Taylor Herb

INFSCI2480 Final Project  
Spring 2021

Content Based News Recommender System

https://github.com/t-4-h/infsci2480finalProject

I. Introduction

News websites and blogs are a highly popular aspect of the internet we use today. With a decrease in printed newspapers, online news websites have become an increased source for people to access local, domestic and international news. Since these news sources are being used by people so frequently, they would likely benefit from including a recommender system to personalize the user’s experience. For example, if a user reads an article about the stock market in Europe, they may want to read additional articles on the topic. This could mean they want to see more articles about the stock market in general, Europe, the stock market in China, or maybe they aren’t sure where to look next. In this situation, a recommender system would be useful to show them other relevant articles on the site that they may be interested in reading.

In this project, I will begin the process of designing a news recommender system by first using content-based recommendation methods. Since there is no user data on this dataset yet, the recommender system must rely on the information about the articles themselves rather than the user. The goal of the current system is to allow the user to search for recommendations based on the article’s text body or the article’s tagged keywords.

II. Dataset and Pre-processing

The original dataset was pre-processed in order to remove unnecessary columns, missing data and duplicate entries. The final dataset included the five columns to be used in the recommender system (id, title, text, keyword, link) and 1720 rows that did not include any missing data or duplicates.

Graphical user interface, text

Description automatically generated with medium confidence

Fig 1: Original data (2190 rows, 9 columns)

Graphical user interface, text, application, email

Description automatically generated

Fig 2: Final dataset after initial pre-processing (1720 rows, 5 columns)

Additionally, text cleaning methods were used to clean the article text and keywords data so that the text could be implemented efficiently within the recommender system. The two text columns were cleaned by converting the text to lowercase, removing special characters, and removing stopwords. In combination with cleaning, the text from the keywords column were stored in a string array.

III. Technologies

For this project I have primarily used Python and Jupyter lab. Since the database is small and static, the .csv database is stored within the Jupyter notebook. The interactive portion works using the ipywidgets interact library. A pseudo front-end is created using the Voila server extension, which creates a simple web application that allows the user to enter an article title and receive the top 20 recommendations based on either the article text or article keywords. The user can view the recommended article titles and then click links to visit the article online.

IV. Recommender Design

This recommender will use content based filtering methods to give recommendations based on the articles text or the articles keywords. The reason for doing both is to understand which system performs more accurately in general or which system may perform best under certain conditions. Logically, it would seem that keywords may be helpful when a user is interested in a broad topic while text may be more helpful for more specific topics (including multiple terms). Therefore, I decided it would be useful to build and test both in order to develop a more well-rounded system.

To build both recommender types (text-based and keyword-based), Term Frequency-Inverse Document Frequency (TF-IDF) was used on the text to evaluate term importance to the document within the corpus. To accomplish this, I used the TfidfVectorizer method from the sklearn.feature library. For the keyword-based recommender an ngram range of 1 to 2 was used (because keywords typically exist of a maximum of 2 terms) with a min\_df of 0 (to include all keywords regardless of document frequency). For the text-based recommender, an ngram range of 1 to 3 was used to allow for longer combinations of terms within the article. Although stopwords were removed, a max\_df of 0.8 was also selected to ignore terms that occur too often in the corpus because they are unlikely useful for recommendations (i.e. words similar to stop words that do not add meaning). Additionally, multiple min\_df values were tested (0.0, 0.01, 0.02, 0.1) which yielded different matrix dimensions based on the amount of terms used [(1720, 1023226), (1720, 4705), (1720, 2546), (1720, 321)]. The results from min\_df 0.1 were not tested due to such a small number of terms. However, the other 3 matrices all yielded seemingly accurate and relevant results in the recommendation system. It was impossible to know which performed best without user feedback. So, min\_df = 0 was ultimately chosen as it contains the most terms but user feedback should be included to understand which parameters perform best.

Graphical user interface, text, application

Description automatically generated

Fig. 3: Matrix dimensions with different min\_df values for text-based recommender

Once the article text or keywords were encoded into tf-idf matrices, I used cosine similarity (sklearn cosine\_similarity method) to obtain a similarity measure based on proximity between the two matrices. Cosine similarity is useful here because it is a measure of distance and orientation rather than magnitude (i.e. Euclidean distance).

Text

Description automatically generated

Fig. 4: Cosine similarity equation showing similarity ranging from -1 (opposite) to 1 (equal) where 0 indicates no relationship.

The recommender function was built to create indices for titles then find recommendations based on the cosine similarity (cosSim). This function returns the top 20 most relevant items and shows the user the title of the article as well as the link to the article.

Graphical user interface, text

Description automatically generated

Fig. 5: Recommender function based on cosine similarity

A picture containing text, newspaper, screenshot

Description automatically generated

Fig 6: Results for text-based recommender (title “Bank of England policymakers warn of bigger risks for UK economy”

Text

Description automatically generated

Fig 7: Results for keyword-based recommender (title “Bank of England policymakers warn of bigger risks for UK economy”

V. Front-End

This recommender system was developed across three Jupyter notebooks. To test and further develop the system, the recommender-textBased.ipynb and recommender-keywordBased.ipynb notebooks (i.e. development notebooks) can be edited, tested and extended. These notebooks allow for testing of new parameters such as the min\_df or editing the results to include a different number of recommendations. When using Jupyter lab, these notebooks can be quickly tested using the interact widget at the bottom of the screen.

The third notebook (i.e. production notebook) contains the code for both recommender systems but is formatted to be presented via Voila. This notebook does not contain any output or markdown, but is useful for presenting the recommender systems.

Graphical user interface, application

Description automatically generated

Fig. 8: Screenshot of part of the keyword-based development notebook (recommender-keywordBased.ipynb)

Text

Description automatically generated

Fig. 9: Screenshot of part of the production notebook (FULL-RECOMMENDER.ipynb)

VI. Conclusions and Future Work

In this project, I have developed a news recommender system based on article text and article keywords. Since there is not yet any user data, content based methods were used to provide the recommendations about the articles. TF-IDF feature extraction and cosine similarity methods were used to create similarity measures for the recommendations.

Currently, it seems that both recommendation systems perform quite well despite the absence of any additional user data. At first glance it seems as though the text based recommender is more consistently accurate, however, it is impossible to guess which model performs better in the absence of user feedback. As the project develops, collaborative filtering methods can be employed based on user behavior data that is gathered while they interact with the system. Ideally, future users will be able to give feedback about their recommendations in order to fine tune the algorithms and personalize their experience. Additional features such as creating profiles, “favoriting” articles and searching by other terms (i.e. search by keyword or author) will be implemented during continued front-end development to help make the system more adaptive and customizable.