Final Report: News Recommender System

Abhimanyu (MS17106), Akash (MS17073), Anshu (MS17098)
Sanat (MS17107), Ved (MS17117)

June 1, 2020

1 The Dataset

1.1 Strategy

1.1.1 Sources of Data:

- We have worked with two data sets.
  Initially, for the purpose of learning, exploring, and finalizing our strategy, we used a static dataset available on Harvard Dataverse (link: Harvard Dataset) consisting of 3824 old news articles from various sources.
  For use in the actual recommender system, we have a non-static data set that is scraped from the BBC website and can be updated daily. In order to save computation time in parsing and downloading articles, our analysis and visualization was initially done using the static data source.
  Both, the Harvard Dataset, as well as the latest scraped BBC dataset have been provided along with the code, which can be run using both of the datasets.

1.1.2 Online Scraping

- For the scraped dataset we decided to scrape from only a single source so as to avoid collecting multiple articles consisting of the same or similar news, as would be the case across multiple news sources.
- For this purpose we tried various sources like The Hindu, Times of India, Al Jazeera and BBC. We settled on BBC because of the ease of navigating the website and because the corpus collected was large and had sufficient variation.
- We scraped the website once everyday, so that the articles get updated, and removed duplicate articles. In this manner, we managed to get around 300 links everyday.

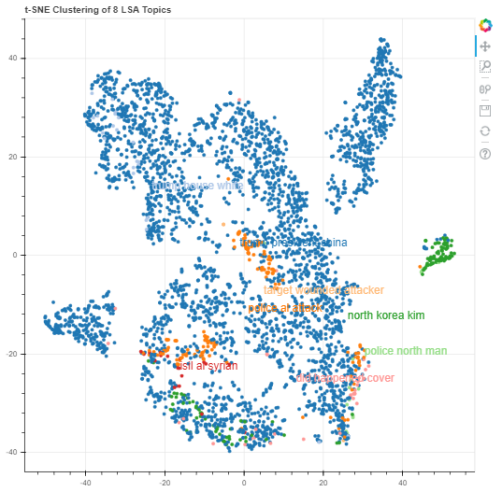
1.2 Challenges

- Removal of links with no text:
  There were some links with entirely visual content, and the ones which had congregation of many news articles rather than being an article itself, for example, headlines.
- Error in parsing some links:
  We also encountered errors while parsing some links, such articles were ignored to be included in corpus. Given the size of our data set (around 2000 articles), skipping few of them hardly changed anything.
- Scraping time too long:
  Since we had to parse, download a large number of Articles, we were restricted by the computation time, for this issue, we scraped articles everyday and stored them for further diagnosis.
- Different character encodings:
  Often certain news articles would have different special character encodings in the text that would lead to errors while running the code later on. These encodings had to be dealt with accordingly.

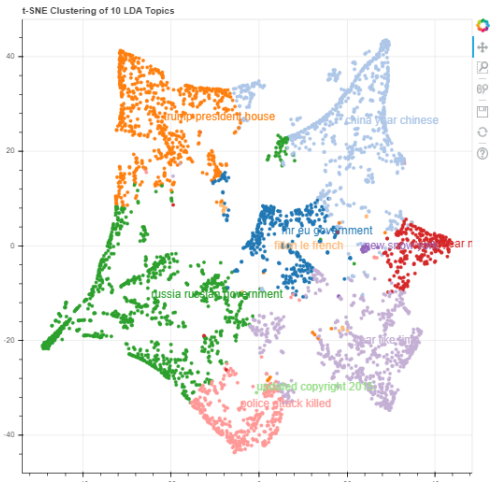
2 TOPIC MODELLING

2.1 Choice of algorithm: LSA vs LDA

We considered two algorithms for the purpose of topic modelling- Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA). More advanced techniques like word embedding were not considered due to computational limitations.
We had initially decided to use LSA for our model, however upon our initial exploration, it did not work as expected.
Given below are the results of LSA and LDA performed upon the Harvard Dataset, where the articles



(a) LSA on Havard dataset



(b) LDA on Havard dataset

have been classified according to topics, and visualization is done in two-dimensions with the help of tSNE dimensionality reduction[1].

Clearly, LSA gave a very poor result with no spatial separation of topics; in fact, almost all of the articles have been classified under one single topic.
On the other hand, LDA seemed to have given a much better result, with clear spatial separations of topics. This led us to decide that LDA was the appropriate algorithm to use. Later on, the BBC dataset also gave us a similar result for LDA vs LSA.

2.2 Implementation of LDA

Based on the above results, we settled upon LDA as our algorithm of choice. LDA is a generative probabilistic model of a corpus, where documents are represented as random mixtures over latent topics. We decided to set the number of latent topics as 10; since we were recommending 10 articles at a time, we wanted to be able to have 10 distinct categories from which we could select articles
Now, we had two choices regarding how to implement LDA- either using the gensim library or the LDA package in the sklearn library. Upon trying both, are findings were:

Using sklearn:

- It was very easy to generate the LDA topic document matrix which was the representation of all documents in the new vector space.
- We however ran into code errors sometimes while trying to extract the important words that each topic consisted of

Using gensim:

- We could build an LDA model in gensim, but we encountered difficulty in generating a topic matrix that had the representations of all the documents, like we had in sklearn
- gensim however did have the support for a lot of advanced visualization tools, like the interactive pyLDAvis package.

We ended up deciding on the LatentDirichletAllocation package of sklearn due to the ease provided by the LDA Topic document matrix in classifying documents and calculating cosine similarity across the whole dataset, on the basis of which recommendations were made. User profiles could also be easily defined on the same space.
For data visualization purposes we used tSNE plot which easily worked with the LDA topic document matrix (image shown earlier).
Given below is the histogram of the distribution of stories according to topics, following the implementation of LDA, on our latest scraped BBC dataset.

We see that apart from one category, all the topics have a healthy number of articles present in them (this would improve as more stories are added to the corpus).

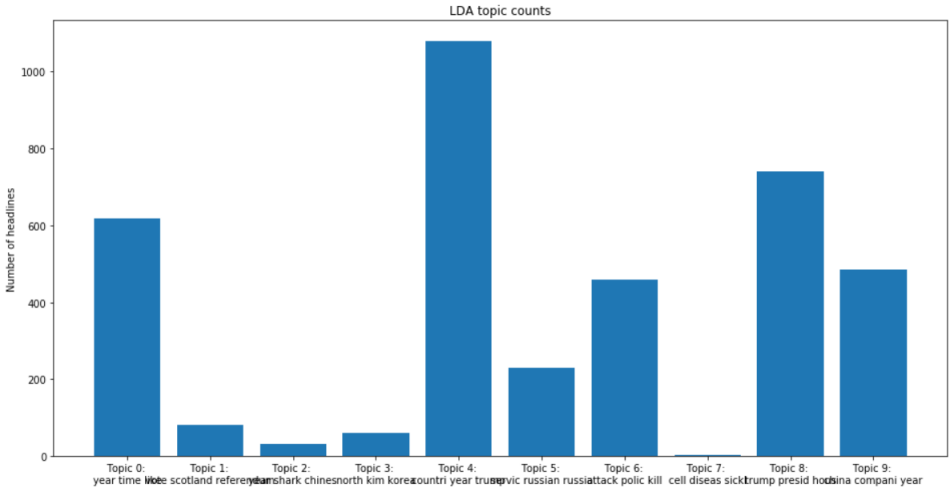


Figure 2: Topic-wise distribution of stories in the BBC dataset

3 Clickstream Generation

3.1 Strategy

The clcikstream generation can be divided into the following 3 parts:

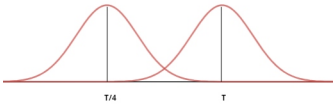
- **Document assignment**
  A new user (in their first session) is presented with 10 articles, each selected randomly from every topic generated from LDA. This is done to maximise the coverage of the corpus as well has to have diversity in the articles presented, so as to learn the user preference. A recurring user is allotted the articles as described within the 'recommendation strategy' section.

- **Probability of Clickthrough**
  A new user has a 1:1 probability of clicking on an article. However, for the regular users, we start incorporating a **"User Preference"** from the next session onwards. This is done to visualize the results because if everything remained completely random, we would see no trend developing over sessions. We can bias the user by manipulating either the clickthrough probability or the amount of time spent reading an article. We went with the former by increasing the probability of a clickthrough to 0.75 if the article presented has a cosine similarity greater than 0.7 with the user vector.

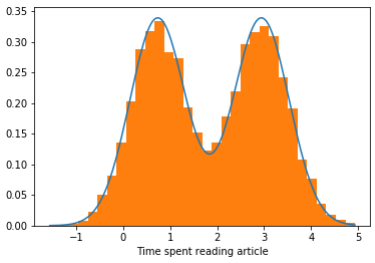
- **Time spent per article**
  We create this distribution using a 2 component Gaussian mixture model (fig). We interpret it as follows: The first mean $T/4$ corresponds to the time taken by someone to skim through, whereas, the second peak with mean at T refers to the people who went through the article properly in its entirety. T, here, is the average time required to read the given article (in minutes). Since according to research, the average reading speed of an adult is estimated to be around 250 words per minute, the average time required to read an article is given by:

$$T = Length\ of\ articles\ (in\ words)/250 \tag{1}$$

  We assigned equal weight to both the components of the mixture model, assuming that out of the users who clickthrough, half would skim, and half would read the article properly. Each component was assigned a variance of $T/5$ based on trial and guess.



(a) Proposed Gaussian Mixture



(b) Values produced by our function plotted against the expected curve

- For the purpose of generating results, we chose to model number of new users per session as a uniform linear model where we get 10 new users per session. Visualization: We create 2 session analysis tables for each iteration, one each for the new and the regular users.

SESSION 2

Average time spent by a new user in this session 2.192016974307482

Average number of articles read by a new user in this session 4.4

	New User ID	Session ID	Total time spent	Articles read	Expected category
0	0	2	0.498594	4	Non regular
1	1	2	0.416859	3	Non regular
2	2	2	2.556340	6	Regular
3	3	2	1.652561	5	Non regular
4	4	2	0.686642	5	Non regular
5	5	2	3.913285	5	Regular
6	6	2	5.501605	5	Regular
7	7	2	0.412268	3	Non regular
8	8	2	0.096126	3	Non regular
9	9	2	6.185890	5	Regular

Average time spent by a regular user in this session 2.4141691647665193

Average number of articles read by a regular user in this session 5.0

	User ID	Session ID	Total time spent	Number of articles read
0	0	2	3.689172	4
1	1	2	0.116980	5
2	2	2	5.628209	6
3	3	2	0.222316	5

Number of regular users before this session 4

Number of regular users after this session 8

Figure 4: Session analysis outputs

4 User Profiling

- **User Profile**
  We make a unique profile for each user based on the articles they read as well as the time spent while reading each article. We make the simplified assumption that more time spent equals greater interest in the article. This is updated after every session to keep the entire history of a regular user.

	User ID	Session ID	Document ID	Topic	click	time_spent
0	0	1	716.0	0.0	0	0.000000
1	0	1	718.0	1.0	1	0.406122
2	0	1	948.0	2.0	1	1.085275
3	0	1	1012.0	3.0	0	0.000000
4	0	1	813.0	4.0	1	0.065773
5	0	1	1.0	5.0	0	0.000000
6	0	1	283.0	6.0	1	0.033246
7	0	1	973.0	7.0	1	0.647199
8	0	1	508.0	8.0	0	0.000000
9	0	1	841.0	9.0	0	0.000000
0	0	2	366.0	2.0	1	0.920231
1	0	2	403.0	2.0	1	2.280550
2	0	2	185.0	2.0	1	0.012264
3	0	2	757.0	2.0	0	0.000000
4	0	2	473.0	2.0	1	0.027848
5	0	2	1040.0	9.0	0	0.000000
6	0	2	561.0	8.0	0	0.000000
7	0	2	174.0	4.0	0	0.000000
8	0	2	695.0	2.0	0	0.000000
9	0	2	441.0	7.0	0	0.000000

Figure 5: User profile of User ID 0 after 2 sessions

- **Regular Users**
  If the user has spent very little time reading, we regard them as a "browser" and don't take them into account while creating a "Regular Users" repository. We assume that once a new user is a regular user, they'll remain so throughout.

- **User Vector**
  We then take the weighted average of the articles read by the user with respect to the time spent reading them. This vector is created for each regular user after first session and updated after every new session.

$$user\_vector = \sum_{i=1}^{10} \frac{t_i}{T} * LDA\_array_i$$

  Where
  i is the articles read by the user for that session
  t_i is time spent reading article i
  T is total time spent by the user in the session
  LDA_array_i is the 10 component array that gives us a probability distribution of the i^{th} article in the 10 topics.
  Thus we can represent the user in the same space as the documents.

5 Recommendation Strategy

- **First Visit:**
  In the first visit we don't have a profile for a user and thus we recommend one random article from each topic, so that there is no bias, corpus coverage is maximized, and user preferences can be learnt.

- **Subsequent Visits:**
  In the subsequent sessions, once we have the user vector, we can begin recommending articles based on their preferences (given below).

- **Similarity Coefficient:**
  Since the user vector is in the document-topic space, we can find documents which are similar to the user vector. To find this similarity coefficient we use cosine similarity.

$$S_{cos} = \frac{User\_Vector * LDA\_Matrix}{\sqrt{\|User\_Vector\| \times \|LDA\_Matrix\|}}$$

- **Recommending Similar articles:**We then select the 5 documents that have the highest similarity coefficient. The user is much likelier to read these articles, since we are exploiting their preference.

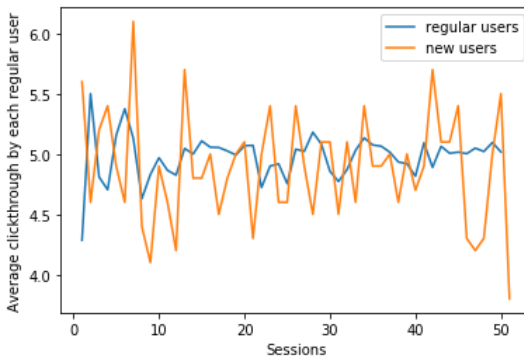
- **Reducing bias:**
  However, we also recommend articles chosen randomly from each of the topics not presented to the user in the top 5 articles. This helps us reduce the bias in user recommendation, as well as continue to explore user preference.

	Document ID	Similarity	Links	Topic
744	744	0.990929	789	4
1030	1030	0.990929	1093	4
14	14	0.990929	15	4
585	585	0.990929	623	4
630	630	0.990929	670	4
171	171	0.990929	180	4
460	460	0.990929	488	4
869	869	0.990929	924	4
914	914	0.990929	971	4
298	298	0.990929	318	4

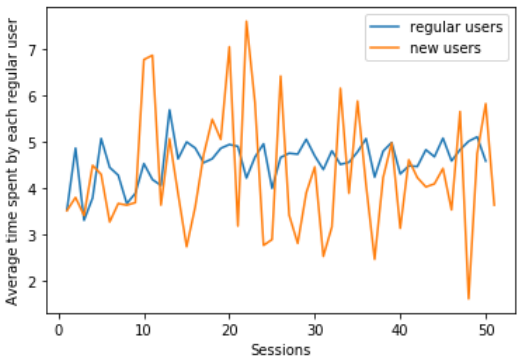
Figure 6: Similarity Coefficient DataFrame of a user

6 Results

- We ran the code for 50 sessions and plotted the average clicks per user over sessions and the average time spent by a user over sessions. We see that the trend isn't as unpredictable for the regular users. There is a steady increase, though not a considerable one. The factors that might have affected this include:
  - Small user count (10 new per session)
  - The user preference created in the clickstream generation might be too mellow
  - The number of sessions isn't optimal. Though we were highly limited by our computational accessibility in this aspect
- **Increase in average time per session** Average time spent by a regular user (personalized recommendation) per session is 4.5877217352 mins while the average time spent by a new user(completely randomized recommendation) is 4.287999501960114 mins
- **Increase in average user clickthrough per session** Average articles clicked by a regular user (personalized recommendation) session is 4.977440232588699 while the average time spent by a new user(completely randomized recommendation) is 4.686274509803921



(a) Average clickthrough over sessions



(b) Average time spent over sessions

7 Future Goals

- Try Topic optimization using Hierarchical Dirichlet Process (HDP)
- Explore *Gensim* further to get the same LDA matrix and use the advanced visualization techniques available
- User profiling in further subsequent visits
- Better weights assignment
- Attempt to incorporate Collaborative Filtering
- We could schedule the various parts of the code, for example, scraping the news sites automatically after a certain interval.

References

[1] LDA vs LSA

[2] tSNE

[3] Dataset from Harvard dataverse