

LOVELY PROFESSIONAL UNIVERSITY

ARTIFICIAL INTELLIGENCE

COURSE CODE: INT404

CLASS ASSIGNMENT

TOPIC: MOVIES RECOMMENDED SYSTEM

SUBMITTED TO: Dr. SUKHVIR KAUR

Section: K18TS

Submitted by:

|  |  |  |  |
| --- | --- | --- | --- |
| Sl.no | NAME | REGISTRATION NO | ROLL NUMBER |
| 1. | D BALAJI | 11805130 | 53 |
| 2. | PRIYAL SHARMA | 11813747 | 35 |
| 3. | SAHEEL KUMAR DAS | 11801350 | 55 |
| 4. | MOGALAPALLI NISCHAL | 11805850 | 58 |

GitHub link: https://github.com/Balaji02D/Arificail-Intelligence.git

Context:

1. Introduction
2. Methodology selected
3. Project description
4. Code explanation
5. Implementation of the testing algorithm
6. Description on each member participation
7. Interactive or the graphical results of the project
8. Testing of the project
9. Audios, Videos to support the project
10. Conclusion
11. Sources
12. INTRODUCTION

In this project, a movie recommendation system is built based on the certain dataset. We used collaborative filtering method to predict user’s movie rating and we can recommend movies to customers, which they potentially give high ratings according to prediction.

Recommendation systems are used to provide personalized recommendations according to user profile and previous search patterns. Recommendation systems are widely used in the Internet Industry.

Services like Hot star, Amazon Prime Video, Netflix, and YouTube are typical examples of recommendation system users. Recommendation systems not only help the users find their favourite products, but also bring potential profit to online service providers. Recommendation systems are highly used in knowing the popularity of the films. These systems can help in predicting the high rated films and they also suggest the films based upon the names of the actors and also based on their characters in those films. So, we can get the best results with the help of these systems. There are widely used in the cinema industry to notice the better outputs.

Collaborative filtering makes predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption is that if a user A has the same opinion as a user B on an issue, A is more likely to have B's opinion on a different issue x than to have the opinion on x of a user chosen randomly.

Recommender systems aim to solve this problem by taking in a user’s past actions, such as articles they’ve read or products they’ve purchased and rated, to identify potential user preferences. Instead of providing a generic experience to every user, recommender systems personalize the experience of each user by surfacing content that is particularly relevant to their observed interests.

we will focus on neighbourhood-based collaborative filtering methods, which are a well-known technique used in recommender systems. Neighbourhood-based collaborative filtering methods are item-based, meaning user preferences are inferred solely from what items the they and other users in the dataset have interacted with and weighted sum approach to fill the missing rating values and Cosine similarity score to identify the nearest neighbours and Top movie recommendations.

It is often referred to as recommender systems, a simple algorithm that aims to provide relevant and accurate information to users by filtering useful information from large data sets. The recommender system discovers information patterns in the data set by learning about consumer choices and generating results relevant to their needs and interests.

MODULES USED IN THE PROJECT: -

Flask:

Flask is a web development module in python similar to Django.

Flask is considered more [Pythonic](http://blog.startifact.com/posts/older/what-is-pythonic.html) than the [Django](https://www.fullstackpython.com/django.html) web framework because in common situations the equivalent Flask web application is more explicit. Flask is also easy to get started with as a beginner because there is little boilerplate code for getting a simple app up and running.

Sample code using Flask:

from flask import Flask

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

@app.route('/')

def hello\_world():

return 'Hello, World!'

#It displays “Hello World” on a web page!

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

app.run() # this method is used to run the code

**Why Flask?**

Python has a number of web frameworks that can be used to create web apps and APIs. The most well-known is Django, a framework that has a set project structure and which includes many built-in tools. This can save time and effort for experienced programmers, but can be overwhelming. Flask applications tend to be written on a blank canvas, so to speak, and so are more suited to a contained application such as our prototype API.

Flask one of the most popular frameworks available. It is used to create interactive APIs (Application Program Interface). It’s easy to implement and easy to understand as well.

It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

Flask has become popular among Python enthusiasts. As of January 2020, it has more stars on GitHub.com than any other Python web-development framework, and was voted the most popular web framework in the Python Developers Survey 2018.

Flask Table:

It is a part of Flask framework used to create Tables or setup a database.

This cud be implemented with HTML as well but Flask table is much easier and less complex to write also HTML is fiddly and all of your tables are basically the same. Flask table is also much more customizable than HTML.

Sample code using flask table:

from flask table import Table,Col

class ItemTable(Table):

name = Col(“Name”)

desc = Col(“Description”)

class Item(object):

def \_\_init\_\_(self, name, desc):

self.name = name

self.desc = desc

items = [Item(‘name1’ , ‘description1’),

Item(‘name2’ , ‘description2’),

Item(‘name3’ , ‘description3’)]

items = [dict(name = ‘name1’, desc = ‘description1’),

dict(name = ‘name2’, desc = ‘description2’),

dict(name = ‘name3’, desc = ‘description3’)]

items = ItemMode1.query.all()

# the above line loads the content from list

table = ItemTable(items)

print(table.\_\_html\_\_())

Sci-kit Learn / sklearn:

sklearn is a Python module integrating classical machine learning algorithms in the tightly-knit world of scientific Python packages (NumPy, SciPy, matplotlib).

It aims to provide simple and efficient solutions to learning problems that are accessible to everybody and reusable in various contexts: machine-learning as a versatile tool for science and engineering.

* Simple and efficient tools for predictive data analysis
* Accessible to everybody, and reusable in various contexts
* Built on NumPy, SciPy, and matplotlib
* Open source, commercially usable - BSD license
* Scikit-learn 0.20 was the last version to support Python 2.7 and Python 3.4. scikit-learn 0.21 and later require Python 3.5 or newer.
* Scikit-learn plotting capabilities (i.e., functions start with plot\_ and classes end with “Display”) require Matplotlib (>= 1.5.1). For running the examples Matplotlib >= 1.5.1 is required. A few examples require scikit-image >= 0.12.3, a few examples require pandas >= 0.18.0.

Installing it is really easy we just have to run the following command-

Pip install scikit-learn

Pandas:

Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on the top of python programming language.

Its most important feature is Series and Data Frame used to create really great tabular data representations easily.

It provides highly optimized performance with back-end source code is purely written in C or Python.

Sample Python code using pandas:

Code Creating Data Frame:

import pandas as pd   # Import Library

  a = pd.DataFrame(Data)  # Create Data Frame with Data

# Program to Create Data Frame with two dictionaries

dict1 ={'a':1, 'b':2, 'c':3, 'd':4}        # Define Dictionary 1

dict2 ={'a':5, 'b':6, 'c':7, 'd':8, 'e':9} # Define Dictionary 2

Data = {'first':dict1, 'second':dict2}  # Define Data with dict1 and dict2

df = pd.DataFrame(Data)  # Create Data Frame

Code Creating a Series:

import pandas as pd  # Import Panda Library

# Create series with Data, and Index

a = pd.Series(Data, index = Index)

# Program to Create series with scalar values

Data =[1, 3, 4, 5, 6, 2, 9]  # Numeric data

# Creating series with default index values

s = pd.Series(Data)

# predefined index values

Index =['a', 'b', 'c', 'd', 'e', 'f', 'g']

# Creating series with predefined index values

si = pd.Series(Data, Index)

NumPy:

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

* a powerful N-dimensional array object
* sophisticated (broadcasting) functions
* tools for integrating C/C++ and Fortran code
* useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Arbitrary data-types can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Sample Code implementing NumPy:

import numpy as np

# Creating array object

arr = np.array( [[ 1, 2, 3],

                 [ 4, 2, 5]] )

# Printing type of arr object

print("Array is of type: ", type(arr))

# Printing array dimensions (axes)

print("No. of dimensions: ", arr.ndim)

# Printing shape of array

print("Shape of array: ", arr.shape)

# Printing size (total number of elements) of array

print("Size of array: ", arr.size)

# Printing type of elements in array

print("Array stores elements of type: ", arr.dtype)

1. PROJECT SOURECE CODE:

from flask import Flask, render\_template, request

import pandas as pd

import numpy as np

from flask\_table import Table, Col

from rake\_nltk import Rake

from sklearn.metrics.pairwise import cosine\_similarity

from sklearn.feature\_extraction.text import CountVectorizer

# building flask table for showing recommendation results

class Item(Table):

name = Col('Recommendations')

description = Col('Score')

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

# Rating Page

@app.route("/", methods=["GET", "POST"])

def rating():

return render\_template('welcome.html')

# Results Page

@app.route("/recommendation", methods=["GET", "POST"])

def recommendation():

if request.method == 'POST':

df = pd.read\_csv('https://query.data.world/s/uikepcpffyo2nhig52xxeevdialfl7')

df = df[['Title', 'Genre', 'Director', 'Actors', 'Plot']]

# discarding the commas between the actors' full names and getting only the first three names

df['Actors'] = df['Actors'].map(lambda x: x.split(',')[:3])

# putting the genres in a list of words

df['Genre'] = df['Genre'].map(lambda x: x.lower().split(','))

df['Director'] = df['Director'].map(lambda x: x.split(' '))

# merging together first and last name for each actor and director, so it's considered as one word

# and there is no mix up between people sharing a first name

for index, row in df.iterrows():

row['Actors'] = [x.lower().replace(' ', '') for x in row['Actors']]

row['Director'] = ''.join(row['Director']).lower()

# initializing the new column

df['Key\_words'] = ""

for index, row in df.iterrows():

plot = row['Plot']

# instantiating Rake, by default is uses english stopwords from NLTK

# and discard all punctuation characters

r = Rake()

# extracting the words by passing the text

r.extract\_keywords\_from\_text(plot)

# getting the dictionary with key words and their scores

key\_words\_dict\_scores = r.get\_word\_degrees()

# assigning the key words to the new column

row['Key\_words'] = list(key\_words\_dict\_scores.keys())

# dropping the Plot column

df.drop(columns=['Plot'], inplace=True)

df.set\_index('Title', inplace=True)

df['bag\_of\_words'] = ''

columns = df.columns

for index, row in df.iterrows():

words = ''

for col in columns:

if col != 'Director':

words = words + ' '.join(row[col]) + ' '

else:

words = words + row[col] + ' '

row['bag\_of\_words'] = words

df.drop(columns=[col for col in df.columns if col != 'bag\_of\_words'], inplace=True)

# instantiating and generating the count matrix

count = CountVectorizer()

count\_matrix = count.fit\_transform(df['bag\_of\_words'])

# creating a Series for the movie titles so they are associated to an ordered numerical

# list I will use later to match the indexes

indices = pd.Series(df.index)

# generating the cosine similarity matrix

cosine\_sim = cosine\_similarity(count\_matrix, count\_matrix)

def recommendations(title):

recommended\_movies = []

# gettin the index of the movie that matches the title

try:

idx = indices[indices == title].index[0]

except:

res = "This movie in not registered in our database"

return res, ['0']

# creating a Series with the similarity scores in descending order

score\_series = pd.Series(cosine\_sim[idx]).sort\_values(ascending=False)

# getting the indexes of the 10 most similar movies

top\_10\_indexes = list(score\_series.iloc[1:11].index)

# populating the list with the titles of the best 10 matching movies

for i in top\_10\_indexes:

recommended\_movies.append(list(df.index)[i

return recommended\_movies, score\_series[1:11]

int\_features = [str(x) for x in request.form.values()]

int\_features = ''.join(int\_features)

output, score = recommendations(int\_features)

if output == "This movie in not registered in our database":

return render\_template('welcome.html', prediction\_text=out

else:

x = [str(i) for i in list(round(score, 3))]

items = [dict(name=output[0], description=x[0]),

dict(name=output[1], description=x[1]),

dict(name=output[2], description=x[2]),

dict(name=output[3], description=x[3]),

dict(name=output[4], description=x[4]),

dict(name=output[5], description=x[5]),

dict(name=output[6], description=x[6]),

dict(name=output[7], description=x[7]),

dict(name=output[8], description=x[8]),

dict(name=output[9], description=x[9])]

table = Item(items)

table.border = True

return render\_template('welcome.html', prediction\_text=table)

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

app.run(debug=True)

CODE EXPLANATION:

This code works on the data collected from the various sources. The data is organized in the csv\_data frames and that we can get the better results from the system quickly with the help of this system. To ease the people, we also used the necessary GUI related models such as bar graphs, pie charts and all other codes that are used for visualization effects to attract the users.

Our code basically programmed using the python programming language and used various libraries and all the libraries for its implementation. we have used lambda functions and data frames concept from numpy library. the code suggests the best 10 films when we give the input. the 10 best results are based upon the input movie name given by the user. it also displays the required login methods to sign up or sign in depending upon the usage of the page.it suggests the best 10 results from the entire data based upon the user’s interests.

We had used the simply python functions and libraries to implement all the AI related topics to build the efficient movies recommended systems. The other most import important topic that code also supports collaborative filtering. collaborative filtering approach is to collect and analyze a large amount of information about user actions and settings and then predict which users will favor their similarity with other users. The advantage of collaborative filtering is that it does not rely on content that can be analyzed and can accurately represent complex items.

As usual we used the dictionaries in the code. **Dictionary**in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data types that hold only single value as an element, Dictionary holds key: value pair. Key value is provided in the dictionary to make it more optimized. In Python, a Dictionary can be created by placing sequence of elements within curly **{}** braces, separated by ‘comma’.

Dictionary holds a pair of values, one being the Key and the other corresponding pair element being its Key: value. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can’t be repeated and must be immutable*.* n Python Dictionary, Addition of elements can be done in multiple ways. This is a developer community based code and we used the simple python programming language to build the multifunctionality code and this can produces us the two different pages one as the welcome page in which we can get all the user details and the requirements sites if any and then he can log on to the system in which a new page appears on the screen and that takes him to the search bar where he can type the name of the movie he wants and that shows him the best 10 results out of all the existing ones and that he can opt one of it.

1. Implementation of the testing algorithm:

ALGORITHM:

import numpy as np

import pandas as pd

from scipy import spatial

load and to check data:

ratings = pd.read\_csv("../input/ratings.csv")

# links = pd.read\_csv("../input/links.csv")

tags = pd.read\_csv("../input/tags.csv")

movies = pd.read\_csv("../input/movies.csv")

Ratings:

print(ratings. shape)

ratings.head(5)

pd.options.display.float\_format = '{:f}'.format

ratings['rating'].describe()

ratings['rating'].hist()

ratings['rating'].plot(kind='box', subplots=True)

Ratings group by:

userRatingsAggr = ratings.groupby(['userId']).agg({'rating': [np.size, np.mean]})

userRatingsAggr.reset\_index(inplace=True) # To reset multilevel (pivot-like) index

# userRatingsAggr.head()

userRatingsAggr['rating'].describe()

userRatingsAggr['rating'].plot(kind='box', subplots=True)

get movie years from titles:

def getYear(title):

result = re.search(r'\(\d{4}\)', title)

if result:

found = result.group(0).strip('(').strip(')')

else:

found = 0

return int(found)

movies['year'] = movies.apply(lambda x: getYear(x['title']), axis=1)

# movies.head(10)

genresList = [“action”,”adventure”,animation”,”children”,”comedy”,”crime”,”documentary”, ”darama”,”fantasty”,”film noyr”,”musical”,horror”]

def setGenresMatrix(genres):

movieGenresMatrix = []

movieGenresList = genres.split('|')

for x in genresList:

if (x in movieGenresList):

movieGenresMatrix.append(1)

else:

movieGenresMatrix.append(0)

return movieGenresMatrix

movieRatingsAggr['rating'].describe(percentiles=[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.98, 0.99])

The desired rating number groups will be:

0 - not rated movie

1 - count of ratings between 1 - 10

2 - count of ratings between 11 - 30

3 - count of ratings between 31 - 100

4 - count of ratings between 101 - 300

5 - count of ratings between 301 - 1000

6 - count of ratings between 1001 -

words in movies title:

titleWordsDict = {}

for index, x in movies.iterrows():

wordlist = str(x['title']).lower().split(' ')

movieId = x['movieId']

for y in wordlist:

if y not in stopWords:

if movieId in titleWordsDict:

titleWordsDict[movieId].append(y)

else:

titleWordsDict[movieId] = [y]

Movies Recommended Algorithm:

def tagsSimilarity(basisMovieID, checkedMovieID, checkType):

# The higher value is the more similar (from 0 to 1)

if checkType == 'tag':

dictToCheck = tagsDict

else:

dictToCheck = titleWordsDict

counter = 0

if basisMovieID in dictToCheck:

basisTags = dictToCheck[basisMovieID]

countAllTags = len(basisTags)

basisTagsDict = {}

for x in basisTags:

if x in basisTagsDict:

basisTagsDict[x] += 1

else:

basisTagsDict[x] = 1

for x in basisTagsDict:

basisTagsDict[x] = basisTagsDict[x] / countAllTags

else: return 0

if checkedMovieID in dictToCheck:

checkedTags = dictToCheck[checkedMovieID]

checkedTags = set(checkedTags) # Make the list unique

checkedTags = list(checkedTags)

else: return 0

for x in basisTagsDict:

if x in checkedTags: counter += basisTagsDict[x]

return counter

def checkSimilarity(movieId):

# print("SIMILAR MOVIES TO:")

# print (movies[movies['movieId'] == movieId][['title', 'rating\_count', 'rating\_avg']])

basisGenres = np.array(list(movies[movies['movieId'] == movieId]['genresMatrix']))

basisYear = int(movies[movies['movieId'] == movieId]['year'])

basisRatingAvg = movies[movies['movieId'] == movieId]['rating\_avg']

basisRatingGroup = movies[movies['movieId'] == movieId]['ratingGroup']

it is a part of the implementation algorithm and here we are describing about the data comparison and the ratings page and the code for the implementation of the results page.

DESCRIPTION OF EACH MEMBER PARTICIPATION:

Here the first member in our project he worked on the implementations of the different modules that are required for the project namely we had used the pandas, numpy, skit learn, nltk and the flask table and many other. So, he totally worked on the topics related to their implementation and the working of the project for these modules. He collected the data required to organise in the csv\_tables and made them in the data frames and then we created a database from where we can access the data to get the results.

The second member of the project worked on the concepts of the AI that are required for the implementation of the algorithm and with that he organised the logic that should be necessary in implementing the search results and categorising them based upon the topics that we applied. Here topics that we used in implementation are “Bag of words model, stop words, count vectorizer, TF\_IDF model, collaborative filtering and the k cliques”. All these topics are efficiently related together for implantation of the algorithm.

The member of the project was totally engaged in writing the results that we made till from the beginning , he organised each and every point in the form and that can be used by the customer and from that he can understand easily. He collected the data from the code and the related topics and made an observation stating the advantages and disadvantages of each and every topic.

At last the fourth one had worked for the gui related topics that are necessary in the searching the results and getting the outputs for the results and connecting the pages in the online to make a link in between them so that they can launch from one page to the other.

TESTING ENVIRONMENT OF THE PROJECT:

Project Jupyter is a non-profit organization created to "develop open source software, open-standards, and services for interacting computing across dozens of programming languages". Spun-off from Ipython in 2014 by Fernando perez, Project Jupyter supports execution environments in several dozen languages. Project Jupyter's name is a reference to the three core programming languages supported by Jupyter, which are julia, python and [R](https://en.wikipedia.org/wiki/R_(programming_language)), and also a homage to galeilo's notebooks recording the discovery of the moons of jupiter. Project Jupyter has developed and supported the interactive computing products Jupyter Notebook, JupyterHub, and JupyterLab, the next-generation version of Jupyter Notebook.

### Jupyter Notebook

Jupyter notebook (formerly IPython Notebooks) is a web-based interactive computational environment for creating Jupyter notebook documents. The "notebook" term can colloquially make reference to many different entities, mainly the Jupyter web application, Jupyter Python web server, or Jupyter document format depending on context. A Jupyter Notebook document is a [JSON](https://en.wikipedia.org/wiki/JSON) document, following a versioned schema, and containing an ordered list of input/output cells which can contain code, text (using Markdoen), mathematics, plots and rich media, usually ending with the ".ipynb" extension.

A Jupyter Notebook can be converted to a number of open standard output formats (HTML, Presentation slides, LaTeX, PDF ,ReStructed Text, markdown , python) through "Download As" in the web interface, via the nbconvert library or "jupyter nbconvert" command line interface in a shell. To simplify visualisation of Jupyter notebook documents on the web, the nbconvert library is provided as a service through NBviewer which can take a URL to any publicly available notebook document, convert it to HTML on the fly and display it to the user.

Jupyter krnels

A Jupyter kernel is a program responsible for handling various types of requests (code execution, code completions, inspection), and providing a reply. Kernels talk to the other components of Jupyter using ZeroMQ over the network, and thus can be on the same or remote machines. Unlike many other Notebook-like interfaces, in Jupyter, kernels are not aware that they are attached to a specific document, and can be connected to many clients at once. Usually kernels allow execution of only a single language, but there are a couple of exceptions.

By default, Jupyter ships with IPython as a default kernel and a reference implementation via the ipykernel wrapper. Kernels for many languages having varying quality and features are available

JupyterHub

JupyterHub is a multi-user server for Jupyter Notebooks. It is designed to support many users by spawning, managing, and proxying many singular Jupyter Notebook servers.[ While JupyterHub requires managing servers, third-party services like Jupyo provide an alternative to JupyterHub by hosting and managing multi-user Jupyter notebooks in the cloud.

JupyterLab

JupyterLab is the next-generation user interface for Project Jupyter. It offers all the familiar building blocks of the classic Jupyter Notebook (notebook, terminal, text editor, file browser, rich outputs, etc.) in a flexible and powerful user interface. The first stable release was announced on February 20, 2018.

Anaconda is a free and open source distribution of the Python and R programming languages for scientific (data sciences, machine learning applications, large-scale data processing, predictive analitics, etc.), that aims to simplify package management and deployment. Package versions are managed by the package management. The Anaconda distribution includes data-science packages suitable for Windows, Linux, and macOS.

Anaconda distribution comes with 1,500 packages selected from PyPL as well as the  conda package and virtual environment manager. It also includes a GUI, Anaconda Navigator[[7]](https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)#cite_note-7), as a graphical alternative to the command line interface (CLI).

The big difference between conda and the pip package manager is in how package dependencies are managed, which is a significant challenge for Python data science and the reason conda exists.

When pip installs a package, it automatically installs any dependent Python packages without checking if these conflict with previously installed packages. It will install a package and any of its dependencies regardless of the state of the existing installation. Because of this, a user with a working installation of, for example, Google Tensorflow, can find that it stops working having used pip to install a different package that requires a different version of the dependent numpy library than the one used by Tensorflow. In some cases, the package may appear to work but produce different results in detail.

In contrast, conda analyses the current environment including everything currently installed, and, together with any version limitations specified (e.g. the user may wish to have Tensorflow version 2,0 or higher), works out how to install a compatible set of dependencies, and shows a warning if this cannot be done.

Interactive or the graphical results of the projects

A recommendation engine (sometimes referred to as a recommender system) is a tool that lets algorithm devolopers predict what a user may or may not like among a list of given items. Recommendation engines are a pretty interesting alternative to search fields, as recommendation engines help users discover products or content that they may not come across otherwise. This makes recommendation engines a great part of web sites and services such as Facebook, YouTube, Amazon, and more.

Recommendation engines work ideally in one of two ways. It can rely on the properties of the items that a user likes, which are analyzed to determine what else the user may like; or, it can rely on the likes and dislikes of other users, which the recommendation engine then uses to compute a similarity index between users and recommend items to them accordingly. It is also possible to combine both these methods to build a much more robust recommendation engine. However, like all other information related problems, it is essential to pick an algorithm this is suitable for the project being addressed.

Sets and Equations

Before implementing a collaborative memory-based recommendation engine, we must first understand the core idea behind such a system. To this engine, each item and each user is nothing but identifiers. Therefore, we will not take any other attribute of a movie (for example, the cast, director, genre, etc.) into consideration while generating recommendations. The similarity between two users is represented using a decimal number between -1.0 and 1.0. We will call this number the similarity index. Finally, the possibility of a user liking a movie will be represented using another decimal number between -1.0 and 1.0. Now that we have modelled the world around this system using simple terms, we can unleash a handful of elegant mathematical equations to define the relationship between these identifiers and numbers.

Conclusion:

we illustrated how to build a scalable neighbourhood-based collaborative filtering recommender system on Apache Spark. We provided the model with an understanding of neighbourhood-based collaborative filtering methods and discussed the challenges of implementing them at scale. We introduced the core concepts behind Spark’s data flow programming model and provided implementations of both user and item-based collaborative filtering algorithms on it.

References:

*Jeong WH, Kim SJ, Park DS, Kwak J (2013) Performance improvement of a* *movie recommendation system based on personal propensity and secure collaborative filtering. J Inf Process Syst*

*ArticleGoogle Scholar*

*Viana P, Pinto JP (2017) A collaborative approach for semantic time-based video annotation using gamification. Hum Cent Comput Inf Sci*

*ArticleGoogle Scholar*

*Desrosiers C, Karypis G (2011) A comprehensive survey of neighborhood-based recommendation methods. In: Ricci F, Rokach L, Shapira B, Kantor P (eds) Recommender systems handbook. Springer, Boston, pp*

*Google Scholar*

*Bagci H, Karagoz P (2016) Context-aware location recommendation by using a random walk-based approach. Knowl Inf Syst*

*ArticleGoogle Scholar*

*Wong F, Lee S, Wong Q, Lee S (2015) Points of interest recommendation based on context-aware. Int J Hybrid Inf Technol*

End of the document