华南理工大学

South China University of Technology

The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Jiaming Li, Junqin Lin and
Haniie Wu

Supervisor:
Qingyao Wu

Student ID：
201530611975, 201530612231
and 201530613115

Grade:
Undergraduate

December 28, 2017

Recommender System Based on Matrix Decomposition

Abstract—Recommendations can be generated by a wide range of algorithms. Matrix factorization techniques are usually effective because they allow us to discover the latent features underlying the interactions between users and items.

I. Introduction

1. Explore the construction of recommended system.
2. Understand the principle of matrix decomposition.
3. Be familiar to the use of gradient descent.
4. Construct a recommendation system under small-scale dataset, cultivate engineering ability.

II. Methods and Theory

matrix factorization is to, obviously, factorize a matrix, i.e. to find out two (or more) matrices such that when you multiply them you will get back the original matrix. from an application point of view, matrix factorization can be used to discover latent features underlying the interactions between two different kinds of entities.

In the recommendation system, there is a group of users and a set of items (movies). Given that each users have rated some items in the system, we would like to predict how the users would rate the items that they have not yet rated, such that we can make recommendations to the users.In this case, all the information we have about the existing ratings can be represented in a matrix.

Hence, the task of predicting the missing ratings can be considered as finding latent features that determine how a user rates an item.

Firstly, we have a set U of users, and a set D of items. Let $\mathbf{R}$ of size $|U| \times |D|$ be the matrix that contains all the ratings that the users have assigned to the items. Also, we assume that we would like to discover K latent features. Our task, then, is to find two matrics matrices $\mathbf{P}$ (a $|U| \times K$ matrix) and $\mathbf{Q}$ (a $|D| \times K$ matrix) such that their product approximates $\mathbf{R}$:

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{R}}$$

In this way, each row of $\mathbf{P}$ would represent the strength of the associations between a user and the features. Similarly, each row of $\mathbf{Q}$ would represent the strength of the associations between an item and the features. To get the prediction of a rating of an item d_j by u_i, we can calculate the dot product of the two vectors corresponding to u_i and d_j:

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$$

Now, we have to find a way to obtain $\mathbf{P}$ and $\mathbf{Q}$. One way to approach this problem is the first intialize the two matrices with some values, calculate how `different' their product is to $\mathbf{M}$, and then try to minimize this difference iteratively. Such a method is called gradient descent, aiming at finding a local minimum of the difference.

III. Experiment

A. Database

1. Utilizing MovieLens-100k dataset.
2. u.data -- Consisting 10,000 comments from 943 users out of 1682 movies. At least, each user comment 20 videos. Users and movies are numbered consecutively from number 1 respectively. The data is sorted randomly
3. u1.base / u1.test are train set and validation set respectively, seperated from dataset u.data with proportion of 80% and 20%. It also make sense to train set and validation set from u1.base / u1.test to u5.base / u5.test.

B. *Implementation*

Using stochastic gradient descent method

1. Use u1.base / u1.test to u5.base / u5.test directly. Populate the original scoring matrix $R_{n,users,items}$ against the raw data, and fill 0 for null values.

```
# 读取数据
train_data = read_data(r"E:\machine learning\lab4\ml-100k\ml-100k\u" + str(counter+1) + ".base")
test_data = read_data(r"E:\machine learning\lab4\ml-100k\ml-100k\u" + str(counter+1) + ".test")
```

```
def read_data(path):
    """
    读取数据
    :param path: 文件路径
    :return: 数据矩阵(943 * 1682)
    """
    ratings = [[0 for column in range(1682)] for row in range(943)]
    with open(path, 'r') as f:
        lines = f.readlines()
    for line in lines:
        id, movie, rating = line.split('\t')[:3]
        ratings[int(id)-1][int(movie)-1] = float(rating)
    return np.array(ratings)
```

2. Initialize the user factor matrix $P_{n,users,K}$ and the item (movie) factor matrix $Q_{n,items,K}$, where K is the number of potential features.

```
# 初始化用户因子矩阵和物品（电影）因子矩阵
P_user_k, Q_item_k = matrix_factorize(train_data, K)
val_data = np.dot(P_user_k, Q_item_k)
```

```
def matrix_factorize(data, k):
    """
    初始化用户因子矩阵和物品（电影）因子矩阵
    :param data: 训练集
    :param k: 潜在特征数
    :return: 用户因子矩阵和物品（电影）因子矩阵
    """
    user_num = len(data)
    item_num = len(data[0])
    P = np.random.rand(user_num, k)
    Q = np.random.rand(item_num, k)
    return P, Q.T
```

3. Determine the loss function and hyperparameter learning rate η and the penalty factor λ.

```
# 初始化参数
learning_rate = 0.01
Lambda = 0.01
K = 20
epoch = 5000
all_losses = []
all_val_losses = []
```

4. Use the gradient descent method to decompose the sparse user score matrix, get the user factor matrix and item (movie) factor matrix:

4.1 Calculate the loss gradient of all rows(columns) of user factor matrix and item factor matrix;

4.2 Use gradient descent to update all rows(columns) of $P_{n,users,K}$ and $Q_{n,items,K}$;

```
def get_loss_update(Y, Y_pre, P, Q, K, learning_rate, Lambda):
    """
    更新参数
    :param Y: 训练集结果
    :param P: 用户因子矩阵
    :param Q: 物品（电影）因子矩阵
    :param K: 潜在特征数
    :param learning_rate: 学习率
    :param Lambda: 正则化系数
    :return: 更新后的用户因子矩阵，物品（电影）因子矩阵，和预测结果矩阵
    """

    counter = 0
    for row in range(len(Y)):
        for column in range(len(Y[row])):
            if Y[row][column] > 0:
                counter += 1
                eij = Y[row][column] - np.dot(P[row, :], Q[:, column])
                for k in range(K):
                    P[row][k] = P[row][k] + learning_rate * (2 * eij * Q[k][column] - Lambda * P[row][k])
                    Q[k][column] = Q[k][column] + learning_rate * (2 * eij * P[row][k] - Lambda * Q[k][column])
    return P, Q, np.dot(P, Q)
```

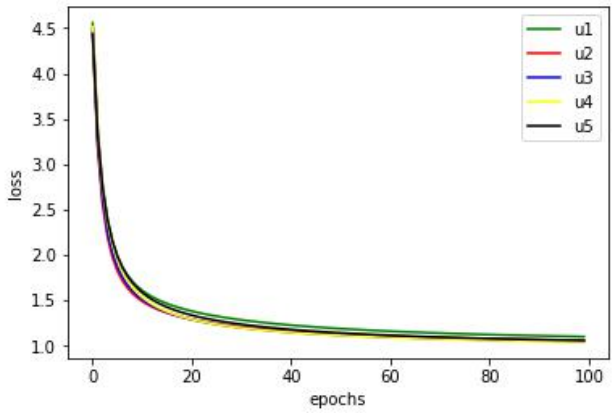
4.3 Calculate the $L_{validation}$ on the validation set, comparing with the $L_{validation}$ of the previous iteration to determine if it has converged.

```
# 计算在测试集上的loss
val_loss = get_val_loss(test_data, val_data, P_user_k, Q_item_k)
val_losses.append(val_loss)
```

```
def get_val_loss(Y, Y_pre, P, Q):
    """
    获得在测试集上的loss
    :param Y: 测试集结果矩阵
    :param Y_pre: 预测结果矩阵
    :param P: 用户因子矩阵
    :param Q: 物品（电影）因子矩阵
    :return: 测试集上的loss
    """
    loss = 0
    counter = 0
    for row in range(len(Y)):
        for column in range(len(Y[row])):
            if Y[row][column] > 0:
                counter += 1
                loss += (Y[row][column] - Y_pre[row][column]) * (Y[row][column] - Y_pre[row][column])
                for k in range(K):
                    loss += Lambda / 2 * (pow(P[row][k], 2) + pow(Q[k][column], 2))
    return loss / counter
```

5. Repeat step 4. several times, get a satisfactory user factor matrix and an item factor matrix , Draw a $L_{validation}$ curve with varying iterations.

```
# loss随迭代次数的变化图
# plt.plot(range(epoch * 2), losses, label="loss", color='red')
plt.plot(range(epoch), all_val_losses[0], label="u1", color='green')
plt.plot(range(epoch), all_val_losses[1], label="u2", color='red')
plt.plot(range(epoch), all_val_losses[2], label="u3", color='blue')
plt.plot(range(epoch), all_val_losses[3], label="u4", color='yellow')
plt.plot(range(epoch), all_val_losses[4], label="u5", color='black')
plt.xlabel("epochs")
plt.ylabel('loss')
plt.legend()
plt.show()
```



6. The final score prediction matrix $\hat{R}_{n\_users,n\_items}$ is obtained by multiplying the user factor matrix $P_{n,users,K}$ and the transpose of the item factor matrix $Q_{n,items,K}$.

IV. CONCLUSION

1. Understand the construction of recommended system.
2. Understand the principle of matrix decomposition and can apply this method to the actual project.
3. Learn the alternate least squares optimization and know more about the stochastic gradient descend, be more familiar to the use of gradient descend.
4. Be able to Construct a recommendation system under small-scale dataset, cultivate engineering ability.