# PERSONALIZED SCHOLARLY ARTICLE RECOMMENDATIONS BASED ON THE RECURRENT NEURAL NETWORK AND PROBABILISTIC MODELS

Eric Brewer, Yiu-Kai Ng, and Alisha Banskota
Computer Science Department
Brigham Young University
Provo, Utah 84602, USA
eric.r.brewer@gmail.com, ng@compsci.byu.edu, baskotaalisha@gmail.com

**ABSTRACT**

Searching scholarly articles, i.e., research papers, on the Web is a challenge even for researchers who are familiar with the searched topics and articles of a particular domain of interest, needless to say for users who are not familiar with the areas of study. Existing research paper recommendation systems either rely on the content-based or collaborative-filtering approach, or its hybrid model. These recommenders, however, are vulnerable to the cold-start problem, either on new users or new scholarly articles. In this paper, we propose an innovative personalized scholarly article recommendation system which suggests research papers of the same subject area using the recurrent neural network model and ranks closely related research papers using the BM25 probabilistic model. Experimental results based on the titles and abstracts of research papers extracted from ACM DL and IEEE Xplore digital library verify the merit of the proposed recommender, which outperforms recently-developed research paper recommendation systems.

**KEYWORDS**

Scholarly article recommendation, deep learning, probabilistic model

## 1 Introduction

According to the 2016 yearly report compiled by Nature (nature.com/articles/d41586-018-00927-4), an International Journal of Science, the combined number of science and engineering research articles published by the European Union, United States, and China adds up to more than 1.4 million. One can imagine the huge number of scholarly articles accumulated over the past two decades alone. The number is a promising trend, since it indicates the progress of various academic disciplines with insightful and innovative ideas developed to improve ordinary lifestyle. The abundant of existing research articles, however, introduces a constant challenge to researchers, i.e., how to find relevant articles addressing a particular scientist problem efficiently?

Searching for references on scholarly articles tailored to a particular problem, ordinary users often rely on keyword search through Google Scholar and popular search engines, such as Google, Bing, or Yahoo!. Researchers also conduct their search using ACM DL (dl.acm.org) and IEEE Xplore Digital Library (ieeexplore.ieee.org/Xplore /home.jsp). These searching tools, however, do not always retrieve useful search results. First, these web search users might not know the exact keywords to formulate their search queries, which often yield undesirable retrieved articles that are non-relevant to their information needs. Second, even if retrieved results contain relevant articles, users are required to scan through the retrieved articles one by one to determine which ones are closely related to their information needs, which is a time-consuming and tedious process.

A solution to the web search problem, which evolves due to the explosive growth and variety of information on the Web that frequently lead users to make poor decisions, is to develop recommendation systems that are designed to suggest for items to be of use to users. The primary design goal of a recommender is to offer ranked lists of items by predicting the most suitable products or services based on the users' preferences and constraints. A good recommendation system for research papers should offer users relevant sets of scholarly articles that meet their information needs. In this paper, we propose an innovative recommender for scholarly articles using the Recurrent Neural Network (RNN) and BM25 probabilistic models to make suggestions. These models are not affected by users' ratings nor new users/items, which are the constraints applied to the content-based and CF approaches.

RNN is a network that can contain multiple feedback connections so that the activations flow round in a loop. Contrary to feed-forward networks, RNN is sensitive and operates not only on an input space but also on an internal state space. Given a user profile, which consists of a number of research papers that a user $u$ has recently published or is interested, and a *scholarly article* $A$ which $u$ would like to find other closely related references, we first apply RNN to determine the topic (or category) to which $A$ belongs so that we can minimize the search space on reference papers related to $A$. Hereafter, we apply the BM25 probabilistic model to determine the most relevant references to $A$. BM25 extends the scoring function for the binary independent probabilistic model to include document and query term weights, which is effective and accurate in rankings.

## 2 Related Work

Li, S. et al. (2018) consider the contents and authorship information of conference papers to suggest papers in conference proceedings to conference attendees. Since the content- and authorship-based approach presented in (Li, S. et al., 2018) restricts to conference papers, it cannot be applied to recommend scholarship articles in general. Lee, Y. et al. (2016), however, combine the content-based and graph-based approaches for research paper recommendation. The content-based approach computes the content similarity among different papers, whereas the graph-based approach captures the citation relationships among papers. A constraint imposed on this hybrid approach is that the graphical representation of the citation relationships is dynamic and must be instantly updated to reflect its accuracy, which is costly.

Asabere, N. et al. (2014) rely on social learning and networking in the design of their folksonomy-based paper recommendation algorithm for conference participants. The proposed social network, which must be established in advance prior to recommending research papers to conference participants with similar preferences at the same conference, requires extra time and effort to set up. Xue, H. et al. (2014), who develop SocialScholar, an online scholar platform, also rely on social relationship to make personalized academic paper recommendations. Since SocialScholar is designed exclusively for computer science researchers, it restricts the domains of research papers that it can recommend.

Hassan, H. (2017) applies deep neural networks in extracting the semantic representation of the scholarly papers' titles and abstracts for paper recommendation, which is close to the design of our scholarly article recommender which applies the recurrent neural network (RNN) and probabilistic models. Hassan, H. (2017), however, develops his recommender straightly based on a deep learning model, whereas we apply RNN simply for classification and use the probabilistic model hereafter for analyzing the similarity among different research papers belonged to the same category, which is seamless and yields optimal solutions.

## 3 Our Proposed Recommendation System

Our recommender considers research papers extracted from digital libraries and provided by its users. A user offers a sample scholarly article $SA$ based on which our recommender suggests similar research papers. Along with that, the user also archives articles of interest in a profile. The first phase of our recommender is to *categorize* the abstract of $SA$ and the second phase is to compute the similarity between $SA$ and the abstracts in the profile against the abstracts collected from digital libraries to make suggestions.

### 3.1 The Recurrent Neural Network (RNN) Model

We employ a recurrent neural network (RNN) as our classifier for categorizing scholarly articles which are provided by users for reference searching or extracted from the Web into their respective category, since RNNs have been proven to produce robust models for text classification A RNN is similar to other deep neural networks (DNNs) in that they are both trained (optimized) by the backpropagation of error and are comprised of a series of layers:

- An *input* layer, as a vector or matrix representation of the data (features) to be modeled.
- A few or many *hidden*, or *latent*, layers of activation nodes, sometimes referred to as "neurons". Each of the hidden layer is designed to map its previous layer to a higher-order (and often higher-dimensional) representation of the features which aims to be more useful in modeling the output than the original features.

- An *output* layer which produces the desired output for classification or regression tasks.

The output is produced by propagating numeric values forward $(a_i, y_i)$. The network is trained by backpropagating the *error*[1] from the output layer backwards. Unlike other network structures, a RNN takes into account the *ordering* of tokens within sequences, rather than simply accounting for the existence of certain values or combinations of values in that sequence. For example, the terms 'car' and 'repair' may appear in a sentence, but the sentiment of that sentence depends on whether or not they appear adjacent to each other and in that order. For complex textual tasks such as this example, RNNs tend to outperform bag-of-words models which are unable to capture important *recurrent patterns* that occur within sentences.

RNNs achieve the recurrent pattern matching through its *recurrent layer(s)*. A recurrent layer is one which contains a single recurrent unit through which each value of the input vector or matrix passes. The recurrent unit maintains a *state* which can be thought of as a "memory". As each value in the input iteratively passes through the unit at time step $t$, the unit updates its state $h_t$ based on a function of that input value $x_t$ and its own previous state $h_{t-1}$ as $h_t = f(h_{t-1}, x_t)$, where $f$ is any non-linear activation. Throughout training, the unit learns, i.e., optimizes, this state-updating function—it learns how much of its current state to keep or discard as it processes certain input values. Although the layer contains just a single unit, it can be visualized to have a number of units equal to its number of time steps, or iterations of processing sequential input values and previous states.

Recurrent layers are designed to "remember" the most important features in sequenced data no matter if the feature appears towards the beginning of the sequence or the end. In fact, one widely-used implementation of a recurrent unit is thus named "Long-Short Term Memory", or LSTM. The designed RNN accurately classifies our data set of scholarly articles solely based on their sequential text properties.

#### 3.1.1 Feature Representation

In order to utilize a RNN, we need to provide the network with sequential data as input and a corresponding ground-truth value as its target output. Each of the data entries has to first been transformed in order to be fed into the RNN. Attributes of article entries were manipulated as follows:

*Label*. The label consists of the category of a scholarly article, which are the top categories pre-defined by ACM DL. Since RNN cannot accept strings as an output target, each unique category string is assigned a unique integer value, which is transformed into a one-hot encoding to be used later as the network's prediction target.

*Features*. Features are extracted from the data set $S$ as the concatenation of the *title* and *abstract* of a scholarly article, which is called a *sentence* of an entry. These sentences are transformed into *sequences*, or ordered lists of tokens, i.e., unigrams and special characters such as punctuation marks. Each sequence is padded with an appropriate number of null tokens such that each sequence was of uniform length. We have considered only the first 72 tokens in each sentence when representing the features, since over 90% of sentences in $S$ contain 72 or fewer tokens. We have also considered only the 6,500 most commonly-occurring tokens in $S$.

*Text*. While extracting features, we have chosen not to remove stopwords, since we prefer not to lose any important semantic meaning, e.g., 'not', within term sequences nor punctuation, since many abstracts include mathematical symbols which especially correlate to certain categories.

#### 3.1.2 Network Structure

In this section, we explain the technical details of the RNN used for classifying scholarly articles.

**The Embedding Layer**. One of the design goals of our neural network is to capture relatedness between different English words (or tokens) with similar semantic meanings. For example, the phrase "he said" has a similar semantic meaning to the phrases "he says" or "she said". Our neural network begins with an embedding layer whose function is to learn a *word embedding* for the tokens in the vocabulary of our dataset. A word embedding maps tokens to respective $n$-dimensional real-valued vectors. Similarities in semantic meanings between different tokens ought to be captured in the word embedding by corresponding vectors which are also similar either by Euclidean distance, or by cosine similarity, or both. From our previous example, the $n$-dimensional vector for the word 'he' may be similar to the vector for 'she' by cosine similarity, or the vector for 'says' may be close in Euclidean space to the vector for 'said'.

The *embedding layer* contains 1,950,000 parameters, since there are 6,500 vectors, one for each token in the vocabulary, and each vector comes with 300 dimensions, and all of which could be trained. Due to the large amount

[1]An error is the relative divergence of the produced output from the ground truth.

of time it would take to properly train the word embedding from scratch, we have performed two different tasks: (i) we have loaded into the embedding layer as weights an uncased, 300-dimensional word embedding, GloVe, which has been pre-trained on documents on the Web, and (ii) we have decided to freeze, i.e., not train, the embedding layer at all. The pre-trained vectors from GloVe sufficiently capture semantic similarity between different tokens for our task and they are not required to be further optimized. Since the embedding layer was not trained, it simply served to transform the input tokens into a 300-dimensional space. Therefore, instead of the 72-element vector which we started with, the embedding layer outputs a 72 $\times$ 300 real-valued matrix.

**The Bi-directional GRU Layer**. Following the embedding layer in our network is one type of recurrent layer – a bi-directional GRU, or Gated Recurrent Unit, layer. A GRU is a current state-of-the-art recurrent unit which is able to ‘remember’ important patterns within sequences and ‘forget’ the unimportant ones.

This layer effectively ‘reads’ the text, or ‘learns’ higher-order properties within a sentence, based on certain ordered sequences of tokens. The number of trainable parameters in a single GRU layer is $3 \times (n^2 + n(m+1))$, where $n$ is the output dimension, or the number of time steps through which the input values pass, and $m$ is the input dimension. In our case, $n$ = 64, since we have chosen to pass each input through 64 time steps, and $m$ = 300 which is the dimensionality of each word vector in the embedding space. Since our layer is bi-directional, the number of trainable parameters is twice that of a single layer, i.e., $2 \times 3 \times (64^2 + 64 \times 301)$ = 140,160, the greatest number of trainable parameters in our network.

The recurrent layer outputs a 72 $\times$ 128 matrix, where 72 represents the number of tokens in a sequence, and 128 denotes the respective output values of the GRU after each of 64 time steps in 2 directions.

**The Global Max-Pooling Layer (1D)**. At this point in the network, it is necessary to reduce the matrix output from the GRU layer to a more manageable vector which we eventually use to classify the token sequence into one of the 13 categories (see Table 3). In order to reduce the dimensionality of the output, we pass the matrix through a *global max-pooling* layer. This layer simply returns as output the maximum value of each column in the matrix. Max-pooling is one of several pooling functions, besides sum- or average-pooling, used to reduce the dimensionality of its input. Since pooling is a computable function, not a learnable one, this layer cannot be optimized and contains no trainable parameters. The output of the max-pooling layer is a 128-dimensional vector.

**The Dropout Layer 1**. Our model includes at this point a dropout layer. Dropout, a common technique used in deep neural networks which helps to prevent a model from overfitting, occurs when the output of a percentage of nodes in a layer are suppressed. The nodes which are chosen to be dropped out are probabilistically determined at each pass of data through the network. Since dropout does not change the dimensions of the input, this layer in our network also outputs a 128-dimensional vector.

**The Dense Hidden Layer**. Our RNN model includes a dense, or fully-connected, layer. A *dense layer* is typical of nearly all neural networks and is used for discovering hidden, or latent, features from the previous layers. It transforms a vector $x$ with $N$ elements into a vector $y$ with $M$ inputs by multiplying $x$ by a $M \times N$ weight matrix $W$. Throughout training, weights are optimized via backpropagation.

**The Dropout Layer 2**. Before classification, our RNN model includes another dropout layer to again avoid overfitting to the training sequences.

**The Dense Output Layer**. At last, our RNN model includes a final dense layer which outputs thirteen distinct values, each value corresponding to the relative probability of the input belonging to one of the thirteen unique categories. Each instance is classified according to the category corresponding to the highest of the 13 output values.

## 3.2 The Probabilistic Model

Upon determining the category $C$ of a research paper $P$ provided by a user $u$ using our RNN, our recommender proceeds to recommend to $u$ scholarly articles relevant to $P$. To suggest relevant papers, we calculate the similarity between the abstracts of $P$ and the scholarly articles specified in $u$’s profile against abstracts in a collection of research papers that fall under the category $C$. There are various methods to accomplish the task. We have applied the BM25 algorithm.

BM25 (Robertson, S. and Zaragoza, H., 2009), a traditional probabilistic model, assumes that the relevance of a document (a candidate research paper in our case) to a query (a given scholarly article in our case) can be determined by aggregating individual contributions of the query terms. We apply BM25 as the probabilistic model of our recommender, since it has influenced the ranking algorithms of commercial search engines, including web search engines, and can be implemented efficiently. In this section, we first introduce the binary independence model, the probabilistic model based on which BM25 is developed.

Table 1: Contingency table in which $r_i$ is the number of relevant documents in a collection $N$ containing keyword $i$, $n_i$ is the number of documents in $N$ containing $i$, and $R$ is the number of relevant documents in $N$

| | Relevant | Non-Relevant | Total |
|---|---|---|---|
| $d_i = 1$ | $r_i$ | $n_i - r_i$ | $n_i$ |
| $d_i = 0$ | $R - r_i$ | $N - n_i - R + r_i$ | $N - n_i$ |
| Total | $R$ | $N - R$ | $N$ |

### 3.2.1 The Binary Independence Model

According to the Bayes Decision Rule, a document $D$ is *relevant* if $P(R|D) > P(NR|D)$, where $R$ ($NR$, respectively) denotes the relevance (non-relevance, respectively) set of documents, and $P(R|D)$ and $P(NR|D)$ are *conditional probabilities*, which are formally defined as follows based on the Bayes Rule.

$$P(R|D) = \frac{P(D|R)P(R)}{P(D)}, P(NR|D) = \frac{P(D|NR)P(NR)}{P(D)} \tag{1}$$

A document $D$ is classified as relevant if

$$\frac{P(D|R)}{P(D|NR)} > \frac{P(NR)}{P(R)} \tag{2}$$

where $\frac{P(D|R)}{P(D|NR)}$ is the *likelihood ratio* of $D$ being relevant, which can be derived from $P(R|D) > P(NR|D)$ and using Equation 1.

Assuming the word independence assumption in the Naïve Bayes model and using individual word, i.e., $d_i$, probabilities

$$P(D|R) = \Pi_{i=1}^{t} P(d_i|R) \tag{3}$$

where $d_i$ ($1 \leq i \leq t$) is a keyword in document $D$.

Further assume that $p_i$ ($s_i$, respectively) is the *probability of occurrence* of keyword $i$ in a relevant (non-relevant, respectively) set of documents. Using $p_i$ and $s_i$, the *likelihood ratio* as defined in Equation 2 can be computed as follows, where $d_i = 1$ ($d_i = 0$, respectively) denotes that $i$ is in (not in, respectively) $D$, and $1 - p_i$ ($1 - s_i$, respectively) is the probability of $i$ not in $R$ ($NR$, respectively).

$$\begin{aligned}\frac{P(D|R)}{P(D|NR)} &= \Pi_{i:d_i=1}(\frac{p_i}{s_i}) \times \Pi_{i:d_i=0}(\frac{1-p_i}{1-s_i}) = \Pi_{i:d_i=1}(\frac{p_i}{s_i}) \times (\Pi_{i:d_i=1}(\frac{1-s_i}{1-p_i}) \times \\ &\quad \Pi_{i:d_i=1}(\frac{1-p_i}{1-s_i})) \times \Pi_{i:d_i=0}(\frac{1-p_i}{1-s_i}) \\ &= \Pi_{i:d_i=1}(\frac{p_i \times (1-s_i)}{s_i \times (1-p_i)}) \times \Pi_i(\frac{1-p_i}{1-s_i}) \cong \Pi_{i:d_i=1}(\frac{p_i \times (1-s_i)}{s_i \times (1-p_i)}) \cong \sum_{i:d_i=1} \log(\frac{p_i \times (1-s_i)}{s_i \times (1-p_i)})\end{aligned} \tag{4}$$

Using the *keyword incidence contingency table* (contingency table for short) as shown in Table 1, $p_i$ and $s_i$ in Equation 4 can be defined as

$$p_i = \frac{r_i + 0.5}{R+1}, s_i = \frac{n_i - r_i + 0.5}{N - R + 1} \tag{5}$$

Note that 0.5 is added to the numerator (and 1 to the denominator) of $p_i$ and $s_i$ in Equation 5 to avoid the computation of *log* 0, e.g., when $r_i$, $n_i$, $R$, and $N$ are zero. Substituting $p_i$ and $s_i$ in Equation 5 into $\sum_{i:d_i=1} \log(\frac{p_i \times (1-s_i)}{s_i \times (1-p_i)})$ in Equation 4 and further simplify it

$$\begin{aligned}\sum_{i:d_i=1} \log(\frac{p_i \times (1-s_i)}{s_i \times (1-p_i)}) &= \sum_{i:d_i=1} \log\frac{(\frac{r_i+0.5}{R+1}) \times (1 - \frac{n_i-r_i+0.5}{N-R+1})}{(\frac{n_i-r_i+0.5}{N-R+1}) \times (1 - \frac{r_i+0.5}{R+1})} = \sum_{i:d_i=1} \log\frac{(\frac{r_i+0.5}{R+1}) \times (\frac{N-R+1-n_i+r_i-0.5}{N-R+1})}{(\frac{n_i-r_i+0.5}{N-R+1}) \times (\frac{R+1-r_i-0.5}{R+1})} \\ &= \sum_{i:d_i=1} \log\frac{(r_i+0.5) \times (N-R-n_i+r_i+0.5)}{(n_i-r_i+0.5) \times (R-r_i+0.5)} = \sum_{i:d_i=1} \log(\frac{r_i+0.5}{R-r_i+0.5} \times \frac{N-R-n_i+r_i+0.5}{n_i-r_i+0.5}) \\ &\approx \sum_{i:d_i=q_i=1} log\frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)}\end{aligned} \tag{6}$$

### 3.2.2 The BM25 Model

BM25 extends the scoring function for the *binary independence model* to include document and query keyword weights. Adding the keyword weights in a document, i.e., a candidate research paper $D$, and a query $Q$, i.e., a user-provided scholarly article in our case, into Equation 6 yields the BM25 ranking formula, which is based on probabilistic arguments and experimental evaluation.

$$BM25(Q, D) = \sum_{i \in Q} log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \times \frac{(k_1 + 1) \times f_i}{K + f_i} \times \frac{(k_2 + 1) \times q_{f_i}}{k_2 + q_{f_i}} \quad (7)$$

where the notation used in Equation 7 are given in Table 2 which are applied to our recommendation system.

Table 2: Notation used in Equation 7

| | |
|---|---|
| $N$ | The number of research papers in a collection of scholarly articles (articles for short) $C$ |
| $n_i$ | The number of articles in $C$ that contains keyword $i$ |
| $R$ | The number of articles in a user's profile $P$ |
| $r_i$ | The number of articles in $P$ that contains $i$ |
| $f_i$ | The frequency of $i$ in a candidate article $CA$ in $C$ being considered for recommendation |
| $q_{f_i}$ | The frequency of $i$ in a user's given article $UA$ for which references are to be recommended |
| $k_1$ | A constant value which plays the role of term-frequency along with $f_i$ |
| $k_2$ | A constant value which plays the similar role as $k_1$, but is applied to keyword weight $q_{f_i}$ |
| $K$ | $K = k_1 \times ((1 - b) + b \times \frac{dl}{avdl})$, where $dl$ is the length of $CA$ and $avdl$ is the average length of articles in $C$ regulated by $b$ ($0 \leq b \leq 1$) |
| | $k_1$, $k_2$, and $b$ are empirically set as 1.2, 100, and 0.75, respectively |

Note that if a user has yet to accumulate any publication in a profile, i.e., when the *cold-start problem* is encountered, both $R$ and $r_i$ in Equation 7 are set to zero, and BM25 can still be applied to rank relevant research papers with respect to a user-given scholarly article, which is the *only* required input. Prior to applying Equation 7, stopwords in all the involved research papers are *removed* and remaining keywords are *stemmed* using the Porter stemming algorithm (Croft, W. et al., 2010).

## 4 Experimental Results

In this section, we verify the merit and usefulness of our proposed scholarly article recommendation system.

### 4.1 Data Set

To conduct an empirical study on the performance of our recommendation system, we rely on the publication records archived in ACM DL and IEEE Xplore. ACM provides an online comprehensive database for computing literature, which contains the ACM Full-Text Collection and ACM Guide to Computing Literature. The former collects all ACM publications, which include journals, conference proceedings, technical magazines, newsletters and books, whereas the letter is the bibliographic database for computing literature. ACM offers the users options to search by using keywords. In addition, ACM has its own classification system (CCS): the primary CCS and its different sub-classifications. The thirteen primary CCS and the corresponding number of extracted scholarly articles used for training and testing our classifier are shown in Table 3. The total number of extracted scholarly articles (titles + abstracts) in all 13 primary CCSs used for classification and recommendation is 1.2 million with the publication year back to 1952, and the size of the data is 247 MB. IEEE Xplore, another scholarly article database, includes 7,262 professional topics in Computer Science, Electrical Engineering & Electronics, and other fields. It consists of a collections of conference proceedings, early access articles, journals and magazines, standards, books, and courses. In addition, it contains options to search by metadata only or full-text and metadata. It also provides users the option to enter the keywords for searching. We collected data in different research areas through IEEE Xplore with the publication year back to 1905 using the keyword search option. There were 3.0 million scholarly articles extracted from 18 publishers, which include IEEE, MITP, IBM, Wiley, etc., which yield 1.8 GB of data.

Table 3: Distribution of article categories using the ACM DL dataset

| Category | Articles | % | Category | Articles | % | Category | Articles | % |
|---|---|---|---|---|---|---|---|---|
| Applied Comp. | 1,753 | 7.44 | Comp. Sys. Org. | 1,944 | 8.25 | Comp. Method. | 1,770 | 7.51 |
| General & Ref. | 1,161 | 4.93 | Hardware | 1,931 | 8.20 | Human Comp. | 2,422 | 10.3 |
| Info. Sys. | 2,139 | 9.08 | Math. of Comp. | 1,326 | 5.63 | Networks | 2,058 | 8.74 |
| Sec. & Privacy | 2,144 | 9.10 | Social & Prof. | 1,582 | 6.72 | Soft. Engg. | 1,534 | 6.51 |
| Theory of Comp. | 1,793 | 7.61 | **Total** | **23,557** | **100** | | | |

## 4.2 Accuracy of Our Classifier

Using a 80/10/10% training/validation/test split of the data as shown in Table 3, we achieved 75% classification accuracy on test data. The accuracy could not be higher likely because of the high amounts of overlap between distinct keywords in titles and abstracts of articles with different categories, such as "Mathematics of Computing" and "Theory of computation". With 75% accuracy, we still successfully classify 3 out of 4 articles, which is way above the baseline "best-guesser" classifier. Other bag-of-words modeling techniques with which we have experimented, i.e., logistic regression, SVM, and Multinomial Naïve Bayes (Croft, W. et al., 2010), showed lower results.

## 4.3 Performance of Our Probabilistic Model

In evaluating the performance of our BM25 probabilistic model in making relevant research paper recommendations, we randomly chose 1,000 scholarly articles from the ACM DL and IEEE Xplore dataset, using them individually as a user-provided query article and creating the corresponding user profile which includes the recent 20 publications of the author(s) of the article. The papers *referenced* in each one of the 1,000 scholarly articles serve as the *ground truth* of the corresponding test case, which dictate whether a research paper recommender successfully predicts the references as recommended papers. We compare the performance of our scholarly article recommender with three recently-developed research paper recommenders in (Haruna, K. et al., 2017; Lee, J. et al., 2015; Lee, Y. et al., 2016) below.

## 4.4 Comparing Our Recommendation System with Others

In this section, we compare the performance of our recommender with various existing research paper recommendation systems. These existing recommenders are based on different design methodologies and recently published in the literature. We have implemented the research paper recommendation approaches discussed below and compared their performance with our recommender which demonstrates the merit and novelty of the classification and probabilistic models of our research paper recommender, using the data set presented in Section 4.1.

- $k$**NN**. Lee, J. et al. (2015) adopt the $k$-Nearest Neighbors clustering algorithm to determine the top-$k$ closely related academic papers for paper recommendation. Lee et al.'s model treats each paper $P$ as a bag of words, with words extracted from its title, keyword, and abstract, and represents $P$ as a vector in the vector space.
- **CB**. Haruna, K. et al. (2017) develop a collaborative approach based on paper-citation relations to recommend research papers. The collaborative approach presented by Haruna et al. identifies latent relationships that exist among different research papers based on their paper-citation relations.
- **CGA**. As discussed in Section 2, Lee, Y. et al. (2016) introduce a hybrid approach that combines a content-based and a graph-based approach for research paper recommendation. The content-based approach measures the content similarity between a research paper and papers downloaded by a user. For the graph-based approach, the authors model citation relationships among different papers as an undirected graph in which a node represents a paper and a link denotes the citation relationship between two papers.

Figure 1 shows the performance evaluation of the three research paper recommendation systems—$k$NN, CB, and CGA–along with our recommender, denoted C+P, based on the Recall, Mean Average Precision (MAP), Mean Reciprocal Rank (MRR), and Normalized Discounted Cumulative Gain (nDCG) (Croft, W. et al., 2010), which are well-known performance measures in information retrieval. The results were based on the 1,000 test cases processed by each of the recommendation systems, and we considered the top-15 recommendations made by each recommender for each test case. Each test case includes a scholarly article randomly chosen from the collection

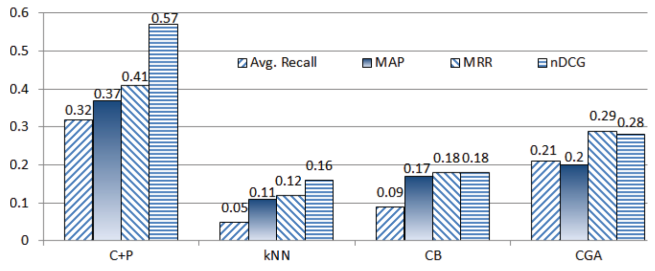

Figure 1: Performance evaluation of various research paper recommendation systems

of research papers extracted from the dataset as discussed in Section 4.1. The performance evaluation clearly indicates that our proposed system outperforms the other three recommenders, and the results are statistically significant ($p < 0.001$).

## 5 Conclusions

Search engines and digital libraries are helpful in retrieving relevant set of papers to a particular research problem; however, users often have trouble formulating their queries with appropriate keywords. In solving this problem, we have designed a two-stage recommendation system which first determines to which category a user-provided research paper belong using a recurrent neural network model. Hereafter, scholarly articles in the same category are recommended using the BM25 probabilistic model based on the content similarity. Experimental results have verified the merit of the proposed system, which outperforms recently-developed research paper recommenders.

## References

Asabere, N. et al. (2014), 'Scholarly Paper Recommendation Based on Social Awareness and Folksonomy', *Parallel, Emergent and Distributed Systems* **30**(3), 1–22.

Cho, K. et al. (2014), On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, *in* 'Proceedings of the 8th Workshop on Syntax, Semantics and Structure in Statistical Translation', pp. 103–111.

Croft, W. et al. (2010), *Search Engines: Information Retrieval in Practice*, Addison Wesley.

Haruna, K. et al. (2017), 'A Collaborative Approach for Research Paper Recommender System', *PLoS ONE* **12**.

Hassan, H. (2017), Personalized Research Paper Recommendation Using Deep Learning, *in* 'Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization', pp. 327–330.

Lee, J. et al. (2015), Personalized Academic Paper Recommendation System, *in* 'Proceedings of SRS'.

Lee, Y. et al. (2016), Recommendation of Research Papers in DBpia: A Hybrid Approach Exploiting Content and Collaborative Data, *in 'Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics', pp. 2966–2971.*

*Li, S. et al. (2018), 'Conference Paper Recommendation for Academic Conferences',* IEEE Access **6***, pp. 17153–17164.*

*Robertson, S. and Zaragoza, H. (2009), 'The Probabilistic Relevance Framework: BM25 and Beyond',* Foundations and Trends in Information Retrieval **3***(4), pp. 333–389.*

*Xue, H. et al. (2014), Personalized Paper Recommendation in Online Social Scholar System,* in 'Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*', pp. 612–619.*