The Experiment Report of Machine Learning

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
JianXiong Li , YaoYing,CaiZhe

Supervisor:
Qingyao Wu

Student ID：201721045640
201721045695 201721045701

Grade:
Graduate

December 20, 2017

Recommender System Based on Matrix Gradient Descent

.[1]Abstract—

In this experiment, we are going to complete a Recommender System Based on Matrix Decomposition. In the mathematical discipline of linear algebra, a matrix decomposition or matrix factorization is a factorization of a matrix into a product of matrices. This system will help us to recommend movies, emancipated from flood - like information.

In short, the system is based on the assumption that people who share your preferences will have the same preferences with you in the future. Another inference is based on the angle of the object - the items that the same person likes are likely to appear in another person's favorite items.

I. INTRODUCTION

Movie recommendation is based on the current hot spots and user personalized data, providing personalized movie recommendation for users, thereby increasing user stickiness and improving website traffic. It is one of the most important functions of major video sites. For online movie providers, the recommendation efficiency of online movie recommendation system will directly affect the economic performance of the company.

The motivation for this experiment is to explore the construction of recommended system, understand the principle of matrix decomposition,e familiar to the use of gradient descent and Construct a recommendation system under small-scale dataset, cultivate engineering ability.

II. METHODS AND THEORY

The experimental movie recommendation system is based on SVD and is optimized using a stochastic gradient descent method.SVD is used to compress a large matrix lossy way to reduce the number of dimensions. Matrix decomposition: In many cases, a small portion of the data carries most of the information in the dataset, while other information is either noise or unrelated information. Matrix decomposition can represent the original matrix as a new, easy-to-manipulate form that is the product of two or more matrices. This decomposition is similar to the factorization in algebra.

Here, our training matrix is an NxM matrix of user-to-movie scoring, and the value of user-unused scoring movies is either null or zero. At the heart of matrix decomposition is decomposing a very sparse scoring matrix into two matrices, one representing the properties of user and one representing the properties of the item. The inner product of each row and column of two matrices yields the corresponding score. Users of some unknown movie ratings, learn through the expected possible score, recommended to the user. The parameters are estimated by iteratively updating and predicting the values so that the errors become smaller and smaller. In this way, we hope to find a local optimal solution so that the error is within an acceptable range.

1

III. EXPERIMENT

1.Purposes:

1. Explore the construction of recommended system.
2. Understan/d 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.

2.Data sets and data analysis:

Utilizing MovieLens-100k dataset. 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 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.

3. Experimental steps

1. Read the data set and divide it (or use u1.base / u1.test to u5.base / u5.test directly). Populate the original scoring matrix R against the raw data, and fill 0 for null values.

2. Initialize the user factor matrix P and the item (movie) factor matrix Q , where K is the number of potential features.

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

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

4.1 Select a sample from scoring matrix randomly;

4.2 Calculate this sample's loss gradient of specific row(column) of user factor matrix and item factor matrix;

4.3 Use SGD to update the specific row(column) of P and Q;

4.4 Calculate Loss the on the validation set, comparing with the Loss of the previous iteration to determine if it has converged.

5. Repeat step 4. several times, get a satisfactory user factor matrix P and an item factor matrix Q.

6 . The final score prediction matrix R is obtained by multiplying the user factor matrix P and the transpose of the item factor matrix Q .

4.The selected loss function and its derivatives:

Scoring matrix p
P:User Factor Matrix
Q:Movie factor matrix

$$R_{(NM)} \approx P_{(NK)} \times Q_{(MK)}^{T} = \hat{R}_{(NM)}$$

User u estimates movie i's rating

$$\hat{r}_{ui} = p_u q_i^{T}$$

loss function:

$$\min_{q^*,p^*} \sum_{(u,i)\in K} (\hat{r}_{ui} - p_u q_i^{T})^2$$

derivatives:

p:

$$-2(\hat{r}_{ui} - p_u q_i^{T}) q_i$$

q:

$$-2(\hat{r}_{ui} - p_u q_i^{T}) p_u$$

Iteration update
learning rate γ and the penalty factor λ

$$p_u \leftarrow p_u + \gamma(e_{ui} q_i - \lambda p_u)$$

$$q_i \leftarrow q_i + \gamma(e_{ui} p_u - \lambda q_i)$$

$$e_{ui} = \hat{r}_{ui} - r_{ui}$$

5.Core code

```
def   factor2(R,       P=None,Q=None,K=2,   steps=5000,
alpha=0.0002, beta=0.02):
    """
        The matrix R is trained on the given parameters.

        :param R:  N x M matrix, about to be trained
        :param P: An initial N x K matrix
        :param Q: An initial M x K matrix
        :param K: Potential features
        :param steps:The maximum number of iterations
        :param alpha: Gradient descent rate of decline
        :param beta:  Punish the parameters

        :returns:  P 和 Q
        """

    if not P or not Q:
        P, Q = initialize(R, K)
    Q = Q.T

    rows, cols = R.shape
    for step in xrange(steps):

        eR = np.dot(P, Q)

        for i in xrange(rows):
            for j in xrange(cols):
                if R[i,j] > 0:
                    eij = R[i,j] - eR[i,j]
                    for k in xrange(K):
                        P[i,k] = P[i,k] + alpha * (2 * eij *
Q[k,j] - beta * P[i,k])
                        Q[k,j] = Q[k,j] + alpha * (2 * eij *
P[i,k] - beta * Q[k,j])

        eR = np.dot(P, Q)   # Compute dot product only
once
        e  = 0

        for i in xrange(rows):
            for j in xrange(cols):
                if R[i,j] > 0:
                    e = e + pow((R[i,j] - eR[i,j]), 2)
                    for k in xrange(K):
                        e = e + (beta/2) * (pow(P[i,k], 2) +
pow(Q[k,j], 2))
        if e < 0.001:
            break
    return P, Q.T

class Recommender(object):

    @classmethod
    def load(klass, pickle_path):
        with open(pickle_path, 'rb') as pkl:
            return pickle.load(pkl)
    def __init__(self, udata):
        self.udata = udata
        self.users = None
        self.movies = None
        self.reviews = None
        self.build_start  = None
        self.build_finish = None
        self.description  = None

        self.model       = None
        self.features     = 2
        self.steps       = 5000
        self.alpha       = 0.0002
        self.beta        = 0.02

        self.load_dataset()

    def dump(self,pickle_path):

        with open(pickle_path, 'wb' ) as pkl:
            pickle.dump(self,pkl)

    def load_dataset(self):
        '''
       Load user and movie index as an array of NxMs
    N is the number of users, M is the number of
    movies; mark this along.Look for the value of the
    matrix
        '''

        self.users = set([])
        self.movies = set([])
        for review in load_reviews(self.udata):
            self.users.add(review['userid'])
            self.movies.add(review['movieid'])

        self.users = sorted(self.users)
        self.movies = sorted(self.movies)

                self.reviews   =   np.zeros(shape=(len(self.users),
len(self.movies)))
        for review in load_reviews(self.udata):
            uid = self.users.index(review['userid'])
            mid = self.movies.index(review['movieid'])
            self.reviews[uid, mid] = review['rating']


    def build(self, output=None):

        '''
        Training model
        '''
        options  = {
            'K' :        self.features,
            'steps' :    self.steps,
            'alpha' :    self.alpha,
            'beta'  :    self.beta
        }

        self.build_start = time.time()
        nnmf = factor2
        self.P, self.Q = nnmf(self.reviews, **options)
        self.model = np.dot(self.P, self.Q.T)
        self.build_finish = time.time()

        if output :
            self.dump(output)
```

6. Experimental results

At the heart of matrix decomposition is decomposing a very sparse scoring matrix into two matrices, one representing the properties of user and one representing the properties of the item. The inner product of each row and column of two matrices yields the corresponding score. svd decomposes an arbitrary real matrix into three matrices P, R, Q, where P and Q are two orthogonal matrices, called right and left singular matrices, and R is a diagonal matrix called a singular value matrix.In this experiment, we decompose the data to get an R matrix, and finally we have to get the best P factor matrix and C factor factor matrix. Through continuous iterative optimization of gradient descent, P and Q with the smallest loss on the training set are obtained. The user on the training set did not rate some movies and multiplied the optimized P and Q to get a predictive score for an ungraded movie. By validating the set of data to verify that the score is correct, one can observe the gap between the score estimate and the actual score as the number of iterations increases.

IV. CONCLUSION

The purpose of the matrix decomposition (SVD) is to disassemble the original matrix into two matrices that are similar to the original matrix by their point technique (inner product, vector product). Collaborative filtering methods either do not work with very large data sets or do not handle very well with very few user comments (ie, when we say sparse data). The matrix decomposition method can be easily expanded linearly with the observed data. Here, our training matrix is an NxM matrix of user-to-movie scoring, and the value of user-unused scoring movies is either null or zero. We hope that by using the matrix factorization model, we can fill in the null values with the dot product as the predictive value of the movie score for the user.

The commonly used recommended system evaluation criteria is RMSE: Root Mean Square Error. We obtain the prediction eigenvalue of the test by finding the RMSEs of the true scores on the validation scores and the validation set, iteratively obtaining the optimal solutions P and Q, and multiplying P and Q to get the prediction matrix we want.