MIE 451/1513 Decision Support Systems
Lab and Assignment 3: Recommender Systems (RecSys)

This lab and assignment involves creating a recommender engine, evaluating it to understand its performance, and making changes to improve performance where you can.

- Programming language: Python (Google Colab Environment)
- Due Date: Posted in Syllabus

Marking scheme and requirements: Full marks will be given for (1) working, readable, reasonably efficient, documented code that achieves the assignment goals and (2) for providing appropriate answers to the questions in a Jupyter notebook (named `rs_assignment.ipynb`) committed to the student's assignment repository.

Please note the plagiarism policy in the syllabus. If you borrow or modify any *multiline* snippets of code from the web, you are required to cite the URL in a comment above the code that you used. You do not need to cite tutorials or reference content that demonstrate how to use packages – you should certainly be making use of such content.

What/how to submit your work:

- All your code should be included in a notebook named `rs_assignment.ipynb`.
- Commit and push your work to your github repository in order to submit it. Your last commit and push before the assignment deadline will be considered to be your submission. You can check your repository online to make sure that all required files have actually been committed and pushed to your repository.
- A link to create a personal repository for this assignment is posted on QUERCUS.

Credit: This lab's notebook material has been prepared based on an Advanced Scikit-Learn tutorial provided by Data Scientist Workbench.

Notes that you should pay attention to

1. The same auto-grade restrictions apply for this assignment as they did for the Python warmup assignment. Please check the pinned posts about autograder on Piazza for more information.

2. During the code review, we will ask questions about your free text answers and your results in general as well as some details about your code and possible alternative approaches you might have taken. If you made both an on-time and late submission, you will have to tell us which version we should use for code review. Note: you will receive a zero for the code review if you did not make an on-time or late submission – in this case you are considered to not have submitted your assignment.

3. Please do not use additional imports aside from the ones provided in rs_assignment.ipynb.

4. Please do NOT use global variables in your functions... they are stripped out by the auto-grader. You can **only** use parameters passed into the functions.

5. Please be aware that we only auto-grade functions highlighted ("your code goes here") in the questions. Any content outside of the functions is ignored.

6. Please do NOT rename the parameter name, function name and class name given.

7. Please check your code with the validation functions provided at the end of the notebook that perform basic input/output tests. (Please note that the validators do **not** check for correct computations, just correctly formatted output.)

8. All non-Python content / assignment answers must go in markdown cells – if your notebook fails to interpret without error from start to finish on an Google Colab instance, we cannot parse the Python in order to extract functions and auto-grade it.

9. During code review, we will use the notebook you submit to GitHub and therefore, please make sure the notebook you submit contains all the plots or tables that you want to show.

This assignment has *7 points* in total and the point allocation is shown below:

- Auto-grading points(4 points):
  - Q1: 1.5 points
  - Q2: 1 points
  - Q3: 1.5 points
- Code review (3 points)
  - Q1-Q8 code review: 2 points
  - Q8(competition): 1 point

1 Before and in the Introductory lab

In the introductory lab, you will be given an introduction to **numpy**, baseline recommendation algorithms, and evaluation methods that will be crucial for completing Assignment 4.

The recommendation dataset we will be using is from a collection called MovieLens, which contains users' movie ratings and is popular for implementing and testing recommender systems. The specific dataset we will be using for this lab is MovieLens 100K Dataset which contains 100,000 movie ratings from 943 users and a selection of 1682 movies.

Please download the lab from the course website and import it into Google Colab. You will also need to download the MovieLens data file (ml-100k.zip) from QUERCUS and upload it into the same folder with the lab ipython notebook.

Complete all sections of the lab notebook. **Understanding all parts will be critical for this assignment, so please ask TAs while in lab.**

1.1 Frequently asked questions

- *Why do we focus on ranking metrics?* Recommendations are most often meant for human consumption, where like information retrieval (IR), we focus on ranking metrics as a primary metric of evaluation.

- *Why do we measure RMSE of rating predictions?* While our primary focus in recommendation is on ranking, better RMSE scores on held-out data indicate better generalization of the learned model and often better rankings (perfect RMSE implies perfect ranking, but good RMSE is not required for good ranking – consider why). Unlike ranking, which focuses more on high-scoring items, RMSE places equal emphasis on high and low ratings.

- *How do we create a similarity function from a distance metric?* Simple: 0 distance is maximally similar and maximum (or infinite) distance is maximally dissimilar. You just have to find a function that appropriately transforms a distance to the proper similarity range (often $[0, 1]$) – note that simple negation does not achieve this transformation.

- *Why is the entry in my similarity matrix larger than 1?* If similarities are not unnormalized, this can happen. However, a similarity should *never* be negative – this is a clear sign of a bug in your code.

- *Why are most of the Cosine similarity values zero?* The cosine similarity is 0 for orthogonal vectors with no common non-zero indices. In recommendation, this would be caused by two users who never rated the same item, or two items never rated by the same user (depending on whether you are taking an item-item or user-user similarity approach).

- *Why do we need to keep the train matrix and test matrix the same shape?* Because we identify users and items by their row and column indices – these indices must be consistent and shared between the train and test matrices.

- *Why do we set test entries of predictions to zero if they are in training matrix?* In short, because train and test data should be disjoint – we only want to evaluate test entries that were not trained on. Further, do we want to recommend you to purchase something you've already purchased? Probably not.

- *Does this lab/assignment use state-of-the-art recommenders?* State-of-the-art methods are based on factorization and/or deep learning approaches, but nearest neighbor methods are still competitive and often used in industry due to their ease of implementation and modification.

- *The running time of Q5 is quite long. Is it normal?* The running time of Q5 is around 40-55 minutes if you run all the algorithms and metrics.

2 Main Assignment

In the introductory lab section, you were presented with code for data manipulation for the MovieLens data, recommendation evaluation based on RMSE and ranking metrics, and baseline recommendation algorithms consisting of (a) average user rating, (b) nearest neighbor collaborative filtering based on user-user similarity, and (c) most popular items.

Please answer the questions below and provide IPython implementations in Google Colab. For all questions that request quantitative comparisons, please report the **5-fold cross validation average and 95% confidence intervals for the 5 predefined train/test splits demonstrated in the introductory lab**. Note that most of the evaluation code is already provided in **CrossValidation**, so you just have to fill in the recommender algorithms and subroutines and call it to produce the results for each train/test split.

Q1. Data Preprocessing and Baseline algorithms

(a) Data in recommendation systems is usually encoded as data frame with three or more columns: (user, item, rating, additional meta-data if present). Complete the function **dataPreprocessor** that takes the data frame, total number of users, total number of items and it should output a **user-item matrix** as demonstrated in the lab. See the function comments for more guidance. The following experiments will all use **dataPreprocessor**.

(b) In this question, we'll port the baseline algorithms from the lab to our evaluation framework for the assignment. To do so, you need to implement the two baseline algorithms (popularity, user average rating). Please fill in the indicated functions(**popularity**, **useraverage**) in class **BaseLineRecSys**; see comments there for more guidance. The rest of **BaseLineRecSys** has been written for you.

Q2. Similarity in Collaborative Filtering

(a) In class **SimBasedRecSys**, there are two similarity measurement functions (**cosine**, **euclidean**). Please fill in the missing part of those functions. Be careful how you convert Euclidean **distance** to a $[0, 1]$ **similarity** for use in the recommender. This implementation is very short and should use pairwise_distance. (Google for "pairwise_distance scikit learn" for a list of distance metrics, more Googling will tell you what they mean.) Which metric works better? Why?

(b) Implement an additional third metric in function **somethingelse** (your choice, see other offerings of pairwise_distance) and justify in a sentence why you think this could be a good similarity metric for user or item comparison in collaborative filtering.

Q3. Collaborative Filtering

(a) Leveraging the user-user collaborative filtering example from lab, implement user-user and item-item based collaborative filtering algorithms by filling out the **predict_all** function in class **SimBasedRecSys**. Note that you should implement vectorized versions of collaborative filtering (example gives in lab) since loop-based versions will take excessively long to run.

(b) Please use the given class **CrossValidation** to report comparative RMSE results (averages and confidence intervals) between user-user and item-item based collaborative filtering for cosine similarity. Can you explain why one method may have performed better? Consider the average number of ratings per user and the average number of ratings per item when you state your answer.

Q4. Probabilistic Matrix Factorization(PMF)

(a) In class **PMFRecSys**, please fill in the missing parts in function **predict_all**:

- Initialize *self.w_Item* and *self.w_User* by sampling from $N(0, 0.1)$ The shape of *self.w_Item* is (num_item, self.num_feat) and the shape of *self.w_User* is (num_user, self.num_feat). You can use use *numpy.random.randn* for this step
- In gradient descent, we will perform gradient updates after computing local gradients over small batches of data. Each batch of data consists of parallel numpy vectors of user and item indices (remember that row and column indices are unique identifiers for users and items). As part of this calculation, you need to compute the rating each user will provide to the item in the batch. The rating predictions will go in a third parallel vector *pred_out* with size (batch_size,). The user and item indices for a batch are provided and stored in *batch_UserID* and *batch_ItemID*. Please note that these ratings are mean rating subtracted. That's why when we calculate the *rawErr*, we need to add the mean rating.
  You can use *np.sum* and *np.multiply* for this step and should not use loop in this part.
- We want to monitor performance during training so after each batch update we want to compute the predictions over all training and validation data. After PMF training is complete for each batch, calculate the ratings for all training data and validation data. Similar as last part, we provide the indices for you. They are stored in *train_user_idx*, *train_item_idx*, *val_user_idx*, *val_item_idx*. You can use *np.sum* and *np.multiply* for this step.

(b) Hyperparameter tuning:

- We already provide an instantiation of hyperparameters for you to define a PMF model. However, this model is overfitting when you train the model for 100 epochs. One way of preventing overfitting is early stopping (i.e., terminating gradient descent before convergence criteria are reached). You can adjust the *maxepoch* to avoid overfitting.
- To see the RMSE plot for training, first set the *test_mode* to True. After calling *predict_all*, you can see the plot by calling *plot_error*.
- Once you find the best *maxepoch*, remember to set the *test_mode* to False, otherwise you may get an excessive number of plots in Q5.

Q5. Performance Comparison

(a) Please use the given class **CrossValidation** to compare all the recommenders in Q1, Q2, Q3 (using cosine similarity) and Q4 on RMSE, P@k, and R-Precision. Show the cleanly formatted results of this comparison.

(b) Some baselines cannot be evaluated with some metrics? Which ones and why?

(c) What is the best algorithm for each of RMSE, P@k, and R-Precision? Can you explain why this may be?

(d) Does good performance on RMSE imply good performance on ranking metrics and vice versa? Why / why not?

Q6. Similarity Evaluation for KNN Collaborative Filtering

(a) Go through the list of movies and pick three not-so-popular movies that you know well. I.e., do not choose "Star Wars" and note that we expect everyone in the class to have chosen different movies. For each of these three movies, list the top 5 most similar movie names according to item-item cosine similarity (you might use a function like `numpy argsort`).

(b) Can you justify these similarities? Why or why not? Consider that similarity is determined indirectly by users who rated both items.

Q7. Visualization of PMF Embeddings

(a) Please use the provided function **plot_embs_one_genre** to create visualizations of the item embeddings obtained through the PMF method. When calling the function, pass the name of a genre as an argument. The function will generate plots displaying a sample of embeddings for items within that genre, as well as embeddings for items not belonging to that genre. Can you observe any discernible differences between these embeddings? If so, in which specific dimension do you notice these distinctions?

(b) Pick two genres of your interest and use the provided function **plot_embs_two_genres** to create visualizations of embeddings for samples of items within the two genres. Can you identify a considerable difference between embeddings of the two genres? Did you expect the embeddings of these genres to be different? Please explain why?

Q8. MovieLens 100k Competition

In this part, you can implement your own recommender system. We continue to use the MovieLens 100k dataset and we only care about the RPrecision in this competition. We provide you a new class called CompetitionRecSys and you need to implement the following the functions:

1. __init__: Initialization of the class. Make sure to fill out the `self.pred_column_name`, the name you give to your competition method. You can change the signature of this function.
2. predict_all: You need to train your model in this function with train_vec, the training set of the dataset. You are not allowed to change the signature of this function.
3. evaluate_test: You need to make predictions after you train your model. You only need to replace *None* with your prediction code in this line, `prediction.loc[index, self.pred_column_name] = None`. Don't change other parts of this function.

4. reset: During cross validation, we need to reuse the instance of the class. Therefore, you need to reset your model.

We will evaluate the performance with the following codes. Don't modify class `CrossValidation`.

```
competition = CompetitionRecSys()
algorithm_instances = [competition]
cv_rp = CrossValidation('RPrecision')
rp = cv_rp.run(algorithm_instances, num_users, num_items)
```

Your score for Q8 will be calculated using the min-max normalization shown below:

$$Q8\_score(RPrecision_{yours}) = \max(0, \frac{RPrecision_{yours} - 0.72}{RPrecision_{max} - 0.72}) \quad (1)$$

where $RPrecision_{yours}$ is your RPrecision score and $RPrecision_{max}$ is the highest RPrecision score obtained across all student submissions. You will not receive credit for Q8 if your RPrecision score is below 0.72 and you will receive full credit(1 point) for Q8 if you have the highest RPrecision across all submissions.

Notes: You need to build your model from scratch and show all code in your answer, i.e., you cannot use external recommendation libraries, *but* you could use SVD functionality from scipy, linear/logistic regression from sklearn, or other libraries (not intended explicitly for recommendation) if permission is first granted on Piazza for the use of those libraries.

Please provide a brief description of your approach.