hw4 电影推荐系统

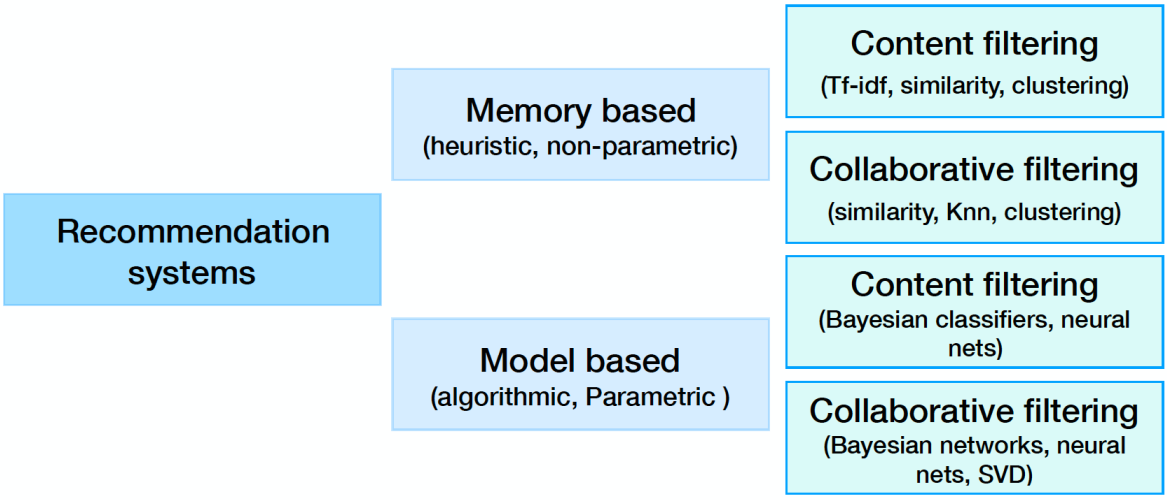
学号：191250016　　姓名：陈梓俊

数据集分析

数据集含有`movies.csv`、`rating.csv`、`tags.csv`三个文件，`movies.csv`中含有`genres`信息，可以用于基于内容的推荐；`rating.csv`含有每个用户对电影的评分信息，可以用于协同过滤推荐；`tags.csv`中含有每个用户对于电影的标记评论，既可以通过语义转换变成量化的评分（好坏的评价），用于协同过滤推荐，同时，也可以作为电影内容的一部分用于基于内容的推荐。

在本电影推荐系统中，我使用`movies.csv`、`rating.csv`、`tags.csv`三个文件信息，同时将`tags.csv`中的标签用作电影内容的一部分用于基于内容的推荐

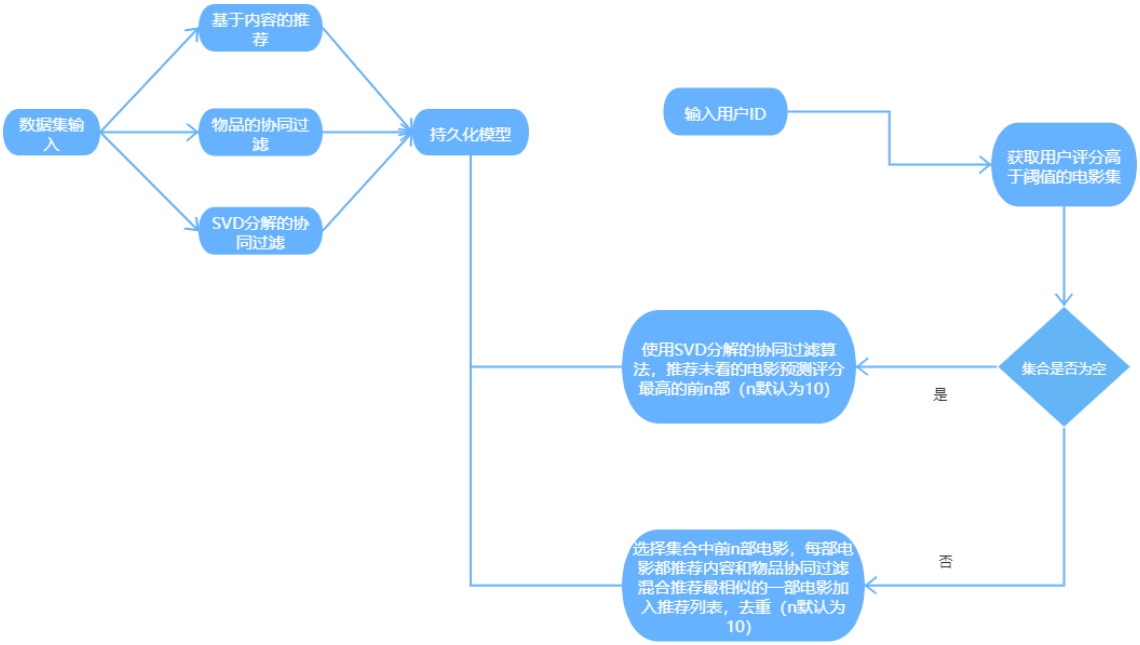
使用的推荐算法



推荐算法主要有基于领域和基于模型两种主要类型。基于领域的推荐算法可解释性更强，可以更好地体现结果相关关系。基于模型的推荐算法可以发掘潜在的关系，有更加惊喜的效果产生。因此我同时使用两种类型的模型，基于领域的推荐算法中，混合使用了基于内容的推荐算法和物品的协同过滤（itemCF）。为什么没有选择用户的协同过滤（userCF）的原因，是物品的性质更加简单，而用户的喜好相似性相同的概率很小，为了使结果相关性更好的体现，使用了物品的协同过滤。混合使用基于内容的推荐算法和物品的协同过滤，既有内容的相关性，又利用了群体智慧，使得推荐算法更加均衡。为了给用户营造惊喜的推荐，不会出现千篇一律的感觉，同时我还使用了基于矩阵分解（使用了SVD）的协同过滤推荐，预测用户未看的电影的评分，选取预测评分高的电影作为推荐。

推荐流程

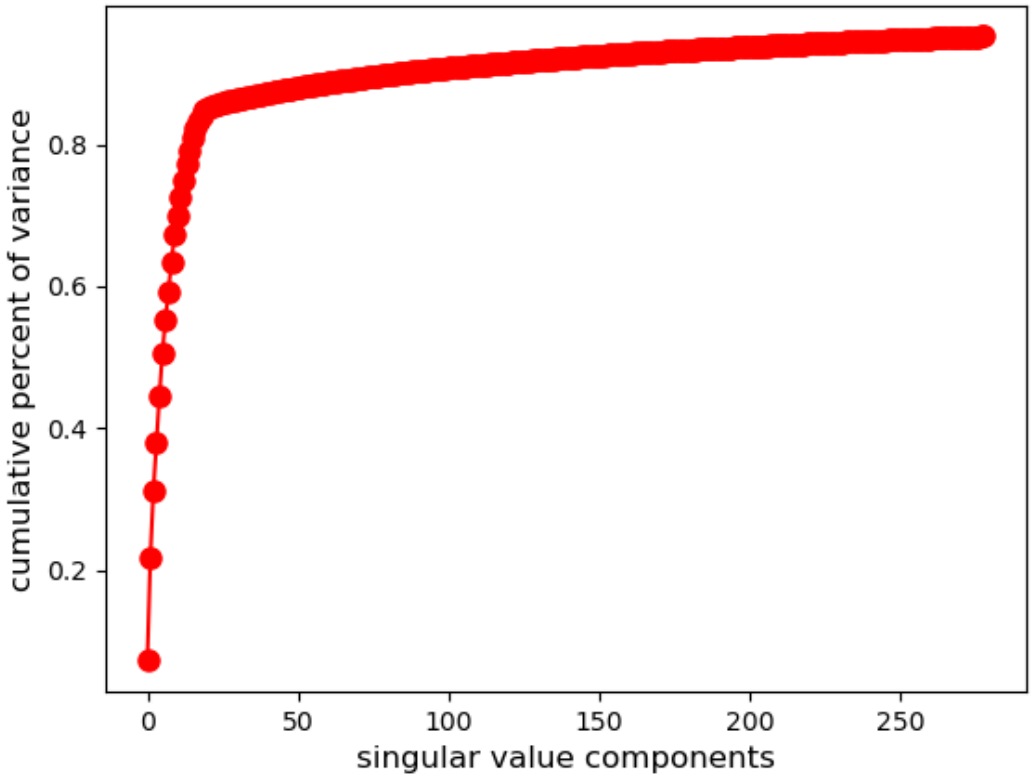
首先对数据集进行处理得到三个推荐模型并持久化存储。对于输入的用户ID，查找大于等于评分阈值的已经评分的电影。如果集合为空，说明用户没有任何一部电影有较高的评分，因此我们使用基于SVD矩阵分解的协同过滤推荐算法，预测用户没有看过的电影中评分会高的，选取前n部作为推荐。如果集合不为空，为了能够覆盖各种电影的类型，我们选取该集合中用户评分前n的电影，每部电影使用混合推荐（基于内容和物品的协同过滤）得到相似度最高的一部电影加入推荐列表。最后对推荐列表去重，得到推荐列表。



数据处理过程

基于内容的推荐

首先分别将`movies.csv`和`tags.csv`中的`genres`和`tag`合并成单词序列（空格间隔），然后使用tf-idf向量化单词序列。由于向量化后的维数过高，使用截断奇异值分解（TruncatedSVD）进行压缩，取原来维数的1/6作为压缩后向量的维度。下面展示了压缩后每个电影累积差异和维度的趋势。可以看到差异累积值接近0.9，已近很好的体现了差异，减少了计算量的同时，还保留了每部电影的内容特征。在sklearn的官方文档中提到：“In particular, truncated SVD works on term count/tf-idf matrices as returned by the vectorizers in `sklearn.feature_extraction.text`. In that context, it is known as latent semantic analysis (LSA).”因此在这里我们使用了电影的隐藏语义分析的方法。最后我们将得到的特征矩阵持久化存储。




```
class content_based_rcmd:
    '''
    基于内容的推荐
    电影的内容主要为tags和genres，我们合并两者，形成新的label，作为电影的内容。
    然后再进行向量化（使用TD-IDF），最后计算（余弦）相似度
    '''
    def __init__(self, movies_file, ratings_file, tags_file):
        self.movies_file = movies_file
        self.ratings_file = ratings_file
        self.tags_file = tags_file
        self.mat_path = "./storage/content_based.pkl"
        self.__pre_process()
    def __pre_process(self):
        movies = pd.read_csv(self.movies_file)
        movies['genres'] = movies['genres'].apply(lambda x: x.replace('|', ' '))
        tags = pd.read_csv(self.tags_file)
        tags.drop(['timestamp'], 1, inplace=True)
        mixed = pd.merge(movies, tags, on='movieId', how='left')
        mixed.fillna("", inplace=True)
        mixed = pd.DataFrame(mixed.groupby('movieId')['tag'].apply(lambda x: '
'.join(x)))
        movies_cont = pd.merge(movies, mixed, on='movieId', how='left')
        movies_cont['content'] = movies_cont[['tag', 'genres']].apply(lambda x:
' '.join(x), axis=1)
        tfidf = TfidfVectorizer(stop_words='english')
        tfidf_matrix = tfidf.fit_transform(movies_cont['content'])

        tfidf_df = pd.DataFrame(tfidf_matrix.toarray(),
index=movies_cont.index.tolist())
        # LSA（ latent semantic analysis）from sklearn doc
        compress_dim = int(tfidf_df.shape[1]/6)
```

```
        svd = TruncatedSVD(n_components=compress_dim)
        latent_mat = svd.fit_transform(tfidf_df)
        cumsum = svd.explained_variance_ratio_.cumsum()
        self.__show_compression(cumsum)
        # 使用movieId来访问df
        latent_mat_df = pd.DataFrame(latent_mat,
index=movies_cont['movieId'].tolist())
        file = open(self.mat_path, 'wb')
        pickle.dump(latent_mat_df, file)
        return
    def __show_compression(self, cumsum):
        plt.plot(cumsum, '.-', ms=16, color='red')
        plt.xlabel('singular value components', fontsize=12)
        plt.ylabel('cumulative percent of variance', fontsize=12)
        plt.show()
    def predict_top_n(self, movieId, n = 10):
        file = open(self.mat_path, 'rb')
        latent_mat_df = pickle.load(file)
        cont_vec = np.array(latent_mat_df.loc[movieId]).reshape(1,-1)
        cont_sim = cosine_similarity(latent_mat_df, cont_vec).resharpe(-1)
        cont_sim_df = pd.DataFrame({'cont_sim' : cont_sim},
latent_mat_df.index.tolist())
        cont_sim_df.sort_values('cont_sim', ascending=False, inplace=True)
        return cont_sim_df.head(n + 1)[1:].index.tolist()
    ......
```

物品的协同过滤推荐

处理方式与基于内容的推荐相似，得到movieId为index，uerId的列的特征矩阵，并持久化存储。

```
class collab_filter_rcmd:
    '''
    itemCF，用户的行为变化频率慢，且电影更新的速度不会太快，因此适合用itemCF
    '''
    def __init__(self, movies_file, ratings_file, tags_file):
        self.movies_file = movies_file
        self.ratings_file = ratings_file
        self.tags_file = tags_file
        self.mat_path = "./storage/collaborative_filter.pkl"
        self.__pre_process()
    def __pre_process(self):
        movies = pd.read_csv(self.movies_file)
        ratings = pd.read_csv(self.ratings_file)
        ratings.drop(['timestamp'], 1, inplace=True)

        ratings_merged = pd.merge(movies, ratings, on="movieId", how="left")
        ratings_mat = ratings_merged.pivot(index='movieId', columns='userId',
values='rating').fillna(0)

        # 这里用户数量小于电影数量，不需要分解降维
        file = open(self.mat_path, 'wb')
        pickle.dump(ratings_mat, file)

    def predict_top_n(self, movieId, n = 10):
        '''
        推荐top n 个相似的电影的id
```

```
        :param movieId:
        :param n:
        :return:
        '''
        file = open(self.mat_path, 'rb')
        latent_mat_df = pickle.load(file)
        collab_vec = latent_mat_df.loc[movieId].reshape(1, -1)
        collab_sim = cosine_similarity(latent_mat_df,collab_vec).reshape(-1)
        collab_sim_df = pd.DataFrame({'collab_sim' : collab_sim},
latent_mat_df.index.tolist())
        collab_sim_df.sort_values('collab_sim', ascending=False, inplace=True)
        return collab_sim_df.head(n+1)[1:].index.tolist()
    def get_collab_sim_mat(self):
        file = open(self.mat_path, 'rb')
        latent_mat_df = pickle.load(file)
        return latent_mat_df
```

混合推荐

将上述两种方式得到的相似度向量相加，获得每部电影的算数平均的相似度，再做推荐。

```
class hybrid_rcmd:
    '''
    两种方法得到的相似度进行结合，选出综合最像的电影
    '''
    def __init__(self, movies_file, ratings_file, tags_file):
        self.content_sim_mat = content_based_rcmd(movies_file, ratings_file,
tags_file).get_cont_sim_mat()
        self.collab_sim_mat = collab_filter_rcmd(movies_file, ratings_file,
tags_file).get_collab_sim_mat()
    def predict_top_n(self, movieId, n=10):
        cont_vec = np.array(self.content_sim_mat.loc[movieId]).reshape(1, -1)
        collab_vec = np.array(self.collab_sim_mat.loc[movieId]).reshape(1, -1)
        cont_sim = cosine_similarity(self.content_sim_mat, cont_vec).reshape(-1)
        collab_sim = cosine_similarity(self.collab_sim_mat,
collab_vec).reshape(-1)
        hybrid_sim = ((cont_sim + collab_sim) / 2.0)
        hybrid_sim_df = pd.DataFrame({'hybrid':hybrid_sim},
self.content_sim_mat.index.tolist())
        hybrid_sim_df.sort_values('hybrid', ascending=False, inplace=True)
        return hybrid_sim_df.head(n+1)[1:].index.tolist()
```

SVD矩阵分解的协同过滤

与物品的协同过滤相似，将movies表格和tags表格按movieId合并，得到所有的数据集。为了验证预测的准确性，我们将使用所有数据的0.2作为测试集，使用了RMSE（root mean square error，均方差根）来衡量。每次误差为0.87左右。最后再将所有数据送入训练，将算法模型持久化。

```
class svd_collab_filter_rcmd:
    def __init__(self, movies_file, ratings_file, tags_file):
        self.movies_file = movies_file
        self.ratings_file = ratings_file
        self.tags_file = tags_file
        self.algo_path = "./storage/model_svd.pkl"
        self.__pre_process()
```

```
    def __pre_process(self):
        movies = pd.read_csv(self.movies_file)
        ratings = pd.read_csv(self.ratings_file)
        ratings.drop(['timestamp'], 1, inplace=True)
        ratings_merged = pd.merge(movies, ratings, on="movieId", how="right") #
right merge，否则有的电影没有被评价
        reader = Reader(rating_scale=(1, 5))
        data = Dataset.load_from_df(ratings_merged[['userId', 'movieId',
'rating']], reader)

        trainset, testset = train_test_split(data, test_size=0.2)

        algorithm = SVD()
        algorithm.fit(trainset)
        accuracy.rmse(algorithm.test(testset))

        # 将所有数据再次训练一次
        algorithm.fit(data.build_full_trainset())
        with open(self.algo_path, 'wb') as f:
            pickle.dump(algorithm, f, pickle.HIGHEST_PROTOCOL)
        return
    def predict_usr_rating(self, userId, n=10):
        f = open(self.algo_path, 'rb')
        algorithm = pickle.load(f)
        ratings = pd.read_csv(self.ratings_file)
        mv_list = ratings[ratings.userId == userId].movieId.tolist()
        pred_list = []
        for i in mv_list:
            predicted = algorithm.predict(userId, i)
            pred_list.append((i, predicted[3]))
        pred_df = pd.DataFrame(pred_list, columns=['movieId', 'rating'])
        pred_df.sort_values('rating', ascending=False, inplace=True)
        return pred_df.head(n)['movieId'].tolist()
```

推荐结果

推荐结果放在movie_recommender_sys目录下的movie.csv中。