



FIT5212-DATA ANALYSIS FOR SEMI-STRUCTURED DATA

(ASSIGNMENT - 2)

RECOMMENDER SYSTEM & NODE CLASSIFICATION

Contents

1. Recommender Systems	3
1.1 Loading the data	3
1.2. Creating the sparse matrix and calculating sparsity.....	3
1.3 Train and test split	4
1.4. Calculate AUC using BPR	4
1.5 Building the model.....	4
1.6 Item Recommendation	5
1.7 Storing the Item Recommendations to every user	5
2. Node Classification:	6
2.1 Graph Creation	6
2.2 Node Embedding	6
2.3 Text Embedding.....	6
2.4 Ensembling the embedding results.....	6
2.5 Train and test dataset split.....	7
2.6 Model Comparison	7
2.7 Classifier Model Building & Prediction	7
2.8 Model Metrics	8
3. References	9

1. Recommender Systems

Collaborative Filtering for implicit feedback method is used in this assignment as the dataset is based on implicit feedback from the interaction of users with the items content which is being available on prior recommendations(in this case-it is the interaction between the user and the item). The user interaction between the items is fed into the recommendation engine. This part provides then an explicit rating for the interacted item. Based on this we could recommend items to the users based on their previous interactions which are recorded. Therefore, all the 3 implicit algorithms are used in this process (Alternating least squares, Logistic Matrix Factorization and Bayesian Personalized Ranking).

Our task is to build a recommendation engine that provides personalized top 10 item recommendations to users based on their interaction with the items. The recommendation system model is built using Implicit Feedback approach (depends on interactions or feedbacks on items made by the users). We would be using **Alternating Least Squares**, **Logistic Matrix Factorization** and **Bayesian Personalised Ranking** models to build and evaluate the performance of the Recommender systems.

1.1 Loading the data

Section in code – Loading the data

The data for the train data, validation data and test data read using *pandas* dataframe using *read_csv* function from the train data, validation data and test data file. The data with the rating value equal to 1 denotes that the item interacts with the user and the rating value 0 denotes it has no interaction with the user. The train data has only items which the users has an interaction with the item and validation set has the items for which the user has interaction and no interaction also. The data for which the user had interaction is filtered and combined with the train dataset. Thus, the new dataset has items for all users who have interacted.

```
Dimension of the train data set is (28449, 3)
Dimension of the test data set is (223900, 2)
Dimension of the validation data set is (223900, 3)
Dimension of the new data set is (30688, 3)
```

Fig 1.dimensions

1.2. Creating the sparse matrix and calculating sparsity

Section in code – Creating sparse Matrices & checking sparsity of the data

Here we build two sparse matrices one is used for fitting the model(item-user) and the other is used for recommendations(user-item) because the implicit library wants the input data to be in the format of sparse matrix. Sparse data in Machine Learning alters the performance and the ability of machine learning algorithms to calculate accurate predictions. Data is called sparse when some predicted values are missing in a data set, which is a common phenomenon in general large-scale analyses of data. The sparsity of the data is calculated by finding the items for which the user has interacted with and find the percentage of the items for which there was no interaction recorded by the users over the total number of items present. Here we get a sparsity of **99.36%** , which is very huge ,inferring that there are many values are sparse.

```
Sparsity : 99.36954375331017
```

Fig 2. Sparsity

1.3 Train and test split

Sections in code – [Splitting into train and test datasets](#)

A function is written, which is used in taking in the original user-item matrix and mask up a small percentage of the ratings where a user-item interaction has taken place for use as a test set. This altered list of users for with a small percentage of ratings for which the data is being masked.

Function Process

- Take a copy of the test set ratings and store it as a binary preference matrix
- Find the ratings which is equal to 1 and combine the user_id and items pairs
- Selecting random number of rows without any replacement or masking
- Get the item row index values and user column index values of the samples created in previous step
- Set the value zero to the user-item pairs which are randomly chosen and remove the zeros the sparse array
- Return the training and test set of ratings and the altered list of distinct users.

1.4. Calculate AUC using BPR

Sections in code – [Calculating the AUC using the Bayesian Personalised Ranking](#)

Bayesian Personalised Ranking is used to provide the user with a ranked list of items. The below function is used to find the area below the ROC curve - Receiver Operating Characteristic using the prediction output and actual result. The second function is to calculate the mean AUC to evaluate the recommendation System for any given user with the users with masked ratings from their user-item sparse matrix.

Function Process

- store the AUC scores for the users for which the items were removed.
- Calculate the sum of the interactions between the user and item to find the most popular items for the user.
- For every altered user, find the interactions that has not occurred
- Calculate the predicted values based on the user-item vectors
- Select all ratings from the MF prediction for this user that originally had no interaction
- Calculate the AUC for the given users
- Return the mean AUC score for user.

1.5 Building the model

Sections in code – [Finding the best parameters for the model & building the model with the best parameters](#)

The parameters used to build the model are

- alpha
- factor
- iterations
- regularization factor

Using the **Alternating Least Squares, Logistic Matrix Factorization and Bayesian Personalized Ranking** the model is being built and using **Bayesian Personalised Ranking** the AUC is calculated and the user is provided with the ranked list of items. After the model is built using ALS and data is fitted to it, we would optimise the performance using BPR by extracting the **item_vector** and **user_vector** matrix and finding the AUC score. When two models have the same AUC score, it is being decided based on the highest **NDCG** score obtained from the **kaggle** submission.

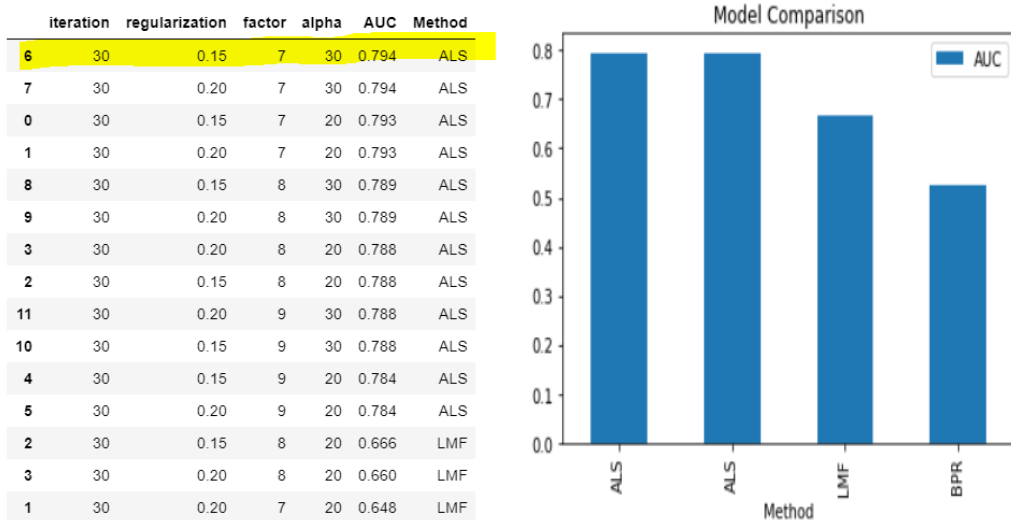


Fig 3. AUC score to find the best implicit model.

From the above evaluation we could find that the model with values for

- *alpha - 30*
- *factors - 7*
- *regularization – 0.15*
- *iterations – 30*

With these values we have built an ALS model and it is fitted with the data, which is then used to recommend the items to the users. This set of parameters got a good NDCG score in Kaggle.

1.6 Item Recommendation

Sections in code – Item Recommendation

Using the above ALS model we can recommend the items for all the users, The top 10 items are then extracted by comparing the test data provided and scores which is obtained from the **recommendation** function. Below is the items recommended for user with user_id = 0

	user_id	item_id
0	0	555
1	0	893
2	0	558
3	0	1128
4	0	156
5	0	1766
6	0	150
7	0	1862
8	0	239
9	0	1363

Fig 4 . Top 10 Item recommendation for the user with user_id = 0

1.7 Storing the Item Recommendations to every user

Sections in code – Storing the results

The top 10 items for every user is recommended and the data frame is written into a csv file, which is submitted to kaggle to find the best NDCG score.

2. Node Classification:

The given task is to classify the nodes in the given graph which is a citation network graph. The nodes in the graph are the papers, the edges in the graph is the relationship between two papers(nodes). The node classification is done based on the labels provided.

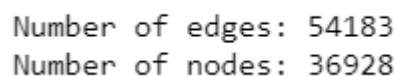
2.1 Graph Creation

Sections in code – [Creating the graph, Reading the nodes, edges, Generating the graph, Reading nodes for classification.](#)

A graph with nodes(papers) and edge (relationship between the papers). The nodes are read from the docs.txt and the title and node_id is stored in the dataframe. The adjacent edges are read from the -adjedges.txt file and the first node are the source node and the rest nodes in the line are the destination nodes.

- The lines which has one node denotes it does not have any connections with other nodes
- The lines which has 2 nodes denotes there is one source and only one destination
- The rest of the lines denote that the source node has many connections to different destination nodes

After which the list with tuples is created with the source and destination nodes to represent every connection. The nodes and edges are added to the empty graph from the node_id and list of tuples with source and destination edges without any duplicate entries using the set datatype. There are totally **54183** edges and **36928** nodes in the graphs.



```
Number of edges: 54183
Number of nodes: 36928
```

Fig 5. Graph nodes and edges

2.2 Node Embedding

Sections in code – [Node2vec Implementation](#)

Node2Vec is used in mapping the nodes to low-dimensional embeddings. Node Embedding is performed to encode nodes so that similarity in the embedding space like the dot product approximates to the similarity in the graph network. Node2Vec model is built for the graph G with 32 dimensions, 20 walk length with 50 walks. The **x** holds the 32-dimensional input features and **y** holds the corresponding target values. Upon giving various values for the parameters on trial and error basis, we can infer that the performance tends to be saturated with higher dimension values and the performance increases by increasing the number and length of the walks. Based on this inference we have chosen values as dimension to be 32, number of walks to be 50 and length to be 20. Basically, the window size is set as 10 for deep-walk and Batch-words=4 gives better vectors.

2.3 Text Embedding

Sections in code – [Text Embedding](#)

The titles of the paper(node) is being embedded using the **TF-IDF** vectorizer by preprocessing and tokenising the titles of the nodes. The stopwords in the titles are removed, document frequency is not considered as it is the title and not a big document as is set as it is generally used. Use of ngrams is not necessary as it is title.

2.4 Ensembling the embedding results

Sections in code – [stacking the embedding results](#)

Hstack function is used to horizontally stack sparse matrices of the results of text embedding and node embedding vectors.

2.5 Train and test dataset split

Sections in code – [stacking the embedding results](#)

The csr matrix is divided into train and test dataset into the **20:80** ratio and **seed** is set using the **random_state** which is an integer that sets the seed for the random number generator during the train and test split. This is because typically for node classification in any given network, we only have a small number of training data. The 3 types data (node embedding, text embedding and the stacked embedding) is split into train and test data to know the best accurate ones.

2.6 Model Comparison

Sections in code – [model comparison](#)

Four different traditional models are built and tested using Cross-validation for 5 folds to identify the best fit model for the classification task. Methods selected are SGD Classifier, SVC, Linear SVC and Random Forest. These models are built for all of the network train data and one of the model which has higher accuracy is selected as the best Classifier model. A boxplot with jitter points is plotted across the accuracy and the most accurate model will be chosen.

The cross-validation is performed for the text embedded, node embedded and finally for the full stacked (node and text embedded) datasets.

From the below graphs for the three classes amongst these Four algorithms, **Linear SVC** has the high accuracy, therefore it is best method with the highest accuracy in the trainset with least variations among five-fold cross-validation. And the accuracy is higher for the ensembled embedded network data with node and text embedding being performed. Therefore, the model is built with higher accuracy method.

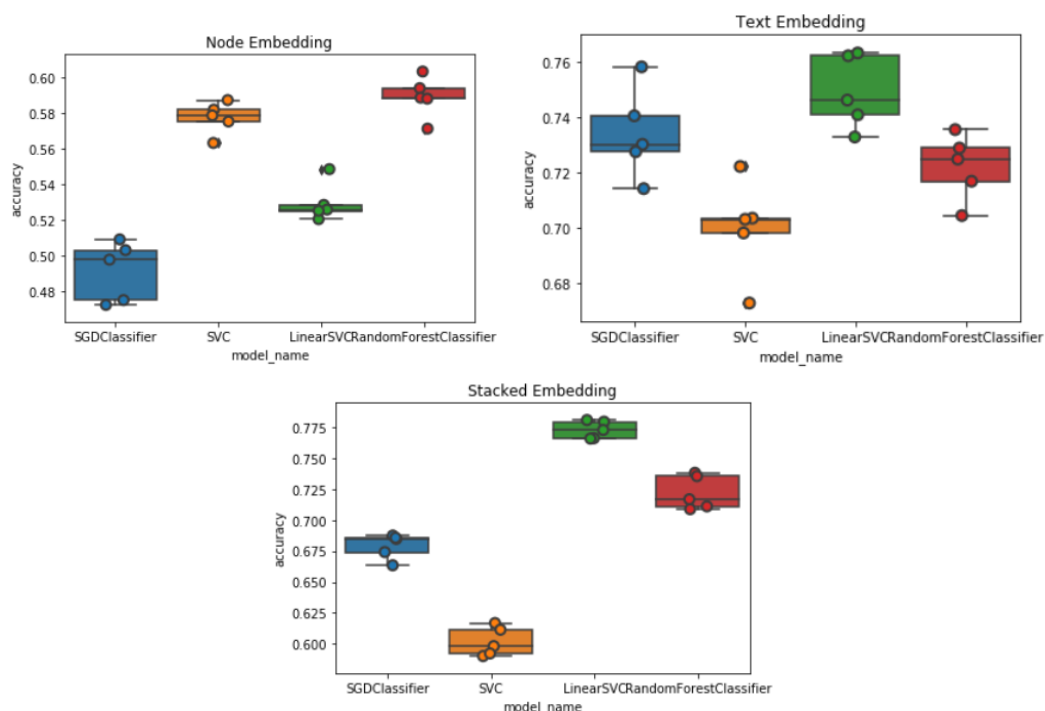


Fig 6 – Box plot for all 3 types (a)node embedding,b)text embedding,c)stacked embedding

2.7 Classifier Model Building & Prediction

Sections in code – [classifier model building & prediction](#)

Based on the all the other techniques we could infer that the Linear SVC is the most accurate model with less distortion, therefore it is used to build the node classifiers for all nodes in the network. Then with classifier model, we predict the for the test dataset.

2.8 Model Metrics

Sections in code – `model metrics`

With the true value and predicted value, we can calculate the metrics like accuracy, f1score, Matthews correlation coefficient of this classifier model. The accuracy is 78.37%. Also, the label wise classification is also generated. Recall provides us with the true positive predictions, or the sensitivity and we can find that its value is 0.744 which indicates there is good number of positive predictions. Precision provides with the exactness of the classifier model and its value is 0.78 which means it has more True positives than False positives so our model is more exact. F1 score is harmonic mean of precision and recall therefore it is very important metric to be considered while building a classifier model. When it is high and almost equal to 1 it would be a best model. For us in this case we have a 0.75, which is a very good score. The classification report below just provides us the metrics based on each of the classification labels or number of classes in which the data must be classified.

Classification Report					Confusion Matrix				
	precision	recall	f1-score	support	[[3981 85 327 98 135]				
label 0	0.74	0.86	0.80	4626	[80 2790 138 125 8]				
label 1	0.83	0.89	0.86	3141	[628 237 2539 139 46]				
label 2	0.79	0.71	0.75	3589	[110 174 101 1777 11]				
label 3	0.81	0.82	0.81	2173	[577 56 106 57 651]]				
label 4	0.76	0.45	0.57	1447	Accuracy: 0.7837873931623932				
accuracy			0.78	14976	Macro Precision: 0.7880486309685482				
macro avg	0.79	0.74	0.76	14976	Macro Recall: 0.744784406496015				
weighted avg	0.78	0.78	0.78	14976	Macro F1 score:0.7567112445083765				
					Matthews Correlation Coefficient:0.7194464325515394				

Fig 7. Metrics of the Model

3. References

RECOMMENDER SYSTEMS:

<https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada>
<https://github.com/AZG0404/Collaborative-Filtering-for-Implicit-Feedback-Datasets/blob/master/collaborative-filtering-for-implicit-feedback-datasets.ipynb>
<https://towardsdatascience.com/recommender-systems-in-practice-cef9033bb23a>
<https://github.com/luisds95/Playground/blob/master/Recommendation%20Systems/Amazon%20Finefoods%20reviews.ipynb>
<http://www.diva-portal.se/smash/get/diva2:1111045/FULLTEXT01.pdf>
<https://github.com/akhilesh-reddy/Implicit-data-based-recommendation-system/blob/master/Implicit%20data%20based%20recommendation%20system%20using%20ALS.ipynb>
<https://implicit.readthedocs.io/en/latest/lmf.html>
https://github.com/raviraju/recommender_system/tree/master/binary_recommender/articles_recommender/notebooks
<https://implicit.readthedocs.io/en/latest/bpr.html>
<https://medium.com/@deepvaghani/understanding-basic-recommendation-systems-using-python-5ee1981601e9#:~:text=AUC%20computes%20the%20area%20under,against%20the%20false%20positive%20rate.&text=A%20classifier%20with%20no%20power,on%20the%20x%3Dy%20diagonal.>
<https://stackoverflow.com/questions/41757653/how-to-compute-aucarea-under-curve-for-recommendation-system-evaluation>
<https://towardsdatascience.com/recommender-system-using-bayesian-personalized-ranking-d30e98bba0b9#:~:text=Instead%20it%20optimizes%20to%20predict,personalized%20rankings%20for%20each%20user>

NODE CLASSIFICATION:

<http://snap.stanford.edu/proj/embeddings-www/files/nrltutorial-part1-embeddings.pdf>
<https://scipython.com/book/chapter-6-numpy/examples/vstack-and-hstack/>
<https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.sparse.hstack.html>
<https://towardsdatascience.com/understanding-word-embeddings-with-tf-idf-and-glove-8acb63892032>
https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html
<https://scikit-learn.org/stable/modules/sgd.html>
<https://stackoverflow.com/questions/14616519/how-to-remove-a-list-within-list-based-on-len-in-python>
<https://kite.com/python/answers/how-to-convert-each-line-in-a-text-file-into-a-list-in-python>
<https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-and-test-datasets-using-scikit-learn-e7cf6eb5e0d>
<https://stellargraph.readthedocs.io/en/stable/demos/node-classification/node2vec-node-classification.html>