智能推荐系统第二次作业报告

Model-based CF for Recommended Items

宋朝芸 10215001419

Kaggle id：NaCloudy

2024 年 4 月 21 日

目录

1 算法介绍

1.1 基于模型的协同过滤算法概述

基于模型的协同过滤算法（Model-based Collaborative Filtering）是一种利用用户和物品自身特征进行推荐的算法。这种算法的思想来自于 SVD 分解，目的是将用户-物品评分矩阵 R 近似分解为用户特征矩阵 P 和物品特征矩阵 Q，其中多余的信息代表了用户和物品的特征，并利用近似结果 PQ 预测新的用户-物品对 (u,i) 的评分结果进行推荐。算法的要点为：

- 确定损失函数
- 确定优化算法
- 预测评分并排序

1. 确定损失函数

主要介绍 LFM（Latent Factor Model，隐变量模型）和 BPR（Bayesian Personalized Ranking，贝叶斯个性化排序）两种模型。

1.1 LFM 模型的思想是要尽可能近似用户-物品评分矩阵 R，因此其优化目标为

$$min_{P,Q} \quad ||R - PQ||_F^2,$$

其中 $||\cdot||_F$ 代表矩阵的 Frobenius 范数，也就是最小化最小二乘损失。这个模型是类似回归的思想。

1.2 BPR 模型的思想是将用户喜爱和不喜爱的商品尽可能区分开来，即对于用户 u、其喜爱的物品 i 和不喜爱的物品 j，希望极大化似然函数

$$max_{P,Q} \quad f(u,i,j|P,Q) = \sigma(PQ)_{ui} - (PQ)_{ik},$$

其中 $\sigma(x)$ 为 logistics 函数。这个模型是类似 SVM 判别器的思想。此外，BPR 模型更适合用于二分类的评分矩阵 R，0 代表用户不喜爱的物品，1 代表用户喜爱的物品。针对实际问题，可以将“用户是否交互过”的隐反馈矩阵作为 R，此时的假设是：用户相比于没有交互过的物品，更偏爱交互过的物品。

在这两个模型的基础上，还可以进一步细化。如评分矩阵 R 往往非常稀疏，可以在目标函数中加入对优化变量的惩罚项，以尽可能得到稀疏估计。另外，考虑用户和物品本身的特征也对评分预测有帮助，即 $f(u,i) = \alpha + \beta_u + \beta_i + P_u Q_i$，其中 α 为所有评分的基项，β_u 代表

用户 u 的评分倾向，β_i 代表物品 i 的评分倾向，这被称作 biased-LFM. 此外，考虑一阶或二阶的时间变量 t、对特征的隐变量 ρ 进行建模，甚至考虑缺失数据机制（因果推断领域，如 missing-not-at-random 非随机缺失，但需要更多的信息）都是拓展模型的方向。

2. 确定优化函数

有了损失函数之后，可以采用不同的优化算法得到参数矩阵 P,Q 的估计。比较常见的是梯度下降和最小二乘两种方法。

2.1 梯度下降法是通过每次计算损失函数的梯度，迭代参数以让梯度逐步减小，在梯度近似为 0 时认为收敛，优化完成。样本量较大时，也常用随机梯度下降法，它梯度下降法的近似实现，其实现利用了 mini-batch 的概念，即每次优化一小批次的样本，样本全部优化完为一个 epoch. 这种优化算法能够提升运算速度，但是收敛的速度可能受到影响。

2.2 最小二乘法的思想源自于简单线性模型，该模型中只有一个优化变量并能得到显式解。而基于模型的协同过滤算法需要优化 P 和 Q，最优解不再为显式解。此时可以交替给定 P、Q（即认为其中一者为常数），利用最小二乘解优化另一变量，多次循环以达到优化目的，这就是交替最小二乘法。该算法类似于 EM 算法，即同时需要优化多个目标时，拆分为分别优化单个目标，并多次迭代；梯度下降法变种之一的坐标下降法也是类似的思想。

3. 预测评分并排序

在得到近似评分矩阵 PQ 后，需要完成 top-N 的推荐任务。给定用户 u，其对物品 i 的评分预测为 $(PQ)_{u,i}$. 接下来遍历物品集合，为用户 u 预测其对所有物品的评分，并在排序后截取前 N 个评分最高的物品，即为推荐结果。

1.2 所用算法描述

本次实验主要采用基于梯度下降法的 LFM 模型，损失函数

$$min_{P,Q} \quad ||R - PQ||_F^2 + \lambda[||P||_F^2 + ||Q||_F^2].$$

输入如下，伪代码见算法 1.

- 训练数据：用户集合 U，物品集合 I，稀疏评分矩阵 R.
- 超参数：隐特征大小 k，正则化参数 λ，学习率 α，最大迭代次数 n，收敛边界 ϵ.
- 预测数据：目标用户 u 和可能的物品集合 I'，其中 $u \in U, I' \subset I$；推荐物品个数 N.

Algorithm 1 LFM with Gradient Descent

1. 随机初始化 P、Q，其中 P 的大小为 $|U|\times k$,Q 的大小为 $k\times|I|$.

2. 迭代优化

for iter = 1,2,...,n **do**

2.1 清除梯度 $grad = 0$

2.2 计算梯度值 $grad = ||P||_F^2 + ||Q||_F^2$，其中

$$\frac{\partial L}{\partial P_{u,k}} = 2((PQ)_{u,i} - R_{u,i})Q_{k,i} + \lambda * P_{u,k}, \frac{\partial L}{\partial Q_{k,i}} = 2((PQ)_{u,i} - R_{u,i})P_{u,k} + \lambda * Q_{k,i}.$$

2.3 梯度下降

$$P_{u,k} = P_{u,k} - \alpha\frac{\partial L}{\partial P_{u,k}}, Q_{k,i} = Q_{k,i} - \alpha\frac{\partial L}{\partial Q_{k,i}}.$$

2.4 判断条件

if $grad < \epsilon$ **then**

退出循环

end if

end for

3. 预测评分

for iter = 1,2,...,|I'| **do**

3.1 计算预测值 $\hat{r}_{u,iter} = (PQ)_{u,iter}$

end for

4. 排序推荐：所有预测值正序排序并取前 N 个评分，对应物品集合记为 $I_{recommend}$.

return 推荐物品集合 $I_{recommend}$;

此外实现了 BPG 模型，其损失函数为

$$max_{P,Q} \sum_{(u,i,j)\in D} \ln\sigma(PQ)_{ui} - (PQ)_{ik} + \lambda[||P||_F^2 + ||Q||_F^2],$$

其中 $D = (u,i,j)|u \in U, i,j \in I$. 算法整体流程和算法 1 一致，只是步骤 2.2 和 2.3 有所差别：

- 2.2 计算导数

$$\frac{\partial L}{\partial \theta} = \sum_{(u,i,j)\in D}\left(\frac{-e^{(P\hat{Q})_{uij}}}{1+\exp\left((P\hat{Q})_{uij}\right)}\frac{\partial(PQ)_{uij}}{\partial\theta}\right) - \lambda\theta,$$

其中

$$\frac{\partial (PQ)_{uij}}{\partial \theta} = \begin{cases} Q_{i,f} - Q_{j,f} & \text{if } \theta = P_{u,f}. \\ P_{u,f} & \text{if } \theta = Q_{i,f}, \\ -P_{u,f} & \text{if } \theta = Q_{j,f}, \\ 0 & \text{else.} \end{cases}$$

- 2.3 梯度上升（因为需要极大化）

$$\theta = \theta + \alpha \frac{\partial L}{\partial \theta}.$$

为了比较不同方法的效果，还使用了第一次作业的 user-based CF，算法描述在此略去。

2 代码说明

2.1 环境与依赖

本次实验的编程语言是 Python 3.8.18，所用库包括:

- pandas 2.0.3
- numpy 1.24.3
- sklearn 1.3.0（用于计算 NDCG 和 MSE）
- surprise 1.1.3（用于划分测试集和验证集）
- tqdm 4.65.0
- matplotlib 3.7.1

2.2 代码思路

代码的总体思路分为模型选择和结果预测两步。

模型选择时，将**训练数据**划分为**测试集**和**验证集**，在**训练集**上计算得到近似评分矩阵 PQ，在**验证集**上计算 NDCG 以选择合适的隐特征大小 k，正则化参数 λ. 在模型选择和可视化的代码上，对作业一的代码进行了复用。

结果预测时，使用选定的 λ, K，在所有**训练数据**上重新计算 PQ，并在**测试数据**上输出预测结果。

下面以 LFM 模型的实现为例说明核心代码，所用的主要函数如下：

- 加载并划分数据 `load_and_split`
- 计算隐变量模型的近似矩阵 `LFM_model`
- 计算验证集上的 NDCG 和 MSE `evaluate_model`
- 对测试集输出预测 `pred_result`

2.3 核心代码

2.3.1 load_and_split

此函数基本沿用了第一次作业的定义。首先读取训练数据和测试数据文件。为了直观性，将训练数据转为 user-item 评分矩阵，以 pandas 数据框的形式存储。

```
def load_and_split(train_path, test_path, test_size=0.2, random_state=None,
                                          choice = 0):
  # 加载csv数据集为pd数据框
  train = pd.read_csv(train_path)
  test = pd.read_csv(test_path)
  # 转为user-item矩阵
  train_df = train.pivot(index='user_id',columns='item_id',values=['rating'])
```

由于矩阵稀疏，为了便于计算 NDCG 的估计值，希望测试集和验证集都不丢失 user 和 item 的信息，因此使用 surprise 库划分数据。而 surprise 库只明确输出验证集，训练集无法使用，因此还需要遍历验证集来手动获取划分结果。

最后，为了使模型选择和结果预测两步能够复用该函数，增加选项 choice 变量。该变量取 0 表示模型选择，取 1 表示结果预测。

```
  # 模型选择步
  if (choice == 0):
    # 加载pd数据框用于surprise包
    reader = Reader(rating_scale=(1, 10))
    load_train = Dataset.load_from_df(train, reader)
    # surprise包进行数据划分
```

```
    _, validset = train_test_split(load_train, test_size=test_size, random_state
                                                =random_state)

    # 遍历validset, 更新train_df和valid_df
    for row in validset:
        userid, itemid, rate = row
        # 将valid_set中出现的值设为nan
        train_df.iloc[userid, itemid] = float('nan')

    # 为了便于计算ndsg, 转为df
    valid_df = pd.DataFrame(validset, columns=['user_id', 'item_id', 'rating'])
    valid_df = valid_df.sort_values(by=['user_id', 'rating'],ascending=[True,
                                                False])
    return train_df, valid_df, test

# 结果预测步
if(choice == 1):
    return train_df, test
```

此外，由于数据中存在 3 个缺失物品（即物品编号小于最大物品编号、大于等于 0，且没有用户与之交互过）、1 个缺失用户，影响了后续的处理。数量较少，因此简单地在训练数据前加上三行新记录，修改后的训练数据开头如下：

```
user_id,item_id,rating
0,0,3
0,1582,3
0,1653,3
1,1,5
1,2,3
```

2.3.2 LFM_model

该函数计算了 LFM 模型。在初始化用户和物品特征矩阵时，为了让结果可比较，增加了`random`选项，在模型选择时，对初始化的参数设定随机数种子。先实现算法 1 的第 1 步：

```
def LFM_model(matrix, k_size=5, max_iter=2000, alpha=0.0002, normlam=0.002,
                                        epsilon=0.001, random = 0):
    # 初始化用户特征矩阵和物品特征矩阵
    m_size = len(matrix)
    n_size = len(matrix[0])
```

```
    if(random == 0):
      np.random.seed(123)
    P = np.random.rand(m_size, k_size)
    if(random == 0):
      np.random.seed(456)
    Q = np.random.rand(n_size, k_size).T
```

然后计算梯度值，同时进行一轮梯度下降，并判断是否收敛，这是算法 1 的第 2 步：

```
    # 梯度下降，更新参数
    for _ in tqdm(range(max_iter)):
        # 清空梯度值
        grad = 0
        # 遍历所有的(user,item)对
        for u in range(m_size):
            for i in range(n_size):
                # 如果存在评分，计算评分误差（稍后将用于导数）
                if not np.isnan(matrix[u][i]):
                    error = np.dot(P[u, :], Q[:, i]) - matrix[u][i]
                    for k in range(k_size):
                        # 计算导数
                        grad_p = alpha * 2 * (error * Q[k][i] + normlam * P[u][k
                                                                            ])
                        grad_q = alpha * 2 * (error * P[u][k] + normlam * Q[k][i
                                                                            ])
                        # 进行梯度下降
                        P[u][k] = P[u][k] - grad_p
                        Q[k][i] = Q[k][i] - grad_q
                        grad = 2*grad_p + 2*grad_q
        # 判断是否达到收敛条件（梯度足够小）
        if grad < epsilon**2:
            break
```

最后返回近似矩阵 PQ 和梯度值：

```
    pred_Mat = np.dot(P,Q)
    print(grad)
    return pred_Mat, grad
```

2.3.3 evaluate_model

计算模型在验证集上的 NDCG 和 MSE 值，以便进行模型选择。考虑到 kaggle 平台上是对排序后的 item 序列计算 NDCG，此处采用同样的处理，以贴近 kaggle 平台得分。初始化评分序列和推荐物品序列为空列表：

```
def evaluate_model(valid_dataframe, pred_Mat):
    real_rank_list = valid_dataframe["item_id"].values
    real_rate_list = valid_dataframe["rating"].values
    pred_rank_list = []
    pred_rate_list = []
```

利用 PQ 矩阵`pred_Mat`，预测得分和推荐序列：

```
    for _,row in valid_dataframe.iterrows():
        # 依次计算评分
        pred_rate_list.append(pred_Mat[int(row['user_id']),int(row['item_id'])])
    # 根据kaggle平台数据提交的要求，相同user_id按pred降序、不同user_id升序，并取
                                                出物品序列
    valid_dataframe['pred'] = pred_rate_list
    valid_dataframe = valid_dataframe.sort_values(by=['user_id', 'pred'],
                                                ascending=[True, False])
    pred_rank_list = valid_dataframe["item_id"].values
```

然后利用 sklearn 库，计算 ndcg 值和 mse 值：

```
    ndcg = ndcg_score(np.asarray([real_rank_list]), np.asarray([pred_rank_list])
                                                )
    mse = mean_squared_error(np.asarray([real_rate_list]), np.asarray([
                                                pred_rate_list]))
    return ndcg, mse
```

2.3.4 pred_result

该函数实现了算法 1 的第 3 和第 4 步。对测试集上每一行，预测并统一存入数据框的 “rating” 列：

```
def pred_result(test_dataframe, pred_Mat, savepath):
    # 预测得分
    pred_list = []
    for _,row in test_dataframe.iterrows():
```

```
        pred_list.append(pred_Mat[row['user_id'],row['item_id']])#预测
    test_dataframe['rating'] = pred_list
```

然后按照要求排序，照搬`evaluate_model`函数的做法，同样是相同的 user_id 按 pred 降序、不同 user_id 升序，并按提交要求存入 csv 文件：

```
    # 按要求排序
    test_dataframe = test_dataframe.sort_values(by=['user_id', 'rating'],
                                                ascending=[True, False])
    result = test_dataframe.iloc[:,:2]
    result.reset_index(drop=True, inplace=True)
    result.insert(0, "id", result.index)
    result.to_csv(savepath,index = False,header=True)
    return result
```

3 结果分析

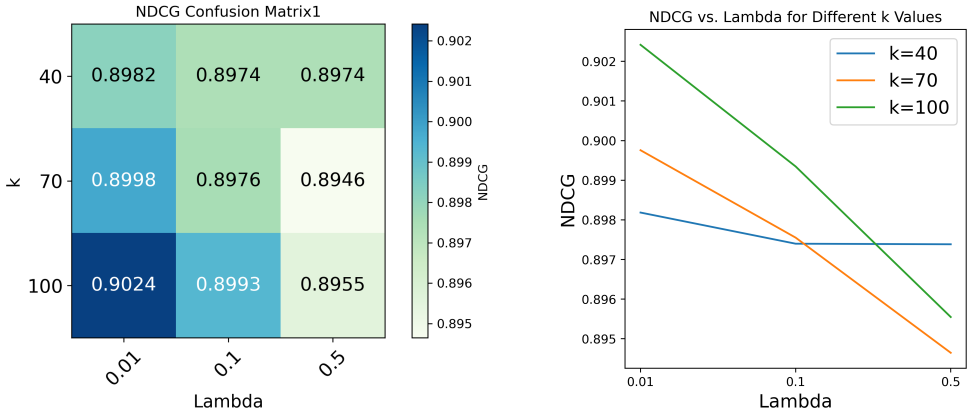
3.1 参数选择

由于数据给出了“用户对物品的评分”而非“是否交互”，LFM 模型能够对更具体的信息建模，因此下面分析 **LFM 模型**的参数选择。

LFM 模型的超参数包含：隐特征大小 k，正则化参数 λ，学习率 α，最大迭代次数 n，收敛边界 ϵ. 综合考虑到笔记本性能和运算时间，固定**学习率** $\alpha = 0.01$，**最大迭代次数** $n = 200$，**收敛边界** $\epsilon = 0.001$，因此接下来对隐特征大小 k 和正则化参数 λ 进行选择。

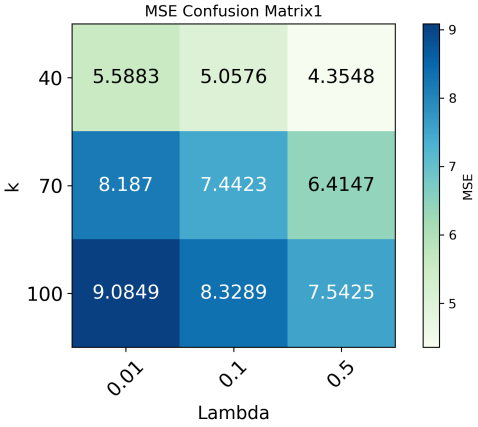
使用验证集方法估计测试集上的 NDCG 和 MSE 进行两次实验，分别绘制 NDCG 和 MSE 的热力图、折线图结果见图 1 和 2。

选择的依据是模型的 NDCG，同时也给出了评分估计 MSE 作为参考。推荐序列的 NDCG 指标取值范围在 $[0, 1]$，图中颜色越深，表示 NDCG 越大，表明模型表现越好。对于评分估计的 MSE，图中颜色越深，表示 MSE 越小，表示模型表现越好。

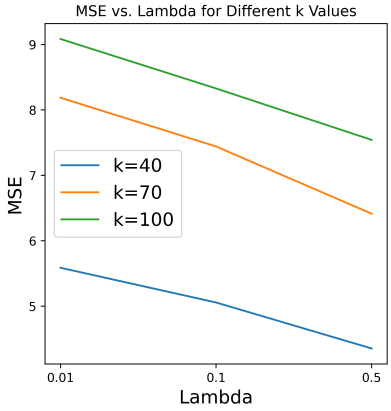


(a) 实验 1 的 NDCG 热力图

(b) 实验 1 的 NDCG 折线图



(c) 实验 1 的 MSE 热力图

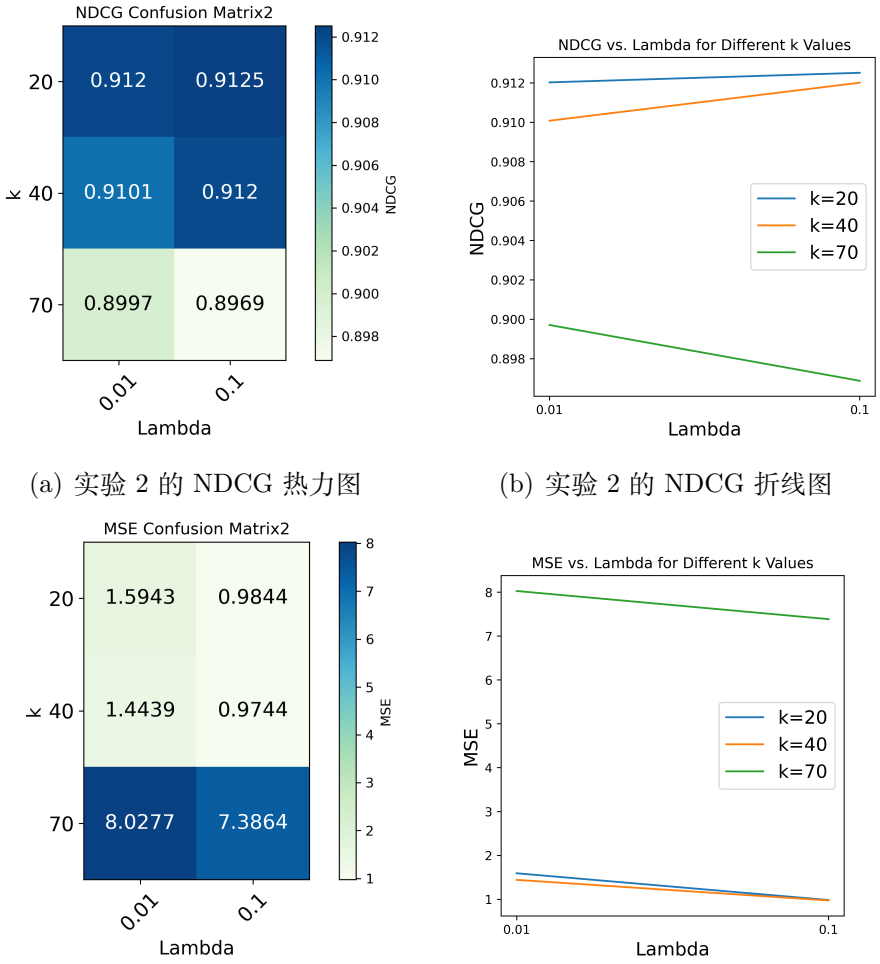


(d) 实验 1 的 MSE 折线图

图 1: LFM 实验 1 的效果

图 1 中，考虑 NDCG 指标，总体而言在 k=100 时模型的效果最好，并且对于不同的 k 值而言，都在正则化参数越大时表现越差，因此在 k=40,70,100 和 λ=0.01,0.1,0.5 的量级下，λ 越小明显对模型的提升更大，而且 K 在 100 附近更优，NDCG 达到了 0.90。但是 K=100 时的计算速度非常慢，并不理想。此外，实验 1 中 NDCG 越大（表现越好）和 MSE 越小（表现越差）竟然同时出现，而理想的最优模型应该同时满足 NDCG 值大且 MSE 值小。故实验 1 并没有找到综合效果最好的模型，需要继续实验。

进一步设计实验 2，希望找到速度更快的方法。图 2 显示 k = 20 时的 NDCG 在 0.91 以上，明显超过了实验一的最佳模型，在 MSE 上的表现也远远优于实验 1. 并且此时训练模型的时长大大降低，故选定**隐特征大小 k=20，正则化参数 λ=0.1**.



(a) 实验 2 的 NDCG 热力图 (b) 实验 2 的 NDCG 折线图

(c) 实验 2 的 MSE 热力图 (d) 实验 2 的 MSE 折线图

图 2: LFM 实验 2 的效果

3.2 预测结果

使用**隐特征大小 k=20，正则化参数** λ**=0.1，学习率** $\alpha = 0.01$，**最大迭代次数** $n = 200$，**收敛边界** $\epsilon = 0.001$ 的 LFM 模型，模型表现如下：

- 训练数据中 20% 划分为验证集时，验证集上 $\hat{\text{NDCG}} = 0.916$，本地模型训练时间为 11.55m.
- 使用完整训练数据进行训练，在 Kaggle 平台测试集上 NDCG = 0.9548（Kaggle id: NaCloudy），本地模型训练时间为 11.55m.

此外，对于 BPR 模型，训练需要更长时间（50 个 epoch 需要 20 分钟，而 200 次梯度下降需要接近一个小时），因此不对参数进行 grid 选择。将**隐特征大小 k=20，正则化参数** λ**=0.1** 的 BPR 模型提交到 kaggle 平台，NDCG 只有 0.9249，远低于 LFM 模型。

还可以将 model-based CF 与 memory-based CF 进行比较。略去参数选择部分，将近邻集合大小为 50、相似度阈值为 0.1 的 UCF 模型提交在 kaggle 平台，NDCG 值和 LFM 模型一致。对比而言，UCF 模型需要的训练时间短，测试时间为 4 分钟；而 LFM 模型的训练时间为接近 12 分钟。因此对于数据量大、需要频繁推荐、不需要频繁更新系统的情况而言，LFM 模型更合适。

4 提交文件列表

压缩包`宋朝芸_10215001419_1`包含以下这些内容:

```
project_report.pdf

prediction_results
    output_2.csv

source_code
    Confusion_Matrix_1.png  # 第1-4次实验的在验证集上的RMSE热力图
    Confusion_Matrix_2.png  #
    Confusion_Matrix_3.png  #
    Confusion_Matrix_4.png  #
    Lines_1.png             # 第1-4次实验的在验证集上的RMSE折线图
    Lines_2.png             #
    Lines_3.png             #
    Lines_4.png             #
    output_2.csv            # 测试集上预测结果
    rmse1.csv               # 第1-4次实验的在验证集上的RMSE矩阵
    rmse2.csv               #
    rmse3.csv               #
    rmse4.csv               #
    source_code.ipynb       # 源代码，jupyter notebook形式
    test.csv                # 训练数据文件
    train.csv               # 测试数据文件
```