Recommending Alpha

Group #16

Code

**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

**Shazidul Islam:**

{

import pandas as pd

import os

import time

os.chdir("C:\\Users\\shazi\\Desktop\\SoftwareEngineeringProject2019\\database")

def BookRetriver(bookList):

#bookList has the goodreads number

my\_tags=pd.read\_csv("book\_tags.csv")

#Stores the tags csv file in my\_tags

tagList=[] #create tag list

for book in bookList:#go through each book in the list

BookTag=(my\_tags.loc[my\_tags["goodreads\_book\_id"]==book, "tag\_id"])

#Retrives only the Tag\_id associated with the books

for i in BookTag:

tagList.append(i)

return tagList

start=time.time()

myBooks=[1,6,45,2,325,8956,56,98,1584,456,789,456,789,456,789,456,789,456,1233,456,7899,456,456,8,648,1132,15,14,13,20]

print (len(myBooks))

print(BookRetriver(myBooks))

end=time.time()

print(end-start)

}

{

import pandas as pd

import os

os.chdir("C:\\Users\\shazi\\Desktop\\Software Engine Project\\SoftwareEngineeringProject2019\\database")

myTags=pd.read\_csv("tags.csv")

myTags=(myTags["tag\_name"])

#print(myTags) ##Test line

tag\_list=[]

for tag in myTags:

tag\_list.append(tag)

#print(tag\_list) ##Test Line

TagCounter=[0]\*len(myTags) #This would be the counter for all the tags. Created new every time

testList=['Horror','childrens-classics','erica-jong','Horror','Adventure','Horror','Adventure', 'Religion','Horror','Horror',]

def UpdateCount(userList):#Function will update the counters

for i in range(len(myTags)):#Length of the list

for j in testList:#gets the tags from the list from user

if myTags[i]==j:#checking if the tags match

TagCounter[i]+=1#add to the counter

UpdateCount(testList) ## Testing Line

total=list(zip(myTags, TagCounter))#combine the counter with the tag ID for a pseudo database

#print(total) ##Test Line

def NonZerolist(tagCount):#function that will eliminate zero count tags to speed up time later

newlist=[] #Temp new list that holds the non zeros

for k in range(len(tagCount)):

tempList=tagCount[k]

if tempList[1]!=0:#if the tag has a count then store it

newlist.append(tempList)

return newlist#Gives back the new list and forgets the old list

total=NonZerolist(total)

print(total) ##Test line

###The only issue with this implementation

#right now would have to be the it is brute force

#the run time is too long

#figure out a way to make the run time shorter

}

**These are the code that I, Shazidul, have worked on. The idea was to first get the tags associated with the books and them depending on how many times a tag appeared, we would put more weights on that tag than the other one.**

**Alan Patel:**

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Sun Mar 17 15:31:37 2019

@author: Alan Patel

"""

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

@app.route('/')

def output():

return render\_template('index.html')

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

import os

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

# Any results you write to the current directory are saved as output.

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import linear\_kernel

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()

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()

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)

# 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)

"""

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()

books['corpus'] = (pd.Series(books[['authors', 'tag\_name']]

.fillna('')

.values.tolist()

).str.join(' '))

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

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, ]')

@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)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

#alpha\_recommendations("The Hobbit", "The Catcher in the Rye", "Romeo and Juliet")

**This is the code that I, Alan Patel, have worked on. It is the main set of code that does the book algorithm and outputs the top 20 most books to the webpage which I have done using a flask web framework. More details can be found in the technical documentation.**