A Personalised Reader for Crowd Curated Content

Gabriella Kazai, Daoud Clarke, Iskander Yusof
Lumi
{gabs,daoud,iskander}@lumi.do

Matteo Venanzi
Lumi and University of Southampton
matteo@lumi.do

ABSTRACT

Personalised news recommender systems traditionally rely on content ingested from a select set of publishers and ask users to indicate their interests from a predefined list of topics. They then provide users a feed of news items for each of their topics. In this demo, we present a mobile app that automatically learns users' interests from their browsing or twitter history and provides them with a personalised feed of diverse, crowd curated content. The app also continuously learns from the users' interactions as they swipe to like or skip items recommended to them. In addition, users can discover trending stories and content liked by other users they follow. The crowd is thus formed of the users, who as a whole act as the curators of the content to be recommended.

Categories and Subject Descriptors

H.4.m [**Information Systems Applications**]: Miscellaneous

Keywords

Recommender system, crowd curation, social filter, mobile

1. INTRODUCTION

Online content is growing at an unprecedented rate, with millions of news stories, blogs, videos, and a wide range of publisher and user generated content being added every day. Users typically navigate this space with the help of search engines, via social media, or through services like RSS feeds, content aggregators or recommender systems. With the proliferation of smart phones, access to this content is shifting away from search engines to consuming content directly within apps. As a result, a number of mobile content apps have been developed, including well known commercial systems like Feedly, Prismatic or Pinterest[1], as well as research prototypes like Focal [2], PEN [1] and others, e.g., [4, 3]. However, these apps rely on users manually defining their topics of interest, based on which articles from selected publishers can be pushed to them.

[1] `feedly.com`, `getprismatic.com/news`, `uk.pinterest.com`

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author. Copyright is held by the owner/author(s).
RecSys'15, September 16–20, 2015, Vienna, Austria.
© 2015 ACM. ISBN 978-1-4503-3692-5/15/09
DOI: http://dx.doi.org/10.1145/2792838.2796552.

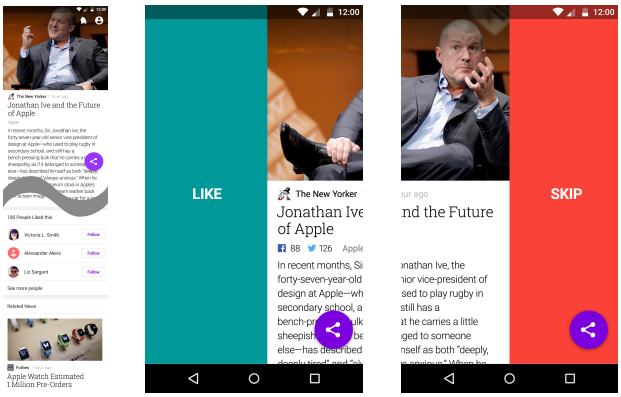

Figure 1: Full view of a recommended article and screenshots of user swipe actions to like or skip

Our system, called Lumi Social News[2], is not limited to specific publishers, but aims to provide users a personalised feed of diverse, crowd curated content, including long tail and user generated content. It automatically learns users' interests either from their public Twitter feed or from public pages in their browsing history. The generated user model is then matched against the stream of incoming content, consisting of public pages visited or tweeted by the crowd (the community of users) or ingested via RSS.

Instead of showing the user a list of recommendations that is typical in recommender systems, Lumi displays a single recommended item at any given time. This item is picked from a given time window of ingested content, where the selection is based on the item's relevance to the user as well as its popularity on social media, i.e., Twitter and Facebook. Figure 1 shows an example recommended article.

The header image is picked based on image quality and positioning in the original text, which is followed by the title and the full text of the article. In the case of a video, the video itself is positioned at the top, followed by the title and any textual description if available on the original site. The top three users who liked the recommended item are listed below the content, ordered by similarity to the user. Finally a list of related articles are shown, based on content similarity.

To get to the next recommendation, the user needs to either like or skip the current item by swiping left or right on the screen, respectively (see Figure 1). All liked items are

[2] `android.lumi.do`

saved by the system and can be accessed through the user's profile area. The liked/skipped actions are used to continually update the user's model, thus learning more about the user's evolving interests and filtering or boosting recommendations based on the user's feedback actions.

Lumi also supports a social network, where users can follow each other and discover interesting content that was liked by those they follow. Recommendations on who to follow are based on the relevance of the suggested user's liked items to the current user's interests as well as based on existing social links in the user's Twitter network.

2. SYSTEM OVERVIEW

The system uses a graph processing architecture, similar to that of Twitter's Storm, where each node is a process and data is passed around from node to node. This is driven by a need for real-time processing, as opposed to processing offered by Hadoop-like tools. The topology of a graph is specified dependent on its function. For example, a bootstrapping graph is designed for real-time processing of large-scale browsing or twitter data at signup in order to create a user model as quickly as possible. The front-end is implemented for Android phones in Java, which connects to the back-end via an API. The main system components are:

- *Ingestion*: Ingestion nodes process a number of incoming streams of content from RSS, Twitter and public pages from user visits. The content is rendered, passed through a quality filter and subsequently a range of features and media are extracted. An SVM classifier is used to assign a category label, e.g., business or technology. Named entities are extracted using multiple open-source tools, e.g., NLTK and OpenNLP[3]. Top ranking entities are assigned as topic tags. Data is stored in Cassandra[4] and Elasticsearch[5]. Note that for additional privacy, user's browsing data are stored completely anonymously and separately from their Twitter data, and no link is made between the two.
- *Bootstrapping*: At signup, the public pages in a new user's browsing history or Twitter feed are analysed and an initial user model is built. We build separate models for their browsing data and for their Twitter data. The only time the two sets of data can be connected is when the user is online and makes a request for recommendations, thus to maintain this level of privacy we need to generate separate models.
- *User model update*: Users' models are updated based on their ongoing online activities, e.g., Twitter feed, as well as based on their in-app actions, e.g., when they read, swipe to like or swipe to skip a recommended item, and when they share a recommendation.
- *Trending content*: Trending content is identified by monitoring social media, e.g., likes on Facebook and tweets on Twitter. In addition, we use clustering to detect breaking stories, i.e., when the same story is being published across multiple outlets.
- *Offline recommender*: Ingested content is passed through a Content Based (CB) and a Collaborative Filtering (CF) based recommender, which calculates relevance scores for each user model and stores their top recommendations. This is run as a background process which generates recommendations for users even when they are not online, so that they don't miss out on interesting content.
- *Online recommender*: When an Android client requests new recommendations, the top ranking items are returned from across a number of different sources, including the offline recommendations store, the latest trending stories as well as the freshest content ingested in the last few hours. The ranking function then takes into account both the relevance of an item, its freshness and its popularity.
- *Related content*: Same stories from multiple outlets are clustered together and can be served to the user as related items. Articles based on broader topical similarity are also linked together. Other articles from the same site may also be of interest to the user and may also be shown below the recommended article.
- *Suggested user to follow*: We recommend users to follow based on the relevance of the items they liked to the current user or based on existing links between them on Twitter.

At the time of writing, we are conducting online experiments with real users, using an A/B testing framework. Among others, we are experimenting with different recommender algorithms and trending story detection methods and different UI features. We are tracking the quality of our recommendations as reflected in likes, skips and reading times, averaged across users and plotted over time.

3. CONCLUSIONS

The presented system is the first in its class to provide personalised feeds by non-intrusively learning users' interests and responding to their feedback actions when reading recommended items. The assessment of how recommender systems may perform in this setting with noisy and sparse data and online user feedbacks is the key challenge to deliver this service.

4. ACKNOWLEDGMENTS

Lumi is a result of the efforts of a great team, see `https://lumi.do/about/team`.

5. REFERENCES

[1] F. Garcin and B. Faltings. Pen recsys: A personalized news recommender systems framework. NRS '13, pages 3–9. ACM, 2013.

[2] F. Garcin, F. Galle, and B. Faltings. Focal: A personalized mobile news reader. RecSys'14, pages 369–370. ACM, 2014.

[3] A. Said, J. Lin, A. Bellogín, and A. de Vries. A month in the life of a production news recommender system. In *Proc. Workshop on Living Labs for IR Evaluation*, pages 7–10. ACM, 2013.

[4] M. Tavakolifard, J. A. Gulla, K. C. Almeroth, J. E. Ingvaldesn, G. Nygreen, and E. Berg. Tailored news in the palm of your hand: A multi-perspective transparent approach to news recommendation. WWW'13 Companion, pages 305–308, 2013.

[3] `github.com/nltk, opennlp.apache.org/`
[4] `cassandra.apache.org/`
[5] `elastic.co/products/elasticsearch`