Task 1: Recommender System Challenge

On social media networks, there is an abundance of semi-structured data. This task's dataset was gathered from Flickr, an online photo-sharing social media network. Flickr allows users to share photographs and communicate with one another (friends). The goal is to recommend a list of items (pictures) to each user.

Dataset

The user-item interaction data is the main data for this challenge. This data is further split into training, validation, and test sets.

- **Training data:** The training dataset contains a set of interactions between users and items. If a user engages with an item, then there will be a record in the dataset.
- **Test data:** Each user is provided with a list of 100 candidates in the test dataset, you will need to check the candidate list and recommend the top 15 items for each user.
- **Validation data.** A validation dataset that you can use to tune your model. The dataset format is similar to the Test dataset, except that the ground truth of the rating is given. Please train your recommender systems and generate the outputs for the test data.

Introduction

Matrix factorization

The intuition behind matrix factorization is to take a large complex matrix into a smaller representation. We end up with two or more lower dimensional matrices whose product equals the original one. These dimensions are called latent or hidden features and we learn them from our data.

"If we can express each user as a vector of their taste values, and at the same time express each item as a vector of what tastes they represent. You can see we can quite easily make a recommendation."

Alternating Least Square (ALS)

ALS is used to get the matrix factorized result. It does it in a very different and in efficient manner. "ALS is an iteration optimization process where for every iteration it tries to arrive closer and closer to the factorized representation of original data."

We have our original matrix **R** of size **u x i** with our users, items, and some type of feedback data. We then want to find a way to turn that into one matrix with users and hidden features of size **u x f** and one with items and hidden features of size **f x i**. In **U** and **V,** we have weights for how each user/item relates to each feature. What we do is we calculate **U** and **V** so that their product approximates **R** as closely as possible: **R ≈ U x V**.

With the **alternating least squares *approach*** we use the same idea but iteratively alternate between optimizing **U** and fixing **V** and vice versa. We do this for each iteration to arrive closer to **R = U x V.**

Methodology

We start by reading the datasets and check for NA values and shape of dataframes.

Alternating Least Square (ALS) Recommendation Model

We will be using implicit package to provide personalised recommendation of images to the user. **Implicit feedbacks** are implied in the user actions. The underlying assumption is that **"if a user rates an item, it is an indication of his/her preference for that item"**. In our case the rating is binary, 1 for positive rating and 0 for no rating or negative rating.

I did this task on Kaggle notebook. I thought of using GPU as well and did get good results with gpu enabled environment, but I noticed that when ALS gpu is enabled the parameter "factors" can only be in the multiples of **32**. As we will see later that the ideal value by hyperparameter doesn't reveal so. So, I switched back to the CPU computations.

With the CPU enabled, I first changed the datatype of some columns to achieve relatively faster computations. The dataframe was then converted to the sparse csr matrix form. Sparse matrix is used to introduce the 0's (as we only had positive rating in our training set) and make our computations faster. Even by converting user_id and item_id to categorical datatype, it reduces computation time.

Since we have a validation set and as the Kaggle scoring is based on the NDCG evaluation, I defined my own NDCG function. The approach here will be to get scores of validation dataset and to get an idea of test dataset scorings. These scores will aid us with output files to submit on Kaggle. We also define the recommend function by referring to the tutorial code by using user vectors and item vectors, to recommend top 15 pictures for the user.

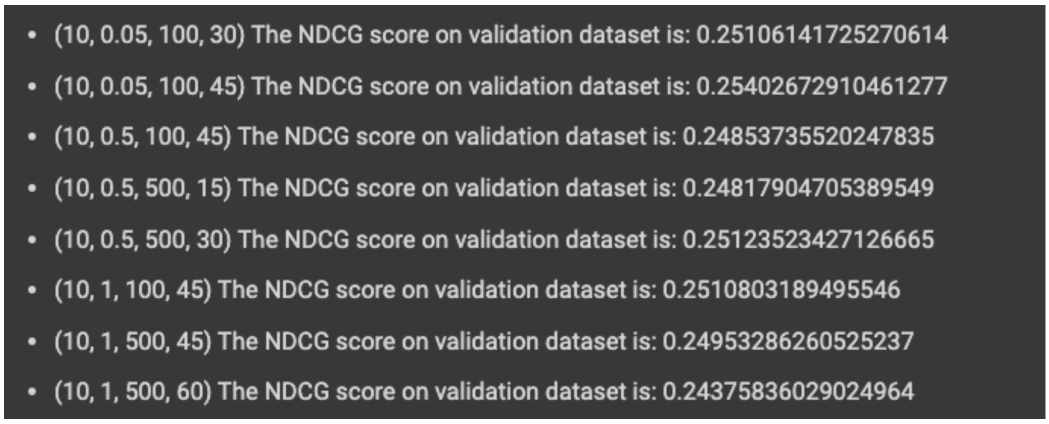
After few tests runs with hit and trial of ALS parameters, I got scores on 50% Kaggle test set around 0.18. Now, hit and trial is all about luck, so for improving why not take some computation power and get the parameters i.e., perform Hyperparameter Tuning.

Parameters

- **factors** (int, optional) – The number of latent factors to compute. Very high value can lead to overfit.
- **regularization** (float, optional) – The regularization factor to use
- **iterations** (int, optional) – The number of ALS iterations to use when fitting data
- **random_state** – random seed for reproducibility
- **alpha** – alpha reflects how much we value observed events versus unobserved events i.e. confidence

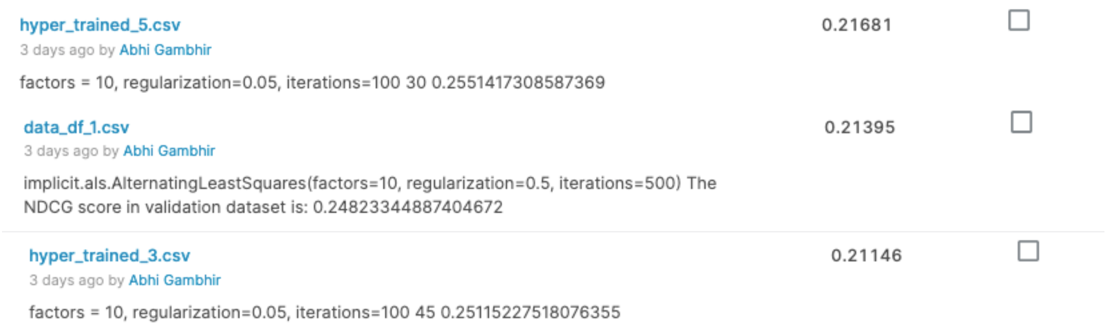
Hyperparameter Tuning

We define some ranges of values of parameters. Few runs before the final set of parameters my perception was that "regularization" parameter are generally in the ranges from 0 – 1 (like 0.05, 0.5, etc). With such parameters I achieved the following **validation NDCG scores**:



```
• (10, 0.05, 100, 30) The NDCG score on validation dataset is: 0.25106141725270614
• (10, 0.05, 100, 45) The NDCG score on validation dataset is: 0.25402672910461277
• (10, 0.5, 100, 45) The NDCG score on validation dataset is: 0.24853735520247835
• (10, 0.5, 500, 15) The NDCG score on validation dataset is: 0.24817904705389549
• (10, 0.5, 500, 30) The NDCG score on validation dataset is: 0.25123523427126665
• (10, 1, 100, 45) The NDCG score on validation dataset is: 0.2510803189495546
• (10, 1, 500, 45) The NDCG score on validation dataset is: 0.24953286260525237
• (10, 1, 500, 60) The NDCG score on validation dataset is: 0.24375836029024964
```

These are pretty good right, but the test set evaluation by Kaggle presented me with mostly scores like:



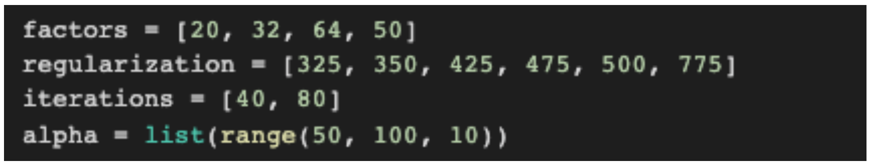
hyper_trained_5.csv
3 days ago by Abhi Gambhir
factors = 10, regularization=0.05, iterations=100 30 0.2551417308587369 — 0.21681 ☐

data_df_1.csv
3 days ago by Abhi Gambhir
implicit.als.AlternatingLeastSquares(factors=10, regularization=0.5, iterations=500) The NDCG score in validation dataset is: 0.24823344887404672 — 0.21395 ☐

hyper_trained_3.csv
3 days ago by Abhi Gambhir
factors = 10, regularization=0.05, iterations=100 45 0.25115227518076355 — 0.21146 ☐

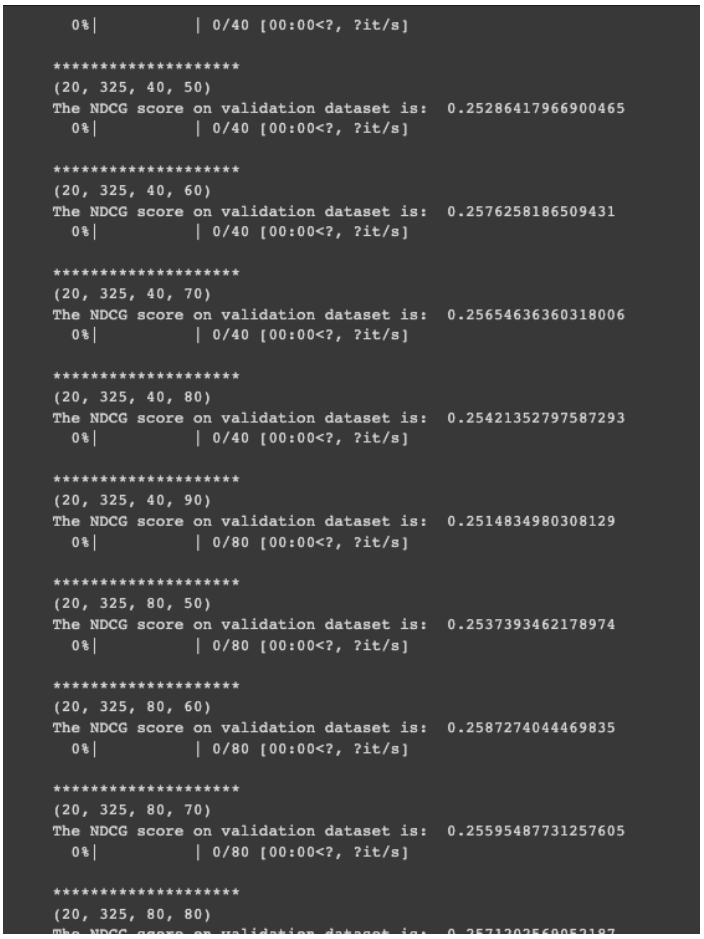
What all can be done now? I did some research about "alpha" and "regularization" parameter and came to the original research paper titled *"Collaborative Filtering for Implicit Feedback Datasets" by Yifan Hu, Yehuda Koren, & Chris Volinsky*. By going through this research my whole perspective of alpha and regularization especially for ALS algorithm changed as you will notice: "They use λ=~150λ=~150 and α=~40"

The final parameter I took for hyperparameter tuning was as follow:
I also took random_state = 624 to make tuning comparable.



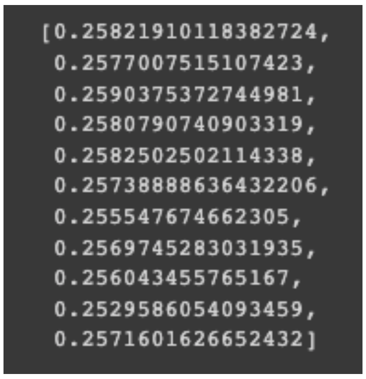
```
factors = [20, 32, 64, 50]
regularization = [325, 350, 425, 475, 500, 775]
iterations = [40, 80]
alpha = list(range(50, 100, 10))
```

Proof of Training:

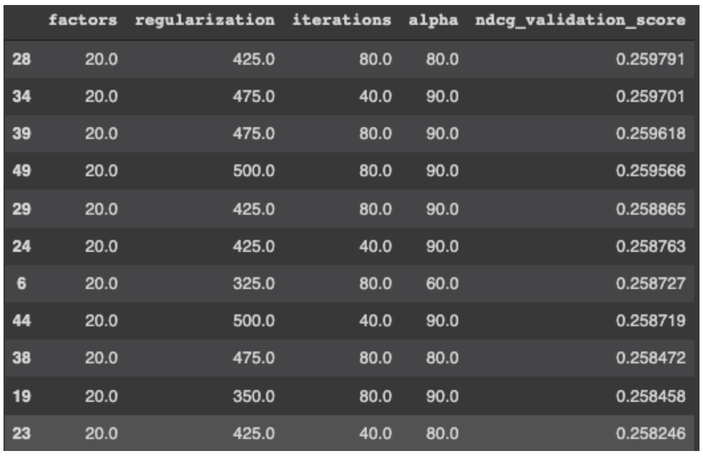


```
  0%|          | 0/40 [00:00<?, ?it/s]

********************
(20, 325, 40, 50)
The NDCG score on validation dataset is:  0.25286417966900465
  0%|          | 0/40 [00:00<?, ?it/s]

********************
(20, 325, 40, 60)
The NDCG score on validation dataset is:  0.2576258186509431
  0%|          | 0/40 [00:00<?, ?it/s]

********************
(20, 325, 40, 70)
The NDCG score on validation dataset is:  0.25654636360318006
  0%|          | 0/40 [00:00<?, ?it/s]

********************
(20, 325, 40, 80)
The NDCG score on validation dataset is:  0.25421352797587293
  0%|          | 0/40 [00:00<?, ?it/s]

********************
(20, 325, 40, 90)
The NDCG score on validation dataset is:  0.2514834980308129
  0%|          | 0/80 [00:00<?, ?it/s]

********************
(20, 325, 80, 50)
The NDCG score on validation dataset is:  0.2537393462178974
  0%|          | 0/80 [00:00<?, ?it/s]

********************
(20, 325, 80, 60)
The NDCG score on validation dataset is:  0.2587274044469835
  0%|          | 0/80 [00:00<?, ?it/s]

********************
(20, 325, 80, 70)
The NDCG score on validation dataset is:  0.25595487731257605
  0%|          | 0/80 [00:00<?, ?it/s]

********************
(20, 325, 80, 80)
The NDCG score on validation dataset is:  0.25712025680052187
```

Now since one random_state cannot let us know the exact or best NDCG score, we randomly generate 10 numbers and retrain top 11 models and store the avg NDCG of validation dataset. Please refer to the images:



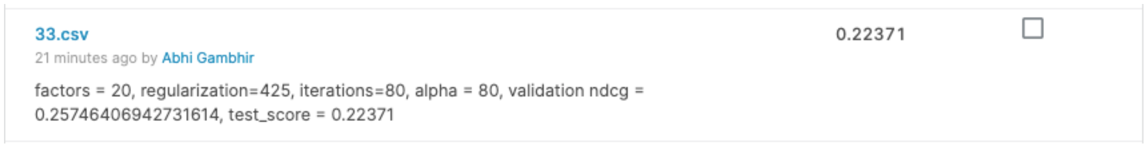
```
[0.25821910118382724,
 0.2577007515107423,
 0.2590375372744981,
 0.2580790740903319,
 0.2582502502114338,
 0.25738888636432206,
 0.255547674662305,
 0.2569745283031935,
 0.256043455765167,
 0.2529586054093459,
 0.2571601626652432]
```



	factors	regularization	iterations	alpha	ndcg_validation_score
28	20.0	425.0	80.0	80.0	0.259791
34	20.0	475.0	40.0	90.0	0.259701
39	20.0	475.0	80.0	90.0	0.259618
49	20.0	500.0	80.0	90.0	0.259566
29	20.0	425.0	80.0	90.0	0.258865
24	20.0	425.0	40.0	90.0	0.258763
6	20.0	325.0	80.0	60.0	0.258727
44	20.0	500.0	40.0	90.0	0.258719
38	20.0	475.0	80.0	80.0	0.258472
19	20.0	350.0	80.0	90.0	0.258458
23	20.0	425.0	40.0	80.0	0.258246

From the averages the best parameter we chose the following parameters and get our best public Kaggle score:

factors = 20, regularization=425, iterations=80, alpha = 80, validation ndcg = 0.25746406942731614, test_score = 0.22371

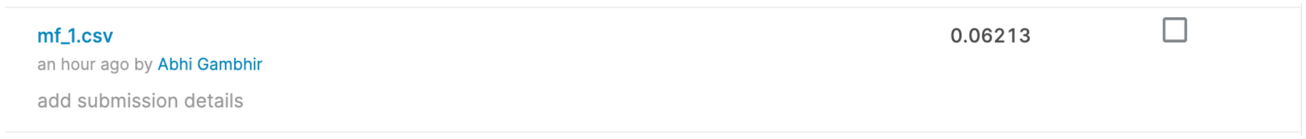


33.csv
21 minutes ago by Abhi Gambhir
factors = 20, regularization=425, iterations=80, alpha = 80, validation ndcg = 0.25746406942731614, test_score = 0.22371

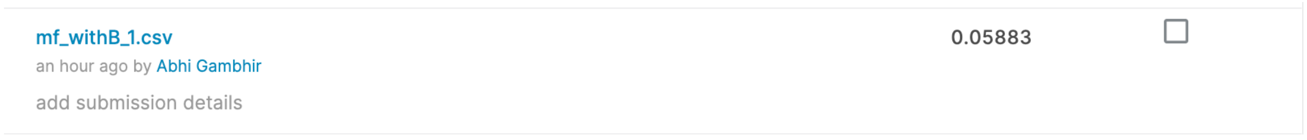
0.22371

Collaborative Filtering with Neural Networks

We will be implementing two models – **Matrix Factorization** and **Matrix Factorization with bias.** The methodology is simple, first we define functions for encoding our train and validation dataframes. Then we define the matrix factorization model with user and item embedding along with the weights. We perform the forward propagation. Finally train the model with suitable parameters. To test several runs we define train_epochs() and generate test_loss (validation loss in our case) for each iteration. The loss used was mse_loss. We then encode our test data and prepare the prediction output file for submission. Unfortunately the public NDCG score is not at all pleasant:


mf_1.csv 0.06213
an hour ago by Abhi Gambhir
add submission details

Let's give it another try by introducing bias this time. Bias terms describe the effect of one dimension on the output. For example, in the Netflix challenge example, the bias of a movie would describe how well this movie is rated compared to the average, across all movies. This depends only on the movie (as a first approximation) and does not take into account the interaction between an user and the movie. We introduce user and item bias according to its size in the model. Finally train them for different parameter and epochs. The test public results is still no better:


mf_withB_1.csv 0.05883
an hour ago by Abhi Gambhir
add submission details

Analysis & Conclusion

ALS model perform the best with scores > 0.2. It is very sophisticated algorithm, as discussed it minimizes the two loss functions alternatively. **It fixes one matrix (users or items) and runs gradient descent on other (items or users) and vice versa.**

Comparing the two matrix factorization models, we see that by introducing the bias mse_loss decreases but the NDCG got reduced for Kaggle. As we took randomly the parameters it will not be wise to state any conclusion here. But by introducing bias, generally the model starts performing better. The major reasons for poor performance of these models as compared to ALS can be that **these models are prone to weight initialization, optimization algorithm and regularization (as stated in tutorial files)**

Conceptually, by hyper parameter tunning model performances can be improved, but in the task on hand **I chose to perform that with the ALS model and get the best score.**

References

- Lecture Slides
- Tutorial Codes
- https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe
- https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4781121
- https://stats.stackexchange.com/questions/155296/alternating-least-squares-als-why-two-different-kinds-of-setting-for-lambd
- https://stats.stackexchange.com/questions/113390/the-role-of-the-bias-terms-in-matrix-factorization-formulas/113976#:~:text=Basically%20in%20a%20factorization%20model,the%20average%2C%20across%20all%20movies
- https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1#:~:text=Alternating%20Least%20Square%20(ALS)%20with%20Spark%20ML&text=Some%20high%2Dlevel%20ideas%20behind%20ALS%20are%3A&text=Its%20training%20routine%20is%20different,gradient%20descent%20with%20user%20matrix

Task 2: Node Clustering in Graphs

In the task on hand, we are provided with dataset files and have to create Graphs to perform the Node Clustering. The dataset provided is about the citation network, where each node is a paper, and an edge indicates the relationship.

Introduction

scipy.sparse.linalg.eigsh

The purpose of generating the Laplacian Matrix will be to have a symmetric matrix. Now since we have a real symmetric square matrix we can find the k eigenvalues and eigenvectors using the **scipy.sparse.linalg.eigsh()** function.

Node2Vec

Node2Vec is a node embedding algorithm whose goal is to encode each node into the embedding space and still preserve the actual network structure. It generates the vectors also called as node embeddings. Node2Vec uses skip-gram with negative sampling (SGNS) and uses random walks to generate the corpus (in our case the nodes).

K-Means Clustering

K-Means is a clustering approach in which the data is grouped into K distinct non-overlapping clusters based on their distances from the K centres. The value of **K** needs to be specified first and then the algorithm assigns the points to exactly one cluster. The idea behind the K-Means clustering approach is that the within-cluster variation amongst the point should be minimum. *Algorithm:*

1. Randomly assign K centres.
2. Calculate the distance of all the points from all the K centres and allocate the points to cluster based on the shortest distance. The model's *inertia* is the mean squared distance between each instance and its closest centroid. The goal is to have a model with the lowest inertia.
3. Once all the points are assigned to clusters, recompute the centroids.
4. Repeat the steps 2 and 3 until the locations of the centroids stop changing and the cluster allocation of the points becomes constant.

In our case we will be using **K-Means++** which is an improvement to K-Means. It introduces a smarter initialization step that tends to select centroids that are distant from one another, and this makes the K-Means algorithm much less likely to converge to a suboptimal solution.

Methodology

First, we read the docs and labels txt file and combined them into a single merged dataframe. To create the graph network, we make use of the networkx read_adjlist() to read adjedges.txt file. The graph contains **36928** nodes and **54183** edges. But when we notice number of **nodes with given true label (labels.txt) are only 18720.**

One would think to discard the nodes not available with us, but if done so we will **lose the relationship among the nodes which will ultimately influence clusters as well.** The reason we continue to train our models on full **36928** dataset.

Algorithm 1: Spectral Clustering Approach

We create the **Laplacian matrix** of the graph, so as to apply **eigsh**() function. This would return us with the **eigen values** and **eigen vectors**. As from our true label dataset we noticed there were **5 clusters (0 to 4)**, hence value for **k = 5** we pass as the argument. We find the **'SM'**: Smallest (in magnitude) eigenvalues. By storing it to variables we perform K-means++ algorithm with n_clusters =5, 50 times**. As K-means results randomly selects centroid, hence we perform the same task for 10 different random seeds and average the NMI scores.**

The point here to note is that as we discussed earlier related to less number of nodes with true labels available, we select the predicted labels for those nodes only. It has been done using **inner join** of the dataframes. The average NMI score came out to be: **0.13373921964575328**

Algorithm 2: Node2Vec

We generate the embedding vector for all nodes and take walk length=**10** and number of walks as **20**, with dimension of **32**. As it takes quite a bit time to train, we save the model using pickle and reuse it. Similar to the Algorithm 1 we perform K-means++ algorithm (for same random seeds) and filter out the predicted labels for those nodes for whom we have true labels. Finally, we pass the true and predicted labels to get the NMI scores. The average NMI score came out to be: **0.3450967480400262**

Analysis

The node clustering (node2vec) performs better, with an NMI score of **0.345**. But many nodes are not connected, so random walk concept won't be the ideal scenario. That is why we speculate not that good score even for an algorithm like node2vec. Though it still performs best in this scenario.

With spectral clustering as we saw we only have a set of nodes with true labels and the number of subgraphs is **10440**, we can say that the network lacks many edges between nodes. Due to this scenario the algorithm is **not able to find much similarity**. Hence the NMI score is quite less of **0.1337.**

Conclusion

Node2Vec with K-means performs better than Eigen Vector approach, but still, it doesn't produce the ideal NMI score. For improvement we could have also tried text clustering along with node clustering, or algorithm like **GCN (Graph Convolutional Network) model**.

References

- Lecture Slides
- Tutorial Codes
- https://towardsdatascience.com/complete-guide-to-understanding-node2vec-algorithm-4e9a35e5d147
- https://snap.stanford.edu/class/cs224w-2017/projects/cs224w-38-final.pdf
- https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.eigsh.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html#:~:text=Normalized%20Mutual%20Information%20(NMI)%20is,and%201%20(perfect%20correlation)