
# 隐语义模型复现

华展博　王知秋　张昊坡　张晓佩

计算机科学与技术学院
中国矿业大学

2023 年 9 月 21 日

# 目录

- 隐语义模型是最近几年推荐系统领域最为热门的研究话题，它的核心思想是通过隐含特征 (latent factor) 联系用户兴趣的物品
- 为了从数据角度出发准确的给物品分类然后进行个性化推荐，隐含语义分析技术采取基于用户行为统计的自动聚类，较好的解决了相关问题。
- 我们将以 LFM 为例，使用 Movielens ml-1m 数据集分析其在推荐系统的应用。

```python
# 导入包
import random
import math
import numpy as np
import time
from tqdm import tqdm, trange
```

Listing 1: Import the package

```
# 定义装饰器，监控运行时间
def timmer(func):
    def wrapper(*args, **kwargs):
        start_time = time.time()
        res = func(*args, **kwargs)
        stop_time = time.time()
        print('Func %s, run time: %s' % (func.__name__, stop_time -
            start_time))
        return res
    return wrapper
```

Listing 2: Generic function definitions

```
class Dataset():

    def ___init___(self, fp):
        # fp: data file path
        self.data = self.loadData(fp)

    @timmer
    def loadData(self, fp):
        data = []
        for l in open(fp):
            data.append(tuple(map(int, l.strip().split('::')[:2])))
        return data
```

Listing 3: LoadData

```
@timmer
    def splitData(self, M, k, seed=1):
        '''
        :params: data, 加载的所有(user, item)数据条目
        :params: M, 划分的数目，最后需要取M折的平均
        :params: k, 本次是第几次划分，k~[0, M)
        :params: seed, random的种子数，对于不同的k应设置成一样的
        :return: train, test
        '''
        train, test = [], []
        random.seed(seed)
        for user, item in self.data:
            # 这里与书中的不一致，修改为M-1，否则会出现list index out of range
            if random.randint(0, M-1) == k:
                test.append((user, item))
            else:
                train.append((user, item))
```

Listing 4: SplitData

```
# 处理成字典的形式，user->set(items)
        def convert_dict(data):
            data_dict = {}
            for user, item in data:
                if user not in data_dict:
                    data_dict[user] = set()
                data_dict[user].add(item)
            data_dict = {k: list(data_dict[k]) for k in data_dict}
            return data_dict

        return convert_dict(train), convert_dict(test)
```

Listing 5: Convert dict

```
class Metric():

    def ___init___(self, train, test, GetRecommendation): # 初始化
        ...
    def getRec(self): # 为test中的每个用户进行推荐
        ...
    def precision(self): # 定义精确率指标计算方式
        ...
    def recall(self): # 定义召回率指标计算方式
        ...
    def coverage(self): # 定义覆盖率指标计算方式
        ...
    def popularity(self): # 定义新颖度指标计算方式
        ...
    def eval(self): # 输出评价指标
        ...
```

Listing 6: Metric

```
def LFM(train, ratio, K, lr, step, lmbda, N):
    '''':params: train, 训练数据
    :params: ratio, 负采样的正负比例
    :params: K, 隐语义个数
    :params: lr, 初始学习率
    :params: step, 迭代次数
    :params: lmbda, 正则化系数
    :params: N, 推荐TopN物品的个数
    :return: GetRecommendation, 获取推荐结果的接口 '''
    all_items = {}
    for user in train:
        for item in train[user]:
            if item not in all_items:
                all_items[item] = 0
            all_items[item] += 1
    all_items = list(all_items.items())
    items = [x[0] for x in all_items]
    pops = [x[1] for x in all_items]
```

Listing 7: Metric

```
# 负采样函数(注意！！！要按照流行度进行采样)
    def nSample(data, ratio):
        new_data = {}
        # 正样本
        for user in data:
            if user not in new_data:
                new_data[user] = {}
            for item in data[user]:
                new_data[user][item] = 1
        # 负样本
        for user in new_data:
            seen = set(new_data[user])
            pos_num = len(seen)
            item = np.random.choice(items, int(pos_num * ratio * 3), pops)
            item = [x for x in item if x not in seen][:int(pos_num * ratio)]
            new_data[user].update({x: 0 for x in item})

        return new_data
```

Listing 8: LFM

```
# 训练
    P, Q = {}, {}
    for user in train:
        P[user] = np.random.random(K)
    for item in items:
        Q[item] = np.random.random(K)

    for s in trange(step):
        data = nSample(train, ratio)
        for user in data:
            for item in data[user]:
                eui = data[user][item] - (P[user] * Q[item]).sum()
                P[user] += lr * (Q[item] * eui - lmbda * P[user])
                Q[item] += lr * (P[user] * eui - lmbda * Q[item])
        lr *= 0.9 # 调整学习率
```

Listing 9: LFM

```
# 获取接口函数
    def GetRecommendation(user):
        seen_items = set(train[user])
        recs = {}
        for item in items:
            if item not in seen_items:
                recs[item] = (P[user] * Q[item]).sum()
        recs = list(sorted(recs.items(), key=lambda x: x[1], reverse=True))[:N]
        return recs

    return GetRecommendation
```

Listing 10: LFM

```
class Experiment():
    def ___init___(self, M, N, ratio=1,K=100, lr=0.02, step=100, lmbda=0.01, fp='
        ../dataset/ml-1m/ratings.dat'):
        '''':params: M, 进行多少次实验 :params: N, TopN推荐物品的个数
        :params: ratio, 正负样本比例 :params: K, 隐语义个数
        :params: lr, 学习率 :params: step, 训练步数
        :params: lmbda, 正则化系数 :params: fp, 数据文件路径'''
        self.M = M
        self.K = K
        self.N = N
        self.ratio = ratio
        self.lr = lr
        self.step = step
        self.lmbda = lmbda
        self.fp = fp
        self.alg = LFM
```

Listing 11: Experiment

```
# 定义单次实验
    @timmer
    def worker(self, train, test):
        '''
        :params: train, 训练数据集
        :params: test, 测试数据集
        :return: 各指标的值
        '''
        getRecommendation = self.alg(train, self.ratio, self.K,
                                     self.lr, self.step, self.lmbda, self.N)
        metric = Metric(train, test, getRecommendation)
        return metric.eval()
```

Listing 12: Experiment

```
@timmer # 多次实验取平均
def run(self):
    metrics = {'Precision': 0, 'Recall': 0,
               'Coverage': 0, 'Popularity': 0}
    dataset = Dataset(self.fp)
    for ii in range(self.M):
        train, test = dataset.splitData(self.M, ii)
        print('Experiment {}:'.format(ii))
        metric = self.worker(train, test)
        metrics = {k: metrics[k]+metric[k] for k in metrics}
    metrics = {k: metrics[k] / self.M for k in metrics}
    print('Average Result (M={}, N={}, ratio={}): {}'.format(\
                        self.M, self.N, self.ratio, metrics))
# LFM实验
M, N = 8, 10
for r in [1, 2, 3, 5, 10, 20]:
    exp = Experiment(M, N, ratio=r)
    exp.run()
```

Listing 13: Experiment

表: 实验结果（比例 =1）

| 实验编号 | 精度 | 召回率 | 覆盖率 | 流行度 |
|---|---|---|---|---|
| 0 | 19.15% | 9.2% | 88.35% | 6.3121 |
| 1 | 19.01% | 9.1% | 87.85% | 6.3099 |
| 2 | 19.06% | 9.11% | 87.71% | 6.3076 |
| 3 | 18.92% | 9.09% | 88.81% | 6.3017 |
| 4 | 18.9% | 9.09% | 88.3% | 6.3162 |
| 5 | 18.96% | 9.15% | 88.24% | 6.3114 |
| 6 | 18.7% | 8.99% | 88.78% | 6.3055 |
| 7 | 18.69% | 8.96% | 88.3% | 6.2934 |
| 平均结果 (M=8, N=10, 比例 =1) | | | | |
| ratio=1 | 18.92% | 9.09% | 88.29% | 6.3072 |

表: 实验结果（比例 =2）

| 实验编号 | 精度 | 召回率 | 覆盖率 | 流行度 |
|---|---|---|---|---|
| 0 | 22.95% | 11.02% | 40.81% | 6.9484 |
| 1 | 23.09% | 11.06% | 40.81% | 6.9389 |
| 2 | 23.24% | 11.11% | 41.06% | 6.9401 |
| 3 | 22.86% | 10.99% | 40.57% | 6.9447 |
| 4 | 22.77% | 10.95% | 41.02% | 6.9419 |
| 5 | 22.82% | 11.00% | 40.4% | 6.9468 |
| 6 | 22.78% | 10.95% | 40.81% | 6.9465 |
| 7 | 22.80% | 10.93% | 41.18% | 6.9406 |
| 平均结果 (M=8, N=10, 比例 =2) | | | | |
| ratio=2 | 22.91% | 11.00% | 40.83% | 6.9435 |

- 随着负样本数目的增加，LFM 的准确率和召回率有明显提高。
- 覆盖率相对较高，说明数据集中的大部分物品都得到了推荐。
- 流行度指标表明推荐的物品不会过于热门，这对于推荐多样性来说是一个积极的迹象。
- 当 ratio>10 以后，准确率和召回率基本稳定在 25% 和 13% 。

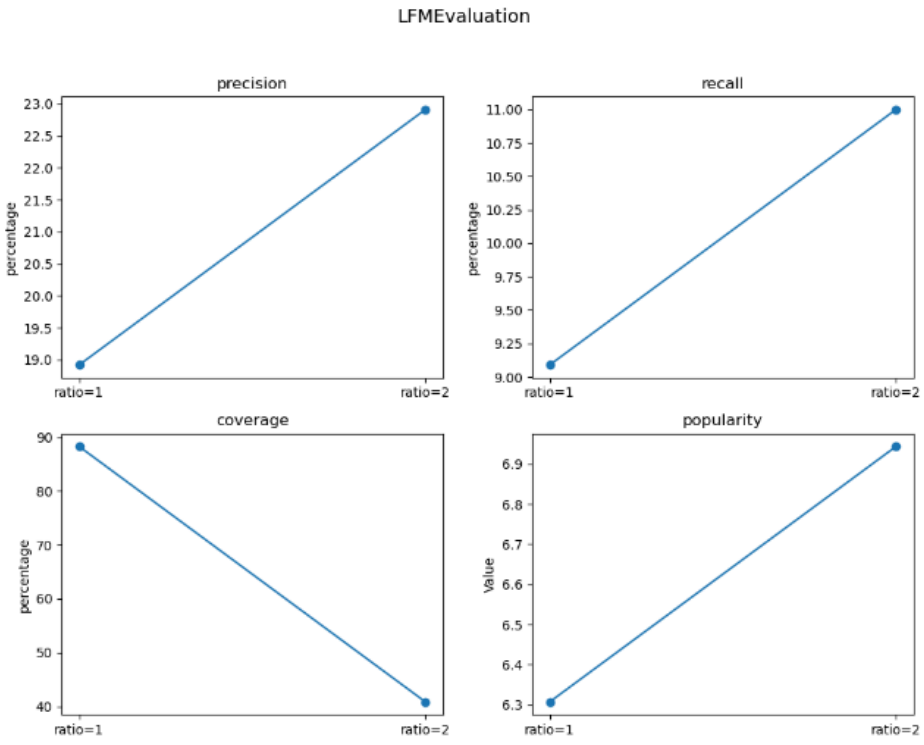


图: LFM Evaluation

# 算法优化

## 加入偏置 (Bias) 部分

- 我们观测到的评分数据大部分都是都是和用户或物品无关的因素产生的效果，即有很大一部分因素是和用户对物品的喜好无关而只取决于用户或物品本身特性的。
- 这些独立于用户或独立于物品的因素称之为偏置 (Bias) 部分。
- 在矩阵分解模型中偏置部分对提高评分预测准确率起的作用要大大高于个性化部分所起的作用。

# 算法优化

## 加入偏置 (Bias) 部分

**目标函数**

偏置部分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 $\mu$，表示了训练数据的总体评分情况，对于固定的数据集，它是一个常数。
- 用户偏置 $\mathrm{b_i}$: 独立于物品特征的因素，表示某一特定用户打分习惯。
- 物品偏置 $\mathrm{b_j}$: 特立于用户兴趣的因素，表示某一特定物品得到的打分情况。

**则偏置部分表示为**:

$$\mathrm{b_{ij}} = \mu + \mathrm{b_i} + \mathrm{b_j}$$

**加入偏置项的优化函数 $\mathrm{J(p,q)}$ 为**:

$$\arg\min_{\mathrm{p_j q_j}} \sum_{\mathrm{i,j}\in\mathrm{K}} (\mathrm{m_{ij}} - \mu - \mathrm{b_i} - \mathrm{b_j} - \mathrm{q_j^T p_i})^2 + \lambda(\|\mathrm{p_i}\|_2^2 + \|\mathrm{q_j}\|_2^2 + \|\mathrm{b_i}\|_2^2 + \|\mathrm{b_j}\|_2^2)$$

同理，这个优化目标也可以用梯度下降法求解，方法相同

**隐式反馈**

用户除了对商品有评分的显式反馈之外还有诸如浏览、点击等隐式反馈：

- 一个用户可能对许多商品有隐式反馈，我们将用户 $\mathrm{i}$ 有过隐式反馈的商品集合记为 $\mathrm{N(i)}$。
- 每一次对于特定商品 $\mathrm{s} \in \mathrm{N(i)}$ 的点击或浏览，都带来对于用户特征 $\mathrm{p_i}$ 的某些偏置 $\mathrm{y_s}$。

**对于用户 $\mathrm{i}$ 的特征 $\mathrm{p_i}$ 可以写为：**

$$\mathrm{p_i} + \sum_{\mathrm{s} \in \mathrm{N(i)}} \mathrm{y_s}$$

**用户 $\mathrm{i}$ 对商品 $\mathrm{j}$ 的评分可以写为：**

$$\mu + \mathrm{b_i} + \mathrm{b_j} + \mathrm{q_j^T}(\mathrm{p_i} + \sum_{\mathrm{s} \in \mathrm{N(i)}} \mathrm{y_s}$$

同理，目标函数同样方法实现

| 方面 | LFM | 基于邻域的方法 |
|---|---|---|
| 模型类型 | 矩阵分解 | 基于用户或物品的方法 |
| 潜在因子 | 利用用户和物品的潜在因子 | 不使用潜在因子 |
| 个性化推荐 | 提供个性化推荐 | 可能不太个性化 |
| 冷启动问题 | 能够处理冷启动问题,利用潜在因子 | 在冷启动情况下可能表现不佳 |
| 可扩展性 | 利用优化技术可扩展到大型数据集 | 针对大型数据集可能计算开销较大 |
| 预测质量 | 通常提供更好的预测质量 | 预测质量可能有所不同 |
| 可解释性 | 由于使用潜在因子,较不可解释 | 基于相似性度量更具可解释性 |
| 数据需求 | 需要用户-物品交互数据 | 需要用户-物品交互数据 |

表: LFM 与基于邻域的方法比较

# 总结

在本研究中，我们对隐语义模型（LFM）在推荐系统中的应用进行了详细的探讨，并将其与基于邻域的方法进行了比较。以下是我们的主要发现：

- LFM 是一种利用潜在因子的矩阵分解模型，能够提供个性化推荐。与基于邻域的方法相比，它通常在预测质量上表现更好。
- LFM 能够较好地处理冷启动问题，通过潜在因子进行推荐，即使对于新用户或新物品，也可以提供有意义的建议。
- 基于邻域的方法具有较高的可解释性，因为它们依赖于用户或物品之间的相似性度量。然而，在个性化推荐方面，LFM 通常更具优势。

我们进行了一系列实验来评估 LFM 的性能，结果显示，在不同的比例下，LFM 的准确率和召回率都有所提高，而覆盖率相对较高，同时避免了过于热门的推荐。

总的来说，LFM 在推荐系统中是一种有效的模型，尤其在个性化推荐方面表现出色。

[1] 项亮编著；陈义，王益审校 (2012)。《推荐系统实践》。北京：人民邮电出版社。

[2] 周超，孙英华，熊化峰，刘雪庆 (2018)。基于用户和项目双向聚类的协同过滤推荐算法。《青岛大学学报 (自然科学版)》，第 1 期，55-60。