Indian Institute of Science Education and Research, Mohali

IDC410 Project Report

Course Code - IDC410

NO NONSENSE NEWS

Date - April 25, 2022

Group Name	-	SASS
Group Members	-	Dhruva, Rupali, Gowri, Shiv Shankar
Date of Submission	-	April 25, 2022

Contents

1 Web Scrapping and Pre-Processing

1.1 Web Scrapping

- Scraped for links of the articles using RSS Feeds. An RSS document includes the full or summarized text and metadata, such as the publication date and author's name. RSS formats are specified using a generic XML file, which makes it easy for us to scrape the data.
- Used *newspaper3k* to extract data from the article. *Newspaper3k* is a Python library that makes it very easy to scrape news articles and also provides us with the functionality of natural language processing (NLP), which provides us with the "Summary" and "Keywords" of the articles, which can be useful tool in topic modeling.
- Scraped 4 different news websites with a total of around 12,000 articles.
  - New York Times
  - BBC
  - Guardian
  - Buzzfeed

1.2 Pre-processing the Corpus

- Removing the capitalisation and converting everything to lowercase. (News and news should be the same word)
- Converting and merging the date and time to Unix Epoch time (Unix epoch is 00:00:00 UTC on 1 January 1970) throughout the corpus for easy date calculations.
- Tokenized the article data on the basis of words using 'word-tokenize' from nltk library.
- Removed the stopwords from the data using 'nltk stopwords' list.
- Removed Non-English words such as numbers, special characters as they will not add any meaning to the data.
- Lemmatized the data using 'WordNetLemmatizer' from nltk library.

For all the pre-processing and cleaning of the data, we have used the nltk library of python and its various trained models such as 'nltk.tokenize','nltk.corpus-stopwords' and 'nltk.stem-WordNetLemmatizer'.

1.3 Updating the corpus

We are updating the corpus daily (can be either daily, weekly, monthly). The rate at which the corpus is updated can be decided arbitrarily.

2 Vectorization

For analysis we need to convert the pre-processed corpus from its raw form i.e. text, into vectors of real numbers before feeding it to the machine learning algorithm. To achieve this we use TF-IDF vectorization method.

2.1 TF-IDF

Term frequency-inverse document frequency vectorizes the document by deriving weights for each token while taking into account the importance of token in describing the document. For a word w in document d, weight $W_{w,d}$ is computed using

$$\boxed{W_{w,d} = tf_{w,d} \times log\left(\frac{N}{df_w}\right)} \tag{1}$$

where, $tf_{w,d}$ is number of occurrences of w in d and df_w is number of documents containing w.

Our entire corpus is vectorized using TF-IDF method, We have put the threshold of atleast two occurrence for a word to be included in the vocabulary . Whenever the article corpus is updated, the new articles are vectorized by doing Count Vectorization for the trained vocablulary, to find the tf and use the trained idf to find the article vector.

2.2 SVD

Singular Value Decomposition factors an $m \times n$ matrix A into a product of three matrices as

$$\boxed{A = U \times S \times V^T} \tag{2}$$

where U is an $m \times k$ matrix, V is a $n \times k$ matrix, and S is a $k \times k$ matrix, where A is the rank of the matrix A.

Truncated SVD is used to the reduce the dimension of the article vector to 25 (arbitrarily decided, though should be decided by looking at where the variance flattens out with increase in dimensions). The dimension corresponds to the 25 abstract categories of any article. As a particular news article does not belong to one broad category, it is expressed as overlap of all these abstract categories in a vector form.

SVD gives us a transformation matrix using which we generate the article vector for the new articles added while updating corpus.

3 Recommendation

Here, we will recommend 5 articles each using content-based recommendation system and collaborative recommendation system to the users.

3.1 User Vectors

The users are also represented as vectors of real numbers of dimension 25. These vectors are based on users' preference of 25 abstract categories. New users are initialised by a zero vector and their preferences are updated by responses to recommended articles.

Upon reading an article A the user preference vector U gets updated to U' using the formula,

$$\boxed{U' = \frac{\beta n U + (r + c)A}{\beta n + 1}} \tag{3}$$

where n is the total number of articles read by the user and β is a hyper-parameter, with value 0.85 ($>$0.5), included to adjust the weightage given to already read article w.r.t. recently read article. c is the clickthrough score which can again be varied.

r in Eq.(3) is rating for the article, which is assumed to vary from -1 to 1. Since not all users give ratings, we use the time spent t as a proxy. We use the below formula to translate t to r

$$r = \frac{t - \mu}{2\sigma} + 0.5 \tag{4}$$

where μ and σ are the mean and average time taken to read the article. We have assumed a mean speed of 250 wpm and a stdev of 10 wpm. This value is then clamped to the expected range of $[-1, 1]$. If however the time taken is more than 2σ, we assume that the user has left the article and a 0 score is given.

This formula assumes that if the user likes an article they will read the entire article spending $t \geq \mu$ hence, rating it high i.e. 0.5 to 1 and if the user is not interested in the article they will spend less or no time reading it.

We have generated 100 synthetic user preference vectors and associated random value of total number of articles read with each in order to have user clusters to be used in collaborative based filtering (section 3.3).

3.2 Content based recommendation systems

The idea here is to recommend the articles which are similar to the content user prefers from the click stream data.

One important distinction of our model is that the articles themselves are not ranked according to user preference. According to a particular user preference vector, we will score the article vector using the algorithm below:

- Take the euclidean distance between the article vector and preference vector (dist). Here, we're not taking cosine similarity as it could mistake two articles in the same direction with different lengths as the same.
- We will take the time difference between the publishing date and the current time as age. As age increases the score must go down.
- Set γ the hyper parameter which is the half life of an article given by $ln2/age$. Here we will set γ same for all articles.

$$\boxed{\text{score } = \frac{e^{-\gamma t}}{1 + d}} \tag{5}$$

- We know that the relevance of an article decreases non linearly. This is best explained using an exponential function. Also we have used the distance parameter. If the distance between user preference and article is infinity the score must be zero. Hence we're dividing distance (dist). We are adding 1 to avoid zero division in case distance is zero.

3.3 Collaborative recommendation systems

Collaborative recommendation systems use an approach that focuses on the relations between users. The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion/taste as a person B on a set of items, A is more likely to have B's opinion for a given item than that of a randomly chosen person.

Here we will cluster our users using k-means clustering algorithm. This will give us get the mean vector of the user preference clusters with similar interests. Then we can apply a content based recommendation on that vector and recommend 5 articles each to all the users in that cluster. But we are clustering the user preferences immediately after each requests. Hence the recommendations will remain dynamic for all the other users in the cluster.

3.4 Cold Start Problem

The cold start problem is addressed well within our model. The new user preferences are initialised as zero vectors. So, the first recommendation will be random based on the content based and collaborative recommendation. But as the user starts interacting, the preference vector is updated and articles are recommended.

4 Advantages and disadvantages of our model

Advantages of our model include -

- Update corpus without retrain
- Does not depend on the clickbait-ness of any article allowing for less "famous" topics to show up as well as mainstream topics
- Allows new topic exploration through collaborative recommendations
- Abstract enough to be used for any and all item recommendations. Only the vectorization step has to be overloaded/changed to make it appropriate

Disadvantages of our model are -

- User based clustering may be redundant
- Updating user preferences on re-train is nontrivial due to multiple translations, but is still possible.
- Model is not immediately translatable to an online learning model

5 Implementation details and Flask Application

The implementation can largely be broken down into five sections -

- User related
  - Structure to hold User data
  - Updating and accessing property interfaces
  - Signup and login procedures - Authentication
- Article related
  - Structure to hold Article data
  - Updating and accessing property interfaces
  - Summarization
  - Preprocessing
- Model related
  - Abstraction of Model
  - Training - Projection matrix calculation
  - Vectorization

  - Clustering of Users
  - Recommendation
  - Preference update
- UI related
  - Login/Signup Screen
  - Article List View
  - Article View
  - Endpoints and routes
  - User clickstream and session recording
- Database and storage
  - Tables with appropriate attributes
  - Reliable and backup-able
  - Scale-able
  - Session and event recording

In order to manage a reasonably large project, we have divided the code into appropriate files and classes, with relevant functions and properties. Further, we have provided bootstrap scripts to make and prefill the database with articles and/or users.

This Object Oriented approach also allows us to abstract out the specifics of our model from the UI and storage mechanisms, allowing for easy update of model parameters, model pipeline and any other features.

5.1 Application Design Approach

We have assumed that this Application is a service that is hosted by anyone with a server and this admin has full access to update the model, change parameters, update article data and manage user data. There is trust between the user and the admin.

Further, the database and recommendation is such that they can be updated without having to stop the flask application, allowing online/automated updates. The model is retrained by simply restarting the application, which greatly minimizes the effort needed on the user end.

5.2 Tool choices and rationality

Python is used as the main backend language for its use and vast library support in the UI and Machine Learning sections. SQLite is used as an file-based database system due to its speed, backup features, and minimal feature set and easy setup, with well supported bindings for python. Flask is used for its minimalist approach to web pplication building.

5.3 User Clickstream collection

Probably the most implementation-ally complex part of the whole project was simplified greatly by the Session and Event recording system in place.

On login, user begins a new session which then records each event by the user. By measuring the time between the READ event (an event recorded when user clicks a new article) and the next event

reaching the web server, we find the interaction time for the preference update mechanism explained before.

This mechanism has the benefit of working even in environments not supporting Javascript such as screen readers, accessibility features, commandline browsers, and computers where Javascript is blocked for security reasons.

5.4 Managing Incomplete Data

Inevitably, some data we receive is incomplete. In case the date of publishing the article is not available, we set current time as date of publishing. This should work on the basis that RSS feeds are generally updated daily. Other data is left as NULL/None since they are not important for our model.

6 What would we do if we had more time?

- Model hyperparameters such as β, γ, c can be found by optimizing over some trial runs.
- Our model does not keep track of the articles the user has read. This will lead to re-recommendations. We would make an addition to accommodate this.
- Coding in updating user preferences on model re-train
- Some stub articles end up in our article set. These article should be dropped on account of their low word count, or exceptionally small article vector length.
- In the scoring function we could check what power of distance d is suitable for the model.
- We would ask the user to rate the articles while using time spent as a proxy for it. This rate would be used in the user preference updating algorithm, with the time spent algorithm as a fallback
- Using the sentiment analysis methods to further supplement our article and user preference space.
- Safer authentication by using POST Method and salt-hashing passwords
- Streamlining user experience by making vectorization prompt instead of lazy.
- Document the code. This, as is for most other student led projects, will be our last priority, though it should be our first.