【达芬奇密码】

数据挖掘训练营第一周项目文档

I 实现任务基本思路：

1.查看处理数据，并基于特征实现基础推荐（无个性）；

2.阅读文档，尝试实现U-U及I-I协同过滤个性化推荐；

3.学习KNN，尝试实现；

4.学习MF及TFIDF，尝试考虑tag及genres影响。

II 基础算法学习

1、协同过滤算法（CF）：

基于用户的协同过滤是在计算用户与用户之间相似度的前提下，为用户推荐与其相似度高的其他用户所使用的产品；

基于物品的协同过滤是在计算物品与物品相似度的前提下，为用户推荐与其购买过的物品相似度高的物品。

1.1基于用户的协同过滤算法:

基于用户的协同过滤算法用在本次训练项目中，即假设观影用户A喜欢看电影1、电影2，而我们通过用户相似度的计算了解到用户A与用户B有相似的观影爱好，所以我们会把用户A喜欢的电影而用户B还未观看的电影1、电影2推荐给用户B。

假设用户Ui，电影Mj，Ui对Mj的评分为γ，基于用户的协同过滤算法主要包括以下步骤：

i 搜集用户和电影的历史信息，计算用户u和其他用户的相似度Sim（u，N），找到和目标用户u兴趣相似的用户集合N（u）；

ii 找到这个集合中用户喜欢的，但目标用户还没有观看过的电影推荐给目标用户。

基于用户的协同过滤，通过下面的公式来计算用户对电影的喜好程度：

$$p_{uj} = \frac{\sum\limits_{N_i \in N(u)} Sim(u, N_i) \times r_{N_{i,j}}}{\sum\limits_{N_i \in N(u)} Sim(u, N_i)}$$

其中P表示用户u对电影j的喜好程度，γ表示用户Ni对电影j的评价，Sim表示用户u和用户Ni的相似度，最后根据P来对电影进行排序，为用户推荐分值最高的Top-N部电影。

1.2基于物品的协同过滤算法:

基于物品的协同过滤算法用在本次训练项目中，即假设观影用户A和B都喜欢看电影1、电影2，那我们就认为电影1和电影2具有较高的相似度，在用户C选择了电影1的情况下，则将电影2推荐给用户C。

基于物品的协同过滤算法并不利用物品的内容属性来计算物品之间的相似度，它主要通过分析用户的行为记录计算物品之间的相似度。该算法认为，电影1和电影2具有很大的相似度是因为喜欢电影1的用户大都也喜欢电影2。

基于用户的协同过滤算法主要包括以下步骤：

i 对于目标用户Ui及待评分电影Mj，根据用户对电影的历史偏好数据，计算电影Mj与其他已评分电影之间的相似度Sim(j,i)，找到与电影Mj相似度高的电影集合N(u)；

ii 根据所有物品N(u)的评分情况，选出N(u)中目标用户 可能喜欢的且没有观看过的推荐给目标用户并预测评分。

基于物品的协同过滤，通过下面的公式来计算用户对电影的喜好程度：

$$r_{uj} = \frac{\sum_{j \in N(u)} Sim(j,i) r_{ui}}{\sum_{j \in N(u)} Sim(j,i)}$$

其中γuj表示用户u对电影j的喜好程度，电影i是用户观看过的电影，γui表示用户u对电影i的偏好程度，然后根据γuj来对候选电影进行排序，为用户推荐分值最高的电影。

2、基于内容的推荐算法（CB）：

基础CB推荐算法利用物品的基本信息和用户偏好内容的相似性进行物品推荐。通过分析用户已经浏览过的物品内容，生成用户的偏好内容，然后推荐与用户感兴趣的物品内容相似度高的其他物品。

比如，用户近期浏览过冯小刚导演的电影“非诚勿扰”，主演是葛优；那么如果用户没有看过“私人订制”，则可以推荐给用户。因为这两部电影的导演都是冯小刚，主演都有葛优。

计算公式为：

$$\mathrm{Sim}(u_i, q_j) = \sum_k \lambda_k \mathrm{Sim}(u_{ik}, q_{jk})$$

3、基于KNN的CB推荐算法:

KNN(k-Nearest Neighbor)算法基于这样的假设：如果在特征空间中，一个样本的k个最邻近样本中的大多数样本属于某一个类别，则该样本也属于这个类别。

KNN在CB推荐算法中的应用于在CF推荐算法中的应用极为相似，它们都是要首先找到与目标物品相似的且已经被用户u评价过的k个物品，然后根据用户u对这k个物品的评价来预测其目标物品的评价。它们的差别在于，CF推荐算法中的KNN是根据用户对物品的评分来计算物品间相似度的，而CB推荐算法中KNN是根据物品画像来计算相似度的，所以对于后者来说，如何通过物品画像来计算物品间的相似度是算法中的关键步骤。相似度的计算可以使用余弦相似度或Pearson相关系数的计算方法。

i 采用余弦相似度公式计算相似度。；

ii 选择最近邻。在用户u评过分的所有物品中，找出k个与目标物品i相似度最高的物品，并用 N(u,i) 来表示这出k个物品的集合;

iii 计算预测值。在第二步的基础上，可使用以下公式计算用户对目标物品的评分：

$$\hat{r}_{u,i} = \frac{\sum\limits_{n \in N(u,i)} S_{i,n} r_{u,n}}{\sum\limits_{n \in N(u,i)} S_{i,n}}$$

4、个人的一些思路及想法:

i 根据用户对所有评价电影里在ratings.csv中选出其评价等级高的movieId；

ii 在genome_scores.csv中 根据movieId寻找与movieId关联度大的tagId；

iii 根据tagId在genome_tags.csv中找出相应的tag；

iv 再根据tag在tag.csv中匹配出相应的movieId、userId.

	movieId
123	296
124	541
125	750
126	858
127	908

	tagId
145	146
168	169
219	220
239	240
241	242

	tag
0	bloody
1	brutality
2	clever
3	clever
4	complex

	tag	userId	movieId	timestamp
0	bloody	6513	70	1368726740
1	bloody	6513	1089	1368726739
2	bloody	6513	6874	1368726740
3	bloody	6513	7022	1368726740
4	bloody	6513	32587	1368726740

III 基本实现步骤：

1.查看处理数据，并基于特征实现基础推荐（无个性）：

```
files = ['links','ratings','tags','movies','genome-scores','genome-tags']
for file in files:
    path = './ml-latest/ml-latest/' + file + '.csv'
    file = pd.read_csv(path)
    print(file.shape,file.dtypes)
```

查看数据结构及类型，得到：

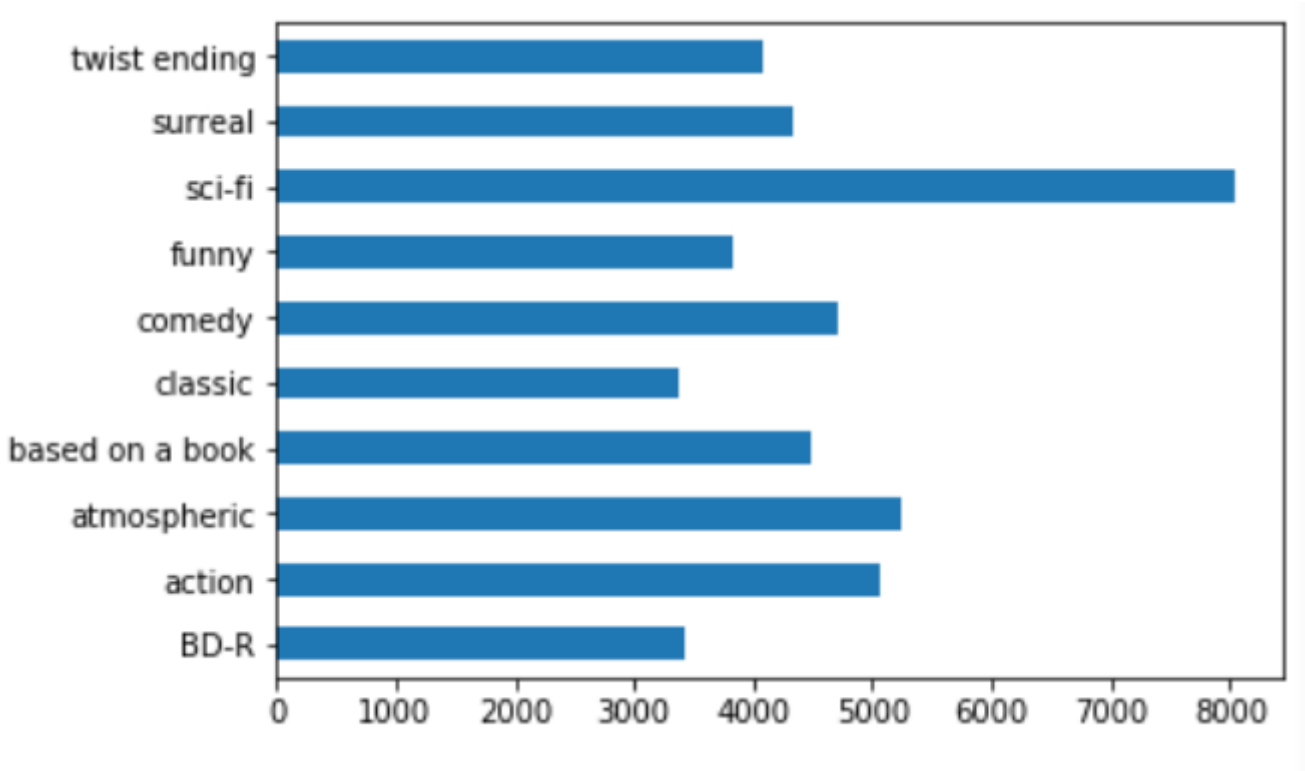
```
ratings:
(26024289, 4)
userId         int64
movieId        int64
rating       float64
timestamp      int64
dtype: object
```

等类似结果；

统计tag数据：

{"narrated": 502, "Dreamworks": 134, "England": 652, "espionage": 897, "jazz": 176, "politics": 1362, "nostalgic": 306, "coming of age": 1668, "dark comedy": 3199, "kafkaesque": 10, "military": 733, "satire": 1712, "Plot Recycling": 80, "comic book": 1342, "Comic Book adaption": 61, "Near Future": 41, "video game adaptation": 196, "suicide attempt": 97, "comedy": 4713, "sweet": 308, "Woody Allen": 407, "90s": 41, "funny": 3840, "great dialogue": 229, "swing dance": 1, "vegas": 9, "chocolate": 44, "colourful": 144, "cult film": 1715, "imagination": 430, "quirky": 3101, "surreal": 4324, "Whimsical": 14, "witty": 1144, "seduction": 38, "feel-good": 635, "hilarious": 878, "original plot": 392, "time loop": 375, "excellent script": 441, "great intelligent comedy": 2, "grumpy man": 1, "neurosis": 30, "sappy": 57, "writing": 111, "1980's cult": 13, "bittersweet": 1746, "charming": 139, "fanciful": 77, "fantasy": 3093, "fun premise": 2, "romance": 2790, "warm": 58, "mermaid": 48, "record store": 16, "romantic comedy": 454, "bookshop": 21, "chatting": 2, "chick flick": 603, "fun": 559, "New York City": 778, "romcom": 60, "British": 688, "London": 315, "1980s": 471, "80's": 54, "toy store": 8, "toys": 62, "creativity": 42, "on the road": 15, "actor talks to audience": 9, "awesome soundtrack": 88, "great soundtrack": 2030, "indie rock": 2, "John Cusack": 251, "mixtape": 3, "overrated": 767, "relationships": 710, "vinyl": 2, "Christmas": 528, "Good Romantic Comedies": 125, "john cusack": 23, "Meet-cute": 2, "serendipity": 13, "Boring": 154, "bullshit science": 3, "complicated": 677, "confusing": 293, "dreamlike": 1590, "high school": 1281, "plot holes": 425, "pretentious": 264, "stereotypes": 134, "atmospheric": 5240, "beautiful": 1043, "beautifully filmed": 440, "crafted": 4, "Cute": 73, "cute couples": 4,

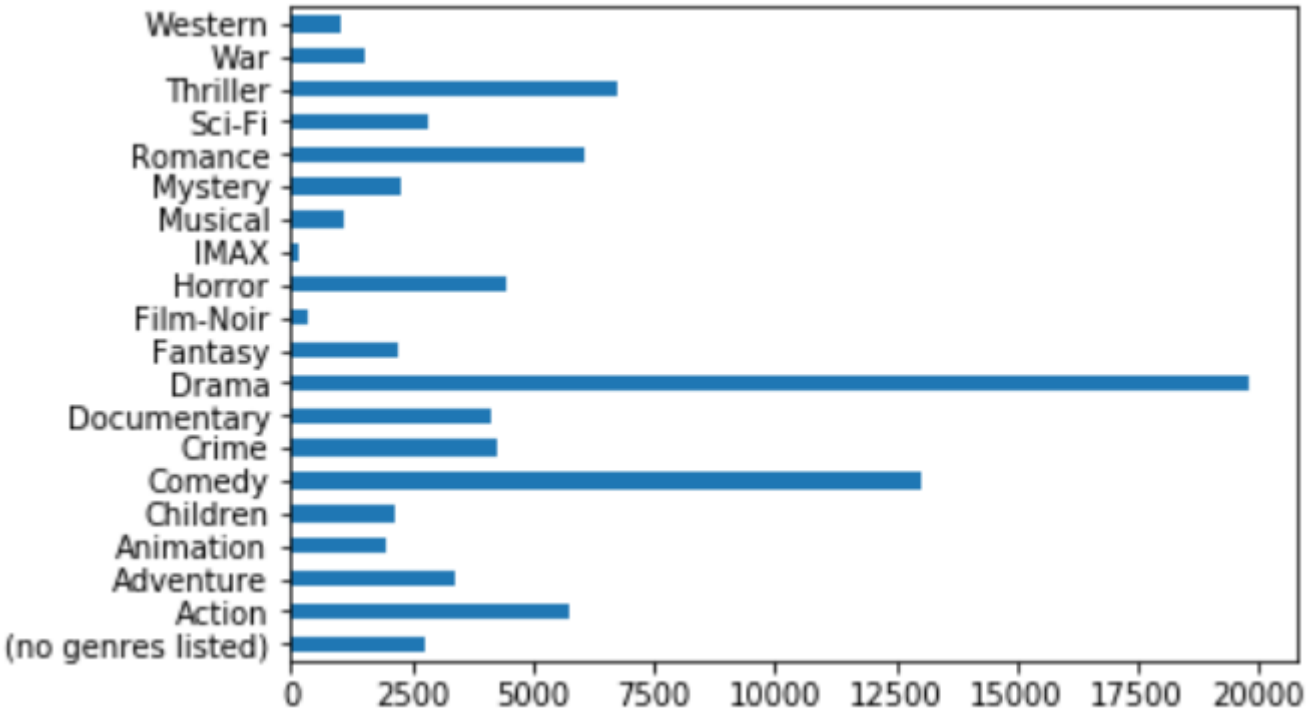
查看数量最多的是个tag：



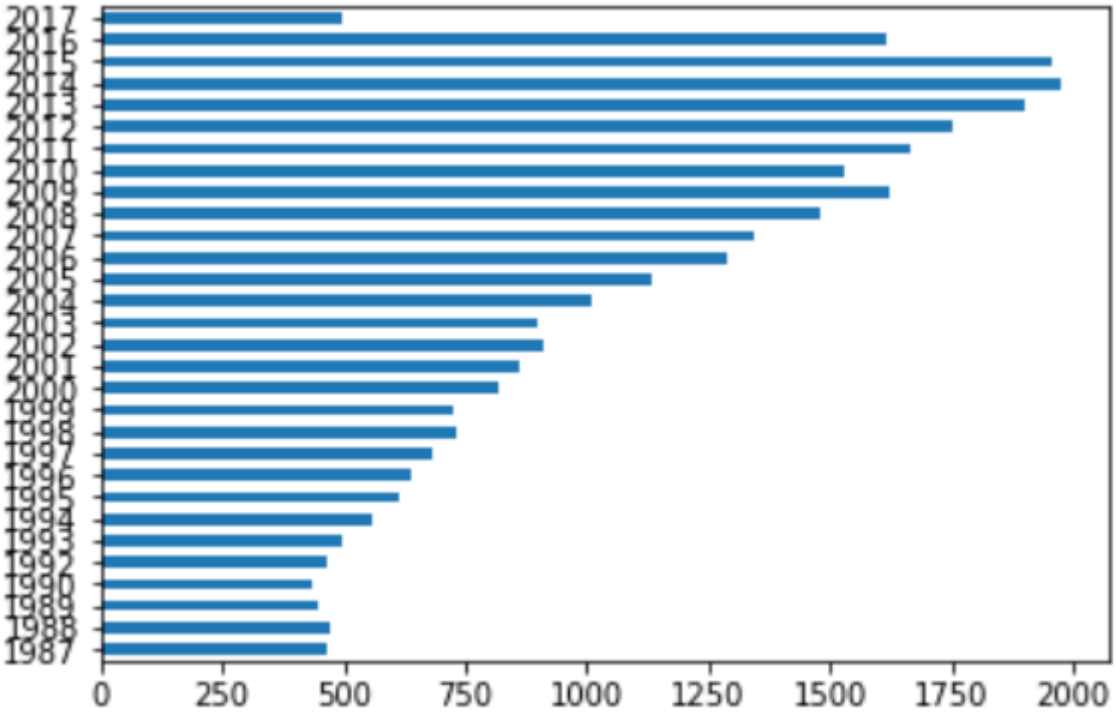
对genres进行编码：

{"Western": 0, "IMAX": 1, "Fantasy": 2, "War": 3, "Documentary": 4, "Romance": 5, "<null>": 6, "Mystery": 7, "Drama": 8, "Children": 9, "Musical": 10, "Thriller": 11, "Action": 12, "Horror": 13, "Crime": 14, "Animation": 15, "Sci-Fi": 16, "Film-Noir": 17, "(no genres listed)": 18, "Comedy": 19, "Adventure": 20}

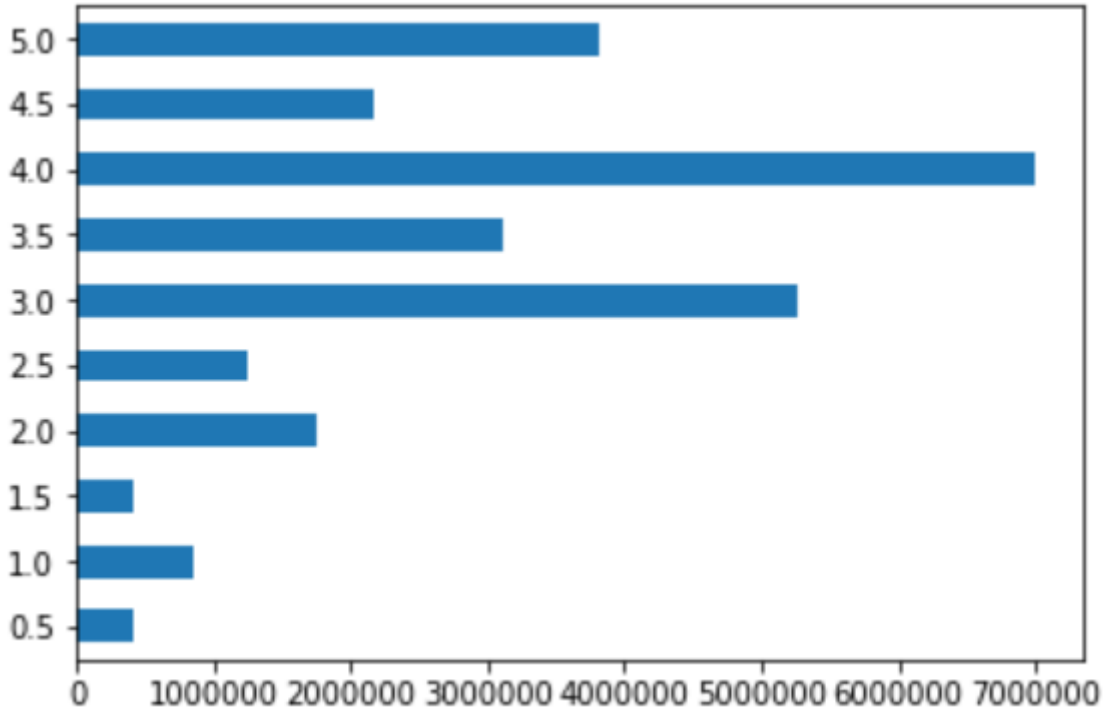
统计type数量：



统计发行时间：



统计rating：



整合rating信息：

	userId	movieId	rating	timestamp	count_rating_user	count_rating_movie	average_rating_user	average_rating_movie
0	1	110	1.0	1425941529	27	66512	4.277778	4.016057
1	1	147	4.5	1425942435	27	4967	4.277778	3.595933
2	1	858	5.0	1425941523	27	57070	4.277778	4.339811
3	1	1221	5.0	1425941546	27	36679	4.277778	4.263475
4	1	1246	5.0	1425941556	27	25752	4.277778	3.912803
5	1	1968	4.0	1425942148	27	26611	4.277778	3.827553
6	1	2762	4.5	1425941300	27	49643	4.277778	4.021282
7	1	2918	5.0	1425941593	27	27822	4.277778	3.926928
8	1	2959	4.0	1425941601	27	60024	4.277778	4.230716
9	1	4226	4.0	1425942228	27	40706	4.277778	4.157078

根据数量排序可完成非个性化的基本推荐；

2.阅读文档，尝试实现U-U及I-I协同过滤个性化推荐:

构建U-I数据集：

```
mtrain = train_data.pivot(index='userId', columns='movieId', values='rating')
mtest = test_data.pivot(index='userId', columns='movieId', values='rating')
```

计算相似矩阵：

```
def similarity(ratings, kind='user', epsilon=1e-9):
if kind == 'user':
    sim = ratings.dot(ratings.T) + epsilon
    #print(sim)
elif kind == 'item':
    sim = ratings.T.dot(ratings) + epsilon
```

```
norms = np.array([np.sqrt(np.diagonal(sim))])
#print(norms.T.dot(norms))
return (sim / (norms.T.dot(norms)))
```

如下：

movieId	1	2	3	5	6	7	10	11	14	16	...	116797	119145
movieId													
1	1.000000	0.454397	0.267114	0.265525	0.381293	0.274405	0.422687	0.341367	0.156561	0.344605	...	0.214639	0.194472
2	0.454397	1.000000	0.274667	0.280887	0.291595	0.254832	0.397229	0.279282	0.130903	0.280690	...	0.154419	0.152849
3	0.267114	0.274667	1.000000	0.351085	0.227530	0.297823	0.266720	0.306551	0.172548	0.232639	...	0.039366	0.039820
5	0.265525	0.280887	0.351085	1.000000	0.190847	0.321593	0.243711	0.314410	0.157480	0.191000	...	0.058944	0.058376
6	0.381293	0.291595	0.227530	0.190847	1.000000	0.214333	0.397899	0.273643	0.194011	0.473401	...	0.142491	0.125130
7	0.274405	0.254832	0.297823	0.321593	0.214333	1.000000	0.256097	0.394462	0.186605	0.205791	...	0.049906	0.041839
10	0.422687	0.397229	0.266720	0.243711	0.397899	0.256097	1.000000	0.306338	0.159514	0.330045	...	0.107339	0.106026
11	0.341367	0.279282	0.306551	0.314410	0.273643	0.394462	0.306338	1.000000	0.225138	0.264658	...	0.060074	0.047474
14	0.156561	0.130903	0.172548	0.157480	0.194011	0.186605	0.159514	0.225138	1.000000	0.220825	...	0.030222	0.019644
16	0.344605	0.280690	0.232639	0.191000	0.473401	0.205791	0.330045	0.264658	0.220825	1.000000	...	0.129082	0.113114

根据相似矩进行预测：

```
if kind == 'user':
    res = similarity.dot(ratings)
    #print(res.shape)
    sums = np.zeros(res.shape)
    array_1 = np.array([np.abs(similarity).sum(axis=1)]).T
    array_temp = np.array([np.abs(similarity).sum(axis=1)]).T
    for i in range(res.shape[1]-1):
        array_temp = np.c_[array_temp,array_1]
        #print(array_temp.shape)
    return res / array_temp
```

I与U类似，得到预测rating：

movieId	1	2	3	5	6	7	10	11	14	16
userId										
11	2.121557	0.959717	0.289000	0.266962	1.088706	0.313790	0.982986	0.536097	0.129351	0.900574
12	2.151664	0.872661	0.336533	0.290248	1.134073	0.402980	0.975425	0.720177	0.195836	0.982566
15	2.115186	0.898371	0.280552	0.242463	1.155682	0.308583	0.984899	0.548222	0.140064	0.942553
16	2.081611	0.904968	0.272895	0.235041	1.115159	0.290732	0.975687	0.523014	0.133496	0.922116
20	2.230679	0.988022	0.285483	0.267467	0.980175	0.316170	0.954179	0.532993	0.118080	0.822868
24	2.152985	0.940498	0.379111	0.323361	1.199846	0.429976	1.054513	0.768304	0.198861	1.029483
34	2.095760	0.929444	0.292999	0.253989	1.163698	0.315942	1.007310	0.558881	0.146687	0.972241
37	2.199920	0.937647	0.326583	0.292474	1.089179	0.374604	1.002342	0.650534	0.155608	0.927636
41	2.241506	0.992724	0.271539	0.273129	0.916498	0.318943	0.886402	0.520689	0.104748	0.767177
43	2.290153	1.097902	0.382555	0.355366	1.028794	0.416188	1.082140	0.688834	0.149311	0.878550

除去偏差：

```
if kind == 'user':
    user_bias = ratings.mean(axis=1)
    col_list = ratings.columns.tolist()
    for i in col_list:
        ratings[i] = (ratings[i] - user_bias).copy()
        print(i)
    res = similarity.dot(ratings)
    array_1 = np.array([np.abs(similarity).sum(axis=1)]).T
    array_temp = np.array([np.abs(similarity).sum(axis=1)]).T
    for j in range(res.shape[1]-1):
        array_temp = np.c_[array_temp,array_1]
        print(array_temp.shape)
    pred = res / array_temp
    print(pred)
    pred_col = pred.columns.tolist()
    for col in pred_col:
        pred[col] = (pred[col] + user_bias).copy()
    print(pred)
```

I与U类似，得到预测rating：

movieId	1	2	3	5	6	7	10	11	14	16	...	116797
userId												
11	1.757370	0.595530	0.000000	0.000000	0.724519	0.000000	0.618799	0.171911	0.000000	0.536387	...	0.152877
12	1.937132	0.658128	0.122001	0.075716	0.919540	0.188447	0.760892	0.505644	0.000000	0.768033	...	0.084301
15	1.941334	0.724519	0.106699	0.068611	0.981830	0.134731	0.811047	0.374369	0.000000	0.768700	...	0.287435
16	1.827662	0.651019	0.018945	0.000000	0.861210	0.036783	0.721738	0.269065	0.000000	0.668167	...	0.231508
20	2.058273	0.815616	0.113077	0.095061	0.807769	0.143764	0.781773	0.360587	0.000000	0.650462	...	0.342304
24	2.387430	1.174943	0.613556	0.557806	1.434291	0.664421	1.288958	1.002749	0.433306	1.263928	...	0.524914
34	2.051219	0.884904	0.248458	0.209449	1.119158	0.271401	0.962769	0.514341	0.102146	0.927701	...	0.406233
37	1.899387	0.637115	0.026051	0.000000	0.788647	0.074072	0.701810	0.350002	0.000000	0.627104	...	0.075197
41	2.003135	0.754352	0.033167	0.034757	0.678126	0.080571	0.648030	0.282317	0.000000	0.528806	...	0.387172
43	2.067430	0.875179	0.159832	0.132643	0.806071	0.193465	0.859417	0.466111	0.000000	0.655827	...	0.159107

可以根据条件排序完成U-U、I-I推荐；

3.学习KNN，尝试实现：

展开movie的分类特征：

	movieId	title	(no genres listed)	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	...	Film-Noir	Horror	IMAX	Musical	Mystery	Romance
0	1	Toy Story (1995)	0	0	1	1	1	1	0	0	...	0	0	0	0	0	0
1	2	Jumanji (1995)	0	0	1	0	1	0	0	0	...	0	0	0	0	0	0
2	3	Grumpier Old Men (1995)	0	0	0	0	0	1	0	0	...	0	0	0	0	0	1
3	4	Waiting to Exhale (1995)	0	0	0	0	0	1	0	0	...	0	0	0	0	0	1
4	5	Father of the Bride Part II (1995)	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0

添加总rating与平均rating特征：

	rating	
	size	mean
movieId		
1	28908.0	3.854884
2	15843.0	3.097993
3	5861.0	2.988569
5	5476.0	2.867421
6	14352.0	3.846154
7	6203.0	3.229889
9	1822.0	2.718990
10	15093.0	3.371331
11	9682.0	3.550196

将总rating归一化，并将电影及特征写入字典：

```
{1: ('Toy Story (1995)',
  [0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  0.8247484788240256,
  3.8548844610488446),
 2: ('Jumanji (1995)',
  [0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  0.4312006747394421,
  3.0979928043931073),
 3: ('Grumpier Old Men (1995)',
  [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
  0.1305199108380023,
  2.988568503668316),
 5: ('Father of the Bride Part II (1995)',
  [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  0.1189228266763058,
  2.8674214755295835),
```

给定电影，计算电影间距离：

特征的余弦距离：

```
genresA = a[1]
genresB = b[1]
genreDistance = spatial.distance.cosine(genresA, genresB)
```

受欢迎程度的绝对距离：

```
popularityA = a[2]
popularityB = b[2]
popularityDistance = abs(popularityA - popularityB)
```

电影间距离为二者之和：

```
distances = []
for movie in movieDict:
    if (movie != movieID):
```

```
        dist = ComputeDistance(movieDict[movieID], movieDict[movie])
        distances.append((movie, dist))
distances.sort(key=operator.itemgetter(1))
neighbors = []
for x in range(K):
    neighbors.append(distances[x][0])
```

```
[(2, 0.6189511348431), (3, 1.3780008019691854), (5, 1.2586120566477619), (6, 1.438460148201699), (7, 1.367698976558016), (9, 1.8158925236460028), (10, 1.1579406367303375), (11, 1.32093238322109), (12, 1.4871037599787038), (14, 1.7956202180854268), (16, 1.500813302006145), (17, 1.5942225435267185), (18, 1.3143383052169528), (19, 1.0528165267965661), (20, 1.5884511115127418), (21, 1.2251434802751295), (22, 1.701457919151756), (23, 1.7861618169769264), (24, 1.720645822037472), (25, 1.5438580637387795), (29, 1.293806855834689), (31, 1.7245315982890537), (32, 1.1296463642388095), (34, 1.0295460757808608), (36, 1.5697029941562746), (39, 1.084609633826526), (41, 1.7962226640159047), (44, 1.1514737790332998), (45, 1.4449157557134087), (47, 1.0784987047412495), (48, 1.2331706729321046), (50, 1.068227001626604), (52, 1.4684111470020407), (58, 1.4544946460080048), (60, 0.9890035476390516), (62, 1.6084402674859932), (65, 1.3408459261579733), (69, 1.3243389076628833), (70, 1.31133506621773), (74, 1.8247484788240256), (76, 1.8130911500692815), (79, 1.810229531899512), (86, 1.5478737652320547), (88, 1.3364480708654858), (94, 1.5403431911010828), (95, 1.2866833320734306), (100, 1.816886559431291), (101, 1.3115369316402314), (104, 1.0526056707208988), (105, 1.7258569793361045), (107, 1.1041058071235796), (110, 1.021296463642388), (111, 1.3505331646484728), (112, 1.2432496854205795), (122, 1.4802659986677815), (125, 1.3429243646181215), (135, 1.3423219186876438), (140, 1.8147779986746189), (141, 1.1040545531836978), (144, 1.3364480708654858), (145, 1.412476655220194), (147, 1.7769142719440931), (150, 0.9372044478032938), (151, 1.6794987649858426), (153, 0.9626604933005722), (158, 1.029060221927406), (160, 1.4247756349266885), (161, 1.5738899933730948), (162, 1.7514609313814087), (163, 1.583468883667691), (164, 1.780679559009579), (165, 1.3633953852641727), (168, 1.6806434122537501), (169, 1.2944040760391438), (170, 1.484568393526604), (172, 1.6435929875293693), (173, 1.6002168805349721), (175, 1.766010000602446), (180, 1.3476375270731071), (185, 1.5427736610639196), (186, 1.4058036816607329), (193, 1.7413398397493824), (194, 1.444179487432165), (196, 1.6334417736008193), (198, 1.6816374480390386), (203, 1.3329237621721912), (204, 1.7535996144346044), (207, 1.7836917886619676), (208, 1.1664351243500737), (215, 1.7251340442
```

选择与给定电影距离最近的K个：

```
7 Neighbors:
Shrek (2001) 3.7539976638337293
Monsters, Inc. (2001) 3.8207094444169596
Toy Story 2 (1999) 3.769925280199253
Finding Nemo (2003) 3.8172483089893148
Monty Python and the Holy Grail (1975) 4.1413101880553835
Lord of the Rings: The Fellowship of the Ring, The (2001) 4.091105338260583
Aladdin (1992) 3.6580278128950696

Estimated avg. rating:
3.8646177195214704

Real avg. rating:
3.8548844610488446
```

注：选取了较小的数据集`len(movieId) = 1085`

4.学习MF及TFIDF，尝试考虑tag及genres影响。

（1）MF：

参考文档http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/

初始的P、Q矩阵：

```
m, n = trainset.shape
P = 3 * np.random.rand(k,m) # user特征矩阵
Q = 3 * np.random.rand(k,n) # movie特征矩阵
```

计算梯度误差并更新P、Q：

```
 e = trainset[u, i] - prediction(P[:,u],Q[:,i])
 P[:,u] += step_size * ( e * Q[:,i] - lmbda * P[:,u])
 Q[:,i] += step_size * ( e * P[:,u] - lmbda * Q[:,i])
```

得到P、Q后预测：

	Actual Rating	Predicted Rating
0	3.5	3.352735
7	2.5	2.649782
30	3.0	3.551785
63	4.0	3.023389
99	2.5	2.521704

根据预测评分进行推荐；

(2) TFIDF

使用sklearn中TfidfVectorizer，参考文档：http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

(i) 得到movieId与title的数据集：

	movieId	title
0	1	Toy Story (1995)
1	2	Jumanji (1995)
2	3	Grumpier Old Men (1995)
3	4	Waiting to Exhale (1995)
4	5	Father of the Bride Part II (1995)

转化`tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 3), min_df=0, stop_words='english')`

对title以词为单位的分析得到的相似矩阵：

movieId	1	2	3	4	5	6	7	8	9	10	...	176255	176257	176259	176263
movieId															
1	1.000000	0.079906	0.045940	0.054529	0.047625	0.089364	0.082916	0.055243	0.057820	0.079906	...	0.0	0.000000	0.000000	0.044276
2	0.079906	1.000000	0.059734	0.070902	0.061925	0.116197	0.107814	0.071831	0.075181	0.103900	...	0.0	0.000000	0.000000	0.057571
3	0.045940	0.059734	1.000000	0.040763	0.035602	0.066804	0.061984	0.041297	0.043223	0.059734	...	0.0	0.000000	0.000000	0.033099
4	0.054529	0.070902	0.040763	1.000000	0.042258	0.079294	0.073573	0.049018	0.051305	0.070902	...	0.0	0.000000	0.000000	0.039287
5	0.047625	0.061925	0.035602	0.042258	1.000000	0.069254	0.064258	0.042812	0.044809	0.061925	...	0.0	0.000000	0.000000	0.034313
6	0.089364	0.116197	0.066804	0.079294	0.069254	1.000000	0.120574	0.080332	0.084079	0.116197	...	0.0	0.000000	0.000000	0.217811
7	0.082916	0.107814	0.061984	0.073573	0.064258	0.120574	1.000000	0.074536	0.078013	0.107814	...	0.0	0.000000	0.000000	0.059740
8	0.055243	0.071831	0.041297	0.049018	0.042812	0.080332	0.074536	1.000000	0.051976	0.071831	...	0.0	0.000000	0.000000	0.039801
9	0.057820	0.075181	0.043223	0.051305	0.044809	0.084079	0.078013	0.051976	1.000000	0.075181	...	0.0	0.000000	0.000000	0.041658
10	0.079906	0.103900	0.059734	0.070902	0.061925	0.116197	0.107814	0.071831	0.075181	1.000000	...	0.0	0.000000	0.000000	0.057571

排序选出最相似的十个标题：

	movieId	1	title
0	1	1.000000	Toy Story (1995)
1	78499	0.454313	Toy Story 3 (2010)
2	3114	0.447155	Toy Story 2 (1999)
3	106022	0.344004	Toy Story of Terror (2013)
4	2274	0.338930	Lilian's Story (1995)
5	120474	0.288689	Toy Story That Time Forgot (2014)
6	295	0.285956	Pyromaniac's Love Story, A (1995)
7	115879	0.250054	Toy Story Toons: Small Fry (2011)
8	115875	0.249358	Toy Story Toons: Hawaiian Vacation (2011)
9	192	0.247898	The Show (1995)

（ii）得到movieId与tag的数据集：

	movieId	tag
0	1	Pixar:time travel:computer animation:funny:Pix...
1	2	Robin Williams:time travel:fantasy:Children:ki...
2	3	old people that is actually funny:funny:Jack L...
3	4	characters:chick flick:revenge:girl movie:char...
4	5	childhood classics:steve martin:childhood clas...
5	6	atmospheric:philosophy:slow paced:tense:Top 10...
6	7	as good maybe better than original:remake:rema...
7	8	Library System:based on a book:Mark Twain:adap...
8	9	action:jean-claude van damme:Peter Hyams:Jean-...
9	10	bond:espionage:james bond:thriller:Bond:Pierce...
10	11	Romance:politics:president:Aaron Sorkin:Michae...
11	12	Leslie Nielsen:Mel Brooks:Leslie Nielsen:Mel B...

与（i）类似，选取相似tag：

	movieId	1	tag
0	1	1.000000	Pixar:time travel:computer animation:funny:Pix...
1	3114	0.821930	animation:Disney:funny:Pixar:sequel:imdb top 2...
2	2355	0.685923	animation:Disney:funny:Pixar:sweet:talking ani...
3	4886	0.636835	hilarious:Pixar:animated:animation:CGI:cute:Di...
4	78499	0.614094	adventure:animation:bittersweet:friendship:Pix...
5	6377	0.595107	myfav:anxiety:father-son relationship:funny:he...
6	50872	0.541241	lawyers:pixar:very nice movie:cooking:France:p...
7	5218	0.530152	pixar:pixar:seen at the cinema:pixar:pixar:ani...
8	8961	0.526469	Adventure:Pixar:superhero:Pixar:watched 2006:a...
9	157296	0.520626	family:friendship:pixar:animation:animation:fi...

5.学习surprise：

学习用于构建和分析推荐系统的Python scikit http://surpriselib.com

构造Dataset：

```
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], reader)
trainingSet = data.build_full_trainset()
```

基本的基于I的协同过滤算法：

```
sim_options = {
'name': 'cosine',#余弦相似度
'user_based': False
}
knn = KNNBasic(sim_options=sim_options)
knn.train(trainingSet)
testSet = trainingSet.build_anti_testset()
predictions = knn.test(testSet)
```

根据预测读取每个U最可能喜欢的5个I：

```
681 ['711', '1122', '1682', '1674', '1536']
330 ['1236', '1293', '1235', '1617', '1624']
866 ['1599', '1520', '1309', '1310', '1156']
322 ['711', '1599', '47', '1674', '1308']
283 ['1431', '1674', '711', '1430', '1599']
674 ['1582', '1561', '1565', '1563', '1557']
208 ['1673', '1358', '1627', '1593', '1536']
563 ['1309', '1310', '1653', '1614', '1156']
624 ['1582', '1561', '1565', '1563', '1557']
911 ['1658', '1601', '382', '649', '1459']
942 ['1505', '88', '596', '404', '1614']
51 ['1237', '1191', '1243', '1662', '1669']
785 ['1653', '1430', '1616', '1654', '1661']
162 ['1430', '1309', '1310', '1654', '1606']
735 ['1582', '1561', '1565', '1563', '1557']
```

读取对应Id的title：

```
681 ['Substance of Fire, The (1996)', 'They Made Me a Criminal (1939)', 'Scream of Stone (Schrei aus Stein) (1991)', 'Mamma
330 ['Other Voices, Other Rooms (1997)', 'Star Kid (1997)', 'Big Bang Theory, The (1994)', 'Hugo Pool (1997)', 'Hush (1998)
866 ["Someone Else's America (1995)", 'Fear, The (1995)', 'Very Natural Thing, A (1974)', 'Walk in the Sun, A (1945)', 'Cyc
322 ['Substance of Fire, The (1996)', "Someone Else's America (1995)", 'Ed Wood (1994)', 'Mamma Roma (1962)', 'Babyfever (1
283 ['Legal Deceit (1997)', 'Mamma Roma (1962)', 'Substance of Fire, The (1996)', 'Ill Gotten Gains (1997)', "Someone Else'
674 ['T-Men (1947)', 'Tigrero: A Film That Was Never Made (1994)', 'Daens (1992)', 'Promise, The (Versprechen, Das) (1994)'
208 ['Mirage (1995)', 'The Deadly Cure (1996)', 'Wife, The (1995)', 'Death in Brunswick (1991)', 'Aiqing wansui (1994)']
563 ['Very Natural Thing, A (1974)', 'Walk in the Sun, A (1945)', 'Entertaining Angels: The Dorothy Day Story (1996)', 'Rel
624 ['T-Men (1947)', 'Tigrero: A Film That Was Never Made (1994)', 'Daens (1992)', 'Promise, The (Versprechen, Das) (1994)'
911 ['Substance of Fire, The (1996)', 'Office Killer (1997)', 'Adventures of Priscilla, Queen of the Desert, The (1994)', '
942 ['Killer: A Journal of Murder (1995)', 'Sleepless in Seattle (1993)', 'Hunchback of Notre Dame, The (1996)', 'Pinocchio
51 ['Twisted (1996)', 'Letter From Death Row, A (1998)', 'Night Flier (1997)', 'Rough Magic (1995)', 'MURDER and murder (19
785 ['Entertaining Angels: The Dorothy Day Story (1996)', 'Ill Gotten Gains (1997)', 'Desert Winds (1995)', 'Chairman of th
```

根据给定'Star Wars (1977)'推荐top10：

```
The 10 nearest neighbors of Star Wars (1977) are:
Empire Strikes Back, The (1980)
Return of the Jedi (1983)
Raiders of the Lost Ark (1981)
Indiana Jones and the Last Crusade (1989)
Sting, The (1973)
L.A. Confidential (1997)
Princess Bride, The (1987)
E.T. the Extra-Terrestrial (1982)
Terminator, The (1984)
Get Shorty (1995)
```

基于I对不同的K：

```
for k in range(10, 100, 10):
    print ("K is " +  str(k))
    algo = KNNBasic(k, sim_options = {'name':'MSD','user_based': False })
    perf = evaluate(algo, data, measures=['RMSE', 'MAE'])
    print_perf(perf)
```

计算均方差差异相似度：

```
algo = KNNBasic(sim_options = {
'name':'MSD',
'user_based': False})
perf = evaluate(algo, data, measures=['RMSE', 'MAE'])
print_perf(perf)
```

计算Pearson相关系数：

```
algo = KNNBasic(sim_options = {'name':'pearson','user_based': False })
perf = evaluate(algo, data, measures=['RMSE', 'MAE'])
print_perf(perf)
```

基于非负矩阵分解的协同过滤算法：

```
data.split(n_folds=3)
algo = NMF()
perf = evaluate(algo, data, measures=['RMSE', 'MAE'])
print_perf(perf)
```

SVD算法：

```
algo = SVD(biased=False) #PMF
perf = evaluate(algo, data, measures=['RMSE', 'MAE'])
print_perf(perf)
```

基本的协同过滤算法，考虑到每个用户的平均评分：

```
data.split(n_folds=3)
algo = KNNWithMeans(sim_options = {'name':'pearson','user_based': False })
perf = evaluate(algo, data, measures=['RMSE', 'MAE'])
print_perf(perf)
```

基于用户：

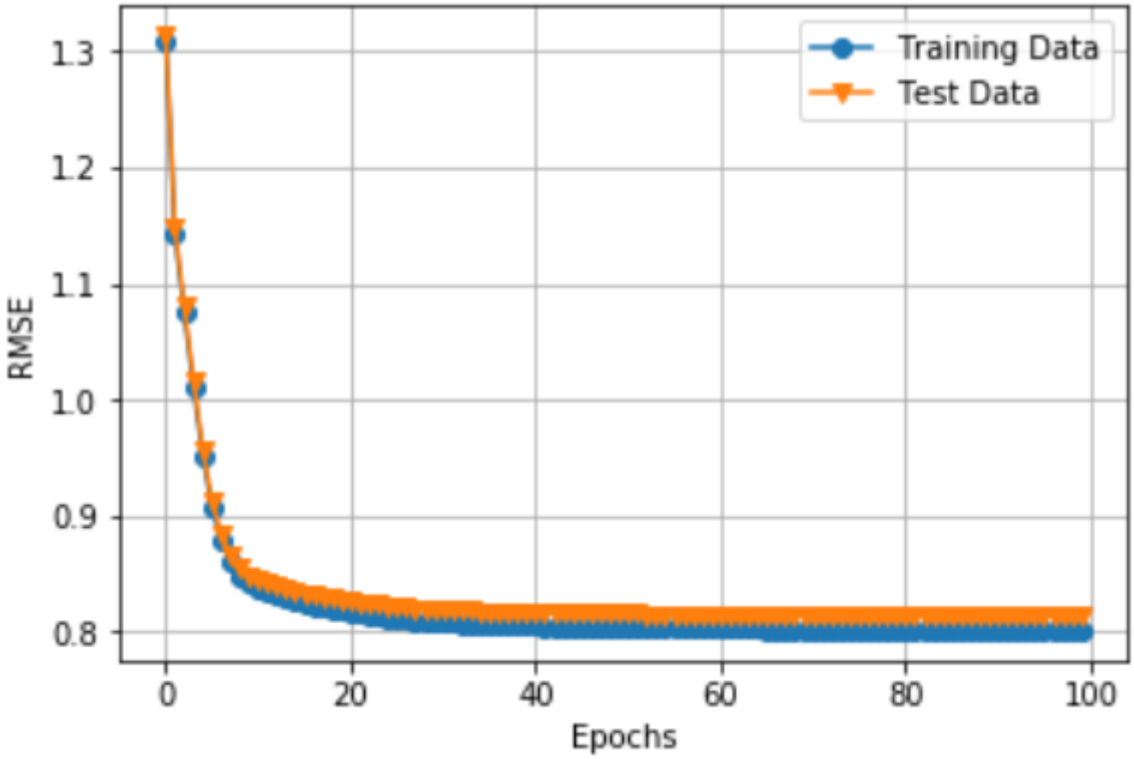
```
algo = KNNWithMeans(sim_options = {'name':'pearson','user_based': True })
perf = evaluate(algo, data, measures=['RMSE', 'MAE'])
print_perf(perf)
```

基于用户的K选择：

```
for k in range(10, 100, 10):
print ("K is " +  str(k))
algo = KNNBaseline(k, sim_options = {'name':'MSD','user_based': True})
perf = evaluate(algo, data, measures=['RMSE', 'MAE'])
print_perf(perf)
```

IV 评测结果展示

1.MF RMSE：



训练误差达到0.801

2.surprise:

- 基于I，选择不同K：

k = 10：

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
MAE	0.8080	0.7963	0.8053	0.7970	0.7960	0.8005
RMSE	1.0233	1.0138	1.0178	1.0076	1.0118	1.0148

k = 20：

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
MAE	0.7819	0.7754	0.7808	0.7703	0.7740	0.7765
RMSE	0.9893	0.9858	0.9868	0.9755	0.9816	0.9838

k = 70：

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean
MAE	0.7797	0.7740	0.7788	0.7706	0.7741	0.7755
RMSE	0.9830	0.9808	0.9815	0.9735	0.9776	0.9793

- 基于I，基本KNN：

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7833	0.7777	0.7783	0.7798
RMSE	0.9895	0.9815	0.9829	0.9846

- 基于I，余弦距离：

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.8189	0.8251	0.8221	0.8220
RMSE	1.0334	1.0399	1.0356	1.0363

- 均方差差异相似：

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7749	0.7822	0.7829	0.7800
RMSE	0.9806	0.9877	0.9862	0.9848

- pearson：

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.8365	0.8427	0.8394	0.8395
RMSE	1.0461	1.0557	1.0456	1.0491

- NMF：

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7684	0.7714	0.7608	0.7669
RMSE	0.9782	0.9811	0.9673	0.9755

- SVD：

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7630	0.7650	0.7592	0.7624
RMSE	0.9685	0.9694	0.9622	0.9667

- KNNWithMeans：

	Fold 1	Fold 2	Fold 3	Mean
MAE	0.7443	0.7472	0.7446	0.7454
RMSE	0.9489	0.9543	0.9499	0.9510

- userBased：

```
        Fold 1  Fold 2  Fold 3  Mean
MAE     0.7518  0.7499  0.7464  0.7494
RMSE    0.9592  0.9555  0.9546  0.9564
```

- user*MSD*K:

k = 30:

```
        Fold 1  Fold 2  Fold 3  Mean
MAE     0.7356  0.7392  0.7408  0.7385
RMSE    0.9328  0.9395  0.9386  0.9370
```

结果较好。

V 遇到的问题

1. 基础CF的RSME计算会产生Memory error的错误，打算减少数据集大小再进行尝试；
2. FAISS安装出现很多问题，暂时还没成功，打算寻找替代的开源近似KNN解决方案;
3. 还没有对运行时间进行记录。

VI 希望的交流方向

希望对于训练营的任务，能得到老师和优秀同学的实现样例。