Research Paper Recommendation System

CMPE-256 Group Project

San Jose State University

Team No. 8

Shreya Hagalahalli Shrinivas (013821405) **Purva Deekshit** (013749723) **Maulin Bodiwala** (013733590) **Kiran Kumar Bijjala** (013772551) **Siddartha Reddy Maryada** (013849238)

ABSTRACT

This paper contains implementation of research paper recommendation system using a hybrid approach. We have combined content-based recommendations, user-based collaborative recommendations and item-based collaborative recommendations to design a hybrid model that can recommend more accurate and relevant papers to the readers. We have used reader's preferences and information of similar peers for title analysis, abstract analysis and implicit ratings to provide the recommendation. By combining reader's ID and keywords entered, this approach is able to provide top N most relevant research papers. We have also implemented a deep learning technique to extract hidden features from text and provide more accurate recommendations. For evaluation, we have designed a UI to enter the user ID and see the recommended papers. Finally, we have stored the recommendation results in a CSV file to evaluate if the recommendations are relevant.

Key Terms: Recommendation system, content-based filtering, user-based collaborative filtering, item-based collaborative filtering, hybrid approach, deep learning approach.

MOTIVATION

Researchers spend a lot of time searching for useful and relevant papers. Existing techniques like [1] [2] use keyword matching using the query entered by the user to search the papers. However, these techniques are limited by the choice of keywords user enters. If there are not enough relevant keywords, the search results may not be accurate. Also, it is possible that the latest and most relevant paper is displayed much later than an older one, because it matched more keywords. There is a possibility of missing out useful papers because of the keywords entered.

Some techniques [3] also used references listed in the papers in order to recommend those papers. However, these techniques assume that the list of references is freely available and accessible, which is not always true in case of research papers.

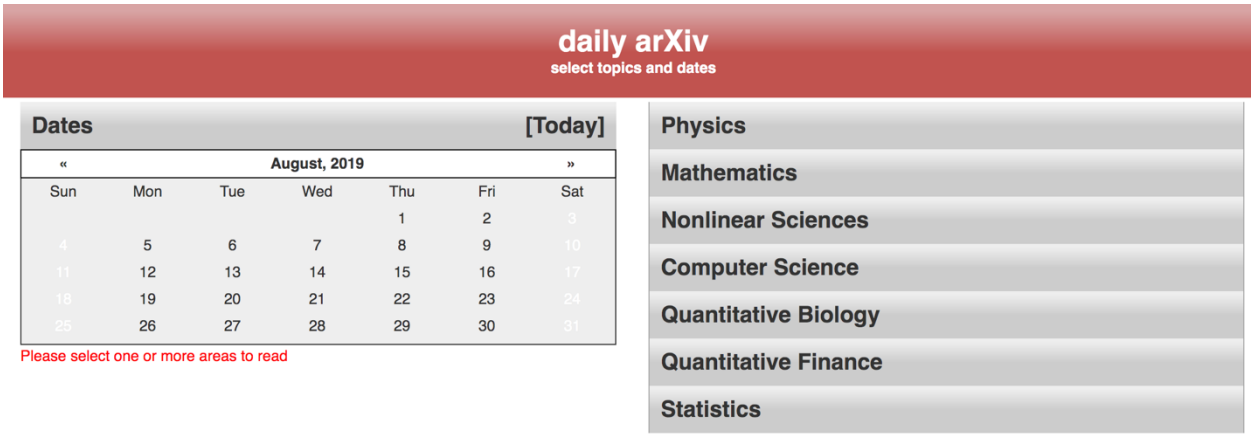
So, in order to extract most accurate results, we have combined the two most popular approaches, content-based filtering and collaborative filtering. We have used historical data based on user's referenced papers and the papers referred by most similar users and implemented a pipelined hybrid approach to give better recommendations.
We have also applied machine learning technique to extract hidden patterns from historical data. We have designed a neural network that can learn implicit features of research papers and give more accurate results.

DATA COLLECTION:

1. Research Paper Metadata:

We have used arXiv API to scrape the data. ArXiv provides a module under Python standard library called as urllib. We have used this module. The documentation is available on:
Link: https://arxiv.org/help/api#documentation.

Arxiv contains research articles from many categories like Physics, Mathematics, Computer Science, etc., as shown below:



We have selected Computer Science as main category and extracted 5000 papers for each of the 6 sub-categories as shown below:

AI: Artificial Intelligence
CE: Computer Engineering
SE: Software Engineering
DB: Database
NI: Networking and Internet Architecture
GR: Graphics

```
#Fetch Research Paper Metadata using arXiv API
paper_data ={}

paper_categories = ['cs.AI','cs.CE','cs.SE','cs.DB','cs.NI','cs.GR']
no_of_papers_per_category = 5000
paper_id = 0
```

There are various fields present for each paper. From those, we have selected only informative categories, as shown below:

```python
def extract_fields(item):
    item_details = {}
    item_details['arxiv_id'] = item['id']
    item_details['published'] = item['published']
    item_details['title'] = item['title']
    item_details['summary'] = item['summary']
    item_details['authors'] = item['authors']
    item_details['arxiv_primary_category'] = item['arxiv_primary_category']['term']
    item_details['pdf_url'] = item['pdf_url']
    item_details['affiliation'] = item['affiliation']
    item_details['journal_reference'] = item['journal_reference']
    item_details['doi'] = item['doi']
    return item_details
```

We have stored this metadata in a pickle file papers.pkl for ease of use. We have a dataset of approximately 25000 papers, as certain categories did not have 5000 research papers. Below is the sample dataframe after loading the pickle file:

```
In [6]:
#Load paper metadata in pandas dataframe
papers_df= pd.DataFrame.from_dict(paper_rows, orient='columns')

In [7]:
papers_df.shape

Out[7]: (24256, 10)

In [8]:
papers_df.head()
```

	affiliation	arxiv_id	arxiv_primary_category	authors	doi	journal_reference	pdf_url	published	summary	title
0	None	http://arxiv.org/abs/cs/9308101v1	cs.AI	[M. L. Ginsberg]	None	Journal of Artificial Intelligence Research, V...	http://arxiv.org/pdf/cs/9308101v1	1993-08-01T00:00:00Z	Because of their occasional need to return to ...	Dynamic Backtracking
1	None	http://arxiv.org/abs/cs/9308102v1	cs.AI	[M. P. Wellman]	None	Journal of Artificial Intelligence Research, V...	http://arxiv.org/pdf/cs/9308102v1	1993-08-01T00:00:00Z	Market price systems constitute a well-underst...	A Market-Oriented Programming Environment and ...
2	None	http://arxiv.org/abs/cs/9309101v1	cs.AI	[I. P. Gent, T. Walsh]	None	Journal of Artificial Intelligence Research, V...	http://arxiv.org/pdf/cs/9309101v1	1993-09-01T00:00:00Z	We describe an extensive study of search in GS	An Empirical Analysis of Search in GSAT

2. User Data:

User data was readily available. So, we have simulated the users as students who refer the research papers. We have considered the same 6 sub-categories (AI, CE, SE, DB, NI, GR) as the departments for students. We have simulated total 120 students, where each department has 20 students. The code snippet for generating students data is:

```
papers_df.to_pickle("./papers.pkl")
```

```
#Students data
student_data ={}

specializations = ['AI','CE','SE','DB','NI', 'GR']
no_of_stuents_per_specialization = 20

for i in range(len(specializations)):
    for j in range(0,no_of_stuents_per_specialization):
        student_id = (i*no_of_stuents_per_specialization) + j + 1
        student_name = "User" +str(student_id)
        student_data[student_id] = {"id" : student_id, "Name" : student_name,
```

```
for k, v in student_data.items():
    print (k,v)
```

```
1 {'id': 1, 'Name': 'User1', 'Specialization': 'AI'}
2 {'id': 2, 'Name': 'User2', 'Specialization': 'AI'}
3 {'id': 3, 'Name': 'User3', 'Specialization': 'AI'}
4 {'id': 4, 'Name': 'User4', 'Specialization': 'AI'}
5 {'id': 5, 'Name': 'User5', 'Specialization': 'AI'}
```

We have also created 9-digit student ID for each student, to resemble IDs in any university, as shown:

```
#generate 9 digit unique studuent id for each student
import random
student_id_list = random.sample(range(100000000, 999999999), 120)
print("student_id_list : ", student_id_list)
```

```
student_id_list :  [621741617, 771796474, 113004597, 711605921, 791937775, 416245271, 745317241, 234344310, 203622525
, 516072574, 721150742, 364345364, 329420086, 287805469, 109243044, 994061355, 824243382, 270137119, 379813742, 24764
3595, 156661207, 801639401, 113212033, 147666985, 189980199, 464889304, 952277614, 937374421, 269011539, 336512426, 7
27638374, 914768382, 101538213, 609478740, 148971539, 962714657, 347453539, 953251492, 852281750, 451761436, 88753276
4, 672000310, 582805572, 817374566, 283903943, 702959591, 438831435, 208890941, 489140413, 948459533, 780847091, 6970
67906, 204179578, 935684965, 970655764, 846644142, 362051458, 742123764, 277519421, 744291833, 282608569, 580172212,
673694590, 694806456, 554465398, 401201707, 721387944, 765727212, 611007001, 764524546, 592645830, 125801474, 9686273
86, 309990264, 669753940, 505615668, 360609518, 551416495, 789158069, 199981261, 460870346, 533095524, 273781905, 327
138881, 269720635, 635051958, 180406698, 794973842, 386909967, 663719059, 777984935, 295531533, 769432073, 171135840,
999187360, 196489910, 365427867, 540571476, 107672897, 520196581, 955895321, 826711045, 898320479, 100263741, 6417495
20, 503706165, 308632160, 538898910, 980413907, 709894110, 821616931, 328072870, 708962761, 334638401, 593148665, 182
197878, 320799253, 466476776, 944755006, 207940778]
```

We have stored students data in a pickle file students.pkl. This is the screenshot of students dataframe before and after adding the 9-digit user IDs:

	Name	Specialization	id
0	User1	AI	1
1	User2	AI	2
2	User3	AI	3
3	User4	AI	4
4	User5	AI	5
20	User21	CE	21
21	User22	CE	22
22	User23	CE	23
23	User24	CE	24
24	User25	CE	25
40	User41	SE	41
41	User42	SE	42
42	User43	SE	43
43	User44	SE	44
44	User45	SE	45

	Name	Specialization	student_id
0	User1	AI	621741617
1	User2	AI	771796474
2	User3	AI	113004597
3	User4	AI	711605921
4	User5	AI	791937775
20	User21	CE	156661207
21	User22	CE	801639401
22	User23	CE	113212033
23	User24	CE	147666985
24	User25	CE	189980199
40	User41	SE	887532764
41	User42	SE	672000310
42	User43	SE	582805572
43	User44	SE	817374566
44	User45	SE	283903943

2. User-To-Paper-Mappings:

For user-to-paper mappings, we have selected AI, CE and SE departments as common departments, from which all department students can refer papers, in order to get similar papers. We have generated common papers from only from these 3 departments, so that similarity is analogous to real world scenarios.

Screenshot of code snippet to generate the mappings is:

```
#Mappings data

import random
mappings={}

for sid in range(1,len(student_data)+1):
    papers_referred = []
    if sid <= 20 :
        for j in range(random.randint(0,20)):
            r = random.randint(1,4991)
            if r not in papers_referred: papers_referred.append(r)
    if sid >= 21 and sid <= 40 :
        for j in range(random.randint(0,20)):
            r = random.randint(5001,8146)
            if r not in papers_referred: papers_referred.append(r)
    if sid >= 41 and sid <=60 :
        for j in range(random.randint(0,20)):
            r = random.randint(8156,13146)
            if r not in papers_referred: papers_referred.append(r)
    if sid >= 61 and sid <= 80 :
        for j in range(random.randint(0,20)):
            r = random.randint(13155,17590)
            if r not in papers_referred: papers_referred.append(r)
    if sid >= 81 and sid <= 100 :
        for j in range(random.randint(0,20)):
            r = random.randint(17591,22590)
            if r not in papers_referred: papers_referred.append(r)
```

After running this code, the data generated is:

```
#Getting 9 digit student id for each student
student_paper_mappings={}

for stid in range(len(student_data)):
    uniq_student_id = student_id_col.iloc[stid]
    papers_referred = mappings[stid+1]
    student_paper_mappings[uniq_student_id]=papers_referred

for key, value in student_paper_mappings.items():
    print ("student_id : ", key, "papers referred : ", value)
```

```
student_id :  621741617 papers referred :  [1039, 642, 3720, 1714, 1197, 3891, 4769, 704, 79, 282, 472, 3000, 139, 48
75, 3342, 973, 2643, 819, 4991, 8155]
student_id :  771796474 papers referred :  [2921, 2550, 3703, 3528, 8146, 8151]
student_id :  113004597 papers referred :  [2399, 3145, 1120, 1810, 3021, 206, 3641, 1751, 3024, 4327, 3929, 3661, 29
11, 364, 3736, 4346, 903, 1820, 19561
```

This data is also stored in a pickle file student_paper_mappings.pkl, so data uniformity is maintained.

After loading the pickle file, below is the screenshot of mappings dataframe:

	papers_referred	student_id
0	1039,642,3720,1714,1197,3891,4769,704,79,282,4...	621741617.0
1	2921,2550,3703,3528,8146,8151	771796474.0
2	2399,3145,1120,1810,3021,206,3641,1751,3024,43...	113004597.0
3	3695,2612,4816,3560,3533,3654,3525,335,1821,39...	711605921.0
4	2495,2739,3161,3635,2213,86,2257,3155	791937775.0
5	3039,4992	416245271.0
6	4173,133,3001,265,3768,4821,4984,3548,848,1829...	745317241.0
7	4403,3012,4359,3986,2295,2417,4477,822,4207,20...	234344310.0
8	4155,3460,2358,764,1961,4030,1477,2057,3370,8153	203622525.0
9	2354,4466,706,379,569,1287,2810,3952,3934,1893...	516072574.0

DATA ANALYZING AND CLEANING:

After scrapping the papers metadata, certain fields like affiliation, DOI numbers and reference numbers were missing. These attributes were not useful for recommendation. So, we dropped those columns.

```
1 papers_df.isnull().sum()
```

```
affiliation                 0
arxiv_id                    0
arxiv_primary_category      0
authors                     0
doi                     18917
journal_reference       18166
pdf_url                     0
published                   0
summary                     0
title                       0
pid                         0
paper_id                    0
dtype: int64
```

Also, the category after scraping the papers data is in the form: cs.AI, cs.CE, cs.SE, etc. So, we removed the first 3 letters in order to match the categories of papers with the departments of students. Sample screenshot:

```
1 papers_df['paper_id'] = papers_df['pid'].astype(str)+ str('_') + papers_df['arxiv_primary_category'].astype(str)
```

Before:

affiliation	arxiv_id	arxiv_primary_category	authors	doi	journal_reference	pdf_url	published	summary
None	http://arxiv.org/abs/cs/9308101v1	cs.AI	[M. L. Ginsberg]	None	Journal of Artificial Intelligence Research, V...	http://arxiv.org/pdf/cs/9308101v1	1993-08-01T00:00:00Z	Because of their occasional need to return to ...
None	http://arxiv.org/abs/cs/9308102v1	cs.AI	[M. P. Wellman]	None	Journal of Artificial Intelligence Research, V...	http://arxiv.org/pdf/cs/9308102v1	1993-08-01T00:00:00Z	Market price systems constitute a well-underst...
None	http://arxiv.org/abs/cs/9309101v1	cs.AI	[I. P. Gent, T. Walsh]	None	Journal of Artificial Intelligence Research, V...	http://arxiv.org/pdf/cs/9309101v1	1993-09-01T00:00:00Z	We describe an extensive study of search in GS...

After:

affiliation	arxiv_id	arxiv_primary_category	authors	doi	journal_reference	pdf_url	published	summary
None	http://arxiv.org/abs/cs/9308101v1	AI	[M. L. Ginsberg]	None	Journal of Artificial Intelligence Research, V...	http://arxiv.org/pdf/cs/9308101v1	1993-08-01T00:00:00Z	Because of their occasional need to return to ...
None	http://arxiv.org/abs/cs/9308102v1	AI	[M. P. Wellman]	None	Journal of Artificial Intelligence Research, V...	http://arxiv.org/pdf/cs/9308102v1	1993-08-01T00:00:00Z	Market price systems constitute a well-underst...
None	http://arxiv.org/abs/cs/9309101v1	AI	[I. P. Gent, T. Walsh]	None	Journal of Artificial Intelligence Research, V...	http://arxiv.org/pdf/cs/9309101v1	1993-09-01T00:00:00Z	We describe an extensive study of search in GS...

We also standardized data into uniform datatypes, like converting floating point numbers into integers.

```
1  student_paper_df['student_id'] = student_paper_df['student_id'].astype(int)
```

```
1  student_paper_df.head(50)
```

Before

papers_referred	student_id
1039,642,3720,1714,1197,3891,4769,704,79,282,4...	621741617.0
2921,2550,3703,3528,8146,8151	771796474.0
2399,3145,1120,1810,3021,206,3641,1751,3024,43...	113004597.0
3695,2612,4816,3560,3533,3654,3525,335,1821,39...	711605921.0
2495,2739,3161,3635,2213,86,2257,3155	791937775.0

After

papers_referred	student_id
1039,642,3720,1714,1197,3891,4769,704,79,282,4...	621741617
2921,2550,3703,3528,8146,8151	771796474
2399,3145,1120,1810,3021,206,3641,1751,3024,43...	113004597
3695,2612,4816,3560,3533,3654,3525,335,1821,39...	711605921
2495,2739,3161,3635,2213,86,2257,3155	791937775

DATA PRE-PROCESSING:

As a part of data preprocessing, we have removed the stopwords from summary field. We have used TF-IDF vectorizer for calculating most and least occurring words and calculated weightage of each term in the title and summary feileds. We have also applied stemming to remove the suffixes and prefixes of words in order to get them in root form for highest keyword inference. Below are screenshots of all the preprocessing steps:

1. Stopwords removal:

```
def tokenize_summary(summary):
    summ = [char for char in summary if char not in string.punctuation]
    summ = ''.join(summ)

    return [word for word in summ.split() if word.lower() not in stopwords.words('english')]

print(tokenize_summary(summary[0]))
```

```
['occasional', 'need', 'return', 'shallow', 'points', 'search', 'tree', 'existing', 'backtracking', 'methods', 'some
times', 'erase', 'meaningful', 'progress', 'toward', 'solving', 'search', 'problem', 'paper', 'present', 'method', '
backtrack', 'points', 'moved', 'deeper', 'search', 'space', 'thereby', 'avoiding', 'difficulty', 'technique', 'devel
oped', 'variant', 'dependency', 'directed', 'backtracking', 'uses', 'polynomial', 'space', 'still', 'providing', 'us
eful', 'control', 'information', 'retaining', 'completeness', 'guarantees', 'provided', 'earlier', 'approaches']
```

2. Applying TF-IDF vectorizer:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(tokenizer=lambda doc: doc,lowercase=False,stop_words='english', use_idf=False, norm='l1')
Y = vectorizer.fit_transform(summary_tokens)
```

```
print(vectorizer.get_feature_names())
```

```
uct', 'abductivebas', 'abil', 'abl', 'abnorm', 'abo', 'abovement', 'abox', 'abrupt', 'absenc', 'absolut', 'absorb', '
absorpt', 'abstract', 'abstractli', 'abund', 'ac3', 'academ', 'academia', 'acceler', 'accept', 'access', 'accid', 'ac
cident', 'accommod', 'accompani', 'accomplish', 'accord', 'accordingli', 'account', 'accumul', 'accur', 'accuraci', '
achiev', 'achil', 'aclassif', 'aclp', 'acoust', 'acquir', 'acquisit', 'acquisiton', 'acr', 'act', 'action', 'actionsr
ecommend', 'activ', 'activatewithcutset', 'activemath', 'activitybas', 'actor', 'actorcrit', 'actual', 'actuat', 'acu
t', 'acycl', 'ad', 'adaboost', 'adam', 'adapt', 'add', 'addit', 'address', 'addresse', 'adequ', 'adequaci', 'adher',
'adhoc', 'adjac', 'adjust', 'adl', 'admiss', 'admit', 'adopt', 'adress', 'adtre', 'advanc', 'advantag', 'advent', 'ad
vers', 'advertis', 'advic', 'advoc', 'ae', 'affect', 'affin', 'afflict', 'aforement', 'african', 'afterward', 'agap',
'age', 'agenc', 'agenda', 'agent', 'agent0', 'agentbas', 'agentenviron', 'agenthuman', 'agentori', 'agglom', 'agglome
r', 'aggreg', 'agm', 'ago', 'agre', 'agreement', 'agreementbas', 'agricultur', 'agronom', 'agronomist', 'agronomistss
pati', 'ahc', 'ahead', 'ahmm', 'ahp', 'ai', 'aibrain', 'aid', 'aiddecis', 'aiframework', 'aij', 'aim', 'aip', 'aips02
', 'aips2000', 'aips98', 'aipsych', 'air', 'airborn', 'aircraft', 'airlin', 'airport', 'airspac', 'airtraff', 'aisbn'
, 'aistyl', 'aisystem', 'aityp', 'aka', 'akin', 'al', 'alarm', 'albeit', 'alc', 'alcnr', 'alcnrknowledg', 'alcohol',
'alcq', 'alcqi', 'alert', 'alfer', 'algebra', 'algorithm', 'algorithmproblem', 'alif', 'align', 'alik', 'alldiffer',
'alldiffpreced', 'allen', 'allevi', 'alli', 'alloc', 'allot', 'allow', 'alloy', 'allpair', 'alm', 'alo', 'alon', 'alo
ngsid', 'alp', 'alpha', 'alpha1', 'alpha1beta1veeve', 'alpha2', 'alphabet', 'alphabeta', 'alphad', 'alphadiscount', '
alphaibetaixi', 'alphain', 'alphap', 'alq', 'alreadi', 'altalt', 'altaltp', 'alter', 'altern', 'alternationfre', 'alt
ogeth', 'alu', 'alvi', 'alway', 'alzheim', 'amalgam', 'ambigu', 'ambiti', 'amd', 'amelior', 'amen', 'american', 'amie
n', 'amort', 'amplif', 'amplifi', 'amplitud', 'analog', 'analogu', 'analys', 'analysi', 'analyst', 'analyt', 'analyz'
```

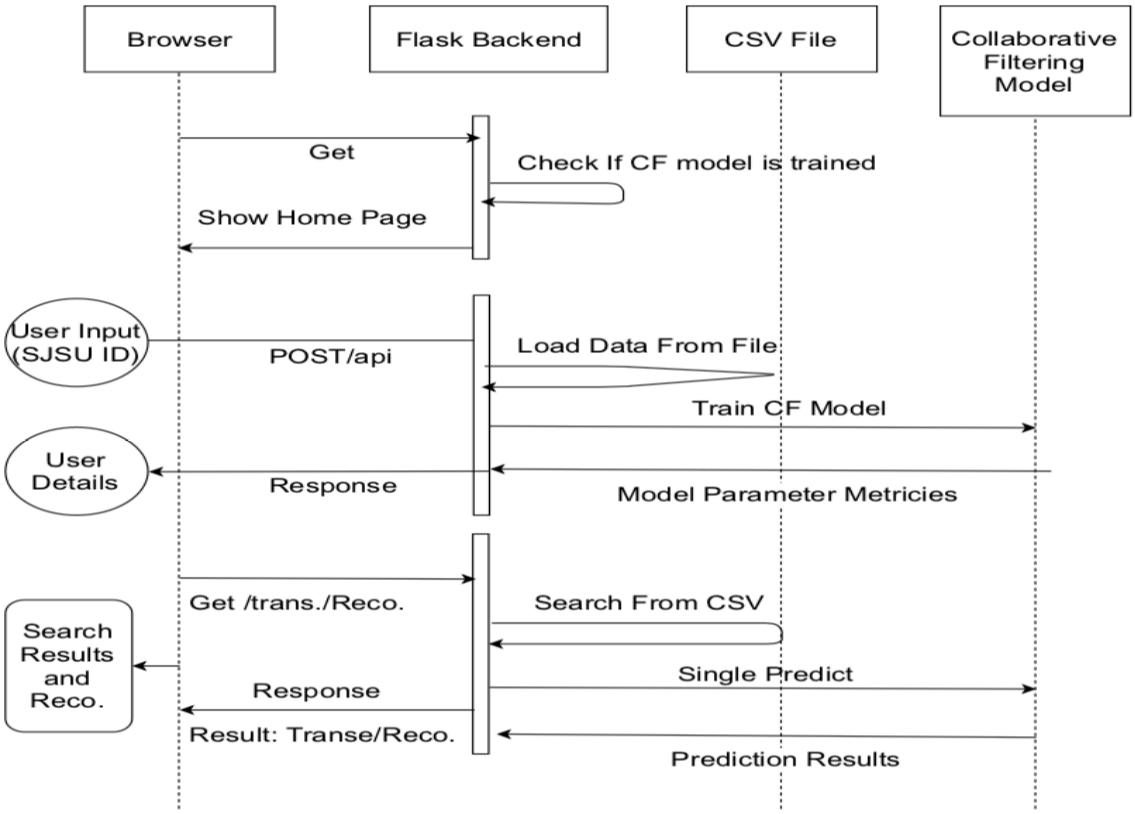
3. Applying stemming:

```
    word = [word for word in summ.split() if word.lower() not in stopwords.words('english')]
    return[porter_stemmer.stem(word) for word in word]
print(tokenize_summary(summary[999]))
```

```
['paper', 'confront', 'problem', 'appli', 'reinforc', 'learn', 'agent', 'perceiv', 'environ', 'mani', 'sensor', 'perf
orm', 'parallel', 'action', 'use', 'mani', 'actuat', 'case', 'complex', 'autonom', 'robot', 'argu', 'reinforc', 'lear
n', 'success', 'appli', 'case', 'strong', 'assumpt', 'made', 'characterist', 'environ', 'learn', 'perform', 'relev',
'sensor', 'read', 'motor', 'command', 'readili', 'identifi', 'introduct', 'assumpt', 'lead', 'stronglybias', 'learn',
'system', 'eventu', 'lose', 'gener', 'tradit', 'reinforcementlearn', 'algorithm', 'line', 'observ', 'realist', 'situa
t', 'reward', 'receiv', 'robot', 'depend', 'reduc', 'subset', 'execut', 'action', 'reduc', 'subset', 'sensor', 'input
', 'possibl', 'differ', 'situat', 'action', 'relev', 'predict', 'reward', 'formal', 'properti', 'call', 'categoriz',
'assumpt', 'present', 'algorithm', 'take', 'advantag', 'categoriz', 'environ', 'allow', 'decreas', 'learn', 'time', '
respect', 'exist', 'reinforcementlearn', 'algorithm', 'result', 'applic', 'algorithm', 'coupl', 'simul', 'realisticro
bot', 'problem', 'landmarkbas', 'navig', 'sixleg', 'robot', 'gait', 'gener', 'report', 'valid', 'approach', 'compar',
'exist', 'flat', 'generalizationbas', 'reinforcementlearn', 'approach']
```

APPROACHES:

In order to get the most accurate results, we implemented various approaches. Below is the workflow that we followed:



1. Content-Based Filtering Recommendations:

Content-based filtering, also referred to as 'Cognitive filtering', recommends items based on a comparison between the description of the item and a user profile. This approach is useful when there are enough number of features that can describe an item, in this case, a research paper. Typically, the words that occur in a document are good descriptors for papers. Since we had title and summary columns, we have implemented this approach. All the words in the titles mostly occur in their summaries. So, we have only considered summary column of all the papers to apply content-based filtering. The summary of each paper is then tokenized in the form of words to extract the keywords. This word tokenization is explained below.

a. Word Tokenization:

Each summary consists of more than 500 words and it is difficult to build a recommendation system as it takes a lot of time to execute for 25000 papers. The summary of each paper consists of many stop words, punctuation marks, etc. Stop words are redundant words which are unnecessary for recommendation. So, the data is preprocessed by removing all the stop words and

punctuation marks from the summary using natural language toolkit. Example of a preprocessed summary after removing stop words and punctuation marks is shown below.

```
import string
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
porter_stemmer = PorterStemmer()

def tokenize_summary(summary):
    summ = [char for char in summary if char not in string.punctuation]
    summ = ''.join(summ)

    return [word for word in summ.split() if word.lower() not in stopwords.words('english')]
print(tokenize_summary(summary[999]))
```

However, some of the words in the preprocessed data are repetitive. For example, the word algorithm and algorithms. So, the words are reduced to their root words by performing stemming on the data. This is done by importing porter stemmer using natural language processing. Sample screenshot of summary after performing stemming is shown below.

```
import string
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
porter_stemmer = PorterStemmer()

def tokenize_summary(summary):
    summ = [char for char in summary if char not in string.punctuation]
    summ = ''.join(summ)

    word=[word for word in summ.split() if word.lower() not in stopwords.words('english')]
    return[porter_stemmer.stem(word) for word in word]

print(tokenize_summary(summary[1]))
```

```
['market', 'price', 'system', 'constitut', 'wellunderstood', 'class', 'mechan', 'certain', 'co
t', 'decentr', 'decis', 'make', 'minim', 'commun', 'overhead', 'marketori', 'program', 'approa
lem', 'solv', 'deriv', 'activ', 'resourc', 'alloc', 'set', 'comput', 'agent', 'comput', 'compe
rtifici', 'economi', 'walra', 'provid', 'basic', 'construct', 'defin', 'comput', 'market', 'st
eriv', 'correspond', 'price', 'equilibria', 'particular', 'realiz', 'approach', 'form', 'multi
em', 'see', 'care', 'construct', 'decis', 'process', 'accord', 'econom', 'principl', 'lead', '
resourc', 'alloc', 'behavior', 'system', 'meaning', 'analyz', 'econom', 'term']
```

b. TF-IDF:

The data after performing stemming and removing stop words results in the tokenized words which can be used for recommendation. For this, we have applied TF-IDF vectorizer to convert all the words in vectors. TF-IDF stands for term frequency-inverse document frequency, and the TF-IDF weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the TF-IDF weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

We have used Sci-Kit Learn library for TF-IDF vectorizer to get the feature names for all the papers. Feature names are the keywords in the summary part of the paper. After converting words into vectors, we have calculated cosine similarity between the papers, using the formula:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

Below is the screenshot of dataframe after calculating TF-IDF for summary column of each paper:

```
import pandas as pd
pd.DataFrame(Y.toarray(),columns=vectorizer.get_feature_names())
```

	0	000	01	02pi	073	080	090	092	095	0approxim	...	zeno	zero	zerocross	zhang	zhao	zhu	zlifo	zone	zoom	zugzwang
0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Below is the example showing cosine similarity between paper 1 and all remaining papers:

```
from sklearn.metrics.pairwise import cosine_similarity
similarity = (cosine_similarity(Y[0:1], Y))
```

```
flattened_list = [y for x in similarity for y in x]
```

```
similarity_sorted=sorted(flattened_list,reverse=True)
print(similarity_sorted)
```

```
[0.9999999999999998, 0.302792296062416, 0.2606951911021698, 0.24696429696323385,
2338, 0.2102190826944047, 0.20043980267565392, 0.18182764732927534, 0.1790586610
250281817, 0.16662398249085253, 0.16458837276872187, 0.15171471894408806, 0.1404
13073138494711317, 0.1304753109547856, 0.129331376957014, 0.12588243615368366, 0.
97, 0.11282990734440268, 0.11204748774969808, 0.11134957993143405, 0.11063174185
7502334795, 0.1085657907014351, 0.10497997500424747, 0.10461329882571975, 0.1039
```

Similarly, the user query is also vectorized in the form of TF-IDF vector. Cosine similarity is used to find the similarity between the user query and the keywords in the paper. This similarity is then sorted in descending order and top 10 papers with highest similarity scores are calculated. We have stored these recommendation results in a CSV file. Below is the function to find the index and similarity score of the query vector and the TF-IDF vector for all the papers in the data. The top 10 recommendations are also shown:

```
def find_similar(tfidf_matrix, index, top_n = 10):
    cosine_similarities = cosine_similarity(Y, tfidf_matrix).flatten()
    related_docs_indices = [i for i in cosine_similarities.argsort()[::-1] if i != index]
    return [(index, cosine_similarities[index]) for index in related_docs_indices][0:top_n]
```

```
find_similar(Z,1,10)
```

```
[(1352, 0.22444918745810566),
 (1309, 0.15387485618504526),
 (1744, 0.09215785317293737),
 (131, 0.09059282070553476),
 (41, 0.0874780245066129),
 (1308, 0.07882219702547631),
 (130, 0.07426812326231409),
 (1668, 0.07156792630093642),
 (1669, 0.06582107431391004),
 (771, 0.054169332929043475)]
```

Here, the function has three arguments, one is a TF-IDF vector of user query, second one is index or ID of the target paper and the last argument is number of recommendations to get. This function only returns the IDs of top N similar papers and not the titles. So, in order to get the titles, we are collecting the titles of all the papers along with the IDs and similarity score. Below is the example of top 10 recommendations when the user query is 'neural network':

```
query1= ['Neural Network']
```

```
query = tokenize_summary(query1)
print(query)
```

```
['neural', 'network']
```

```
corpus = []
for i in data["title"]:
    corpus.append(i)  #Corpus is the list of all the titles of papers in the data.
```

```
for index,score in find_similar(Z,1,10):
    print(score,corpus[index])
```

```
0.22444918745810566 Node Splitting: A Scheme for Generating Upper Bounds in Bayesian
  Networks
0.15387485618504526 Fuzzy Knowledge Representation Based on Possibilistic and Necessary
  Bayesian Networks
0.09215785317293737 On Quantified Linguistic Approximation
0.09059282070553476 Fages' Theorem and Answer Set Programming
0.0874780245066129 Generalization of Clauses under Implication
0.07882219702547631 The Causal Topography of Cognition
0.07426812326231409 DES: a Challenge Problem for Nonmonotonic Reasoning Systems
0.07156792630093642 Vector-space Analysis of Belief-state Approximation for POMDPs
0.06582107431391004 Value-Directed Sampling Methods for POMDPs
0.054169332929043475 Scientific Collaborations: principles of WikiBridge Design
```

2. Collaborative Filtering Recommendations:

In this approach, we have collected papers referred by a set of similar users. Collaborative approach is useful when there is enough user information about the likes and dislikes for an item. In this case, we had students that have referred papers in the past. We have considered this as implicit rating and built student to paper profile to implement both user-based and item-based recommendation systems:

a. User-Based Collaborative Filtering:

In this approach, we have calculated similar users based on the research papers they have referred. We have implemented a mathematical formula to calculate distance between each the target user and remaining users. Top 10 nearest neighbors are considered. Finally, papers referred by the most similar users are recommended to the target user.

Below is the code snippet to calculate similarity between the target user and other users and also the common papers which they have referred:

```
[ ]  # Arrays to maintain the most matched user, and no. of common papers
     match=[0]*60
     count=[0]*60
     # for i, val in enumerate(arr):
     #     print(i, val)
     for (index1,it2) in enumerate(arr):
         #print(it2)
         for(index2,it2) in enumerate(it2):
             #print(index1,index2)
             a = len(arr[index1][index2])
             #print(len(arr[index1][index2]))
             if(count[index1]<len(arr[index1][index2])):
                 match[index1]=index2
                 count[index1]=len(arr[index1][index2])
     print("user_id","matched_user_id","no. of matches")
     for (index,i) in enumerate(match):
         print(index,match[index],count[index])
```

```
user_id matched_user_id no. of matches
0 3 1
1 11 1
2 12 1
```

b. Item-based Collaborative Filtering:

In this approach, cosine similarity between the common papers is calculated using TF-IDF vectorizer. The top 10 similar papers from similar users, which are not referred by the target user are selected for recommendation.

Below is the code snippet to calculate similarity between the target paper and other papers:

```
        with open('papers.pkl', 'rb') as f:
            data_papers = pickle.load(f)
        uid=list(data_ref.student_id).index(uid)
        s=(set(matrix[match[uid]][0])-set(matrix[uid][0]))
        l=list(s)
        cleanedList = [x for x in l if str(x) != 'nan']
        cl = [int(i) for i in cleanedList]
        list_papers=[]
        with open('user_rec.csv','w') as csvfile:
            fieldnames=['paper_id','title','category']
            writer=csv.DictWriter(csvfile,fieldnames=fieldnames)
            writer.writeheader()
            for i in cl:
                writer.writerow({'paper_id':i,'title':data_papers.title[i],'category':data_papers.arxiv_primary_category[i]})
                list_papers.append(i)
                #print("User ID ",uid)
                print('paper_id ',i,'title ',data_papers.title[i],'category ',data_papers.arxiv_primary_category[i])

tocsv(621741617)
```

Below is the screenshot of recommendations after collaborative filtering:

```
tocsv(621741617)
```

```
paper_id  10724 title  CrashScope: A Practical Tool for Automated Testing of Android
  Applications category  SE
paper_id  9503 title  Hierarchical Variability Modeling for Software Architectures category  SE
paper_id  10156 title  Software Architecture Decision-Making Practices and Challenges: An
  Industrial Case Study category  SE
paper_id  8312 title  EuSpRIG 2006 Commercial Spreadsheet Review category  SE
paper_id  8844 title  Ontology for Mobile Phone Operating Systems category  SE
paper_id  11909 title  The Wall and The Ball: A Study of Domain Referent Spreadsheet Errors category  HC
paper_id  12333 title  HTTP Mailbox - Asynchronous RESTful Communication category  SE
paper_id  8823 title  Towards a better understanding of testing if conditionals category  SE
paper_id  8809 title  Examining the Impact of Platform Properties on Quality Attributes category  SE
paper_id  8423 title  Software Components for Web Services category  SE
paper_id  9784 title  Service-Oriented Architecture in Industrial Automation Systems - The
  case of IEC 61499: A Review category  SE
```

3. Pipelined Hybrid Approach:

We have designed a hybrid recommendation system using pipeline technique. We found the most similar users for a target user using user-based collaborative filtering. Then we applied content-based filtering on the recommendations obtained after user-based recommendations. We passed the list of recommended papers using collaborative filtering along with user ID to get TF-IDF scores for the list of papers and sorted them in descending order to get the topmost 10 recommendations.

Below is the code snippet for pipelined hybrid approach:

```
def tocsv_hybrid(uid):
    import pickle
    import csv
    with open('student_paper_mappings.pkl', 'rb') as f:
        data_ref = pickle.load(f)
        matrix=data_ref.as_matrix()
        for index,i in enumerate(matrix):
            matrix[index][0]=matrix[index][0].split(',')

        arr=[]
        for (index,it1) in enumerate(matrix):
            arr.append([])
            for it2 in matrix:

                it1_set = set(it1[0])
                it2_set = set(it2[0])
                arr[index].append(it1_set & it2_set)
        # set (i,i ) to empty set before finding most matched user
        for (index,it) in enumerate(arr):
            arr[index][index]=set()
        # Arrays to maintain the most matched user, and no. of common papers
        match=[0]*120
        count=[0]*120
        # for i, val in enumerate(arr):
        #     print(i, val)
        for (index1,it2) in enumerate(arr):
            #print(it2)
```

The recommendations which we got after running the pipelined hybrid approach for the user ID 621741617 are:

```
61 tocsv_hybrid(621741617)
62
63
```

```
[11909, 10724, 8809, 8312, 8844, 9784, 8423, 8823, 12333, 9503, 10156]
paper_id  11909 title  The Wall and The Ball: A Study of Domain Referent Spreadsheet Errors category  HC
paper_id  10724 title  CrashScope: A Practical Tool for Automated Testing of Android
  Applications category  SE
paper_id  8809 title  Examining the Impact of Platform Properties on Quality Attributes category  SE
paper_id  8312 title  EuSpRIG 2006 Commercial Spreadsheet Review category  SE
paper_id  8844 title  Ontology for Mobile Phone Operating Systems category  SE
paper_id  9784 title  Service-Oriented Architecture in Industrial Automation Systems - The
  case of IEC 61499: A Review category  SE
paper_id  8423 title  Software Components for Web Services category  SE
paper_id  8823 title  Towards a better understanding of testing if conditionals category  SE
paper_id  12333 title  HTTP Mailbox - Asynchronous RESTful Communication category  SE
paper_id  9503 title  Hierarchical Variability Modeling for Software Architectures category  SE
paper_id  10156 title  Software Architecture Decision-Making Practices and Challenges: An
  Industrial Case Study category  SE
```

4. Deep Learning Approach:

This type of approach can be useful when there is sufficient number of both papers (items) and students (users). So, data can be split into train and test datasets and the model can learn from the historic data. We have implemented deep learning model, where we have used Doc2Vec model from Gensim library. We have divided the dataset in 80:20 ratio of train : test datasets and ran for 100 epochs. Below is the code snippet showing 5/100 epochs:

```
#Initialize model with params :
#     dm = 1 .. to use distributed memory training algo
#.    min_count=1 .. ignore words lower than this frequency
#     window = 25 length of window size
#     workers = 5 .. threads for training
model = Doc2Vec(dm =1,window=25,alpha=0.025, min_alpha=0.00025,min_count=1,workers=5)
model.build_vocab(processed_data)
```

```
#train model
for i in range(100):
    print("Training epoch : ", i+1)
    model.train(processed_data, total_examples=model.corpus_count,epochs=1)

model.save('nlpmodel')
```

```
Training epoch :  1
Training epoch :  2
Training epoch :  3
Training epoch :  4
Training epoch :  5
```

We have tested this model with title of one research paper and below are the most similar papers using deep learning approach:

```
rec_model = Doc2Vec.load('nlpmodel')
test_text = 'Because of their occasional need to return to shallow points in a search\ntree, existing backtracking
test_title = 'Dynamic Backtracking'
test_category = 'AI'
test_vector = rec_model.infer_vector([test_text.lower()])
similar_papers = rec_model.docvecs.most_similar([test_vector], topn = 10)
print(similar_papers)
```

```
[('2297', 0.3084923028945923), ('6470', 0.3019839823246002), ('11096', 0.3008095920085907), ('8206', 0.28329569101333
62), ('1081', 0.28075459599494934), ('14552', 0.2792961597442627), ('1380', 0.2749897241592407), ('6358', 0.273240059
6141815), ('2922', 0.2723822593688965), ('2674', 0.2711024284362793)]
```

Below are the recommendations obtained for this test datapoint:

```
print("Recommended papers are : ")
for i in similar_papers:
    paperid = int(i[0])
    print(paperid, category_data [paperid-1], title_data[paperid-1])
```

```
Recommended papers are :
323 AI Neuronal Spectral Analysis of EEG and Expert Knowledge Integration for
  Automatic Classification of Sleep Stages
3765 AI A computer program for simulating time travel and a possible 'solution'
  for the grandfather paradox
11419 SE Replication Can Improve Prior Results: A GitHub Study of Pull Request
  Acceptance
20767 NI DAWN: Delay-Aware Wi-Fi Offloading and Network Selection
9714 SE Proceedings 12th International Workshop on Formal Engineering approaches
  to Software Components and Architectures
5396 CE Three-dimensional nonlinear micro/meso-mechanical response of the
  fibre-reinforced polymer composites
16 AI A System for Induction of Oblique Decision Trees
1057 AI A Combinatorial Optimisation Approach to Designing Dual-Parented
  Long-Reach Passive Optical Networks
4141 AI Learning to Schedule Deadline- and Operator-Sensitive Tasks
3677 AI Automatic Bridge Bidding Using Deep Reinforcement Learning
```

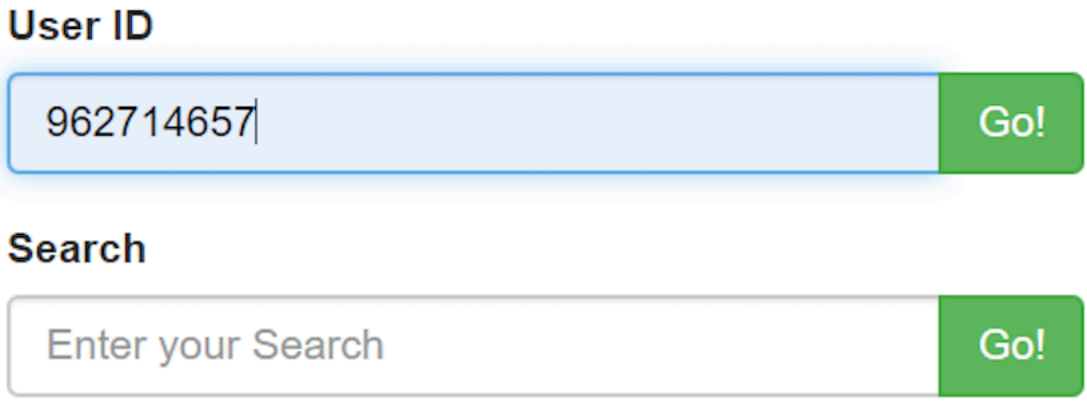
Finally, we have stored these results in a CSV file to evaluate the model performance.

Below is the screenshot of CSV file generated:

	A	B	C
1	PID	Category	Title
2	323	AI	Neuronal Spectral Analysis of EEG and Expert Knowledge
3	3765	AI	A computer program for simulating time travel and a possible
4	11419	SE	Replication Can Improve Prior Results: A GitHub Study of Pull
5	20767	NI	DAWN: Delay-Aware Wi-Fi Offloading and Network Selection
6	9714	SE	Proceedings 12th International Workshop on Formal Engineering
7	5396	CE	Three-dimensional nonlinear micro/meso-mechanical response
8	16	AI	A System for Induction of Oblique Decision Trees
9	1057	AI	A Combinatorial Optimisation Approach to Designing Dual-
10	4141	AI	Learning to Schedule Deadline- and Operator-Sensitive Tasks
11	3677	AI	Automatic Bridge Bidding Using Deep Reinforcement Learning

TESTING AND EVALUATION:

We have designed a UI using Flask frame work. Below is the screenshot of website than opens when we run the code:



We have used render_template for rendering 'GET' and 'POST' HTTP requests. We have used request.form() for receiving the request form user. User can enter the 9-digit SJSU ID (User ID) and the keywords for searching the research papers to get top 10 results from recommendations. There are two options on the website:

1. Search using both SJSU ID and keywords:

If the user is already present in the database, then this recommender system will return top 10 recommendations considering the preferences of user and also its peers.

Below is the screenshot of recommendations when user is present in the database:

Please Click on Find Recommendation to get the search result for UserId -962714657

Find Recommendation

paper_id	title	category
7328	"Dynamical Models of Stock Prices Based on Technical Trading Rules Part I: The Models"	in.TR
7062	"Numerical extraction of a macroscopic pde and a lifting operator from a lattice Boltzmann model"	CE
5818	Hospital Case Cost Estimates Modelling - Algorithm Comparison	CE

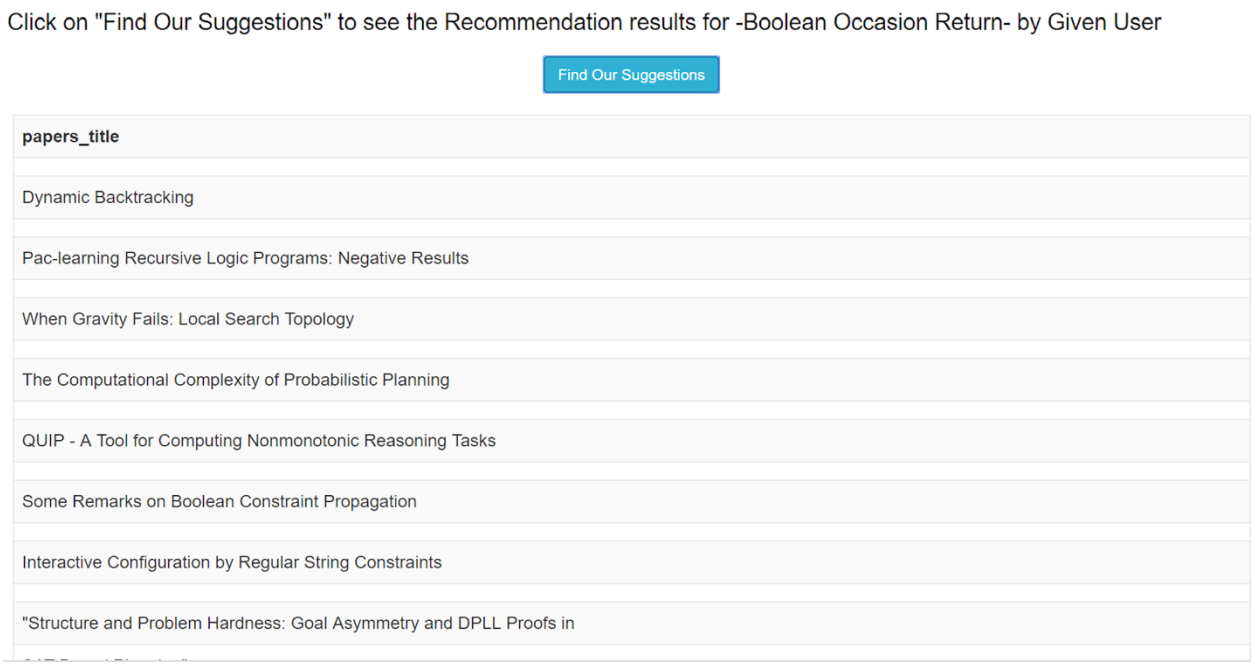
This result is also stored in a CSV file, as shown:

A	B	C
paper_id	title	category
7328	Dynamical Models of Stock Prices Based on Technical Trading	in.TR
7062	Numerical extraction of a macroscopic pde and a lifting operator	CE
5818	Hospital Case Cost Estimates Modelling - Algorithm Comparison	CE
5593	Encoding Candlesticks as Images for Patterns Classification Using	CE
5891	Efficient Dealiased Convolutions without Padding	CE
5441	Optimal prevention with possibilistic and mixed background risk	CE

2. Search using only keywords:

If the user is new and does not have an entry in the database, then this recommender system will return top 10 recommendations considering the keywords entered by the users.

Below is the screenshot of recommendations when user is not present in the database:



This result is also stored in a CSV file, as shown:

	A
1	papers_title
2	
3	Dynamic Backtracking
4	Pac-learning Recursive Logic Programs: Negative Results
5	When Gravity Fails: Local Search Topology
6	The Computational Complexity of Probabilistic Planning
7	QUIP - A Tool for Computing Nonmonotonic Reasoning Tasks
8	Some Remarks on Boolean Constraint Propagation
9	Interactive Configuration by Regular String Constraints
10	Structure and Problem Hardness: Goal Asymmetry and DPLL
11	Ensemble Learning for Free with Evolutionary Algorithms ?
12	Learning to Bluff
13	

Libraries used:

- Natural Language Processing Kit (NLTK)
- Word Tokenizer and Sentence Tokenizer from NLTK
- Porter_Stemmer from NLTK
- Word2Vec using Bag-of-Words or N-grams model
- Doc2Vec model from Gensim library
- Flask for implementation and evaluation
- Render_template to create HTTP requests
- Request.form to receive data from user

RESULTS:

By using hybrid approach, the recommendation results are more diverse and refined. The cold-start problem where item is new and not many ratings are available can be solved using content-based filtering, as the features preferred by the users can be matched with that item. Similarly, using deep learning approaches there is an increased possibility of recommending new items due to learning of hidden, implicit features by the model. In conclusion, combining multiple recommendation techniques together and applying machine learning based on the historic data is a better way to provide relevant and useful recommendations.

REFERENCES:

[1] Haruna, Khalid, and Maizatul Akmar Ismail. "Research paper recommender system evaluation using collaborative filtering." AIP Conference Proceedings. Vol. 1974. No. 1. AIP Publishing, 2018.
https://aip.scitation.org/doi/10.1063/1.5041583

[2] W. Waheed, M. Imran, B. Raza, A. K. Malik and H. A. Khattak, "A Hybrid Approach Toward Research Paper Recommendation Using Centrality Measures and Author Ranking," in *IEEE Access*, vol. 7, pp. 33145-33158, 2019. doi: 10.1109/ACCESS.2019.2900520. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8654587&isnumber=8600701

[3] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," Knowledge-Based Systems, vol. 46, pp. 109–132, 2013. https://www.sciencedirect.com/science/article/abs/pii/S0950705113001044?via%3Dihub