Tinderbook: Fall in Love with Culture

Enrico Palumbo[1,2,3(✉)] , Alberto Buzio[1], Andrea Gaiardo[1],
Giuseppe Rizzo[1] , Raphael Troncy[2] , and Elena Baralis[3]

[1] LINKS Foundation, Turin, Italy
{enrico.palumbo,alberto.buzio,andrea.gaiardo,
giuseppe.rizzo}@linksfoundation.com
[2] EURECOM, Sophia Antipolis, France
raphael.troncy@eurecom.fr
[3] Politecnico di Torino, Turin, Italy
elena.baralis@polito.it

Abstract. More than 2 millions of new books are published every year
and choosing a good book among the huge amount of available options
can be a challenging endeavor. Recommender systems help in choosing
books by providing personalized suggestions based on the user reading
history. However, most book recommender systems are based on collab-
orative filtering, involving a long onboarding process that requires to
rate many books before providing good recommendations. Tinderbook
provides book recommendations, given a single book that the user likes,
through a card-based playful user interface that does not require an
account creation. Tinderbook is strongly rooted in semantic technolo-
gies, using the DBpedia knowledge graph to enrich book descriptions
and extending a hybrid state-of-the-art knowledge graph embeddings
algorithm to derive an item relatedness measure for cold start recom-
mendations. Tinderbook is publicly available (http://www.tinderbook.
it) and has already generated interest in the public, involving passion-
ate readers, students, librarians, and researchers. The online evaluation
shows that Tinderbook achieves almost 50% of precision of the recom-
mendations.

Keywords: Recommender systems · Books · Knowledge graphs ·
DBpedia · Embeddings

1 Introduction

In recent years, the explosion of information available on the Web has made ever
more challenging the task of finding a good book to read. In 2010, the number
of books in the world was more than one hundred millions[1] and approximately
2,210,000 new books are published every year[2]. At the same time, a survey shows

---

[1] https://www.telegraph.co.uk/technology/google/7930273/Google-counts-total-number-of-books-in-the-world.html.

[2] https://en.wikipedia.org/wiki/Books_published_per_country_per_year.

that in the US, a reader typically reads 4 books in one year[3] and a study shows that most readers typically give up on a book in the early chapters[4]. These figures show the importance and the complexity of the process of selecting a book to read among the enormous amount of available options. Recommender systems (RS) have provided a great deal of help in this task, using algorithms that predict how likely it is for a user to like a certain item, leveraging the history of the user preferences. Most of the existing book recommender systems are typically based on collaborative filtering, which suffers from the cold start problem [1], and are thus based on long onboarding procedures, requiring users to log-in and to rate a consistent number of books (Sect. 5). On the other hand, content-based recommender systems suffer the risk of overspecialization, i.e. tend to recommend over and over again similar types of items [14]. Hybrid recommender systems combine the best of collaborative filtering and content-based similarity and are able to provide good recommendations even when user ratings are few [1]. Knowledge graphs provide an ideal data structure for such systems, as a consequence of their ability of encompassing heterogeneous information, such as user-item interactions and item-item relations in the same model. Besides, knowledge-aware recommender systems have also the advantage of being able to naturally leverage Linked Open Data [4], which provide a rich database of item descriptions and model item-item relations with semantic properties [10].

In this paper, we describe Tinderbook, a book recommender system based on knowledge graph embeddings that provides book recommendations given a single book that the user likes. To achieve this, we extend a state-of-the-art knowledge graph embeddings algorithm [15] to compute item-to-item recommendations using a hybrid item relatedness measure. In Sect. 2, we describe the recommendation algorithm, the dataset and the experimental validation of the methodological choice. In Sect. 3, we provide a high-level description of the TinderBook end-user application. In Sect. 4, we report the results obtained during the online experiment with users. In Sect. 5, we compare TinderBook with existing competing applications. In Sect. 6, we discuss the main findings and lessons learned from the deployment of the application into a production environment, as well as the future work and possible improvements of the application.

2   Recommendation Algorithm

2.1   Definitions

Definition 1. *A knowledge graph is a set $K = (E, R, O)$ where E is the set of entities, $R \subset E \times \Gamma \times E$ is a set of typed relations between entities and O is an ontology. The ontology O defines the set of relation types ('properties') Γ, the set of entity types Λ, assigns entities to their type $O : e \in E \rightarrow \Lambda$ and entity types to their related properties $O : \epsilon \in \Lambda \rightarrow \Gamma_\epsilon \subset \Gamma$.*

---

[3] https://www.irisreading.com/how-many-books-does-the-average-person-read/.

[4] https://www.nytimes.com/2016/03/15/business/media/moneyball-for-book-publishers-for-a-detailed-look-at-how-we-read.html.

Definition 2. *Users are a subset of the entities of the knowledge graph, $u \in U \subset E$. Items are a subset of the entities of the knowledge graph, $i \in I \subset E$. Users and items form disjoint sets, $U \cap I = \emptyset$.*

Definition 3. *The property 'feedback' describes an observed positive feedback between a user and an item. Feedback only connects users and items, i.e. only triples such as $(u, feedback, i)$ where $u \in U$ and $i \in I$ can exist.*

Definition 4. *Given a user $u \in U$, the set of candidate items $I_{candidates}(u) \subset I$ is the set of items that are taken into account as being potential object of recommendation.*

The problem of top-N item recommendation is that of selecting a set of N items from a set of possible candidate items. Typically, the number of candidates is order of magnitudes higher than N and the recommender system has to be able to identify a short list of very relevant items for the user. The goal of the Tinderbook application is that of recommending books to read, given a single book that the user likes. In a more formal way, we need to define a measure of item relatedness $\rho(i_j, i_k)$ which estimates how likely it is that the user will like the book i_k, given that the user likes the book i_j. The item relatedness $\rho(i_j, i_k)$ is used as a ranking function, i.e. to sort the candidate items $i_k \in I_{candidates}(u)$ given the 'seed' item i_j. Then, only the top N elements are selected and presented to the user.

2.2   Approach

The approach to define the measure of item relatedness $\rho(i_j, i_k)$ is based on entity2rec [15]. entity2rec builds property-specific knowledge graph embeddings applying node2vec [13] on property-specific subgraphs, computes user-item property-specific relatedness scores and combines them in a global user-item relatedness score that is used to provide top-N item recommendations. In this work, we extend entity2rec to a cold start scenario, where user profiles are not known and item-to-item recommendations are needed. To this end, we apply entity2rec to generate property-specific knowledge graph embeddings, but then, we focus on item-item relatedness, rather than on user-item relatedness. Property-specific item-item relatedness scores are then averaged to obtain a global item-item relatedness score that is used as a ranking function (Fig. 1). We define:

$$\rho_{entity2rec}(i_j, i_k) = avg(\rho_p(i_j, i_k)) \tag{1}$$

where $\rho_p(i_j, i_k) = cosine\_sim(x_p(i_j), x_p(i_k))$ and x_p is the property-specific knowledge graph embedding obtained using node2vec on the property-specific subgraph. We compare this measure of item relatedness with that of an ItemKNN [20], which is a purely collaborative filtering system. The relatedness between the items is high when they tend be liked by the same users. More formally, we define:

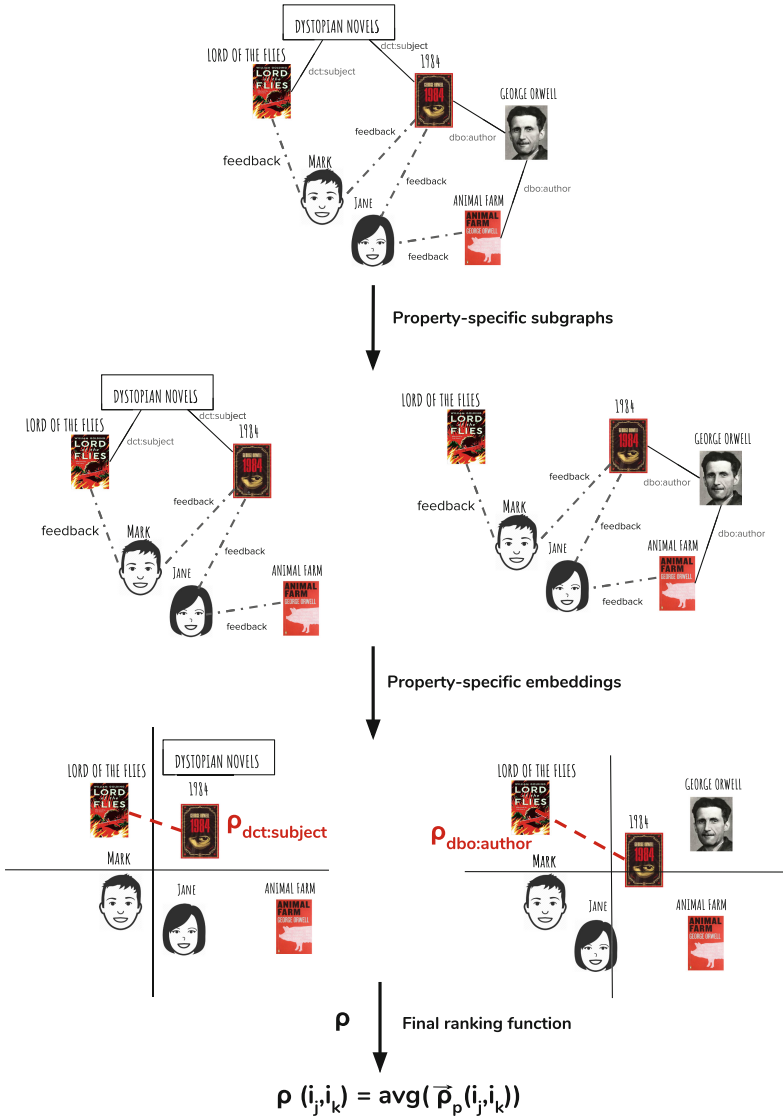$$\rho_{itemknn}(i_j, i_k) = \frac{|U_j \cap U_k|}{|U_j \cup U_k|} \tag{2}$$

Fig. 1. The knowledge graph represents user-item interactions through the special property 'feedback', as well as item properties and relations to other entities. The knowledge graph allows to model both collaborative and content-based interactions between users and items. In this figure, 'dbo:author' and 'dct:subject' properties are represented as an example, more properties are included in the experiments. Property-specific subgraphs are created from the original knowledge graph. Property-specific embeddings are computed, and property-specific item relatedness scores are computed as cosine similarities in the vectors space. Finally, property-specific relatedness scores are averaged to obtain a global item-item relatedness score.

where U_j and U_k are the users who have liked item i_j and i_k respectively. We also use as a baseline the MostPop approach, which always recommends the top-N most popular items for any item i_j. Finally, we compare entity2rec with a measure of item relatedness based on knowledge graph embeddings built using RDF2Vec [18]. RDF2Vec turns all DBpedia entities into vectors, including the books that are items of the recommender system. Thus, we simply use as a measure of item relatedness the cosine similarity between these vectors:

$$\rho_{RDF2Vec}(i_j, i_k) = cosine\_sim(RDF2Vec(i_j), RDF2Vec(i_k)) \qquad (3)$$

where $RDF2Vec(i_j)$ stands for the embedding of the item i_j built using RDF2Vec. Note that this is a purely content-based recommender such as the one implemented in [19], as DBpedia does not contain user feedback.

2.3 Offline Evaluation

The dataset used for the application and for the offline evaluation is Library-Thing[5], which contains 7,112 users, 37,231 books and 626,000 book ratings ranging from 1 to 10. LibraryThing books have been mapped to their corresponding DBpedia entities [8] and we leverage these publicly available mappings to create the knowledge graph K using DBpedia data. As done in previous work [17], we select a subset of properties of the DBpedia Ontology[6] to create the knowledge graph: ["dbo:author", "dbo:publisher", "dbo:literaryGenre", "dbo:mediaType", "dbo:subsequentWork", "dbo:previousWork", "dbo:series", "dbo:country", "dbo:language","dbo:coverArtist", "dct:subject"]. We create a 'feedback' edge between a user and a book node when the rating is $r \geq 8$, as done in previous work [8,17]. For the offline evaluation, we split the data into a training X_{train}, validation X_{val} and test set X_{test} containing, for each user, respectively 70%, 10% and 20% of their ratings. Users with less than 10 ratings are removed from the dataset, as well as books that do not have a corresponding entry in DBpedia. After the mapping and the data splitting, we have 6,789 users (95.46%), 9,926 books (26.66%) and 410,199 ratings (65.53%).

We use the evaluation protocol known as AllUnratedItems [22], i.e. for each user, we select as possible candidate items all the items present in the training or in the test set that the user has not rated before in the training set:

$$I_{candidates}(u) = I \setminus \{i \in X_{train}(u)\} \qquad (4)$$

We use standard metrics such as precision (P@k) and recall (R@k) to evaluate the ranking quality.

$$P(k) = \frac{1}{|U|} \sum_{u \in U} \sum_{j=1}^{k} \frac{hit(i_j, u)}{k} \qquad (5)$$

---

[5] https://www.librarything.com.
[6] https://wiki.dbpedia.org/services-resources/ontology.

$$\text{R}(k) = \frac{1}{|U|} \sum_{u \in U} \sum_{j=1}^{k} \frac{\text{hit}(i_j, u)}{|\text{rel}(u)|} \tag{6}$$

where the value of hit is 1 if the recommended item i is relevant to user u (rating $r \geq 8$ in the test set), otherwise it is 0, $rel(u)$ is the set of relevant items for user u in the test set and i_j are the top-k items that are recommended to u. Items that are not appearing in the test set for user u are considered as a miss. This is a pessimistic assumption, as users typically rate only a fraction of the items they actually like and scores are to be considered as a worst-case estimate of the real recommendation quality. In addition to these metrics, which are focused on evaluating the accuracy of the recommendation, we also measure the serendipity and the novelty of the recommendations. Serendipity can be defined as the capability of identifying items that are both attractive and unexpected [12]. [11] proposed to measure it by considering the precision of the recommended items after having discarded the ones that are too obvious. Equation 7 details how we compute this metric. hit_non_pop is similar to hit, but top-k most popular items are always counted as non-relevant, even if they are included in the test set of user u. Popular items can be regarded as obvious because they are usually well-known by most users.

$$\text{SER}(k) = \frac{1}{|U|} \sum_{u \in U} \sum_{j=1}^{k} \frac{\text{hit\_non\_pop}(i_j, u)}{k} \tag{7}$$

In contrast, the metric of novelty is designed to analyze if an algorithm is able to suggest items that have a low probability of being already known by a user, as they belong to the long-tail of the catalog. This metric was originally proposed by [23] in order to support recommenders capable of helping users to discover new items. We formalize how we computed it in Eq. 8. Note that this metric, differently from the previous ones, does not consider the correctness of the recommended items, but only their novelty.

$$\text{NOV}(k) = -\frac{1}{|U| \times k} \cdot \sum_{u \in U} \sum_{j=1}^{k} \log_2 \text{P}_{\text{train}}(i_j) \tag{8}$$

The function $P_{train} : I \to [0, 1]$ returns the fraction of feedback attributed to the item i in the training set. This value represents the probability of observing a certain item in the training set, that is the number of ratings related to that item divided by the total number of ratings available. In order to avoid considering as novel items that are not available in the training set, we consider $\log_2(0) \doteq 0$ by definition.

The offline experiment simulates the scenario in which the user selects a single item he/she likes i_j (so-called 'seed' book) and gets recommendations according to an item-item relatedness function $\rho(i_j, i_k)$, which ranks the candidate items i_k. We iterate through the users of the LibraryThing dataset, and for each user we sample with uniform probability an item i_j that he/she liked in the training

set. Then, we rank the candidate items $i_k \in I_{candidates}(u)$ using $\rho_{entity2rec}(i_j, i_k)$, $\rho_{itemknn}(i_j, i_k)$, $\rho_{RDF2Vec}(i_j, i_k)$ and MostPopular, and we measure P@5, R@5, SER@5, NOV@5. The results show that entity2rec obtains better precision, recall and serendipity with respect to competing systems (Table 1).

Table 1. Results for different item-item relatedness measures. entity2rec provides more accurate recommendations with respect to pure collaborative filtering such as ItemKNN and to the Most Popular baseline. It also scores better with respect to the content-based RDF2Vec, although RDF2Vec has the best novelty. Scores can be considered as without error, as the standard deviation is negligible up to the reported precision.

System	P@5	R@5	SER@5	NOV@5
entity2rec	**0.0549**	**0.0508**	**0.0514**	11.099
itemknn	0.0484	0.0472	0.0463	12.2
RDF2Vec	0.0315	0.0288	0.0311	**13.913**
mostpop	0.0343	0.0256	0.007	8.4525

3 Application

In this section, we describe the Tinderbook application.

3.1 Session

A complete usage session can be divided in two phases (Fig. 2):

1. **Onboarding:** the user lands on the application and gets books that are sampled with a chance that is proportional to the popularity of the book. More in detail, a book is sampled according to:

$$p(book) \sim P^+(book)^{\frac{1}{T}} \tag{9}$$

where P^+ is the popularity of the book, which is defined as the fraction of positive feedback (ratings $r \geq 8$) obtained by the book in the LibraryThing dataset. T is a parameter called "temperature" that governs the degree of randomness in the sampling. $T \to 0$ generates a rich-gets-richer effects, i.e. most popular books become even more likely to appear in the extraction. On the contrary, when T grows the distribution becomes more uniform, and less popular books can appear more often in the sampling. The user has to discard books (pressing "X" or swiping left on a mobile screen) until a liked book is found. The user can get additional information about the book (e.g. the book abstract from DBpedia) by pressing on the "Info" icon.

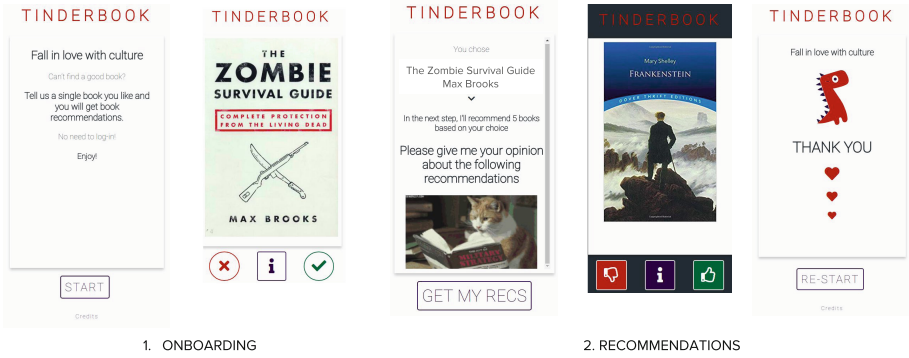1. ONBOARDING   2. RECOMMENDATIONS

Fig. 2. A complete session of use of the application. The user selects a book that she likes, gets book recommendations based on her choice and provides her feedback. User can get info about the book by pressing on the "Info" icon.

2. **Recommendations**: after the user has selected a book ("seed book"), she receives five recommended books based on her choice, thanks to the item-item relatedness $\rho_{entity2rec}$ (see Sect. 2.2). The user can provide feedback on the recommended books using the "thumbs up" and "thumbs down" icons, or swiping right or left. The user can again get additional information about the book (book abstract from DBpedia) by pressing on the "Info" icon.
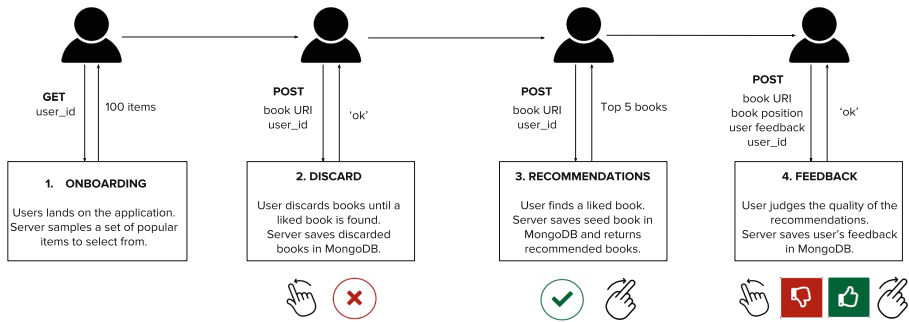

Fig. 3. Tinderbook interactions and corresponding API calls. In 1. ONBOARDING, books are sampled with a chance proportional to their popularity, as described in Eq. 9. In 2. DISCARD, the user goes through proposed books until he/she finds a liked book. In 3. RECOMMENDATIONS, the user receives five book recommendations related to his/her chosen book. In 4. FEEDBACK, the user judges the quality of the recommendations.

The graphical user interface of Tinderbook aims to engage users using playful interaction on popular like/dislike interaction [7]. The graphical representation of

cards and a slot-machine-like interaction engage the users into an infinite swipe left and right loop as the popular Tinder interface [3]. We choose to adopt digital cards interface for Tinderbook because it can be applied to a variety of contexts and, combined with ubiquitous swipe gesture, can alleviate information overload and improve the user experience aspect of apps[7]. Moreover, Tinderbook can further leverage engagement data, i.e. each individual user-swipe interaction, to get insights on users' satisfaction in the usage of the application. The interactions of the user with the application are described in Fig. 3.

3.2   Architecture

The overall architecture is presented in Fig. 4. DBpedia is the main data source for the application. DBpedia is queried to get book title, author and abstract. Google images is queried to retrieve thumbnails for images, using the book title and author extracted from DBpedia to disambiguate the query. The model is a key-value data structure that stores item-item similarities as defined in Eq. 1 and it is used to get the five most similar books to the chosen book. MongoDB is used to store the discarded books, seed books, and the feedback on the recommended books ("thumbs up" or "thumbs down"), in order to evaluate the application in the online scenario (Sect. 4). Book metadata are collected once for all the books at the start of the server and kept in memory to allow faster recommendations.
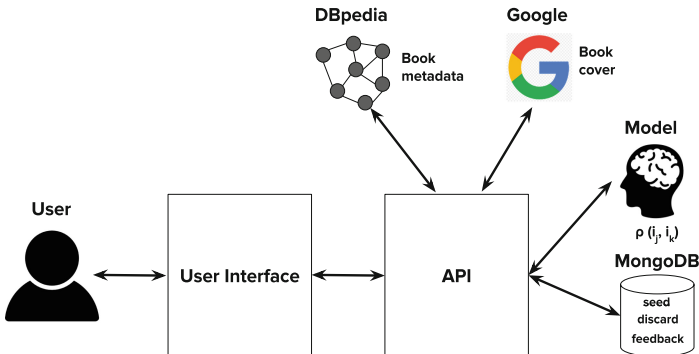

Fig. 4. The architecture of the Tinderbook application.

4   Online Evaluation

Tinderbook has been deployed on Nov 22nd, 2018. In this section, we report the results of usage data collected for two weeks, going from Nov, 22nd to Dec, 6th (Table 2). To evaluate the application, we have defined a set of Key Performance

---

[7] https://www.nngroup.com/articles/cards-component/.

Indicators (KPIs), which are specific to the online scenario, in addition to the metrics defined in Sect. 2.2. In the online experiment, we define the recommendation as a 'hit' if the user provides positive feedback ("thumb up" or swipe right), and as a 'miss' if the user provides negative feedback ("thumb down" or swipe left) in the recommendation phase. Recall cannot be measured in the online experiment, as we do not have a test set to measure $rel(u)$.

Definition 5. *We define* **completeness** *as the average percentage of rated books per session, given that the user has entered the recommendation phase.*

Definition 6. *We define* **discard** *as the average number of discarded books in the onboarding phase.*

Definition 7. *We define* **dropout** *as the percentage of users who leave the application during the onboarding phase.*

Definition 8. *We define* **seed popularity** *as the average popularity of the seed books.*

Definition 9. *We define* **recommendation time** τ *as the average time required to provide the list of recommended books in the recommendation phase.*

In the first week, we have experimented an onboarding phase with a temperature parameter set to $T = 0.3$. In the second week, we have increased this temperature to the value of $T = 1$. As described in Sect. 3, the temperature T governs the degree of randomness in the popularity-driven book sampling of the onboarding phase. The first effect observed as a consequence of the increase in temperature in the onboarding phase was the fact that less popular books were chosen during the onboarding phase. In Fig. 5, we represent the distribution of seed books falling in the top x% popular items for $T = 0.3$ and $T = 1$. The picture shows that $T = 1$ has made less popular books appear more frequently in the choices of the users in the onboarding phase with respect to the initial configuration $T = 0.3$. However, it is worth noticing that most seed books are still concentrated among the most popular books (80% in the top 20% popular books). The change of temperature has also had effects on the other KPIs. In order to compare the two different onboarding configurations $T = 0.3$ and $T = 1$, we have measured the KPIs mean values and standard deviations and run a statistical test to assess whether the observed differences were statistically significant or not. More specifically, we have run a Welch's t-student test [24] with a confidence value of $\alpha = 0.05$, only $p < \alpha$ are considered as statistically significant. As shown in Table 3, the onboarding configuration $T = 1.0$ decreases the average popularity of the seed books in a statistically significant way. This leads to the fact that users have to discard more items before finding a liked book in the onboarding phase, as it can be noticed by the increase of the average number of discarded books. However, the number of dropouts does not increase in a statistically significant way, meaning that we cannot say that this fact is pushing users to get bored during the onboarding and leave the application more

easily. In fact, it shows that users are engaged enough to keep using the application even if they have to discard more books in the onboarding. Interestingly, the configuration with $T = 1.0$ is also increasing the novelty, meaning that less popular books also appear more often in the recommendations. Overall, we can claim that $T = 1.0$ is the best configuration for the application, as it leads to more novelty without significantly increasing the number of dropouts.

Table 2. Total usage stats for the online experiment for the whole experiment (22 Nov–6 Dec), for $T = 0.3$ configuration only (22nd Nov–29th Nov) and for the $T = 1$ configuration only (30th Nov–6th Dec).

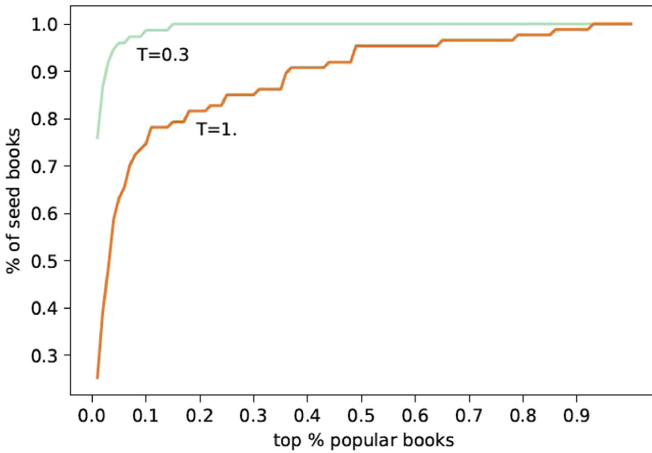|	All	T = 0.3	T = 1
tot. # seeds	470	358	112
tot. # feedback	1,936	1495	441
tot. # discarded books	3,668	2263	1405

Fig. 5. Showing how different values of the temperature affect the popularity of the books chosen as "seeds" for the recommendations in the onboarding phase. In both cases, seeds are strongly concentrated among the most popular books. However, in the case of $T = 0.3$ the effect is stronger, with all of the seeds falling into the top 20% most popular books. In the case of $T = 1$, roughly 80% of the seed books fall into the top 20% most popular books.

The recommendation time is very short, roughly 12 ms, as it involves accessing values from a key-value data store, which can be done in unitary time. More specifically, we measure $\tau = 12.4 \pm 0.3$ ms across the whole experiment.

Table 3. Online evaluation results. KPIs for the $T = 0.3$ configuration only (22nd Nov–29th Nov) and for the $T = 1$ configuration only (30th Nov–6th Dec). Welch's t-student test is used to compare the KPIs with a confidence value $\alpha = 0.05$.

	T = 0.3	T = 1	p value	Significant
P@5	0.497368 ± 0.026381	0.495833 ± 0.052701	9.79E–01	no
SER@5	0.417105 ± 0.024892	0.437500 ± 0.047382	7.07E–01	no
NOV@5	8.315443 ± 0.176832	10.095039 ± 0.347261	2.30E–05	**yes**
completeness	0.903947 ± 0.018229	0.937500 ± 0.025108	2.86E–01	no
discard	6.321229 ± 0.663185	12.544643 ± 2.070238	2.09E–03	**yes**
dropout	0.131285 ± 0.019150	0.178571 ± 0.039930	2.45E–01	no
seed pop	0.002626 ± 0.000060	0.000835 ± 0.000086	2.74E–48	**yes**

5   Competing Systems

Existing book recommender systems are typically based either on content-based or collaborative filtering [2]. In Fig. 6, we report a comparison of TinderBook with existing book recommender systems. The first point that makes TinderBook stand out from competitors is the recommendation algorithm, a hybrid approach based on knowledge graph embeddings. In the past years, several works have shown the usefulness of knowledge graphs for recommender systems, and more specifically, of Linked Open Data knowledge graphs [10]. More in detail, knowledge graphs are often used to create hybrid recommender systems, including both user-item and item-item interactions. For instance, in [9], the authors use hybrid graph-based data model utilizing Linked Open Data to extract metapath-based features that are fed into a learning to rank framework. Recently, some works have used feature learning algorithms on knowledge graphs, i.e. knowledge graph embeddings for recommender systems, reducing the effort of feature engineering and resulting in high-quality recommendations [16–18, 21, 25]. In particular, entity2rec [15], on which TinderBook is based, has shown to create accurate recommendations using property-specific knowledge graph embeddings.

The second point that makes TinderBook stand out is the Graphical User Interface (GUI) and the quick onboarding process, with no necessity of log-in or account creation. Card-based GUI are a great way to deliver information at a glance. Cards help avoid walls of text, which can appear intimidating or time-consuming and allow users to deep dive into their interests quicker. Many apps can benefit from a card-based interface that shows users enough necessary information to make a quick maybe/no decision [5]. Cards serve as entry points to more detailed information. According to Carrie Cousins, cards can contain multiple elements within a design, but each should focus on only one bit of information or content [6]. A famous example of card-based GUI is that of the dating application Tinder, and according to Babich: "Tinder is a great example of how utilizing discovery mechanism to present the next option has driven the app to emerge as one of the most popular mobile apps. This card-swiping

mechanism is curiously addictive, because every single swipe is gathering information - cards connect with users and offer the best possible options based on the made decisions" [3].

Finally, TinderBook leverages DBpedia [4] and this allows to leverage a wealth of multi-language data, such as book descriptions, without the cost of creating and maintaining a proprietary database.

	RECOMMENDATION MODE			USER PROFILING		USER EXPERIENCE		
	RECOMM. APPROACH	MIN # BOOKS FOR RECCOM.	FEEDBACK MODE	MANDATORY LOGIN	INFO REQUIRED	BOOK DESCRIPTION	WEB	MOBILE (OPTIMIZED)
goodreads	collaborative filtering	20	rating	✓	user data, favourite genres	✓	✓	✓
WHAT SHOULD I READ NEXT?	collaborative filtering	1	✗	✗	book liked	✗	✓	✓
LibraryThing	collaborative filtering	10	rating	✓	book ratings & liked	✗	✓	✗
whichbook	content-based filtering	0	✗	✗	book tags	✓	✓	✗
BookBub	human recommendation	0	like	✓	user data, favourite genres, favorite authors	✓	✓	✓
YOURNEXTREAD.COM	collaborative filtering	1	like & dislike	✗	book ratings & liked	✓	✓	✗
Readgeek	collaborative filtering	2	rating	✓	book ratings & liked	✓	✓	✓
TINDERBOOK	Hybrid filtering	1	like & dislike	✗	book liked	✓	✓	✓

Fig. 6. Comparison of existing book recommender systems.

6   Conclusions and Lessons Learned

In this paper, we have described TinderBook, a book recommender system that addresses the "new user" problem using knowledge graph embeddings. The knowledge graph is built using data from LibraryThing, containing book ratings from users, and DBpedia. We have explained the methodological underpinnings of the system, reporting an offline experiment showing that the entity2rec item relatedness based on knowledge graph embeddings is outperforming a purely collaborative filtering algorithm (ItemKNN) as well as a purely content-based system based on RDF2Vec. This is in line with the claim that hybrid recommender systems typically outperform purely collaborative and content-based systems. Then, we have provided a high-level description of the application, showing the typical usage session, the architecture and how user interactions are mapped to server API calls. We have reported the main findings of the online evaluation with users, showing that providing less popular books in the

onboarding phase improves the application, increasing the novelty of the recommendations while achieving almost 50% of precision. We have also discussed how TinderBook stands out from competing systems, thanks to its recommendation algorithm based on knowledge graph embeddings, its easy onboarding process and its playful user interface.

Semantic technologies play a fundamental role in TinderBook. DBpedia has allowed to create the knowledge graph for the recommendation algorithm, connecting books through links to common entities and complementing the collaborative information coming from LibraryThing ratings with content-based information. Furthermore, DBpedia has enabled to obtain rich book descriptions (e.g. abstract), without the cost of creating, curating and maintaining a book database. The multilinguality of DBpedia will also be a great advantage when, in the future, we will extend TinderBook to multiple languages. On the other hand, using DBpedia data has some pitfalls. The first one is the data loss during the mapping, as only 11,694 out of a total of 37,231 books (31.409%) in the LibraryThing dataset are mapped to DBpedia entities[8]. The second one is that, in some cases, the information in DBpedia resulted to be inaccurate. For example, during some preliminary tests, we have noticed that in many cases the thumbnail reported in the 'dbo:thumbnail' property is far from ideal to represent accurately the book (see Jurassic Park novel[9]), and we had to rely on Google to find better book covers. The loss of coverage and the data quality issues are relevant ones, and they open the question of whether the use of other knowledge graphs might give better results. Finally, it has to be noted that in our specific case, the cost of building the knowledge graph has been strongly mitigated by the re-use of existing DBpedia mappings. The generalization of this approach to a new dataset would require also this effort.

In spite of these challenges, users generally give positive feedback about the application, saying that it is fun to use and that recommendations are accurate. So far, it has been used by passionate readers, librarians, students and researchers, and has been promoted through the personal network of its creators, word-of-mouth, as well as popular social media. Although we do not have a precise number, we estimate that, only during the online evaluation of two weeks, more than 100 users have used the application. Some of the complaints that we have received from users is that recommendations lacked diversity or novelty. Thus, we will keep gathering data from the application and as a future work, we will try to improve other dimensions of the recommendation quality such as the diversity and the novelty, as most of the work so far has been done in optimizing the accuracy of the recommendations in an offline setting.

---

[8] https://github.com/sisinflab/LODrecsys-datasets/tree/master/LibraryThing.
[9] http://dbpedia.org/page/Jurassic_Park_(novel).

References

1. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Trans. Knowl. Data Eng. **17**(6), 734–749 (2005)
2. Alharthi, H., Inkpen, D., Szpakowicz, S.: A survey of book recommender systems. J. Intell. Inf. Syst. **51**(1), 139–160 (2018)
3. Babich, N.: Designing card-based user interfaces (2016)
4. Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. In: Semantic Services, Interoperability and Web Applications: Emerging Concepts, pp. 205–227 (2009)
5. Cai, T.: The tinder effect: swipe to kiss (keep it simple, stupid!) (2018)
6. Cousins, C.: The complete guide to an effective card-style interface design (2015)
7. David, G., Cambre, C.: Screened intimacies: tinder and the swipe logic. Soc. Media+ Soc. **2**(2) (2016)
8. Di Noia, T.: Recommender systems meet linked open data. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (eds.) ICWE 2016. LNCS, vol. 9671, pp. 620–623. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-38791-8_61
9. Di Noia, T., Ostuni, V.C., Tomeo, P., Di Sciascio, E.: SPRank: semantic path-based ranking for top-n recommendations using linked open data. ACM Trans. Intell. Syst. Technol. (TIST) **8**(1), 9 (2016)
10. Figueroa, C., Vagliano, I., Rocha, O.R., Morisio, M.: A systematic literature review of linked data-based recommender systems. Concurrency Comput. Pract. Experience **27**(17), 4659–4684 (2015)
11. Ge, M., Delgado-Battenfeld, C., Jannach, D.: Beyond accuracy: evaluating recommender systems by coverage and serendipity. In: 4th International Conference on Recommender Systems (RecSys), pp. 257–260 (2010)
12. de Gemmis, M., Lops, P., Semeraro, G., Musto, C.: An investigation on the serendipity problem in recommender systems. Inf. Process. Manage. **51**(5), 695–717 (2015)
13. Grover, A., Leskovec, J.: node2vec: scalable feature learning for networks. In: 22nd International Conference on Knowledge Discovery and Data Mining (SIGKDD), pp. 855–864 (2016)
14. Lops, P., de Gemmis, M., Semeraro, G.: Content-based recommender systems: state of the art and trends. In: Ricci, F., Rokach, L., Shapira, B., Kantor, P.B. (eds.) Recommender Systems Handbook, pp. 73–105. Springer, Boston (2011). https://doi.org/10.1007/978-0-387-85820-3_3
15. Palumbo, E., Rizzo, G., Troncy, R.: Entity2rec: learning user-item relatedness from knowledge graphs for top-n item recommendation. In: 11th International Conference on Recommender Systems (RecSys), pp. 32–36 (2017)
16. Palumbo, E., Rizzo, G., Troncy, R., Baralis, E., Osella, M., Ferro, E.: Knowledge graph embeddings with node2vec for item recommendation. In: European Semantic Web Conference (ESWC), Demo Track, pp. 117–120 (2018)
17. Palumbo, E., Rizzo, G., Troncy, R., Baralis, E., Osella, M., Ferro, E.: Translational models for item recommendation. In: Gangemi, A., et al. (eds.) ESWC 2018. LNCS, vol. 11155, pp. 478–490. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98192-5_61
18. Ristoski, P., Rosati, J., Di Noia, T., De Leone, R., Paulheim, H.: RDF2Vec: RDF graph embeddings and their applications. Semant. Web J. **10**(4) (2019)

19. Rosati, J., Ristoski, P., Di Noia, T., de Leone, R., Paulheim, H.: RDF graph embeddings for content-based recommender systems. In: CEUR Workshop Proceedings, vol. 1673, pp. 23–30 (2016)
20. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: 10th International World Wide Web Conference, pp. 285–295 (2001)
21. Shi, C., Hu, B., Zhao, X., Yu, P.: Heterogeneous information network embedding for recommendation. IEEE Trans. Knowl. Data Eng. 357–370 (2018)
22. Steck, H.: Evaluation of recommendations: rating-prediction and ranking. In: 7th ACM Conference on Recommender Systems, pp. 213–220 (2013)
23. Vargas, S., Castells, P.: Rank and relevance in novelty and diversity metrics for recommender systems. In: 5th ACM Conference on Recommender Systems, pp. 109–116 (2011)
24. Welch, B.L.: The generalization of student's problem when several different population variances are involved. Biometrika **34**(1/2), 28–35 (1947)
25. Zhang, F., Yuan, N.J., Lian, D., Xie, X., Ma, W.Y.: Collaborative knowledge base embedding for recommender systems. In: 22nd International Conference on Knowledge Discovery and Data Mining (SIGKDD), pp. 353–362 (2016)