ITMD – 513 – Open Source Programming

Final Project

Movie Recommendation based on Emotion's

Prof. James Papademas

Project Documentation

Group Member's: Gomathi Sambasivan, Yash Rawani

Project Description:

- One of the hidden focuses of motion pictures is to evoke emotion in their viewers.
- IMDb offers every one of the motion pictures for all genre.
- Subsequently, the movie picture titles can be scratched from the IMDb list to prescribe the user.
- IMDb does not have an API, to get data on movies.
- So, we need to perform scraping. Scraping is utilized for getting to data from a site which is typically finished with APIs.
- We have used BeautifulSoap libraries for Scrapping the data from IMDB website and display it using Flask for GUI.
- Once an Image is been captured it displays user emotion and based on that movies are been recommended.

Libraries Used:

- OpenCV – Taking a photo using Web Camera
- Google.cloud.vision (Sentiment Analysis Machine Learning)
- BeautifulSoup (Web Scrapping)
- Requests (HTTP Requests)
- Flask (GUI micro web framework)
- Sqllite3 – Storing Data in Database
- Tools used: IDLE (Python 3.6 64 bit)

Source Code:

I. CaptureImage.py

```
import cv2


global camera_port, ramp_frames, camera


#Captures a single image from the camera and returns it in PIL format

def clickPicture():
    # Camera 0 is the integrated web cam on my netbook
    camera_port = 0

    #Number of frames to throw away while the camera adjusts to light levels
    ramp_frames = 30
    #Now we can initialize the camera capture object with the cv2.VideoCapture class.
    #All it needs is the index to a camera port.
    camera = cv2.VideoCapture(camera_port)


    def get_image():

     # read is the easiest way to get a full image out of a VideoCapture object.

     retval, im = camera.read()
     return im
    # Ramp the camera - these frames will be discarded and are only used to allow v4l2
    # to adjust light levels, if necessary
    #for i in range(ramp_frames):
     #temp = get_image()
    print("Taking image...")
    # Take the actual image we want to keep
    camera_capture = get_image()
    #file = "D:\Images\Image.jpeg"
    file = "imgs\Image.jpeg"
    # A nice feature of the imwrite method is that it will automatically choose the
    # correct format based on the file extension you provide. Convenient!
    cv2.imwrite(file, camera_capture)

    # You'll want to release the camera, otherwise you won't be able to create a new
    # capture object until your script exits
    del(camera)

clickPicture()
```

II. CloudVisionAnalysis.py

```
import cv2
import argparse
```

```
import io
import os
# Imports the Google Cloud client library
from google.cloud import vision
from google.cloud.vision import types

def getSentiment():
    #Emotions
    emo = ['Angry', 'Surprised','Sad', 'Happy']
    emotion = 'No sentiment'
    ############## Spanish version #################
    os.environ["GOOGLE_APPLICATION_CREDENTIALS"]="Emotion detection movie recom-
485e2d9da22c.json"
    # construct the argument parse and parse the arguments
    ap = argparse.ArgumentParser(description='Process some image to find sentiment in
faces (if any)')
    ap.add_argument("-f", "--file_name", required=False, default="imgs\Image.jpeg",
help="path to image")
    args = vars(ap.parse_args())

    file_name = args["file_name"]
    #path = 'test.jpg'
    # Instantiates a client
    vision_client = vision.ImageAnnotatorClient()
    with io.open(file_name, 'rb') as image_file:
        content = image_file.read()
        image = types.Image(content=content)
    #image = vision_client.image(filename=file_name)

    #faces = image.detect_faces(limit=20)
    faces = vision_client.face_detection(image=image).face_annotations
    #print ('Number of faces: ', len(faces))

    img = cv2.imread(file_name)
    likelihood_name = ('UNKNOWN', 'VERY_UNLIKELY', 'UNLIKELY', 'POSSIBLE',
                       'LIKELY', 'VERY_LIKELY')
    for face in faces:

        sentiment =
[likelihood_name[face.surprise_likelihood],likelihood_name[face.anger_likelihood],like
lihood_name[face.sorrow_likelihood],likelihood_name[face.joy_likelihood] ]

        for item, item2 in zip(emo, sentiment):
            print(item, ": ", item2)

        if not (all( item == 'VERY_UNLIKELY' for item in sentiment) ):

            if any( item == 'VERY_LIKELY' for item in sentiment):
                state = sentiment.index('VERY_LIKELY')
                # the order of enum type Likelihood is:
                #']\', 'POSSIBLE', 'UNKNOWN', 'UNLIKELY', 'VERY_LIKELY',
'VERY_UNLIKELY'
                # it makes sense to do argmin if VERY_LIKELY is not present, one would
espect that VERY_LIKELY
                # would be the first in the order, but that's not the case, so this
special case must be added
            elif any( item == 'LIKELY' for item in sentiment):
                state = sentiment.index('LIKELY')
            elif any( item == 'UNLIKELY' for item in sentiment):
                state = sentiment.index('UNLIKELY')

            else:
                state = sentiment.index('POSSIBLE')#np.argmin(sentiment)
```

```
            emotion = emo[state]
        print(emotion)
        box = [(vertex.x, vertex.y)
                    for vertex in face.bounding_poly.vertices]
        pts = box + [box[0]]
        print(pts[0],pts[2])

        cv2.putText(img,emotion, pts[0], cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,255), 2)
        cv2.rectangle(img, pts[0], pts[2], (0, 255, 0), 2)

    cv2.imshow("Analysis", img)
    cv2.waitKey(0)
    cv2.imwrite('static/output_'+emotion+'.jpeg',img)
    return emotion
```

III. ImdbWebScrap.py

```
# Python3 code for movie
# recommendation based on
# emotion

# Import library for web
# scrapping
from bs4 import BeautifulSoup as SOUP
import requests as HTTP

# Function for scraping
def getMovies(emotion):
        urlhere="No url"
        data = []
   # IMDb Url for Drama genre of
   # movie against emotion Sad
        if(emotion == "Sad"):
                urlhere =
'http://www.imdb.com/search/title?genres=drama&amp;title_type=feature&amp;sort=movieme
ter, asc'

   # IMDb Url for Musical genre of
   # movie against emotion Disgust
        elif(emotion == "Disgust"):
                urlhere =
'http://www.imdb.com/search/title?genres=musical&amp;title_type=feature&amp;sort=movie
meter, asc'

   # IMDb Url for Family genre of
   # movie against emotion Anger
        elif(emotion == "Angry"):
                urlhere =
'http://www.imdb.com/search/title?genres=family&amp;title_type=feature&amp;sort=moviem
eter, asc'


   # IMDb Url for Thriller genre of
   # movie against emotion Anticipation
        elif(emotion == "Anticipation"):
                urlhere =
'http://www.imdb.com/search/title?genres=thriller&amp;title_type=feature&amp;sort=movi
```

```
emeter, asc'

    # IMDb Url for Sport genre of
    # movie against emotion Fear
        elif(emotion == "Surprised"):
                urlhere =
'http://www.imdb.com/search/title?genres=sport&amp;title_type=feature&amp;sort=movieme
ter, asc'

    # IMDb Url for Thriller genre of
    # movie against emotion Enjoyment
        elif(emotion == "Happy"):
                urlhere =
'http://www.imdb.com/search/title?genres=thriller&amp;title_type=feature&amp;sort=movi
emeter, asc'

    # IMDb Url for Western genre of
    # movie against emotion Trust
        elif(emotion == "Trust"):
                urlhere =
'http://www.imdb.com/search/title?genres=western&amp;title_type=feature&amp;sort=movie
meter, asc'

    # IMDb Url for Film_noir genre of
    # movie against emotion Surprise
        elif(emotion == "Surprise"):
                urlhere =
'http://www.imdb.com/search/title?genres=film_noir&amp;title_type=feature&amp;sort=mov
iemeter, asc'
        if(urlhere != "No url"):

                # HTTP request to get the data of
                # the whole page
                response = HTTP.get(urlhere)
                data = response.text

                # Parsing the data using
                # BeautifulSoup
                soup = SOUP(data, "lxml")

                # Extract movie titles from the
                # data using regex

                samples = soup.find_all("div", "lister-item")
                ratings = soup.find_all("div","ratings-bar")
                #print(ratings[0].contents[1].attrs['data-value'])
                #data = []
                name = []
                img = []
                rating = []
                for a in samples:
                    name.append(a.contents[5].contents[1].contents[3].text)

img.append(a.contents[3].contents[1].contents[1].attrs['loadlate'])

                for rate in ratings:
                    rating.append(rate.contents[1].attrs['data-value'])


                data = zip(name,img,rating)
                data = list(data)
        return data
```

```
# Driver Function
"""if __name__ == '__main__':

    emotion = input("Enter the emotion: ")
    a = main(emotion)
    count = 0

    if(emotion == "Happy" or emotion == "Angry"
                          or emotion=="Surprise"):

        for i in a:

            # Splitting each line of the
            # IMDb data to scrape movies
            tmp = str(i).split('>;')

            if(count > 13):
                break
            count += 1
            #print(tmp)
    else:
        for i in a:
            tmp = str(i).split('>')

            if(len(tmp) == 3):
                print(tmp[1][:-3])

            if(count > 11):
                break
            count+=1
"""
```

IV. Main.py

```
from flask import Flask,request,render_template
#user-defined modules
import ImdbWebScrape
import CaptureImage
import CloudVisionAnaylsis
import watchlist_db

app = Flask(__name__)

#Capture User Picture
CaptureImage.clickPicture()
#Run analysis with Google Cloud Vision Api
emotion = CloudVisionAnaylsis.getSentiment()

#Get scraped data by passing the emotion state to imdb website
data = ImdbWebScrape.getMovies(emotion)

#Storing in database
watchlist_db.dropTable()
watchlist_db.createTable()
watchlist_db.insertMovies(data)
```

```
@app.route("/")
def index():
    movies = []
    img = '/static/output_'+emotion+'.jpeg'
    movies=watchlist_db.fetchMovies()
    return render_template("index.html",movies=movies,emotion=emotion,img=img)
@app.route("/test")
def test():

    return render_template("test.html",data=data)

@app.route("/watchlist")
def watchlist():
    movies = []
    movies=watchlist_db.fetchWatchlist()
    return render_template("watchlist.html",movies=movies)

@app.route("/<int:id>")
def updateWatchlist(id):
    movies = []
    img = '/static/output_'+emotion+'.jpeg'
    watchlist_db.updateMovie(id)
    movies=watchlist_db.fetchMovies()
    return render_template("index.html",movies=movies,emotion=emotion,img=img,id=id)

if __name__ == '__main__':
    app.run()
```

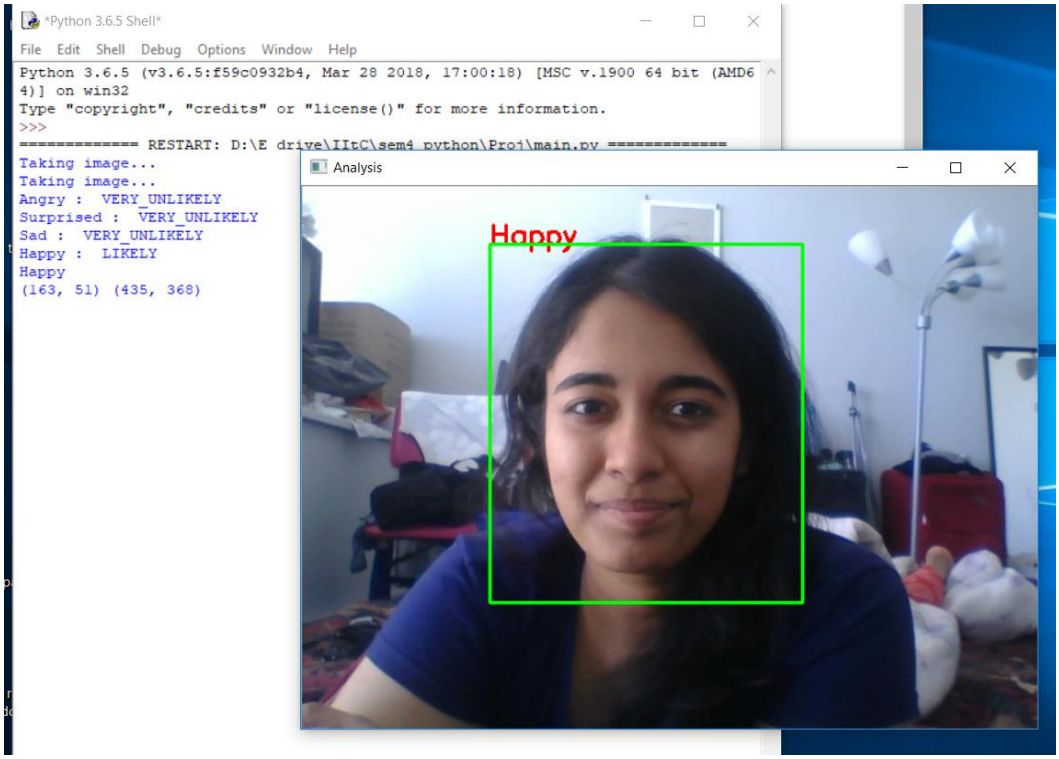
V. WatchList_db.py

```
import sqlite3 as lite

def createTable():
    try:

        conn.execute('CREATE TABLE IF NOT EXISTS WATCHLIST(movie_ID INTEGER PRIMARY
KEY AUTOINCREMENT, title TEXT NOT NULL, img TEXT NOT NULL, rating TEXT NOT NULL,
isWatch INTEGER);')
        print("Table created successfully.")

    except Exception as e:
        print("\n Error in creating Table. Table already exists.%s" % e.args[0])
        exit()

def insertMovies(data):
    try:
        for i,j,k in data:
            values=(i,j,k,0)
            conn.execute("INSERT INTO WATCHLIST (title,img,rating,isWatch) VALUES
(?,?,?,?)",values)
            conn.commit()


    except Exception:
        conn.rollback()
        print("Error:", name, ".The contact already exists.")
```

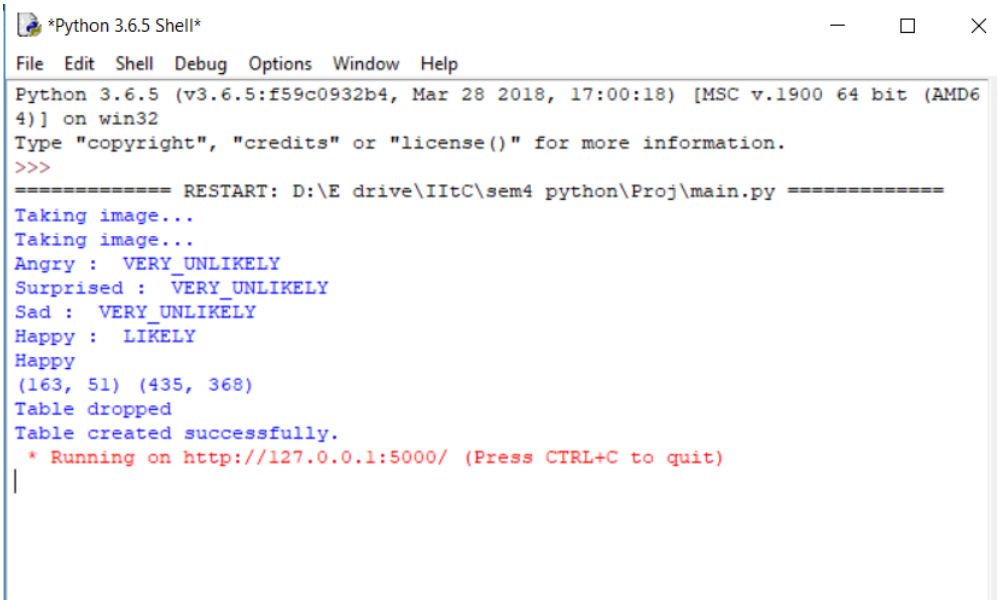
```
def fetchWatchlist():
    try:
        saved = []
        cur.execute("SELECT * FROM WATCHLIST WHERE isWatch=1")
        rows = cur.fetchall()
        for movie in rows:
            saved.append(movie)
        return saved
    except Exception:
        return 0

def fetchMovies():
    try:
        movies = []
        value = 0
        cur.execute("SELECT * FROM WATCHLIST WHERE isWatch=0")
        rows = cur.fetchall()
        for movie in rows:
            movies.append(movie)
        return movies
    except Exception as e:
        print(e)
        return 0

def updateMovie(movieid):
    try:
        values=(1,movieid)
        cur.execute(
            "UPDATE WATCHLIST SET isWatch=? WHERE movie_ID=?",values)
        conn.commit()

    except Exception:
        conn.rollback()
        print("Error in updating the contact")

def dropTable():
     conn.execute('DROP TABLE IF EXISTS WATCHLIST')
     conn.commit()
     print("Table dropped")
conn = None



try:
    conn = lite.connect('watchlist.db')
    cur = conn.cursor()


except lite.Error as e:
    print ("Error %s" % e.args[0])
    sys.exit(1)
```

Screenshots:

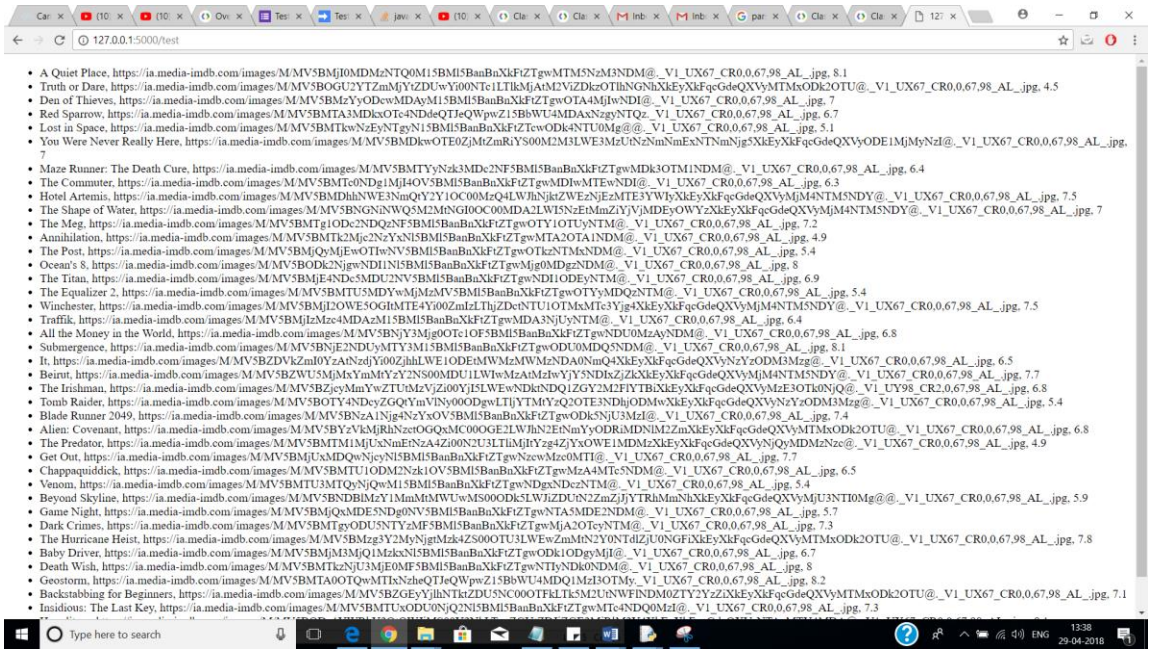
i. Displaying Image taken from web camera and displaying user sentiments.



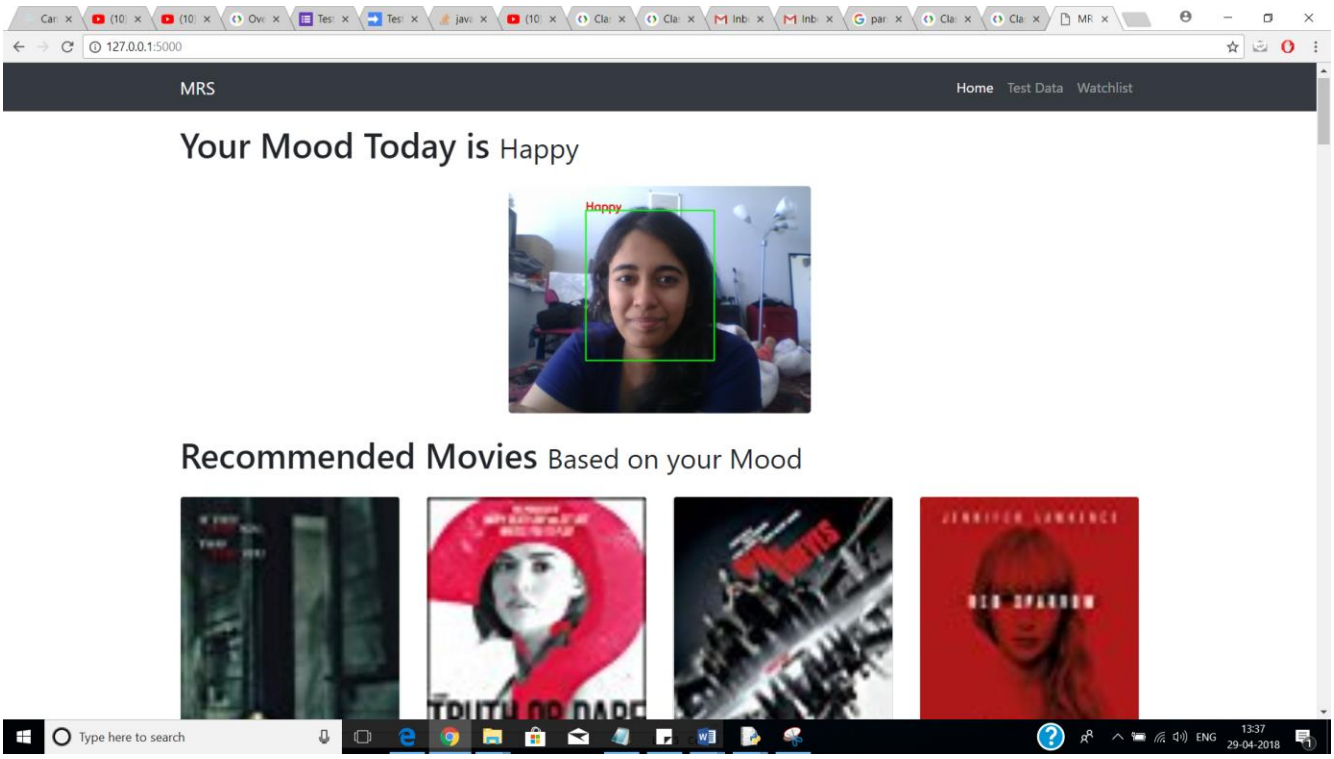
ii. Image of running project with the server up and running.



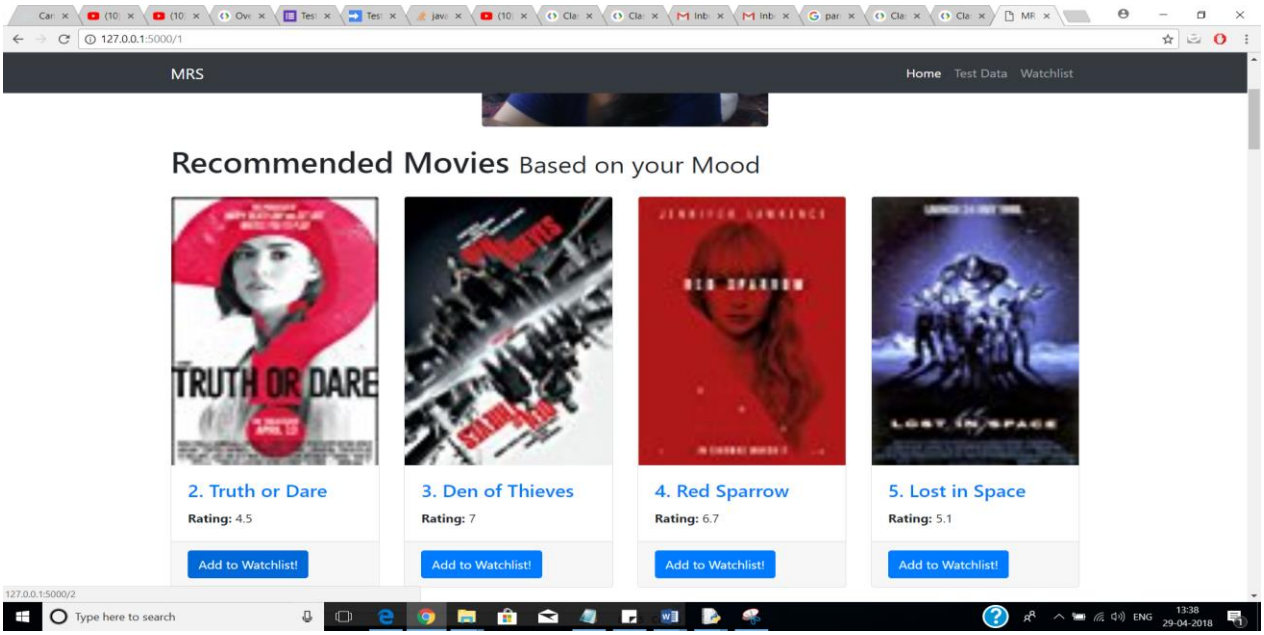
iii. Screenshot of scrapping data from IMDB website.



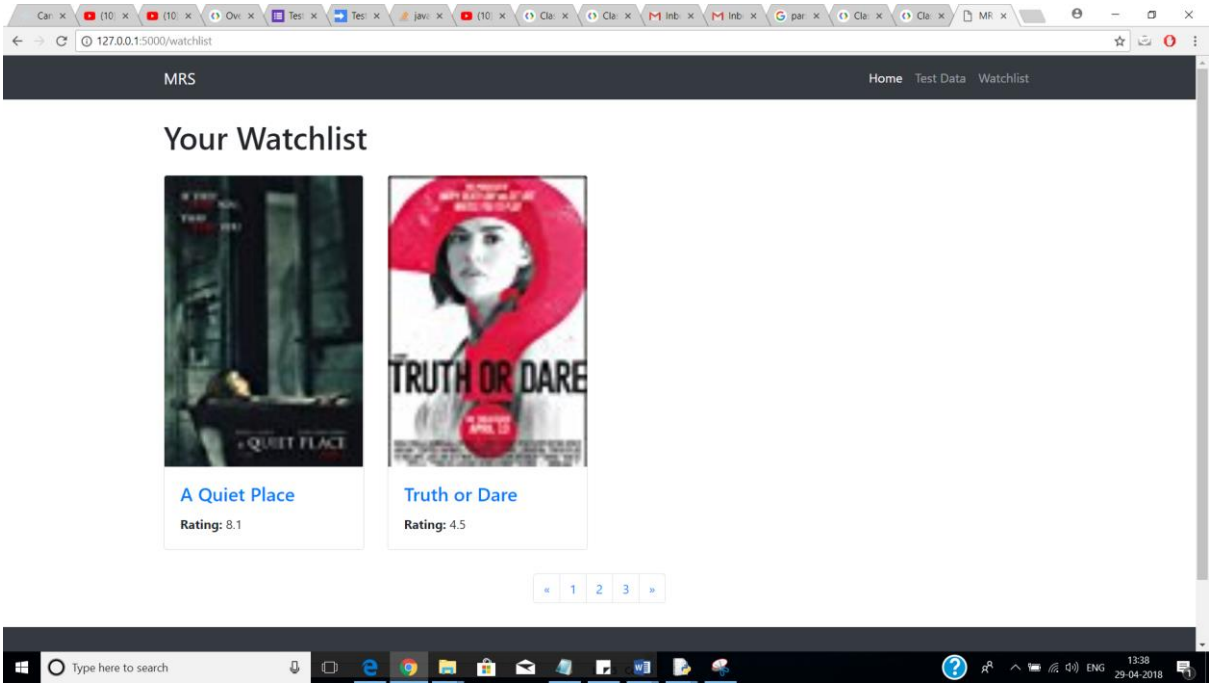
iv. Screenshot of running flask server on port 5000 and the scraped data is inserted into sqlite db and retrieved in the home page.



v. Once a user clicks add to watchlist button the isWatch flag field is set to true and the movie info goes to the watchlist route where only the movies set to isWatch is retrieved.



vi. Watchlist grid.


127.0.0.1:5000/watchlist
MRS
Home
Test Data
Watchlist
Your Watchlist
A QUIET PLACE
TRUTH OR DARE
A Quiet Place
Rating: 8.1
Truth or Dare
Rating: 4.5
« 1 2 3 »
Type here to search
ENG
13:38
29-04-2018