Web 实验报告 Lab3

金哲欣 PB17111663

许世晨 PB17030846

本次实验对豆瓣评分进行了预测。实验中使用了SVD，SVD++及爬虫等方式。由于数据规模过大，并且硬件算力有限，我们放弃了使用深度学习的方法对时间序列和社交网络的应用。综合数据集的大小，属性等因素，SVD是非常适合的模型。同时，为了获得更精确的结果，也使用了SVD++。除此之外，注意到评分可以通过豆瓣网站获得，故也尝试了使用爬虫获取评分数据。为了快速，准确并不被反爬机制拦截地获取所需要的数据，本次实验也在爬虫技术上做了不同的尝试和努力。

0. 分工

姓名	分工
金哲欣	SVD、KNN、Co-Clustering、爬虫
许世晨	SVD++、NMF、SlopeOne、Random、爬虫

1. SVD及SVD++

1.1 理论基础

SVD，即奇异值分解，是在机器学习领域广泛应用的算法，它不光可以用于降维算法中的特征分解，以及自然语言处理等领域，还可以用于推荐系统。事实上，在深度推荐算法流行之前，SVD及其变种一直是最受工业界青睐的推荐算法模型之一。

SVD的思想在推荐系统中的应用极为巧妙。我们把评分矩阵计作V， $V \in R^{n \times m}$，那么V的每一行Vi代表一个人的所有评分，每一列Vj代表某一部电影所有人的评分，Vij代表某个人i对某部电影j的评分。对应电影推荐来说，V必定是稀疏的，因为电影数量（列的数目）是巨大的，V中必定有很多很多项为0。

我们接下来看这两个矩阵U（Users Features Matrix ）和M（Movie Features Matrix）。U为用户对特征的偏好程度矩阵，M为物品对特征的拥有程度矩阵。$U \in R^{f \times n}, U \in R^{f \times n}$的每一行表示用户，每一列表示一个特征，它们的值表示用户与某一特征的相关性，值越大，表明特征越明显。矩阵$M \in R^{f \times m}$，的}每一行表示物品，每一列表示电影与特征的关联。

我们期望$UM \rightarrow V$, 也就是说用户和物品的隐藏因子相乘能够尽量逼近它的真实评分值。这个假设即有数学上的优美性，也有现实上的合理性。当然，直接靠矩阵分解从V计算出U, M是不明智的，可以考虑使用梯度下降法。

但仅此还不够。考虑不同用户的评分习惯等因素，我们在SVD模型中加入偏置项和其他影响因子，获得更进一步的SVD++模型。考虑全局平均数，用户偏置和物品偏置，有

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

进一步考虑其他影响,还能获得更多模型. 将用户的历史评分加入模型, 能够获得SVD++模型.其一种形式为

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

1.2 训练过程

经过调研，我们尝试了多个推荐系统领域的工具包：`surprise`、`lightfm`、`recsys`、`recQ`等。

最终我们选择了`surprise`，因为它提供了丰富的模型，并且在 movielens 数据集中进行了测试：

Movielens 100k	RMSE	MAE	Time
SVD	0.934	0.737	0:00:11
SVD++	0.92	0.722	0:09:03
NMF	0.963	0.758	0:00:15
Slope One	0.946	0.743	0:00:08
k-NN	0.98	0.774	0:00:10
Centered k-NN	0.951	0.749	0:00:10
k-NN Baseline	0.931	0.733	0:00:12
Co-Clustering	0.963	0.753	0:00:03
Baseline	0.944	0.748	0:00:01
Random	1.514	1.215	0:00:01

为了在豆瓣数据上验证各种模型的效果，我们使用了k折交叉验证的方法：（5折）

- 将全部训练集 S分成 k个不相交的子集，假设 S中的训练样例个数为 m，那么每一个子 集有 m/k 个训练样例，相应的子集称作 {s1,s2,...,sk}。
- 每次从分好的子集中里面，拿出一个作为测试集，其它k-1个作为训练集
- 根据训练训练出模型或者假设函数。
- 把这个模型放到测试集上，得到分类率。
- 计算k次求得的分类率的平均值，作为该模型或者假设函数的真实分类率。

这个方法充分利用了所有样本. 但是计算的时间成本极高。

由于数据规模太大，五折交叉验证花费了很长时间，SVD++由于时间过长（至少60小时还没结束），我们没有得到验证结果，其他模型结果如下：

```
BaselineOnly:
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.3666  1.3676  1.3668  1.3667  1.3660  1.3667  0.0005
MAE (testset)     1.0270  1.0278  1.0268  1.0275  1.0267  1.0272  0.0004
Fit time          30.43   49.50   50.81   50.24   49.94   46.18   7.89
Test time         23.47   22.10   23.94   22.08   24.28   23.18   0.92

Coclustering:
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.3986  1.4001  1.4017  1.3997  1.4003  1.4001  0.0010
MAE (testset)     1.0294  1.0315  1.0341  1.0318  1.0299  1.0313  0.0017
Fit time          219.40  225.22  227.43  228.03  234.43  226.90  4.84
Test time         27.99   26.32   27.71   27.00   29.08   27.62   0.93

KNNBaseline:
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.3545  1.3551  1.3548  1.3540  1.3558  1.3548  0.0006
MAE (testset)     0.9882  0.9889  0.9883  0.9873  0.9891  0.9884  0.0006
Fit time          701.48  721.27  812.15  693.96  692.37  724.25  45.14
Test time         1619.66 1723.80 1912.56 1893.13 1870.28 1803.89 113.61

KNNWithMeans:
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.3737  1.3720  1.3722  1.3722  1.3723  1.3725  0.0006
MAE (testset)     1.0107  1.0092  1.0092  1.0087  1.0095  1.0095  0.0007
Fit time          670.87  783.62  660.33  662.39  669.77  689.40  47.29
Test time         1645.46 1838.68 1819.14 1823.99 1753.39 1776.13 71.62

NormalPredictor:
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    2.4916  2.4942  2.4937  2.4905  2.4909  2.4922  0.0015
MAE (testset)     2.0368  2.0395  2.0392  2.0355  2.0363  2.0375  0.0016
Fit time          10.66   15.13   14.89   13.58   15.04   13.86   1.70
Test time         26.32   24.05   25.64   24.15   25.66   25.16   0.90

SVD:
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.3552  1.3556  1.3565  1.3563  1.3562  1.3560  0.0005
MAE (testset)     0.9645  0.9643  0.9651  0.9654  0.9658  0.9650  0.0006
Fit time          370.47  369.12  370.42  374.15  374.62  371.76  2.21
Test time         30.91   27.62   29.67   27.68   29.63   29.10   1.27
```

可以看到 SVD 和 KNN-Baseline 的效果最好。我们后续使用 SVD++ 直接训练模型，输出预测结果，效果超过了 SVD ，获得了所有模型中最好的结果。

1.3 模型融合

受到实验三分享的同学的启发，我们对以上模型进行了不同比例的线性加权融合，效果大大提升。

2. 爬虫

2.1 分析数据来源

经过分析数据来源，我们发现数据来源于该用户看过的电影和书籍。

比如用户 1000030 的数据的初始网页来源于：

电影：https://movie.douban.com/people/zoomq/collect

书籍：https://book.douban.com/people/zoomq/collect

只需设置 start= 起始评分位置（由于一页显示30条记录，所以是30的倍数），就能进行下一页跳转。

比如：https://movie.douban.com/people/zoomq/collect?start=60

由于我们的实验数据日期最晚到2012年，所以我们从最大页码处开始爬取记录，由后往前。

2.2 解析 HTML

我们使用 `lxml` 中的 `etree` 解析 HTML，然后就可以使用 xpath 来找到相应的元素内容，获得评分、日期、电影ID、书籍ID等信息。

2.3 设置 Header

除了常规的设置 `User-Agent` 之外，我们发现如果想要访问评分的内容，必须在 Header 中加入 `Cookie` 信息。

所以我们使用 `http.cookiejar` 先访问 www.douban.com ，获取到 `Cookie` 之后，再把它添加到 Header 中，然后访问评分的URL。

2.4 IP代理和并行

由于本地IP在多次爬取之后，会被识别为非法IP，并且单线程速度过慢，时间都浪费在了等待响应上，所以我们使用了HTTPS高匿代理（从虎头代理中购买）和多线程并发爬虫。

我们将一个用户的ID分配给一个爬虫，该爬虫负责爬取该用户所有的评分信息，然后保存在 `用户ID.json` 文件中。这样有一个好处就是，可以避免并行读写文件，免去并行操作数据库的繁琐，JSON文件也能很方便的检查爬取效果。

2.5 断点续爬

由于IP代理不稳定，并且爬取的数据量太大，我们无法一次性爬完所有数据，如果重新开始将浪费之前所有的努力。所以我们在程序开始前检查以爬取的用户（通过已存在的 JSON 文件），这样就可以随时开始，随时结束，只会浪费一部分未完全爬取完成的时间。

3. 结果

截至 2020.1.3 ，我们为第二名：

Filename	MAE	RMSE
PB17111630_陶柯宇_PB17111643_付佳伟_lab3_submission_13.txt	0.12130153720416656	0.5396953623296304
PB17111630_陶柯宇_PB17111643_付佳伟_lab3_submission_12.txt	0.1231030411127931	0.5460229943325844
PB17111663_lab3_submission_6.txt	0.3589787339042988	0.8702235200426475

4. 代码运行

4.1 环境

Ubuntu 18.04

Python 3.6.9

4.2 安装相关包

```
$ pip3 install -r requirements --user
```

4.3 Surprise

```
$ python3 surprise_models.py -m SVD
```

你可以在 `-m` 之后选择 `['NormalPredictor', 'BaselineOnly', 'KNNBasic', 'KNNWithMeans', 'KNNWithZScore', 'KNNBaseline', 'SVD', 'SVDpp', 'NMF', 'SlopeOne', 'CoClustering']` 中的任意模型（有些模型会失败，因为对数据有特殊要求）。

4.4 爬虫

```
$ python3 proxy_filter.py
$ python3 spider.py
```

如果要使用爬虫，你可能需要重新购买IP代理，并将其复制到 `Data/proxies.txt` 中。