推荐系统实验报告

小组成员：孙家宜，李彦欣，郝旭

一、实验相关统计信息

用户数量: 19835

物品数量: 624961

打分数量: 5002419

打分平均值: 49.65618273879098

二、实验原理

1．基本实验思路

(2)处理数据：根据 train.txt 中的数据得到[userid, itemid, rating]形式的 list，构建出 user 对 item 打分的矩阵。

(1) 由于以上矩阵是稀疏的，因此定义聚类对未打分项进行填充，降低矩阵的稀疏程度。具体做法是：选取 itemAtribute.txt 中所给的每个 item 的属性值对 item 聚类。

(3)开始训练：将数据集分为 trainset 和 testset，然后用 trainset 进行训练，用 testset 评估算法的准确度，不断调整参数，提升算法的准确度。

(4)最后用模型预测 test.txt 中数据的得分并按照要求存储到 result.txt 中，再算出 RMSE。

2.主要算法

本小组使用潜在因子模型 SVD（奇异值分解）的方法构建推荐系统。

- **MiniBatchKmeans 聚类**

此算法主要为了解决矩阵稀疏问题。

在统的 K-Means 算法中，要计算所有的样本点到所有的质心的距离。如果样本量非常大，比如达到 10 万以上，特征有 100 以上，此时用传统的 K-Means 算法非常的耗时，就算加上 elkan K-Means 优化也依旧。在大数据时代，这样的场景越来越多。此时 Mini Batch K-Means 应运而生。

顾名思义，Mini Batch，也就是用样本集中的一部分的样本来做传统的 K-Means，这样可以避免样本量太大时的计算难题，算法收敛速度大大加快。当然此时的代价就是我们的聚类的精确度也会有一些降低。一般来说这个降低的幅度在可以接受的范围之内。

在 Mini Batch K-Means 中,我们会选择一个合适的批样本大小 batch size，我们仅仅用 batch size 个样本来做 K-Means 聚类。batch size 个样本一般是通过无放回的随机采样得到的。

为了增加算法的准确性，我们一般会多跑几次 Mini Batch K-Means 算法，用得到不同的随机采样集来得到聚类簇，选择其中最优的聚类簇。

- **SVD 算法**

我们设 user 给 item 的评分的矩阵为 A，SVD 算法就是将矩阵 A 分解为成矩阵 U 和 V，其中 A,U,V 满足：

$$A = U\Sigma V^T$$

A：输入的矩阵

U：左奇异矩阵

V：右奇异矩阵

Σ：奇异值向量

上述算法成立是在 A 中所有值都已知的情况下，但是实际上打分矩阵中有大量未知值。所以我们构建另一种分解。

$$R = QP^T$$

这里

$$A = R, U = Q, \Sigma V^T = P^T$$

利用基于求函数最小值问题的方法：求出所有满足如下条件的向量

p_u（组成P^T的列）

q_i（组成Q的行）

对所有 u，i，$\hat{r}_{ui} = q_i \cdot p_u$

使优化函数$\min_{P,Q} \sum_{(i,u)\in R}(r_{ui} - q_i \cdot p_u)^2$最小

这样我们就可以尽可能使r_{ui}与$q_i \cdot p_u$接近。得到优化函数最小时的q_i, p_u，就可以得到 P,Q。

3.求解过程

- **梯度下降找近似解**

梯度下降是一种查找函数最小值的方法。这里我们使用随机梯度下降的方法，所以首先要构造损失函数：

$$\min_{P,Q} \sum_{training} (r_{ui} - q_i \cdot p_u)^2 + [\lambda_1 \sum_u ||p_u||^2 + \lambda_2 \sum_i ||q_i||^2]$$

这里公式的后半部分为正则项，防止过拟合。接下来求出函数关于 p,q 的导数，就可以应用梯度下降。

$$P \leftarrow P - \eta \cdot \nabla P$$
$$Q \leftarrow Q - \eta \cdot \nabla Q$$

$$\nabla Q = [\nabla q_{if}] \; and \; \nabla q_{if} = \sum_{u,i} -2(r_{ui} - q_i p_u)p_{uf} + 2\lambda_2 q_{if}$$

- **预测评分**

对未知 u,i 对的分数进行预测

公式：

$$r_{ui} = \mu + b_u + b_i + q_i \cdot p_u$$

μ：所有打分的平均值

b_u：用户 u 的偏好

b_i：物品 i 的偏好

新的损失函数：

$$\min_{P,Q} \sum_{(u,i)\in R} (r_{ui} - (\mu + b_u + b_i + q_i \cdot p_u))^2 + (\lambda_1 \sum_u ||p_u||^2 + \lambda_2 \sum_i ||q_i||^2 + \lambda_3 \sum_u ||b_u||^2 + \lambda_4 \sum_i ||b_i||^2)$$

- **评估模型**

这里我们使用 RMSE（均方根误差）

$$RMSE = \sqrt{\frac{\sum_{ui}(r_{ui} - \hat{r}_{ui})^2}{N}}$$

三、关键部分代码解析

（1）普通函数：

myfind 函数：用来查找一个 list 里某个值的索引

【关键】cluster 函数：（MiniBatchKmeans 聚类算法）根据 itemAttribute.txt 进行聚类，返回 itemid 的 list 和聚类得到的每个点的分类 label，代码如下

```python
def cluster(filename):

    F=150
    batch_size = 50000
    # 加载itemAttribute.txt中的数据
    itemid=[]
    attr1=[]
    attr2=[]
    f = open(filename,encoding='UTF-8')
    line = f.readline()
    # 读取文件每行内容并进行切片，添加到相应的list
    while line:
        line=line.replace('None','0')
        item_id,item_x,item_y=line.split('|')
        itemid.append(int(item_id))
        attr1.append(int(item_x))
        attr2.append(int(item_y.split('\n')[0]))
        line = f.readline()
    f.close()
    X=[attr1,attr2]
    X=np.transpose(X)
    # 使用MiniBatchKmeans算法进行聚类
    KM=MiniBatchKMeans(init='k-means++', n_clusters=F, batch_size=batch_size, n_init=10, max_no_improvement=10, verbose=0)
    # y_pred和centers分别是聚类得到的每个点的分类label和每个类的中心点坐标
    y_pred = KM.fit_predict(X)

    return  itemid, y_pred
```

LoadTrainset 函数：加载训练集，主要两部分

1.根据 train.txt 生成元素为[userid, itemid, rate/10]的 rating_list

2.根据聚类结果，对评分数<15 的 user 进行打分填充

最后返回 rating_list，user 总数，item 总数

LoadTestset 函数：加载测试集，根据 test.txt 生成元素为[userid，itemid]的 test_list，返回 test_list

（2）Trainset 类中的函数

get_all_ratings 函数：返回所有[userid, itemid, rate] list

get_user_ratings 函数：返回某个 user 的所有[userid, itemid] list

global_mean 函数：返回训练集所有打分平均值

construct_trainset 函数：从 train.txt 中加载数据并创建训练集，返回一个 Trainset 的对象。

（3）SVD 类中的函数

InerProduct 函数：返回两个矩阵的内积。

【关键】train 函数：SVD 算法，运用梯度下降进行训练，得到训练好的一系列参数值和矩阵，代码如下

```
def train(self, trainset):
        # 随机初始化user和item的factors
        self.trainset = trainset

        global_mean = self.trainset.global_mean
        bu = np.zeros(self.trainset.n_users, np.double)
        bi = np.zeros(self.trainset.n_items, np.double)
        randstate = np.random
        pu = randstate.normal(self.init_mean, self.init_std_dev, (self.trainset.n_users, self.n_factors))
        qi = randstate.normal(self.init_mean, self.init_std_dev, (self.trainset.n_items, self.n_factors))


        # 迭代
        for cur_epoch in range(self.n_epochs):
            # 更新学习率
            learningRate = self.learningRate * 0.95
            print('第 %d 次训练!' % (cur_epoch + 1))
            for uid, iid, rate in self.trainset.get_all_ratings():
                # 求内积
                multi = self.InerProduct(qi, pu, uid, iid, self.n_factors)
                predict = 0

                predict = global_mean + bu[uid] + bi[iid] + multi
                # 不能超出分数的范围0-100
                # predict = max(predict, 0)
                # predict = min(predict, 10)
                err = rate - predict

                # 更新参数bu,bi
                bu[uid] += learningRate * (err - self.regularization * bu[uid])
                bi[iid] += learningRate * (err - self.regularization * bi[iid])

                # 更新参数pu,qi
                for f in range(self.n_factors):
                    temp = pu[uid, f]
                    pu[uid, f] += learningRate * (err * qi[iid, f] - self.regularization * temp)
                    qi[iid, f] += learningRate * (err * temp - self.regularization * qi[iid, f])

        self.bu = bu
        self.bi = bi
        self.pu = pu
        self.qi = qi
        return
```

【关键】predict 函数：对 test.txt 中未知的某对 u,i 对的打分进行预测（由于前面存储的已知打分值[rate/10]，所以这里返回值要乘 10）返回打分值，代码如下

```
def predict(self, ruid, riid):
        ruid=int(ruid)
        riid=int(riid)
        grade = self.trainset.global_mean + self.bu[ruid] + self.bi[riid] + np.dot(self.qi[riid], self.pu[ruid])
        # 不能超出分数的范围0-100
        grade = max(grade, 0)
        grade = min(grade, 10)
        return round(grade)*10
```

predict_all 函数：对 test.txt 中未知的所有 u,i 对进行打分预测，返回所有分值的 list

split_train_and_test 函数：根据 train.txt 分割出新的训练集和测试集，返回训练集 trainset 和测试集 testset。

RMSE 函数：计算均方根误差的函数

【关键】evaluation 函数：计算训练出的模型的均方根误差，返回均方根误差的值，代码如下

```
def evaluation():
    # 加载数据集并进行分割，切分为训练集和测试集
    dataset,n_users,n_items = LoadTrainset('./Data/train.txt')
    trainset, testset = split_train_and_test(dataset, 10, 1, 1)
    dataset = construct_trainset(dataset,n_users,n_items)
    # 创建训练集并进行训练
    trainset = construct_trainset(trainset,n_users,n_items)
    svd = SVD()

    svd.train(trainset)
    targets = [grade*10 for (_, _, grade) in testset]
    items = [[userid, itemid] for (userid, itemid, grade) in testset]
    ur = dataset.raw_users_id
    ir = dataset.raw_items_id
    predictions = svd.predict_all(items, ur, ir)
    # 计算RMSE
    grade = RMSE(predictions, targets)
    print('RMSE:',grade)
```

四、结果分析

我们将数据集随机分成训练集和测试集（训练集 9 成测试集 1 成），测试集上 RMSE 结果如下：

```
begin predicting...
100%|██████████| 454887/454887 [00:02<00:00, 174031.87it/s]
RMSE: 32.03552406803331
```

最后作为预测结果的 result 文档结构如下:

<user id>|<numbers of rating items>

<item id>　　<score>

结果预览：（详情见 result.txt）

```
0|6
550452  80.0
323933  80.0
159248  60.0
554099  80.0
70896  80.0
518385  50.0
```

测试环境：

Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz

内存　　8.0 GB

时间消耗：343s

空间消耗：2g

五．分析算法的改进与不足

初步测试中，由于参数设定的太大，消耗时间太长。

降低迭代次数，将评分范围缩小至【0,10】，同时引入 k-means 聚类，得以将时间缩短至 6min，rmse 也减小到 32。

考虑使用 tensorflow 来加速实现，但由于未知原因内存使用量太高而放弃。