

智能推荐系统第三次编程作业

——Latent Factor Model for Rating Prediction

学号：10185102142

姓名：李泽浩

指导老师：张伟

项目名称：Latent Factor Model for Rating Prediction

时间：2021年5月17日

目录

[1.隐语义模型LFM](#)

1.1基础算法

1.2优化

1.3伪代码

1.4缺点

1.5LFM和基于邻域的方法的比较

[2.代码](#)

2.1 通用定义

2.2 LFM实现

2.3 LFM 实验

[3.结果分析及准确率](#)

[4.问题总结](#)

[5.提交文件](#)

5.1实验报告PDF

5.2Python代码 `source_code.ipynb`

5.3预测补充后的`test.csv`文件与上次预测

1、隐语义模型LFM简介

自从Netflix Prize比赛举办以来，LFM(latent factor model)隐语义模型逐渐成为推荐系统领域耳熟能详的名词。其实该算法最早在文本挖掘领域被提出，用于找到文本的隐含语义。相关的名词有LSI、pLSA、LDA和Topic Model。

隐语义模型LFM和LSI，LDA，Topic Model其实都属于隐含语义分析技术，是一类概念，他们在本质上是相通的，都是找出潜在的主题或分类。这些技术一开始都是在文本挖掘领域中提出来的，近些年它们也被不断应用到其他领域中，并得到了不错的应用效果。比如，在推荐系统中它能够基于用户的行为对item进行自动聚类，也就是把item划分到不同类别/主题，这些主题/类别可以理解为用户的兴趣。

1.1基础代码

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)联系用户兴趣和物品。

LFM通过如下公式计算用户u对物品i的兴趣：

$$\text{Preferencr}(u,i) = r_{ui} = P_u^T Q_i = \sum_{k=1}^F P_{u,k} Q_{i,k}$$

R矩阵是user-item矩阵，矩阵值 R_{ij} 表示的是user i 对item j的兴趣度，这正是我们要求的值。对于一个user来说，当计算出他对所有item的兴趣度后，就可以进行排序并作出推荐。LFM算法从数据集中抽取出若干主题，作为user和item之间连接的桥梁，将R矩阵表示为P矩阵和Q矩阵相乘。其中P矩阵是user-class矩阵，矩阵值 P_{ij} 表示的是user i对class j的兴趣度；Q矩阵是class-item矩阵，矩阵值 Q_{ij} 表示的是item j在class i中的权重，权重越高越能作为该类的代表。所以LFM根据上述公式来计算用户U对物品I的兴趣度，这个公式中 $p_{u,k}$ 和 $q_{i,k}$ 是模型的参数，其中 $P_{u,k}$ 度量了用户u的兴趣和第k个隐类的关系，而 $q_{i,k}$ 度量了第k个隐类和物品i之间的关系。

发现使用LFM后，

- 不需要关心分类的角度，结果都是基于用户行为统计自动聚类的，全凭数据自己说了算。
- 不需要关心分类粒度的问题，通过设置LFM的最终分类数就可控制粒度，分类数越大，粒度约细。
- 对于一个item，并不是明确的划分到某一类，而是计算其属于每一类的概率，是一种标准的软分类。
- 对于一个user，我们可以得到他对于每一类的兴趣度，而不是只关心可见列表中的那几个类。
- 对于每一个class，我们可以得到类中每个item的权重，越能代表这个类的item，权重越高。

接下去的问题就是如何计算矩阵P和矩阵Q中参数值。一般做法就是最优化损失函数来求参数。在定义损失函数之前，我们需要准备一下数据集并对兴趣度的取值做一说明。

数据集应该包含所有的user和他们有过行为的（也就是喜欢）的item。所有的这些item构成了一个item全集。对于每个user来说，我们把他有过行为的item称为正样本，规定兴趣度 $R_{UI}=1$ ，此外我们还需要从item全集中随机抽样，选取与正样本数量相当的样本作为负样本，规定兴趣度为 $R_{UI}=0$ 。因此，兴趣的取值范围为[0,1]。

采样之后原有的数据集得到扩充，得到一个新的user-item集 $K=\{(U,I)\}$ ，其中如果 (U,I) 是正样本，则 $R_{UI}=1$ ，否则

$R_{UI}=0$ 。损失函数如下所示：

$$C = \sum_{(U,I) \in K} (R_{UI} - \hat{R}_{UI})^2 = \sum_{(U,I) \in K} (R_{UI} - \sum_{k=1}^K P_{U,k} Q_{k,I})^2 + \lambda \|P_U\|^2 + \lambda \|Q_I\|^2$$

式子中最后一项是用来防止过拟合的正则化项， λ 需要根据具体应用场景反复实验得到。

1.2 优化

· 通过求参数PUK和QKI的偏导确定最快的下降方向

$$\frac{\partial C}{\partial P_{Uk}} = -2(R_{UI} - \sum_{k=1}^K P_{U,k} Q_{k,I}) Q_{kI} + 2\lambda P_{Uk}$$

$$\frac{\partial C}{\partial Q_{kI}} = -2(R_{UI} - \sum_{k=1}^K P_{U,k} Q_{k,I}) P_{Uk} + 2\lambda Q_{kI}$$

· 迭代计算不断优化参数（迭代次数事先设置），直到参数收敛

$$P_{Uk} = P_{Uk} + \alpha((R_{UI} - \sum_{k=1}^K P_{U,k} Q_{k,I}) Q_{kI} - \lambda P_{Uk})$$

$$Q_{kI} = Q_{kI} + \alpha((R_{UI} - \sum_{k=1}^K P_{U,k} Q_{k,I}) P_{Uk} - \lambda Q_{kI})$$

其中，α是学习速率，α越大，迭代下降的越快。α和λ一样，也需要根据实际的应用场景反复实验得到。本书中，作者在*[MovieLens](#)*数据集上进行实验，他取分类数F=100，α=0.02，λ=0.01。

综上所述，执行LFM需要：

1. 根据数据集初始化P和Q矩阵（这是我暂时没有弄懂的地方，这个初始化过程到底是怎么样进行的，还恳请各位童鞋予以赐教。）
2. 确定4个参数：分类数F，迭代次数N，学习速率α，正则化参数λ。

1.3 伪代码实现

```
def LFM(user_items, F, N, alpha, lambda):  
    #初始化P,Q矩阵  
    [P, Q] = InitModel(user_items, F)  
    #开始迭代  
    For step in range(0, N):  
        #从数据集中依次取出user以及该user喜欢的items集  
        for user, items in user_item.items():  
            #随机抽样，为user抽取与items数量相当的负样本，并将正负样本合并，用于优化计算  
            samples = RandSelectNegativeSamples(items)  
            #依次获取item和user对该item的兴趣度  
            for item, rui in samples.items():  
                #根据当前参数计算误差  
                eui = eui - Predict(user, item)  
                #优化参数  
                for f in range(0, F):
```

```
P[user][f] += alpha * (eui * Q[f][item] - lambda * P[user][f])
Q[f][item] += alpha * (eui * P[user][f] - lambda * Q[f][item])
#每次迭代完后，都要降低学习速率。一开始的时候由于离最优值相差甚远，因此快速下降；
#当优化到一定程度后，就需要放慢学习速率，慢慢的接近最优值。
alpha *= 0.9
```

1.4缺点

1. 数据稀疏会导致性能降低。甚至不如UserCF和 ItemCF的性能。
2. 不能在线实时推荐。因为LFM在生成推荐列表时速度太慢。
3. 不能给出推荐解释。因为LFM计算的是各个隐类，但每个类别不会自动生成这个类的类别标签。

1.5方法比较

1.理论基础：

基于邻域的方法更多是一种基于统计的方法，并没有学习过程。

LFM是一种基于机器学习的方法，具有比较好的理论基础，通过优化一个设定的指标建立最优的模型。

2.离线计算的空间复杂度：

基于邻域的方法需要维护一张离线的相关表，在离线计算相关表的过程中，如果用户/物品数很多，将会占据很大的内存。假设有M个用户和N个物品，如果是用户相关表，则需要 $O(MM)$ 的空间；如果是物品相关表，则需要 $O(NN)$ 的空间。

LFM在建模过程中，如果是F个隐类，那么它需要的存储空间是 $O(F*(M+N))$ ，这在M和N很大时可以很好地节省离线计算的内存。

3.离散计算的时间复杂度：

一般情况下，LFM的时间复杂度要稍微高于UserCF和ItemCF，主要是因为该算法需要多次迭代，总体上两种算法在时间复杂度上没有质的差别。

4.在线实时推荐：

基于邻域的方法需要将相关表缓存在内存中，然后可以在线进行实时的预测。

LFM在给用户生成推荐列表时，需要计算用户对所有物品的兴趣权重，然后排名，返回权重最大的N个物品，时间复杂度比较高，因此不能在线实时推荐。

5.推荐解释：

ItemCF算法支持很好的推荐解释，它可以利用用户的历史行为解释推荐结果。

LFM计算出的隐类虽然在语义上确实代表了一类兴趣和物品，却很难用自然语言描述并生成解释展现给用户。

2、代码详解

导入必要的函数库

```
#导入必要的函数库
import numpy as np
import random
import pandas as pd
import math
from sklearn.model_selection import train_test_split
from sklearn import datasets
from collections import defaultdict
import time
from tqdm import tqdm, trange
```

2.1通用函数定义

数据处理

```
# 定义装饰器，监控运行时间
def timer(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
```

数据处理 (load data ; split data)

```
class Dataset():

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

    @timer
    def loadData(self, fp):
        data = []
        for l in open(fp):
            data.append(tuple(map(int, l.strip().split('::')[1:2])))
        return data

    @timer
    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:
    if random.randint(0, M-1) == k:
        test.append((user, item))
    else:
        train.append((user, item))

# 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)

```

评价指标

```

class Metric():

    def __init__(self, train, test, GetRecommendation):
        '''
        :params: train, 训练数据
        :params: test, 测试数据
        :params: GetRecommendation, 为某个用户获取推荐物品的接口函数
        '''

        self.train = train
        self.test = test
        self.GetRecommendation = GetRecommendation
        self.recs = self.getRec()

    # 为test中的每个用户进行推荐
    def getRec(self):
        recs = {}
        for user in self.test:
            rank = self.GetRecommendation(user)
            recs[user] = rank
        return recs

    # 定义精确率指标计算方式

```

```

def precision(self):
    all, hit = 0, 0
    for user in self.test:
        test_items = set(self.test[user])
        rank = self.recs[user]
        for item, score in rank:
            if item in test_items:
                hit += 1
        all += len(rank)
    return round(hit / all * 100, 2)

```

定义召回率指标计算方式

```

def recall(self):
    all, hit = 0, 0
    for user in self.test:
        test_items = set(self.test[user])
        rank = self.recs[user]
        for item, score in rank:
            if item in test_items:
                hit += 1
        all += len(test_items)
    return round(hit / all * 100, 2)

```

定义覆盖率指标计算方式

```

def coverage(self):
    all_item, recom_item = set(), set()
    for user in self.test:
        for item in self.train[user]:
            all_item.add(item)
        rank = self.recs[user]
        for item, score in rank:
            recom_item.add(item)
    return round(len(recom_item) / len(all_item) * 100, 2)

```

定义新颖度指标计算方式

```

def popularity(self):
    # 计算物品的流行度
    item_pop = {}
    for user in self.train:
        for item in self.train[user]:
            if item not in item_pop:
                item_pop[item] = 0
            item_pop[item] += 1

    num, pop = 0, 0
    for user in self.test:
        rank = self.recs[user]
        for item, score in rank:
            # 取对数，防止因长尾问题带来的被流行物品所主导

```



```

        pop += math.log(1 + item_pop[item])
        num += 1
    return round(pop / num, 6)

def eval(self):
    metric = {'Precision': self.precision(),
              'Recall': self.recall(),
              'Coverage': self.coverage(),
              'Popularity': self.popularity()}
    print('Metric:', metric)
    return metric

```

2.2 LFM 算法实现

```

def LFM(train, ratio, K, lr, step, lambda, 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: lambda, 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]

    # 负采样函数
    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

# 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 - lambda * P[user])
            Q[item] += lr * (P[user] * eui - lambda * Q[item])
    lr *= 0.9 # 调整学习率

# 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

```

2.3 LFM实验

```

class Experiment():

    def __init__(self, M, N, ratio=1,
                 K=100, lr=0.02, step=100, lambda=0.01, fp='test.csv'):
        ...

        :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: lambda, 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

# 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()

# 多次实验取平均
@timmer
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()

```

3、结果分析及准确率

计算均方误差

```
#均方误差函数
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions, labels)]
    return sum(differences) / len(differences)

# 实际打分平均值
alwaysPredictMean = [ratingMean for d in dataset]

#预测打分
cfPredictions = [prdictRating(d['user_id'], d['business_id']) for d in dataset]

labels = [d['stars'] for d in dataset]

MSE(alwaysPredictMean, labels)

MSE(cfPredictions, labels)
```

结果如下：

1	MSE(alwaysPredictMean, labels)
executed in 7ms, finished 20:10:52 2021-05-19	
1.0033436473005906	

1	MSE(cfPredictions, labels)
executed in 6ms, finished 20:10:52 2021-05-19	
1.0792256987993392	

4、问题总结

5、提交文件列表

实验报告lab3-10185102142-李泽浩

源代码source_code.ipynb

预测结果后的test.csv