



# AUTOMATING LITERATURE REVIEWS USING RECOMMENDER SYSTEMS

## Table of Contents

<b>1. Abstract.....</b>	<b>2</b>
<b>2. Introduction.....</b>	<b>3</b>
<b>3. Automatic literature review process and approach .....</b>	<b>3</b>
3.1 Addressing cold-start problems with content-based filtering: .....	4
<b>4. Data creation, processing, and analysis .....</b>	<b>5</b>
<b>5. Data exploration and analysis.....</b>	<b>8</b>
<b>6. Recommendation algorithms .....</b>	<b>11</b>
6.1 Collaborative filtering models: .....	11
6.2 User-based collaborative filtering:.....	12
6.3 Item-based collaborative filtering:.....	12
6.4 Basic algorithms from Surprise library: .....	12
6.5. Matrix Factorization-based algorithms: .....	16
6.6. Restricted Boltzmann Machines (RBM) and Autoencoders: .....	20
6.7. Recommendations using Deep learning Embedding matrix: .....	21
<b>7. Results and discussion .....</b>	<b>22</b>
<b>8. Conclusion .....</b>	<b>32</b>
<b>9. Prospects.....</b>	<b>32</b>
<b>10. User privacy and legal implications .....</b>	<b>33</b>
<b>11. Acknowledgments.....</b>	<b>34</b>
<b>12. References.....</b>	<b>34</b>

# 1. Abstract

The literature review is a vital part of developing a research idea. Due to the availability of large amounts of scientific publications in the digital media, this can be a complex and time-consuming process. An essential part of the process is identifying relevant publications; this usually involves researchers carefully going through scientific literature and understanding the publication's relevance to their research area. This can be automated and expedited using natural language processing techniques and recommender systems. Recommender systems have shown their prowess across several areas such as E-commerce, entertainment, academics, education, and scientific research. This study will first go through the advantages of incorporating recommender systems in literature reviews. Secondly, we will review the different kinds of recommender systems, namely content-based recommenders, collaborative-filtered recommenders, recommenders based on deep learning principles. We will also address and discuss the issues and shortcomings of recommendation systems such as cold-start problems, data sparsity, scaling and deployment in commercial applications, and ethical issues such as user privacy and data collection.

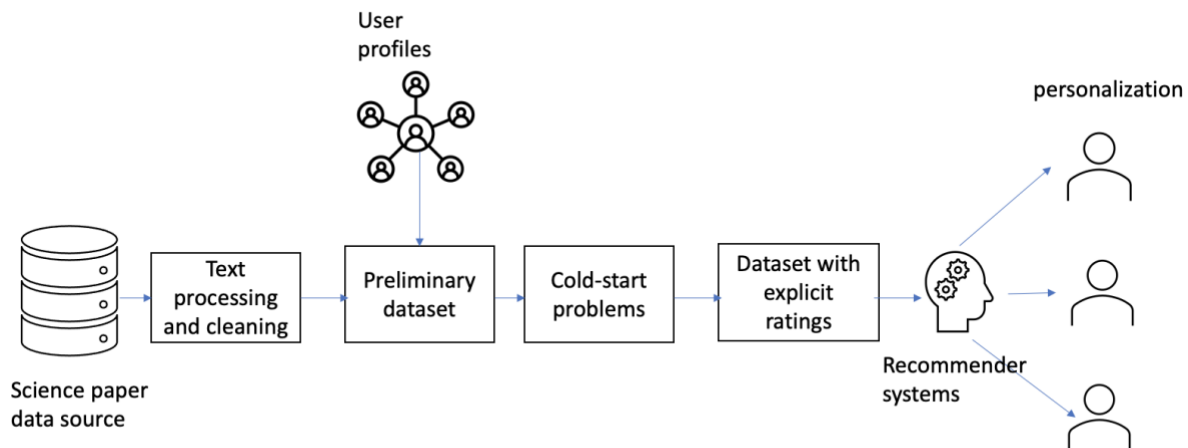
## 2. Introduction

The growth of digital media and data in the past decade has been enormous. With the advancement in data storage and processing technologies, the data volumes we deal with daily have grown exponentially. Web applications are now driven by large amounts of data behind the scenes. The true potential of the data can be tapped by using intelligent algorithms capable of personalizing information based on user preferences. This can improve user engagement and provide a better user experience. Recommender systems are one such smart systems that can learn about a user and provide relevant and valuable content from the plethora of data behind the curtains. This scientific paper recommendation system can be used extensively in academic and medical research, which regularly conducts literature reviews. For example, associate researchers and students can be recommended papers that are cornerstones in their areas of interest, new articles that keep coming up to widen their knowledge. As for seasoned researchers, Professors, and Scientists who have advanced knowledge of their areas, the recommendations can be personalized to align closely with the specific areas of their research. In both cases, the suggestions can help filter out the excess information and allow users reach the content they are looking for quicker than traditional keyword searches.

## 3. Automatic literature review process and approach

The literature review automation process has the following steps, and recommender systems can be a handy addition to the pipeline. The first step is to get the scientific papers and articles from paid and free platforms on the web. Platforms like Science Direct, IEEE, and other allied article sources can be scraped regularly using their APIs. The data can be stored in a machine-readable

format in a database. We can extract the papers and the related information from this data source in a tabular format ideal for analysis. The preliminary data is mainly a vast collection of scientific papers and related information in a clean format ready to be worked upon. An integral step in the process would be to understand the needs of the users/researchers in parallel. User Profiles are a helpful tool where data related to the user is collated.



*Figure 1- Automation process flow*

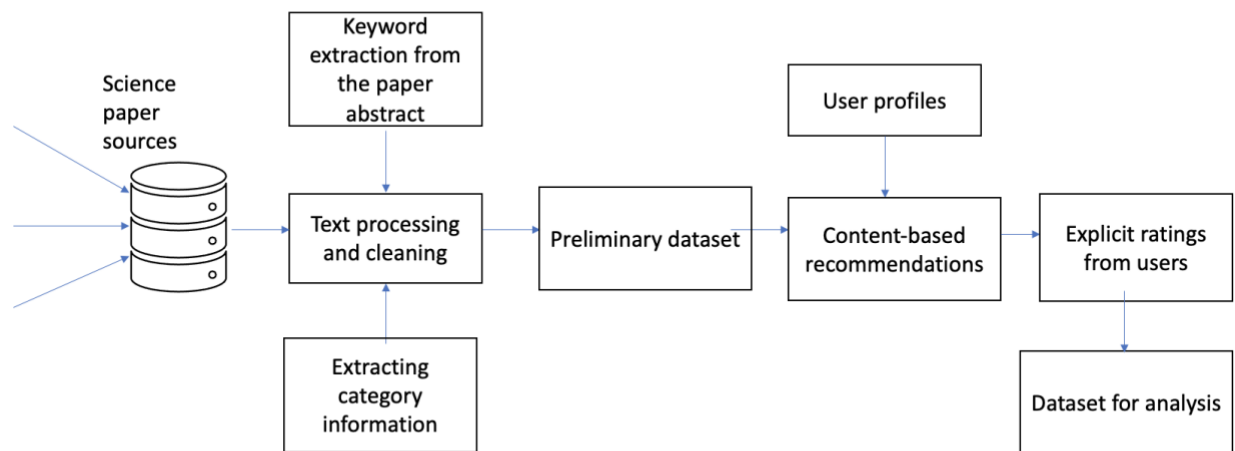
User profiles typically contain information regarding the area of interest for the user/researcher, and user identification number to numerically identify the user, user's past search actions. With the development of any recommendation system, one of the first hurdles we face is the cold-start problem. The recommender is not familiar with the user's preferences, the user is not aware of what data is kept behind by the system. This is known as the cold-start problem.

### **3.1 Addressing cold-start problems with content-based filtering:**

As a starting point, content-based filtering and recommendations can be used to give content to the user and understand their level of satisfaction with explicit ratings. Once we have the explicit ratings provided by the user for the content they are presented with, we can come up with more

accurate recommendations and personalization for the researcher. This end-to-end process can be automated using data pipelines and continuously collecting user ratings and using that feedback to reinforce learning and improve personalization.

## 4. Data creation, processing, and analysis



*Figure 2- Dataset creation flow*

The dataset we have used for analysis and recommendation building follows the above process.

The data we have used as the source is originally a subset of the **arXiv Dataset**. \*Quoted from the website, arXiv is a free distribution service and an open-access archive for scholarly articles in the fields of physics, mathematics, computer science, quantitative biology, quantitative finance, statistics, electrical engineering, and systems science, and economics. The dataset is also hosted in Kaggle. The processed dataset was obtained from this [location](#), which was a tidied tsv version of the original dataset. From this, we were able to get several folders of scientific publications and we were able to create our dataset of about 6664 records with the following attributes and scientific articles from areas that are familiar to us such as Deep learning, Database

research, big data, and parallel processing, Computer science areas such as data mining, clustering, and classification. The data dictionary for our data set can be found in the below table.

Attributes	Description
abstract	Abstract of the publication
acm_class	For submissions to the cs archive, this field is used to indicate the classification code according to the ACM Computing Classification System
arxiv_id	ArXiv ID can be used to access the paper <a href="https://arxiv.org/abs/{arxiv_id}">https://arxiv.org/abs/{arxiv_id}</a> : Page for this paper including its abstract and further links <a href="https://arxiv.org/pdf/{arxiv_id}">https://arxiv.org/pdf/{arxiv_id}</a> : Direct link to download the PDF
author_text	Author(s) names
categories	Each arXiv article has a primary category and may also have one or more cross-lists to other categories. The categories control what mailings an article is announced in, and also provide a way to limit searches to subsets of arXiv.
comments	General comments and information about the publication
created	Date of creation
doi	The digital object identifier of the publication
num_authors	Number of authors involved in the publication
num_categories	Number of categories the publication falls under
primary_cat	The primary category of the publication
title	Title of the publication
updated	Last updated date
categories_list	Categories of the publication in a list format

This is a crude dataset. This dataset still needs to be cleaned and structured to make it viable for applying machine learning algorithms and generating recommendations. We have applied

several text cleaning methods to remove extra spaces, typos, special characters, etc. The `arxiv_id` column is clean, and a numerical value is extracted to generate the `paperId` attribute. This number is unique for each paper and can be used to identify the paper from the dataset.

There is also a keyword extraction using RAKE that is being applied to the abstract and the title attributes to get a gist of the content. RAKE is a powerful keyword extraction algorithm that uses stopwords and phrase delimiters to make a matrix of co-occurring words and present the important words that make up the content. We can use these keywords attribute to speed up our algorithms which calculates similarity coefficients between different sets of text.

For recommendation algorithms to work effectively, explicit ratings are an important and integral component. The algorithms need ratings to recommend, and users need recommendations to rate, hence we use a content-based algorithm to present recommendations that are textually closer to the user's input and produce a set of recommendations. The user is now asked to rate the recommendations on a scale of 1-5, 5 being the most relevant and 1 being the least relevant. This is how we arrive at the rating attribute. This is then merged to the original dataset using the `paperId`. The final dataset has the following attributes, and the data dictionary is as follows.

user	Numerical identification number for a user	
Area of interest	The area of research the user is interested in	
Input	User's input history	
Title	Title of the paper	
Rating	Rating given by the user on a scale of 1-5	



arxiv_id	ArXiv ID can be used to access the paper <a href="https://arxiv.org/abs/{arxiv_id}">https://arxiv.org/abs/{arxiv_id}</a> : Page for this paper including its abstract and further links <a href="https://arxiv.org/pdf/{arxiv_id}">https://arxiv.org/pdf/{arxiv_id}</a> : Direct link to download the PDF	
paper_Id	the unique numerical value extracted from arxiv_id	
created	Date of creation	
categories	The areas of studies the paper belong to	
authors	Author(s) names	
keywords	The important keywords of the abstract and the title extracted using the RAKE algorithm	

This is the dataset that will be used for analysis, rating predictions, and recommendation generation.

## 5. Data exploration and analysis

Data exploration is an important part of the analysis to understand the intrinsic features of the dataset. It is aimed at creating a visual understanding of the data to map out the relationship between the attributes of the dataset and develop algorithms to handle it.

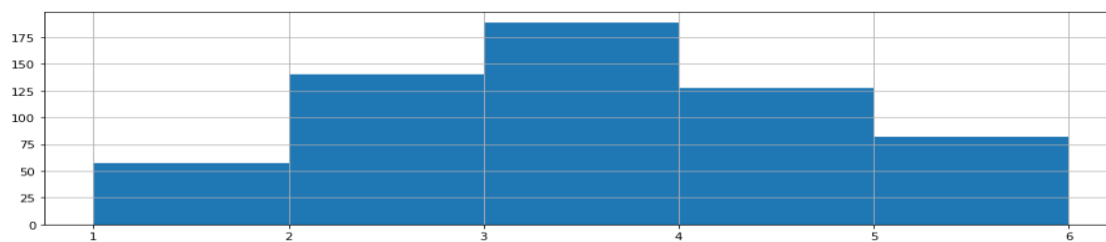
In our case, the final dataset needs to be analyzed for relationships and correlations.

There are 2294 rows with explicit ratings from user profiles we created using content-based filtering. The ratings range from 1 to 5, 5 being the highest rating for relevancy and 1 being the lowest.

One of the important aspects of the data is sparsity. Data sparsity is a measure of how empty the data is. A high sparsity can sometimes be a problem when we want to apply certain machine learning algorithms. However, some algorithms can still handle high sparsity. Understanding the sparsity percentage can help us anticipate what kind of algorithms we can apply. In our case, we

have a very high sparsity of 98% as we have a total of 222 user profiles that have 597 science papers. This is a good proportional representation of the real-world datasets that are recommender engine candidates. Many times, explicit ratings are not present as real users do not take the time to rate something they liked or disliked. Appropriate application of machine learning algorithms will help derive good results from sparse datasets as well.

Rating frequency is another important aspect of the dataset. Since we want to be able to predict the ratings for a user for any given article and present recommendations based on that, understanding rating frequency is imperative.



*Figure 3- Rating distribution*

The majority of the ratings fall under the 3 to 4 range. The occurrence is almost normal. The summary statistics of the rating attribute is count 597.000000

mean 3.829146

std 3.032977

min 0.000000

25% 2.000000

50% 2.000000

75% 4.000000

max 28.000000

This shows that the users have got relevant information during the user profile creation phase.

The relationship between ratings and rating frequency can be seen in the below-combined plot as well for more clarity concerning occurrence.

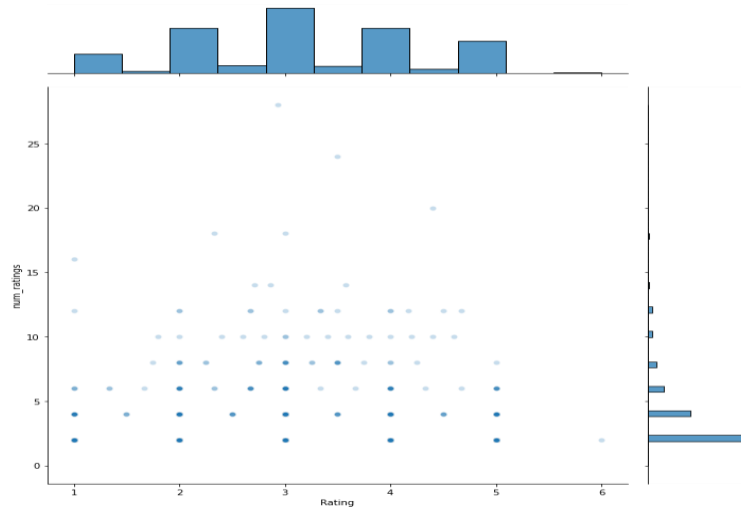


Figure 4 - Relationship between ratings and number of ratings

Understanding the correlation between ratings and the other attributes will also help pick the right features for analysis.

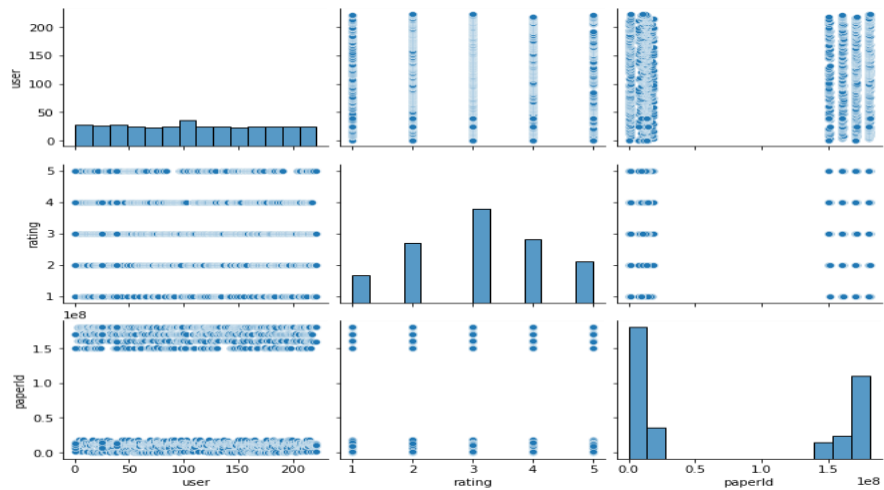


Figure 5 - Correlation between features

Linear regression is not typically used in building recommendation models, but it can be a powerful tool to understand the relationship between the attributes. Though the plots can be very clustered around the extremities for this dataset, we can still see that the users prefer a certain set

of scientific articles which can also mean that these science articles are the benchmark in their respective areas. Also, while analyzing the correlations between features, the users, the keywords, authors, and categories have the most impact on the ratings. This is a direct translation of our real-world understanding that users pick papers that are relevant to their area of interest, authors greatly influence the importance of the paper and the extract of the keywords of the scientific article or publishing.

From the preliminary exploratory analysis of the final dataset, we can understand that the data is sparse, and the ratings concentrate around certain categories, some papers have more ratings than others. This implies that some machine learning models can successfully predict ratings for users based on this dataset. However deep learning models may need more data for training. The following section reviews all the models that have been applied to this dataset and we can pick the models that have the best performance numbers in terms of accuracy.

## 6. Recommendation algorithms

We have predominantly used content-based filtering to address the cold start problem and get explicit ratings from the user. The personalization is mainly based on the explicit ratings each user has given for the recommendations presented earlier. Based on the characteristics of our final dataset, recommendations based on collaborative filtering would produce effective results and satisfy the expectations of the user.

### 6.1 Collaborative filtering models:

Collaborative filtering models work on finding similarities between the user and items and items based on a user-item pairwise rating. To implement a collaborative filtering-based

recommendation system, we will make use of the surprise library. This is a scikit-python based library that can help speed up the process.

There are two types of collaborative filtering: user-based and item based

## **6.2 User-based collaborative filtering:**

User-based collaborative filtering is completely based on the user's interests. The filtering checks for the users who are similar and predicts the interests of a new user(user with similar interests)

## **6.3 Item-based collaborative filtering:**

Item-based collaborative filtering can be used to find the similarity between the items using the ratings by users. Here the relation between the pair of items is observed. We find the missing rating with the help of the ratings given to the other items by the user.

To start with the model-building process, we need to benchmark the recommendation algorithms and understand which model works best with our dataset. To gauge the performance of the models, we will make use of the root mean square error accuracy parameter. The various models that we implemented making use of the surprise library are as follows.

The definitions and formulae for the algorithms are taken from the Surprise documentation.

## **6.4 Basic algorithms from Surprise library:**

### **6.4.1. NormalPredictor**

NormalPredictor algorithm predicts a random rating based on the distribution of the training set, which is assumed to be normal. These algorithms are not very sophisticated when it comes to the

ability to predict ratings and generate recommendations, but it is useful for accuracy comparisons.

The prediction is  $\hat{r}_{ui}$ , generated from the normal distribution  $N(\mu^\wedge, \sigma^\wedge^2)$  where  $\mu^\wedge$  and  $\sigma^\wedge$  are estimated from the training data using maximum likelihood estimation.

$$\hat{\mu} = \frac{1}{|R_{\text{train}}|} \sum_{r_{ui} \in R_{\text{train}}} r_{ui}$$

$$\hat{\sigma} = \sqrt{\sum_{r_{ui} \in R_{\text{train}}} \frac{(r_{ui} - \hat{\mu})^2}{|R_{\text{train}}|}}$$

#### 6.4.2 BaselineOnly

BaselineOnly algorithm predicts the baseline estimate for a given user and item.

$$\hat{r}_{ui} = b_{ui} = \mu + b_u + b_i$$

If a user is unknown, the bias is assumed to be zero. This is how the rating predictions are arrived at, where  $b_u$  and  $b_i$  are biases.

#### 6.4.3 k-NN algorithms:

k-NN algorithms are based on the k-nearest neighbors' approach which enables many useful applications such as classification and imputing missing values. As the name suggests, the algorithm makes use of adjacent values to a data point and suggests predicting the data class or a continuous value for a new data point. Surprise library offers the following implementations which can be further tuned and analyzed to improve rating prediction accuracies.

KNNBasic, which is a basic collaborative filtering algorithm based on the k-nearest neighbor approach.

The prediction  $\hat{r}_{ui}$  is set as:

$$\hat{r}_{ui} = \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot r_{vi}}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)}$$

or

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} \text{sim}(i, j) \cdot r_{uj}}{\sum_{j \in N_u^k(i)} \text{sim}(i, j)}$$

Depending on the user\_based field of the sim\_options parameter in the code implementation.

#### 6.4.4 KNNWithMeans

KNNWithMeans is a basic collaborative filtering algorithm but takes into account the mean ratings of each user and predicts user ratings accordingly.

The prediction  $\hat{r}_{ui}$  is set as:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in N_i^k(u)} \text{sim}(u, v) \cdot (r_{vi} - \mu_v)}{\sum_{v \in N_i^k(u)} \text{sim}(u, v)}$$

or

$$\hat{r}_{ui} = \mu_i + \frac{\sum_{j \in N_u^k(i)} \text{sim}(i, j) \cdot (r_{uj} - \mu_j)}{\sum_{j \in N_u^k(i)} \text{sim}(i, j)}$$

Depending on the user\_based field of the sim\_options parameter in the code implementation.

#### 6.4.5 KNNWithZScore

KNNWithZScore is a basic collaborative filtering algorithm, taking into account the z-score normalization of each user.

The prediction  $\hat{r}_{ui}$  is set as:

$$\hat{r}_{ui} = \mu_u + \sigma_u \frac{\sum_{v \in N_l^k(u)} \text{sim}(u, v) \cdot (r_{vi} - \mu_v) / \sigma_v}{\sum_{v \in N_l^k(u)} \text{sim}(u, v)}$$

or

$$\hat{r}_{ui} = \mu_i + \sigma_i \frac{\sum_{j \in N_u^k(i)} \text{sim}(i, j) \cdot (r_{uj} - \mu_j) / \sigma_j}{\sum_{j \in N_u^k(i)} \text{sim}(i, j)}$$

#### 6.4.6 KNNBaseline

KNNBaseline is a basic collaborative filtering algorithm taking into account a baseline rating depending on the user\_based field of the sim\_options parameter. The user\_based and the sim\_options parameters can be fine-tuned to improve the prediction accuracy.

The prediction  $\hat{r}_{hi}$  is set as:

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{v \in N_1^\lambda(u)} \text{sim}(u, v) \cdot (r_{u1} - b_{vi})}{\sum_{v \in N_l^\alpha(u)} \text{sim}(u, v)}$$

or

$$\hat{r}_{ut} = b_{ut} + \frac{\sum_{j \in N^k(i)} \text{sim}(i, j) \cdot (r_{aj} - b_{sj})}{\sum_{j \in N_\alpha^k(i)} \text{sim}(i, j)}$$

Where the common parameters are,

k (int) – The (max) number of neighbors to take into account for aggregation (see this note).

Default is 40.

- min\_k (int) – The minimum number of neighbors to take into account for aggregation. If there are not enough neighbors, the neighbor aggregation is set to zero (so the prediction ends up being equivalent to the baseline). Default is 1.



- `sim_options` (dict) – A dictionary of options for the similarity measure. See Similarity measure configuration for accepted options. It is recommended to use the `pearson_baseline` similarity measure.
- `bsl_options` (dict) – A dictionary of options for the baseline estimates computation. See Baselines estimates configuration for accepted options.
- `verbose` (bool) – Whether to print trace messages of bias estimation, similarity, etc. Default is True.

## 6.5. Matrix Factorization-based algorithms:

Matrix factorization is a mathematical concept to generate latent features by multiplying two different kinds of entities. Collaborative filtering is technically an application of this mathematical model. This is particularly useful for our dataset because of the sparsity percentage. Since not every user gives ratings to all papers, there are many missing values in our dataset. Matrix factorization helps discover the latent features that are present between the users and items based on their preferences and interest. Hence, we could arrive at a prediction on a rating based on the similarity coefficients between users. The two main similarity coefficients we use are the cosine similarity and Pearson similarity.

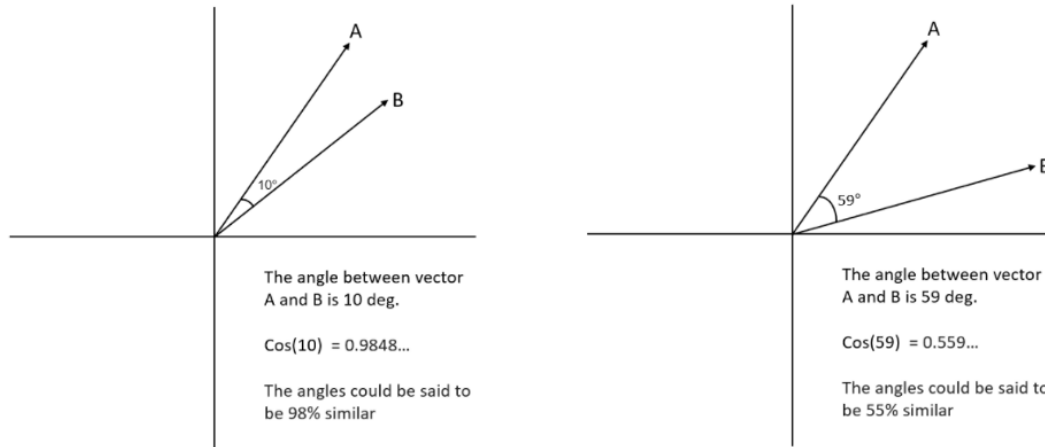
### 6.5.1 Cosine similarity:

The metric which is used to measure the similarity between the users and or items regardless of their size is known as cosine similarity. Analytically, the cosine of the angle between two vectors projected in a multi-dimensional space is measured. Though the two similar entities are far apart by the Euclidean distance (due to the size of the document), there is a higher probability that these documents may still be aligned together. In a way, this metric becomes superior. The closer the documents are by angle, the higher is the Cosine Similarity.

Cosine similarity can be computed using the following formula:

$$\text{similarity}(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n A_i^2} \times \sqrt{\sum_{i=1}^n B_i^2}}$$

The values lie between -1 and 1 by which -1 is perfectly dissimilar and 1 is perfectly similar. The cosine similarity library has procedures and functions in it. The function can be implemented for similarity on smaller datasets whereas the procedures can be implemented for similarities on bigger datasets as they can complement computation.



*Figure 6 - Visual representation of Cosine similarity*

As the cosine similarity measurement gets closer to 1, the angle between the two vectors A and B becomes smaller. The above images depict it more clearly.

It apprehends the orientation of the data objects but not the magnitude.

This metric can be utilized as recommendation systems, plagiarism detectors, data mining, etc

### **6.5.2 Pearson similarity**

The Pearson correlation coefficient or the Pearson similarity calculates the correlation between two jointly distributed random variables. Mathematically, it is the ratio between the covariance of two variables X and Y and the product of their standard deviations.

The values lie between -1 and 1. If the value is -1, then the two random variables are perfectly negatively correlated and if the value is 1, then the two random variables are perfectly positively correlated.

Pearson similarity can be computed using the following formula:

$$\begin{aligned}
 \rho_{X,Y} &= \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} \\
 &= \frac{\mathbb{E}[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y} \\
 &= \frac{\mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]}{\sqrt{\mathbb{E}[X^2] - (\mathbb{E}[X])^2} \sqrt{\mathbb{E}[Y^2] - (\mathbb{E}[Y])^2}} \\
 &= \frac{(\sum_{i=1}^n x_i y_i) - (n\bar{x}\bar{y})}{\sqrt{\sum_{i=1}^n x_i^2 - n\bar{x}^2} \sqrt{\sum_{i=1}^n y_i^2 - n\bar{y}^2}}
 \end{aligned}$$

The metric can be used when quantities vary. The Pearson similarity can be utilized when the accuracy of a variable increases when the data is not normalized. And also, it can rectify any attribute scaling while the final variable is being tabulated.

Singular vector decomposition:

Singular vector decomposition, SVD for short, is an algorithm developed for recommender systems and popularized during the Netflix Prize. This is equivalent to probabilistic matrix factorization in the absence of baselines.

The prediction  $\hat{r}_w$  is set as:

$$\hat{r}_{\alpha i} = \mu + b_\mu + b_i + q_1^T p_u$$

If user  $u$  is unknown, then the bias  $b_\alpha$  and the factors  $p_u$  are assumed to be zero. The same applies for item  $i$  with  $b_i$  and  $q_i$

We can also control the learning rate and regularization terms when we implement the algorithm from Surprise. They can be further fine-tuned to improve our prediction accuracies.

Another variation of the SVD algorithm is the SVDpp algorithm is an extension of SVD that takes into account implicit ratings to predict the ratings.

### 6.5.3 Non-negative Matrix Factorization (NMF):

NMF is a collaborative filtering algorithm based on Non-negative Matrix Factorization, similar to SVD. This is a methodology in multivariate analysis where a matrix is factorized into two matrices with the property that there should be no negative elements. NMF is particularly useful in data imputations as it takes the missing values into account by applying a minimal cost function instead of zeros.

Using this we can take into consideration a second-order effect from the missing values instead of eliminating them.

The prediction  $\hat{r}_{ui}$  is set as:

$$\hat{r}_{ui} = q_i^T p_u$$

Where the user and item factors are kept positive.

Slope One:

Slope One is a straightforward implementation of the SlopeOne algorithm. The implementation is relatively simple and the accuracy is on par with the sophisticated recommender algorithms.

While most other collaborative algorithms use the similarity between vectors of items to personalize recommendations to users, the Slope one approach is built on the average difference between the preferences.

The prediction  $\hat{r}_{ui}$  is set as:

$$\hat{r}_{ui} = \mu_u + \frac{1}{|R_i(u)|} \sum_{j \in R_i(u)} \text{dev}(i, j),$$

Where  $R_{i(u)}$  is the set of relevant items rated by user  $u$  that have one common user with user  $i$ .

$\text{dev}_{(i,j)}$  is the average difference between the ratings of the two users  $i$  and  $j$ .

#### 6.5.4 Co-clustering

Co-clustering is a collaborative filtering algorithm based on simultaneous clustering of items and users. This implies the methodology of making automatic predictions about the interests of a user based on the engagement information in the user profile with other users' engagement data. Co-clustering helps identify hidden patterns among user preferences and items in the data and exploit them to produce personalized recommendations.

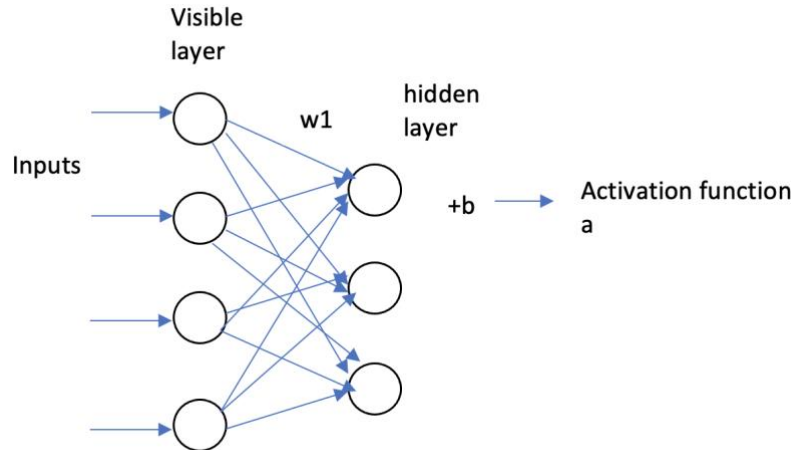
The prediction  $\hat{r}_{ui}$  is set as:

$$\hat{r}_{ui} = \overline{C_{ui}} + (\mu_u - \overline{C_u}) + (\mu_i - \overline{C_i}),$$

Where  $\overline{C_{ui}}$  is the average rating of the co-cluster and  $\overline{C_u}$  is the average rating of the user's cluster and  $\overline{C_i}$  is the average rating of the item cluster  $C_i$ .

### 6.6. Restricted Boltzmann Machines (RBM) and Autoencoders:

Restricted Boltzmann Machines are useful algorithms that come under the classification of Neural networks. They are relatively simple and have been used in machine learning areas like dimensionality reduction, classification, regression, collaborative filtering, feature learning, and topic modeling for a long time now. The RBM model is a shallow neural network with two layers, one is a visible layer and the other is a hidden layer. Each visible node takes a low-level feature from an item in the dataset to be learned and is fed into the hidden layer with randomly assigned weights( $w$ ). Each input is then multiplied with the weights and then they are summed to a bias, and the result is passed through an activation function to produce the node's output.



*Figure 7- Visual representation of Restricted Boltzmann machine*

There is a backward propagation of the error that takes place inside the hidden layer based on which the reconstruction of the original data is done by the model. Though RBM models are used occasionally, it is deprecated to pave the way for the updated models in the area, namely the autoencoders. Since our dataset is very sparse, we experimented with RBM models to study the feasibility of the application and eventually move to autoencoders once we have gained enough user-profiles and explicit ratings to train and test autoencoder models.

## **6.7. Recommendations using Deep learning Embedding matrix:**

The embedding matrix approach is designed based on teaching the model the meanings of the content that it is supposed to process. This is modeled after the way our brains identify and process a word. The concept of an embedding matrix is an effort to represent the words in the vector spaces and process the relationships between them. Conceptually words can be explained as vector representations in 3-dimensional space but in reality, the language is so complex that the representations typically require about 300 dimensions.

The concept is very foreign upon the first introduction and Keras library which is very useful in processing embedding matrix models has very little documentation about how the concept is implemented internally other than the conversion of positive integers into dense vectors of fixed size. While processing text in NLP algorithms, one-hot encoding typically gives a sparse matrix representation of the word in high dimensions, however embedding matrices can reduce that overhead by representing words in vector space and placing them in a matrix for algorithms to consume and produce desirable results.

## 7. Results and discussion

As part of the model evaluation process, the final dataset was taken into consideration to build, train and test the models we have elaborated as part of the Recommendation algorithms section. Throughout the model evaluation phase, the dataset was split into training and validation datasets for training and testing the efficacy of the models respectively.

Evaluation metrics:

We must use a quantifiable parameter to be able to compare the efficacy of the algorithms. While recommendations are one way to do this, it is not quantifiable. It can be very subjective to each user's preference. Hence a better way is to predict the rating for each scientific paper by each user when presented as a recommendation and personalize the results based on the high predicted rating. In this section, we will look at the performances of each algorithm on the root mean square accuracy of the rating prediction. We will not be going into the realms of recommendations for ease of analysis and comparison.

We have developed 2 broad categories of models for the dataset as mentioned above, the first set being the models from the Surprise library with tunable hyperparameters and the second category being deep learning models. Though these models are not directly comparable owing to their differences in architecture and operation, we can still analyze how the algorithm is effective for the nature of our dataset. Having established the basis on which we are going to analyze the results, we can dive into the evaluation parameters.

The models are trained to learn about the user preferences by making use of the features of the scientific publications such as authors, abstract, created date and explicit ratings from the user. We then predict the ratings an user will give for a publication the model has not seen before and based on the ratings we derive the personalization.

We have used the root mean square (RMSE) for the rating prediction as the main evaluation criterion so that we can compare models across different inherent natures.

Let us start from the Basic algorithms from Surprise library:

To start with we tested the waters by benchmarking all the algorithms Surprise has to offer to understand how the models behave with our dataset.

Algorithm		Test RMSE	Fit time	Test time
KNNBaseline		0.489644	0.003319	0.006978
KNNBasic		0.530911	0.001076	0.006187
SlopeOne		0.559549	0.005834	0.011946
KNNWithMeans		0.642481	0.002958	0.006314
NMF		0.703808	0.091013	0.002839
KNNWithZScore		0.723439	0.008812	0.007993
SVDpp		0.839546	0.193377	0.009942
SVD		0.945545	0.070327	0.003086
BaselineOnly		0.963611	0.002503	0.002374
CoClustering		0.964316	0.063051	0.002541
NormalPredictor		1.618165	0.001031	0.003111



From the RMSEs, we can see that kNN algorithms work well with our dataset.

KNNBaseline has the most promising RMSE of the lot. Based on this initial analysis, we have dived deeper into the kNN algorithms to try different variations of the model and see if we can improve any of the other kNN models to get better results than the basic model.

### Analysis of the kNN models without a baseline:

kNN models can have n-number of variations as they have a lot of variable parts.

For each of the kNN models, we will run cross-validation with hyperparameter tuning for k and similarity metrics and choose the best model.

Defining the similarity options, to be used by different models. The four options are

MSD, Cosine, Pearson, and Pearson baseline alongside a tuning for k values 1,2, and 3.

Algorithm	Similarity	k	Train RMSE	Test RMSE
KNNWithZScore	pearson	1	0.098576	0.645603
KNNWithZScore	pearson	2	0.098398	0.669194
KNNWithZScore	pearson_baseline	3	0.056269	0.681811
KNNWithZScore	pearson_baseline	1	0.069463	0.682994
KNNWithZScore	pearson_baseline	2	0.05354	0.685416
KNNWithZScore	pearson	3	0.10612	0.699809
KNNWithZScore	cosine	3	0.442674	0.71027
KNNWithZScore	MSD	3	0.321293	0.713074
KNNWithZScore	MSD	2	0.334879	0.71635
KNNWithZScore	cosine	2	0.489074	0.759207
KNNWithZScore	MSD	1	0.371554	0.76912
KNNBasic	MSD	1	0.074629	0.785859
KNNWithZScore	cosine	1	0.569671	0.80076
KNNBasic	MSD	3	0.235883	0.82134
KNNBasic	MSD	2	0.157064	0.838627
KNNBasic	pearson_baseline	3	0.061432	0.990183
KNNBasic	pearson_baseline	2	0.054155	1.008831
KNNBasic	pearson_baseline	1	0.073403	1.027164
KNNBasic	pearson	3	0.156887	1.145806
KNNBasic	pearson	1	0.171616	1.160322
KNNBasic	pearson	2	0.153312	1.161881
KNNBasic	cosine	3	1.026217	1.192359
KNNBasic	cosine	2	1.098542	1.215896
KNNBasic	cosine	1	1.329253	1.397355

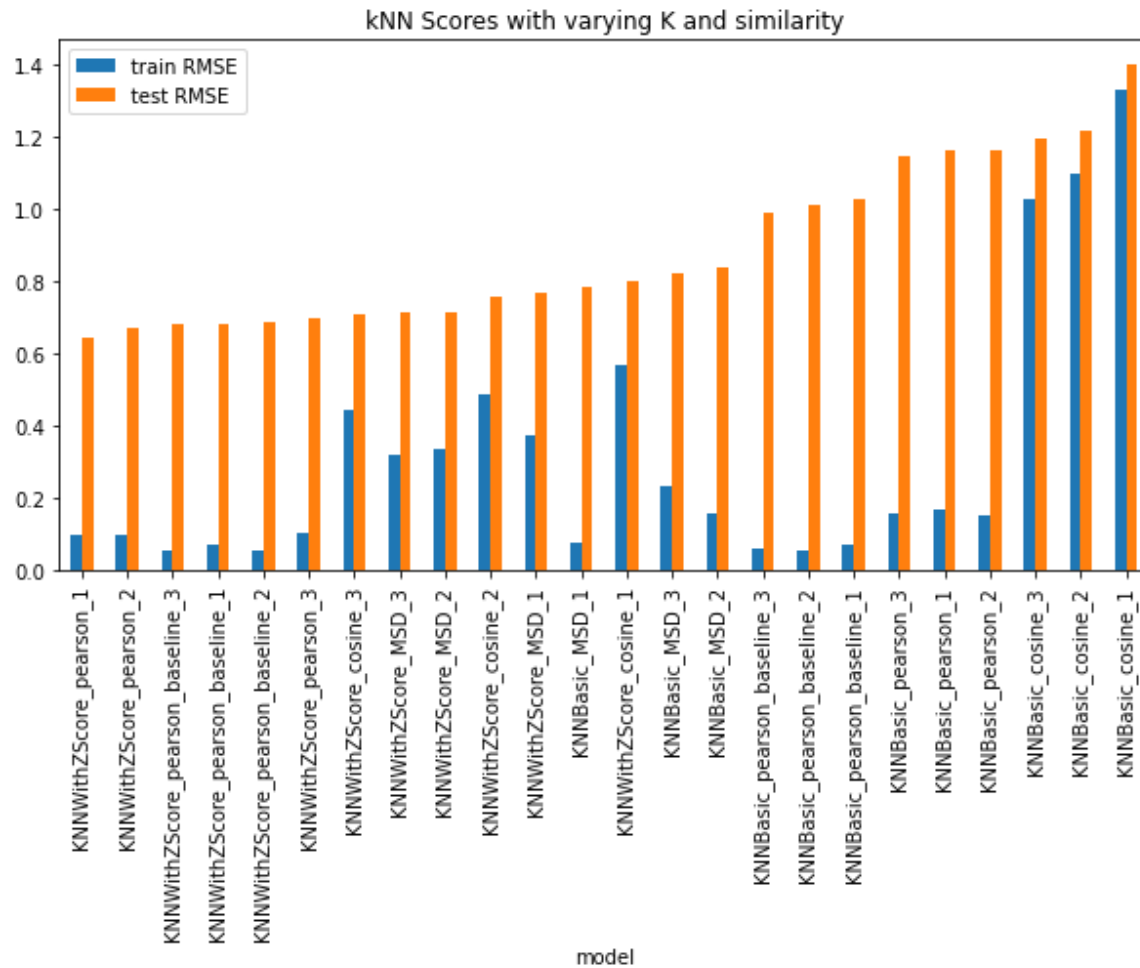


Figure 8 - kNN scores with varying similarity and k

Based on the above table we can see that KNNWithZscore using Pearson similarity along with a k of 1 has an RMSE of **0.645603**.

When we use the best model without a baseline to fit the data, the RMSE comes to **0.5784**.

### Analysis of the kNN models with baseline:

Next up is a similar analysis on the baseline model with tuned parameters.

We can use two different methods to tune and establish the model for the baseline.

The first one is using the stochastic gradient descent (SGD) scores and the second one is the alternating least squares (ALS) method.

### Hyper tuning the KNNBaseline using the SGD scores:

While using SGD, we can potentially tune the learning rate and regularization of the model. The variations are as follows.

Algorithm	Regularization	Learning rate	k	Train RMSE	Test RMSE
KNNBaseline_sgd	0.02	0.01	1	0.075601	0.765418
KNNBaseline_sgd	0.1	0.01	2	0.051935	0.768898
KNNBaseline_sgd	0.02	0.01	2	0.054145	0.773562
KNNBaseline_sgd	0.05	0.01	3	0.057023	0.774306
KNNBaseline_sgd	0.05	0.01	2	0.053454	0.774909
KNNBaseline_sgd	0.1	0.01	1	0.073769	0.780179
KNNBaseline_sgd	0.05	0.01	1	0.063691	0.785762
KNNBaseline_sgd	0.1	0.01	3	0.060724	0.791136
KNNBaseline_sgd	0.02	0.01	3	0.055255	0.79405
KNNBaseline_sgd	0.1	0.005	1	0.073788	0.834466
KNNBaseline_sgd	0.02	0.005	3	0.056555	0.83465
KNNBaseline_sgd	0.1	0.005	3	0.05563	0.838239
KNNBaseline_sgd	0.05	0.005	1	0.066893	0.83902
KNNBaseline_sgd	0.02	0.005	2	0.052237	0.840942
KNNBaseline_sgd	0.1	0.005	2	0.054486	0.843294
KNNBaseline_sgd	0.02	0.005	1	0.067538	0.843786
KNNBaseline_sgd	0.05	0.005	3	0.059004	0.849339
KNNBaseline_sgd	0.05	0.005	2	0.054849	0.855272

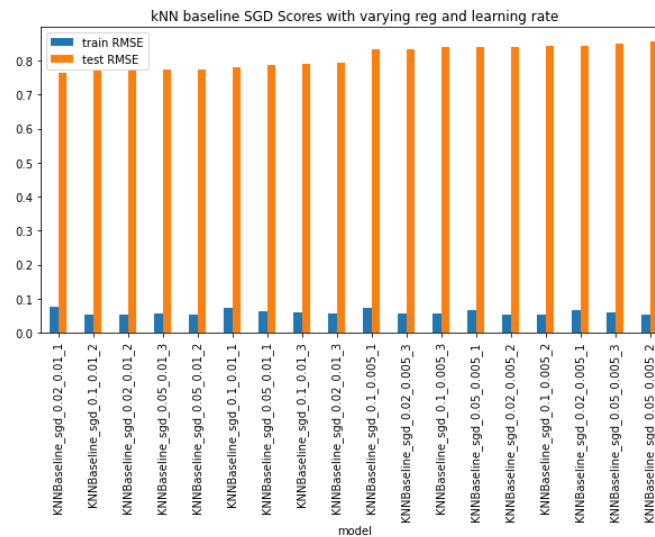


Figure 9 - KNN scores with varying regularization and learning rate

Based on the above table and bar chart, the chosen model has a regularization of 0.02, a learning rate of 0.010, and k=1. This yields an RMSE of 0.7235 on the test set.

### Hyper tuning the KNNBaseline using the ALS method:

ALS is a two-step optimization process to solve for sparse data. First, it fixes the rows for a factorized matrix and then uses the row to fix columns.

Hyperparameter Tuning - ALS With als, there are two that we can tune: reg\_i and reg\_u.

reg\_i is the regularization parameter for items reg\_u is the regularization parameter for users.

The variations are as follows.

Algorithm	regularization_i	regularization_u	k	Train RMSE	Test RMSE
KNNBaseline_als	20	30	3	0.05735	0.910142
KNNBaseline_als	20	30	3	0.056414	0.913836
KNNBaseline_als	20	30	1	0.068604	0.930681
KNNBaseline_als	20	30	2	0.057223	0.931621
KNNBaseline_als	20	30	1	0.079207	0.939833
KNNBaseline_als	20	30	2	0.053781	0.939842
KNNBaseline_als	40	60	2	0.058216	0.952319
KNNBaseline_als	40	60	3	0.062698	0.962818
KNNBaseline_als	40	60	2	0.058973	0.967163
KNNBaseline_als	40	60	3	0.055268	0.9704
KNNBaseline_als	40	60	1	0.076488	0.97313
KNNBaseline_als	40	60	1	0.069881	0.989275

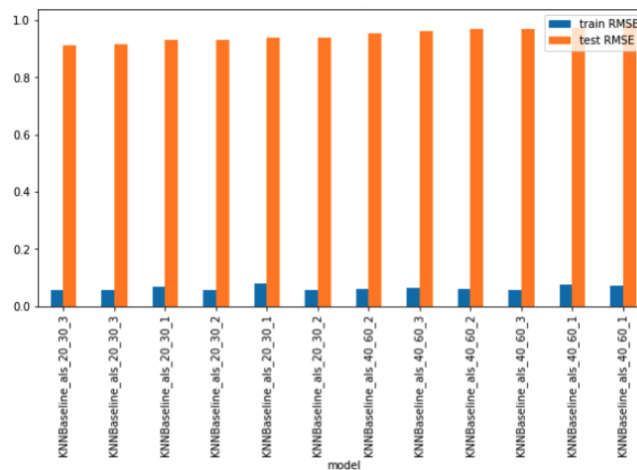


Figure 10 - KNN scores with varying reg\_i and reg\_u

The best model is the one with reg\_i of 20, reg\_u of 30, and a k of 3. Applying this to the test set gave an RMSE of 0.8808

### The best of the chosen models:

Description	Model	RMSE
kNN without baseline	KNNWithZScore_pearson_1	0.6456
kNN with baseline_SGD	KNNBaseline_reg=0.02_learningrate=0.01_k=1	0.7235
kNN with baseline_ALS	KNNBasic_reg_i=20_reg_u=30_k=3	0.8808

KNNWithZScore model using the Pearson similarity and a k of 1 is the best performer of the chosen lot.

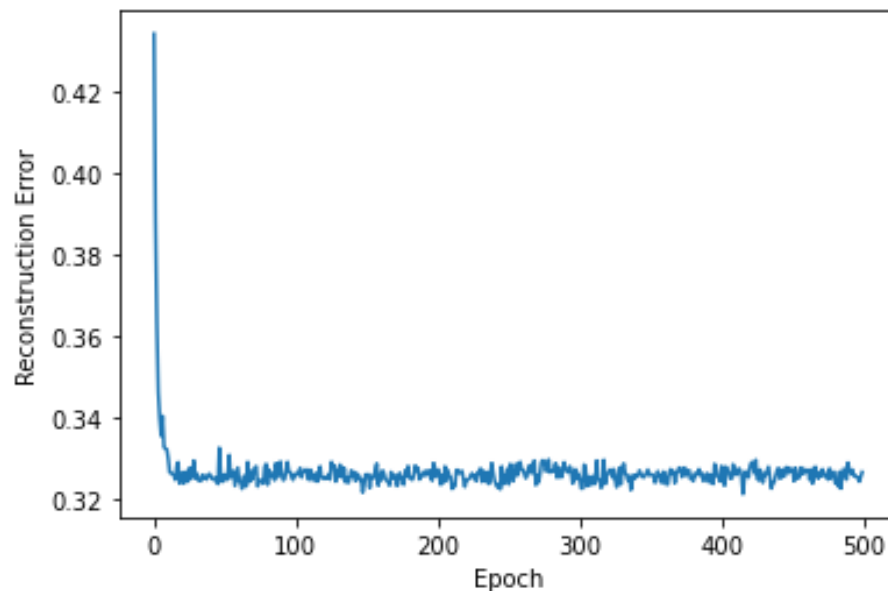
### Analysis of the Restricted Boltzmann models and autoencoders:

Restricted Boltzmann models are not predominantly in use and have been long deprecated in favor of autoencoder models in deep learning. However, we experimented with RBM models for ease of implementation and simplicity and to get a feel of how the dataset would potentially behave with autoencoder models. As we know that our dataset is very sparse and that RBM or autoencoders models would require high volumes of training data, this is a good place to start our understanding.

### RBM with a sample size of 100 unique users and 100 unique papers:

The main function of RBM models and or autoencoders is the reconstruction of the input into the output. Looking at the reconstruction error for the RBM model with 100 unique samples, we can see that as the number of epochs increases, the training error comes down. This is the typical

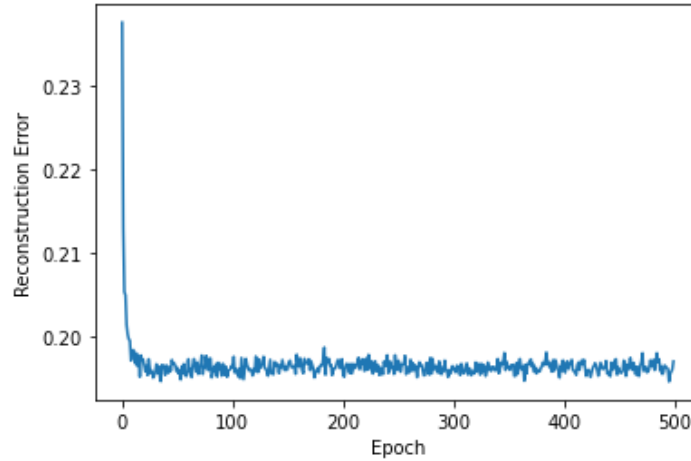
behavior of the RBM models. The reconstruction error graph overall looks impressive, which means that the model can learn.



However, the RMSE for the test set was very high at **6.811**. This may be attributed to the sparse dataset and the lack of opportunity for the model to train and to meet the standards of predictions we expect from it. This implies that the dataset is not compatible with deep learning models like RBM and autoencoders, but we took this as an opportunity to understand how deep learning models like Autoencoders and their precursors would behave with a typical sparse dataset.

### **RBM with a sample size of 200 unique users and 200 unique papers:**

Though we know intuitively that the RBM model needs more data to train and give better results, it is imperative to secure the evidence to justify the high RMSE value. When we increased the sample size to 200, there was a drop in RMSE. However, the ability of the model to reconstruct the data is impressive. It remained slightly lower than the variance range as the 100-sample model.



The 200 sample RBM model produced a test RMSE of **6.104**. Though this is a high value, the minute drop in error with an increase in samples shows that we can still put autoencoders to use as our user profile dataset grows.

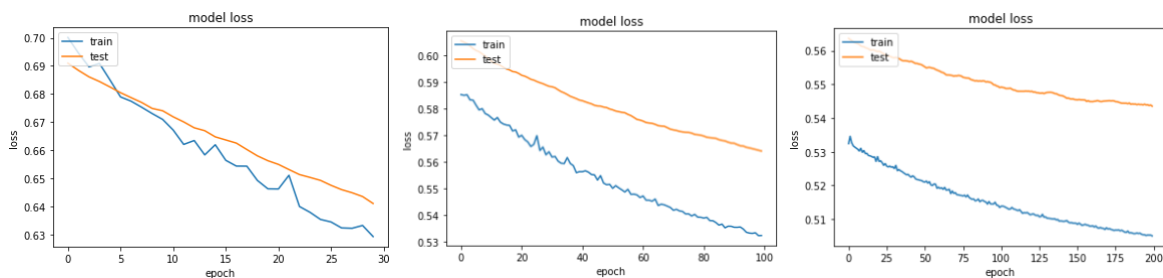
#### **Analysis of the embedding matrix model:**

Embedding matrix models are known to work well even in the absence of large training data. As expected, the model was able to perform well at 30 epochs with a training loss at 0.69. The test loss was coming close to 0.71 with 30 epochs.

At 100 epochs, we saw a training loss of 0.588 and a test loss of 0.62.

At 200 epochs, we saw a training loss of 0.535 and a test loss of 0.567.







Beyond 300 epochs, we could see that the model was struggling to fit the data properly during the test, so we kept the ideal epoch number at 200.



*Figure 11- A model loss for the Embedding matrix model with epochs at 30,100 and 200.*

Looking at the plot for the loss and epoch, we can see that the model is not overfitting, and it can distinguish the similarities and differences present in the dataset. This is a promising result for us as we can explore more into the deep learning realm even with relatively sparse data.

**The final comparison of the models can be seen below:**

Description	Model		RMSE
kNN without baseline	KNNWithZScore_pearson_1		0.646
kNN with baseline_SGD	KNNBaseline_reg=0.02_learningrate=0.01_k=1		0.723
kNN with baseline_ALS	KNNBasic_reg_i=20_reg_u=30_k=3		0.880
KNN Baseline	KNN basic algorithm with baseline		0.489
Restricted Boltzmann machines	RBM with 200 samples		6.104
Embedding matrix model	Embedding matrix model with 200 epochs		0.567

The basic KNN algorithm implementation using the Surprise library shows the best results for our dataset now. In general, KNN algorithms have proven to be very successful for rating prediction for the dataset we have generated. The embedding matrix factorization model has also shown good performance with little data to work with. As our user profile grows with more users and explicit ratings, we can venture into the deep learning realm of models and see if we can come up with better personalization.



## 8. Conclusion

From our extensive analysis of the recommendation systems and the algorithms used we can see that we do not always need extra powerful algorithms or large amounts of training data. There is a lot of scope in using basic machine learning algorithms and harnessing their ability to make predictions. Many times, deep learning algorithms need extensive training, large data handling capabilities, powerful computing resources to handle the large number of calculations which all add to the complexity and cost of scalability. In our case, the KNN algorithm can produce accurate results with a simpler implementation, less use of resources, and the need for training data.

Though at an experimental level we aim for higher accuracies, and we are willing to go the extra mile to code complex algorithms, there is always a need for simpler and cost-effective solutions especially when we need to scale up the solution. There is always a tradeoff between performance and cost. Hence depending on the business case, datasets, and resource constraints we can choose the right kind of model for our problem.

## 9. Prospects

The prospects for this project would be in improving the user profiles, collecting more user preferences to build the knowledge base. This will help the models perform better and provide accurate personalization.

Building a web application with an increased ability to handle unstructured data in the backend along with data collection capabilities will give a well-rounded experience for the user. Now we are primarily focusing on collaborative filtering and deep learning algorithms to produce rating predictions and narrow down recommendations. We should explore more in the areas of hybrid

recommendation systems and graph-based recommendation systems to develop a truly AI-driven recommendation.

More visualized representations of the connected scientific publications will help the researchers explore more knowledge materials. A graph network of connected articles, authors, and publications will be an efficient tool to reduce literature review times.

## 10. User privacy and legal implications

Recommender systems have developed a reputation for user privacy concerns and excessive data collection across applications for any user. This is a concerning trend and it needs to be addressed as part of this study. Recommender systems and allied algorithms constantly collect user information with the sole aim of providing better content recommendations. However, this data contains a lot of user-sensitive information, personal information, in our application, there may be data pertaining to research and patents. As the solution providers, it is within our scope and abilities to restrict the collection of extra-sensitive information that will put the user's privacy under threat. Some of the steps to handle these concerns are,

1. Let the user know what kind of data will be collected while using the algorithms.
2. Give the user the freedom to choose the level of personalization and restrict data collection according to the levels.
3. Protect sensitive information that gets collected and stored.
4. Adhering to the local government's user privacy laws.
5. Establishing best practices from previous experiences to create user-friendly policies and help protect sensitive data.

## 11. Acknowledgments

Dataset:

Geiger, R. Stuart, 'ArXiV Archive: A Tidy and Complete Archive of Metadata for Papers on arxiv.org', <http://doi.org/10.5281/zenodo.146324>

Surprise library and documentation:

Referred and quoted the documentation for formulas and explanations.

Implementation of Surprise library scikit for several of our recommenders

<http://surpriselib.com/>

Keras library and tutorials:

Embedding matrix implementation and evaluation.

<https://keras.io/about/>

## 12. References

1. X. Bai, M. Wang, I. Lee, Z. Yang, X. Kong and F. Xia, "Scientific Paper Recommendation: A Survey," in *IEEE Access*, vol. 7, pp. 9324-9339, 2019, doi: 10.1109/ACCESS.2018.2890388.
2. Tauchert, Christoph & Bender, Marco & Mesbah, Neda & Buxmann, Peter. (2020). Towards an Integrative Approach for Automated Literature Reviews Using Machine Learning. 10.24251/HICSS.2020.095.
3. Harshdeep Gupta, Recommender Systems using Deep Learning in PyTorch from scratch. Retrieved from <https://towardsdatascience.com/recommender-systems-using-deep-learning-in-pytorch-from-scratch-f661b8f391d7>  
[https://github.com/HarshdeepGupta/recommender\\_pytorch](https://github.com/HarshdeepGupta/recommender_pytorch)

4. Eric Le, 'How to build a simple song recommender system'. Retrieved from  
<https://towardsdatascience.com/how-to-build-a-simple-song-recommender296fcbc8c85>
5. Anne-Marie Tousch, How robust is MovieLens? A dataset analysis for recommender systems  
arXiv:1909.12799v1.
6. Rounak Banik, Hands-On Recommendation Systems with Python: Start Building Powerful and Personalized Recommendation Engines with Python.