FIT 5212 Semester 1 in 2020 : Assignment 2

STUDENT NAME : SAI TEJA POTHNAK

STUDENT ID : 30323460

TUTOR : Mohammad Haqqani, [Tuesday 10am-12pm]

Contents

Task 1: Recommender Systems : 1
Collaborative Filtering : 1
Implicit Data: 2
Alternating Least Squares 2
Bayesian Personalized Ranking 3
Model 1 : Implementation of ALS Model with only Training Data: 4
Model 2 : Implementation of ALS Model by concatenating Training Data and validation Data: 4
Model 3: Implementation of BPR Model on same dataset(Train+Validation Dataset): 6
Task 2 Node Classification 6
Node2Vec 7
Model 1 : Using only Node Data and building model on node vectors : 7
Model 2 : Word2Vec Text Embedding: 8
Model 3 : Node embedding + Text Embedding 9

Task 1: Recommender Systems :

Collaborative Filtering :

This is a technique to filter out the products that customer might prefer based on the behaviour by similar customers. When we talk about collaborative filtering for recommender systems we want to solve the problem of our original matrix having million dimensions.

Instead we can use matrix factorisation to mathematically reduce dimensionality of original matrix into smaller matrix i.e. All users by all items into all items by some taste dimensions and all users into some taste dimensions.

If we could represent user as a vector of their taste value and at the same time represent item as a vector of their taste value, we can quite easily make a recom/mendation.

This can also help us find connections between users who have no specific items in common but share common tastes.

Implicit Data:

Implicit Data is the type of feedback data that is generated depending on user's behaviour with no specific actions from the user. For example, how many times a user has visited a site, time spent by the user on particular product, items that are purchased by the user. This kind of data is gathered through user interaction and contains lots of data. The downside is that most of this data is noisy and not always apparent what it means.

About the Data Sets

1. **Train Dataset** : It contains set of interactions between user id and item id. i.e. if a user interacts with an item, there will be record as rating in the dataset.

2. **Test Dataset** : This dataset contains list of users where each user is provided with a list with 100 item candidates. This is later used for comparision after generating recommended items from the model and final top 10 items are recommended.

3. **Validation Dataset** : This dataset is similar to Test dataset and is concatenated with train dataset and used for model construction and tuning.

Recommendation Models used for this Task : ALS and BPR

Alternating Least Squares

Alternating Least Squares (ALS) is a recommendation model we will use to fit our data and recommend items to each user.

Methodology :

In Als, optimization process is performed for every iteration till the matrix arrives closer to factorized representation of original data.

Initially, random values for users(U) and items(I) are assigned and using least squares method iteratively such that say in first instance it optimises U and fixes V and vice versa until both weights are optimised to reach the best approximation of original matrix. It also contains a regulariser term 'lambda' to reduce overfitting.

This algorithm minimises the loss functions and finds out the optimised user and item vectors.

Hence, by performing alternative iteration and optimisation process we generate one matrix with user vectors and one with item vectors that can be later used find similarities and also suggest item recommendations to the user.

Parameters :

1. **Confidence** : It is calculated using magnitude of the feedback. More the user spent time with the product more is the confidence.

2. Confidence is calculated by `c = 1 + alpha * r ( where alpha is rate at which the confidence increases and r is the rating information)`

3. `**lambda**` is the regularisation term as discussed earlier

4. `**factors**` is the number of latent factors to compute the model

Bayesian Personalized Ranking

It provides user with ranked list of items. In this process, instead of taking single item as training data, it takes pair of items. The optimization is performed on rank of the user. For example, let's say there are two items i1, i2. user interacted with i1 and not with i2. This approach gives i1 a positive sign anf prefers item i1 over i2. Moreover,bayesian approach maximises the posterior probability for each user. It assumes the user interacted item pair independent of every other item pair.

User specific log likelihood function is given with following assumptions:

1. Users are independent to each other.

2. Item pair association is independent to one another.

Model Development

Three Models are used and compared here:

1. Alternative Least Squared with only Training Data

2. Alternative Least Squares with both Training and Validation Data

3. Bayesian Personalized Ranking method with both Training and Validataion Dataset

Comparision is done using submitted Kaggle scores.

Methodology for recommending top 10 items for each user. Same methodology for all the three models.

Steps :

1. All the users from test data are stores **user_id**
2. A nested list **item_list** is created to store all the item corresponding to each user.
3. `.recommend()` this is an inbuilt package in implicit.als to recommend items for each user.
   Its parameters are :
   - **userid**: The userid to calculate recommendations for.{stored as `user_id` in this case}
   - **user_items**: A sparse matrix of shape(number_users,numer_items).{stored as `data_sparse_user` in our case}
   - **`N`** : Number of results to return {Size of all unique items ids in our case}

   It returns : List of itemid and corresponding score as a tuple.
   For each user we generate the set of item recommendations by giving parameters as above.
   Fetching only the list of recommended item ids from the tuple for each user

4. From the nested loop of recommended items for each user, we compare with the item list{`item_list`} from test data and append only the items that meet this condition. Finally, only top 10 from this new list are taken and stored as a nested list `top_recommendations` where each inner list contains top 10 recommendations for each user.
5. `chain.fom_iterable` is performed on the nested lists `user_lists` and `top_recommendations` to generate a new single list `final users` and `top_10_recommendations`.
6. A dataframe `recommendations_dataframe` is created with above two lists where each row corresponds to a user and a recommended item.
7. This dataframe `recommendations_dataframe` is conveted into a csv file and used for final submissions

Model 1 : Implementation of ALS Model with only Training Data:

- User item and item-user interaction matrices are created using sparse.csr_matrix()
- ALS model is built using als class of implicit package. Class is **implicit.als.AlternatingLeastSquares(factors=,regularisation=,iterations=)**
- Parameters used are :

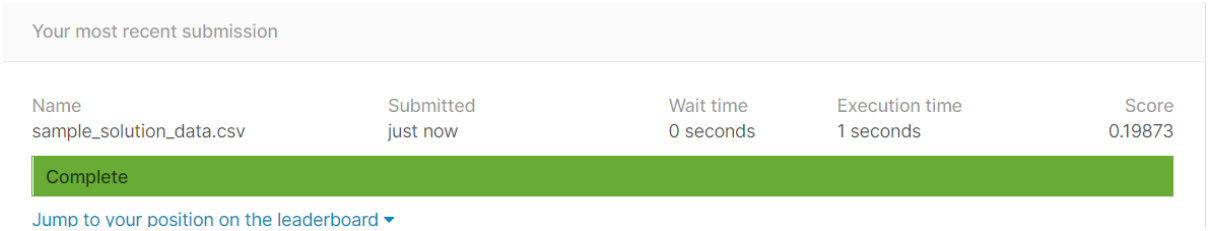
  Factors chosen is 5

  Regularisation is 0.15

  Iterations is 30

  Scaling factors worked well for values between 25 to 40. Its chosen as 35

- This model is fitted on only training Data
- Score achieved for this is lower as the ratings of the training dataset contains only 1, henceforth the data given to the model is highly biased.
- Kaggle score for this model is shown below



Model 2 : Implementation of ALS Model by concatenating Training Data and validation Data:

- It can be noticed from shape of train dataset that its size is smaller compared to test dataset and validation dataset.

- It can also be inferred that trainData set merely has only 1 as rating value, hence using only this dataset wouldn't help building a generalised model and results in overfitting as the labels(ratings) of trained data is highly biased. On the other side, validation dataset has combinations of both 0's and 1's. To achieve a generalised model with lesser overfitting, both training and validation dataset are merged.

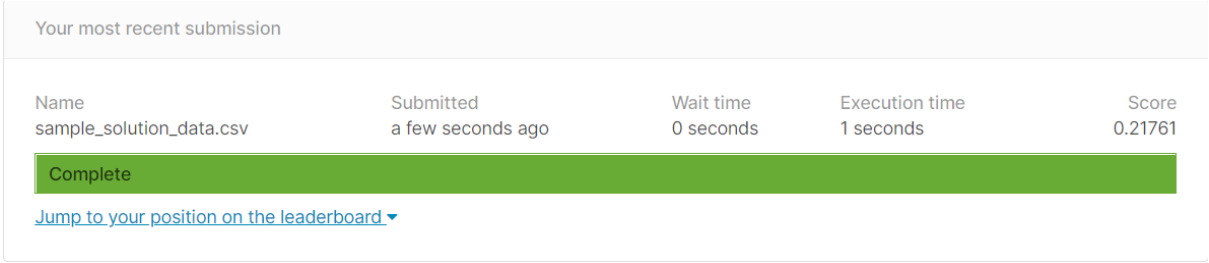
- Proceeding to concatenate both training dataset and validation dataset to achieve a bigger dataset and significantly get a better model.

- Similar steps as Model1 are performed to model to as well.
  Parameters used are :

  Factors chosen is 5, Regularisation is 0.15

  Iterations is 30, Scaling factor is : 35

- Model has improvement in the final score after adding validation dataset to train dataset.
- Kaggle score for this model is shown below
- **FINAL SUBMISSION SCORE**



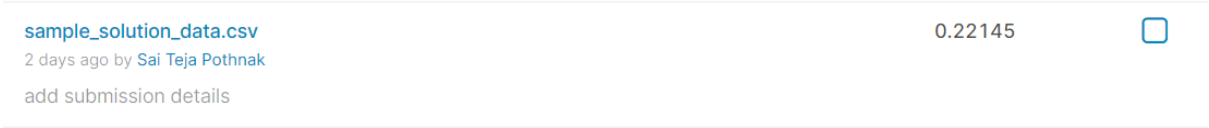
Final Submission Parameters:
The scaling factor worked well, when chosen in the range of 25-40. For my final submission, I used scaling factor as **35**.

The regularisation term is used to prevent overfitting. I chose lambda value to be **0.15** for my final submission.

As having more iterations would lead to overfitting, I have chosen iterations=**30** for my final submission.

Latent factors chosen for my final submission is **5**.

Note:
During one of my submissions, I had achieved a score of 0.22145 but unfortunately, I was not able to achieve the same score with similar parameters.



Parameters for achieving this score were :

Factors chosen is 8, Regularisation is 0.1

Iterations is 500, Scaling factor is : 25

Model 3: Implementation of BPR Model on same dataset(Train+Validation Dataset):

- Bayesian Personalised Ranking is implemented using BayesianPesonalised Ranking() function of bpr class of implicit module.
- Similar steps to above model is performed to this model as well.

  Parameters used are :

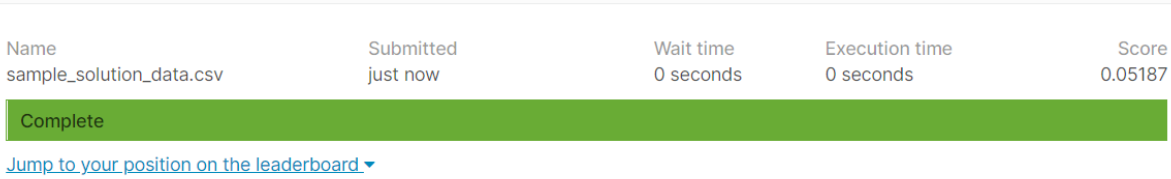
  Factors chosen is 5

  Regularisation is 0.1

  Iterations is 50

  Scaling factor is : 35

- This model performs poorly when compared to tuned ALS model.
- Kaggle score for this model is 0.05



In terms of score, ALS model built on concatenated Train and Validation Dataset seems to be the best model among all. The recommendations obtained by this model is used for my final submission.

Advantages of ALS :

- ALS is easy to parallelize
- ALS has efficient optimization techniques than others such as Stochastic Gradient Descent.

Disadvantages of ALS :

- ALS has test statistics that are not reliable when data doesn't show normal distribution.
- With ALS, model tends to overfit.

Task 2 Node Classification

About Datasets:

1. **docs.txt** : Contains title information for each node ID
2. **adjedges.txt** : Contains neighbour nodes of each node in a network. In each row, first item is a node ID and the rest items are nodes that are linked to the first item.
3. **labels.txt** : Contains class labels for each node. Each row represents a node id and its corresponding class label.

Node2Vec

In Node2Vec we learn to map nodes to a low dimensional space of features that maximises the likelihood of preserving network neighbourhood of nodes.

In node classification task, we are interested in predicting most probable labels of nodes in a network.

In prediction problem on network, one has to construct a feature vector representation of edges and nodes.

While this supervised procedure results in good accuracy, it comes at the cost of high training time complexity due to a blow up in the number of parameters that need to be estimated.

Algorithm should learn node representations that embed nodes from the same network community close to each other. And also nodes that share similar roles have similar embeddings. This would allow feature learning algorithms to generalise across a wide variety of domains and prediction tasks.

In summary, `Node2vec` is a semi-supervised learning algorithm for scalable feature learning in networks. By choosing an appropriate notion of a neighborhood, node2vec can learn representations that organise the nodes based on their network roles. We achieve this by developing a family of biased random walks, which efficiently explore neighborhoods of given node.

Initialising `Node2vec` on the citation network with the following parameters.

- **walk_length**: Number of nodes the algorithm visits in each walk
- **num_walk** : Number of times the algorithm performs the random walks
- **dimensions** : dimensions of the vector generated by the algorithm
- **workers** : number of workers the algorithm uses for parallelisation

After precomputing the probabilities and general walks, `node2vec.fit` is performed and the vector generated by node2vec algorithm and runs word2vec algorithm on top of it

Its parameters are :

- **window** : defines total number of words before and after current word used by word2vec algorithm
- **min_count** : frequency of words that need to be dropped
- **batch_words** : size of chunk sent to each worker

Model 1 : Using only Node Data and building model on node vectors :

Steps Followed :

- Reading labels dataset and doing some pre processing to create a final dataframe with two columns i.e. nodes and their corresponding labels
- Read graph in adjacency list format from path. Using **nx.read_adjlist** and creating citation network.
- Initialising **Node2vec** on the above citation network. Parameters given are : dimensions=15, walk_length=30, num_walks=20,workers=4.

- In next step. Model is fit using **node2vec.fit** with parameters as : window=10, min_count=1, batch_words=4 to use vector generated by node2vec algorithm and run word2vec on top of it.
- In the next step, Using the generated model and constructing the vectorised form of each node and storing all node vectors as a list(X). Storing all the corresponding labels as a list(y) aswell.
- Splitting the final data (that contains node vectors and their corresponding labels for each node ID) into train, test split set with stratified sampling.
- Comparing and evaluating the performance of different classification algorithms such as Logistic Regression. Bernoulli NB, Linear SVC, Random Forest Classifier.

Observations :

- By doing Node Embedding, among the above 4 classifiers, accuracy is comparitively high for **Random Forest Classifier().** Its accuracy is around 0.55.
- But, Accuracy achieved seems to be very less when node embedding is performed.

Let's leverage the **docs.txt** file to see if there is any improvement in the accuracy

This file contains title information of each node and convert into vectors in latent space.

Generate vectors for each node ID by doing Text embedding and seeing if there is any improvement in accuracy

Model 2 : Word2Vec Text Embedding:

Steps followed :

- Reading the docs file and doing some preprocessing to perform further operations and stored as dataframe that contains node id and their corresponding sentence.
- Defining a function **clean_text** to clean the sentence and convert into tokens, corresponding to each node.
- Checking if there are any empty tokens and dropping those tokens.
- Using Word2Vec model with sg=1(skip-gram) on clean text that contains tokens. Each word is converted into vector representation.
- A new column is created which contains list of vectors that are vector representations of the tokens.
- For each row, all these tokens are summed up and assuming the summed vector represents each node.
- Storing all the text vectors as a list(X_text) and their corresponding labels as list(Y-text) which are further used for model building and calculate accuracies.
- Splitting the final data (that contains text vectors and their corresponding labels for each node ID) into train, test split set with stratified sampling.
- Comparing and evaluating the performance of different classification algorithms such as Logistic Regression, Bernoulli NB, Linear SVC, Random Forest Classifier.

Observations :

- By doing Node Embedding, among the above 4 classifiers, accuracy is comparitively high for **Linear SVC().** Its accuracy is around 0.70.

- Accuracy achieved through text embedding seems to be comparatively high.

Furthermore, lets concatenate vectors generated through node embedding and text embedding for each node ID and use this final vector for model development and see if there is an improvement in accuracy.

Model 3 : Node embedding + Text Embedding

Steps :

- Concatenating vectors generated through node embedding and text embedding for each node ID.
- The concatenated vector is stored as a list(X_final) and their corresponding labels as list (Y_final) which are further used for model building and calculating accuracies.
- Comparing and evaluating the performance of different classification algorithms such as Logistic Regression, Bernoulli NB, Linear SVC, Random Forest Classifier.

Observations :

- By concatenating Text Embedding and Node Embedding, among the above 4 classifiers, accuracy is comparitively high for **Linear SVC().** Its accuracy is around 0.75.
- Model developed by concatenating node embedding vectors + text embedding vectors seems to be best among all the models.

Final Data Frame :

	Model Name	Embedding Type	Accuracy
0	Logistic Regression	Node Embedding	0.494324
1	Bernoulli NB	Node Embedding	0.366587
2	Linear SVC	Node Embedding	0.496595
3	Random Forest Classifier	Node Embedding	0.540665
4	Logistic Regression	Text Embedding	0.679165
5	Bernoulli NB	Text Embedding	0.536684
6	Linear SVC	Text Embedding	0.694231
7	Random Forest Classifier	Text Embedding	0.637549
8	Logistic Regression	Node + Text	0.728685
9	Bernoulli NB	Node + Text	0.553574
10	Linear SVC	Node + Text	0.740711
11	Random Forest Classifier	Node + Text	0.687677

Best Model and Corresponding Accuracy :

	Model Name	Embedding Type	Accuracy
10	Linear SVC	Node + Text	0.740711

For this task, different embedding methods are used, models are built and evaluations are compared. Among the built models, Linear SVC() performed better than the rest on data obtained from concatenating Node and Text Embedding. Accuracy for this is 0.74 as it can be seen above.

#the End