New and Improved: Modeling Versions to Improve App Recommendation

Jovian Lin[1,2] Kazunari Sugiyama[1] Min-Yen Kan[1,2] Tat-Seng Chua[1,2]
[1]School of Computing, National University of Singapore, Singapore
[2]Interactive and Digital Media Institute, National University of Singapore, Singapore
jovian.lin@gmail.com
{sugiyama,kanmy,chuats}@comp.nus.edu.sg

ABSTRACT

Existing recommender systems usually model items as static — unchanging in attributes, description, and features. However, in domains such as mobile apps, a version update may provide substantial changes to an app as updates, reflected by an increment in its version number, may attract a consumer's interest for a previously unappealing version. Version descriptions constitute an important recommendation evidence source as well as a basis for understanding the rationale for a recommendation. We present a novel framework that incorporates features distilled from version descriptions into app recommendation. We use a semi-supervised topic model to construct a representation of an app's version as a set of latent topics from version metadata and textual descriptions. We then discriminate the topics based on genre information and weight them on a per-user basis to generate a version-sensitive ranked list of apps for a target user. Incorporating our version features with state-of-the-art individual and hybrid recommendation techniques significantly improves recommendation quality. An important advantage of our method is that it targets particular versions of apps, allowing previously disfavored apps to be recommended when user-relevant features are added.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information filtering, Search process

Keywords

Version sensitive, Recommender systems, Mobile apps, App store

1. INTRODUCTION

Mobile applications (apps) are now a part of our daily life, creating economic opportunities for companies, developers, and marketers. While the growing app market[1] has provided users with a motley collection of apps, their sheer number leads to information overload, making it difficult for users to find relevant apps.

Recommender systems have been deployed to alleviate the overload of information in app stores by helping users find relevant apps [30, 31, 6, 15, 32]. Existing recommender systems typically apply one of two methods: collaborative filtering (CF), which recommends items to target users based on other similar users' preferences, or content-based filtering (CBF), which recommends based on the similarity of an item's content and the target user's interests. However, unlike conventional items that are static, apps change and evolve with every revision. Thus, an app that was unfavorable in the past may become favorable after a version update. For example, Version 1.0 of App X did not interest a user at first, but a recent update to Version 2.0 — which promises to provide the functionality of high definition (HD) video capture — may arouse his interest in the revised app. A conventional recommender system that regards an app as a static item would fail to capture this important detail. This is why it is vital for app recommender systems to process nascent signals in version descriptions to identify desired functionalities that users are looking for.

We focus on the uniqueness of the app domain and propose a framework that leverages on *version features*; *i.e.*, textual descriptions of the changes in a version, as well as version metadata. First, with the help of semi-supervised topic models that utilize these features, we generate latent topics from version features. Next, we discriminate the topics based on genre information and use a customized popularity score to weight every unique genre-topic pair. We then construct a profile of each user based on the topics, and finally compute a personalized score of recommending the latest version of an app to a target user. Furthermore, we show how to integrate this framework with existing recommendation techniques that treat apps as static items.

Figure 1 provides an overview of our approach. App X has five different versions (1.0, 1.1, 1.2, 2.0, and 3.0). Each version is characterized by a set of latent topics that represents its contents, whereby a topic is associated with a functionality, such as the ability to capture HD videos. For instance, Version 1.0 has Topics 1, 2, and 4; whereas Version 3.0 only has Topic 5. At the same time, based on a user's app consumption history, we can model which topics they are interested in. Therefore, if Bob has a keen interest in Topic 5, the chance that he adopts App X at Version 3.0 would be higher because Topic 5 attracts Bob's interest. Likewise, there is a higher chance of both Alex and Clark adopting App X at Version 1.2 because Topics 1 and 3 attract their interests.

We show that the incorporation of version features complements other standard recommendation techniques that treat apps as static items, and this significantly outperforms state-of-the-art baselines.

[1]https://www.apple.com/pr/library/2014/01/07App-Store-Sales-Top-10-Billion-in-2013.html

SIGIR'14, July 6–11, 2014, Gold Coast, Queensland, Australia.

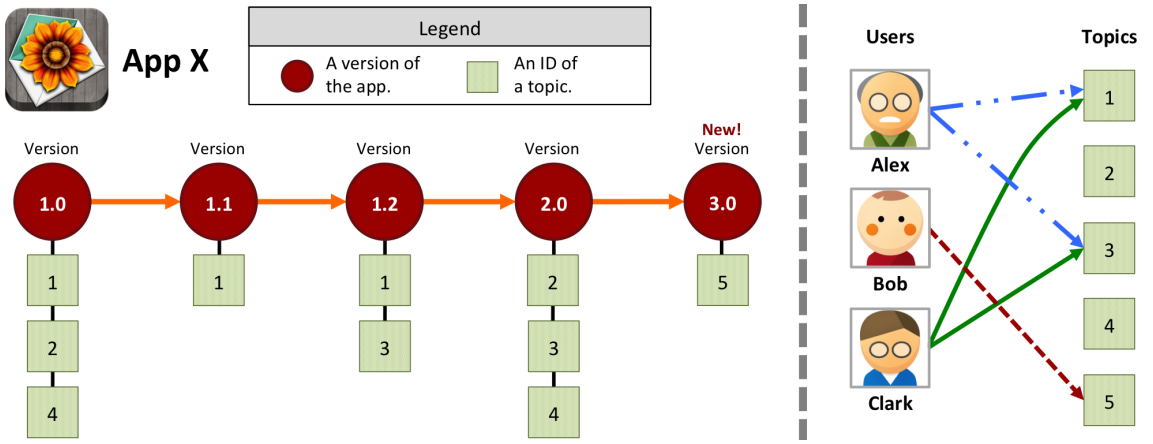


Figure 1: App X has five versions (red circles, on the left). The contents of each version is represented by a set of topics (green squares) in which each version consists of at least one topic. At the same time, based on the consumption history of users, we model them by identifying which topics they are interested in (on the right).

Our experiments identify which topic model best utilizes the available version features to provide the best recommendation, and examine the correlation between various version metadata and recommendation accuracy. Furthermore, we provide an in-depth micro-analysis that investigates: *(i)* whether our approach recommends relevant apps at the most suitable version, *(ii)* what information can we gather by scrutinizing the latent topics, and *(iii)* which is the most influential version-category. To the best of our knowledge, this is the first work that investigates version features in recommender systems. Our contributions are summarized as follows:

- We show that version features are important in app recommendation, as apps change with each version update, unlike conventional static items. This ultimately influences the needs of users and the recommended apps.
- We show how to synergistically combine our version-sensitive method with existing recommendation techniques.

2. RELATED WORK

Collaborative filtering (CF) has been widely studied. This technique can be classified into neighborhood models [5, 8, 23] and latent factor models [10, 12, 9]. Matrix factorization (MF) is one of the most successful realizations of latent factor models and is superior to the neighborhood models [25, 13]. MF associates users and items with several latent factors where observed user-item ratings matrix is their product [32].

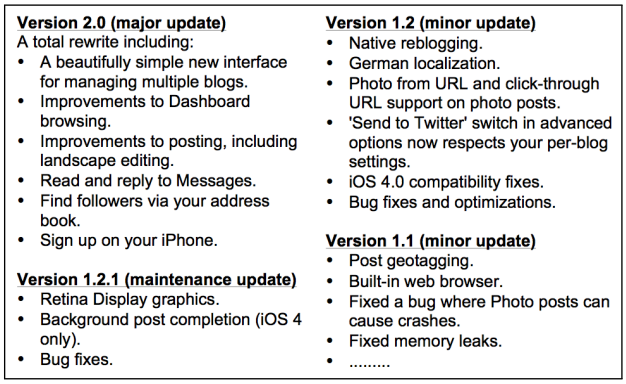
On the other hand, content-based filtering (CBF) has been applied mostly in textual domains such as news recommendation [28] and scholarly paper recommendation [18, 26, 24]. In recent years, topic models such as latent Dirichlet allocation (LDA) [4] have been used to provide an interpretable and low-dimensional representation of textual documents for recommender systems. For example, Ramage *et al.* [19] used a variant of LDA to characterize microblogs in order to recommend which users to follow, while Moshfeghi *et al.* [17] also employed LDA to combine content features and emotion information for movie recommendation.

2.1 Recommendation for Mobile Apps

In order to deal with the recent rise in the number of apps, works on mobile app recommendation are emerging. Some of these works focus on collecting additional information from the mobile device to improve recommendation accuracy. Xu *et al.* [30] investigated the diverse usage behaviors of individual apps by using anonymized network data from a tier-1 cellular carrier in the United States. Yan and Chen [31] and Costa-Montenegro *et al.* [6] constructed app recommender systems by analyzing the usage patterns of users. Other works utilize external information to improve recommendation quality. Zheng *et al.* [34] made use of GPS sensor information to provide context-aware app recommendation. Lin *et al.* [15] utilized app-related information on Twitter to improve app recommendation in cold-start situations. Yin *et al.* [32] considered behavioral factors that invoke a user to replace an old app with a new one, and introduced the notion of "actual value" (satisfactory value of the app after the user used it) and "tempting value" (the estimated satisfactory value that the app may have), thereby regarding app recommendation as a result of the contest between these two values. While the above works recommend apps that are similar to a user's interests, Bhandari *et al.* [2] proposed a graph-based method for recommending serendipitous apps.

2.2 Time-Sensitive Recommendation

Works in information retrieval (IR) have also handled items that change and evolve. For example, past works have also viewed Web pages as entities that evolve over time. Keyaki *et al.* [11] explored XML Web documents (*e.g.*, Wikipedia articles) that are frequently updated, and proposed a method for fast incremental updates for efficient querying of XML element retrieval. This is different from our work as they dealt with items only, whereas our method also generates *personalized* recommendation of items for users, *i.e.*, our work also considers the users. Furthermore, our work focuses on a secondary item unit — version updates, which is a separate entity from primary item (*i.e.*, the app). Liang *et al.* [14] proposed a method to capture the temporal dynamics of relevant topics in micro-blogs (*e.g.*, Twitter) where a topic centers around a certain theme such as the U.S. presidential election or Kate Middleton's baby which, in the micro-blogging community, may change quickly with time. Our work differs from theirs as the "items" in their system are the topics, which is an indefinite discourse. Apps, on the other hand, are definite items that users download and use. Wang and Zhang [27] explored the problem of recommending the right product at the right time, which uses a proposed opportunity model to explicitly incorporate time in an e-commerce recommender system. Their work explores the time of purchase, and does not focus on items that change with time.



Version 2.0 (major update)
A total rewrite including:
- A beautifully simple new interface for managing multiple blogs.
- Improvements to Dashboard browsing.
- Improvements to posting, including landscape editing.
- Read and reply to Messages.
- Find followers via your address book.
- Sign up on your iPhone.

Version 1.2.1 (maintenance update)
- Retina Display graphics.
- Background post completion (iOS 4 only).
- Bug fixes.

Version 1.2 (minor update)
- Native reblogging.
- German localization.
- Photo from URL and click-through URL support on photo posts.
- 'Send to Twitter' switch in advanced options now respects your per-blog settings.
- iOS 4.0 compatibility fixes.
- Bug fixes and optimizations.

Version 1.1 (minor update)
- Post geotagging.
- Built-in web browser.
- Fixed a bug where Photo posts can cause crashes.
- Fixed memory leaks.
- .....

Figure 2: An app's changelog chronicles the details of every version update; shown here is an excerpt of the Tumblr app changelog. Version updates typically include new features, enhancements, and/or bug fixes.

The works mentioned at the outset (Sections 2 and 2.1) can be characterized as static item recommendation, as the items do not undergo any change or evolution. In contrast, our work focuses on items that evolve. Furthermore, as shown later in Section 5, our work complements these techniques that treat items as static. In addition, our work differs from the previous works in time-sensitive recommendation (Section 2.2) in that the nature and requirements of app recommendation differs from the retrieval of Web articles and topic recommendation in micro-blogs.

3. OUR APPROACH

Our framework processes the version texts and metadata to decide whether a particular version of an app entices a target user. We first generate latent topics from version features using semi-supervised topic models in order to characterize each version. Next, we discriminate the topics based on genre metadata and identify important topics based on a customized popularity score. Following that, we incorporate user personalization, and then compute a personalized score for a target user with respect to an app and its version. Our system then recommends the top k target apps:

$$A : a \in \arg\max_k(score(d(a, v), u)), \quad (1)$$

where an app a and its specific version v are treated as a tuple that characterizes a document d, and is scored with respect to a target user u. Lastly, we explain how to integrate this framework with existing recommendation techniques.

3.1 Version Features

App versioning is the process of assigning unique version numbers to unique states of the app. Within a given version number category (*e.g.*, *major*, *minor*), these numbers are generally assigned in increasing order and correspond to new developments in the app. Figure 2 shows an example of an app's changelog that consists of four different version updates (or *version-snippets*) in reverse order: Versions 2.0, 1.2.1, 1.2, and 1.1. Hereafter, we will use the terms "version-snippet" and "document" interchangeably to refer to the textual description of each version update.

Versions are identified using a conventional numbering structure of "X.Y.Z" where X, Y, and Z represent *major*, *minor*, and *maintenance* categories, respectively:

1. **Major** — Major versions indicate significant changes to the app and is incremented when new major releases are made. This usually denotes that substantial architectural changes have taken place. For example, in Figure 2, Version 2.0 is a *major* version-category.

2. **Minor** — The minor version category is often applied when new functionality is introduced or important bug fixes are introduced. The dependant *maintenance* number (covered next) is reset to zero. For example, in Figure 2, Versions 1.1 and 1.2 are *minor* version-categories.

3. **Maintenance** — The maintenance version category is associated with non-breaking bug fixes. For example, in Figure 2, Version 1.2.1 is a *maintenance* version-category.

Hereafter, we will use "version-category" as shorthand for version number category.

Besides textual descriptions and version-categories, each version-snippet is also associated with the following information:

1. **Genre Mixture** — Every app is assigned to a subset of pre-defined genres by the developer. For example, the app "Instagram[2]" is assigned to the genres "photo & video" and "social networking." As a version is essentially one of many unique states of an app, the genre mixture in which an app is assigned to is also inherited by its versions. Additionally, as the aforementioned "Instagram" example shows, each app or version is typically assigned to multiple genres[3]. Figure 3 shows all of the 40 pre-defined genre labels in the case of Apple's iOS app store (our focus in this study).

2. **Ratings** — It is commonly known that user ratings are directly paired to apps, *i.e.*, user u gives app a a numerical rating r. However, to be strictly pedantic, the app stores of Apple and Google pair ratings to a particular version of an app: user u gives version v of app a a numerical rating r. Therefore, every version — even if it is the *same* app — receives a different set of ratings from different users. A version that was rated poorly in the past may receive more favorable ratings for later versions.

Books, Business, Catalogs, Education, Entertainment, Finance, Food & Drink, Health & Fitness, Lifestyle, Medical, Music, Navigation, News, Photo & Video, Productivity, Reference, Social Networking, Sports, Travel, Utilities, Weather

Action, Adventure, Arcade, Board, Card, Casino, Dice, Educational, Family, Kids, Music, Puzzle, Racing, Role Playing, Simulation, Sports, Strategy, Trivia, Word

Figure 3: The 40 pre-defined genre labels on Apple's iOS app store (as of January 2014). The bottom set are gaming sub-genres and only appear on gaming apps.

3.2 Generating Latent Topics

In order to find an interpretable and low-dimensional representation of the text in the version-snippets (or documents), we focus on the use of topic modeling algorithms (topic models). A topic model

[2]http://itunes.apple.com/lookup?id=389801252

[3]This information (of having more than one genre) is only displayed through the API calls to the app store, and is not displayed in the regular app store that consumers use; instead, only one (primary) genre is shown to the consumer.

takes a collection of texts as input and discovers a set of "topics" — recurring themes that are discussed in the collection — and the degree to which each document exhibits those topics.

We first explore the use of two different topic models: *(i)* latent Dirichlet allocation (LDA) [4] and *(ii)* Labeled-LDA (LLDA) [20], which are unsupervised and semi-supervised topic models, respectively. We also investigate a corpus-enhancing strategy of incorporating version metadata directly into the corpus prior to the application of topic models. This is to improve the quality of the topic distribution discovered by the topic models.

3.2.1 Modeling Version-snippets with Topic Models

LDA is a well-known generative probabilistic model of a corpus; it generates automatic summaries of latent topics in terms of: *(i)* a discrete probability distribution over words for each topic, and further infers *(ii)* per-document discrete distributions over topics, which are respectively defined as:

$$p(w|z), \tag{2}$$
$$p(z|d), \tag{3}$$

where z, d, and w denote the latent topic, the document, and a word, respectively.

However, a limitation of LDA is that it cannot incorporate "observed" information as LDA can only model the text in version descriptions, *i.e.*, LDA is an unsupervised model. In the context of our work, this means that we cannot incorporate the observed version metadata (*e.g.*, version-category and genre mixture) into the latent topics. This leads us to Labeled-LDA (or LLDA), an extension to LDA that allows the modeling of a collection of documents as a mixture of some observed, "labeled" dimensions [20], representing supervision.

LLDA is a supervised model that assumes that each document is annotated with a set of observed labels. It is adapted to account for multi-labeled corpora by putting "topics" in one-to-one correspondence with "labels", and then restricting the sampling of topics for each document to the set of labels that were assigned to the document. In other words, these labels — instead of topics — play a direct role in generating the document's words from per-label distributions over terms. However, LLDA does not assume the existence of any global latent topics, only the document's distributions over the observed labels and those labels' distributions over words are inferred [21]. This makes LLDA a *purely supervised* topic model.

Although LLDA appears to be a supervised topic model initially, depending on the assignment of the set of labels to the documents, it can actually function either as an *unsupervised* or *semi-supervised* topic model. To achieve an *unsupervised* topic model like LDA, we first disregard all the observed labels (if any) in the corpus, and then model K latent topics as labels named "Topic 1" through "Topic K" and assign them to *every* document in the collection. This makes LLDA mathematically identical to traditional LDA with K latent topics [19]. On the other hand, to achieve a *semi-supervised* topic model, we first assign *every* document with labels named "Topic 1" through "Topic K" for the *unsupervised* portion, and then use the observed labels[4] (that are unique to each document) for the *supervised* portion.

The *semi-supervised* method of implementing LLDA allows us to quantify broad trends via the latent topics (as in LDA) while at the same time uncover specific trends through labels associated with document metadata. In our work, we treat the version categories and genre mixture as observed labels, and rely on *semi-supervised* LLDA to discover the words that are best associated with the different version-categories and genres, respectively.

[4]Note that the number of *observed* labels varies with every document.

Algorithm 1 How to create "pseudo-terms" from metadata and incorporate them into the corpus (in pseudocode).

```
For each doc "d" in corpus:
// Note that each doc is a version-snippet.
    verText = d.getText();
    verCategory = d.getVersionCategory();
    // We assume verCategory already has the
    // hash-prefix (i.e., "#"-prefix).
    appId = d.getAppId();
    genres = getGenres(appId);
    // We assume genres are comma separated values
    // and already have the hash-prefix.
    verText += genres + "," + verCategory;
    d.setText(verText);
```

Similar to LDA, LLDA generates the *topic-word* and *document-topic* distributions in Equations (2) and (3), respectively, allowing us to obtain the mixture weights of topics for every document. Hereafter, semi-supervised LLDA will be the default LLDA model, and we will also use the terms "topic" and "label" interchangeably.

3.2.2 Corpus-enhancement with Pseudo-terms

Aside from employing topic models, we identify another way of incorporating metadata into the latent topics. Inspired by how hashtags are used in Twitter to add content to Twitter messages, we create "pseudo-terms" from the metadata and incorporate them into the set of documents before performing topic modeling. These pseudo-terms can be identified by their "#" prefix. Algorithm 1 shows how metadata in the form of pseudo-terms are automatically "injected" into the corpus of version-snippets, as we want to associate these pseudo-terms with the latent topics.

Because LDA and LLDA generate automatic summaries of topics in terms of a discrete probability distribution over words for each topic [20], incorporating pseudo-terms into the corpus allows the topic models to learn the posterior distribution of each pseudo-term (in addition to the natural words) in a document conditioned on the document's set of latent topics. Incorporating these unique pseudo-terms will help in getting topic distributions that are more consistent with the nature of version-snippets. Note that the difference between using the enhanced-corpus and the normal corpus is that the former allows both the words and pseudo-terms to be associated with the latent topics, while the latter only allows (natural language) words to be associated with the latent topics. To differentiate between the normal corpus and the enhanced-corpus, we add the prefix "inj"ection to LDA and LLDA; in shorthand, "inj+LDA" and "inj+LLDA," respectively, to denote these approaches.

3.3 Identifying Important Latent Topics

We can now model each version-snippet (or document) as a distribution of topics. However, we do not know which topics are important for recommendation. For example, if we knew that users prefer a topic that is related to the promise of high-definition (HD) display support, we would rather recommend an app that includes HD display support in its latest version update over similar apps that do not. Therefore, the importance of each topic differs from app to app, and this is a key contribution of our work.

Furthermore, apps are classified into various genres; each genre works differently to the same type of version update. For example, a version update that offers HD display support would be more enticing and relevant on a *game* app instead of a *music* app. Later, in Section 5.2, we will show how the inclusion of genre information significantly improves the recommendation accuracy. Because of this, our method includes genre information by default. Table 1

Table 1: Genre-topic weighting matrix, where g and z denote a genre and a latent topic, respectively. Every genre-topic pair has a unique weight from weighting scheme. Also, $x \in$ {LDA, inj+LDA, LLDA, and inj+LLDA}.

Genre	Latent Topic						
	z_1	z_2	...	z_j	...	z_{K-1}	z_K
g_1	$w_{x_{1,1}}$	$w_{x_{1,2}}$	...	$w_{x_{1,j}}$	...	$w_{x_{1,K-1}}$	$w_{x_{1,K}}$
g_2	$w_{x_{2,1}}$	$w_{x_{2,2}}$	...	$w_{x_{2,j}}$	...	$w_{x_{2,K-1}}$	$w_{x_{2,K}}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
g_i	$w_{x_{i,1}}$	$w_{x_{i,2}}$	...	$w_{x_{i,j}}$	...	$w_{x_{i,K-1}}$	$w_{x_{i,K}}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
g_G	$w_{x_{G,1}}$	$w_{x_{G,2}}$	...	$w_{x_{G,j}}$	...	$w_{x_{G,K-1}}$	$w_{x_{G,K}}$

shows how we uniquely weight every genre-topic pair with multiplicative weight w_x, where $x \in$ {LDA, inj+LDA, LLDA, and inj+LLDA}. Note that each genre has a different distribution of importance weights with respect to the set of latent topics.

To compute the weight w, we first introduce a measurement for "popularity" for a document. We use a variant of the popularity measurement detailed in [33] whereby the popularity is reflected by the votes it receives; as intuitively, the more positive votes it receives, the more popular it is and vice versa. While one may argue that an item receiving a large number of votes (whether they are positive or not) is popular, in this work, we define popular items as those that are "liked" by the majority of the service users, whereby a "like" translates to a rating of 3 and above on the 5-point Likert scale, whereas a "dislike" is a rating of 2 and below.

We formally define the popularity score $\pi(d)$ that outputs a value between 0 and 1, which factors user ratings into account:

$$\pi(d) = \begin{cases} \frac{pv_d - nv_d}{pv_d + nv_d + 1} & \text{if } pv_d - nv_d > 0 \\ 0 & \text{otherwise,} \end{cases} \quad (4)$$

where pv_d and nv_d denote the number of positive and negative ratings of document d, respectively.

We use this popularity score to define the importance weight of a genre-topic pair, w_x:

$$w_x(g, z) = \frac{\sum_{d \in D(g)} p(z|d) \cdot \pi(d)}{\sum_{z' \in Z} \sum_{d \in D(g)} p(z'|d) \cdot \pi(d)}, \quad (5)$$

where Z is the set of all K topics, $D(g)$ is the set of all documents that belongs to genre g, $\pi(d)$ is the popularity score of document d, $p(z|d)$ is the *document-to-topic* distribution in Equation (3), and $x \in$ {LDA, inj+LDA, LLDA, and inj+LLDA}. The denominator is used solely for normalization. In other words, w_x is discriminated by the genre, and information from the ratings, along with the distribution of topics, are used to identify its weights.

3.4 User Personalization

To incorporate personalization, we need to know each user's preference with respect to the set of latent topics. We determine this importance by analyzing the topics present in the apps that a user u has previously consumed. To compute this factor with respect to a latent topic z, we define the following equation:

$$p(z|u) = \frac{\sum_{d \in D(u)} p(z|d)}{\sum_{z' \in Z} \sum_{d \in D(u)} p(z'|d)}, \quad (6)$$

where $p(z|d)$ is the *document-to-topic* distribution defined in Equation (3) and $D(u)$ is the set of documents consumed by user u. As in Equation (5), the denominator is solely for normalization.

3.5 Calculation of the Version-snippet Score

Finally, we calculate the score defined by Equation (1). We combine the *document-to-topic* distribution defined in Equation (3), the weighting schemes defined by Equation (5), the user-personalization factor defined by Equation (6), and compute the score as follows:

$$score_x(d, u) = \sum_{z \in Z} p(z|d) \cdot w_x(genre(d), z) \cdot p(z|u), \quad (7)$$

where d, u, and z are the document, target user, and latent topic, respectively, $w_x(\cdot)$ denotes the weighting schemes (where $x \in$ {LDA, inj+LDA, LLDA, and inj+LLDA}), $genre(d)$ is the *genre* of document d, $p(z|d)$ is the *document-to-topic* distribution in Equation (3), and $p(z|u)$ is the probability that the target user u prefers topic z. Thus, for each app, we calculate its score based on its latest version to see if it should be recommended.

3.6 Combining Version Features with Other Recommendation Techniques

Our work aims at exploring how version features can improve the recommendation accuracy of existing recommendation techniques such as CF and CBF. A simple way to integrate version features with the other recommendation techniques is to use a weighted combination scheme, but we also explore a more advanced approach, Gradient Tree Boosting (GTB) [7], which is a machine learning technique for regression problems that produces a prediction model in the form of an ensemble of prediction models. We show the results of GTB in our work as it is more superior.

For each of the users, we fit a GTB model to their training data (for each app in the training data that a user has consumed). Each training sample contains the prediction scores of the various recommendation techniques and the actual rating value of the user for the particular app. Note that for our version-sensitive recommendation (VSR) score, we map the score of the version-snippet to the app. We assume a recommendation technique — such as CF and CBF or any other — provides a probability of the likelihood of user u consuming or downloading app a. The features given to GTB are a set of probability scores of each of the recommendation techniques, VSR, CF, and CBF; the output of GTB is a predicted score between 0 and 5. The predicted ratings are then ranked in reverse order for recommendation.

4. EVALUATION

We preface our evaluation proper by detailing: 1) how we constructed our dataset, 2) how we chose our evaluation metric, 3) our setting for the dataset, and 4) the baselines that we compare our approach against.

4.1 Dataset

We constructed our dataset by culling from the iTunes App Store[5] and AppAnnie[6]. The dataset consists of the following elements:

1. **App Metadata**. App metadata consists of an app ID, title, description, and genre. The metadata is collected by first getting all the app IDs from the App Store, and then retrieving the metadata for each app via the iTunes Search API[7].

2. **Version Information**. For each app, we utilize a separate crawler to retrieve all its version information from AppAnnie, which resembles the changelog in Figure 2. We treat each app's version as a *document*.

[5] https://itunes.apple.com/us/genre/ios/id36?mt=8

[6] http://appannie.com

[7] https://www.apple.com/itunes/affiliates/resources/documentation/

3. **Ratings**. For each version, we utilize yet another crawler to collect its reviews from the iTunes App Store. A review contains an app's ID, its version number, its rating, the reviewer's ID, the subject, and the review comments. This is the source of the rating feature. Note that a rating here is associated to a particular *version* of an app.

We further process the dataset by selecting apps with at least 5 versions, documents (*i.e.*, version-snippets) with at least 10 ratings, and users who rated at least 20 apps. With these criteria enforced, our dataset consists of 9,797 users, 6,524 apps, 109,338 versions, and 1,000,809 ratings. We then perform a 5-fold cross validation, where in each fold, we randomly select 20% of the users as target users to receive recommendations. For each target user, we first remove 25% of their most recent downloaded apps, by default. Additionally, among the training data, 70% is used for training the latent topics while the remaining 30% is used for the training of GTB. Recommendation is evaluated by observing how many masked apps are recovered in the recommendation list.

4.2 Evaluation Metric

Our system ranks the recommended apps based on the ranking score. This methodology leads to two possible evaluation metrics: precision and recall. However, a missing rating in the training set is ambiguous as it may either mean that the user is not interested in the app, or that the user does not know about the app (*i.e.*, truly missing). This makes it difficult to accurately compute precision [26]. But since the known ratings are true positives, we believe that recall is a more pertinent measure as it only considers the positively rated apps within the top M, namely, a high recall with a lower M will be a better system.

As previously done in [26, 15], we chose Recall@M as our primary evaluation metric. Let n_u and N_u be the number of apps the user likes in the top M and the total number of apps the user likes, respectively. Recall@M is then defined as their ratio: n_u/N_u. We compare systems using average recall, where the average is computed over all test users.

4.3 Optimization of Parameters

For the number of topics K of LDA and LLDA, we experimented on a series of K values between 100 to 1200 for each topic model, and selected the K that maximizes the recall in each model. For the α and β hyperparameters of LDA and LLDA, we used a low α-value of 0.01 as we want to constrain a document to contain only a mixture of a few topics; likewise, we used a low β-value of 0.01 to constrain a topic to contain a mixture of a few words. For the parameters of GTB, we used the default values in scikit-learn[8], whereby we employed 500 trees, a depth level of 3, and the least square for the loss function.

4.4 Baselines

We considered two state-of-the-art recommendation techniques as baselines: *(i)* probabilistic matrix factorization (PMF) [22] which represents collaborative filtering (CF); and *(ii)* latent Dirichlet allocation (LDA) [3] which represents content-based filtering (CBF).

PMF has been widely used in previous works [22, 1, 16] as an implementation of CF as it is highly flexible and easy to extend. On the other hand, LDA has been used in previous works [3, 17, 26, 15] as an implementation of CBF as it effectively provides an interpretable and low-dimensional representation of the items. Note that in the context of our experiments, LDA's implementation of CBF uses the apps' descriptions as documents — *not* the version features. Besides pure CF and CBF, we also show the recommendation accuracy obtained by hybrid of individual techniques, namely, *(i)* CF+CBF, *(ii)* CF+VSR, *(iii)* CBF+VSR, and *(iv)* CF+CBF+VSR, where VSR represents our version-sensitive recommendation approach proposed in Section 3.

[8]http://scikit-learn.org/

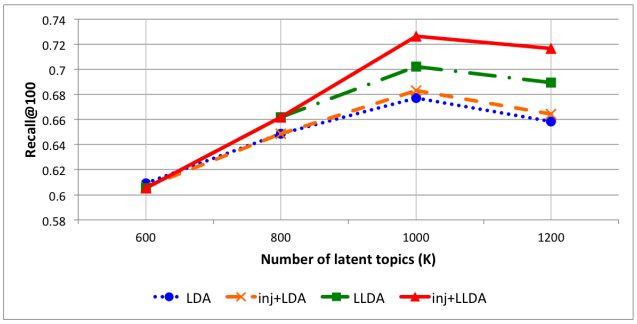


Figure 4: For each of the four topic models, we experimented with various K between K=100 and K=1200, and show a sub-sampled chart of K intervals that are fixated at Recall@100.

5. EXPERIMENTS

We first show the recommendation accuracy evaluated with recall by varying the number of latent topics K, and then show how recall is affected when we exclude an app's genre information. After which, we show the performance of the four topic models proposed in Section 3.2. Finally, we compare our approach with other recommendation techniques, including hybrid methods described in Section 4.4.

5.1 Recommendation Accuracy Obtained by Different Number of Latent Topics

We optimize the number of topics, K, for our VSR approach with respect to our four new topic models. Figure 4 shows the recall when varying K for LDA, inj+LDA, LLDA, and inj+LLDA, respectively. We observe that K=1000 gives the best recall scores for all four models, and that the recall scores generally show a steep increase towards the optimum (*i.e.*, between K=600 and K=1000), and then gradually decline once K exceeds this optimum (*i.e.*, between K=1000 and K=1200). K=1000 may be seen as a large number of topics, but as observed by [29], larger datasets like ours (we have 109,338 documents) may necessitate a larger number of topics to be modeled well. Additionally, as we had previously constrained both hyperparameters of the topic models to be small (resulting in low topic-mixture per document), more topics are needed to represent the set of documents.

5.2 Importance of Genre Information

Our framework allows each genre to assign different weights to identical latent topics. In order to determine the importance of genre information, we compare the recommendation accuracies between models with and without genre information. Both variants are based on the best-performing model (inj+LLDA). Figure 5 shows that the variant incorporating genre outperforms the plain model with a statistical significance at $p < 0.01$. We conclude that genre information is an important discriminatory factor, as each genre weights the same type of version update differently. For example, a version update that offers the support for HD displays would be more attractive and relevant to a *game* app instead of a *music* app. Therefore, by discriminating the genres, we assign

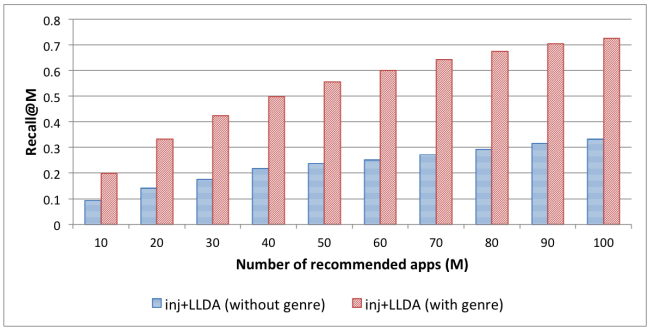


Figure 5: Recall scores between the inj+LLDA model that uses genre information and another that does not.

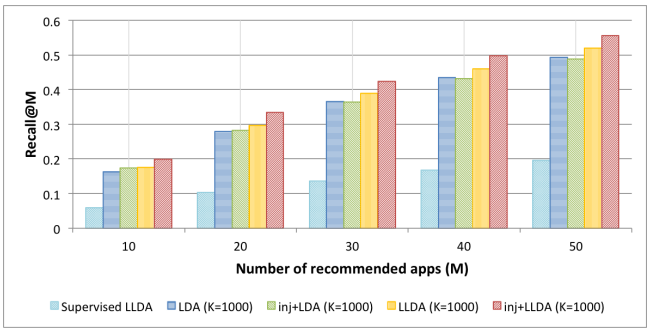


Figure 6: Recall scores of different topic modeling schemes with K=1000 as the optimal number of topics.

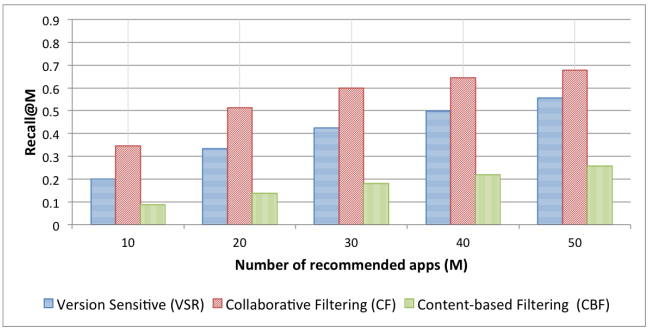


Figure 7: Recall scores of our version-sensitive model (VSR) against other individual recommendation techniques.

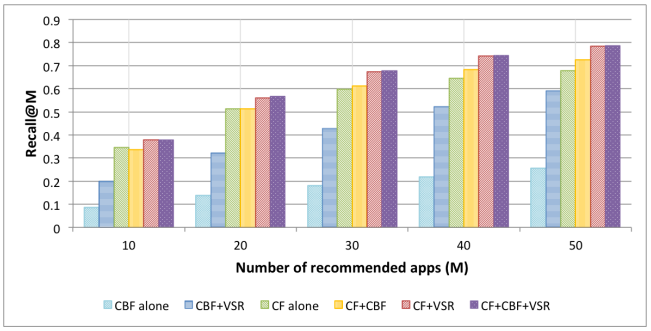


Figure 8: Recall scores of various combinations of recommendation techniques.

more relevant weights, which results in better recall. As such, we use genre information in all of the subsequent experiments.

5.3 Comparison of Different Topic Models

Figure 6 shows the performance of the five different topic model variants: *(i)* supervised-LLDA (*i.e.*, without K latent topics), *(ii)* LDA, *(iii)* inj+LDA, *(iv)* LLDA, and *(v)* inj+LLDA. So that we can compare unsupervised, supervised, and semi-supervised models, we added supervised-LLDA for the purpose of completeness.

We see that recall is consistently improved as the basic LDA model is incrementally enhanced through inj+LDA, LLDA, and inj+LLDA. Between the inj+LDA and inj+LLDA models that use the enhanced-corpus (*cf.* Section 3.2.2) and the LDA and LLDA models that do not, we observe that the enhanced-corpus generally provides better recall, with inj+LLDA showing more significant performance against LLDA. Furthermore, both models of LLDA (*i.e.*, LLDA and inj+LLDA) consistently outperform the pure LDA models, which shows that semi-supervised LLDA models are superior to LDA, which is due to LLDA's ability to quantify broad trends via latent topics while at the same time uncovering specific trends through observed metadata.

We added supervised-LLDA as a baseline for this specific evaluation, but we see that it performs worst among all the baselines. The reason why supervised-LLDA is the worst model despite having "supervision" is that it does not have sufficient topics to properly capture the essence of the corpus.

As inj+LLDA is the best-performing model among the topic models we have tested, we use it in subsequent comparisons. We see that use of version metadata improves recall, as the three models that utilize metadata (*i.e.*, inj+LDA, LLDA, and inj+LLDA) consistently outperform the LDA model that only utilizes the text from version-snippets.

5.4 Comparison against Other Recommendation Techniques

Figure 7 shows the recall scores of the three individual techniques — VSR, CF, and CBF — where the VSR approach uses inj+LLDA at the optimal settings of K=1000. While VSR underperformed against CF, it does outperform CBF. We believe this is because the textual features in the app descriptions are noisy [15], resulting in poor recommendation. Thus, among the content-based recommendation approaches of the app domain, version features are promising replacements for app descriptions.

Figure 8 shows the combination of individual techniques using GTB. We observe that combining VSR with CBF or CF (*i.e.*, CBF+VSR or CF+VSR) improves both CF or CBF alone. This suggests that version features are a good complement to the traditional recommendation techniques that treat apps as static items. As version features focus on the unique differences between various states of an app, they play a natural complementary role for CF or CBF alone. In addition, we have further confirmed that feature-wise, version features are better content descriptions as CF+VSR outperforms CF+CBF. Furthermore, we note that the best performing hybrid is CF+CBF+VSR, though it is roughly on par with CF+VSR. Finally, the hybrid methods CF+VSR and CF+CBF+VSR outperform the pure CF model with a statistical significance of $p < 0.01$ at Recall@50.

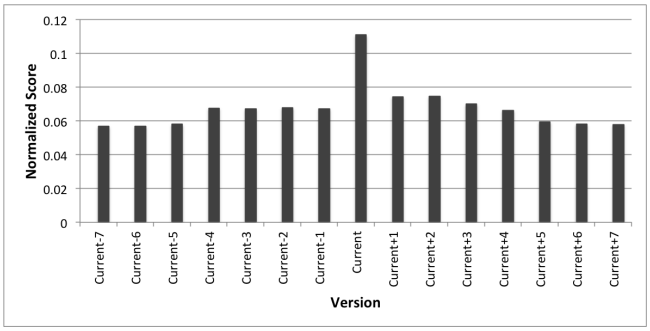


Figure 9: Comparison of normalized score among past (current -1 to -7), current, and future (current +1 to +7) versions.

6. DISCUSSION

We examine the experimental results obtained by the use of version features in detail. First, we perform an in-depth study that compares a recommended version against previous and future versions of the same app. Next, we perform a micro-analysis on individual latent topics and investigate the terms that are found in each topic. Finally, we investigate the effect of injecting more complex version-category information.

6.1 Comparison of Previous, Current, and Future Versions of Apps

From our dataset, we only know which version of an app a target user has downloaded. However, we do not know whether the user has or has not seen previous versions of the app before downloading the current version. For example, we only know that Bob downloaded AngryBirds Version 2.1 but we do not know whether:

- Bob had seen previous versions of AngryBirds (*e.g.*, Version 1.0) but was not interested in downloading it at that time, or
- Bob's first encounter with the AngryBirds app was in fact at Version 2.1 and that it was the version that he downloaded.

Hence, we need to consider the situation where a user did not download a target app earlier even though it might be available for download; and that it was only *after* a version-update did the app attract him. For this reason, based on every app that each target user in the training set downloaded, we input the current version (*i.e.*, the version which the target user downloaded) *as well as* the previous and future versions of the same app, and find out whether our system can recommend the exact version that the target user downloaded.

In order to conduct a fair study, we have to take into account the fact that every app has different number of version updates. For example, some apps may only have 5 different version updates while others may have as many as 20 version updates. To solve this problem, we fit the versions of every app into three sets of bins: The first set of bins denotes the previous versions (*i.e.*, bins #1 to #7), the second set of bins denotes the version of the app that a user has downloaded (*i.e.*, bin #8), the last set of bins represent the future versions (*i.e.*, bins #9 to #15). Then, for every app that a target user has consumed, we calculate the score for each version (explained in Section 3.5), and enter the score into the respective bins. Finally, we normalize the score of every bin.

Figure 9 shows the normalized score of this analysis for all target users in the training set. We observe that our approach favors the current version (*i.e.*, the one that was downloaded by the target users) the most, thereby indicating that our VSR model effectively targets the version of an app that maximizes its chances of being acquired by the target user. This also reflects that apps tend to go through a series of revisions before being generally favorable; after which the subsequent versions show a decline in general interest, and this suggests the peripheral nature of the subsequent revisions.

6.2 Dissecting Specific LDA Topics

To further understand why injecting pseudo-terms into the corpus improves recommendation accuracy, we perform a micro analysis by exploring the latent topics discovered by inj+LLDA. We selected the three most important latent topics based on the expectation of each latent topic over the set of training data. Note that each latent topic contains a set of words as well as the injected pseudo-terms.

Figure 10 shows the three topics. We observe that every topic coincides with a certain theme. In addition, from the pseudo-terms found in the topic, we can discern the kind of *version-category* and *genre mixture* information the topic belongs to. For example, Topic #385 contains words like "retina" and "resolution", correctly suggesting that the update is display-related. In addition, we observe what genres of apps most likely have such updates, which are the *Utilities* and *Productivity* apps (in red). Furthermore, we observe that updates in Topic #385 are strongly related to the version-category *minor* (in blue). On the other hand, Topic #47 is associated with *navigation* and *traveling*, as the genre-related pseudo-terms (in red) suggests. The top natural language words found in Topic #47 also agree with the hashtags, in that the related updates include improvements in mapping and routing, and that the updates also include alerts and notifications with regards to traveling-related information, such as fuel, points of interests (POIs), and accidents. Finally, as we recall that inj+LLDA allows the incorporation of "observed" labels as topics, the third topic is related to the "medical genre" label and it is closely associated with apps in the neighboring "Health & Fitness" genre. This "observed" label/topic mainly deals with providing users visual reports (such as graphs and charts) about their personal health (such as periods and pregnancy) as well as the provision of personal tracking and reminders. We observe that the injected pseudo-terms act as a guide for inj+LLDA's inferencing process, which contributes to better latent topic generation. It also helps in understanding the topics further as the metadata (*i.e.*, version-categories and genre mixture) that is imbued in the topics gives users a more comprehensible understanding of the topics.

6.3 Importance of Version Categories

To verify the importance of various version-categories (*i.e.*, major, minor, and maintenance), we calculate their respective scores based on *(i)* the topic-word distribution from the topic model, and *(ii)* the importance score of the latent topics (Section 3.3), which is essentially: $\sum_{g \in G} \sum_{z \in Z} w_{\text{inj+LLDA}}(g, z) \cdot p(w = m|z)$, where m represents one of the strings: "#major", "#minor", or "#maintenance." Note that $p(w|z)$ is the *topic-word* distribution in Equation (2). Also note that the equation must be normalized, which results in the score being between 0 and 1.

The importance of each of the three version-categories are as follows: *(i)* "#major": 0.128, *(ii)* "#minor": 0.656, and *(iii)* "#maintenance": 0.216. It is evident that the "minor" version category is the one that is generally more favorable. This is because *major* updates tend to be buggy, while *minor* or *maintenance* updates after a *major* update would likely fix the bugs that occurred in the *major* release, leading to higher user satisfaction. The reason why *minor* performs better than *maintenance* (*i.e.*, 0.656 vs 0.216) is that a *minor* update

	Latent Topic #385	Latent Topic #47	Observed Topic "Medical Genre"
Top Terms	retina: 0.065947 display: 0.048744 support: 0.046697 graphic: 0.034819 resolut: 0.029084 full: 0.026627 #minor_update: 0.024579 touch: 0.024579 fix: 0.020074 ipad: 0.020074 #utilities_genre: 0.019664 #productivity_genre: 0.018435 high: 0.017207 optim: 0.016387 auto: 0.015159	map: 0.052109 #navigation_genre: 0.043457 traffic: 0.030958 rout: 0.023651 improv: 0.023267 locat: 0.023011 trip: 0.019485 #travel_genre: 0.019421 road: 0.016857 crash: 0.014935 address: 0.014294 poi: 0.013204 fuel: 0.011986 auto: 0.011794 alert: 0.011602	#medical_genre: 0.072711 #health_&_fitness_genre: 0.056700 pain: 0.032384 report: 0.026726 medic: 0.026605 pregnanc: 0.022512 graph: 0.018781 period: 0.015290 health: 0.015169 inform: 0.014326 track: 0.012521 diari: 0.012400 chart: 0.011798 drug: 0.011076 calcul: 0.010595

Figure 10: Three most important topics. Each topic shows the top terms, with the inclusive of hashtags. Terms in red are injected terms from genre labels; those in blue, injected terms from version information. Not only does this identify latent topics associated with app updates, it also gives a general overview of the kinds of features found in various version-categories.

Standard Version Categories		
#major	#minor	#maintenance

Advanced Version Categories		
#major	#minor	#maintenance
#minor_after_major	#maintenance_after_major	
#maintenance_after_minor	#major_after_minor	
#major_after_maintenance	#minor_after_maintenance	

Figure 11: List of standard and advanced hashtags for corpus-injection.

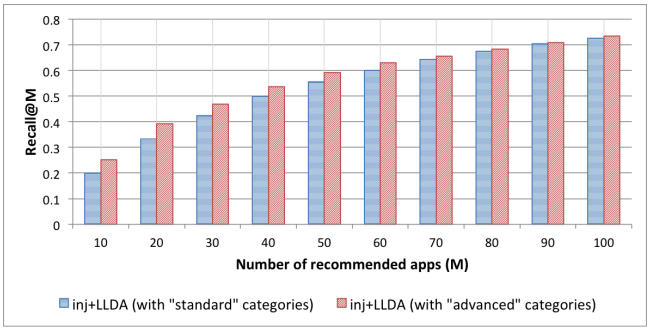


Figure 12: Recall scores between the use of "standard" and "advanced" version-categories.

typically introduces important bug fixes or functionalities, which is more appreciable than a *maintenance* update that resolves trivial issues of the app.

As version-categories are valuable features, we hypothesize that the recommendation accuracy can be improved if we further augment the version-categories. More specifically, as we previously only considered three standard version-categories: #major, #minor, and #maintenance, we consider improving the recommendation performance by injecting a more comprehensive list of version-categories into the corpus (as in Figure 11). Figure 12 shows the comparison between the standard and such an advanced set of version-categories (both models using inj+LLDA). Incorporating the advanced version-categories improves recommendation accuracy, as instead of identifying only 3 standard version-categories, we can discriminate among 6 additional scenarios. The additional details and specifications given by advanced version-categories effectively improve recommendation accuracy. We observe that advanced version-category model outperforms the standard model, particularly at the lower (more important) app recommendation ranks ("M"), although not statistically significantly so.

A more comprehensive modeling of version may be promising, and as such, since there is evidence that the sequence of versions would help, we plan to model the sequence of versions in future work.

7. CONCLUSION

In this paper, we leverage the unique properties in the app domain and explored the effectiveness of using version features in app recommendation. Our framework utilizes a semi-supervised variant of LDA that accounts for both text and metadata to characterize version features into a set of latent topics. We used genre information to discriminate the topic distributions and obtained a recommendation score for an app's version for a target user. We also showed how to combine our method with existing recommendation techniques. Experimental results show that genre is a key factor in discriminating the topic distribution while pseudo-terms based on version metadata are supplementary. We observed that a hybrid recommender system that incorporates our version-sensitive model statistically outperforms a state-of-the-art collaborative filtering system. This shows that the use of version features complements conventional recommendation techniques that treat apps as static items. We also performed a micro-analysis to show how our method targets particular versions of apps, allowing previously disfavored apps to be recommended.

In our future work, we plan to investigate the use of a variant of the tf-idf scheme to further vary the weights of the latent topics, allowing us to reward less common topics. We also plan to investigate more advanced techniques such as treating versions as inter-dependent and using a decaying exponential approach to model how versions are built upon one another in sequence.

8. ACKNOWLEDGEMENTS

This research is supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

9. REFERENCES

[1] D. Agarwal and B.-C. Chen. fLDA: Matrix Factorization through Latent Dirichlet Allocation. In *Proc. of the 3rd ACM International Conference on Web Search and Data Mining (WSDM'10)*, pages 91–100, 2010.

[2] U. Bhandari, K. Sugiyama, A. Datta, and R. Jindal. Serendipitous Recommendation for Mobile Apps Using Item-Item Similarity Graph. In *Proc. of the 9th Asia Information Retrieval Societies Conference (AIRS'13)*, pages 440–451, 2013.

[3] D. M. Blei and J. D. Lafferty. Topic Models. *Text mining: Classification, Clustering, and Applications*, 10:71, 2009.

[4] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[5] J. S. Breese, D. Heckerman, and C. Kadie. Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proc. of the 14th Conference on Uncertainty in Artificial Intelligence (UAI '98)*, pages 43–52, 1998.

[6] E. Costa-Montenegro, A. B. Barragáns-Martínez, and M. Rey-López. Which App? A Recommender System of Applications in Markets: Implementation of the Service for Monitoring Users' Interaction. *Expert Systems with Applications: An International Journal*, 39(10):pages 9367–9375, 2012.

[7] J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29:1189–1232, 2001.

[8] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl. An Algorithmic Framework for Performing Collaborative Filtering. In *Proc. of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99)*, pages 230–237, 1999.

[9] T. Hofmann. Latent Semantic Models for Collaborative Filtering. *ACM Transactions on Information Systems (TOIS)*, 22(1):89–115, 2004.

[10] T. Hofmann and J. Puzicha. Latent Class Models for Collaborative Filtering. In *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 688–693, 1999.

[11] A. Keyaki, J. Miyazaki, K. Hatano, G. Yamamoto, T. Taketomi, and H. Kato. Fast and Incremental Indexing in Effective and Efficient XML Element Retrieval Systems. In *Proc. of the 14th International Conference on Information Integration and Web-based Applications & Services (iiWAS'12)*, pages 157–166, 2012.

[12] Y. Koren and R. Bell. Advances in Collaborative Filtering. *Recommender Systems Handbook*, pages 145–186, 2011.

[13] Y. Koren, R. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. *IEEE Computer*, 42:30–37, 2009.

[14] H. Liang, Y. Xu, D. Tjondronegoro, and P. Christen. Time-aware Topic Recommendation based on Micro-blogs. In *Proc. of the 21st ACM International Conference on Information and Knowledge Management (CIKM'12)*, pages 1657–1661, 2012.

[15] J. Lin, K. Sugiyama, M.-Y. Kan, and T.-S. Chua. Addressing Cold-Start in App Recommendation: Latent User Models Constructed from Twitter Followers. In *Proc. of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'13)*, pages 283–292, 2013.

[16] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King. Recommender Systems with Social Regularization. In *Proc. of the 4th ACM International Conference on Web Search and Data Mining (WSDM'11)*, pages 287–296, 2011.

[17] Y. Moshfeghi, B. Piwowarski, and J. M. Jose. Handling Data Sparsity in Collaborative Filtering using Emotion and Semantic-based Features. In *Proc. of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'11)*, pages 625–634, 2011.

[18] C. Nascimento, A. H. F. Laender, A. S. da Silva, and M. A. Gonçalves. A Source Independent Framework for Research Paper Recommendation. In *Proc. of the 11th ACM/IEEE Joint Conference on Digital Libraries (JCDL'11)*, pages 297–306, 2011.

[19] D. Ramage, S. Dumais, and D. Liebling. Characterizing Microblogs with Topic Models. In *Proc. of the 14th International AAAI Conference on Weblogs and Social Media (ICWSM'10)*, pages 130–137, 2010.

[20] D. Ramage, D. Hall, R. Nallapati, and C. D. Manning. Labeled LDA: A Supervised Topic Model for Credit Attribution in Multi-labeled Corpora. In *Proc. of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP'09)*, pages 248–256, 2009.

[21] D. Ramage, C. D. Manning, and S. Dumais. Partially Labeled Topic Models for Interpretable Text Mining. In *Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11)*, pages 457–465, 2011.

[22] R. Salakhutdinov and A. Mnih. Bayesian Probabilistic Matrix Factorization using Markov Chain Monte Carlo. In *Proc. of the 25th International Conference on Machine Learning (ICML'08)*, pages 880–887, 2008.

[23] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based Collaborative Filtering Recommendation Algorithms. In *Proc. of the 10th International Conference on World Wide Web (WWW'01)*, pages 285–295, 2001.

[24] K. Sugiyama and M.-Y. Kan. Exploiting Potential Citation Papers in Scholarly Paper Recommendation. In *Proc. of the 13th ACM/IEEE Joint Conference on Digital Libraries (JCDL'13)*, pages 153–162, 2013.

[25] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Matrix Factorization and Neighbor based Algorithms for the Netflix Prize Problem. In *Proc. of the 2nd ACM Conference on Recommender Systems (RecSys'08)*, pages 267–274, 2008.

[26] C. Wang and D. M. Blei. Collaborative Topic Modeling for Recommending Scientific Articles. In *Proc. of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'11)*, pages 448–456, 2011.

[27] J. Wang and Y. Zhang. Opportunity Model for e-Commerce Recommendation: Right Product; Right Time. In *Proc. of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'13)*, pages 303–312, 2013.

[28] C. Wei and P. Seung-Taek. Personalized Recommendation on Dynamic Content Using Predictive Bilinear Models. In *Proc. of the 18th International World Wide Web Conference (WWW'09)*, pages 691–700, 2009.

[29] X. Wei and W. B. Croft. LDA-based Document Models for Ad-hoc Retrieval. In *Proc. of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'06)*, pages 178–185, 2006.

[30] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman. Identifying Diverse Usage Behaviors of Smartphone Apps. In *Proc. of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference (IMC'11)*, pages 329–344, 2011.

[31] B. Yan and G. Chen. AppJoy: Personalized Mobile Application Discovery. In *Proc. of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys '11)*, pages 113–126, 2011.

[32] P. Yin, P. Luo, W.-C. Lee, and M. Wang. App Recommendation: A Contest between Satisfaction and Temptation. In *Proc. of the 6th International Conference on Web Search and Data Mining (WSDM'13)*, pages 395–404, 2013.

[33] P. Yin, P. Luo, M. Wang, and W.-C. Lee. A Straw Shows Which Way the Wind Blows: Ranking Potentially Popular Items from Early Votes. In *Proc. of the 5th International Conference on Web Search and Data Mining (WSDM'12)*, pages 623–632, 2012.

[34] V. W. Zheng, B. Cao, Y. Zheng, X. Xie, and Q. Yang. Collaborative Filtering Meets Mobile Recommendation: A User-Centered Approach. In *Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI'10)*, pages 236–241, 2010.