Personalized Book Recommendation Based on a Deep Learning Model and Metadata

Yiu-Kai Ng and Urim Jung

Computer Science Department, Brigham Young University, Provo, Utah 84602, USA
ng@compsci.byu.edu, urimjung1@gmail.com

Abstract. Reading books is one of the widely-adopted methods to obtain knowledge. Through reading books, one can obtain life-long knowledge and maintain them. Additionally, if multiple sources of information can be obtained from various books, then obtaining relevant books is desirable. This can be done by book recommendation. There are, however, a number of challenges in designing a book recommender system. One of the challenges is to suggest relevant books to users without accessing their actual content. Unlike websites or blogs, where the crawler can simply scrape the content and index the websites for web search, book contents cannot be accessed easily due to copyright laws. Because of this problem, we have considered using data such as book records, which contains various metadata of a book, including book description and headings. In this paper, we propose an elegant and simple solution to the book recommendation problem using a deep learning model and various metadata that can infer the content and the quality of books without utilizing the actual content. Metadata, which include Library Congress Subject Heading (LCSH), book description, user ratings and reviews, which are widely available on the Internet. Using these metadata are relatively simple compared to approaches adopted by existing book recommender systems, yet they provide essential and useful information of books.

Keywords: book recommendation · deep learning · metadata

1 Introduction

Reading books enhances our understanding on the content covered in a book and offer us an opportunity to learn new knowledge. According to [8], many of the college students believe reading book is directly linked to academic success in college. For people who are not in college, reading books helps them learn throughout their lives.

Instead of accessing the content of a book using its hard-copy archived in a library or made available in a book store, electronic copies became available online through online services such as Google Books or Amazon. In addition, book reviews and ratings can be downloaded by customers and users so that they can filter sub-standard books and make the best choice. Several book recommendation systems have been developed to recommend relevant books to users [7] based on machine learning algorithms or other techniques such as data mining. However, these algorithms require accessing the actual book content which is not widely available due to the copyright law. Instead, we propose an elegant and effective solution to the problem by using metadata associated with

books. Metadata are useful, since they offer useful information of the corresponding books. We consider book descriptions, LCSH, user ratings, and reviews to rank books.

Our book recommender is designed for solving the *information overload* problem while minimizing the *time* and *efforts* imposed on readers in discovering unknown, but suitable, books for pleasure reading or knowledge acquisition. Our recommender first identifies a set of *candidate books*, among the ones archived at a website, with topics related to a number of books preferred by the user U. Our recommender is a self-reliant recommender which, unlike others, does not rely on personal tags nor access logs to make book recommendation. It is unique, since it explicitly determines categories of books that match the one preferred by users using a deep learning algorithm, besides considering the subject headings, user ratings, content descriptions, and sentiment on books that are available online.

Our proposed solution provides book stores and libraries diverse and effective book recommendation. In addition, the users can have a satisfying experience with the book recommendation system in terms of saving time and efforts in searching for relevant and interested books to read. Furthermore, our book recommender system is significantly differed from existing approaches, since we do not consider any data mining technique. By simply aggregating the information provided by metadata of books, we effectively recommend books that are relevant to the user's information needs.

2 Related Work

A number of book recommenders [6, 15] have been proposed in the past. Amazon's recommender [6] suggests books based on the purchase patterns of its users. Yang et al. [15] analyze users' access logs to infer their preferences and apply the collaborative filtering (CF) strategy, along with a ranking method, to make book suggestions. Givon and Lavrenko [4] combine the CF strategy and social tags to capture the content of books for recommendation. Similar to the recommenders in [4, 15], the book recommender in [12] adopts the standard user-based CF framework and incorporates semantic knowledge in the form of a domain ontology to determine the users' topics of interest. The recommenders in [4, 12, 15] overcome the problem that arises due to the lack of initial information to perform the recommendation task, i.e., the cold-start problem. However, the authors of [4, 15] rely on user access logs and social tags, respectively to recommend books, which may not be publicly available and are not required by our recommender. Furthermore, the recommender in [12] is based on the existence of a book ontology, which can be labor-intensive and time-consuming to construct [2].

Zhu and Wang [17] adopt relational data mining algorithm for recommending books. They apply the Apriori data mining algorithm to eliminate mismatched book records and effectively perform data mining using optimization. This approach reduces the amount of book data to be considered. Mooney and Roy [7] apply the contend-based book recommendation approach to obtain the descriptions of books and develop a machine learning algorithm to categorize the text. After categorizing the text, they utilize user profile and use the Bayesian learning algorithm to find the appropriate book for the specific user. Sohail et al. [14] solve the book recommendation problem by constructing an opinion-mining algorithm which relies on the reviews written by users to extract the users' opinions on books for making recommendation. All of these approaches

are significantly differed from ours, since the latter simply relies on topic analysis and matadata of books in making book recommendation to its users.

3 Our Book Recommender System

We first utilize a deep neural network model to classify a book B given by a user U who also provides a number of preferred books in a profile. Based on the category of B, we filter books in a collection that are in the same category as B, called *candidate books* CB. Hereafter, we consider different *features* (presented in Sections 3.2 to 3.5) of books in CB to rank them (in Section 3.6) accordingly.

3.1 The Recurrent Neural Network (RNN) Model

We employ a recurrent neural network (RNN) as our classifier, since RNNs produce robust models for classification. Similar to other deep neural networks, RNNs are both trained (optimized) by the backpropagation of error and comprised of a series of layers.

- An *input* layer is a vector or matrix representation of the data to be modeled.
- A few *hidden*, or *latent*, layers of activation nodes, sometimes referred to as "neurons", are included. 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 produces the desired output for classification or regression tasks.

The output is produced by propagating numeric values forward.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.

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 books solely based on their sequential text properties.

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

Table 1. Dimensions and number of parameters of layers in the RNN

Layer	Output Dimensions	Total Parameters	Trainable Parameters
Input	72	0	0
Embedding	72×300	1,950,000	0
Bi-directional GRU	72×128	140,160	140,160
Global Max Pooling (1D)	128	0	0
Dropout 1	128	0	0
Dense Hidden	64	8,256	8,256
Dropout 2	64	0	0
Dense Output	31	845	845
Total		**2,099,279**	**149,261**

Feature Representation 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 data entry has to first be transformed in order to be fed into the RNN. Attributes of book entries were manipulated as follows:

Label. The label consists of the category of a book, each of which is the top 31 categories pre-defined by Thriftbooks[2]. 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[3] to be used later as the network's prediction target.

Features. Features are extracted from the data set S as the *brief description* of a book, which is called a *sentence* of an entry, and is accessible from the book-affiliated websites such as Amazon[4]. Words in a brief description 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 considered 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, e.g., '|', which especially correlate to certain categories. We did, however, convert all of the text in a sentence to lowercase because the particular *word embedding* which we used did not contain cased characters.

Network Structure We first discuss our RNN used for classifying book categories. Table 1 summaries different layers, their dimensions, and their parameters in our RNN.

The Embedding Layer. A design goal 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

[2] https://www.thriftbooks.com/sitemap/

[3] A one-hot encoding of an integer value i among n unique values is a binarized representation of that integer as an n-dimensional vector of all zeros except the i^{th} element, which is a one.

[4] www.amazon.com

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. For example, the n-dimensional vector for '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 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 72×300 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 72×128 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 31 categories. 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 31 distinct values, each value corresponding to the relative probability of the input belonging to one of the 31 unique categories. Each instance is classified according to the category corresponding to the highest of the 31 output values.

3.2 LCSH

The Library of Congress provides a unique tag, known as Library of Congress Subject Heading, denoted LCSH, for each book prior to its publication. Unlike social media, where users can create a tag to a post suitable to their taste, Library of Congress maintains standardized tags, which come from a controlled vocabulary, from where a subject heading is constructed [3]. Based on this fact, we can effectively measure the closeness of two books in terms of their subject areas by applying our *word correlation factor (WCF)* to compute the *similarity* between their corresponding tags, which consists of a sequence of keywords, in LCSH.

The word-correlation factor between keywords i and j, denoted $Sim(i, j)$, is precomputed using 880,000 documents in the Wikipedia collection (wikipedia.org/)[5] based on their *frequency of co-occurrence* and *relative distance* in each Wikipedia document.

$$Sim(i,j) = \frac{\sum_{w_i \in V(i)} \sum_{w_j \in V(j)} \frac{1}{d(w_i,w_j)+1}}{|V(i)| \times |V(j)|} \tag{1}$$

where $d(w_i, w_j)$ is the *distance* between words w_i and w_j in any Wikipedia document D, $V(i)$ ($V(j)$, respectively) is the set of *stem* variations of i (j, respectively) in D, and $|V(i)| \times |V(j)|$ is the *normalization* factor.

Although WordNet[6] provides synonyms, hypernyms, holonyms, and antonyms for a given word, there is no partial degree of similarity measures (closeness), i.e., weights, assigned to any pair of words. For this reason, word-correlation factors are more sophisticated in measuring word similarity than word pairs in WordNet.

The word correlation factor of keywords w_1 and w_2 is assigned a value between 0 and 1, such that '1' denotes an *exact* match and '0' denotes *total dissimilarity* between

[5] Words within the Wikipedia documents were *stemmed* and *stopwords* were removed.
[6] wordnet.princeton.edu/

w_1 and w_2. Note that even for highly similar, non-identical words, they are on the order of 5×10^{-4} or less. For example, the degree of similarity between "tire" and "wheel" is 3.1×10^{-6}, which can be treated as 0.00031% *similar* and 99.99% *dissimilar*. As we prefer to ascertain how likely the words are on a scale of 0% to 100% in sharing the same semantic meaning, we further *scale* the word-correlation factors. Since correlation factors of non-identical word pairs are less than 5×10^{-4} and word pairs with correlation factors below 1×10^{-7} do not carry much weight in the similarity measure, we use a logarithmic scale, i.e., $ScaledSim$, which assigns words w_1 and w_2 the similarity value V of 1.0 if they are *identical*, 0 if $V < 1 \times 10^{-7}$, and a value between 0 and 1 if $1 \times 10^{-7} \leq V \leq 5 \times 10^{-4}$, which is formally defined as

$$ScaledSim(w_1, w_2) = \begin{cases} 1 & \text{if } w_1 = w_2 \\ Max(0, 1 - \frac{ln(\frac{5\times10^{-4}}{Sim(w_1,w_2)})}{ln(\frac{5\times10^{-4}}{1\times10^{-7}})}) & \text{Otherwise} \end{cases} \quad (2)$$

where $Sim(w_1, w_2)$ is the *word-correlation factor* of w_1 and w_2 defined in Equation 1.

We computes the *degree of similarity* of any two LCSHs L and C using

$$LimSim(L, C) = \frac{\sum_{i=1}^{m} Min(1, \sum_{j=1}^{n} ScaledSim(i, j))}{m} \quad (3)$$

where m and n denote the number of keywords in the LCSHs L and C, respectively, i and j are the keywords in L and C, respectively, and $ScaledSim(i, j)$ is as defined in Equation 2.

Using the $LimSim$ function, instead of simply adding the $ScaledSim$ value of each keyword in L with respect to each keyword in C, we *restrict* the highest possible sentence-similarity value between L and C to 1, which is the value for *exact* matches. By imposing this constraint, we ensure that if L contains a keyword K that is (i) an *exact* match of a keyword in C, and (ii) similar to (some of) the other words in C, then the degree of similarity of L with respect to C cannot be significantly impacted/affected by K to ensure a balanced similarity measure of L with respect to C.

3.3 User Ratings

Making recommendations for users based on their past behaviors is crucial and is in essence learning hidden factors which drive users' decision-making process, and rating prediction is such an approach. In this paper, we apply rating prediction for making book recommendations. The higher a predicted rating on a book B for user U using the ratings of books previously viewed by U is, the more likely B is appealed to U. To reduce the problem of finding a user's decision latent-factor model to finding the set of users who make similar decisions, matrix factorization (MF) is a sophisticated rating prediction approach to use such a decision latent-factor model.

To predict unknown ratings on books, a recommender is given a $m \times n$ sparse matrix of known user-book ratings. Singular value decomposition (SVD) [5] can be employed to deduce each user and book latent-factor vectors by factoring out the user and book latent-factor matrices from the user-book rating matrix. Traditional SVD, however, requires the given matrix to be dense. Assuming that all the missing entries are either

zero or averages of other entries and applying classical SVD to fill the matrix is going to result in intolerable inaccuracy in the predictions. To handle the sparseness problem, we apply the Funk SVD Learning Algorithm, which is the current state-of-the-art SVD algorithm popularized by Simon Funk in solving the Netflix 100M rating problem. The basic idea is to employ techniques of gradient descent to iterate through the set of known ratings to minimize the squared error of the predicted rating. This iterative process involves the following steps: (i) before the training starts, a predicted rating was guessed to be the average book rating plus the user offset, (ii) for each given user-book rating, the prediction in the previous iteration is updated in the opposite direction of the gradient, and (iii) step (ii) was repeated until prediction error converges to zero.

3.4 User Reviews

In addition to user ratings, we consider common user reviews on books, which can be used for measuring the overall sentiment [12] towards books, to determine the most desirable books to be recommended. Quite often a user writes a user review on a book without providing a rating, and vice versa. Given that user ratings offer only an absolute value without any additional information on a book, while the user reviews contribute additional sentiments to the book. For example, assume that a user gives the same ratings on two different books. Based on the ratings we have to assume that the two books are equally good or equally bad. However, suppose the user makes the comment "Decently written" on the first book, and "Decently written, but I liked the concept" on the second book. With the additional comments, we can claim that the second book is more desirable than the first, since positive sentiment is made towards the second book. For this reason, users' reviews can be used as a supplement to the users' ratings to make suitable book recommendations to users. Sentiment book reviews can easily be found through multiple book websites.

In order to apply users' book reviews in our recommender system, we first determine the polarity of each word w in each review r of a book BK such that w is positive (negative, respectively) if its positive (negative, respectively) SentiWordNet[7] (sentiwordnet.isti.cnr.it) score is higher than its negative (positive, respectively) counterpart. We calculate the overall sentiment score of the reviews made on BK, denoted $StiS(BK)$, by subtracting the sum of its negative words' scores from the sum of its positive words' scores, which reflects the overall sentiment orientation, i.e., positive, negative, or neutral, of the reviews on BK. As the length of the comments on BK can significantly affect the overall sentiment on BK, i.e., the longer each review is, the more sentiment words are in the review, and thus the higher (lower, respectively) its sentiment score is, we normalize the sentiment score of BK by dividing the sum of the SentiWordNet scores of the words in the reviews with the number of sentiment words in the reviews on BK, which yields

$$StiS(BK) = \sum_{i=1}^{n} \frac{\sum_{j=1}^{m} SentiWordNet(Word_{i,j})}{|Rev_i|} \tag{4}$$

[7] SentiWordNet, a lexical resource for opinion mining, assigns to each word in WordNet three sentiment scores: positivity, objectivity (i.e., neutral), and negativity. A SentiWordNet score is bounded between -1 and 1, inclusively.

Table 2. TF-IDF weighting scheme used in the enhanced cosine similarity measure in Equation 6

Condition	**Weight Assignment**
$B_i \in B$ and $P_{B_i} \in P_B$	$V_{B_i} = tf_{B_i,B} \times idf_{B_i}$ and $V_{P_{B_i}} = tf_{P_{B_i},P_B} \times idf_{P_{B_i}}$
$B_i \in B$ and $P_{B_i} \notin P_B$	$V_{B_i} = tf_{B_i,B} \times idf_{B_i}$ and $V_{P_{B_i}} = \frac{\sum_{c \in HS_{B_i}} tf_{c,P_B} \times idf_c}{\lvert HS_{B_i} \rvert}$
$B_i \notin B$ and $P_{B_i} \in P_B$	$V_{B_i} = \frac{\sum_{c \in HS_{P_{B_i}}} tf_{c,B} \times idf_c}{\lvert HS_{P_{B_i}} \rvert}$ and $V_{P_{B_i}} = tf_{P_{B_i},P_B} \times idf_{P_{B_i}}$

where n is the number of reviews on BK, m is the number of words in the k^{th} ($1 \leq k \leq n$) review on BK, $Word_{i,j}$ ($1 \leq i \leq n$, $1 \leq j \leq m$) is the j^{th} word in the i^{th} review, and $|Rev_i|$ is the number of words in the i^{th} review of BK.

As the highest (lowest, respectively) SentiWordNet score of any word is 1 (-1, respectively), $LS < StiS(BK) \leq HS$, where $-0.9 \leq HS \leq 1$, $-1 \leq LS \leq 0.9$, and $HS - LS = 0.1$. $StiS(BK)$ is further scaled so that its value, denoted $StiS_{Scaled}(BK)$, is bounded between 0 and 1, since a *negative* $StiS(BK)$ value can be returned if the overall sentiment of BK leans towards the negative region. Equation 5 assigns the normalized value to $StiS(BK)$.

$$StiS_{Scaled}(BK) = CL(StiS(BK)) + \frac{0.9 - FL(StiS(BK))}{2}$$
$$CL(StiS(BK)) = \frac{\lceil StiS(BK) \times 10 \rceil}{10}, FL(StiS(BK)) = \frac{\lfloor StiS(BK) \times 10 \rfloor}{10} \quad (5)$$

3.5 Content Similarity Measure

We depend on the user profile P of a user U[8], which is a set of books preferred by U, to infer U's interests/preferences. To determine the degree to which the content of a candidate book B in appeals to U, we compute the *content similarity* between B and each book P_B in P, denoted $CSim(B, P)$ as defined in Equation 6, using a "bag-of-words" representation on the *brief descriptions* of B and P_B obtained from book-affiliated websites, such as Amazon[9]. To compute $CSim(B, P)$, we employ an enhanced version of the *cosine similarity measure*, which relaxes the exact-matching constraint imposed by the cosine measure and explores words in the description of B that are *analogous* to, besides the *same* as, words in the description of P_B.

$$CSim(B, P) = \max_{P_B \in P} \frac{\sum_{i=1}^{n} VB_i \times VP_{B_i}}{\sqrt{\sum_{i=1}^{n} VB_i^2} \times \sqrt{\sum_{i=1}^{n} VP_{B_i}^2}} \quad (6)$$

where B and P_B are represented as n-dimensional vectors $VB = <VB_1, \ldots, VB_n>$ and $VP_B = <VP_{B_1}, \ldots, VP_{B_n}>$, respectively, n is the number of distinct words in the descriptions of B and P_B, and VB_i (VP_{B_i}, respectively), which is the $weight$ assigned to word B_i (P_{B_i}, respectively), is calculated as shown in the equations in Table 2.

[8] If a user does not offer a user profile P, then we simply treat the book provided by the user as the only book in P.

[9] www.amazon.com

HS_w in Table 2 is the set of words that are *highly similar* to, but not the $same$ as, a given word w in the description of a book Bk, which is either B or P_B, $|HS_w|$ is the size of HS_w, $tf_{w,Bk} = \frac{f_{w,Bk}}{\sum_{w \in Bk} f_{w,Bk}}$ is the normalized *term frequency* of w in Bk, and $idf_w = log\frac{N}{n_w}$ is the *inverse document frequency* for w in the collection of books N archived at a social bookmarking site, where n_w is the number of books in N that include w in their descriptions. Relying on the tf-idf weighting scheme, we prioritize discriminating words that capture the content of its respective book.

The *max* function in Equation 6 emulates the "most pleasure" strategy (commonly applied in game theory and group profiling [10]). Applying this strategy, we select the $highest$ possible score among the ones computed for each P_B in P and B. The $larger$ the number of exact-matched or highly-similar words in the descriptions of both B and P_B is, the *more likely* B is a relevant recommendation for U, and guarantees that B is highly similar to at least one of the books of interest to U. We adopt the cosine measure (in Equation 6), which has been effectively applied to determine the degree of resemblance between any two items in content-based recommenders.

3.6 Combining Ratings

Based on computed scores of *LCSH*, *user ratings*, *user reviews*, and *content similarity measure* for each candidate book B, we apply the *Borda Count voting scheme* [1] to determine the *ranking* score for B. The Borda Count voting scheme is a positional-scoring procedure such that given k (≥ 1) candidates, each voter casts a vote for each candidate according to his/her preference. A candidate that is given a first-place vote receives k-1 points, a second-ranked candidate k-2 points, and so on up till the last candidate, who is awarded no points. Hereafter, the points assigned to each candidate across all the voters are added up and the candidate with the $most$ points $wins$.

We employ the Borda Count strategy to generate a single ranking score for B, denoted $Borda(B)$, that regards all the features scores of B as equally important in determining the degree to which a user is interested in B. Using Equation 7, we assign (i) k = $|CandBks|$, which is the number of candidate books selected for a user U, and (ii) C = 4, which is the number of voters, i.e., the four ranked lists of the four features. Candidate books with the top-10 Borda scores are recommended to U.

$$Borda(B) = \sum_{c=1}^{C} (k - S_c^B) \tag{7}$$

where S_c^B is the position on the ranking of B based on the c^{th} ranked list to be fused.

We adopt Borda, since its combination algorithm is *simple* and *efficient*, which requires neither training nor compatible relevance scores that may not be available [1], and its performance is competitive with other existing aggregation strategies [1].

4 Experimental Results

In this section, we evaluate our recommender and compare its performance with others.

4.1 Datasets

We have chosen a number of book records included in the Book-Crossing dataset to conduct the performance evaluation of our recommender[10]. The book-crossing dataset was collected by Cai-Nicolas Ziegler [18] in 2004 with data extracted from BookCrossing.com. It includes 278,858 users who provide, on the scale of 1 to 10, 1,149,780 ratings on 271,379 books. Each book record includes a user_ID, the ISBN of a book, and the rating provided by the user (identified by user_ID) on the book. We used Amazon.com AWS advisement API to verify that the ISBNs from the book-crossing dataset are valid. The 271,379 books in the Book-Crossing dataset is denoted as BKC_DS.

4.2 Accuracy of Our RNN Classifier

Using a 80/10/10% training/validation/test split of the data as mentioned in Section 4.1, we achieved 73% classification accuracy on book test data. The accuracy could not be higher likely because of the high amounts of overlap between distinct keywords in the brief description of books with different categories, such as "Deep Learning Computing" and "Theory of computation". With 73% 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 [10], showed lower results.

4.3 Evaluation Using Individual versus Combined Features

In order to justify the necessity of employing all of the four features adopted by our recommender for identifying and ranking appealing books for a user, we have conducted an empirical study which analyzes the capability of each individual feature in making useful book recommendations and compares its performance with employing all the features. As shown in Figure 1, our book recommender that consider all the features significantly outperforms each of the individual features in terms of obtaining the lowest prediction error rates among all the features and thus in making useful suggestions to its users based on the rating prediction errors. The combined feature model achieves the highest prediction accuracy, which is less than *half* a rating (out of 10) away from the actual rating. The results clearly indicate that we take the advantage of the individual strength of each feature and greatly improves its effectiveness and the ranking of its suggested books. The overall prediction error of using all the features is 0.41 (see Figure 1), is a statistically significant improvement ($p < 0.01$) over the prediction error achieved by any individual feature based on the Wilcoxon signed-ranked test.

4.4 Comparing Book Recommendation Systems

In this section, we compared our recommender with exiting book recommenders that achieve high accuracy in recommendations on books based on their respective model.

[10] Other datasets can be considered as long as they contain user_IDs, book ISBNs, and ratings.

Fig. 1. Prediction error rates of the individual features and the combined prediction model

- **MF**. Yu et al. [16] and Singh et al. [13] predict ratings on books and movies based on matrix factorization (MF), which can be adopted for solving large-scale collaborative filtering problems. Yu et al. develop a non-parametric matrix factorization (NPMF) method, which exploits data sparsity effectively and achieves predicted rankings on items comparable to or even superior than the performance of the state-of-the-art low-rank matrix factorization methods. Singh et al. introduce a collective matrix factorization (CMF) approach based on relational learning, which predicts user ratings on items based on the items' genres and role players, which are treated as unknown values of a relation between entities of a certain item using a given database of entities and observed relations among entities. Singh et al. propose different stochastic optimization methods to handle and work efficiently on large and sparse data sets with relational schemes. They have demonstrated that their model is practical to process relational domains with hundreds of thousands of entities.
- **ML**. Besides the matrix factorization methods, probabilistic frameworks have been introduced for rating predictions. Shi et al. [11] propose a joint matrix factorization model for making context-aware item recommendations.[11] Similar to ours, the matrix factorization model developed by Shi et al. relies not only on factorizing the user-item rating matrix but also considers contextual information of items. The model is capable of learning from user-item matrix, as in conventional collaborative filtering model, and simultaneously uses contextual information during the recommendation process. However, a significant difference between Shi et al.'s matrix factorization model and ours is that the contextual information of the former is based on mood, whereas ours makes recommendations according to the contextual information on books.
- MudRecS [9] makes recommendations on books, movies, music, and paintings similar in content to other books, movies, music, and paintings, respectively that a MudRecS user is interested in. MudRecS does not rely on users' access patterns/histories, connection information extracted from social networking sites, collaborated filtering methods, or user personal attributes (such as gender and age) to perform the recommendation task. It simply considers the users' ratings, genres, role players (authors or artists), and reviews of different multimedia items. MudRecS predicts the *ratings* of multimedia items that match the interests of a user to make recommendations.

[11] The system was originally designed to predict ratings on *movies* but was implemented by [9] for additional comparisons on *books* as well.

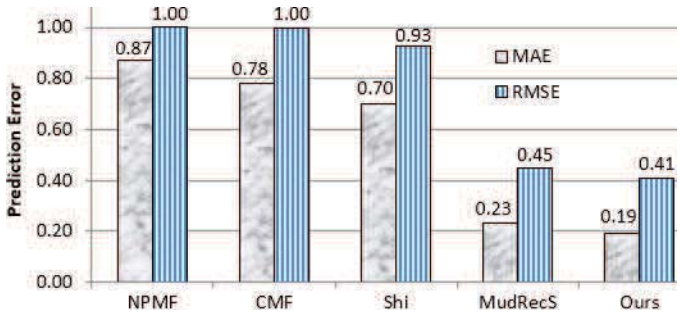

Fig. 2. The MAE and RMSE scores for various book recommendation systems based on BKC_DS, the BookCrossing dataset

Figure 2 shows the Mean Absolute Error and RMSE scores of our and other recommender systems on the BKC_DS dataset. *Root Mean Square Error* (RMSE) and *Mean Absolute Error* (MAE) are two performance metrics widely-used for evaluating rating predictions on multimedia data. Both RMSE and MAE measure the *average magnitude* of *error*, i.e., the average prediction error, on incorrectly assigned ratings. The error values computed by RMSE are squared before they are summed and averaged, which yield a relatively *high* weight to errors of *large* magnitude, whereas MAE is a *linear* score, i.e., the absolute values of individual differences in incorrect assignments are weighted equally in the average.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(f(x_i) - y_i)^2}{n}}, MAE = \frac{1}{n}\sum_{i=1}^{n}|f(x_i) - y_i| \tag{8}$$

where n is the total number of items with ratings to be evaluated, $f(x_i)$ is the rating predicted by a system on item x_i ($1 \leq i \leq n$), and y_i is an expert-assigned rating to x_i.

As the MAE and RMSE scores shown in Figure 2, our book recommender significantly outperforms other book recommender systems on rating predictions of the respective books based on the Wilcoxon Signed-Ranks Test ($p \leq 0.05$).

4.5 *Human Assessment on Our Recommender*

We further evaluated our recommender to determine whether its suggestions are perceived as preferable by ordinary users, which offers another perspective on the performance of the recommender. The additional evaluation is based on real users' assessments of the recommender which goes beyond the offline performance analysis conducted and presented in previous subsections. To accomplish this task, we conducted a user study using Amazon's Mechanical Turk (MT)[12], a "marketplace for work that requires human intelligence", which allows individuals or businesses to programmatically access thousands of diverse, on-demand workers and has been used to collect user feedback for multiple information retrieval tasks.

[12] https://www.mturk.com/mturk/welcome

Table 3. Sampled books and their corresponding subject area employed in the user study conducted using Mechanical Turk

Book Title	Subject Area
The Autobiography of Benjamin Franklin	History
Fast Food Nation: The Dark Side of ...	Cooking
The 7 Habits of Highly Effective Teens	Parenting
Think and Grow Rich: The Landmark ...	Business
Code Complete	Computer & Tech
Healthy Sleep Habits, Happy Child	Medical
Scary Stories to Tell in the Dark	Horror

In the user study, we used a set of 100 randomly-sampled books with diverse subject areas. (A number of sampled books used in this study and their corresponding subject areas is shown in Table 3.) We created a HIT (Human Intelligent Task) on MT so that for each sampled book, each appraiser was presented with a list of *five* ranked recommended books suggested by our recommender, CMF, Shi, and MudRecS, respectively and asked to select the ones that are relevant to the sampled book. The user study was conducted between March 12 and March 23, 2019 on MT. Altogether, there were 715 responses among the HITs used in the study. Based on the corresponding set of responses provided by MT appraisers, we have verified that users tend to favor our recommended books for a given book. (See Figure 3 for the results of the empirical study.)

We evaluated and compared the performance of our recommender with CMF, Shi, and MudRecS based on average P@1 (Precision at rank position 1), P@3, and P@5, and MRR (Mean Reciprocal Rank). These values are easy to compute to produce a single performance value and is readily understandable. Figure 3 shows the performance ratios computed using MT appraisers, which indicates that highly-ranked books recommended by us were treated as relevant by the MT appraisers, and the results are statistically significant ($p < 0.03$).

5 Conclusions

Reading books can enrich one's life with knowledge and deep understanding of various topics, and over the years the book industry has become an influential global consumer market. According to Statista[13], approximately 74% of the population in the U.S.A. consumed at least one book and books published in the higher education market generated nearly 4 billion US dollars in the year of 2017. With the huge amount of books available these days, various book recommendation systems have been proposed to meet user's book searching needs. Unlike many of the existing book recommender systems, our proposed book recommender simply relies solely on a deep learning model and book metadata to make personalized book recommendations. The empirical study demonstrates that our recommender outperforms well-known book recommenders.

References

1. Aslam, J., Montague, M.: Models for Metasearch. In: ACM SIGIR. pp. 267–276 (1997)

[13] https://www.statista.com/topics/1177/book-market/

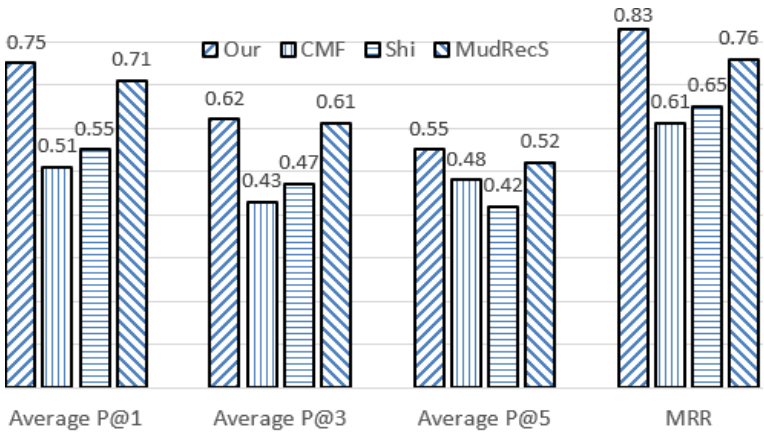

Fig. 3. Evaluation using Mechanical Turks for our book recommender

2. Ding, Z.: The Development of Ontology Information System Based on Bayesian Network and Learning. Advances in Intelligent and Soft Computing **129**, 401–406 (2012)
3. Elmer E. Rasmuson Library, University of Alaska Fairbanks: Library of Congress Subject Headings (November 2014), https://library.uaf.edu/ls101-lc-subject
4. Givon, S., Lavrenko, V.: Predicting Social-Tags for Cold Start Book Recommendations. In: ACM RecSys. pp. 333–336 (2009)
5. Kleibergen, F., Paap, R.: Generalized Reduced Rank Tests Using the Singular Value Decomposition. Econometrics **133**(1), 97–126 (2006)
6. Linden, G., Smith, B., York, J.:Amazon.com Recommendations: Item-to-item Collaborative Filtering. Internet Computing **7**(1), 76–80 (2003)
7. Mooney, R., Roy, L.: Content-based book recommending using learning for text categorization. In: ACM DL'00. pp. 195–204 (2000)
8. Owusu-Acheaw, M., Larson, A.: Reading Habits Among Students and Its Effect on Academic Performance: A Study of Students of Koforidua Polytechnic. LPP **1130** (2014)
9. Qumsiyeh, R., Ng, Y.:Predicting the Ratings of Multimedia Items for Making Personalized Recommendations In: SIGIR. pp. 475–484 (2012)
10. Ricci, F., Rokach, L., Shapira, B., Kantor, P.: Recommender Sys. Handbook. Springer (2011)
11. Shi, Y., Larson, M., Hanjalic, A.: Mining Mood-Specific Movie Similarity with Matrix Factorization for Context-Aware Recommendation. In: Workshop on CARS. pp. 34–40 (2010)
12. Siersdorfer, S., Chelaru, S., Nejdl, W., Pedro, J.: How Useful are Your Comments: Analyzing and Predicting YouTube Comments and Comment Ratings. In: WWW. pp. 891–990 (2010)
13. Singh, A., Gordon, G.: Relational Learning via Collective Matrix Factorization. In: ACM SIGKDD. pp. 650–658 (2008)
14. Sohail, S., Siddiqui, J., Ali, R.: Book recommendation system using opinion mining technique. In: IEEE ICACCI. pp. 1609–1614 (2013)
15. Yang, C., Wei, B., Wu, J., Zhang, Y., Zhang, L.: CARES: A Ranking-oriented CADAL Recommender System. In: JCDL. pp. 203–212 (2009)
16. Yu, K., Zhu, S., Lafferty, J., Gong, Y.: Fast Nonparametric Matrix Factorization for Large-Scale Collaborative Filtering. In: ACM SIGIR. pp. 211–218 (2009)
17. Zhu, Z., Wang, J.: Book recommendation service by improved association rule mining algorithm. In: ICMLC. pp. 3864–3869 (2007)
18. Ziegler, C., McNee, S., Konstan, J., Lausen, G.: Improving Recommen-dation Lists Through Topic Diversification In: WWW. pp. 22–32 (2005)