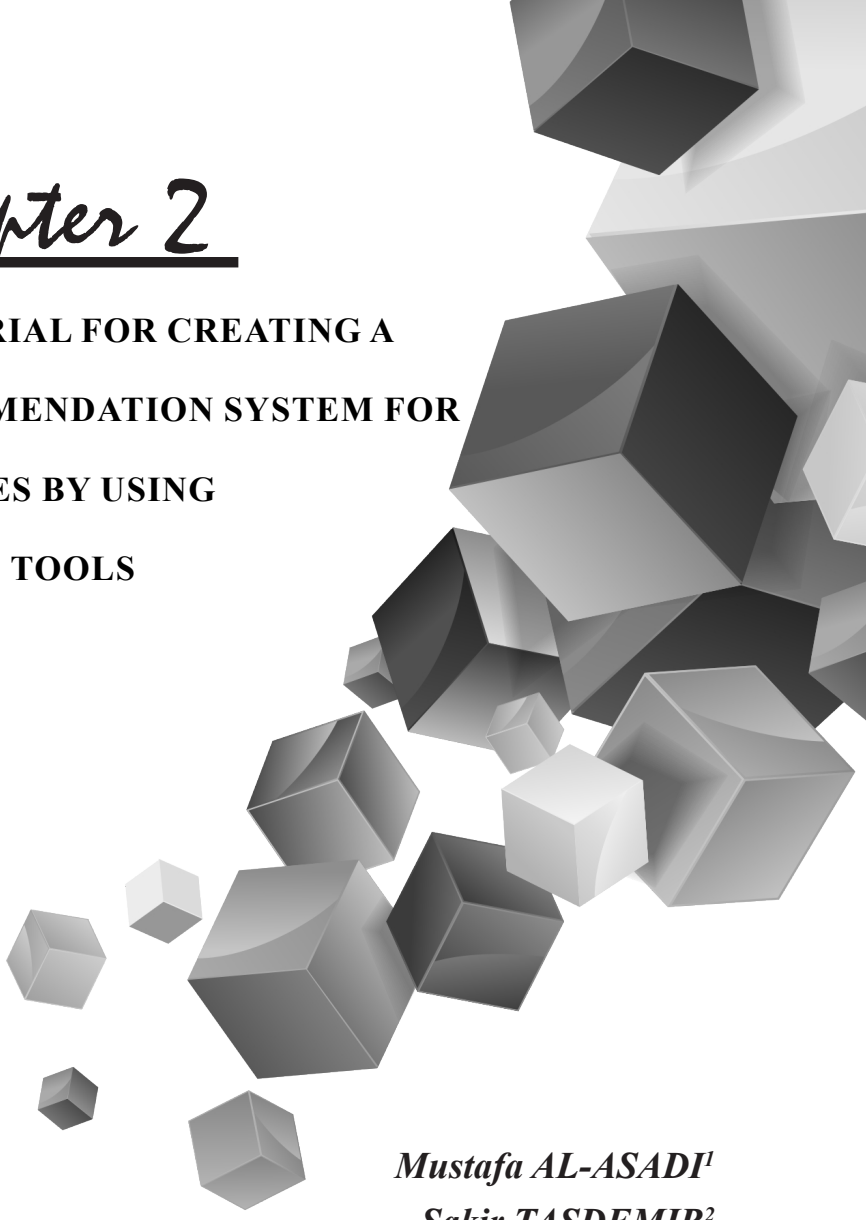
Chapter 2

A TUTORIAL FOR CREATING A RECOMMENDATION SYSTEM FOR ARTICLES BY USING PYTHON TOOLS



Mustafa AL-ASADI[1]
Sakir TASDEMIR[2]
Ilker Ali OZKAN[3]

1 Selcuk University, Department of Computer Engineering, Konya, Turkey, masadi@lisansustu.selcuk.edu.tr
2 Selcuk University, Department of Computer Engineering, Konya, Turkey, stasdemir@selcuk.edu.tr
3 Selcuk University, Department of Computer Engineering, Konya, Turkey, ilkerozkan@selcuk.edu.tr

INTRODUCTION

Recommendation systems mainly produce a list of recommendations in any field, using one of two methods - via Collaborative Filtering or Content-Based Filtering (Jafarkarimi, Sim, & Saadatdoost, 2012). Collaborative filtering depends on the behaviour of the previous user, such as products that he or she previously purchased or previous assessments, in line with similar decisions made by other users, so that the first user's wishes can be expected based on the decisions of other users (Melville & Sindhwani, 2010). Content-based filtering is based on bringing similar products with their characteristics to the purchased product (Mooney & Roy, 2000). These methods are often used simultaneously to form a single system called Hybrid Recommender Systems (Balabanović & Shoham, 1997).

The difference between the collaborative approach and the content-based method can be illustrated by comparing two known music recommender systems - Last.fm and Pandora Radio.

Last.fm makes a list of recommended songs by observing the bands and sections that are regularly heard by the user, and comparing them with similar lists for other users. And then tells the fm to run sections that do not appear in the current user list, but the sections that are heard by users who have listed similar to the current user list, it is an example of a collaborative filtering method.

Pandora uses specific features of the musical composition or artist - a set of 400 properties - to feed the playlist of users that are similar to the heard tracks, it is an example of a Content-Based Filtering (Freire, 2008).

For both systems, there are strengths and weaknesses, in the first example, for the Last.fm, the system requires a large amount of information per user to get accurate recommendations. This is an example of the so-called cold-start problem, which is common in the Collaborative Filtering systems (Elahi, Ricci, & Rubens, 2016; Rubens, Elahi, Sugiyama, & Kaplan, 2015; Schein, Popescul, Ungar, & Pennock, 2002). Whereas Pandora requires little information to start, but it is far more limited in scope (e.g., it can only make recommendations that are comparable to the original kernel).

Recommendation systems are a good alternative to search algorithms because they help users discover items they may not find. Where recommendation systems are often implemented using search engines to index non-traditional data (Vyas, 2018). Recommendation systems were first mentioned in a working paper entitled "digital bookshelf" in 1990 by Jussi Karlgren at The Royal Institute of Technology and Stockholm University (Karlgren, 1990).

Due to the increasing number of articles published on the web every day. So, the search for resources can be great, especially for beginners. Therefore, the articles recommendation system will be useful because of its ability to provide a customized platform for the articles proposed (Bancu et al., 2012).

The recommendation systems and their application have been searched in different areas, like music ((Bu et al., 2010), (Van den Oord, Dieleman, & Schrauwen, 2013), (X. Wang & Wang, 2014)), videos (Davidson et al., 2010), people (Badenes et al., 2014), jobs ((Bastian et al., 2014), (Kenthapadi, Le, & Venkataraman, 2017), (Mishra & Reddy, 2016)), and research papers ((C. Wang & Blei, 2011), (Joeran et al., 2013), (Beel, Langer, Genzmehr, & Nürnberger, 2013)), and others.

This chapter is a tutorial to creating a recommendation system for articles that match user interests and predicts which articles the user has not read by using python tools. This chapter addresses the following topics: Data Set, Evaluate the Performance of Algorithms, Implement Collaborative Filtering, Testing the Model, Results, Conclusion

DATASET

In this section, we used the Deskdrop dataset (Moreira, 2017), which contains a sample of one year (from Mar. 2016 - to Feb. 2017) from DeskDrop (CI&T's Internal Communication platform). Deskdrop is an internal communications platform, that developed by CI&T, Deskdrop is focused in companies that using Google G Suite. The main features of this platform are allowing for companies' employees to share articles with their peers, and collaborate between them.

DeskDrop dataset contains 73k logged users' interactions on more than 3k public articles that shared in the DeskDrop platform. DeskDrop is consist of two CSV files, are shared_articles.csv, and users_interactions.csv.

Shared Articles

It contains details about the articles that shared in the DeskDrop platform. Each article contains its date of sharing (timestamp), title, the original URL, content in plain text, article language (pt: Portuguese or en: English) and details about the user that shared the article (author). In a given timestamp, there are two possible event types:

CONTENT SHARED: The article has been shared in the platform and is available to users.

CONTENT REMOVED: The article has been removed from the platform.

To simplify matters, we used only “CONTENT SHARED” as event type, assuming that all articles were available during the whole year, then we upload data as Table 1.

Users Interactions

Contains records of user interactions in shared articles (Table 2). The eventType values contain:

VIEW: The users have opened the article only.

LIKE: The users have liked the article.

COMMENT CREATED: The users created a comment in the article.

FOLLOW: The users want to be notified on any new comment in the article.

BOOKMARK: Users added a bookmark to easily retrieve the article in the future

Table 1. Loading Shared_articles data for the first two rows and first four columns.

n	Timestamp	URL	Title	Lang.
1	1459193988	http://www.nytimes.com/2016/03/28/business/dea...	Ethereum, a Virtual Currency, Enables Transact...	en
2	1459194146	http://cointelegraph.com/news/bitcoin-future-w...	Bitcoin Future: When GBPcoin of Branson Wins O...	en

Table 2. Loading users_interactions data for the first two rows and first four columns

n	timestamp	eventType	sessionId	userAgent
1	1465413032	VIEW	1.2642E+18	NaN
2	1465412560	VIEW	3.62174E+18	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2...

Dataset Analysis

After loading the dataset and import the main libraries of python like pandas and NumPy, we are going to take a look at and explore the data by a few different ways:

- Dimensions of Dataset
- Statistical Summary
- Missing Data

Dimensions of Dataset

We can get a quick idea of how many instances (rows and columns) in the dataset by using "shape" property. From Figure 1 and 2, we have 3047 instances and 13 attributes in the dataset of Shared Articles, and 72312 instances and 8 attributes in the dataset of Users Interactions.

```
# shape
print(articles_df.shape)

(3047, 13)
```

Figure 1. Use shape command in python code for Shared Articles dataset.

```
# shape
print(interactions_df.shape)

(72312, 8)
```

Figure 2. Use shape command in python code for Users Interactions dataset.

Statistical Summary

We can take a look at a summary for each numerical attribute by use "describe" property. This includes count, mean, the min and max values and percentiles (Figure 3, 4).

	timestamp	contentId	authorPersonId	authorSessionId
count	3.047000e+03	3.047000e+03	3.047000e+03	3.047000e+03
mean	1.468865e+09	1.969568e+16	4.198685e+17	1.694961e+17
std	7.573604e+06	5.376353e+18	4.390382e+18	5.391587e+18
min	1.459194e+09	-9.222795e+18	-9.120686e+18	-9.212055e+18
25%	1.462401e+09	-4.673420e+18	-1.570135e+18	-4.656768e+18
50%	1.467176e+09	3.455744e+16	-7.092877e+17	3.910429e+17
75%	1.473944e+09	4.716572e+18	3.609194e+18	4.821078e+18
max	1.488308e+09	9.222265e+18	9.210531e+18	9.221043e+18

Figure 3. Use describe command in python code for Shared Articles dataset.

	timestamp	contentId	personId	sessionId
count	7.231200e+04	7.231200e+04	7.231200e+04	7.231200e+04
mean	1.470103e+09	-3.033423e+16	1.252026e+16	3.421273e+16
std	7.258130e+06	5.344755e+18	5.022333e+18	5.344355e+18
min	1.457964e+09	-9.222795e+18	-9.223122e+18	-9.222505e+18
25%	1.464876e+09	-4.726309e+18	-3.596627e+18	-4.613476e+18
50%	1.468343e+09	1.893099e+16	-1.088422e+17	5.029492e+16
75%	1.474461e+09	4.441012e+18	3.766319e+18	4.667962e+18
max	1.488310e+09	9.222265e+18	9.210531e+18	9.223314e+18

Figure 4. Use describe command in python code for Users Interactions dataset.

Missing Data

We can explore any missing values in the data set (NaN value) and length of data by using "info" property. In Figure 5 and 6, it seems no missing value in the dataset.

```
Data columns (total 13 columns):
timestamp          3047 non-null int64
eventType          3047 non-null object
contentId          3047 non-null int64
authorPersonId     3047 non-null int64
authorSessionId    3047 non-null int64
authorUserAgent    669 non-null object
authorRegion       669 non-null object
authorCountry      669 non-null object
contentType        3047 non-null object
url                3047 non-null object
title              3047 non-null object
text               3047 non-null object
lang               3047 non-null object
dtypes: int64(4), object(9)
memory usage: 333.3+ KB
```

Figure 5. Test missing data use describe command in python for Shared Articles dataset.

```
Data columns (total 8 columns):
timestamp      72312 non-null int64
eventType      72312 non-null object
contentId      72312 non-null int64
personId       72312 non-null int64
sessionId      72312 non-null int64
userAgent      56918 non-null object
userRegion     56907 non-null object
userCountry    56918 non-null object
dtypes: int64(4), object(4)
memory usage: 4.4+ MB
```

Figure 6. Test missing data use describe command in python for Users Interactions dataset.

Data wrangling

Data wrangling, sometimes referred to as Data munging, is the process of transforming data from one "raw" into other formats with to make it more appropriate to make analytics (Ramesh, 2015). More specifically, the munging process consists of a number of operations that are applied to an initial data set to be converted to a different but related data set. These operations will be located in several categories: recognition, analysis, filtering, and transformation (Cross, 2001). Furthermore, munging can mean processing raw data to achieve a final form. It can mean analyzing or filtering data, or any of the steps required to identify the data (Kroger,

2016).

In this work, due to there are different interactions types in articles, therefore we associate them with a weight, for example, a comment in an article indicates (4.0) a higher interest of the user on the item than a view (1.0), or than a simple like (2.0) as in Figure 7.

```
event_type_strength = {
   'VIEW': 1.0,
   'LIKE': 2.0,
   'BOOKMARK': 2.5,
   'FOLLOW': 3.0,
   'COMMENT CREATED': 4.0,
}
```

Figure 7. Data munging

User cold-start

Cold-start is a potential problem when you build a recommender system (Schein et al., 2002). When a new user uses the system, the system often does not know anything about that user so that he can build the appropriate suggestions for it. There are several suggested solutions to eliminate the cold start problem. Content-based filtering approaches is one of the most effective solutions to this problem (Lika, Kolomvatsos, & Hadjiefthymiades, 2014). Therefore, we are kept only users that have at least 5 interactions in the dataset as in Figure 8.

```
users_interactions_count_df = interactions_df.groupby
(['personId', 'contentId']).size().groupby('personId').size()
print('# users: %d' % len(users_interactions_count_df))
```

Figure 8. Solve the problem of cold start

Deskdrop allows users to view the article several times and interact with it in different ways (e.g. comment or follow). Therefore, to smooth the distribution and to model the user interest on a given article, we gathering all user interactions in an item by collecting the weight of the force type of the interaction and applying the log transformation to smooth the distribution as in Figure 9. It is worth mentioning that a log transformation can help make a relationship clear and describe the relationship between logs and the geometric mean. Furthermore, the log transformation can be used to make highly skewed distributions less skewed (Log, 2012).

```
def smooth_user_preference(x):
    return math.log(1+x, 2)

interactions_full_df = interactions_from_selected_users_df \
                    .groupby(['personId', 'contentId'])['eventStrength'].sum() \
                    .apply(smooth_user_preference).reset_index()
print('# of unique user/item interactions: %d' % len(interactions_full_df))
interactions_full_df.head(10)
```

```
# of unique user/item interactions: 39106
```

	personId	contentId	eventStrength
0	-9223121837663643404	-8949113594875411859	1.000000
1	-9223121837663643404	-8377626164558006982	1.000000
2	-9223121837663643404	-8208801367848627943	1.000000
3	-9223121837663643404	-8187220755213888616	1.000000
4	-9223121837663643404	-7423191370472335463	3.169925
5	-9223121837663643404	-7331393944609614247	1.000000
6	-9223121837663643404	-6872546942144599345	1.000000
7	-9223121837663643404	-6728844082024523434	1.000000
8	-9223121837663643404	-6590819806697898649	1.000000
9	-9223121837663643404	-6558712014192834002	1.584963

Figure 9. Basic code for log transformation function

EVALUATE THE PERFORMANCE of ALGORITHMS

Data split

Resampling methods are the best statistical techniques to evaluate the performance of an algorithm. Where it permits to make accurate estimates of how the algorithm performs on new data (Brownlee, 2016).

The common types of resampling techniques are Hold-out (Train and Test Split), Cross-Validation (CV), and Repeated Random Hold-out. In this work, we are using Repeated Random Hold-out, in this method we seek to create a random split of the dataset as a hold-out method but repeat the process of dividing (split) and evaluating the algorithm for several times, like K-fold cross-validation. we use, splits the data into an 80% train, 20% test split and repeats the process 42 times (Figure 10).

```
interactions_train_df, interactions_test_df = train_test_split(interactions_full_df,
                                   stratify=interactions_full_df['personId'],
                                   test_size=0.20,
                                   random_state=42)

print('# interactions on Train set: %d' % len(interactions_train_df))
print('# interactions on Test set: %d' % len(interactions_test_df))
```

Figure 10. Basic code for Data split.

Evaluation Methods

In recommender systems, there are several metrics used for evaluation. In this work, we chose Top-N accuracy metrics, which seeks to evaluate

the accuracy of the better recommendations provided to a user, comparing with the items the user has interacted in test data (Ziegler, McNee, Konstan, & Lausen, 2005).

We choose Recall@N to determine the Top-N accuracy metric. In this work, Recall@N evaluates whether the interacted item is between the top N items in the ranked list of recommendations for a user (Cremonesi, Koren, & Turrin, 2010) (Figure 11).

```
#Top-N accuracy metrics consts
EVAL_RANDOM_SAMPLE_NON_INTERACTED_ITEMS = 100

class ModelEvaluator:


    def get_not_interacted_items_sample(self, person_id, sample_size, seed=42):
        interacted_items = get_items_interacted(person_id, interactions_full_indexed_df)
        all_items = set(articles_df['contentId'])
        non_interacted_items = all_items - interacted_items

        random.seed(seed)
        non_interacted_items_sample = random.sample(non_interacted_items, sample_size)
        return set(non_interacted_items_sample)

    def _verify_hit_top_n(self, item_id, recommended_items, topn):
            try:
                index = next(i for i, c in enumerate(recommended_items) if c == item_id)
            except:
                index = -1
            hit = int(index in range(0, topn))
            return hit, index
```

Figure 11. Calculate Top-N accuracy

IMPLEMENT COLLABORATIVE FILTERING

Collaborative filtering technique works through building a database (user-item matrix) of the preferences for items by users. It then matches users with pertinent interest and preferences by calculating similarities between their profiles to make recommendations (Herlocker, Konstan, Terveen, & Riedl, 2004). These users build a group called a neighborhood. A user gets recommendations to all those elements (items) that he has not rated before but that were already positively rated by users in his neighborhood. Recommendations that are produced by Collaborative Filtering can be of either prediction or recommendation. Prediction is a numerical value, Rij, expressing the predicted score of item j for the user i (Figure 12), while Recommendation is a list of top N items that the user will like the most as shown in Figure 13. The technique of collaborative filtering (CF) has two main strategies for implementation, memory-based and model-based (Bobadilla, Ortega, Hernando, & Gutiérrez, 2013; Breese, Heckerman, & Kadie, 1998).

Memory-based

This method uses previous memory for user interactions to calculate the similarity between users based on the items they interact with (user-based approach) or calculate the similarity between items based on the users they interacted with (item-based approach). The most commonly

used example of this method is User Neighbourhood-based CF, in which the top-N similar users (Which are calculated by Pearson correlation) for a user are selected and used to recommend elements those similar users liked, but the existing user has not interacted till now.

Model-based

This method, models are developed by using different machine learning algorithms (ML) to recommend elements to users (Sarwar, Karypis, Konstan, & Riedl, 2001). There are many types of model-based CF algorithms, like Bayesian networks, neural networks, and latent factor models like as Singular Value Decomposition (SVD).

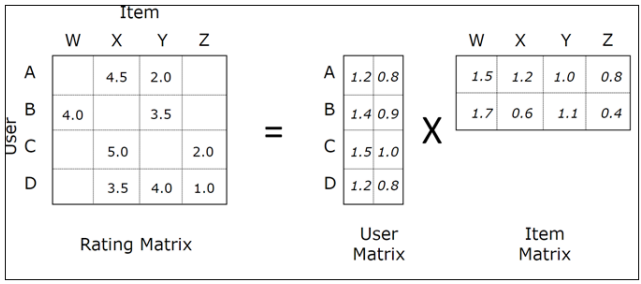


Figure 12. Recommendation by Matrix Factorization (Seo, 2018).

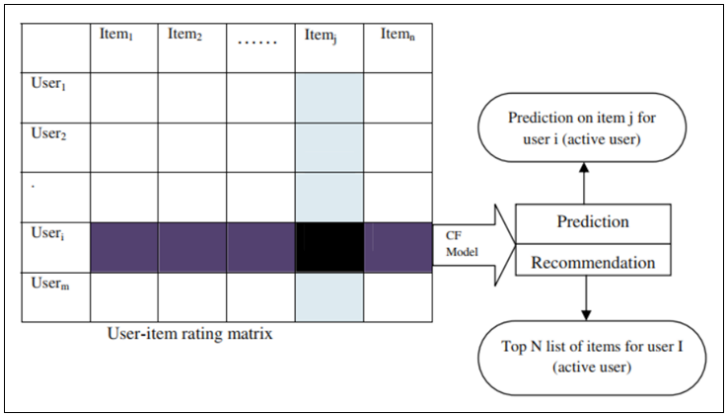


Figure 13. Block diagram for process of collaborative filtering in the model

Matrix Factorization

Latent factor models used Matrix Factorization to represent items and users in a lower-dimensional latent space (Ma, Zhou, Liu, Lyu, & King, 2011). One of the main advantages of this method is that instead of having a high-dimensional matrix that contains a large number of missing values, a smaller matrix will be dealt with a smaller matrix in lower-dimensional space. In this work, we use the common latent factor model called Singular Value Decomposition (SVD). Table 3 shown the initial matrix which consists of 10 rows × 2926 columns.

Table 3. Calculate Top-N accuracy

	personId	contentId	eventStrength
0	9223121837663643404	8949113594875411859	1.000000

An important decision in SVD is the number of factors to factor the user-element matrix. The greater the number of factors, the more precise is the factorization in the original matrix rebuilding (Golub & Reinsch, 1970). Therefore, decrease the number of factors leads to increases the model generalization (Figure 12, 14).

```
In [15]: #The number of factors to factor the user-item matrix.
         NUMBER_OF_FACTORS_MF = 15
         #Performs matrix factorization of the original user item matrix
         U, sigma, Vt = svds(users_items_pivot_matrix, k = NUMBER_OF_FACTORS_MF)

In [16]: U.shape

Out[16]: (1140, 15)

In [17]: Vt.shape

Out[17]: (15, 2926)

In [18]: sigma = np.diag(sigma)
         sigma.shape

Out[18]: (15, 15)
```

Figure 14. The number of factors to factor the user-element matrix

After the factorization, we try to reconstruct the original matrix by multiplying its factors. After factorization of the matrix, we tried to reconstruct the original matrix via multiplying its factors. The new matrix is not sparser. Predictions have been created for elements that the user has not till now interacted with and we will use them for recommendations.

TESTING THE MODEL

For testing the model, we selected one user (for example -1479311724257856983). In Figure 15., the model displays the first 20 articles suggested for the user which match to its interactions in Deskdrop from a train set. It can be seen that among the main user interests are Artificial Intelligence, Machine Learning, Deep Learning and the Google Cloud Platform.

```
In [38]: inspect_interactions(-1479311724257856983, test_set=False).head(20)
```

Out[38]:

	eventStrength	contentId	title	url	lang
115	4.285402	7342707578347442862	At eBay, Machine Learning is Driving Innovativ...	https://www.ebayinc.com/stories/news/at-ebay-m...	en
38	4.129283	621816023396605502	AI Is Here to Help You Write Emails People Wil...	http://www.wired.com/2016/08/boomerang-using-a...	en
8	4.044394	-4460374799273064357	Deep Learning for Chatbots, Part 1 - Introduction	http://www.wildml.com/2016/04/deep-learning-fo...	en
116	3.954196	-7959318068735027467	Auto-scaling scikit-learn with Spark	https://databricks.com/blog/2016/02/08/auto-sc...	en
10	3.906891	2589533162305407436	6 reasons why I like KeystoneML	http://radar.oreilly.com/2015/07/6-reasons-why...	en

Figure 15. Testing model

RESULTS

Collaborative filtering has been referred to as the similar learning pattern in machine learning, which is used for predicting recommendations based on learning similarity amongst users (Resnick, Iacovou, Suchak, Bergstrom, & Riedl, 1994). The chapter reviews some state-of-the-art recommendation systems applied in the past. In addition, we conducted some analytical operations on the data for the purpose of understanding, representing and converting them into easy-to-use formats. Finally, our proposed system seeks to actively recommend the contents of articles relevant to the reader based on its current history of preferences.

This section from chapter focuses on the results obtained for the Collaborative Recommendation part of the system. When evaluating the Collaborative Filtering model (SVD matrix factorization), we observe that we obtain Recall@10 (46%) and Recall@5 (33%) values as in Figure 16.

```
print('Evaluating Collaborative Filtering (SVD Matrix Factorization) model...')
cf_global_metrics, cf_detailed_results_df = model_evaluator.evaluate_model(cf_recommender_model)
print('\nGlobal metrics:\n%s' % cf_global_metrics)
cf_detailed_results_df.head(10)
```

```
Evaluating Collaborative Filtering (SVD Matrix Factorization) model...
1139 users processed

Global metrics:
{'modelName': 'Collaborative Filtering', 'recall@5': 0.33405778573254924, 'recall@10': 0.46816670928151366}
```

Figure 16. Testing model

CONCLUSION

Designing and developing RecSys is a multi-disciplinary effort that has benefited from results obtained in various computer science fields especially machine learning and data mining. And this clear in this tutorial chapter and the results presented above. Many RecSys are centred around the use of various machine learning and data mining algorithms to predict user evaluations for items, or for learning how to correctly rank items for a user. in this tutorial chapter, we created a recommendation system for articles that match user interests and predicts which articles the user has

not read by using python tools and using CI&T Deskdrop dataset. Overall, we have noted that the proposed approach works well via traditional matrix factorization methods that are called Singular Value Decomposition (SVD), and predicts well for articles that are not read by the user. in this tutorial, we used these traditional techniques for didactic purposes although there are more advanced techniques in recommendation systems research community, like neural networks and deep learning models.

REFERENCES

Badenes, H., Bengualid, M. N., Chen, J., Gou, L., Haber, E., Mahmud, J., Smith, B. A. (2014). *System U: automatically deriving personality traits from social media for people recommendation.* Paper presented at the Proceedings of the 8th ACM Conference on Recommender systems.

Balabanović, M., & Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Communications of the ACM, 40*(3), 66-72.

Bancu, C., Dagadita, M., Dascalu, M., Dobre, C., Trausan-Matu, S., & Florea, A. M. (2012). *ARSYS--Article Recommender System.* Paper presented at the Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2012 14th International Symposium on.

Bastian, M., Hayes, M., Vaughan, W., Shah, S., Skomoroch, P., Kim, H., . . . Lloyd, C. (2014). *LinkedIn skills: large-scale topic extraction and inference.* Paper presented at the Proceedings of the 8th ACM Conference on Recommender systems.

Beel, J., Langer, S., Genzmehr, M., & Nürnberger, A. (2013). *Introducing Docear's research paper recommender system.* Paper presented at the Proceedings of the 13th ACM/IEEE-CS joint conference on Digital libraries.

Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-based systems, 46*, 109-132.

Breese, J. S., Heckerman, D., & Kadie, C. (1998). *Empirical analysis of predictive algorithms for collaborative filtering.* Paper presented at the Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence.

Brownlee, J. (2016). Machine Learning Mastery with Python. *Machine Learning Mastery Pty Ltd*, 100-120.

Bu, J., Tan, S., Chen, C., Wang, C., Wu, H., Zhang, L., & He, X. (2010). *Music recommendation by unified hypergraph: combining social media information and music content.* Paper presented at the Proceedings of the 18th ACM international conference on Multimedia.

Cremonesi, P., Koren, Y., & Turrin, R. (2010). *Performance of recommender algorithms on top-n recommendation tasks.* Paper presented at the Proceedings of the fourth ACM conference on Recommender systems.

Davidson, J., Liebald, B., Liu, J., Nandy, P., Van Vleet, T., Gargi, U., . . . Livingston, B. (2010). *The YouTube video recommendation system.* Paper presented at the Proceedings of the fourth ACM conference on Recommender systems.

Elahi, M., Ricci, F., & Rubens, N. (2016). A survey of active learning in collaborative filtering recommender systems. *Computer Science Review, 20*, 29-50.

Freire, A. M. (2008). Remediating radio: Audio streaming, music recommendation and the discourse of radioness. *Radio Journal: International Studies in Broadcast & Audio Media, 5*(2-3), 97-112.

Golub, G. H., & Reinsch, C. (1970). Singular value decomposition and least squares solutions. *Numerische mathematik, 14*(5), 403-420.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS), 22*(1), 5-53.

Jafarkarimi, H., Sim, A. T. H., & Saadatdoost, R. (2012). A naive recommendation model for large databases. *International Journal of Information and Education Technology, 2*(3), 216.

Joeran, B., Langer, S., Genzmehr, M., Gipp, B., Breitinger, C., & Nürnberger, A. (2013). *Research paper recommender system evaluation: A quantitative literature survey.* Paper presented at the Workshop on Reproducibility and Replication in Recommender Systems Evaluation (RepSys) at the ACM Recommender System Conference (RecSys' 13). ACM, Hong Kong, China.

Karlgren, J. (1990). *An Algebra for Recommendations*. Retrieved from

Kenthapadi, K., Le, B., & Venkataraman, G. (2017). *Personalized job recommendation system at linkedin: Practical challenges and lessons learned.* Paper presented at the Proceedings of the Eleventh ACM Conference on Recommender Systems.

Lika, B., Kolomvatsos, K., & Hadjiefthymiades, S. (2014). Facing the cold start problem in recommender systems. *Expert Systems with Applications, 41*(4), 2065-2073.

Log, A. (2012). 16. Transformations. *Brain, 3000*, 4000.

Ma, H., Zhou, D., Liu, C., Lyu, M. R., & King, I. (2011). *Recommender systems with social regularization.* Paper presented at the Proceedings of the fourth ACM international conference on Web search and data mining.

Melville, P., & Sindhwani, V. (2010). Encyclopedia of machine learning: Springer-Verlag, chapter Recommender systems.

Mishra, S. K., & Reddy, M. (2016). *A bottom-up approach to job recommendation system.* Paper presented at the Proceedings of the Recommender Systems Challenge.

Mooney, R. J., & Roy, L. (2000). *Content-based book recommending using learning for text categorization.* Paper presented at the Proceedings of the fifth ACM conference on Digital libraries.

Moreira, G. (2017). Articles sharing and reading from CI&T DeskDrop. Retrieved from https://www.kaggle.com/gspmoreira/articles-sharing-reading-from-cit-deskdrop

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). *GroupLens: an open architecture for collaborative filtering of netnews.* Paper presented at the Proceedings of the 1994 ACM conference on Computer supported cooperative work.

Rubens, N., Elahi, M., Sugiyama, M., & Kaplan, D. (2015). Active learning in recommender systems *Recommender systems handbook* (pp. 809-846): Springer.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). *Item-based collaborative filtering recommendation algorithms.* Paper presented at the Proceedings of the 10th international conference on World Wide Web.

Schein, A. I., Popescul, A., Ungar, L. H., & Pennock, D. M. (2002). *Methods and metrics for cold-start recommendations.* Paper presented at the Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval.

Seo, J. D. (2018). Matrix Factorization Techniques for Recommender Systems. Retrieved from https://towardsdatascience.com/paper-summary-matrix-factorization-techniques-for-recommender-systems-82d1a7ace74

Van den Oord, A., Dieleman, S., & Schrauwen, B. (2013). *Deep content-based music recommendation.* Paper presented at the Advances in neural information processing systems.

Vyas, M. (2018). Recommendation Systems. Retrieved from https://blogs.oracle.com/meena/recommendation-systems

Wang, C., & Blei, D. M. (2011). *Collaborative topic modeling for recommending scientific articles.* Paper presented at the Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining.

Wang, X., & Wang, Y. (2014). *Improving content-based and hybrid music recommendation using deep learning.* Paper presented at the Proceedings of the 22nd ACM international conference on Multimedia.

Ziegler, C.-N., McNee, S. M., Konstan, J. A., & Lausen, G. (2005). *Improving recommendation lists through topic diversification.* Paper presented at the Proceedings of the 14th international conference on World Wide Web.

THEORY AND RESEARCH IN ENGINEERING

EDITOR:
PROF.DR.ADNAN HAYALOĞLU

gece kitaplığı

İmtiyaz Sahibi / Publisher • Yaşar Hız
Genel Yayın Yönetmeni / Editor in Chief • Eda Altunel
Editörler / Editors • Prof. Dr. Adnan Hayaloğlu
Kapak & İç Tasarım / Cover & Interior Design • Gece Kitaplığı

Birinci Basım / First Edition • © Eylül 2020
ISBN • 978-625-7243-79-7

© copyright
Bu kitabın yayın hakkı Gece Kitaplığı'na aittir.

Kaynak gösterilmeden alıntı yapılamaz, izin almadan hiçbir yolla çoğaltılamaz.

The right to publish this book belongs to Gece Kitaplığı. Citation can not be shown without the source, reproduced in any way without permission.

Gece Kitaplığı / Gece Publishing
Türkiye Adres / Turkey Address: Kızılay Mah. Fevzi Çakmak 1. Sokak Ümit Apt. No: 22/A Çankaya / Ankara / TR
Telefon / Phone: +90 312 384 80 40
web: www.gecekitapligi.com
e-mail: gecekitapligi@gmail.com

Baskı & Cilt / Printing & Volume
Sertifika / Certificate No: 47083

Theory and Research in Engineering

EDITOR

Prof. Dr. Adnan Hayaloğlu

gece kitaplığı

CONTENTS