Recommending Alpha

Group #16

Technical Documentation

Group Members:
Vedanta Dhobley: vjd41@scarletmail.rutgers.edu
Avani Bhardwaj: ab1572@scarletmail.rutgers.edu
Shazidul Islam: si194@scarletmail.rutgers.edu
Akshat Shah: avs91@scarletmail.rutgers.edu
Kutay Kerimoglu: kk851@scarletmail.rutgers.edu
Alan Patel: akp122@scarletmail.rutgers.edu
Anthony Matos: amm720@scarletmail.rutgers.edu
Joel Cruz: jc2125@scarletmail.rutgers.edu

Python Code that contains the algorithm for the recommendation and the flask server to communicate with the website
Bold text in the code is the comments/documentation

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
```
Created on Sun Mar 17 15:31:37 2019

@author: Alan Patel

This section of the code uses a python web framework called Flask. It makes it easy to integrate our python code with the HTML code that we have written. In this part flash is just taking our index.html file and rendering it into python code internally so we can read and write to it. If the flask does not run properly that may mean it is not installed into the python packages and if it still does not work then in the utils.py files change the file argument in the echo method to sys.stdout and do the same in in the secho method in the termui.py method. After the flask is running open up the local server on any internet browser to see the webpage.

```
from flask import Flask, render_template, request
app = Flask(__name__)
@app.route('/')
def output():
    return render_template('index.html')
```

Here I am just importing the necessary packages to be used.

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

 # Input data files are available in the "../input/" directory.
 # For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory
Make sure the argument inside the os.listdir matches whatever folder all the code is in.

```
import os
print(os.listdir("../database"))
```

 # Any results you write to the current directory are saved as output.
The sklearn is a powerful statistical package that essentially does all of the brute calculations necessary for our cosine similarity algorithm.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
```

The code below is just taking the data we have from the csv files and using pandas to store them as dataframes. This may take a few seconds but it only has to be done once.

```
books = pd.read_csv('books.csv', encoding = "ISO-8859-1")
books.head()

ratings = pd.read_csv('ratings.csv', encoding = "ISO-8859-1")
ratings.head()

book_tags = pd.read_csv('book_tags.csv', encoding = "ISO-8859-1")
book_tags.head()

tags = pd.read_csv('tags.csv')
tags.tail()
```

This is doing a inner join to combine the tag and the book tags to create a dataframe that has all the tag numbers and tags in one dataframe.

```
tags_join_DF = pd.merge(book_tags, tags, left_on='tag_id', right_on='tag_id', how='inner')
tags_join_DF.head()

to_read = pd.read_csv('to_read.csv')
to_read.head()
```

This code uses the sklearn package to transform all the authors into vectors and then used the linear_kernel to give a matrix that stores all the similarity scores between each vector.

```
tf = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix = tf.fit_transform(books['authors'])
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

This code is not used but contains the methods to recommend books using just authors and just book tags. Look for the next set of comments that detail how I combined authors and tags into the alpha recommendations.

```
# Build a 1-dimensional array with book titles
titles = books['title']
indices = pd.Series(books.index, index=books['title'])
"""
def authors_recommendations(title):
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:21]
```

```
    book_indices = [i[0] for i in sim_scores]
    return titles.iloc[book_indices]

authors_recommendations('The Hobbit')
"""
books_with_tags = pd.merge(books, tags_join_DF, left_on='book_id',
right_on='goodreads_book_id', how='inner')

tf1 = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0, stop_words='english')
tfidf_matrix1 = tf1.fit_transform(books_with_tags['tag_name'].head(10000))
cosine_sim1 = linear_kernel(tfidf_matrix1, tfidf_matrix1)

# Build a 1-dimensional array with book titles
titles1 = books['title']
indices1 = pd.Series(books.index, index=books['title'])
"""
def tags_recommendations(title):
    idx = indices1[title]
    sim_scores = list(enumerate(cosine_sim1[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:21]
    book_indices = [i[0] for i in sim_scores]
    return titles.iloc[book_indices]

tags_recommendations('The Hobbit').head(20)
"""
```

Here we are grouping all the books that hae tags into a dataframe that we can apply the algorithm to because books with no tags and tags with no corresponding books are pointless to us.

```
temp_df = books_with_tags.groupby('book_id')['tag_name'].apply(' '.join).reset_index()
temp_df.head()

books = pd.merge(books, temp_df, left_on='book_id', right_on='book_id', how='inner')
books.head()
```

Creating a series allows us to just use the author and tag name to find a book.

```
books['corpus'] = (pd.Series(books[['authors', 'tag_name']]
                .fillna('')
                .values.tolist()
                ).str.join(' '))
```

The data that has the author names and tags are not put into the vectorizer to transfrom into vectors and then the linear kernel creates a matrix of similarity scores between all the vectors(aka books) that we can use for the ranking.

```
tf_corpus = TfidfVectorizer(analyzer='word',ngram_range=(1, 2),min_df=0,
stop_words='english')
tfidf_matrix_corpus = tf_corpus.fit_transform(books['corpus'])
cosine_sim_corpus = linear_kernel(tfidf_matrix_corpus, tfidf_matrix_corpus)

titles = books['title']
indices = pd.Series(books.index, index=books['title'])
```

#this is the main function, the rest of functions for just author or just genre
This function takes 3 book titles as arguments. It then finds the index of where that book is located in the book series which was made above. Then it adds up the similarity scores for all three books into one matrix that has the sum values of the 3 books. The matrix is then converted into a list that can be sorted from largest to least value. Then we takes the indices of the top 20 books but not including the top 3 because the top 3 is the 3 books that were inputed and using those indices we can return the books using a simple iloc.

```
def alpha_recommendations(title,title2,title3):
    idx = indices1[title]
    idx2 = indices1[title2]
    idx3 = indices1[title3]
    sim_scores = list(enumerate(cosine_sim_corpus[idx]))
    #sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores2 = list(enumerate(cosine_sim_corpus[idx2]))
    sim_scores3 = list(enumerate(cosine_sim_corpus[idx3]))
    #sim_scores2 = sorted(sim_scores2, key=lambda x: x[1], reverse=True)
    total = [(c, e+h) for (c, e), (d, h) in zip(sim_scores, sim_scores2)]
    total  = [(c, e+h) for (c, e), (d, h) in zip(total, sim_scores3)]
    #total = list( map(add, sim_scores, sim_scores2))
    #total = list( map(add, total, sim_scores3))
    total = sorted(total, key=lambda x: x[1], reverse=True)
    total = total[3:23]
    book_indices = [i[0] for i in total]
    return titles.iloc[book_indices]

import re
regex = re.compile('[^a-zA-Z, ]')
```

This function is the same function as the alpha recommendation function but I had to copy and paste the code in the result function because the result function is the function that flask is using to send the output to our webpage.

```
@app.route('/result',methods = ['POST','GET'])
def  result():
    if request.method == 'POST':
        result = request.form.getlist('bookname')
        result = regex.sub('',str(result))
        b = result.split(',')
        #print(result)
        idx = indices1[b[0]]
        idx2 = indices1[b[1]]
        idx3 = indices1[b[2]]
        sim_scores = list(enumerate(cosine_sim_corpus[idx]))
        #sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
        sim_scores2 = list(enumerate(cosine_sim_corpus[idx2]))
        sim_scores3 = list(enumerate(cosine_sim_corpus[idx3]))
        #sim_scores2 = sorted(sim_scores2, key=lambda x: x[1], reverse=True)
        total = [(c, e+h) for (c, e), (d, h) in zip(sim_scores, sim_scores2)]
        total  = [(c, e+h) for (c, e), (d, h) in zip(total, sim_scores3)]
        #total = list( map(add, sim_scores, sim_scores2))
        #total = list( map(add, total, sim_scores3))
        total = sorted(total, key=lambda x: x[1], reverse=True)
        total = total[4:24]
        book_indices = [i[0] for i in total]
    abc = titles.iloc[book_indices]
   # return str(abc)
   # return str(result)
       # return indices.astype('str')
    return render_template("result.html",result = abc)
```

This just makes sure our flask server is running.

```
if __name__ == '__main__':
    app.run(debug = True)
#alpha_recommendations("The Hobbit", "The Catcher in the Rye", "Romeo and Juliet")
```

@author : Anthony Matos & Akshat Shah

Following is the description of the Frontend section for HTML. The code is divided into two HTML files, index.html and result.html, describing how the webpage is presented to the user in the beginning and after executing the search. Index.html is the main frontend component that the user interacts with initially and result.html. The index html is the

foundation for the recommendation software and contains all the formatting properties responsible for making this user friendly.

This segment of code below is the content delivery network and its purpose is to provide faster delivery, and highly available content. The CDN makes it easier for content to be in many places at once without sacrificing speed. What is also included is a library of all the different styles of buttons and fonts for texts.

Index.html

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<meta name = "viewport" content = "width=device-width, inital-scale=1">
<link rel="stylesheet"
```

These are the queries we accessed from Bootstrap to set up our web page designing and styling.

```
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa" crossorigin="anonymous"></script>
```

Here is the CSS portion of the code that customizes the font and positioning of items like buttons and toolbars on the screen. Any name that starts before this { refers to an element of the page.

```
body{
        background:#202a3f;
}
.container{
        margin-top:200px;
}
.glyphicon-search{
        font-size:20px;
}

.btn-default{
        background: orange;
        width:100px;
        padding:12.5px;
}

.form-control{
```

```
        padding:25px;
        font-size:20px;

}

.input-group-btn{
width:device-width;


}

}
.navbar {
  overflow: hidden;
  background-color: #333;
}

.navbar a {
  float: left;
  font-size: 16px;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
}

.dropdown {
  float: left;
  overflow: hidden;
}

.dropdown .dropbtn {
  font-size: 16px;
  border: none;
  outline: none;
  color: white;
  padding: 14px 16px;
  background-color: inherit;
  font-family: inherit;
  margin: 0;
}

.navbar a:hover, .dropdown:hover .dropbtn {
```

```
  background-color: red;
}

.dropdown-content {
  display: none;
  position: absolute;
  background-color: #f9f9f9;
  min-width: 160px;
  box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
  z-index: 1;
}

.dropdown-content a {
  float: none;
  color: black;
  padding: 12px 16px;
  text-decoration: none;
  display: block;
  text-align: left;
}

.dropdown-content a:hover {
  background-color: #ddd;
}

.dropdown:hover .dropdown-content {
  display: block;
}
}
```

In this form, we have a local host server for our algorithm to perform via its python code and serves to receive the input (all three books) from the user and pass it through the algorithm. An input group contains all the essentials of a working search bar. These include a responsive text bar with a dropdown list when clicked on. In order to get search results, a submit input type is needed to direct the user to the local host page where all the recommendations are expected to be displayed.

```
  <div class="container">
    <form action="http://localhost:5000/result" method = "POST">
                    <div class = "input-group">
                            <input type="text" class = "form-control" placeholder="Search"
name="bookname">
```

```
                    <div class ="input-group-btn ">
                    <!--<input type = "submit" value = "submit" />-->
                    <button  type="submit" class = "btn btn-default">

                    <i class="glyphicon glyphicon-search"></i></button>
                    </div>
                    </div>
        </form>
 </div>
```

Result.html

```
<!doctype html>
<html>
        <body>
```

Title refers to what the new tab read reads for user-friendly look after the search output page is loaded

```
                    <title>Alpha Recommendation</title>
                    <ol>
```

Looping through the elements of result object referenced from the Python code, which sends the output to frontend stored in result. Result is only referenced by result.html only after the server is launched by Flask.

```
                    {% for i in result%}
```

Return the output in the form of numbered list form to display an organized data.

```
                            <li>{{i}}</li>
                            <br>
                    {% endfor %}
                    </ol>

        </body>
</html>
```