1. 传统的SVD算法

将这个用户物品对应的 $m \times n$ 矩阵 M 进行SVD分解，并通过选择部分较大的一些奇异值来同时进行降维，即矩阵M此时分解为：

$$M_{m\times n)} = U_{m\times k}\Sigma_{k\times k}V^T_{k\times n}$$

- 其中k是矩阵 M 中较大的部分奇异值的个数，一般会远远的小于用户数和物品树。
- 如果我们要预测第 u 个用户对第 j 个物品的评分 $\hat{r}_{uj}$, 则只需要计算 $u^T_u\Sigma v_j$ 即可。通过这种方法，我们可以将评分表里面所有没有评分的位置得到一个预测评分。通过找到最高的若干个评分对应的物品推荐给用户。
- 致命的 `缺点`
  SVD分解要求矩阵是 `稠密` 、大维度矩阵做SVD分解非常 `耗时`

2. Funk-SVD

2006年，Simon Funk在博客上公开发表了一个只考虑已有 `评分` 记录的矩阵分解方法，称为 `Funk-SVD` ，即被Yehuda Koren称为 `隐语义模型` (LFM)的矩阵分解方法。
它的出发点为，既然将一个矩阵做SVD分解成3个矩阵很耗时，同时还面临稀疏的问题，那么我们能不能避开稀疏问题，同时只分解成两个矩阵呢？即期望我们的矩阵M这样进行分解：

$$M_{(m,n)} = P_{m\times k}Q^T_{n\times k}$$

- `矩阵P` (u,k)：用户 u 对特征 k 的偏好程度.
- `矩阵Q` (j,k)：物品 j 拥有特征 k 的程度.

采用 `线性回归` 的思想，目标是使用户真实评分 $(r_{u,j})$ 与矩阵乘积(预测函数)的 `残差` 尽可能的小。其中，预测函数表示用户 u 对物品 j 的偏好，即

$$\hat{r}_{u,j} = p_u q^T_j$$

用 `均方差` 表示损失函数

$$\arg\min_{p_u q_j} = \sum \frac{1}{2}(r_{u,j} - \hat{r}_{u,j})^2$$

为防止过拟合，加入一个 L_2 的 `正则化项` ，因此 Funk-SVD的优化目标函数为:

$$\arg\min_{p_u q_j} \sum_{u=1}^{m}\sum_{j=1}^{n}\frac{1}{2}(r_{u,j} - \hat{r}_{u,j})^2 + \frac{\lambda}{2}(||p_u||^2_2 + ||q_j||^2_2)$$

- $m_{u,j}$ 为用户 u 对物品 j 的真实评分

- λ 是正则化系数，需要调参。

评分矩阵为 $M_{m \times n}$，通过直接优化以上损失函数得到用户特征矩阵 $P(m \times k)$ 和物品特征矩阵$Q(n \times k)$，其中 $k << m, n$。对于这个优化问题，一般通过 梯度下降法 来进行优化得到结果。
将上式分别对 p_u , q_j求导，得：

$$\frac{\partial J}{\partial p_u} = -(r_{uj} - \hat{r}_{uj})q_j + \lambda p_u$$
$$\frac{\partial J}{\partial q_j} = -(r_{uj} - \hat{r}_{uj})p_u + \lambda q_j$$

在梯度下降法迭代时，p_u,q_j 的迭代公式为 (η : 学习率)：

$$p_u = p_u + \eta[(r_{uj} - \hat{r}_{uj})q_j - \lambda p_u]$$
$$q_j = q_j + \eta[(r_{uj} - \hat{r}_{uj})p_u - \lambda q_j]$$

- 梯度下降

```
def sgd(self, u, j, y_true):
    '''
    梯度下降更新参数
    '''
    err = y_true - self.predict(u, j)
    self.P[u] += self.learning_rate * (err * self.Q[j] - self._lambda * self.P[u])
    self.Q[j] += self.learning_rate * (err * self.P[u] - self._lambda * self.Q[j])
```

- 模型预测

```
def _know_u(self, u:int):
    return u!=None and u>=0 and u<self.P.shape[0]

def _know_j(self, j:int):
    return j!=None and j>=0 and j<self.Q.shape[0]

def predict(self, u:int, j:int):
    '''
    预测u用户对物品i的评分
    :param u:   用户
    :param j:   物品
    '''
    if self._know_u(u) and self._know_j(j):
        return np.dot(self.P[u, :], self.Q[j, :])
    else:
        return None
```

2. Bias-SVD

BiasSVD矩阵分解主要是在正则化项中加入偏置约束。这些约束都是由独立于用户或物品的因素组成，与用户对物品的偏好无关。

- 个性化部分：用户和物品的交互，即用户对物品的喜好程度
- `偏置(Bias)` 部分： 独立于用户或独立于物品的因素。主要由三个子部分组成，分别是

- 训练集中所有评分记录的全局平均数 μ，表示了训练数据的总体评分情况
- 用户偏置 b_u，表示某一特定用户的打分习惯。例如乐观型用户则打分比较保守，总体打分要偏高。
- 物品偏置 b_j，表示某一特定物品得到的打分情况。例如好电影获得的总体评分偏高。

综上，偏置部分可以表示为

$$b_{uj} = \mu + b_u + b_j$$

预测评分函数表示为

$$\hat{r}_{u,j} = p_u q_j^T + b_{uj} = p_u q_j^T + \mu + b_u + b_j$$

从而优化目标函数 $J(p, q, b_u, b_j)$ ($r_{uj} = m_{uj}$)表示为:

$$\arg\min \sum_{r_{uj} \in R_{train}} \frac{1}{2}(r_{uj} - \hat{r}_{uj})^2 + \frac{\lambda}{2}\left(b_u^2 + b_j^2 + ||q_j||^2 + ||p_u||^2\right)$$

将上式分别对 p_u、q_j、b_u、b_j求导，得:

$$\frac{\partial J}{\partial p_u} = -(r_{uj} - \hat{r}_{uj})q_j + \lambda p_u$$
$$\frac{\partial J}{\partial q_j} = -(r_{uj} - \hat{r}_{uj})p_u + \lambda q_j$$
$$\frac{\partial J}{\partial b_u} = -(r_{uj} - \hat{r}_{u,j}) + \lambda b_u$$
$$\frac{\partial J}{\partial b_j} = -(r_{uj} - \hat{r}_{u,j}) + \lambda b_j$$

在梯度下降法迭代时，p_u、q_j、b_u、b_j 的迭代公式为 (η : 学习率):

$$p_u = p_u + \eta[(r_{uj} - \hat{r}_{uj})q_j - \lambda p_u]$$
$$q_j = q_j + \eta[(r_{uj} - \hat{r}_{uj})p_u - \lambda q_j]$$
$$b_u = b_u + \eta[(r_{uj} - \hat{r}_{u,j}) - \lambda b_u]$$
$$b_j = b_j + \eta[(r_{uj} - \hat{r}_{u,j}) - \lambda b_j]$$

- 梯度下降

```
def sgd(self, u, j, y_true):
    '''
    梯度下降更新参数
    '''
    e_uj = y_true - self.predict(u, j)
    self.P[u] += self.learning_rate * (e_uj * self.Q[j] - self._lambda * self.P[u])
    self.Q[j] += self.learning_rate * (e_uj * self.P[u] - self._lambda * self.Q[j])
    self.bu[u] += self.learning_rate * (e_uj - self._lambda * self.bu[u])
    self.bj[j] += self.learning_rate * (e_uj - self._lambda * self.bj[j])
```

- 预测部分

```
def predict(self, u:int, j:int):
    '''
    预测u用户对物品i的评分
    '''
```

```python
    rating = self.mu
    know_u = self._know_u(u)
    know_j = self._know_j(j)
    if know_u: rating += self.bu[u]
    if know_j: rating += self.bj[j]
    if know_u and know_j:
        rating += np.dot(self.P[u, :], self.Q[j, :])
    return rating
```

4. SVD++

后来又提出了对BiasSVD改进的SVD++。它是基于这样的假设：除了显示的评分行为以外，用户对于商品的 浏览记录 或 购买记录 （隐式反馈）也可以从侧面反映用户的偏好。相当于引入了额外的信息源，能够解决因显示评分行为较少导致的冷启动问题。其中一种主要信息源包括：用户 u 产生过行为(显示或隐式)的商品集合 $N(u)$, 可以引入用户 u 对于这些商品的隐式偏好 y_i 。

y_i 是隐藏的、对于商品 i 的个人喜好偏置（相当于每种产生行为的商品都有一个偏好 y_i ）。并且 y_i 是一个向量 (维度=商品数・隐因子个数)，每个分量代表对该商品的某一隐因子成分的偏好程度。 而用户 u 对这些隐因子的偏好程度 (implicit feedback) 实际上是将 所有产生行为 的商品对应的隐因子分量值进行分别求和，并除以一个规范化因子 $\sqrt{|N_u|}$，其中，引入 $\sqrt{|N_u|}$ 是为了消除不同 $|N(u)|$ 个数引起的差异。 即

$$\mathrm{ifb}_u = \frac{\sum_{i\in N(u)} y_i}{\sqrt{|N_u|}}$$

预测评分函数表示为

$$\begin{aligned} b_{uj} &= \mu + b_u + b_j \\ \hat{r}_{u,j} &= (p_u + \mathrm{ifb}_u) q_j^T + b_{uj} = (p_u + \frac{\sum_{i\in N(u)} y_i}{\sqrt{|N_u|}}) q_j^T + \mu + b_u + b_j \end{aligned}$$

从而优化目标函数 $J(p, q, b_u, b_j, y_i)$ 表示为:

$$\arg\min \sum_{r_{uj}\in R_{train}} \frac{1}{2}(r_{uj} - \hat{r}_{uj})^2 + \frac{\lambda}{2}(b_u^2 + b_j^2 + ||q_j||^2 + ||p_u||^2 + \sum_{i\in N(u)} ||y_i||^2)$$

将上式分别对 p_u、q_j、b_u、b_j、 $\underset{i\in N(u)_i}{y}$ 求导，得：

$$\begin{aligned} \frac{\partial J}{\partial p_u} &= -(r_{uj} - \hat{r}_{uj}) q_j + \lambda p_u \\ \frac{\partial J}{\partial q_j} &= -(r_{uj} - \hat{r}_{uj}) p_u + \lambda q_j \\ \frac{\partial J}{\partial b_u} &= -(r_{uj} - \hat{r}_{u,j}) + \lambda b_u \\ \frac{\partial J}{\partial b_j} &= -(r_{uj} - \hat{r}_{u,j}) + \lambda b_j \\ \frac{\partial J}{\partial y_i} &= -\frac{(r_{uj} - \hat{r}_{u,j}) q_j}{\sqrt{|N_u|}} + \lambda y_i \end{aligned}$$

在梯度下降法迭代时，p_u、q_j、b_u、b_j、 $\underset{i\in N(u)_i}{y}$ 的迭代公式为 (η：学习率):

$$
\begin{aligned}
p_u &= p_u + \eta[(r_{uj} - \hat{r}_{u,j})q_j - \lambda p_u] \\
q_j &= q_j + \eta[(r_{uj} - \hat{r}_{u,j})p_u - \lambda q_j] \\
b_u &= b_u + \eta[(r_{uj} - \hat{r}_{u,j}) - \lambda b_u] \\
b_j &= b_j + \eta[(r_{uj} - \hat{r}_{u,j}) - \lambda b_j] \\
y_i &= y_i + \frac{(r_{uj} - \hat{r}_{u,j})q_j}{\sqrt{|N_u|}} - \lambda y_i
\end{aligned}
$$

- 梯度下降

```
def sgd(self, u, j, y_true):
    '''
    梯度下降更新参数
    '''
    # 残差
    e_uj = y_true - self.predict(u, j)

    # 更新显示因子
    self.P[u] += self.learning_rate * (e_uj * self.Q[j] - self._lambda * self.P[u])
    self.Q[j] += self.learning_rate * (e_uj * self.P[u] - self._lambda * self.Q[j])

    # 更新偏置
    self.bu[u] += self.learning_rate * (e_uj - self._lambda * self.bu[u])
    self.bj[j] += self.learning_rate * (e_uj - self._lambda * self.bj[j])

    # 更新隐式因子
    ui = self.ui[u]
    ui_sqrt = np.sqrt(len(ui))
    self.yi[ui] = self.learning_rate * (e_uj * self.Q[j] / ui_sqrt - self._lambda * self.yi[ui])
    # for i in ui:
    #     self.yi[i] = self.learning_rate * (e_uj * self.Q[j] / ui_sqrt - self._lambda * self.yi[i
    self.u_implicit_fb[u] = np.sum(self.yi[ui], axis=0) / ui_sqrt
```

- 预测函数

```
def predict(self, u:int, j:int):
    '''
    预测u用户对物品i的评分
    '''
    rating = self.mu
    know_u = self._know_u(u)
    know_j = self._know_j(j)
    if know_u:
        rating += self.bu[u]
    if know_j:
        rating += self.bj[j]
    if know_u and know_j:
        rating += np.dot(self.P[u, :] + self.u_implicit_fb[u], self.Q[j, :])
    return rating
```

5. NMF

`非负矩阵分解` 是在上述基础上，加入了隐向量的非负限制。然后使用非负矩阵分解的优化算法求解。

$$
\begin{aligned}
p_{uf} &\leftarrow p_{uf} \cdot \frac{\sum_{j \in j_u} q_{jf} \cdot r_{uj}}{\sum_{j \in j_u} q_{jf} \cdot \hat{r_{uj}} + \lambda_u |j_u| p_{uf}} \\
q_{jf} &\leftarrow q_{jf} \cdot \frac{\sum_{u \in U_j} p_{uf} \cdot r_{uj}}{\sum_{u \in U_j} p_{uf} \cdot \hat{r_{uj}} + \lambda_j |U_j| q_{jf}}
\end{aligned}
$$

其中$\hat{r}_{uj}$, 既可以使用FunkSVD求法 $r\hat{}u_i = P_u Q_j^T$，也可以使用BiasSVD求法 $r\hat{}u_i = P_u Q_j^T + \mu + b_u + b_j$, 当然也可以改进成使用SVD++的求法。(*不知道为啥RMSE很不稳定且预测出现负值*)

- 训练部分

```python
def fit(self, train_set:DataSet):
    # 输入数据、参数初始化
    self.init_weights(train_set)

    # 开始训练
    epoch = 0
    while epoch < self.n_epochs and self.loss > self.epsilon:
        loss = 0
        user_num = np.zeros((train_set.n_users, self.n_factors))
        user_denom = np.zeros((train_set.n_users, self.n_factors))
        item_num = np.zeros((train_set.n_items, self.n_factors))
        item_denom = np.zeros((train_set.n_items, self.n_factors))

        for u, j, y_true in train_set.all_ratings():
            # 预测
            y_hat = self.predict(u, j)

            # 残差
            err = y_true - y_hat

            # 更新偏置
            self.bu[u] += self.learning_rate * (err - self._lambda * self.bu[u])
            self.bj[j] += self.learning_rate * (err - self._lambda * self.bj[j])

            # compute numerators and denominators
            user_num[u] += self.Q[j] * y_true
            user_denom[u] += self.Q[j] * y_hat
            item_num[j] += self.P[u] * y_true
            item_denom[j] += self.P[u] * y_hat

            loss += np.square(y_true - self.predict(u, j))

        # 更新用户矩阵
        for u in range(train_set.n_users):
            n_rating = len(train_set.ui[u])
            user_denom[u] += n_rating * self._lambda * self.P[u]
            self.P[u] *= user_num[u] / user_denom[u]

        # 更新物品矩阵
```

```
        for j in range(train_set.n_items):
            n_rating = len(train_set.iu[j])
            item_denom[j] += n_rating * self._lambda * self.Q[j]
            self.Q[j] *= item_num[j] / item_denom[j]

        epoch += 1
        self.loss = loss
        mse = self.loss/train_set.N
        print(f'Epoch {epoch}, loss={self.loss}, MSE={mse}, RMSE={np.sqrt(mse)}')
    print(f'Train Done, Q.shape={self.Q.shape}, P.shape={self.P.shape}')
```

- 预测

```
def predict(self, u:int, j:int):
    '''
    预测u用户对物品i的评分
    '''
    rating = self.mu
    know_u = self._know_u(u)
    know_j = self._know_j(j)

    if know_u:
        rating += self.bu[u]

    if know_j:
        rating += self.bj[j]

    if know_u and know_j:
        rating += np.dot(self.P[u, :], self.Q[j, :])

    return rating
```

6. 总结与参考链接

6.1 总结

算法	别名	内容
SVD	traditional SVD	奇异值分解
FunkSVD	LFM, basic MF, MF	LFM
regularized SVD	regularized MF	LFM+正则项
bias SVD	bias MF	LFM+正则项+偏置项
SVD++	*	LFM+正则项+偏置项+隐性反馈
NMF	*	对隐向量非负限制，可用在bias SVD等不同模型上

6.2 优缺点

- 优点
  a. 比较容易编程实现，随机梯度下降方法依次迭代即可训练出模型。

b. 预测的精度比较高，预测准确率要高于基于领域的协同过滤以及基于内容CBR等方法。

c. 比较低的时间和空间复杂度，高维矩阵映射为两个低维矩阵节省了存储空间，训练过程比较费时，但是可以离线完成；评分预测一般在线计算，直接使用离线训练得到的参数，可以实时推荐。

d. 非常好的扩展性，如由SVD拓展而来的SVD++和 TIME SVD++。

- 缺点：

a. 训练模型较为费时。

b. 推荐结果不具有很好的可解释性，无法用现实概念给分解出来的用户和物品矩阵的每个维度命名，只能理解为潜在语义空间。

6.3 引用链接

- [1] 奇异值分解(SVD)原理与在降维中的应用
- [2] Simon-Funk的博客
- [3] 推荐系统-矩阵分解技术
- [4] Surprise框架MF的实现
- [5] 基于矩阵分解的协同过滤
- [6] 推荐系统算法调研->理论讲解Nice
- [7] 推荐系统概述（一）