Computational Journalism: Assignment 2

Pankhuri Kumar (pk2569)

1. The filter that I'm designing is for the public that reads news (I'm assuming that I can borrow the New York Times recommendation engine, and can run it on any article, as well as any person reading the article, on my app) on the internet.

2. This algorithm will filter news articles based on user preferences, user network and article content. These news articles do not have to be constrained to New York Times articles. I also assume that my app makes it mandatory for the user to sign up with their Facebook or Twitter account to get better recommendations, or to see what their friends are reading.

3. The inputs to my algorithm will include:

- all Google News articles (https://newsapi.org/s/google-news-api), which covers a lot of the internet news sources.
- Considering that what I'm designing is an app of some form, I will have information about the clicks and reads by a user (exactly the same way that the NYT engine can distinguish between links opened and links read)
- If a user links their Facebook/Twitter accounts to the app, I would also have access to their social network graphs, which will be an input to the algorithm

4. One of the problems I have wondered about (since joining this program, and in the current divisive political climate) is how to bridge the gap between the two sides of any issue, be it political or not. People often envelope themselves in real-world bubbles by surrounding themselves with like-minded people. The only way they get exposure to other kinds of opinions is when one of the people in their like-minded circles brings a new suggestion (again, how this person came across a new opinion is also worth considering) or the person seeks out people with different opinions and engages with them.

My algorithm aims to do the same, in similar ways. One way is to introduce an element of *serendipity* into recommender systems (research here, here and here): the NYT engine seems to bring some element of this by using the *back-off* method. The other way would be to give the user a way to control the content they see on my app. As suggested in Prof. Stray's article, my app would provide the user with the option to see three different types of articles: "only things I like," "things I

might like but haven't discovered" (which relates to the serendipity element) and "things that are popular but I may not like" (which relates to people trying to engage outside of their like-minded circles).

To the user, these three options provide them with three different ways to engage with the app, and be exposed to different kinds of content. As a publisher and app creater, it could lead to people engaging more often with the app and reading more articles, which can be used to receive more funding to create the app, and be monetized through ads. From a social perspective, this app could help bridge the gap between people with opposite views on a variety of issues.

If a user wants to chooses to see "only things I like," the algorithm would be based off of their user history and recommend articles from only those topics that the user likes.

On *serendipity* mode, where the user has chosen "things I might like but haven't discovered," the app would take in two inputs – topics the user has opened articles from previously but hasn't read, and articles that the user's friends might be reading. Suggesting articles from topics that the user has shown *some* interest in, or articles that their friends have read, would give a good estimate of the boundaries and edges of a user's interests, and this can be used to "surprise" the user with recommendations. The friends considered here would have at least two degrees of separation from our user, from the assumption that most social media influence stems from weak contacts.

In the third mode, I follow Prof. Stray's advice from the previously linked article (and add some of my own research). One way would be to find topics that the user likes, and then present him with articles that the user (including all of his friends, or immediate network) has not read. These could potentially be opposing or neutral articles on the same issues. They could also be similar articles on the issue, but from different sources, which is why they might not show up on the user's network.

But this method could fail when topics do not have a clear divide along political or religious lines. In such cases, using NLP, a sentiment/tone/emotional analysis of the news articles could give us an idea of not just the topic, but also the opinion on the topic. Then, the user could be presented with articles from the same topic, but having different (or opposing) tones and sentiments.

5. This is what the algorithm would (roughly) look like:

a) Use LDA to model articles based on their content, to find different topics an article can belong to. This step of the algorithm would be run only on new articles that are added to app every time. Once a month, this would could be run on the entire corpus to check if new topics can be created from our new corpus of articles.

b) Extend LDA to all our users reading these articles, to find different topics that a user is interested in. This step of the algorithm would be run only on new users every time. Once a

month, the LDA would run on all users to re-categorize them based on changing preferences. Users will have the option to force this update immediately and reflect changes on their tastes. This will also incorporate the new topics that might have been found in the previous step.

c) Run and update Collaborative Topic Modelling (CTM) algorithm as used by New York Times, to adjust topic postions for articles and users. This would be run only for new articles and users, and once a month over the entire corpus of users and articles.

d) Using the click-data and read-data, create two different scores for each user: one using only those articles read by the user (read-score) and one using the *back-off* approach (back-off score), as described in the New York Times blogpost. This step of the algorithm will be run only on new users, and once a month for existing users. The app could provide the user with an option to force this update immediately.

e) Using Facebook/Twitter data, cluster users together based on whether they are friends/followers. A simple algorithm that iteratively clusters people who are connected by a direct link would create a map of all the various "echo chambers" on the social media. For Twitter, we would look at the people our user follows, as well as the people following them to create this graph. This step of the algorithm will be run only once a month, and updated accordingly. The app could provide the user with an option to force this update immediately.

f) Use Linguistic Inquiry and Word Count (LIWC) to find the different "emotions, thinking styles, social concerns" in each article, and store this information for each article, along with the topic scores. This step of the algorithm would be run only on new articles that are added to app every time.

g) If the user has selected "Only Things I Like," use the read-score to find topics a user is interested in. Then, make recommendations of articles from these topics that are not read by the user. Give priority to articles that match most closely with the maximum number of topics from the user's topic scores. If you need to make a choice, choose closer scores over more matching topics. Present this as "Your Favorites."

h) If the user has selected "Things I Might Like But Haven't Discovered," use the back-off score to find topics of the user's interests. Then suggest articles from these topics which have not been read by the user. Give preference to articles that belong to these topics, and match the most closely with the maximum number of topics from the user's topic scores. Present this list as "You Might Have Seen These Before." If you need to make a choice, choose closer scores over more matching topics. Also recommend articles from topics that the user's friends have read (use read-scores), but are not read by the user. Choose friends that are at least two direct connections away from the user. Present this as "Popular With Your Friends."

i) If the user has selected “Things that are Popular But I May Not Like,” use the read-scores to find a user’s topics, and then recommend articles from these topics that are not being read in the user’s network. This would require a simple case of finding all users that read articles from these topics, choosing users that do not belong to our user’s network, and suggesting articles from their read-list. Present this list as “From Outside Your Network.” Using topics from the read-score, also recommend articles from these topics that have opposing scores (hence, opposing emotions, thinking styles and social concerns) in the LIWC analysis. Present this list as “Different Views on Your Favorite Topics.”