FIT5212 Assignment 2 Report

Student ID: 30367689
Student name (Kaggle name) : Kewei Sun

Task 1: Recommender System Challenge

1. Introduction

This report compares four different recommender system models: the ALS method and LMF based on matrix decomposition, and Light-FM model with only uses training set data, as well as Light-FM model that uses user and item features, and a simple neural network model.

This paper mainly compares the NDCG of various models in the validation dataset and the NDCG of half of the test set on Kaggle to evaluate the performance of each model.

2. Model and algorithm explanation

2.1 ALS (Alternating Least Squares)

In terms of collaborative filtering, the ALS algorithm is belonging to User-item collaborative filtering. The basic idea of ALS is to decompose the original matrix into sparse matrixes and evaluate the value of the missing item. This model can be used to evaluate new items recommend to users.

Since this problem is based on the feedback of users' clicks, users do not have a clear preference for the feedback items. The feedback in this form is implicit feedback, so we need introduce the confidence. r_{ij} is the rating of user i to item j.

$$p_{ij} = \begin{cases} 1 = & r_{ij} > 0 \\ 0 = & r_{ij} < 0 \end{cases}$$

And also, we should know that the implicit feedback of 0 doesn't mean the user doesn't like it at all. It could mean the user doesn't see the item or if a user clicks on an item, it does not mean that he or she likes it, which may be a mistake.

So a trust level is needed to show that the user likes an item. In general, the larger r_{ij} is, it more indicates that the user likes an item. Therefore, the variable c_{ij} is introduced to measure the trust of p_{ij}.

$$c_{ij} = 1 + \alpha r_{ij}$$

- The least squares (LS) algorithm is the basis of ALS, and it is a common machine learning algorithm. It seeks the best match of data by minimising the sum of squares of errors, and uses the least square method to find the optimal unknown data, so as to ensure the minimum error between the obtained data and the known data.

Since the number of users and items is very large, the traditional matrix decomposition method is difficult to deal with, and a user will not rating all the items, so matrix R must be a sparse matrix. In this kind of situation, we can assume that there is some relationship (we don't need to know) between the users and the items, which is the latent variable, and we can map the matrix into these dimensions, which are mathematically expressed as:

$$R_{m\times n} \approx X_{m\times k} Y^T_{nxk}$$

And we have the cost function regarding u and item v:

$$J(U,V) = \Sigma_i^m \Sigma_j^n [c_{ij}(p_{ij} - u_i v_i^T)^2 + \lambda(||u_i||^2 + ||v_j||^2)]$$

And then we can use the ALS algorithm to optimisation to solve for this cost function. The optimisation process of ALS is similar to that of the EM algorithm.

Now we fix all the other dimensions and take the derivative of one of them to obtain:

$$\begin{aligned}\frac{\partial L}{\partial x_u} &= -2\Sigma_i(r_{ui} - x_u^T y_i)y_i + 2\lambda x_u \\ &= -2\Sigma_i(r_{ui} - y_i^T x_u)y_i + 2\lambda x_u \\ &= -2Y^T r_u + 2Y^T Y x_u + 2\lambda x_u\end{aligned}$$

And then we set the derivative equal to 0, we get:

$$Y^T Y x_u + \lambda I x_u = Y^T r_u \rightarrow x_u = (Y^T Y + \lambda I)^{-1} Y^T r_u \qquad --1$$

In the same way, we calculate the partial derivative of, since X and Y are symmetric, we can get a similar conclusion:

$$y_i = (X^T X + \lambda I)^{-1} X^T r_i \qquad --2$$

The iteration process is:

1. Random generation of X and Y.
2. Fix y and update x_u using formula 1
3. Fix x, update y_i with formula 2
4. Repeat step 2&3 until convergence
5.

The disadvantages of the ALS algorithm are:

1. It's an offline algorithm.

2. Inability to accurately evaluate new users or products (Cold Start problem).

1.2 LMF (Logistic Matrix Factorisation)

Unlike most previous matrix factorisation models, LogisticMF uses a probabilistic approach instead use RMSE as its loss function.

Specifically, given an observation matrix R, it is the dot product of two other low dimension matrix X_{nxf}, Y_{mxf}, the f is the number of latent variable.

We can get the probability of user u likes item i, shown as below:

$$p(l_{ui} \mid x_u, y_i, \beta_i, \beta_j) = \frac{exp(x_i y_i^T + \beta_u + \beta_i)}{1 + exp(x_u y_i^T + \beta_u + \beta_i)}$$

The β_i, β_j here is the bias of items and users.

We also need to introduce the variable c_{ij} to measure the confidence level of p_{ij}.

We can use

$$c_{ij} = 1 + \alpha r_{ij} \text{ OR } c_{ij} = 1 + \alpha log(1 + r_{ui}\epsilon)$$

If we integrate these two, we get:

$$L(R \mid X, Y, \beta) = \Pi_{ui}(l_{ui} \mid x_u, y_i, \beta_u, \beta_i)^{\alpha r_{ui}}(1 - p(l_{ui} \mid x_u, y_i, \beta_u, \beta_i))$$

And we assume that the latent factors of the items follow a Gaussian distribution:

$$p(X \mid \sigma^2) = \Pi_u N(x_u \mid 0, \sigma_u^2 I), p(Y \mid \sigma^2) = \Pi_i N(y_i \mid 0, \sigma_u^2 I)$$

The posterior probability is:

$$logp(X, Y, \beta \mid R) = \Sigma \alpha r_{ui}(x_u y_i^T + \beta_u + \beta_i) - (1 + \alpha r_{ui}) log(1 + exp(x_u y_i^T + \beta_u + \beta_i)) - \frac{\lambda}{2} \mid x_u \mid^2 - \frac{\lambda}{2} \mid y_i \mid^2$$

Then we need to get: $argmaxX, Y, \beta logp(X, Y, \beta \mid R)$

Then we can use alternating gradient descent to optimise it:

$$\frac{\partial}{\partial x_u} = \Sigma_i \alpha r_{ui} y_i - \frac{y_i(1 + \alpha r_{ui}) exp(x_u y_i^T + \beta_u + \beta_i)}{1 + exp(x_u y_i^T + \beta_u + \beta_i)} - \lambda x_u$$

$$\frac{\partial}{\partial \beta_u} = \Sigma_i \alpha r_{ui} - \frac{(1 + \alpha r_{ui}) exp(x_u y_i^T + \beta_u + \beta_i)}{1 + exp(x_u y_i^T + \beta_u + \beta_i)}$$

The disadvantages of the LMF algorithm are:

1. Model training may time-consuming.
2. Recommendation results may not well interpretable, each dimension of the decomposed user and item matrix cannot be explained with concepts in real life, and the latent variable cannot be named with real concepts.

1.3 Light-FM

I trained two kinds of light-FM models with different input values. The relationship between the light-FM and collaborative MF model is determined by the use of user and item feature sets. If the feature sets contain only indicative variables for each user and item, LightFM is equivalent to the MF model, so the processes can referring to the two different matrix decomposition models in 1.1 and 1.2.

If feature sets also contain metadata features that are shared by at least one item or user, then LightFM extends the MF model by allowing implicit factors of the feature to explain part of the structure of user interaction.

We set U is the set of users, I is the set of items, F^u is the set of user features, F^i is the set of item features. All User-item cross pairs $(u, i) \in U \times I$ are the union of positive interaction $S+$ and negative interaction $S-$.

In this model, each feature f is parameterised through D-dimensional user and item feature embeddings e_f^U and e_f^I. Each feature can also be described by a scalar bias item (b_f^U for user features and b_f^I for item features).

We have the latent presentation of user u and item i:

$$q_u = \Sigma_{j \in f_u} e_j^U \qquad p_i = \Sigma_{j \in f_i} e_j^I$$

And the biases of them:

$$b_u = \Sigma_{j \in f_u} b_j^U \qquad b_i = \Sigma_{j \in f_i} b_j^I$$

Then we can use these vectors and bias terms to get the prediction of the model:

$$r_{ui} = f(q_u \dot{p}_i + b_u + b_i)$$

The characteristics of Light-FM:

1. Light-FM is equivalent to the MF model if we only specify variables.

2. In most time, there are fewer metadata features than users or items, this means that fewer parameters need to be estimated from the limited training data to reduce the probability of overfitting and improve the generalisation effect.
3. When encounter a cold-started problem we can use those features to estimated and make predictions.

1.4 Neural Network model

Firstly, the NCF framework can be divided into two parts:

- The embedding layer and the input layer.
- Neural collaborative filtering layer.

The embedding layer is the fully connected layer of the last layer of the afferent nerve collaborative filtering layer in the NCF framework, while the afferent embedding layer is the binary sparse vector with unique thermal coding. The process of the former representing the sparse vector of the latter as a dense vector is similar to the the LFM model.

In addition, it is easier to use content features to represent users and projects through general feature representations such as sparse binary values, so as to alleviate the impact of cold start and weak interpretability encountered by matrix decomposition.

The neural collaborative filtering layer can be a multi-layer neural network used to discover some potential structures of user-item interaction, and its prediction model can be expressed as the following formula:

$$\begin{aligned}\hat{y}_{ui} &= f(P_T v_u^U, Q_T V_i^I \mid P, Q, \theta_f) \\ &= \phi_{out}(\phi_x(\ldots \phi_2(\phi_1(P^T V_u^U, Q_T V_i^I))\ldots))\end{aligned}$$

θ_f : The parameter of user-item interaction function f.

ϕ_{out} : Mapping function of the output layer

ϕ_x: Mapping function of the x-th layer of neural collaborative filtering layer

2. Model performance and comparison

- **ALS**

Among all the above recommendation models, the ALS has the best performance after adjusted the parameters, its NDCG score reaching 0.253 validation dataset, and got a score of 0.22 in the half test set in Kaggle.

The ALS model is sensitive to the number of latent variables k, the larger of k means more accurate it will be, and the longer the calculation time will be. However, at the same time, the data set we got is relatively small. Blindly increasing the number of latent variables will only lead to overfitting, which reduces the accuracy of our recommendation.

Theoretically, every algorithm could obtain a good NDCG score after adjusting the parameters. However, due to the limited number of submissions on Kaggle, I had to only adjust the parameters of ALS to obtain the highest NDCG.

- **FM**

Compared with ALS and FM, they both carry out implicit matrix decomposition and can solve the sparse matrix decomposition problem through different algorithms. Theoretically, FM runs faster than ALS because the time complexity of the FM training step and prediction step is linear. Compared with ALS, FM's model is more general, and it can be applied to any situation with real numbers. These two models are similar, they all

based on matrix decomposition, the reason of LMF only take 0.15 is the use of parameter may different between ALS.

- **Light-FM**

The algorithm principle of Light-FM with only use training dataset is the same as the MF method. By testing different loss functions in Light-FM, we can see that LMF and Light-FM models have similar performance. I think that if I can adjust the parameters to a correct range, these two models can have the same performance as the ALS model.

The score obtained by LightFM using features of users and items is not ideal. The highest NDCG in the validation dataset with loss function BPR is only 0.059, which is inconsistent with my expectation. It may be because all our data sets are not large enough, and the prediction using features can only represent part of the time period. Or because the features does not match the training data, which will reduce the accuracy of our prediction.

- **Neural Network**

The accuracy of the neural network is only about 0.6 and is very unstable, sometimes the training error is 0.4, but in the validation set NDCG is only 0.1, this may be because I use only two hidden layers neural network, and only 10 hidden neurons, compared to the training set in the neural network is small, it is also the cause of the result is not stable. Another reason is that neural network needs a large amount of data for training, and our data set is too small to complete the training of the large-scale neural network, which is also the reason why I did not choose to use a neural network.

3. Conclusion

After comparing the NDCG score of the validation set and the half test set on Kagge, at present, ALS has the best performance with the NDCG score of 0.22. However, I think MF and Light-Fm only use the training set, can get a good score after adjusting the parameters because they are both based on the matrix decomposition method. For the Light-FM using user-item features, we cannot prove that the features are related to the training set. However, I think it will save time if features are used to predict in a large-scale recommendation system, and the accuracy can be improved by combining the user and item features with the training data. In the neural network model, due to our lack of data and use of a simple neural network, we can only get a low NDCG score.

References

Johnson, C. C. (2014). Logistic matrix factorization for implicit feedback data. Advances in Neural Information Processing Systems, 27(78), 1-9.

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. Computer, 42(8), 30-37.

Weston, J., Makadia, A., & Yee, H. (2013, May). Label partitioning for sublinear ranking. In International conference on machine learning (pp. 181-189). PMLR.

Task 2: Node Clustering in Graphs

• Introduction

In this task, I used four different clustering methods：

- Spectral clustering,
- Node clustering based on node2vec,
- Text clustering based on word2vec,
- Combination of nodes and Sentence

In order to control the dependent variable to observe different clustering methods, all the final clustering algorithms using K-means. After clustering, the clustering method based on node2vec has the highest NMI score.

• Spectral Clustering

First, we need to calculate the Laplacian matrix of graph G according to the formula, where E is the adjacency matrix of the graph and D is the degree matrix of the graph. Due to this task is a multi- clustering problem, so we need to choose multiple eigenvectors base on the eigenvalues, the clustering number is equal to 5 (label 0 to 4). Generally, we took five eigenvalues and the corresponding eigenvectors (although we can take a different number of eigenvectors, but here also in order to save running time, I set k = 5). Then apply the k-means algorithm to predict labels.

After many runs to take the average value, I found that the NMI score of the spectral clustering method could only reach about 0.06.

According to the structure of the graph G, we can see the number of nodes and the number of edges:

```
[94] print('There are', len(G.nodes()), 'nodes,', 'and', len(G.edges()), 'edges in graph G.')

     There are 36928 nodes, and 54328 edges in graph G.
```

And the number of subgraphs:

```
print('There are', i, 'numbers of sub-graph in G.')

There are 10440 numbers of sub-graph in G.
```

We can see that there is a lot of nodes we don't need to classify, and there are 10440 sub-graph in graph G, thus we can speculate that the graph has a lot of nodes that don't have an interlinked walk. the similarity matrix of an unconnected graph can not well show the relationship between the nodes, and it will affect the performance of spectral clustering, that is the reason of low NMI score of spectral clustering.

• Node clustering

Node2vec is Similar to deep-walk, generates the sequence through the random walk, then does the node embedding using those walks. But unlike Deep-walk, Node2Vec is a skewed random walk, it can get the nearest neighbour by calculating the probability of their occurrence and then carry out node embedding. In short, if two points have more

similar neighbours, the node embedding vector will be similar, and then we can apply k-means to carry out clustering.

Although Node clustering performs the best among all four clustering algorithms, with an NMI score of 0.3. But many nodes are not connected, so random walk can not find their neighbours. In addition, the relationship between documents can not be completely described by graphs (we can also use text), so the score is not very ideal. However, this clustering method can well classify those large subgraphs, so it has the highest NMI scores than the other three.

- **Text clustering**

I first did the pre-processing and tokenise step, and then used word2vec to embed words. To make every document have one corresponding embedding vector, I also embedded sentences, then apply the k-means clustering algorithm. But the text clustering method performance is very bad, only around 0.08, but also in my expectations. Because we only have the title of each document and some even only left one word, that will be significant decrease the performance of text embedding, it can be broadly understood as in clustering algorithm point of view, most of the embedding vectors are very similar, so the clustering result is poor.

- **Text clustering + Node clustering**

By combining the embedded matrices obtained in the steps of Text Clustering and Node Clustering, we can get a matrix with more features (node embedding size(64) + text embedding size (100)). Then we can apply k-mean by using both Node and Text information for clustering. In my prediction, this clustering method should have a higher NMI score than both clustering methods, but I did not expect it to be it is lower than using node clustering alone. I think this is because the text we got (title) is too little, so the text clustering is not very ideal. In addition, I chose the default embedding size 100 for text embedding, which the scores very low, and node embedding size with only 64. In the combined matrix, the proportion of text embedding is greater than the node clustering, so the text embedding will lower the NMI scores although it uses more information,