F-Droid Application: Arxiv Papers

Yiqiao Yin
W.Y.N. Associates, LLC

Abstract

This is the capstone project of this course. This report examines an F-Droid application called ArXiv Explorer that search, archive, and download papers at ease and with a move of finger. The report has two parts. First, the report investigates the ArXiv Explorer application. Next, the report discusses the procedure of design and launch of this project. The report also explores interesting facts that can be provided to the readers including benefits and area of improvements. Last, the report provides several novel solutions to tackle the areas for improvements. For the second part, the project has also finished a complimentary presentation slides where a video recording is prepared.

1 Introduction

This study reviewed more than 200 research papers and investigated the research-paper recommendation system (Beel et al., 2016). Amongst these 200 studies, approximately 55% of the papers are content-based filtering, 18% of the papers are collaborative filtering, and 16% of the papers are graph-based recommendations. There are other recommendation concepts and they proposed stereotyping, item-centric recommender, and hybrid recommender systems. Based on citation contexts, many evaluations have attempted to develop tasks and models to be used on paper recommender systems Teufel et al. (2006); Ghosh et al. (2016); He et al. (2010); Ebesu and Fang (2017); Mohammad et al. (2009); Chen and Zhuge (2019)

2 Examination of ArXiv Explorer Repository

This section of the report focus on the examination of ArXiv Explorer Repository. The section consists of the following subsections.

- "README": This subsection summarizes the benefits of a README file for the repository of ArXiv Explorer.

- "Features": This subsection covers the important features of the package at interest.

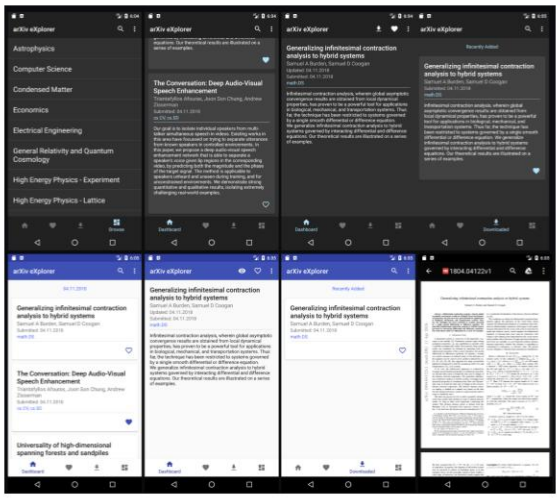


Figure 1: **Screenshots of ArXiv Explorer Application UI**.

- "Download Instructions": This subsection introduces the instructions of downloading such package and some user manual of navigating the package.

- "Repository Management": This subsection covers the important pipeline of the management of data-flow of this repository.

2.1 README

The repository of ArXiv Explorer has an instruction manual to lead readers through steps of installation and deploying the application. This instruction manual is known as a "READ ME" file. This READ ME file covers a list of features used in the application. In addition, it also provides links to deploy on Google Play (mobile application store of android devices). The F-Droid link is also made available for collaboration purposes.

The most useful information and import features from the User Interface (UI) is presented in Figure 1. These are the screenshots of the application UI. The application has navigation menu in the bottom of the screen. For simplicity, the application has four buttons. The "Home" button is placed first to navigate to the main dashboard. The "Favoite" button (the second one that looks like a heart shape) is a location when the user saved articles that are to be read later. The "Download" button (the third one that looks like an arrow pointing down) is a function key to allow offline reading. The "Browse" button (the fourth one that looks like a grid of squares) is to view a list of articles in ArXiv Explorer.

2.2 Features

The entire main function is written in java. The most fundamental feature of the main function is the "browse" functionality. To establish the connection, an ArXiv Application Programming Interface (API) is required instantiate the connection from local server to the cloud server of the ArXiv website. The "browse" function has an access key which is

used to extract the information of the paper. There is a "getPapers" function coded inside the "browse" script and related attributes are also built in such as sort by order, sort by group, and maximum result to ensure the UI is comfortable to the end-users. There is also a "callback" function in the "browse" script to respond to a user interaction. Upon a click of the button, a query request is sent and the information of the paper is extracted.

The next fundamental feature of the main function is "dashboard". The "dashboard" functionality is wrapped around the "browse" function, because the "dashboard" will generate a list of papers of which the information is extracted using "browse" functions. To ensure the query list is comfortable and easy to read, there is a "toggleCategory-Names" function written in the script. This implementation is to ensure categories and subcategories are up to date and limited to certain length so that readers do not have to scroll through pages of papers before they see the next section. In this script, information of the paper can be concatenated and added to the list for review.

The "category" is introduced above as a function. However, the behind-the-scene action of "category" is actually written as an independent script. The script starts off with a "viewHolder" where information of papers can be hosted and stored as a placeholder. Then functionalities such as "binding", "recycling", "rowCount" are implemented to allow further categorization of list of queries. Afterwards, there is also a "category presenter" script.

An attribute built upon "category" is the "category presenter" feature. This feature allow the article in designated category to be reviewed upon a click motion, and it is programmed as an reactive feature with the backend.

A download attribute is also made available for the UI. This feature allows users to read offline and without the access of the internet after the file is successfully downloaded. The feature currently is not supported with particular VPN setting.

The main model feature in the script is a function called "model". The model is defined using a "Paper" function where the function is a public object defined globally in the application. This object carries attributes such as title, author, summary, publish date, paper ID, URL, category, and so on. The object, once defined and fed into the information, should present the end-user a full list of information from its attributes. The API is presented in the following.

```
public Paper(String title,
             String author,
             String summary,
             String updatedDate,
             String publishedDate,
             String paperID,
             String pdfURL,
             String categories,
             String paperURL) {
    this.title = title;
    this.author = author;
    this.summary = summary;
    this.updatedDate = updatedDate;
    this.publishedDate = publishedDate;
    this.paperID = paperID;
    this.pdfURL = pdfURL;
    this.categories = categories;
    this.paperURL = paperURL;
    this.downloaded = false;
```

```
108     this.favorited = false;
109     this.read = false;
110 }
111 
112 public String getTitle() {
113     return title;
114 }
115 
116 public void setTitle(String title) {
117     this.title = title;
118 }
119 
120 public String getPublishedDate() {
121     return publishedDate;
122 }
123 
124 public String getPaperID() {
125     return paperID;
126 }
127 
128 public String getSummary() {
129     return summary;
130 }
131 
132 public String getPDFURL() {
133     return pdfURL;
134 }
135 
136 public String getAuthor() {
137     return author;
138 }
139 
140 public String getUpdatedDate() {
141     return updatedDate;
142 }
143 
144 public void setDownloaded(boolean downloaded) {
145     this.downloaded = downloaded;
146 }
147 
148 public void setFavorited(boolean favorited) {
149     this.favorited = favorited;
150 }
151 
152 public void setRead(boolean read){this.read = read;}
153 
154 public String getCategories() {
155     return categories;
156 }
157 
158 public String getPaperURL() {
159     return paperURL;
160 }
161 
```

The "paper" object is fed in the query of information from an API developed called "network". The network is an API that feeds access into the paper object. It requires a sufficient URL as "baseURL" which is a parameter in the ArxivAPI object. This ArxivAPI is a class object in java and it is built to ensure the client information is communicated without error. A high level brief of the code structure is provided in the following.

```
public class ArxivAPI {

    private static final String baseURL = "http://export.arxiv.org
        ↪ /api/";
    private static final String querySegment = "query";
    private static OkHttpClient client;

    public static OkHttpClient getClient() {
        if (client == null) {
            client = new OkHttpClient();
        }
        return client;
    }
    ...
}
```

The "ArxivAPI" class object is supported by a builder function of which arguments are parsed into the object to be able to carry on the attributes throughout the rest of the script. There is also a private static object that is called "addANDParamToQuery", which allows the app to concatenate the query, key, and parameters together to form a more complete query names.

There is a public search function that can allow users to enter search queries such as name, category, author, title, and so on. This search query, upon request of the user, then sends the information request into the app. The app can use the information to retrieve the corresponding articles, assuming that there is valid internet connection. If there is not a completely match of those key words, a close or a similar one will be provided. Afterwards, the same "download" feature is introduced to allow user to download the paper if desired.

Another important feature is the search algorithm. The current edition of the app provides UI of the search function. However, the query is fed into ArXiv using URL and the list of results are directly retrieved using this URL provided sufficient internet connection. However, this is sub-optimal, because the app itself does not carry any search algorithm and can be improved based on the users' needs.

2.3 Download Instructions

The ArXiv Explorer application or package can be downloaded using the following code.

```
git clone https://github.com/GarrettBeatty/arXiv-eXplorer.git
bundle install
```

2.4 Repository Management

The ArXiv Explorer application is publicaly available on F-Droid website. In addition, the source code of the application can be found on the Github with the following URL.

It is made very convenient for software developers. If desired, one can simply go to the following URL to downlaod the source code and install the package.

```
https://github.com/GarrettBeatty/arXiv-eXplorer
```

3 Areas of Improvement

This section discusses some areas of improvement.

3.1 AI-based Solution

Despite many of its accomplishments, it is still unclear which recommendation system is in effect and which can lead to promising search results during literature review.

3.2 Category Refinement

3.2.1 User Recommendation

The category has attributes such as category presenter mode. With a click of a button, the presenter mode present summary information such as the title, author, and abstract. However, to further tailor the need of the software package to researcher or PhD student, it is often times beneficial to have the paper linked to a certain author or topic.

This is an area where user recommendation system can be implemented heavily while the current version of the package is not equipped. In research period of graduate program, the literature review is heavily built upon and tailored by each individual research topic. Due to this nature, the review process could be drastically different and varying student by student. In this case, a conventional system without user recommender could easily provide insufficient or unrelated articles that may be a huge waste of time, because it is up to the end-user to screen and filter whether the article can be used. However, if features about the article can be extracted (which according to the current design of the package it does have), then it is a small leap of a way to introduce machine learning or even AI-based solutions.

3.2.2 Build Machine Learning Model for User Recommendation

To further investigate this direction, let us denote the interest of an article to be the target which is what the model should be producing as a first screening process. Denote this target Y. The features such as author, topic, keywords, abstract, can then be extracted as features, which can be denoted as X. The task is to learn from X and try to produce Y. In other words, the desired task is a supervised learning model that attempts to understand and potentially guess the real relationship between X and Y. In other words, there exist in nature a map, $f(\cdot)$ such that $Y = f(X)$. The goal is to train a machine to understand and make the best guess that it can about the map, hence, the goal is to search for the best $\hat{f}(\cdot)$ such that $\hat{Y} = \hat{f}(X)$ with the least error possible. In this learning model, the error is measured between Y and $\hat{Y}$ where Y can be the real target in the training data and $\hat{Y}$ is the educated guess from the model based on the features. The proposed idea can be illustrated in Figure 2.

In this proposal, it is important to write the application to create a user-centric training data set. The idea is that the application is built to learn from each and every user. There could be an environment built and to collect the data only about the user at interest. This

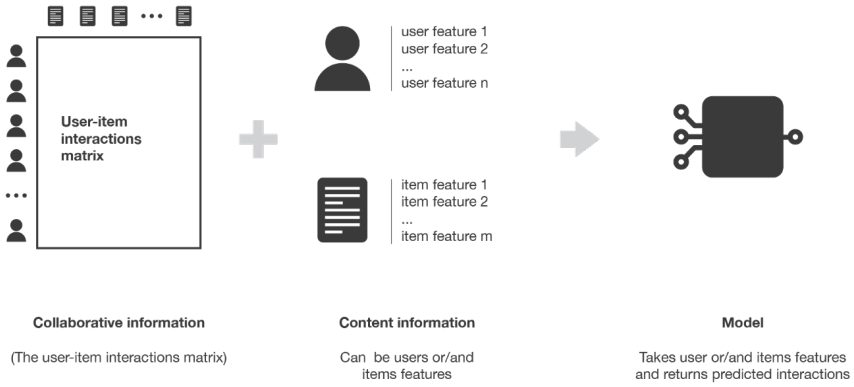


Figure 2: **User Recommendation System**. How does it work?

way the target variable Y is tailored to the interest and the passion of that user only, which can be much more effective at recommending articles.

The downside of this approach is the potential lack of the data in the beginning of the user experience, which is the most crucial component of the user period. During this time, if the user does not use the application often enough for the machine to be well trained, it is possible to create similar dataset based on the user's demographics information. For example, a form can be created to collect this information at the sign-up stage. Information such as name, school, education level, researcher status, and so on can be extracted to build the model.

The following models can be available candidates for this task. For simple linear regression, it takes the form of

$$Y = \beta_0 + \beta_1 X + \epsilon \tag{1}$$

where Y is the target which states the interest a user has on an article and X being a feature such as education level. Under general scenario, the model can have multiple features or explanatory variables, and hence can take the form of

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon = \beta_0 + \sum_{j=1}^{p} X_p + \epsilon \tag{2}$$

where the explanatory data set is a p dimensional feature. In practice, the assumption of continuous variable for Y rarely holds true. An alternative is to use logistic regression model. To transform into logistic regression model, a non-linear link function such as sigmoid is used. The linear model is fed into the link function and hence produce the educated guess. The logistic regression model takes the following form

$$Y = \frac{1}{1 + \exp\big(-(\beta_0 + \sum_{j=1}^{p} \beta_j X_j)\big)} \tag{3}$$

where the sigmoid function $1/(1 + \exp(-z))$ maps the output onto a range from 0 and 1. Hence, this is also called a probabilistic model. The logistic model is actually a very simple neural network with one neuron. Instead of using ordinary least square to search for the parameters, we can use optimization techniques to search for the parameters. This is the fundamental idea for neural networks. A simple neural network is demonstrated in

Figure 3: **Diagram of an Artificial Neural Network**. Artificial Neural Network or ANN has input layers, hidden layers, and output layers. These layers are fully connected. An ANN regressor has one neuron in the output layer and an ANN classifier has an output layer with a number of neurons the same with the number of classes.

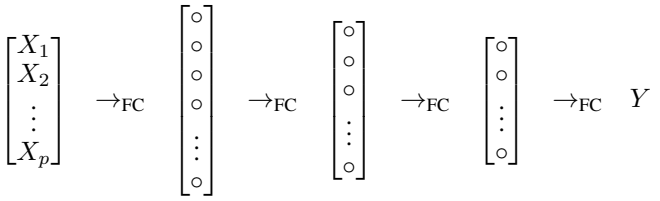
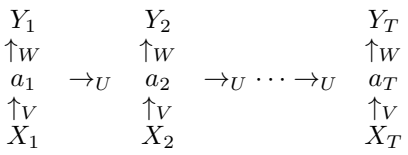


Figure 4: **Diagram of a Recurrent Neural Network**. A Recurrent Neural Network or RNN has layers designed as a Markov Chain. The weights are shared across different time stamps. Unlike ANN, RNN is not fully connected, because the time stamp is an important feature to capture in the network architecture.



the following diagram, Figure 3. The Artificial Neural Network or ANN has an input layer which are the explanatory features in the data, i.e. denoted as $X_1, X_2, ..., X_p$. The input layer is sent into the ANN architecture with 3 hidden layers. Each hidden layer has certain number of neurons where the actual number is a tuning parameter. The layers are fully connected, denoted as an arrow with "FC". This means that each neuron from the prior layer is connected with each of the neuron in the next layer. The final layer outputs a prediction which is the target.

The data serves as the key component of the research question in user recommendation system. In the beginning, it is discussed above that the system might be lack of data because the user is in the beginning stage of using the package. However, as time moves on there might be a change of situation where the user starts to use the application more dependent on their research demands. The alternative can also be true as well where a user might find the application not helpful and stops using it. In this situation, information changes across time and the features also change across time. It is somewhat beneficial to develop a Recurrent Neural Network to tackle and address some of the sequential pattern in the dataset. A Recurrent Neural Network (RNN), just like Artificial Neural Network (ANN), is constructed using forward propagation and backward propagation. However, the difference is that in RNN the features have time stamps, which is not the case for ANN. The diagram of a RNN is presented below.

3.2.3 Interpret Linear Regression Models

Another area to focus on is the interpretability of the machine learning component in category refinement. The purpose of category refinement is to be able to provide students and other end-users better and improved recommended articles in increase the

online traffic of the application. In addition to build more complicated models for better performance, the implication of models need to be analyzed as well. This functionality, if implemented well, will be able to give the organizers fundamental ideas what additional articles to recommend and why. In other words, a key component to introduce is the model interpretability and explainability of the machines used to predict category ratings of a user. In many cases, this can be treated as a model interpretability problem or even an inference problem.

In simple terms, a linear model is fundamentally intepretable. There is no additional work needs to be done to try to explain the model. A linear model such as one that follows ordinary least square (OLS) assumption searches for the linear coefficient of the model by using least square assumption. Denote the target to be Y and the educated guess to be $\hat{Y}$. The least square formula is written as

$$\sum_i (Y - \hat{Y})^2 \tag{4}$$

where the summation has a running index i to track the number of observations. If it is assumed that the model is a simple linear regression model, then it takes the form of equation 1. Under the OLS assumption, the least square is used (equation 4) to find the linear coefficient. The educated guess from the model is $\hat{Y}$, so the model formula can be used to replace it in order to update the least square formula.

$$\sum_i (Y - (\beta_0 + \beta_1 X))^2 \tag{5}$$

and the goal is to search for the linear coefficients β_0 and β_1. This can be done by treating it as an optimization problem. At the optimal point, the gradient of least square is valued at 0. Hence, the set of equations can be set up as the following

$$\begin{array}{l} \frac{\partial}{\partial \beta_0} \sum_i (Y - (\beta_0 + \beta_1 X))^2 = 0 \\ \frac{\partial}{\partial \beta_1} \sum_i (Y - (\beta_0 + \beta_1 X))^2 = 0 \end{array} \tag{6}$$

where the system is a two-equation-two-unknown. Hence, it can be solved and the OLS solution can be obtained

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}, \hat{\beta}_1 = \frac{\sum_i (X_i - \bar{X})(y_i - \bar{Y})}{\sum_i (X_i - \bar{X})^2} \tag{7}$$

One important assumption for simple linear regression model is that the variance of the model is constant, i.e. σ^2 is constant. Though it is difficult to defend this assumption in practice, it does provide some convenient interpretation for us to understand the model once it is fitted. using σ^2, it is mathematically sufficient to derive the following results

$$\text{SE}(\beta_0) = \sqrt{\frac{\sigma^2 \sum_i X_i^2}{n \sum_i (X_i - \bar{X})^2}}, \text{SE}(\beta_1) = \sqrt{\frac{\sigma^2}{\sum_i (X_i - \bar{X})^2}} \tag{8}$$

These estimators allow the end-users to make inference on the linear coefficients sought under the OLS assumption. For example, a simple model can be built to predict the housing price using square footage. The housing price is continuous and so is the square footage. One practical question useful for us is: "does square footage affect the housing price?" This question can be answered using linear regression model. A linear regression model can be built using square footage as X to explain the housing price Y. The housing price, under OLS assumption, is a linear additive form of the square footage. The linear coefficients along with their standard error are all known to the end-users

once the model is fitted. To answer the above question, a hypothesis test can be set up in the following form

$$\begin{aligned} H_0 &: \text{Square footage has no impact on housing price} \\ H_1 &: \text{Square footage has a big impact on housing price} \end{aligned} \tag{9}$$

where the alternate hypothesis points to the obvious answer that end-user has in mind and the null hypothesis is set up as the opposite end of the stick. To translate this into a convention hypothesis testing problem with linear coefficients, the statement can be rephrased in the following form

$$\begin{aligned} H_0 : \beta_1 = 0, &\quad \text{there is no coefficient,} \\ &\quad \text{hence the square footage should not be in the model,} \\ &\quad \text{so it's not important} \\ H_1 : \beta_1 \neq 0, &\quad \text{the coefficient should not be zero,} \\ &\quad \text{hence the square footage should be in the model,} \\ &\quad \text{so it's important} \end{aligned} \tag{10}$$

To execute the hypothesis test, it is important to find the test statistics which is computed as the follows

$$\mathrm{t} = \frac{\hat{\beta}_1 - 0}{\mathrm{SE}(\beta_1)} \tag{11}$$

and the statistics follow standard normal distribution under the null hypothesis. This means that if the null hypothesis is correct the scientist should expect the test statistics t to be small. Alternatively, if the test statistics is sufficient large, it gives the scientist more confident to say that there is enough evidence to reject the null hypothesis. Hence, it is possible to make inference which conclusion is more confident to be used. This helps the end-user to establish basic intuition of whether the square footage is important. It will be important if the test statistics is high. In addition, it is also available to end-user the marginal impact the square footage has on housing price, because that information is registered in the linear coefficient. The value of the estimator $\hat{\beta}_1$ indicates the numerical changes on Y given that X changes by one unit. This is the crucial information to have and it is fortunate by product coming with the model. Hence, it is theoretically sound to say that linear regression models are interpretable. There is no additional work required to help end-user to understand the internal structure after the model is fitted.

In addition, an example that is tailored to the ArXiv Explorer can be shown and discussed in the following. Suppose the goal is to predict how much a user likes an article given the features describing the article. Then the "interest" can be rated in 1-10 stars. In this case, it can be treated as a continuous variable. Suppose the feature that the data scientist is interested in is the length of the abstract. Then the length of the abstract can be X while the rating of the article can be Y. A linear model can be built using $Y = \beta_0 + \beta_1 X$ and the linear coefficients can be computed. To answer the question: does length of the abstract affect the user's rating of the article? It is convenient to compute the standard error of the β_1 estimator and a hypothesis test can be established. The test statistics $\hat{\beta}_1/\mathrm{SE}(\beta_1)$ can be computed and then it can be compared with the z-score (the statistics under standard normal distribution).

3.3 VPN for Download

The download attribute is not supported via VPN. This could be a problem where in some countries the internet server may or may not have access to certain websites.

A diagram of the VPN working procedure is provided in Figure 5. The user device is connected to the internet service provider. There are two scenarios provided in Figure

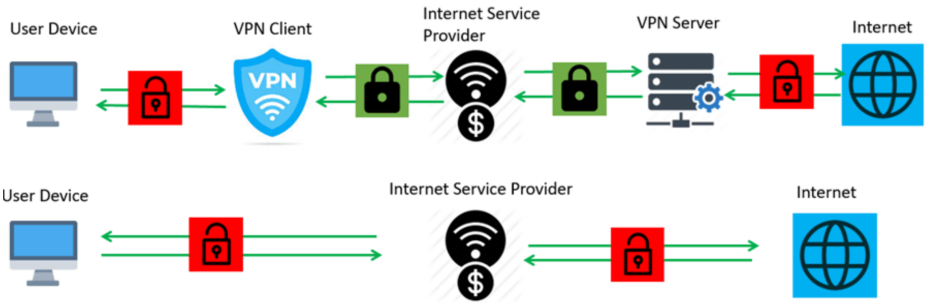


Figure 5: **VPN diagram**. How does it work?

5. The one in the bottom is the conventional scenario without the aid of VPN to secure the information. Then the information goes out of the internet service provider and is released to the internet. In this scenario, the user device directly communicates with the internet service provider and the information is not encrypted.

When VPN is provided, it serves as an additional client server to filter the information, of which is the information will be encrypted before it communicates in and out. Specifically, the VPN is set between the user device and the internet provider. The information coming out of the internet will be encrypted before it goes into the user device.

3.4 Search Algorithm

The current download function and the category function do not have any sorting or ranking system provided inside the application. This is an area of improvement, because the query produced from the Arxiv website may or may not provide desired order for the user. As the first-hand UI to interact with the human user, it is essential to learn and understand what the user needs and intends to read. This is where a potential PageRank-like algorithm can be born.

PageRank Algorithm, the founding algorithm for Google, is a linkage and sorting analysis algorithm that assigns a numerical weight score to each entry of a list of documents provided from the database. In this case, the database is the World Wide Web or WWW.

For simplicity, let us say there are only 3 documents in this small toy universe. Let us denote them to be A, B, and C. The initialization of the PageRank (denoted as $\mathcal{R}$) is equal weight. Hence, we have $\mathcal{R}(A) = \mathcal{R}(B) = \mathcal{R}(C) = 1/3$. Now suppose B and C has links to A, then each of them would transfer their value to A according to the following formula

$$\mathcal{R}(A) = \mathcal{R}(B) + \mathcal{R}(C) \tag{12}$$

and hence A would be valued at $1/3 + 1/3 = 2/3$.

In a separate situation, suppose the document B has a link to both C and A while the document C has a link to document A only. Then B has to allocate points to A and the same thing goes for C. However, to value the PageRank for A, the scores need to be divided by the number of outbound links $\mathcal{L}$. Thus, in this case, the PageRank for A is stated as the following

$$\mathcal{R}(A) = \frac{\mathcal{R}(B)}{\mathcal{L}(B)} + \frac{\mathcal{R}(C)}{\mathcal{L}(C)} = \frac{1}{2} + \frac{1}{1} = \frac{3}{2} \tag{13}$$

In general situation, we can formally write the PageRank score for any document d to be

$$\mathcal{R}(d) = \sum_{d' \in \Pi} \frac{\mathcal{R}(d)}{\mathcal{L}(d')} \tag{14}$$

where Π is the document space and d' is a dummy variable to trace every document in the document space.

A sample PageRank algorithm in python is provided below.

```
"""PageRank algorithm with explicit number of iterations.

Returns
-------
ranking of nodes (pages) in the adjacency matrix

"""

import numpy as np

def pagerank(M, num_iterations: int = 100, d: float = 0.85):
    """PageRank: The trillion dollar algorithm.

    Parameters
    ----------
    M : numpy array
        adjacency matrix where M_i,j represents the link from 'j'
            ↪ to 'i', such that for all 'j'
        sum(i, M_i,j) = 1
    num_iterations : int, optional
        number of iterations, by default 100
    d : float, optional
        damping factor, by default 0.85

    Returns
    -------
    numpy array
        a vector of ranks such that v_i is the i-th rank from [0,
            ↪ 1],
        v sums to 1

    """
    N = M.shape[1]
    v = np.random.rand(N, 1)
    v = v / np.linalg.norm(v, 1)
    M_hat = (d * M + (1 - d) / N)
    for i in range(num_iterations):
        v = M_hat @ v
    return v

M = np.array([[0, 0, 0, 0, 1],
              [0.5, 0, 0, 0, 0],
              [0.5, 0, 0, 0, 0],
              [0, 1, 0.5, 0, 0],
              [0, 0, 0.5, 1, 0]])
v = pagerank(M, 100, 0.85)
```

References

Beel, J., Gipp, B., Langer, S., and Breitinger, C. (2016). Paper recommender systems: a literature survey. *International Journal on Digital Libraries*, 17(4):305–338.

Chen, J. and Zhuge, H. (2019). Automatic generation of related work through summarizing citations. *Concurrency and Computation: Practice and Experience*, 31(3):e4261.

Ebesu, T. and Fang, Y. (2017). Neural citation network for context-aware citation recommendation. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, pages 1093–1096.

Ghosh, S., Das, D., and Chakraborty, T. (2016). Determining sentiment in citation text and analyzing its impact on the proposed ranking index. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 292–306. Springer.

He, Q., Pei, J., Kifer, D., Mitra, P., and Giles, L. (2010). Context-aware citation recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 421–430.

Mohammad, S., Dorr, B., Egan, M., Hassan, A., Muthukrishnan, P., Qazvinian, V., Radev, D., and Zajic, D. (2009). Using citations to generate surveys of scientific paradigms. In *Proceedings of human language technologies: The 2009 annual conference of the North American chapter of the association for computational linguistics*, pages 584–592.

Teufel, S., Siddharthan, A., and Tidhar, D. (2006). Automatic classification of citation function. In *Proceedings of the 2006 conference on empirical methods in natural language processing*, pages 103–110.