练集上融合K个预测器，得到线性加权系数，就会造成过拟合的问题。因此，在模型融合时一般采用如下方法。
  - 假设数据集已经被分为了训练集A和测试集B，那么首先需要将训练集 A按照相同的分割方法分为A 1和A 2，其中A 2的生成方法和B的生成方法一致，且大小相似。
  - 在A 1上训练K个不同的预测器，在A 2上作出预测。因为我们知道A 2上的真实评分值，所以可以在A 2上利用最小二乘法⑨计算出线性融合系数 α_k
  - 在A上训练K个不同的预测器，在B上作出预测，并且将这K个预测器在B上的预测结果按照已经得到的线性融合系数加权融合，以得到最终的预测结果。

评分预测算法

表8-4　Netflix Prize上著名算法的RMSE

方　　法	参　　数	RMSE
Global Average		1.1296
Item Average		1.0526
ItemCF	$K=25$	0.9496
RSVD	$F=96$	0.9094②
Bias-RSVD	$F=96$	0.9039③
SVD++	$F=50$	0.8952④
TimeSVD++	$F=50$	0.8824⑤

内容回顾

1. 离线实验方法
2. 评分预测算法

Thank you !