Course name: Data Science(ITE4005)
Student id: 2016024793
Name: 김유진

Data Science – Term Project

IMPLEMENTATION OF RECOMMENDER SYSTEM

ENVIRONMENT

OS: Windows 10

Language: python 3.6.8

Used module: numpy(version: 1.16.2), pickle, gzip, math, sys

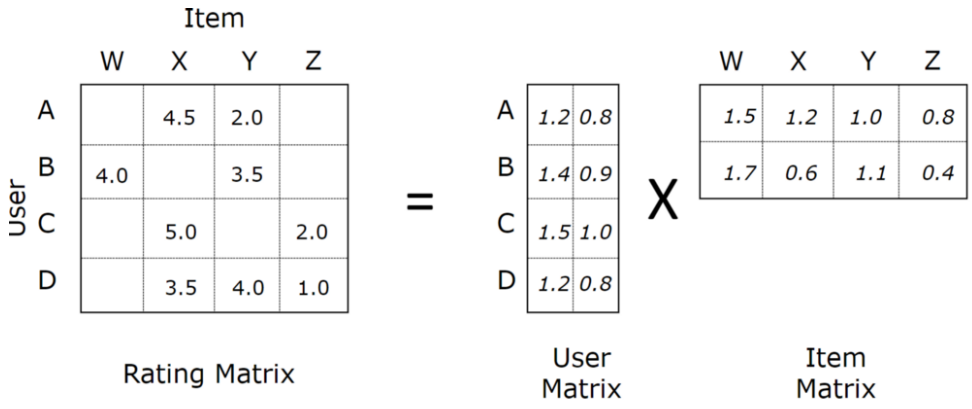
SUMMARY OF ALGORITHM

Collaborative filtering algorithm was implemented to predict the ratings of movies. A rating matrix was created using training data where each row represents each user, while each column represents different movies.

One problem that collaborative filtering approach faces is the sparsity problem. Matrix factorization technique was implied to solve this problem. Using matrix factorization, information that are not directly given can be derived. After filling up non-rated items with matrix factorization, user-based approach was used to predict the actual rating values.

DETAILED DESCRIPTIONS

1. **src/MatrixFactorization.py – filling up missing values**



When rating matrix R with size U x M is given, R can be viewed as a dot product between two matrices with each matrix having dimensions of U x K and K x M just like the above image. Also, each matrix (P and Q) could be interpreted as user-latent factor matrix and item-latent factor matrix.

So, the obective of matrix factorization is training matrix P and Q in order to fill up missing values in the rating matrix. The concept of matrix factorization can be written mathematically like below.

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

$$r_{ui} \approx \hat{r}_{ui} = p_u^T q_i = \sum_{j=1}^{k} p_{uj} q_{ji}$$

In this project, P and Q was trained with training data with minimizing errors between the given matrix R and estimated matrix (dot product of P and Q). RMSE between the actual rating value and predicted rating was used as

the cost function. Next, calcuate gradient of the cost function with respect to p_{uj} and q_{ji}, and then update each values using it. The calculation needed in gradient descent can be written like below, where β is the regression parameter and α is the learning rate.

.

$$e_{ui}^2 = (r_{ui} - \sum_{j=1}^{k} p_{uj}q_{ji})^2 + \frac{\beta}{2}\sum(\|P\|^2 + \|Q\|^2)$$

$$p'_{uj} = p_{uj} + \alpha\frac{\partial}{\partial p_{uj}}e_{ui}^2 = p_{uj} + \alpha(2e_{uj}q_{ji} - \beta p_{uj})$$

$$q'_{ji} = q_{ji} + \alpha\frac{\partial}{\partial q_{ji}}e_{ui}^2 = q_{ji} + \alpha(2e_{uj}p_{uj} - \beta q_{ji})$$

When a rating Matrix R, latent feature length k, learning rate, regression parameter, number of epochs are given, MatrixFactorization class trains itself using the calculations described above. Matrix P and Q are initialized with random values, and additionally global bias was used.

Here are some brief summaries of functions of MatrixFactorization class.

- fit(self): train matrix P and Q with given hyperparameters.
- cost(self): returns RMSE cost (described above)
- gradient_descent(self, I, j, rating, e): update P and Q using gradient descent method.
- get_prediction(self, i, j): get estimated rating of user i of item j.
- get_complete_matrix(self): returns the estimated matrix
- print_results(self): prints out some information of the matrix and hyper parmeters

2. src/user.py – user-based collaborative filtering

User-based collaborative filtering methods recommend items rated high by neighbors who have preferences similar to that of the active user. First, it finds a group of users (neighbors) whose preferences are similar to that of a target user. Then estimate the rating of target item for target user, based on the ratings given to target item by target user's neighbors.

2.1. Step 1 - Getting neighbors

Two kinds of similarity measure for users a and i used in finding neighbors was implemented. Only values that both users have were reflected when calculating similarity values.

- Pearson correlation coefficient (PCC)

$$\mathrm{w(a,i)} = \frac{\sum_j (v_{a,j} - \bar{v}_a)(v_{i,j} - \bar{v}_i)}{\sqrt{\sum_j (v_{a,j} - \bar{v}_a)^2 \sum_j (v_{i,j} - \bar{v}_i)^2}}$$

- Cosine similarity

$$\mathrm{w(a,i)} = \sum_j \frac{v_{a.j}}{\sqrt{\sum_{k\in I_a} v_{a,k}^2}} \frac{v_{i.j}}{\sqrt{\sum_{k\in I_i} v_{i,k}^2}}$$

```
def pcc(self, a, i):
        '''
        Pearson correlation coefficient
        '''
        v_a = self._R[a, :] - self._user_avg[a]
        v_i = self._R[i, :] - self._user_avg[i]
        norm_a = self._user_norm[a]
        norm_i = self._user_norm[i]

        ret = np.sum((v_a) @ (v_i).T) / (norm_a * norm_i)
        if ret < 0.:
            return 0.
        return ret

    def cosine(self, a, i):
        '''
        cosine similarity
        '''
        v_a = self._R[a, :]
        v_i = self._R[i, :]
        norm_a = self._user_norm[a]
        norm_i = self._user_norm[i]
        try:
            return np.dot(v_a,v_i) / (norm_a*norm_i)
        except:
            print("Divide by 0!")
            print("(a,i)=({},{}), norm_a: {}, norm_i: {}".format(a,i,norm_a, norm_a))
            exit()

def get_neighbors(self, target_user):

        tmp = np.zeros(self._num_users)
        for i in range(self._num_users):
            # don't contain target_user's value
            if target_user == i:
                continue

            # calculate similarity
            if self._sim == 'pcc':
                sim = self.pcc(target_user, i)
            elif self._sim == 'cosine':
                sim = self.cosine(target_user, i)
            tmp[i] = sim

        # sort neighbors and similarities in descending order
        neighbors = np.flip(np.argsort(tmp))
        similarities = np.array(tmp[neighbors])

        return neighbors, similarities
```

2.2. Step 2 – Estimate the rating by aggregating ratings on a target item given by the neighbors

Finally, predict the rating of target item of target user. The rating of target item of target user's neighbors are summed up. To reflect user's bias, the mean values of neighbors were subtracted and then target user's bias was added. The estimated rating on user c on item s could be written mathematically as

$$r_{c,s} = \bar{r}_c + k \sum_{c' \in \hat{C}} sim(c, c') \times (r_{c',s} - \bar{r}_{c'})$$

, where $\bar{r}_c$ is mean value of rated items of user c, $\hat{C}$ is the neighbors of target user c that rated on target item s, and k is $\sum_{c' \in \hat{C}} sim(c, c')$.

Lastly, if the estimated value is not proper (for example, if it is smaller than 0) then it was replaced with $\bar{r}_c$.

```
def predict(self, target_user, target_item):
        '''
        predict ratings of target_user and target_item
        '''
        if target_item >= self._num_items:
            return self._user_avg[target_user]
        elif target_user >= self._num_users:
            return np.mean(self._user_avg)

        neighbors, similarities = self._user_neighbors[target_user]

        k = list()
        p = 0
        for n, s in zip(neighbors, similarities):
            r = self._R[int(n), target_item]
            # if values of k neighbors are considered, stop
            if len(k) == self._k:
                break
            # if rating of user n is 0, skip
            if r == 0.:
                continue
            p += s * (r - self._user_avg[int(n)])
            k.append(s)
        p /= np.sum(k)
        p += self._user_avg[target_user]

        # if predicted rating is not properly calculated,
        # replace it with target_user's average rating value.
        if math.isnan(p) or math.isinf(p) or p < 0.:
            p = self._user_avg[target_user]
        return p
```

3. src/util.py – other utility functions

Other utility are implemented in this file.

```
def readFile(filename):
    '''
    Read the file and return a list
    '''
    f = open(filename, "r")
    D = list()
    while True:
        line = f.readline()
        if not line:
            break
        usr_id, item_id, rating, _ = line.strip('\n').split('\t')
        tmp = [int(usr_id), int(item_id), float(rating)]
        D.append(tmp)
    f.close()
    return D

def toMat(data):
    '''
    Make rating matrix using data
    rows represent users, and columns represent items
    '''
    users, items, ratings = [], [], []
    for u, i, r in data:
        users.append(int(u))
        items.append(int(i))
        ratings.append(r)
    num_users = (max(set(users)))
    num_items = (max(set(items)))
    A = np.zeros((num_users, num_items)).astype(np.float64)

    # print("{} users, {} movies".format(num_users, num_items))
    for u, i, r in data:
        try:
            A[int(u)-1][int(i)-1] = r
        except:
            print("u: {}, i: {}".format(u,i))
    return A

def printResult(filename, result):
    '''
    Print the predictions on a file
    '''
    filename = filename.split('/')[-1]
    outputfilename = filename + '_prediction.txt'
    f = open(outputfilename, "w")
    for user_id, item_id, rating in result:
        f.write('{}\t{}\t{}\n'.format(user_id, item_id, rating))
    f.close()
```

4. recommender.py – recommender system

The recommender system loads the pickle file where the result of matrix factorization is saved. If the corresponding pickle file is not reachable, MatrixFactorization class starts the training process. The estimated matrix created matrix factorization is then used in user-base collaborative filtering. Only 30 percent of users were considered as neighbors and Pearson correlation coefficient was used as similarity measure.

```
if __name__=="__main__":
    '''
    python recommender.py [training data] [test data]
    '''

    if len(sys.argv) != 3:
        print("need 2 arguments")
    else:
        base_filename = sys.argv[1]
        test_filename = sys.argv[2]
        pickle_filename = base_filename

        D = readFile(base_filename)
        A = toMat(D)

        pickle_filename = base_filename.split('/')[1]
        pickle_filename = 'model/' + pickle_filename.split('.')[0] + '.pkl'

        try:
            with gzip.open(pickle_filename, 'rb') as f:
                factorizer = pickle.load(f)
        except:
            print('\nmodel {} not found. Start training...'.format(pickle_filename))
            factorizer = MatrixFactorization(A, k=150, learning_rate=0.01, reg_param=0.01,
epochs=800, verbose=True)
            factorizer.fit()
            with gzip.open(pickle_filename, 'wb') as f:
                pickle.dump(factorizer, f)

        factorizer.print_results()
        matrix = factorizer.get_complete_matrix()

        num_users,_ = A.shape
        k = int(0.3 * num_users)

        userBasedpredictor = UserBasedPrediction(matrix, k=k, sim='pcc')

        T = readFile(test_filename)
        count = 0
        prediction = list()

        for user, item, _ in T:
            i = int(user)
            j = int(item)

            p = userBasedpredictor.predict(i-1, j-1)
```

```
            prediction.append([i, j, p])

            count += 1
            # if count % 1000 == 0:
            #     print("processing {}/{}".format(count, len(T)))

        printResult(base_filename, prediction)
```

5. **train/u#.pkl : results of Matrix Factorization**

These are pre-trained models using matrix factorization method. These models were trained with

- Latent feature dimension = 150
- Learning rate = 0.01
- Regularization parameter = 0.01
- 800 epochs

How to Run

Please use the command below.

python recommender.py [*base file name*] [*test file name*]

For example, `>python recommender.py data/u1.base data/u1.test`.

The file containing the result will be saved in the same directory where the program ran.

Since the program uses previously trained model, please keep the models in proper directory. If the model file is not reachable, training process will be started, and it will take about 30 minutes to complete training.

Test Results

Theses are the result of test using pretrained models in 'model' directory (epochs = 800).

data	pcc	cosine
U1	**0.9527246**	0.9622327
U2	**0.9398728**	0.9487014
U3	**0.9332089**	0.9414392
U4	**0.9291553**	0.9386919
U5	**0.9341144**	0.9413347

Using Pearson correlation coefficient performed slightly better than using cosine similarity when getting neighbors.

REFERENCES

https://towardsdatascience.com/paper-summary-matrix-factorization-techniques-for-recommender-systems-82d1a7ace74

https://medium.com/@wwwbbb8510/comparison-of-user-based-and-item-based-collaborative-filtering-f58a1c8a3f1d