NEWS RECOMMENDER SYSTEM

proud history and a bright future..
DEAR EU, IT'S TIME TO GO
The Sun
DOVER & OUT
The Daily Telegraph
Unite behind Brexit, says May
After our Legs-it storm SARAH VINE tells her critics: Oh, DO get a life
Daily Mail
CENSORED BY THE LEFT
FREEDOM!
theguardian
Today Britain steps into the unknown
Brexit begins
THE TIMES
The eyes of history are watching
May triggers Britain's EU departure
DAILY EXPRESS
OUTRAGEOUS! WARTIME LAND
Getty Images

Akshay(MS15185), Sumith(MS15171), Rutik(MS15086), Adhil(MS17150), Vaishnav(MS16047)

01.06.2020

INTRODUCTION

Reading the news online has exploded as the web provides access to millions of news sources from around the world. The sheer volume of articles can be overwhelming to readers. Therefore, building a news recommendation system to help users find articles that are most interesting to them is a crucial task for every online news service.

News recommendation systems must be able to handle the challenge of fresh content: breaking news that hasn't yet been viewed by many readers. We tackle the problem of building the recommender system in 4 steps:

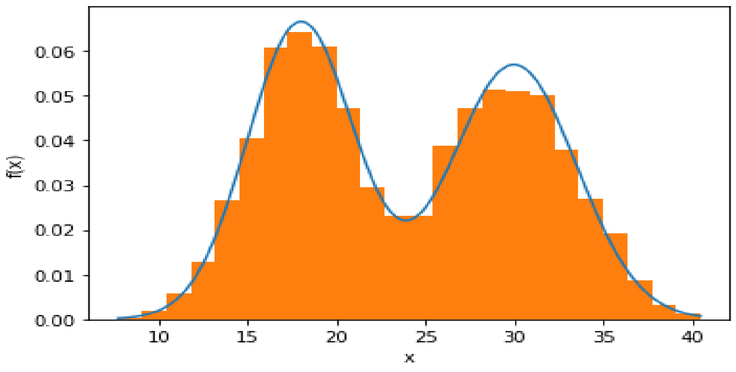
- Creating the corpus of news articles by scraping from web
- Creating a topic model using LDA or TF-IDF vectorizer
- Incorporate topic model to build content based recommender for news articles
- Generate a user profile using multi-normal distributions corresponding to different user features and then perform collaborative filtering to recommend articles liked by similar users.

Initially, before having a user base, the news recommender suffers from cold-start, due to unavailability of user preferences or article features. Since the real estate of the mobile device through which we plan to broadcast news recommendations is limited to 10 articles, we propose to use a topic distribution based on a study conducted on habits of online news readers [1]. This would let us choose specific articles to cater for the individual.

GENERATING USER PROFILE

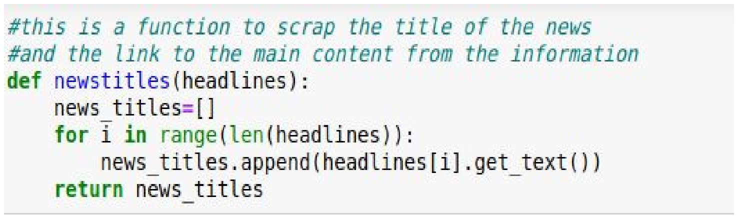
As we don't have access to any user preference or user behavior data, it is important to sample our own probability distribution that might capture the dynamics of a real-world user. For this we use a mixture of gaussians. The target audience to which we cater are working professionals who fall in the age range of 21-40. We note that time is an important resource for this target group, which can be analysed on the basis of age. Moreover, we have real life examples of successful news apps like InShorts, which delivers news in less than 60 words, and thus exploit time as a resource to retain their user base. These facts have motivated us to choose time spent by a user on a selected article as a honest indicator of the relevance of an article. Dividing the user base to people aged below 30 and above 30, we can make a set of assumptions:

- People aged below 30 are more connected through social media platforms and have a greater overlap with mobile technology. The analysis of ratings for the InShorts app in Google Playstore suggest that people in this age range love the app as they only need to spend very less time on an article. This is also supported by a study by Tim Groot *et al.*[2]. Hence, we assign lesser time to this user group, based on a normal distribution.
- People aged above 30 are more connected to traditional media like newspapers and magazines, as they are from a pre-mobile era. This has naturally affected their reading time, and we assume that this user group takes more time to complete an article, which is an honest indicator of the quality of the article. The difference however, is only a few seconds to avoid any bias.

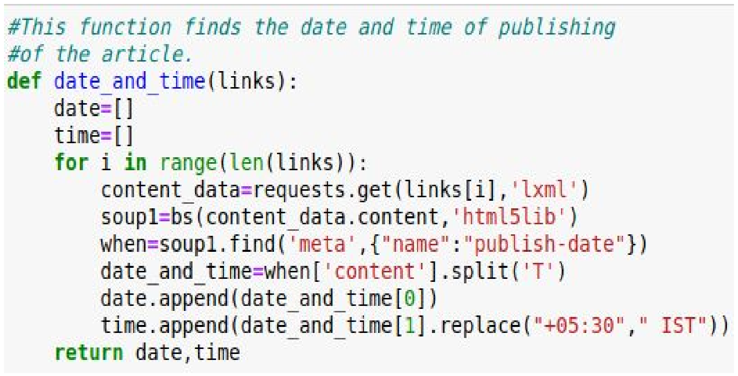


The adjoint figure shows the probability distribution from which time spent on articles by different user groups is sampled. This measure is used as a substitute for rating in the recommender. This distribution is used to create a sparse matrix of user-article relation, as given below:

```
A1 = sp.random(n_users, n_articles, density=0.01, random_state = 311, \
               data_rvs = ss.norm(*(norm_params[0])).rvs)
B = sp.random(n_users, n_articles, density=0.01, random_state = 311, \
              data_rvs = ss.norm(*(norm_params[1])).rvs)
A = 0.5*A1+0.5*B
```



```
#this is a function to scrap the title of the news
#and the link to the main content from the information
def newstitles(headlines):
    news_titles=[]
    for i in range(len(headlines)):
        news_titles.append(headlines[i].get_text())
    return news_titles
```



```
#This function finds the date and time of publishing
#of the article.
def date_and_time(links):
    date=[]
    time=[]
    for i in range(len(links)):
        content_data=requests.get(links[i],'lxml')
        soup1=bs(content_data.content,'html5lib')
        when=soup1.find('meta',{"name":"publish-date"})
        date_and_time=when['content'].split('T')
        date.append(date_and_time[0])
        time.append(date_and_time[1].replace("+05:30"," IST"))
    return date,time
```

BUILDING THE CORPUS : SCRAPING ARTICLES

We scrapped articles mostly from the Hindu and the IB Times. As the current situation on COVID-19 may bias our corpus towards such instances of news, we mixed the dataset with older news so as to get more topics. The modules BeautifulSoup4 and Newspaper3k were used for scraping articles. Since for a good recommender, we need a large enough corpus, we scrapped about 6000 articles. We extracted the title, content, date and time of publishing and links to the news as a part of scraping the article.

IMPLEMENTING A CONTENT BASED RECOMMENDER SYSTEM

Once we have scrapped enough news articles, we proceed to build the recommender system based on LDA (Latent Dirichlet Allocation). LDA is a type of unsupervised learning algorithm in which topics are inferred from a dictionary of text corpora whose structures are not known (are latent). Using LDA, we can generate a weighted sum of words which would correspond to a topic. The number of topics is a parameter for the LDA model which is fixed using standard scoring metrics like perplexity and coherency score. Based on our analysis, we find a peak on the coherency score when the number of topics is 17. Before applying LDA, we split the data into training and testing datasets.

```
#We now split the pre-processed corpus into training and testing data sets
random.seed(75) # For reproducibility
train_indices = random.sample(list(range(0,len(texts))),len(texts)-1000)
test_indices = [x for x in list(range(0,len(texts))) if x not in train_indices]

train_data = [texts[i] for i in train_indices]
test_data = [texts[i] for i in test_indices]
```

```
dictionary = gensim.corpora.Dictionary(train_data)
doc_term_matrix = [dictionary.doc2bow(doc) for doc in train_data]
```

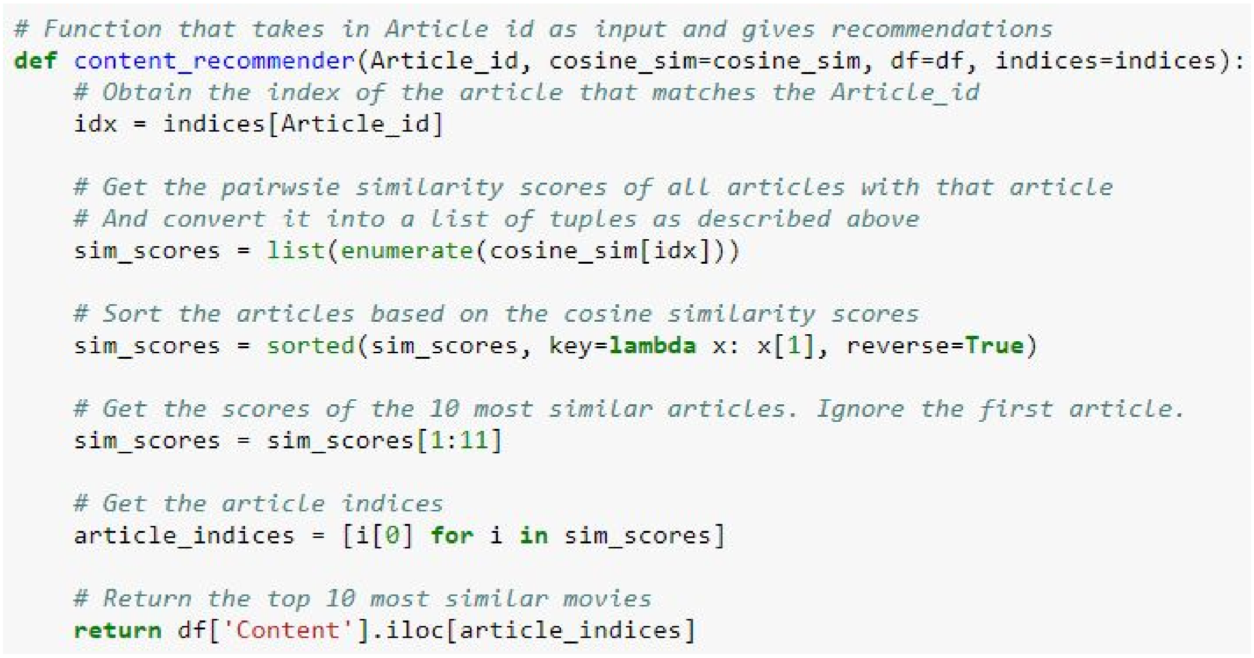
```
Lda = gensim.models.ldamodel.LdaModel
ldamodel = Lda(doc_term_matrix, num_topics=17, id2word = dictionary, passes=50)
```

From this , we can calculate various topics that are relevant in an article

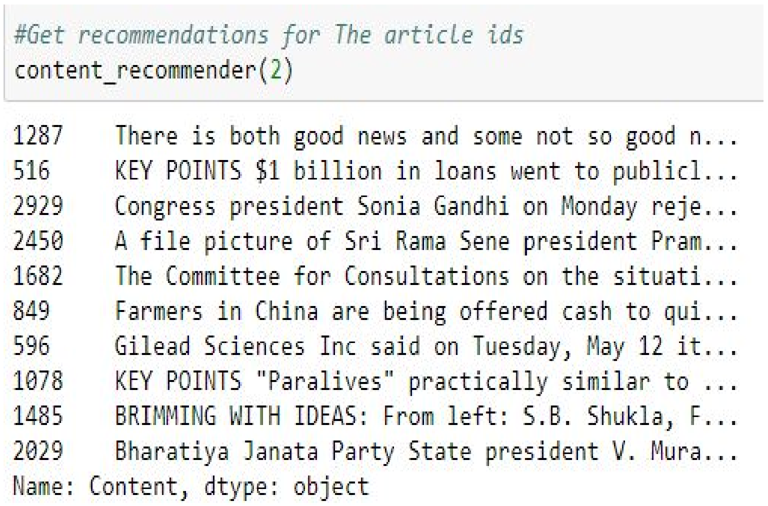
```
corpus_lda_model = ldamodel[doc_term_matrix]
for article in corpus_lda_model[5]:
    print(article)
```

```
(4, 0.6090433)
(14, 0.20145865)
(16, 0.18348165)
```

This shows that a specific article in the corpus is made of topics 4, 14, and 16 with corresponding probabilities. Now, the model can be used for content based recommendation. We also performed TF-IDF vectorization on the articles and generated a corresponding content based recommendation system. Cosine similarity is used as the metric to evaluate similarity between articles. The following function is used to generate content based recommendation, depending on an input article_id. The function compares the contents of input article_id to find similar articles.



```
# Function that takes in Article id as input and gives recommendations
def content_recommender(Article_id, cosine_sim=cosine_sim, df=df, indices=indices):
    # Obtain the index of the article that matches the Article_id
    idx = indices[Article_id]

    # Get the pairwsie similarity scores of all articles with that article
    # And convert it into a list of tuples as described above
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the articles based on the cosine similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar articles. Ignore the first article.
    sim_scores = sim_scores[1:11]

    # Get the article indices
    article_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return df['Content'].iloc[article_indices]
```



```
#Get recommendations for The article ids
content_recommender(2)
```

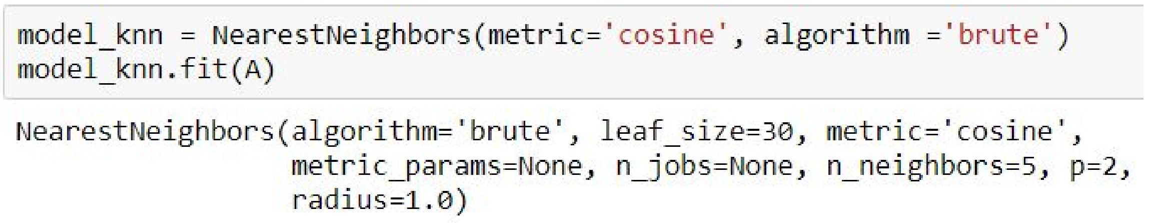
```
1287    There is both good news and some not so good n...
516     KEY POINTS $1 billion in loans went to publicl...
2929    Congress president Sonia Gandhi on Monday reje...
2450    A file picture of Sri Rama Sene president Pram...
1682    The Committee for Consultations on the situati...
849     Farmers in China are being offered cash to qui...
596     Gilead Sciences Inc said on Tuesday, May 12 it...
1078    KEY POINTS "Paralives" practically similar to ...
1485    BRIMMING WITH IDEAS: From left: S.B. Shukla, F...
2029    Bharatiya Janata Party State president V. Mura...
Name: Content, dtype: object
```

Now, any article_id of choice can be sent in as a query, which would generate a set of articles most similar to the queried article.

Content based recommenders in conjunction with LDA is particularly advantageous when the corpus is updated with new unrated articles. This method does not depend on user behaviour and ensures that new content is discoverable to a potential user, which is important for user base retention.

IMPLEMENTING A COLLABORATION BASED RECOMMENDER SYSTEM

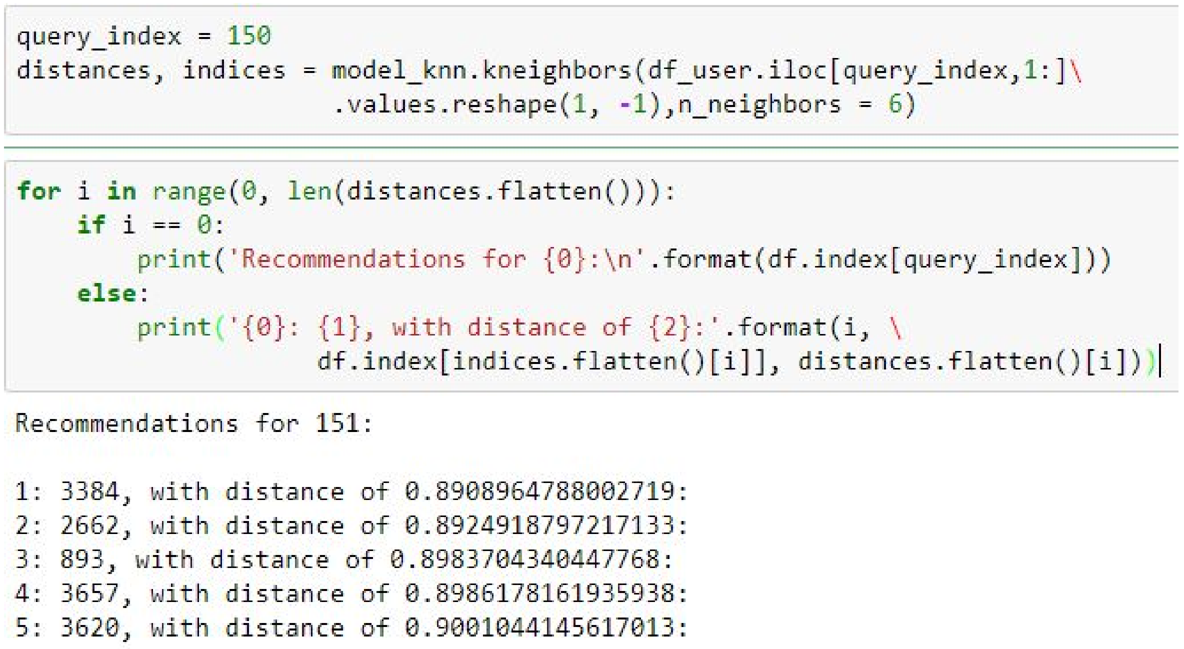
Recommending articles based on collaborative filtering of users preferences is advantageous as users get more personalized news. For a collaborative filter, we require data on user behavior and preferences, which is generated as stated before. The sparse matrix generated can be passed onto the NearestNeighbor clustering instance with similarity metric as the cosine similarity, so that clustering of similar users takes place.



```
model_knn = NearestNeighbors(metric='cosine', algorithm ='brute')
model_knn.fit(A)
```

```
NearestNeighbors(algorithm='brute', leaf_size=30, metric='cosine',
                 metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                 radius=1.0)
```

After the model is fit to the sparse matrix, we can query any user_id, and the model would return a set of articles related to the preferences of a common user group within the neighborhood of the queried user_id.



```
query_index = 150
distances, indices = model_knn.kneighbors(df_user.iloc[query_index,1:]\
                     .values.reshape(1, -1),n_neighbors = 6)
```

```
for i in range(0, len(distances.flatten())):
    if i == 0:
        print('Recommendations for {0}:\n'.format(df.index[query_index]))
    else:
        print('{0}: {1}, with distance of {2}:'.format(i, \
                            df.index[indices.flatten()[i]], distances.flatten()[i]))
```

```
Recommendations for 151:

1: 3384, with distance of 0.8908964788002719:
2: 2662, with distance of 0.8924918797217133:
3: 893, with distance of 0.8983704340447768:
4: 3657, with distance of 0.8986178161935938:
5: 3620, with distance of 0.9001044145617013:
```

Now, this approach can be combined with the function that we wrote for content based recommender, thus resulting in a hybrid recommender system. This function would take two arguments, (user_id, article_id), to recommend relevant articles to users.

WHAT MORE CAN BE DONE?

The recommender systems developed here don't consider the time instance at which a user starts reading an article. If multiple users are reading a specific article within a neighborhood of their corresponding timestamps, it could mean that the article is more relevant at this time. Incorporating the timestamp of a user clicking an article is an attribute which we tried to include in the collaborative filtering approach. This method was not performing as expected. However, many studies have shown that making a recommender system time aware leads to gains in accuracy [3,4,5].

CONCLUSION

A news recommender system was built using collaborative and content based filtering approaches. LDA was used for topic modelling of the corpus acquired through web scraping news websites.

REFERENCES

1. Tewari, P. (2016). The habits of online newspaper readers in India. *Intermedia International e-Journal ISSN: 2149-3669*, *2*(3), 295-304.
2. Groot Kormelink, T., & Costera Meijer, I. (2020). A user perspective on time spent: temporal experiences of everyday news use. *Journalism Studies*, *21*(2), 271-286.
3. Delpisheh, E., An, A., Davoudi, H., & Boroujerdi, E. G. (2016). Time aware topic based recommender system. *Big Data & Information Analytics*, *1*(2&3), 261-274.
4. Grönberg, D., & Denesfay, O. (2019). Comparison and improvement of time aware collaborative filtering techniques: Recommender systems.
5. de Zwart, T. W. (2018). Time-Aware Neighbourhood-Based Collaborative Filtering. *Research Paper Business Analytics. Vrije Universiteit Amsterdam*.
6. https://blog.insightdatascience.com/news4u-recommend-stories-based-on-collaborative-reader-behavior-9b049b6724c4
7. https://github.com/huangy22/NewsRecommender/blob/master/3_topics_extraction_with_lda.ipynb
8. https://humboldt-wi.github.io/blog/research/information_systems_1819/is_lda_final/