**CHAPTER 1**

**INTRODUCTION**

As we know that, the world is growing faster like never before. Everyone is rushing for their ultimate goals. This thirst results into the development of almost every sector. Online business is one of them. We people, don’t have time to shop from market and this is not the end. We don't even have time to choose the object from the collection .This created the embryo of online shopping, which nowadays, became a huge tree, of tons of branches.

As the online market grows exponentially, it’s obvious that competition will entered in this field also. Now, owners of their respective sites need to attract their users by providing attractive facilities. Movie Recommender Engines are one of the facilities given to users. Recommender engine are the most immediately recognizable machine learning technique in use today. We will have seen services or sites that attempt to recommend books or movies or articles based on our past actions. Netflix similarly recommends DVDs that may be of interest, and famously offered a $1,000,000 prize to researchers who could improve the quality of their recommendations.

* 1. **Problem Definition:**

Chatbot is a tool to retrieve information and generate humanlike conversation. It is mainly a dialog system aimed to solve/serve a specific purpose. Chatbots have gained increasing importance for research and practice with a lot of applications available today including Amazon’s Alexa or Apple’s Siri. In this project, we present the underlying methods and technologies behind a Chatbot for movie recommendation that allows people textually communicate with the purpose of movie recommendation, ratings, reviews, cast and other interesting insights. It is a simple bot that answers questions about movies.

The user can ask about ratings, #people voted for the movie, genre, movie overview, similar movies, imdb and tmdb links, budget, revenue and adult content. The data and processing is all handled in the local system. Even though we use IBM, it is used as an API service and none of the internal data is sent to IBM. This way the entire design can be implemented in your workplace without having to worry about data transfers. Besides the underlying foundations, we provide a use case from the intended movie domain to show how such a model chatbot effectively can be used in practice.

* 1. **Scope Of the Project:**

It is important here to define the scope of the project. Although vital to any user operating in the real world, no attempt is made in this project at portfolio management. Portfolio management is largely an extra step done after a user has made a decision on which direction to proceed. The system should ensure that customer satisfaction is increased and profitability of the organisation is increased. While at the same time, the costs involved in publicising a movie should be reduced. To further elaborate on the scope of the project, the existing system is collated with the proposed system below.

**Existing System:**

Odeon’s chatbot, developed by social technology company Gruvi, requires user to like the brand’s Facebook page and then either click “Message” or type “Odeon” into a chat search. After a greeting from the bot Users are asked for their location or what film they are interested in seeing. The bot then informs the customer of nearby cinemas or where, and what time, their selected film is showing. Once a decision has been made, the customer is sent a link to a booking page. This is developed in “EUROPE” for Odeon’s Cinemas.

After several months of development, ODEON launched on 28 November a Facebook Chatbot that helps user discover what is playing in cinema near them and book tickets. The chatbot, accessible through the official ODEON Facebook page, has been developed by Gruvi. Chatbots are emerging technology that leverage messaging habits to help business communicate more efficiently to their clients. Chatbots intermediate and help users with specific task. The future scope is limitless. First there was traditional ticket booking i.e. Window Booking then came a Smart Application i.e. BookMyShow now came an automated chatbot.

**Proposed System:**

The system is built on windows 2007 operating system. The system uses advanced java technology along with machine learning concepts. MySQL is used for storing data. This system uses three-tier architecture. The web service layer provides the android user to rate movies, view similar recommendations given by the system and comment on it. The proposed system is a better system than any other existing systems. This system has added the positive features of existing systems and has overcome the drawbacks of existing systems. The system uses all the existing algorithms i.e. content based, context based and collaborative based algorithms. All these algorithms are combined to give more precise result. The following modules are developed as:

**A. Admin**

The system admin will be able to add a movie in a database, view movies and update it. Simultaneously, the admin will be able to create chat rooms with the chatbot and add or remove members to it.

**B. Recommendation Engine**

This recommendation engine will calculate the similarities between the different users as well as between different movies. On the basis of that similarities calculated, this engine will recommend movie to a user.

**C. Movie Web Service**

This will allow user to view movies on their official website and rate movie or comments on movies. This service will also show the movie recommendation to the users.

**D. End User**

The end user can rate a movie, can comment on any movie, and can see similar movies recommended by other users who are similar to this user. The user can also ask questions to the chat bot such as movie genre, overview, budget, revenue, rating, voters, IMDB or TMDB links, adult content, etc.

**CHAPTER 2**

**REVIEW OF LITERATURE**

We studied various papers describing the research done in this domain and also the comparison of various models. This chapters summaries some of those research papers.

**2.1 Papers**

MOVREC [10] is a movie recommendation system presented by D.K. Yadav et al. based on collaborative filtering approach. Collaborative filtering makes use of information provided by user. That information is analyzed and a movie is recommended to the users which are arranged with the movie with highest rating first. The system also has a provision for user to select attributes on which he wants the movie to be recommended.

Luis M Capos et al. [5] has analyzed two traditional recommender systems i.e. content based filtering and collaborative filtering. As both of them have their own drawbacks he proposed a new system which is a combination of Bayesian network and collaborative filtering. The proposed system is optimized for the given problem and provides probability distributions to make useful inferences.

A hybrid system has been presented by Harpreet Kaur et al. [9]. The system uses a mix of content as well as collaborative filtering algorithm. The context of the movies is also considered while recommending. The user - user relationship as well as user - item relationship plays a role in the recommendation.

The user specific information or item specific information is clubbed to form a cluster by Utkarsh Gupta et al. [12] using chameleon. This is an efficient technique based on Hierarchical clustering for recommender system. To predict the rating of an item voting system is used. The proposed system has lower error and has better clustering of similar items.

Urszula Kużelewska et al. [6] proposed clustering as a way to deal with recommender systems. Two methods of computing cluster representatives were presented and evaluated. Centroid-based solution and memory-based collaborative filtering methods were used as a basis for comparing effectiveness of the proposed two methods. The result was a significant increase in the accuracy of the generated recommendations when compared to just centroid-based method.

Costin-Gabriel Chiru et al. [3] proposed Movie Recommender, a system which uses the information known about the user to provide movie recommendations. This system attempts to solve the problem of unique recommendations which results from ignoring the data specific to the user. The psychological profile of the user, their watching history and the data involving movie scores from other websites is collected. They are based on aggregate similarity calculation. The system is a hybrid model which uses both content based filtering and collaborative filtering.

To predict the difficulty level of each case for each trainee Hongli LIn et al. proposed a method called content boosted collaborative filtering (CBCF).The algorithm is divided into two stages, First being the content-based filtering that improves the existing trainee case ratings data and the second being collaborative filtering that provides the final predictions.

* 1. **Techniques studied**
     1. Content-based Filtering Systems (CBF based systems)

In content-based filtering, items are recommended based on comparisons between item profile and user profile. A user profile is content that is found to be relevant to the user in form of keywords(or features). A user profile might be seen as a set of assigned keywords (terms, features) collected by algorithm from items found relevant (or interesting) by the user. A set of keywords (or features) of an item is the Item profile. For example, consider a scenario in which a person goes to buy his favorite cake ‘X’ to a pastry. Unfortunately, cake ‘X’ has been sold out and as a result of this the shopkeeper recommends the person to buy cake ‘Y’ which is made up of ingredients similar to cake ‘X’. This is an instance of content-based filtering.

2.2.2 Collaborative filtering based systems (CF based systems)

Collaborative filtering system recommends items based on similarity measures between users and/or items. The system recommends items preferred by similar users. This is based on the scenario where a person asks his friends, who have similar tastes, to recommend him some movies.

2.2.3 Nearest Neighbors Collaborative Filtering

This approach relies on the idea that users who have similar rating behaviors so far, share the same tastes and will likely exhibit similar rating behaviors going forward. The algorithm first computes the similarity between users by using the row vector in the ratings matrix corresponding to a user as a representation for that user. The similarity is computed by using either cosine similarity or Pearson Correlation. In order to predict the rating for a particular user for a given movie j, we find the top k similar users to this particular user and then take a weighted average of the ratings of the k similar users with the weights being the similarity values

2.2.4 Latent Factor Method

The latent factor algorithm looks to decompose the ratings matrix R into two tall and thin matrices Q and P, with matrix Q having dimensions num\_users × k and P having the dimensions numitems × k where k is the number of latent factors. The decomposition of R into Q and P is such that

R = Q.PT

Any rating rij in the ratings matrix can be computed by taking the dot product of row qi of matrix Q and pj of matrix P. The matrices Q and P are initialized randomly or by performing SVD on the ratings matrix. Then, the algorithm solves the problem of minimizing the error between the actual rating value rij and the value given by taking the dot product of rows qi and pj. The algorithm performs stochastic gradient descent to find the matrices Q and P with minimum error starting from the initial matrices.

**CHAPTER 3**

**PROPOSED SYSTEM**

**3.1 Design:**

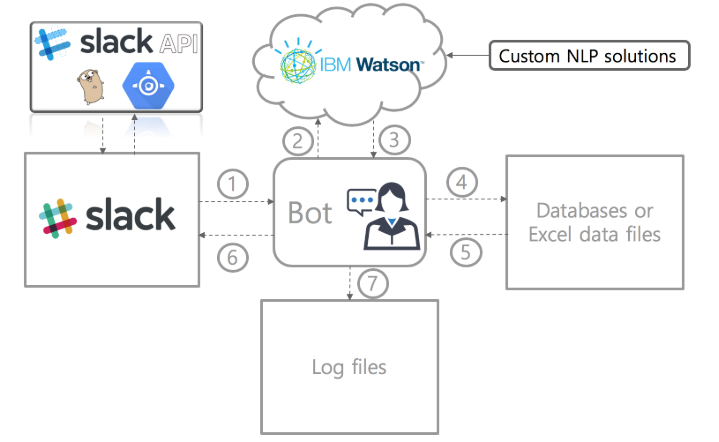
****

Fig 3.1: Block Diagram

Fig 3.1 illustrates the Block Diagram, which represents the design of our system. The movie recommending chatbot has two parts the frontend and the backend the frontend is using the Application Program Interface provided by the Slack technologies, the slack is a collaborative platform where groups can collaborate and discuss on various projects they are working on, the best feature of the Slack application program interface is that it provides a domain which is hosted on the slack platform servers and create separate workspace for each, with this excellent feature you can access your slack platform from anywhere and anytime because your workspace is hosted and you are individually assigned a domain which can be accessed on the slack website or any personal computer application or a mobile application. You can directly talk to the Slack Chabot about along with your chatbot name, ask about the movies, genre,etc. Also, IBM Watson Assistant is used for smooth processing of NLP solutions and handling components like intents, dialogues and entities.

**3.2 Implementation Methodology:**

**3.2.1 Open Domain**

Open Domain bots are otherwise known as Generalist bots. Today we use Alexa, Google Home, Siri, Cortana which fall under this category(open domain/generative-based). These bots try to imitate humanlike conversation. Again, it answers questions (like FAQ’s) asked by most humans. However, they cannot answer a specific domain based question. For example: How did my company sales division performed in the last quarter? That is one of the reason, open domain/retrieval-based bots is impossible to build.

**3.2.2 Closed Domain**

Closed Domain bots are otherwise known as Specialist bots. Depending upon the type, it can be easy (retrieval-based) or hard(generative-based) to develop. The bot discussed in this article is a specialist bot and it falls under the closed domain/retrieval-based category. Other bots in this category include — order a pizza, book flights/restaurants/hotel/appointments. On the other hand, generative bots include customer service chatbots which try to imitate like a agent while answering the questions from customer. These bots are hard to build since the bots try to make the customer believe that they are talking to a actual human.

**3.2.3 Components of the System:**

Chatbots needs to understand the following to respond to an user question.

What is the user talking about? (Intent)

Did the user mention anything specific? (Entities)

What should the bot ask to get further details from the user? (Dialog/Maintaining Context)

How to fulfill the user request? (Response/Fulfillment)

**Intent**

The intent of the user is to book flights.

**Entities**

Entities are also known as keywords or slots. Here there are multiple entities.

**Dialog/Maintaining Context**

Dialogs are back and forth communication between bot and user. A context let’s the bot know what state the bot is currently in. There are three states — Previous, Present and Future. When the user initiates the dialog, the bot reiterates the user itinerary and then checks with the user “Is this info correct?”. Here the previous state is blank, present state is “user validation” and future state is to “Provide a response based on user validation”. When the user responds “Yes”, then the bot state changes to “User validation”, “Provide a response based on user validation” and “Book a flight” for Previous, present and future state respectively. Without maintaining the context, bots cannot establish the back and forth communication. In the flight bot example, if the context is not maintained the bot would be asking “Is this info correct?” every time until the user gives up.

**Response/Fulfillment**

Fulfilling the user request is the final step in the bot conversation. Here the bot provides the results in the form of links “See all results”. So when the user clicks the link, they will be able to see the flights and make a reservation. Generative-based bots use AI and Machine learning to generate user responses. So they need not have to understand the Intents and Entities to respond to a user.

**3.2.4 Operation and Workflow**

The operation and workflow of the system is discussed below as a sequence of 7 steps:

Step 1 User asks question:

Users can interact with chatbot via Slack. Once the user post a question, it is passed to the backend system for analysis.

Step 2 and 3 (NLP processing and Return the NLP results):

All the natural language processing happens in step 2. This includes IBM Watson processing, similarity search, recommendation based on collaborative filtering. After the NLP processing is completed, we have three outputs from it

1. Intents — What the user is trying to ask or query?
2. Entities — What is the exact field or column they are looking for?
3. Dialog/Interaction — Provide the appropriate request/response for the user question.

Step 4 and 5(Query the data):

Currently, the data resides in a excel file. However, you can add multiple databases/excel files if needed, to access different sources. Based on the results from step 3, the appropriate database/excel file is queried and the results are returned.

Step 6 Post the result to user:

The results obtained from the backend is posted to user via Slack

Step 7 Log maintenance:

The interactions between the users are logged and stored in a text file. Also, if the bot is not able to identify the user questions it will add those questions to a followup file.

**3.3 Details of Hardware and Software:**

Recommended System Requirements:

* Processors: Intel Core i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores,2 threads per core), 8 GB of DRAM
* Disk space:3 to 5 GB
* Operating systems: Windows 10, macOS, and Linux

Minimum System Requirements:

* Processors: Intel Core i3 processor
* Disk space: 2 GB
* Operating systems: Windows 7 or later, macOS, and Linux
* Python versions: 3.6.X

Software Requirements:

* Python 3.6
* Web Browser such as Google Chrome

**CHAPTER 4**

**IMPLEMENTATION PLAN**

**4.1 Data Sources:**

The data for this project is taken from the Kaggle. The name of the dataset is “movies\_metadata.csv”. The dataset contains a lot of information related to movies with less preprocessing required from users. We import the dataset using Pandas and then prepare our data.

**4.2 Slack:**

The first step is to create a slack bot and install it in the workspace. In the configuration file, you need to edit the Slack Bot token and Slack verification token. That is all the setup required for slack. Visit https://api.slack.com/apps to create a slack app. Provide the App name and the slack workspace you would like to install the app. After that, you can click on create app and you will be redirected to the app page. Click on the bot user features on the left panel of the app page and fill in the details. Please make sure to turn on the toggle option to show your bot always online. Once the details are provided, click "Add Bot User". Now, you are ready to install the app on your workspace. Navigate to the Install App panel and click on the button "Install App to Workspace". Make a note of the "Bot User OAuth Access Token". This token will be needed to access your slack app from python. Some application requires bot verification as well. This token can be found in "Basic Information" tab under "App Credentials" section" as shown below.

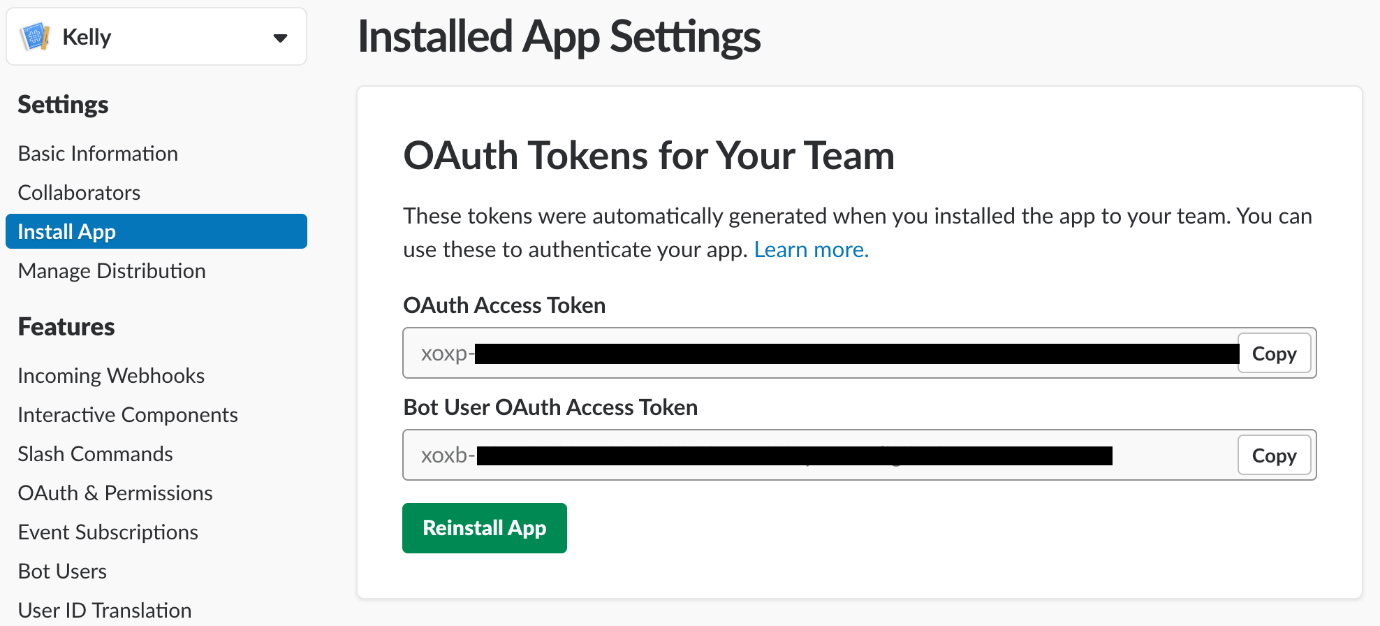
****

Fig 4.1: Getting Slack Credentials

**4.3 IBM Watson:**

A quick and easy way to develop chatbots is to use frameworks. There are lot of frameworks in the market, that can assist us to build bots. A few of them include — IBM Watson, Luis, Wit, Dialogflow, Rasa NLU, Botkit, Microsoft Bot Framework and so on. You can chose any one of the framework to build your bots. Again, the complexity of coding might vary depending upon the framework you choose. For this project, we have used IBM Watson framework. IBM Watson framework is used as an API service to perform Natural language processing. Chatbot is built in such a way that none of the information is stored in IBM Watson. All the processing of information is happening in your local system. Only the question from the user is processed by IBM Watson service to provide the Intents, Entities, Dialog and Response. Other than that, none of the information is stored in IBM Watson.

In the AI section of your home page, you will see \_\_"Watson Assistant (formerly Conversation)"\_\_. Please click on that to create a conversation resource. Please follow the steps below to create a resource. If you have already created your resource, you can access resource via the dashboard which is shown below.

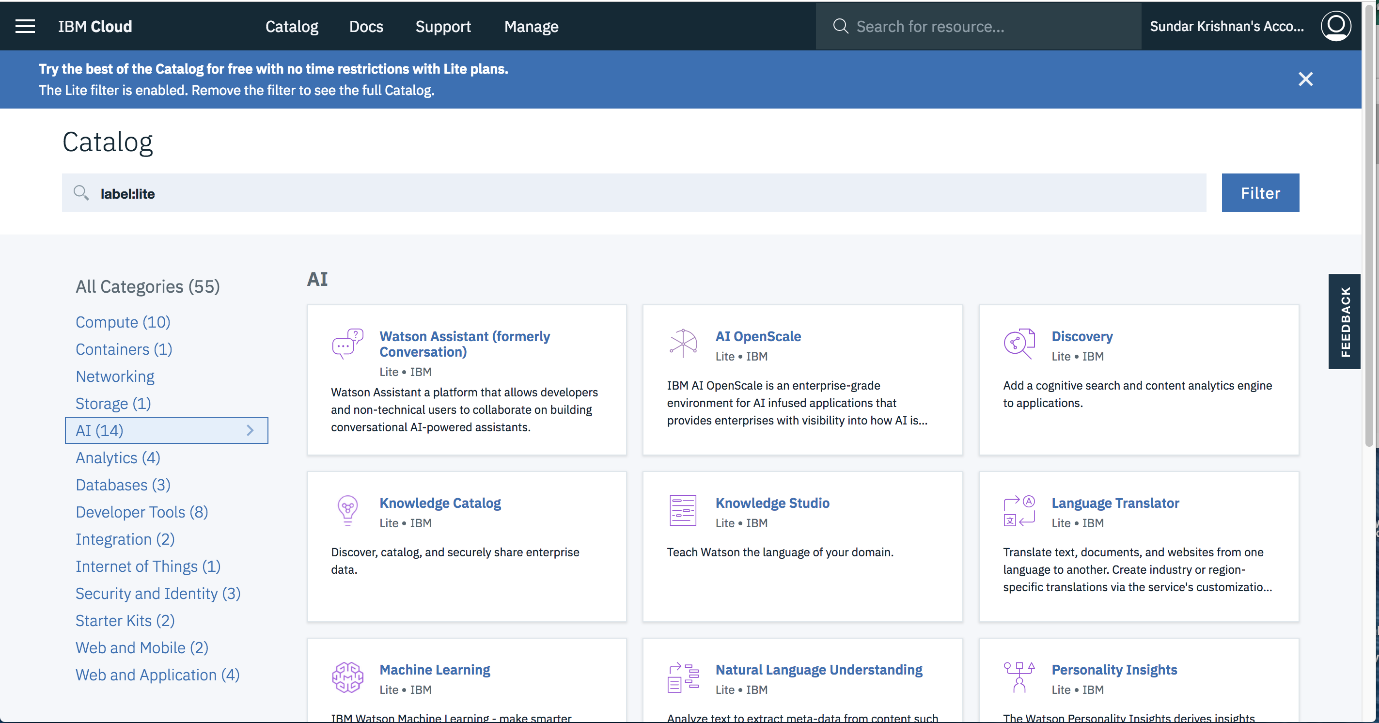


Fig 4.2 IBM Watson Dashboard

Now you can click on "Launch tool" and start adding skills for your bot. Create a Bot workspace and add skills. Get the API details for your bot, by clicking on the "View API Details" and you need to grab two pieces of information to add to configuration file. This information is highlighted in orange color.

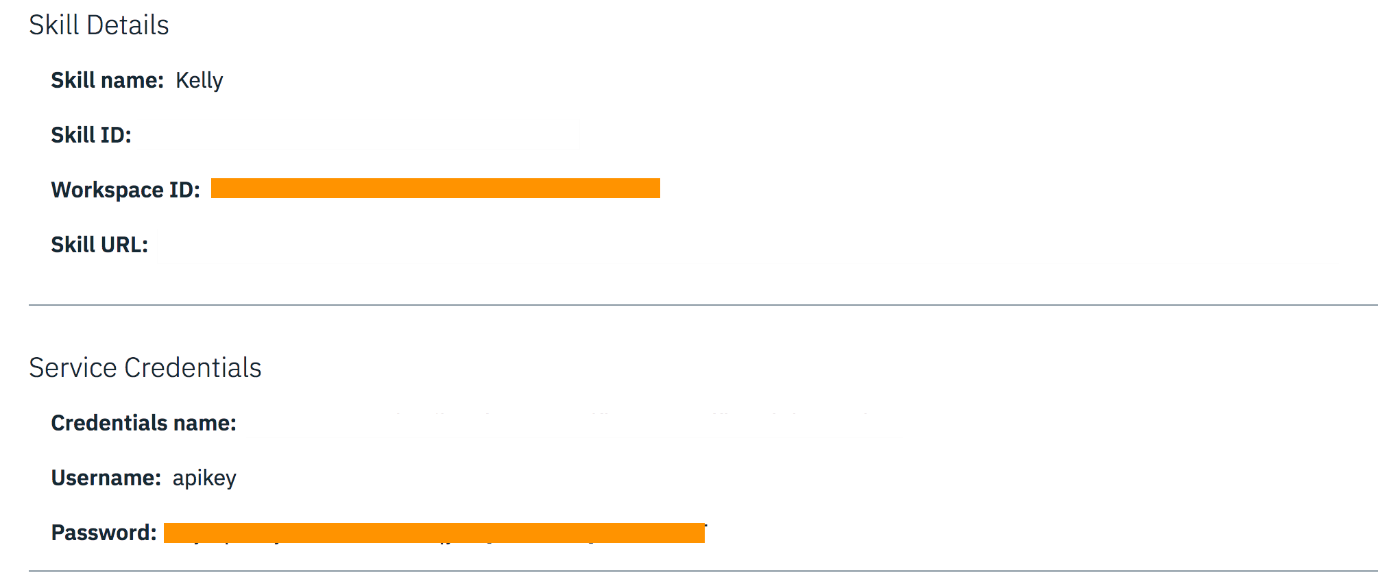


Fig 4.3: Getting IBM Watson Credentials

Please make sure that you modified the API details both for Slack and Watson in the config.py file.

**4.4 Implementation Process:**

**main.py**

###################################################################

######## It all starts here #############

###################################################################

"""

This code talks to the entire bot Framework.

"""

import os,string

import time

import datetime

from slack.slack\_commands import parse\_bot\_commands, output\_command

from config import slack\_client, log\_commands\_path

from nlp.nlp\_commands import handle\_command

import pandas as pd

import json

# Initialize with empty value to start the conversation.

user\_input = ''

context = {}

current\_action = ''

follow\_ind = 0

session\_df = pd.DataFrame({},columns=['timestamp', 'user', 'context']) #stores the session details of the user

# bot's user ID in Slack: value is assigned after the bot starts up

bot\_id = None

# constants

RTM\_READ\_DELAY = 1 # 1 second delay between reading from RTM

if \_\_name\_\_ == "\_\_main\_\_":

if slack\_client.rtm\_connect(with\_team\_state=False):

print("Kelly connected and running!")

# Read bot's user ID by calling Web API method `auth.test`

bot\_id = slack\_client.api\_call("auth.test")["user\_id"]

while True:

user\_id,message\_user,message,team,channel,start\_timestamp = parse\_bot\_commands(slack\_client.rtm\_read(),bot\_id) #slack processing

if message: # If a User has typed something in Slack

try:

context = json.loads(session\_df.loc[session\_df.user == message\_user+channel,'context'].values[0])

except:

context = {}

session\_df = session\_df.append({'timestamp': start\_timestamp, 'user': message\_user+channel, 'context': json.dumps(context)}, ignore\_index=True)

context,slack\_output,current\_action = handle\_command(message,channel, message\_user,context) #nlp processing

session\_df.loc[session\_df.user == message\_user+channel,'context'] = json.dumps(context)

output\_command(channel, slack\_output) #slack processing

conversation\_id = context['conversation\_id']

try:

if context['currentIntent'] in ['anything\_else']:

follow\_ind = 1

else:

follow\_ind = 0

except:

pass

if current\_action == 'end\_conversation': # Based on the this, the context variables are reset

session\_df = session\_df[session\_df.user != message\_user+channel]

context = {}

current\_action = ''

end\_timestamp = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')

processing\_time = str((datetime.datetime.strptime(end\_timestamp, '%Y-%m-%d %H:%M:%S') - datetime.datetime.strptime(start\_timestamp, '%Y-%m-%d %H:%M:%S')).total\_seconds())

string\_to\_run = string.Template("""python3 -W ignore "${log\_commands\_path}" "${user\_id}" "${message\_user}" "${conversation\_id}" "${message}" "${slack\_output}" "${team}" "${channel}" "${start\_timestamp}" "${end\_timestamp}" "${processing\_time}" "${follow\_ind}" &""").substitute(locals()) #logs processing

os.system(string\_to\_run)

time.sleep(RTM\_READ\_DELAY)

else:

print("Connection failed. Exception traceback printed above.")

**config.py**

import os

import watson\_developer\_cloud

from slackclient import SlackClient

location = "/Users/Arvind/Downloads/Movie\_Bot/" # replace with the full folder path where you downloaded the github repo

###################################################################

######## Slack configuration ##########################

###################################################################

SLACK\_BOT\_TOKEN='xoxb-762458134353-768410665828-TWM301hVVlAx1EGRTgmui17j'

SLACK\_VERIFICATION\_TOKEN='g3quhQgS81aDAGoB1XQjl5bR'

# instantiate Slack client

slack\_client = SlackClient(SLACK\_BOT\_TOKEN) # do not change this parameter

###################################################################

######## Watson service configuration ##########################

###################################################################

service = watson\_developer\_cloud.AssistantV1(

iam\_apikey = 'dy3E6rp1EXmi-7TKsiAkDAz23iW5WNnEy7nYkzfBH1qD', # replace with Password

version = '2018-09-20'

)

workspace\_id = '2c1264f9-1510-411b-9988-42e40a5088bd' # replace with Assistant ID

###################################################################

######## Log files configuration ##########################

###################################################################

log\_commands\_path = location + "logs/log\_commands.py" # do not change this parameter

follow\_up\_path = location + "logs/followup\_email.py" # do not change this parameter

###################################################################

######## Temp files configuration ##########################

###################################################################

onetime\_path = location + "nlp/nlp\_solutions/onetime\_run\_file.py" # do not change this parameter

onetime\_file = location + "nlp/nlp\_solutions/onetime.txt" # do not change this parameter

**onetime\_run\_file.py**

from sklearn.externals import joblib

import pandas as pd

import os,sys

sys.path.append(os.path.normpath(os.getcwd()))

from config import onetime\_file

from nlplearn import \*

if \_\_name\_\_ == "\_\_main\_\_":

metadata = pd.read\_csv('data/metadata\_prep.csv')

###################################################################

######## Metadata based collobarative filtering #############

###################################################################

documents = metadata['overview'].fillna('')

cosine\_sim = metadata\_filtering(documents)

indices = pd.Series(metadata.index, index=metadata['title']).drop\_duplicates()

###################################################################

######## Title search based on Keywords #############

###################################################################

documents = list(metadata['title'].fillna(''))

tfidf\_fit, tfidf\_matrix = tfidf\_fit(documents)

###################################################################

######## dump the variables to a text file #############

###################################################################

i = [cosine\_sim,indices,tfidf\_fit, tfidf\_matrix]

joblib.dump(i,onetime\_file)

**nlplearn.py**

from nltk.tokenize import word\_tokenize

import re

import nltk

from nltk.corpus import stopwords

from sklearn.externals import joblib

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from sklearn.metrics.pairwise import linear\_kernel

import pandas as pd

import os,sys

sys.path.append(os.path.normpath(os.getcwd()))

from config import onetime\_file

try:

cosine\_sim,indices, tfidf\_fit1, tfidf\_matrix1 = joblib.load(onetime\_file)

except:

cosine\_sim, indices, tfidf\_fit1, tfidf\_matrix1 = ['','','','']

metadata = pd.read\_csv('data/metadata\_prep.csv')

REPLACE\_BY\_SPACE\_RE = re.compile('[/(){}\[\]\|@,;]')

BAD\_SYMBOLS\_RE = re.compile('[^0-9a-z #+\_]')

STOPWORDS = set(stopwords.words('english'))

def text\_prepare(doc):

doc = doc.lower()

doc = REPLACE\_BY\_SPACE\_RE.sub(' ',doc)

doc = BAD\_SYMBOLS\_RE.sub('',doc)

doc = " ".join([w for w in word\_tokenize(doc) if not w in STOPWORDS])

return doc

def tfidf\_fit(docs):

docs = [text\_prepare(text) for text in docs]

tfidf\_vectorizer = TfidfVectorizer()

tfidf\_fit = tfidf\_vectorizer.fit(docs)

tfidf\_matrix = tfidf\_fit.transform(docs)

return tfidf\_fit, tfidf\_matrix

def similarity\_search(doc, list\_index, tfidf\_fit=tfidf\_fit1,tfidf\_matrix=tfidf\_matrix1):

out = cosine\_similarity(tfidf\_fit.transform([text\_prepare(doc)]), tfidf\_matrix.tocsr()[list\_index,:])

a = list(out[0])

b = sorted(range(len(a)), key=lambda i: a[i], reverse=True)[:5]

return b

def metadata\_filtering(docs):

tfidf = TfidfVectorizer(stop\_words='english')

docs = docs.fillna('')

tfidf\_matrix = tfidf.fit\_transform(docs)

cosine\_sim = linear\_kernel(tfidf\_matrix, tfidf\_matrix)

return cosine\_sim

def get\_recommendations(title, metadata=metadata,indices=indices, cosine\_sim=cosine\_sim):

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[1:4]

movie\_indices = [i[0] for i in sim\_scores]

return metadata['title'].iloc[movie\_indices]

**4.5 Web Interfaces:**

We have created user friendly and easy to use client interface. It supports our powerful

backend AI model. Web interface will be hosted on all local machines who have the scripts installed. Web interface helps us to manage the presentation of the results to the user. Using web interface users will be able to view the recommendations, visit the official website and perform other queries.

**Snapshots:**

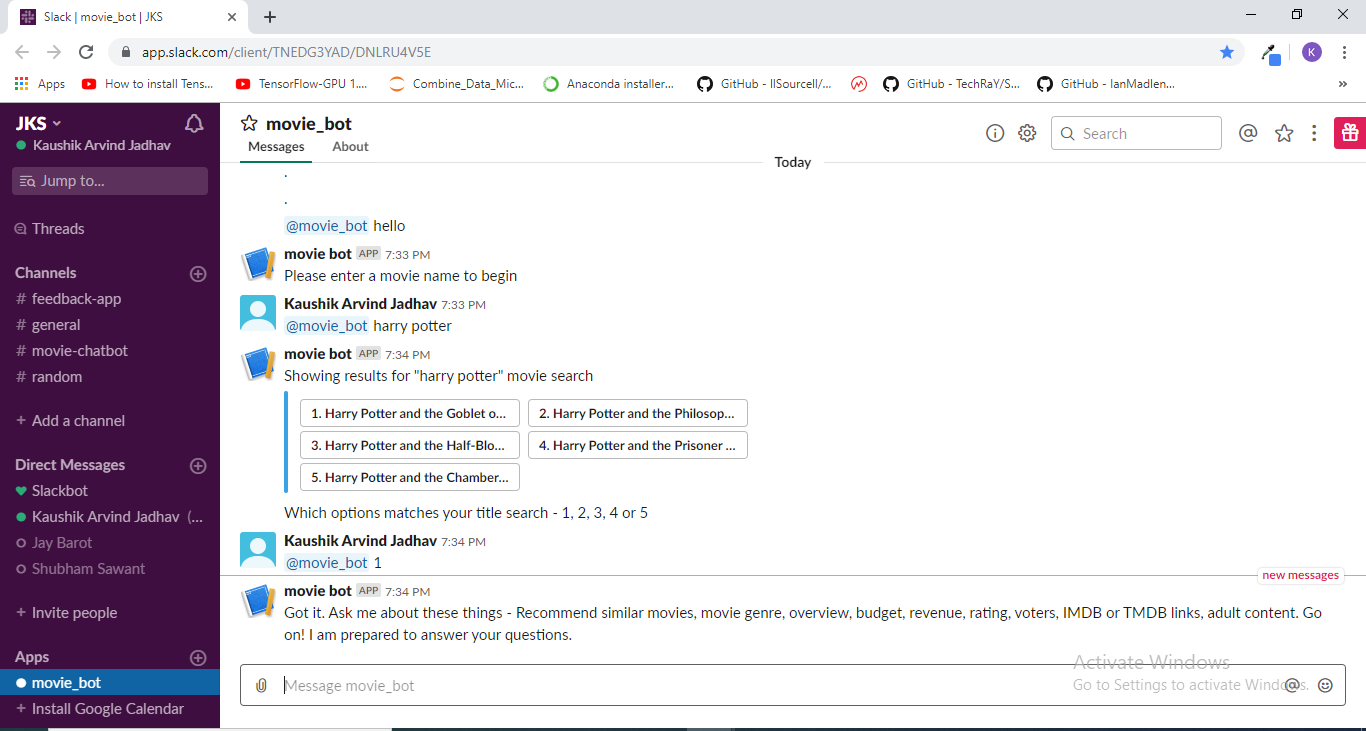


Fig 4.4: Chatbot in Browser Window

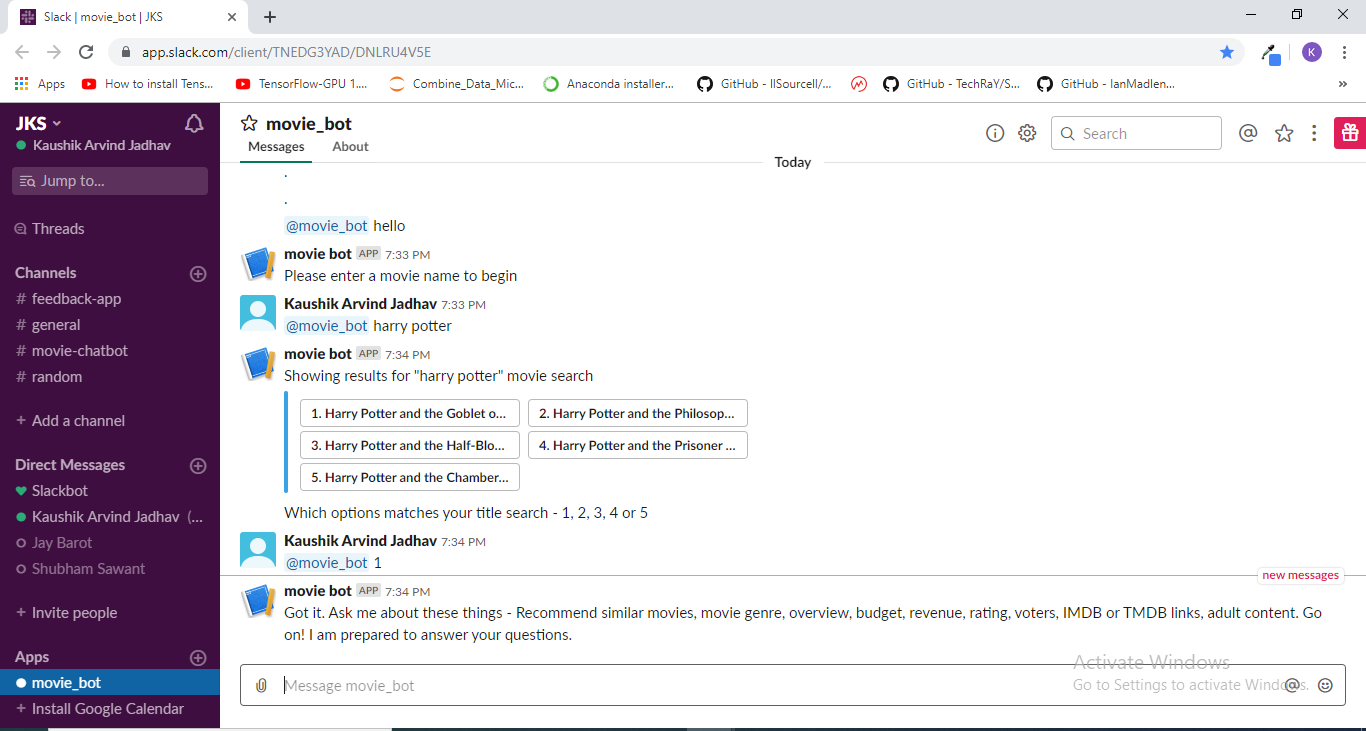


Fig 4.6: Conversation Snapshot-1

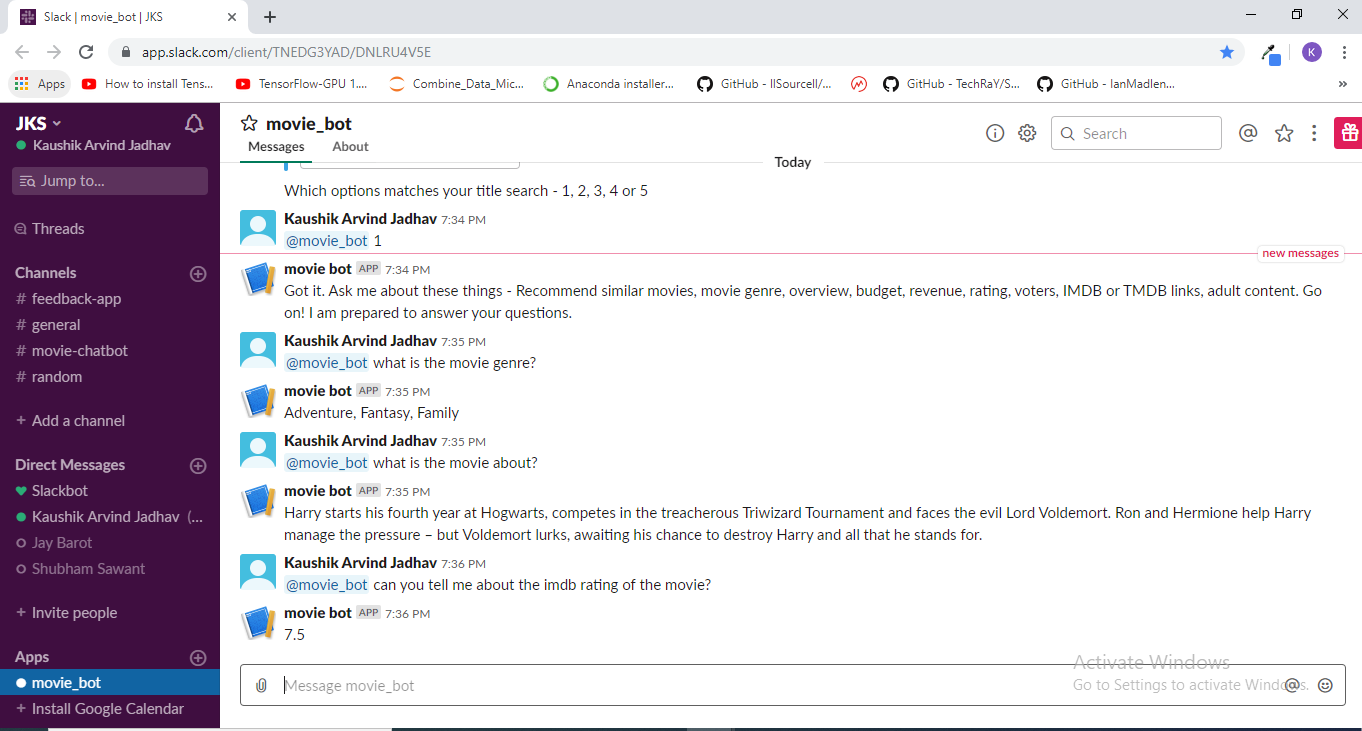


Fig 4.7: Conversation Snapshot-2

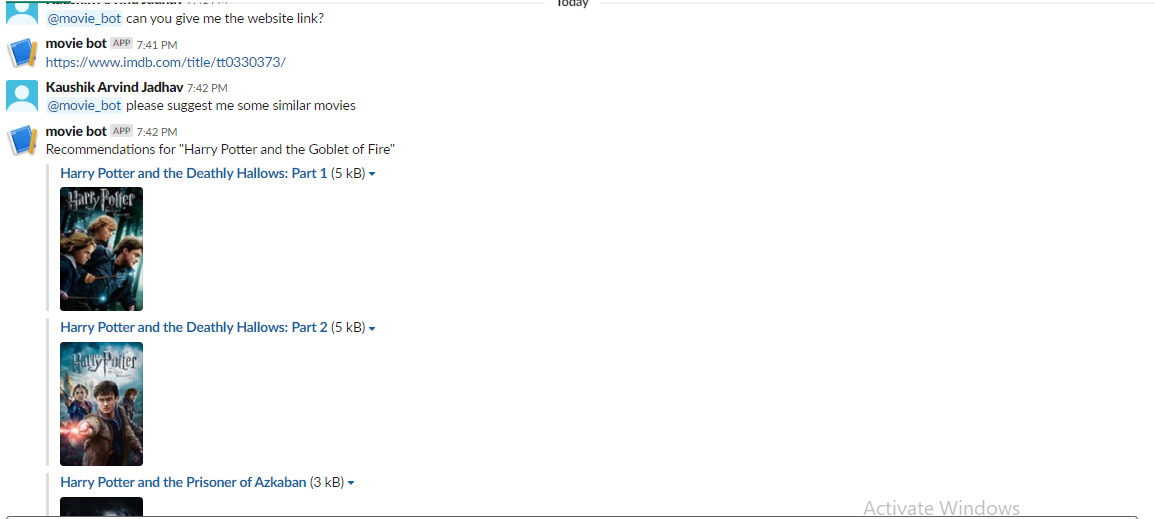


Fig 4.8: Conversation Snapshot-3

**CHAPTER 5**

**CONCLUSION AND FUTURE SCOPE**

**5.1 Conclusion:**

A recommendation system has been implemented based on hybrid approach of collaborative filtering engine and context based engine. We have tried to combine the existing algorithms for recommendation to come up with a hybrid one. It improves the performance by overcoming the drawbacks of traditional recommendation systems. It describes the conventional Content, Collaborative Filtering and Context Filtering recommendation approaches along with their precision, recall and accuracy parameters. This paper has presented a number of utilized evaluation metrics, from which some were used to measure quality, while others to measure performance. Recommender systems make the selection process easier for the users. Hybrid recommendation engine is a competent system to recommend Movies for e-users, whereas the other recommender algorithms are quite slow with inaccuracies. This recommender system will assuredly be a great web application, which can be clubbed with today’s high demanding online purchasing web sites. Our approach can be extended to various domains to recommend books, music, etc.

**5.2 Future Scope:**

There are plenty of way to expand on the work done in this project. Firstly, the content based method can be expanded to include more criteria to help categorize the movies. The most obvious ideas is to add features to suggest movies with common actors, directors or writers. In addition, movies released within the same time period could also receive a boost in likelihood for recommendation. Similarly, the movies total gross could be used to identify a user’s taste in terms of whether he/she prefers large release blockbusters, or smaller indie films. However, the above ideas may lead to overfitting, given that a user’s taste can be highly varied, and we only have a guarantee that 20 movies (less than 0.2%) have been reviewed by the user.

In addition, we could try to develop hybrid methods that try to combine the advantages of both content-based methods and collaborative filtering into one recommendation system.

**References**

[1] James Bennett, Stan Lanning ; “The Netflix Prize”, In KDD Cup and Workshop in conjunction with KDD, 2007.

[2] Mohammad Yahya H. Al-Shamri , Kamal K. Bharadwaj; “A Compact User Model for Hybrid Movie Recommender System ” in International Conference on Computational Intelligence and Multimedia Applications 2007

[3] Costin-Gabriel Chiru, Vladimir-Nicolae Dinu , Ctlina Preda, Matei Macri ; “Movie Recommender System Using the User's Psychological Profile” in IEEE International Conference on ICCP, 2015.

[4] Christina Christakou, Leonidas Lefakis, Spyros Vrettos and Andreas Stafylopatis; “A Movie Recommender System Based on Semi-supervised Clustering ”, IEEE Computer Society Washington,

DC, USA 2015.

[5] Luis M. de Campos, Juan M. Fernández-Luna \*, Juan F. Huete, Miguel A. Rueda-Morales;

“Combining content-based and collaborative recommendations: A hybrid approach based on Bayesian

networks”, International Journal of Approximate Reasoning, revised 2010.

[6] Urszula Kużelewska; “Clustering Algorithms in Hybrid Recommender System on MovieLens Data”,

Studies in Logic, Grammar and Rhetoric, 2014.

[7] Dietmar Jannach, Gerhard Friedrich; “Tutorial: Recommender Systems”, International Joint

Conference on Artificial Intelligence, Beijing, August 4, 2013.

[8] Gaurangi, Eyrun, Nan; “MovieGEN: A Movie Recommendation System”, UCSB.

[9] Harpreet Kaur Virk, Er. Maninder Singh,” Analysis and Design of Hybrid Online Movie

Recommender System ”International Journal of Innovations in Engineering and Technology

(IJIET)Volume 5 Issue 2,April 2015.

[10]Manoj Kumar, D.KYadav, Ankur Singh, Vijay Kr. Gupta,” A Movie Recommender System: MOVREC” International Journal of Computer Applications (0975 – 8887) Volume 124 – No.3, August

2015.

[11]Prerana Khurana , Shabnam Parveen; ‘Approaches of Recommender System: A Survey’; International Journal of Computer Trends and Technology (IJCTT) – Volume 34 Number 3 - April 2016.

[12]Utkarsh Gupta1 and Dr Nagamma Patil2,” Recommender System Based on Hierarchical Clustering Algorithm Chameleon” 2015 IEEE International Advance Computing Conference(IACC).