# Title

New and Improved: Modeling Versions to Improve App Recommendation

# Citation

Jovian Lin, Kazunari Sugiyama, Min-Yen Kan, Tat-Seng Chua

# Abstract

Focusing on the app domain, author proposes a framework that leverages on version features instead of modeling item as static. Furthermore, author shows how to integrate this framework with existing recommendation techniques.

# Issues

Existing recommender systems usually model items as static. However, in domains such as mobile apps, a version update may provide substantial changes to an app as updates, may attract a consumer’s interest for a previously unappealing version. Version descriptions constitute an important recommendation evidence source for a recommendation. Are version features really so important in app recommendation? How could we utilize the version features in app recommendation?

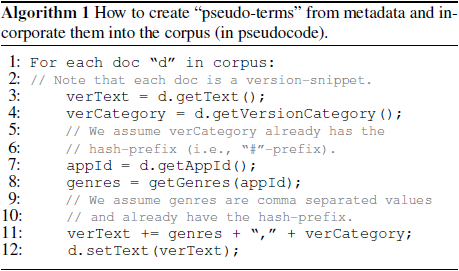
# Approach

First, generate latent topics from version features using semi-supervised topic models in order to characterize each version. Next, author discriminates the topics based on genre metadata and identify important topics based on a customized popularity score. Following that, author incorporates user personalization, and then compute a personalized score for a target user with respect to an app and its version. This system then recommends the top k target apps:

, (1)

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

## Generating Latent Topics



First, incorporate version metadata directly into the corpus to improve the quality of the topic distribution discovered by the topic models. Algorithm 1 shows how metadata in the form of pseudo-terms are automatically “injected” into the corpus of version-snippets, to associate these pseudo-terms with the latent topics. Then, model version-snippets with LLDA topic model. In this work, author treats 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.

But how to make LLDA semi-supervised? To achieve a semi-supervised topic model, author first assign every document with labels named “Topic 1” through “Topic K” for the unsupervised portion, and then use the observed labels (that are unique to each document) for the supervised portion.

LLDA 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:

, (2)

, (3)

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

Author uses “inj+LLDA” to denote this approach.

## Identifying Important Latent Topics

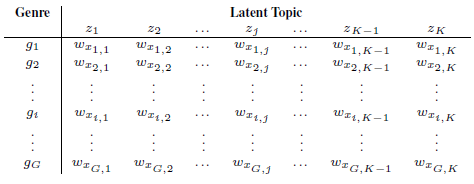


Table 1: Genre-topic weighting matrix, where g and z denote a genre and a latent topic, respectively. Every genre-topic pair has a unique weight from weighting scheme.

Apps are classified into various genres; each genre works differently to the same type of version update. Tabel 1 shows how author uniquely weight every genre-topic pair with multiplicative weight , where .

In this work, author defines 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.

Author formally defines the popularity score that outputs a value between 0 and 1, which factors user ratings into account:

, (4)

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

Author uses this popularity score to define the importance weight of a genre-topic pair, :

, (5)

where is the set of all topics, is the set of all documents that belongs to genre , is the popularity score of document , is the document-to-topic distribution in Equation (3), and .

## User Personalization

Author determines 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, author defines the following equation:

, (6)

where is the document-to-topic distribution defined in Equation (3) and is the set of documents consumed by user . As in Equation (5), the denominator is solely for normalization.

## Calculation of the Version-snippet Score

Finally, calculate the score defined by Equation (1) as follows:

, (7)

where , , and are the document, target user, and latent topic, respectively, denotes the weighting schemes (where ), is the genre of document , is the document-to-topic distribution in Equation (3), and is the probability that the target user prefers topic .

## Combining Version Features with Other Recommendation Techniques

Author uses Gradient Tree Boosting (GTB), which is a machine learning technique for regression problems that produces a prediction model in the form of an ensemble of prediction models, to integrate version features with the other recommendation techniques.

To produce a GTB model, author feed it training data, which contain the prediction scores of the various recommendation techniques and actual rating value of the user for the particular app. For a produced GTB model, the input is a set of probability scores of each of the recommendation techniques, the output is a predicted score between 0 and 5.

# Conclusion

Focusing on the app domain and the effectiveness of version features in app recommendation, author’s 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. Author used genre information to discriminate the topic distributions and obtained a recommendation score for an app’s version for a target user. Besides, author also shows how to combine our method with existing recommendation techniques.

By experiments, results show that genre is indeed a key factor in discriminating the topic distribution, and the use of version features complements conventional recommendation techniques that treat apps as static items.

# My Idea

1. For an app, its current versions is strongly associate to its previous version and later version. So maybe we could consider this association (or the sequence of versions) into the model.
2. The latent topics, generated from the version snippet, are not all topics of current version, but the topics of added or modified features of current version.
3. The latent topics, generated from the version snippet, may be the topics of deleted features. One of methods is to do semantic analysis on document, then penalize the deleted feature.
4. For an app, if a certain version and the other version have the same topics which user interested in, the latest one should be recommended, since the latest version may have less bugs. One of methods is to weight the time of version.
5. When update an app, this approach has to calculate all versions of all apps, consuming costly. But I have no idea.
6. Serial number of Version may influence recommendation, which just I guess.