Machine Learning Framework for predicting Empathy using Eye tracking and Pupil Dilation

MSc Research Project
Data Analytics

Akshay Dilip Sayar
Student ID: 20211121

School of Computing
National College of Ireland

Supervisor: Dr. Anu Sahni
Dr. Paul Stynes

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Akshay Dilip Sayar
Student ID:	20211121
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Anu Sahni
Submission Due Date:	19/09/2022
Project Title:	Machine Learning Framework for predicting Empathy using Eye tracking and Pupil Dilation
Word Count:	2367
Page Count:	16

Signature:	Akshay Dilip Sayar
Date:	18th September 2022

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual
Machine Learning Framework for predicting Empathy using Eye tracking and Pupil Dilation

Akshay Dilip Sayar
20211121

1 Introduction

The purpose of the report is provide step by step instructions to implement the research - 'Machine Learning Framework for predicting empathy using Eye tracking and Pupil Dilation'. The configuration manual describes the step-by-step details performed in the completion of the research. The research aims to determine how efficiently a empathetic person can be recognized based on pupil dilation and eye tracking features. For the implementation of this research, we have used multiple machine learning algorithms to detect empathetic people and compare the performance of these machine learning algorithms. The following points shows structure of the report.

- Section 2 Hardware and Software Requirements:
- Section 3 Data Collection: In this section data collection steps are discussed.
- Section 4 Data Pre-processing: Shows how data was preprocessed and features were extracted from the raw data.
- Section 5 Implementation: This show how and which machine learning models were applied to predict the empathy of a person.
- Section 6 Conclusion: Conclusion of the report.

2 Hardware and Software Requirements

The detail of the system configuration used in the research is shown in table 1.

Operating System	Windows 10
Installed Memory (RAM)	8 GB
Processor	Ryzen 7 5800
Graphic Processor	Nvidia GTX 1650

Table 1: System Configuration

For implementation of the research, Python Programming Language has been used, using Jupyter Notebook as IDE. The detail of libraries and programming languages used for the research are mentioned below : - Python 3.8.5

- Jupyter Lab 3.0.14
- Pandas
- Numpy
- Tensorflow
- Dlib
- Yolov5
- Cv2
- Ellseg framework
- scikit learn – os

3 Data Collection

Data collection starts by conducting experiments on participants. The experiment starts with signing the consent form by the participant. After that self reported parameters are noted down like age, gender, sadness level on the scale of 1-10 (1 being least sad and 10 being extremely sad) before the experiment begins. The participants sits on a chair and a laptop is place in front of him. The participants gets a headphone to listen to the audio clearly. Then eye tracking glasses are given to the participant which is then calibrated on three point scale, the three points being top left of the laptop screen, top right and bottom centre of the screen. Then the pupil recording starts on the mobile and video is played on the laptop. The participant is left alone to watch the video. The video is 13 minute long video which is narrated by actors. The stories are sad emotion based stories narrated by four different actors and are used from the research papers **?**. After the video is finished the participants are asked to report the sadness level after watching the video again on the scale of 1-10. Then the participant is asked to memory questionnaire which has ten questions based on the video they just watched. For each right questions is given 1 mark and no negative marks for wrong answers. The final step of the experiment is that the participants has to answer the emotion based empathy questionnaire **?** the score of which will be used as to check if the parson is highly empathetic or not. Data collection for one person ends her and then these same steps are used on 53 participants.

4 Data Pre-processing

In this section we will discuss step by step process of data pre-processing for the research.

4.1 Data Preparation

After the experiment is conducted, the video and the folder which contains the data related to the experiment is transferred to the laptop in which the BeGaze software was installed. The video is processed and point of gaze is extracted which is shown on the screen as orange color circle. The event matrix was also extracted from the BeGaze software. The video extracted from the BeGaze software was in 1000 frame per seconds

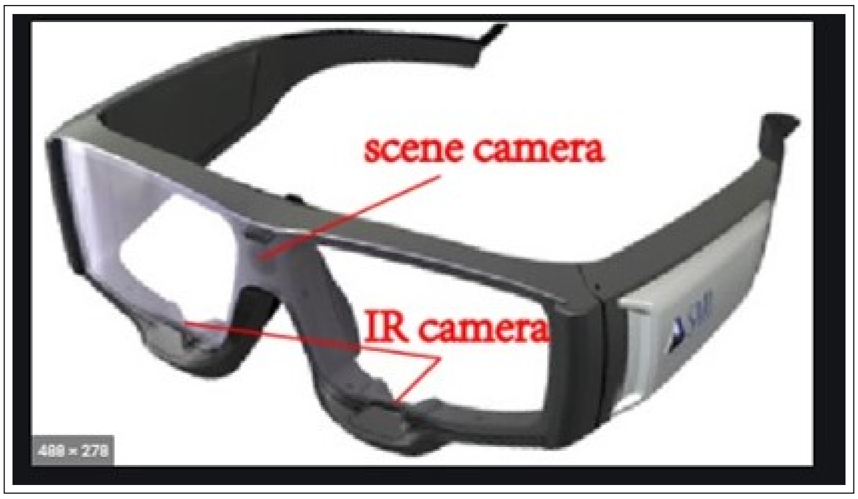
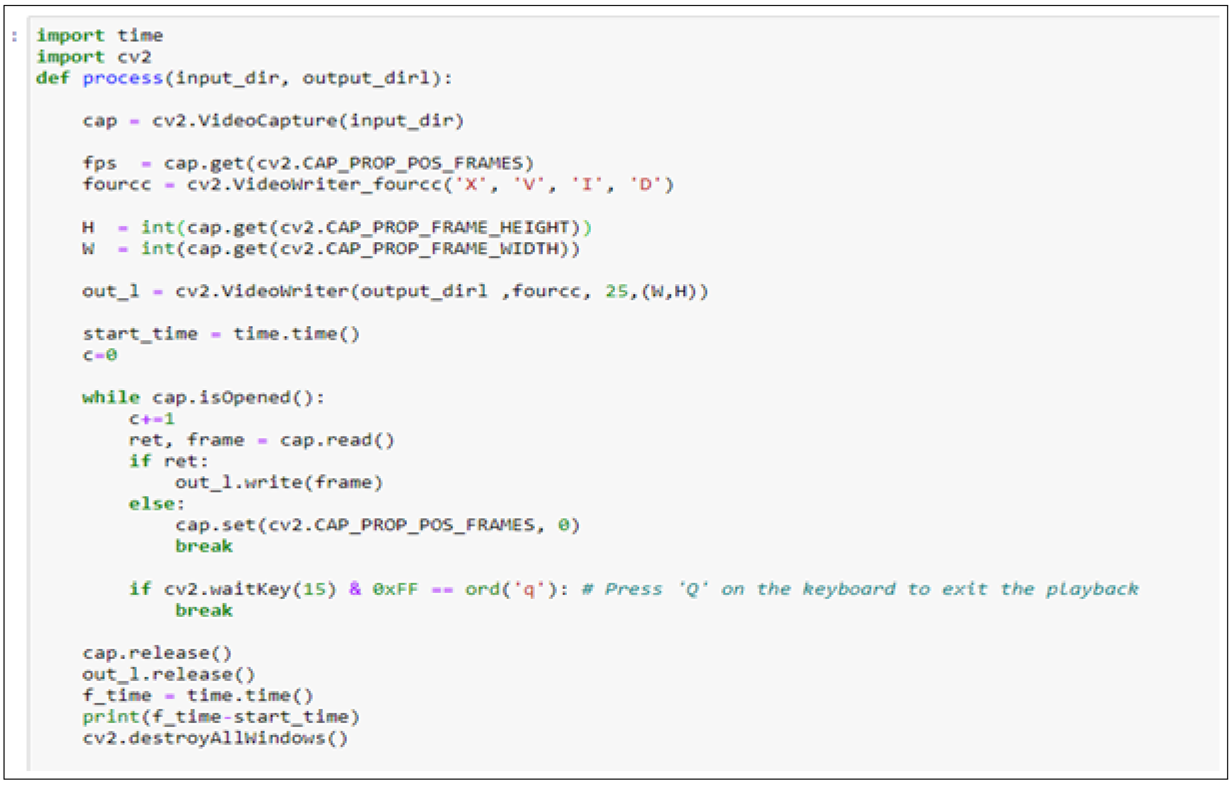


Figure 1: Eye tacking glasses

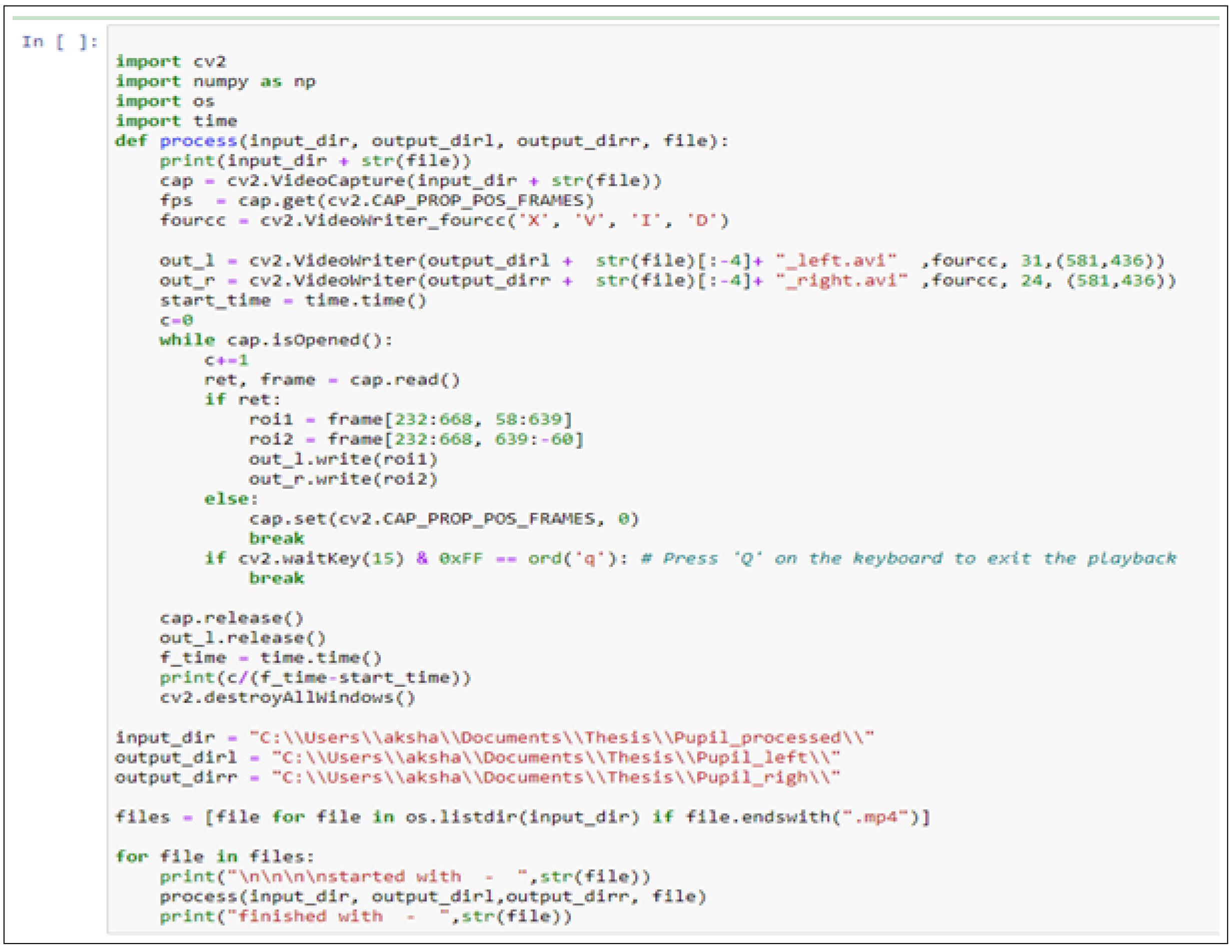
to convert it into 25 frame per seconds the script shown in figure 2 was used and the name of the file which has the script is '1000to25fps.py'. The output of the script is a video in mp4 format.



```
import time
import cv2
def process(input_dir, output_dirl):

    cap = cv2.VideoCapture(input_dir)

    fps  = cap.get(cv2.CAP_PROP_POS_FRAMES)
    fourcc = cv2.VideoWriter_fourcc('X', 'V', 'I', 'D')

    H  = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    W  = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))

    out_l = cv2.VideoWriter(output_dirl ,fourcc, 25,(W,H))

    start_time = time.time()
    c=0

    while cap.isOpened():
        c+=1
        ret, frame = cap.read()
        if ret:
            out_l.write(frame)
        else:
            cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
            break

        if cv2.waitKey(15) & 0xFF == ord('q'): # Press 'Q' on the keyboard to exit the playback
            break

    cap.release()
    out_l.release()
    f_time = time.time()
    print(f_time-start_time)
    cv2.destroyAllWindows()
```

Figure 2: Script to convert 1000 fps videos to 25 fps

The pupil video contained videos recording of both the eyes in one frame and hence it was videos using the script shown in figure 3. The output of the script are 2 videos in mp4 format. The name of the file is 'pupil_video_crop.py'.



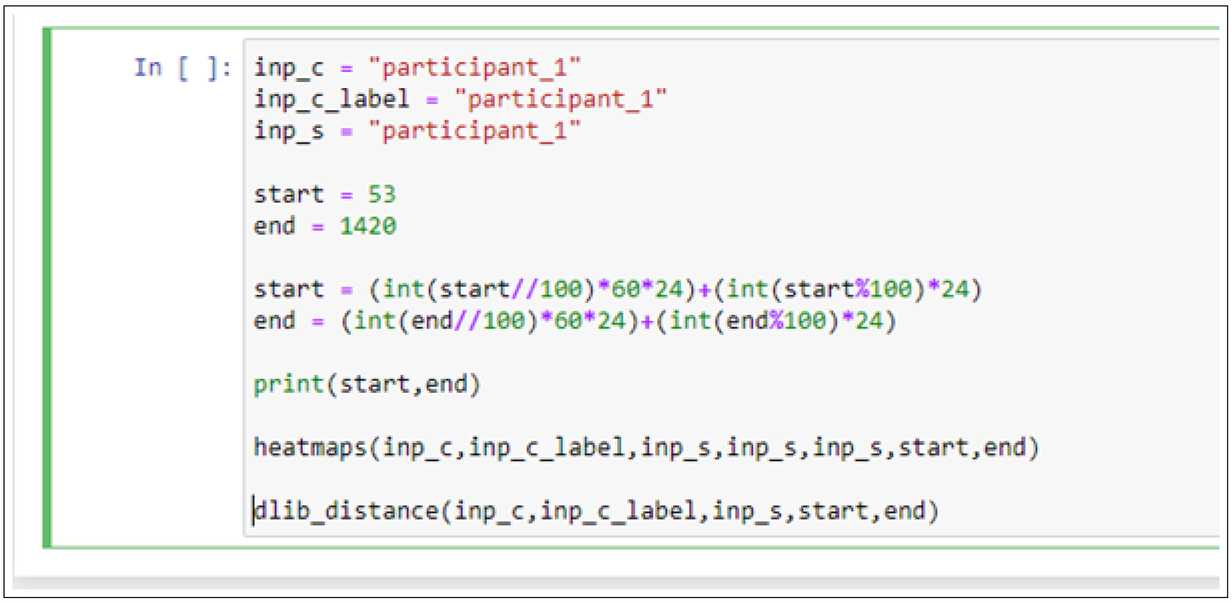
```
In [ ]:
import cv2
import numpy as np
import os
import time
def process(input_dir, output_dirl, output_dirr, file):
    print(input_dir + str(file))
    cap = cv2.VideoCapture(input_dir + str(file))
    fps  = cap.get(cv2.CAP_PROP_POS_FRAMES)
    fourcc = cv2.VideoWriter_fourcc('X', 'V', 'I', 'D')

    out_l = cv2.VideoWriter(output_dirl +  str(file)[:-4]+ "_left.avi"  ,fourcc, 31,(581,436))
    out_r = cv2.VideoWriter(output_dirr +  str(file)[:-4]+ "_right.avi" ,fourcc, 24, (581,436))
    start_time = time.time()
    c=0
    while cap.isOpened():
        c+=1
        ret, frame = cap.read()
        if ret:
            roi1 = frame[232:668, 58:639]
            roi2 = frame[232:668, 639:-60]
            out_l.write(roi1)
            out_r.write(roi2)
        else:
            cap.set(cv2.CAP_PROP_POS_FRAMES, 0)
            break
        if cv2.waitKey(15) & 0xFF == ord('q'): # Press 'Q' on the keyboard to exit the playback
            break

    cap.release()
    out_l.release()
    f_time = time.time()
    print(c/(f_time-start_time))
    cv2.destroyAllWindows()

input_dir = "C:\\Users\\aksha\\Documents\\Thesis\\Pupil_processed\\"
output_dirl = "C:\\Users\\aksha\\Documents\\Thesis\\Pupil_left\\"
output_dirr = "C:\\Users\\aksha\\Documents\\Thesis\\Pupil_righ\\"

files = [file for file in os.listdir(input_dir) if file.endswith(".mp4")]

for file in files:
    print("\n\n\n\nstarted with  -  ",str(file))
    process(input_dir, output_dirl,output_dirr, file)
    print("finished with  -  ",str(file))
```

Figure 3: Script to split a pupil recording video in 2 video (one for right eye and one for left eye)

4.2 Feature Extraction

In this section we will discuss the feature extraction performed for the research. To extract heat map of point of gaze, two yolo models were created one for detecting the point of gaze circle and the other for detecting laptop screen. Yolov5 folder was downloaded from the github site of Ultralytics. Data was created by taking screenshot of the frame of videos and labelling them using labelimg software which is opensource. Total of 124 images were used to train point of gaze circle and 215 images for laptop screen detection. The train.py from yolov5 was used to train a model which would give us weights of the model in ".pt" format. Both the model were trained for 300 epochs and batch size of 8. To detect the point of gaze circle and screen the following command was used, which outputs a ".txt" file which has coordinates of circle/screen for one single frame of the video. python detect.py –weights /path_to_model_weights/best_circle_weight.pt –img 640 –conf 0.25 –source path_to_videoparticipant_1.avi –save-txt –nosave –save-conf –name participant_1 –project circle_or_screen/

To generate average distance from eye dlib library was used to get the face landmarks of the actors present in the screen of field of gaze. The script used for the same is in jupyter file "get_face_landmarks_dlib.ipynb" which used weight stored in "shape_predictor_68_face _landmarks.dat". The scripts outputs a ".csv" file which contains coordinates of eyes of actors from each frame of the video. Using the data from the above yolov5 inference heat map could be created and using point of gaze circle coordinate from yolov5 and coordinates of eyes from the dlib module average distance from each eye can be calculated. For both heat map and average distance of point and gaze and eyes of actors a single scripts was written, which can be called (figure 4 for all the participants.



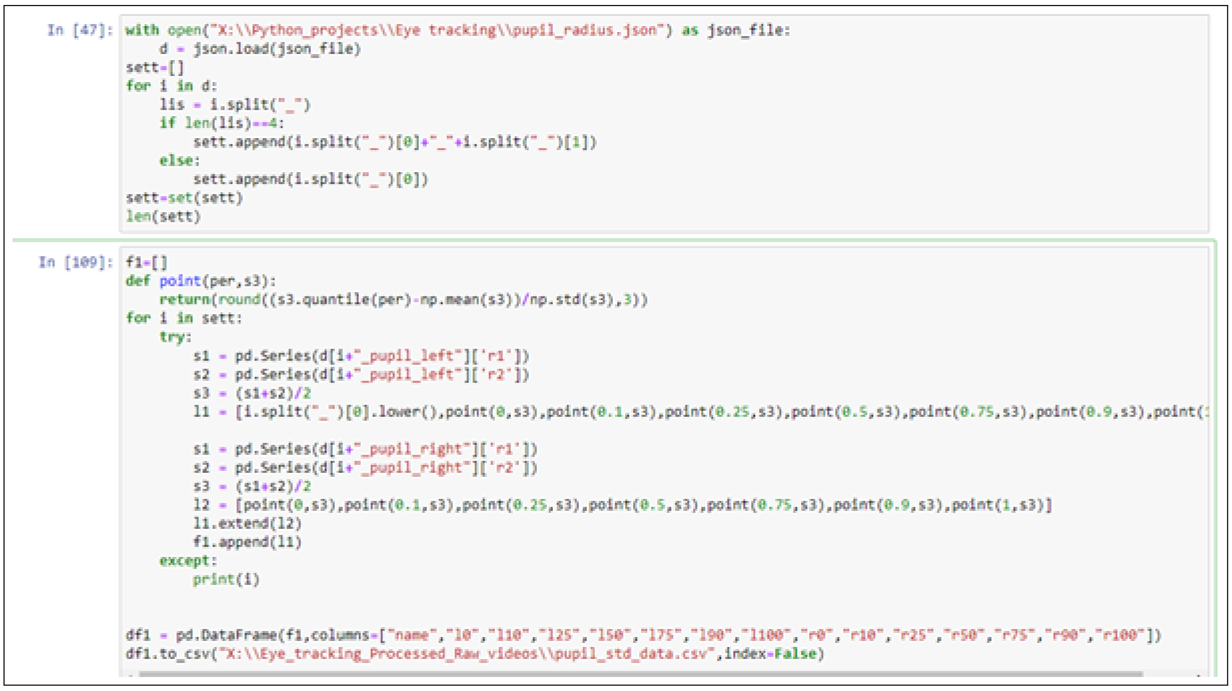
```
In [ ]: inp_c = "participant_1"
        inp_c_label = "participant_1"
        inp_s = "participant_1"

        start = 53
        end = 1420

        start = (int(start//100)*60*24)+(int(start%100)*24)
        end = (int(end//100)*60*24)+(int(end%100)*24)

        print(start,end)

        heatmaps(inp_c,inp_c_label,inp_s,inp_s,inp_s,start,end)

        dlib_distance(inp_c,inp_c_label,inp_s,start,end)
```

Figure 4: Calling definition to create heatmap and average distance from eyes.

In the figure 6, "heatmaps" creates the heat map of point and gaze and "dlib_distance" calculates the average distance between point of gaze and eyes of actor. Start and end is the start and end of time of a video in seconds. The output of the script is a CSV file which has average distance from point of gaze to actors left and right eyes and heatmap in ".png" format.

To get peak pupil dilation and other pupil features, ellseg framework from was used.

Some modifications were made to extract just the pupil radii from the script. The pupil video for left and right eyes which were extracted in the last step will be used as input and as the output two radii for the pupil is extracted for each frame of the video. Hence for each participant the script was ran twice, one for left eye and one for right eye which gives data for r1 and r2 (two radii of ellipse) and this was saved in json format. Once all the data was extracted for all the participants and saved to json file format, featured from this was extracted. First the average of r1 and r2 was taken for each frame for each eye of a participant. The features include peak pupil dilation, 0,10,25,50,75,90 and 100 percentile of the distribution. All the features were extracted and to normalize the data was divided by the smallest size (radius) of pupil. This was done for all the participants and features from the pupil dilation was extracted. "evaluate_ellseg_thesis.py" was used to extract pupil radii of each eye of a participant. The script is taken from github and the research paper is cited in the report. The script has been updated to extract only useful information. The output of the script is ".json" file which has list of all the radii of pupil for each frame of the video. "pupil_feature_from_raw_json.ipynb" contains the script (shown in figure 5) which extracts information from the raw json file like peak pupil dilation and 0,10,25,50,75,90 and 100 percentile of the distribution. The output of the script is a CSV file which has all the feature for each participant.

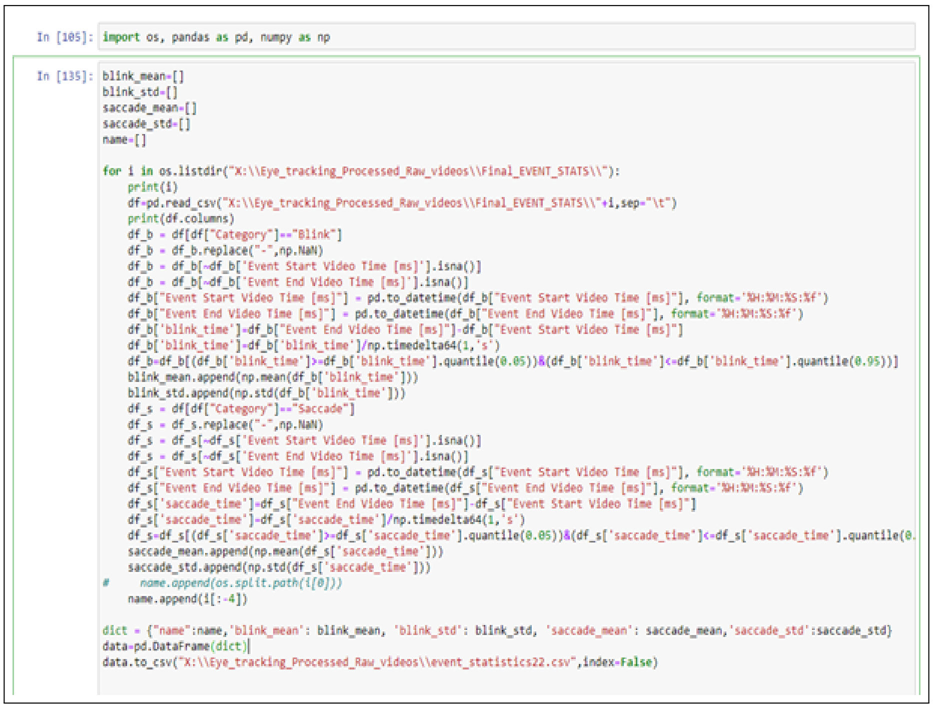


```
In [47]: with open("X:\\Python_projects\\Eye tracking\\pupil_radius.json") as json_file:
             d = json.load(json_file)
         sett=[]
         for i in d:
             lis = i.split("_")
             if len(lis)==4:
                 sett.append(i.split("_")[0]+"_"+i.split("_")[1])
             else:
                 sett.append(i.split("_")[0])
         sett=set(sett)
         len(sett)
```

```
In [109]: f1=[]
          def point(per,s3):
              return(round((s3.quantile(per)-np.mean(s3))/np.std(s3),3))
          for i in sett:
              try:
                  s1 = pd.Series(d[i+"_pupil_left"]['r1'])
                  s2 = pd.Series(d[i+"_pupil_left"]['r2'])
                  s3 = (s1+s2)/2
                  l1 = [i.split("_")[0].lower(),point(0,s3),point(0.1,s3),point(0.25,s3),point(0.5,s3),point(0.75,s3),point(0.9,s3),point(

                  s1 = pd.Series(d[i+"_pupil_right"]['r1'])
                  s2 = pd.Series(d[i+"_pupil_right"]['r2'])
                  s3 = (s1+s2)/2
                  l2 = [point(0,s3),point(0.1,s3),point(0.25,s3),point(0.5,s3),point(0.75,s3),point(0.9,s3),point(1,s3)]
                  l1.extend(l2)
                  f1.append(l1)
              except:
                  print(i)


          df1 = pd.DataFrame(f1,columns=["name","l0","l10","l25","l50","l75","l90","l100","r0","r10","r25","r50","r75","r90","r100"])
          df1.to_csv("X:\\Eye_tracking_Processed_Raw_videos\\pupil_std_data.csv",index=False)
```

Figure 5: Script to create features from pupil dilation

To create features from the event statistics which was generated from the BeGaze software. The script from figure 6 was used. The output of the script is extracted features like blinks_per, blink_mean, saccade_per etc in CSV format for all the participants.



```
In [105]: import os, pandas as pd, numpy as np

In [135]: blink_mean=[]
          blink_std=[]
          saccade_mean=[]
          saccade_std=[]
          name=[]

          for i in os.listdir("X:\\Eye_tracking_Processed_Raw_videos\\Final_EVENT_STATS\\"):
              print(i)
              df=pd.read_csv("X:\\Eye_tracking_Processed_Raw_videos\\Final_EVENT_STATS\\"+i,sep="\t")
              print(df.columns)
              df_b = df[df["Category"]=="Blink"]
              df_b = df_b.replace("-",np.NaN)
              df_b = df_b[~df_b['Event Start Video Time [ms]'].isna()]
              df_b = df_b[~df_b['Event End Video Time [ms]'].isna()]
              df_b["Event Start Video Time [ms]"] = pd.to_datetime(df_b["Event Start Video Time [ms]"], format='%H:%M:%S:%f')
              df_b["Event End Video Time [ms]"] = pd.to_datetime(df_b["Event End Video Time [ms]"], format='%H:%M:%S:%f')
              df_b['blink_time']=df_b["Event End Video Time [ms]"]-df_b["Event Start Video Time [ms]"]
              df_b['blink_time']=df_b['blink_time']/np.timedelta64(1,'s')
              df_b=df_b[(df_b['blink_time']>=df_b['blink_time'].quantile(0.05))&(df_b['blink_time']<=df_b['blink_time'].quantile(0.95))]
              blink_mean.append(np.mean(df_b['blink_time']))
              blink_std.append(np.std(df_b['blink_time']))
              df_s = df[df["Category"]=="Saccade"]
              df_s = df_s.replace("-",np.NaN)
              df_s = df_s[~df_s['Event Start Video Time [ms]'].isna()]
              df_s = df_s[~df_s['Event End Video Time [ms]'].isna()]
              df_s["Event Start Video Time [ms]"] = pd.to_datetime(df_s["Event Start Video Time [ms]"], format='%H:%M:%S:%f')
              df_s["Event End Video Time [ms]"] = pd.to_datetime(df_s["Event End Video Time [ms]"], format='%H:%M:%S:%f')
              df_s['saccade_time']=df_s["Event End Video Time [ms]"]-df_s["Event Start Video Time [ms]"]
              df_s['saccade_time']=df_s['saccade_time']/np.timedelta64(1,'s')
              df_s=df_s[(df_s['saccade_time']>=df_s['saccade_time'].quantile(0.05))&(df_s['saccade_time']<=df_s['saccade_time'].quantile(0.
              saccade_mean.append(np.mean(df_s['saccade_time']))
              saccade_std.append(np.std(df_s['saccade_time']))
          #     name.append(os.split.path(i[0]))
              name.append(i[:-4])

          dict = {"name":name,'blink_mean': blink_mean, 'blink_std': blink_std, 'saccade_mean': saccade_mean,'saccade_std':saccade_std}
          data=pd.DataFrame(dict)
          data.to_csv("X:\\Eye_tracking_Processed_Raw_videos\\event_statistics22.csv",index=False)
```

Figure 6: Script for Feature extraction from event statistics

All the data from above was merged together to form the final dataset. And dataset of heatmap contained only the images of heat map of point of gaze. All the features is shown in the table 3 and table 2.

No	Features
1	Age
2	Sex
3	before
4	after
5	difference
6	memory
7	eeq
8	ADR
9	ADL
10	blink_mean
11	blink_std
12	saccade_mean
13	saccade_std
14	saccade_per
15	blink_per

Table 2: Features from eye tracking and during questionnaire

4.3 Exploratory Data Analysis

The Dataset contains 47 rows and 41 features and one target variable. The distribution of target variable is shown in the figure 7.

No	Features
1	std_l0
2	std_l10
3	std_l25
4	std_l50
5	std_l75
6	std_l90
7	std_l100
8	std_r0
9	std_r10
10	std_r25
11	std_r50
12	std_r75
13	std_r90
14	std_r100
15	min_l10
16	min_l25
17	min_l50
18	min_l75
19	min_l90
20	min_l100
21	min_r10
22	min_r25
23	min_r50
24	min_r75
25	min_r90
26	min_r100

Table 3: Features from pupil dilation

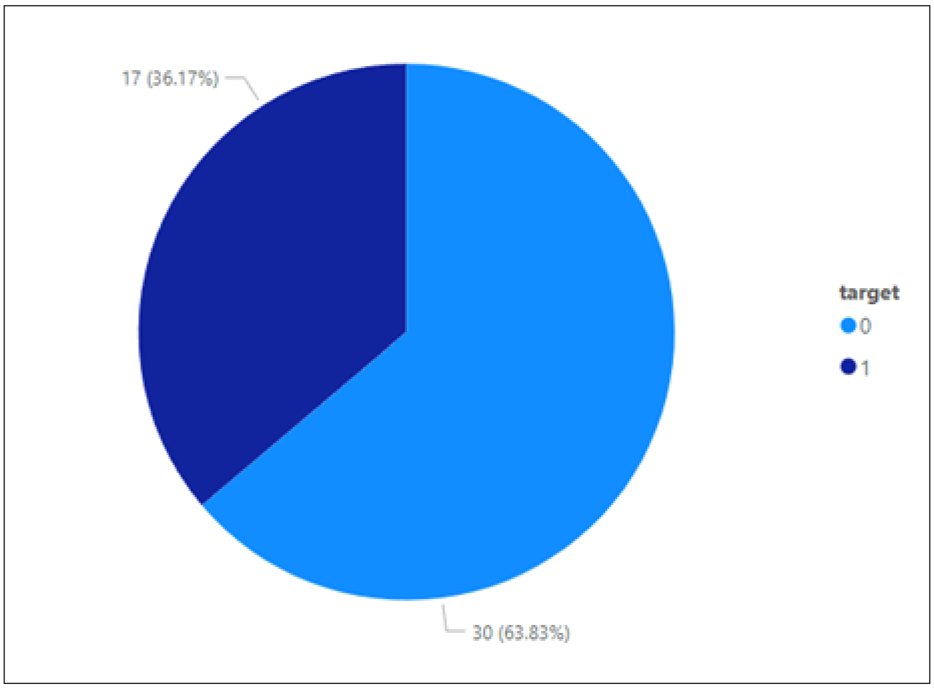


Figure 7: Distribution of target variable empathetic and non empathetic.

The distribution of gender in the data is unequal as shown in figure 8.

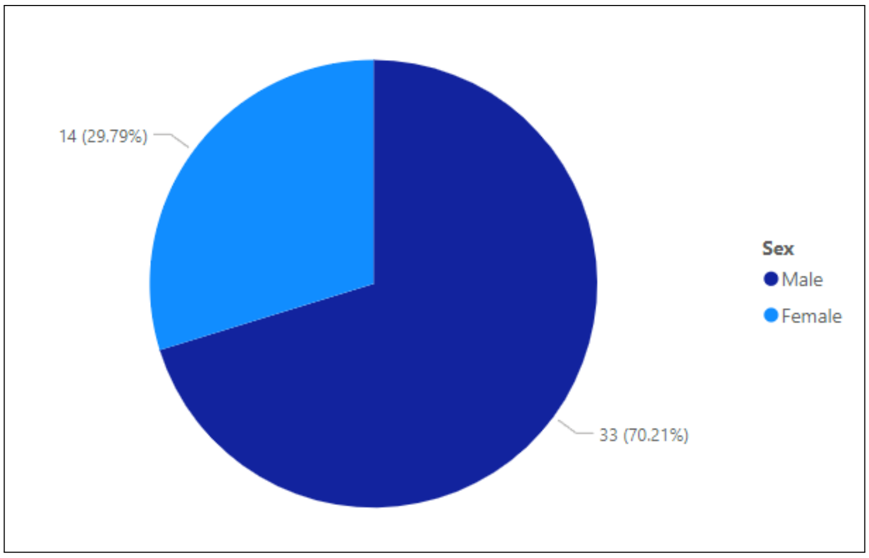


Figure 8: Count of gender in the dataset

Females are generally more empathetic than male and can be seen in figure 9.

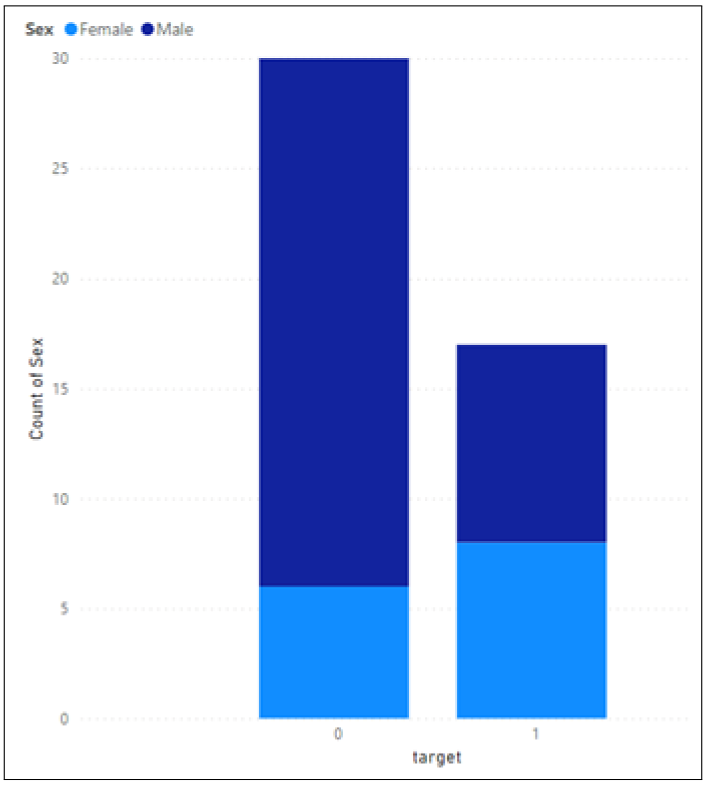


Figure 9: Count of gender for each of the categories (empathetic and not empathetic)

The distribution of age of participants are as shown in figure 10

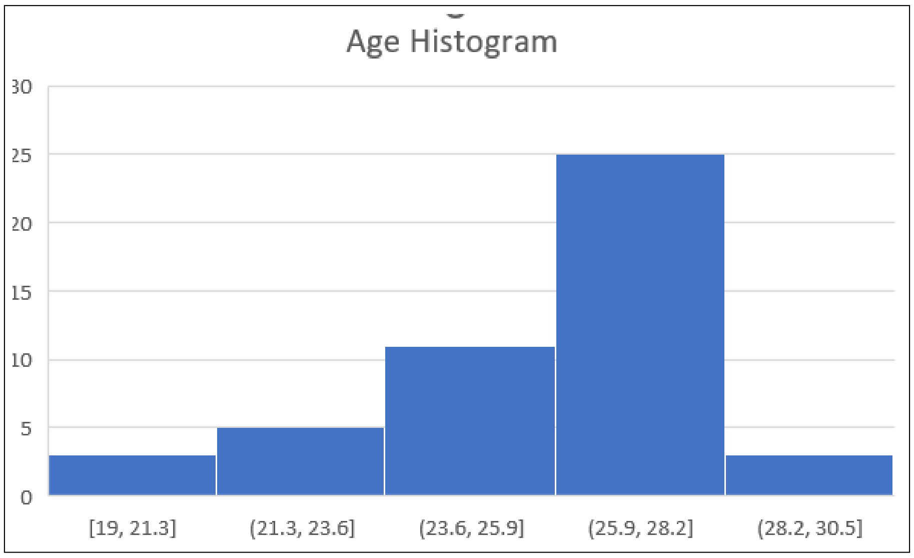


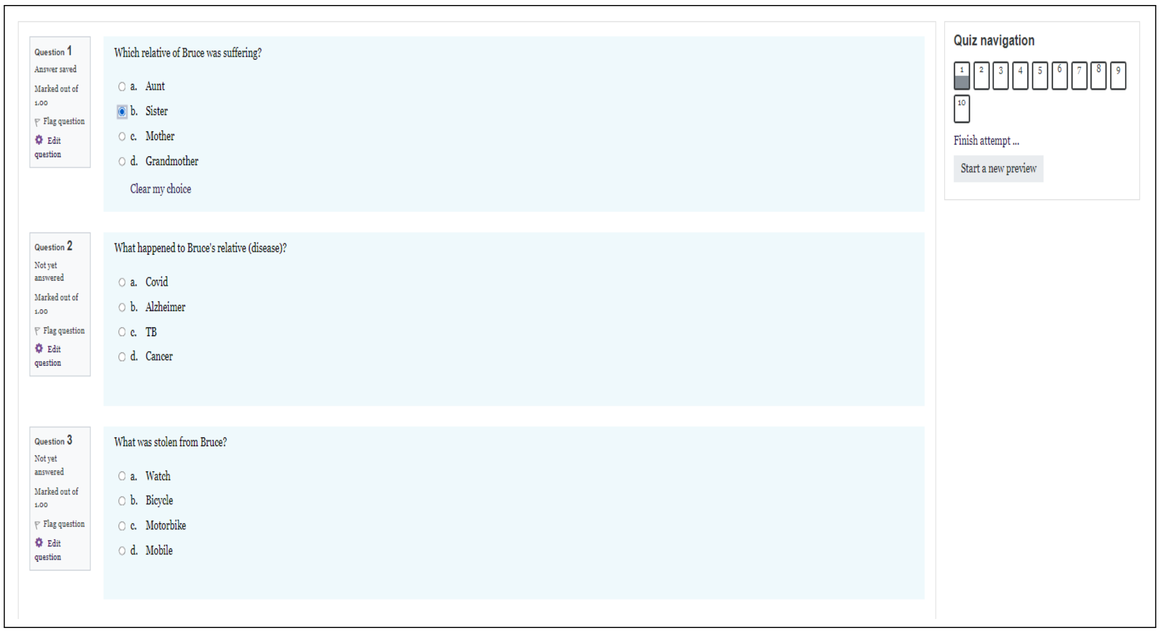
Figure 10: Distribution of age of participants

5 Implementation

The models implemented on the structured data and image data has been discussed in this section. A total of four experiments were conducted in this research. First experiment was model training on self reported features, the second experiment was ML model on eye tracking features. The third experiment was model on pupil dilation based features. The fourth experiment was modelling on all the features. In experiment 1,2 and 3 only three models were applied - random forest, logistic regression, and gradient boosting. In fourth experiment along with the three models extreme gradient boosting was also applied and the code snippet are attached in the following subsections. The code snippets contains hyperparameters space and applying of tuned model on test data for all the models. In experiment three which is modelling on pupil based features, PCA was not applied on the features and were taken as raw features in the experiment three. But in experiment four the pupil features were reduced to four using PCA which is discussed in the following subsection.

5.1 Questionnaires

Two questionnaires were asked while conducting the experiments. The first questionnaire was memory based questionnaire. The memory based questionnaire is accessed using the moodle, as it is saved as moodle quiz. The moodle quiz are opened in the laptop in which the participant watches the video. The score are shown immediately on the completion of the test which has to be noted down. The interface of the quiz/memory questionnaire is as shown in the figure 11.


Question 1
Answer saved
Marked out of 1.00
Flag question
Edit question

Which relative of Bruce was suffering?

- a. Aunt
- [✓ b. Sister]
- c. Mother
- d. Grandmother

Clear my choice

Question 2
Not yet answered
Marked out of 1.00
Flag question
Edit question

What happened to Bruce's relative (disease)?

- a. Covid
- b. Alzheimer
- c. TB
- d. Cancer

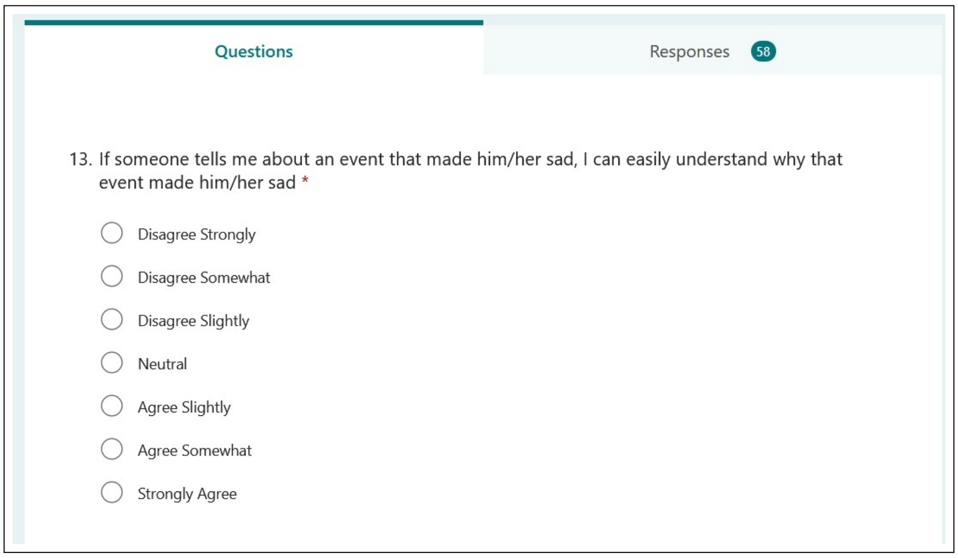
Question 3
Not yet answered
Marked out of 1.00
Flag question
Edit question

What was stolen from Bruce?

- a. Watch
- b. Bicycle
- c. Motorbike
- d. Mobile

Figure 11: Interface of memory questionnaire

The sad emotion based empathy questionnaire is created in Microsoft survey. The link of the same is used to open the questionnaire at the end of the experiment as the last step. There are a total of thirteen question where the first three questions are participant name, gender and age. The next ten questions are empathy questionnaire. Each question is multiple choice question and has seven options in it and marks are given based on the option chosen. For "Strongly disagree" -3 is given for neutral "0" and for "Strongly agree" 3 is given and the other question's mark are given linearly. The output of the questionnaires are send to a google excel sheet. The out of the sheet contains options chosen for each question and a total average score which lies between -3 to 3. A sample questions and survey's interface is shown in the figure 12. The memory questionnaire's result and empathy questionnaire's result along with gender and age is merged in the final dataset with other features.



13. If someone tells me about an event that made him/her sad, I can easily understand why that event made him/her sad *

- Disagree Strongly
- Disagree Somewhat
- Disagree Slightly
- Neutral
- Agree Slightly
- Agree Somewhat
- Strongly Agree

Figure 12: Interface of Empathy questionnaire

5.2 PCA on Pupil Data and Heat map images

PCA was applied to the 26 pupil features and the result is shown in figure 12. The figure 13 shows that 99% of variance was shown in only first four features and hence the dimension of data was reduces and 26 features were reduced to 4 features using the PCA.

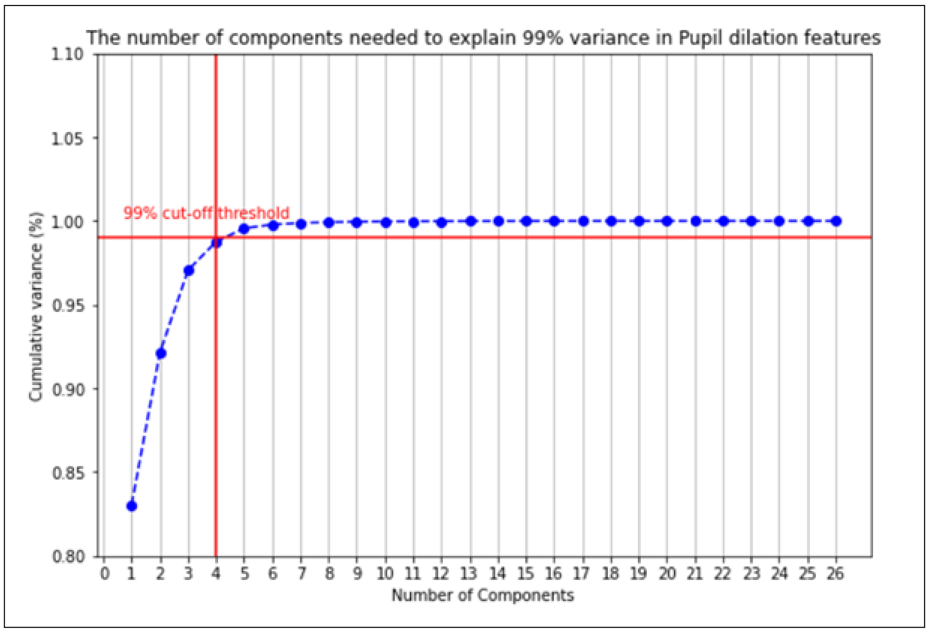


Figure 13: PCA on pupil data shows that 99% of variance in combination of 4 features.

PCA was applied to the 12288 heat map flatten image's features and the result is shown in figure 14. The figure shows that 95% of variance was shown in only first thirty four features and hence the dimension of data was reduces and 12288 features were reduced to only 34 features using the PCA.

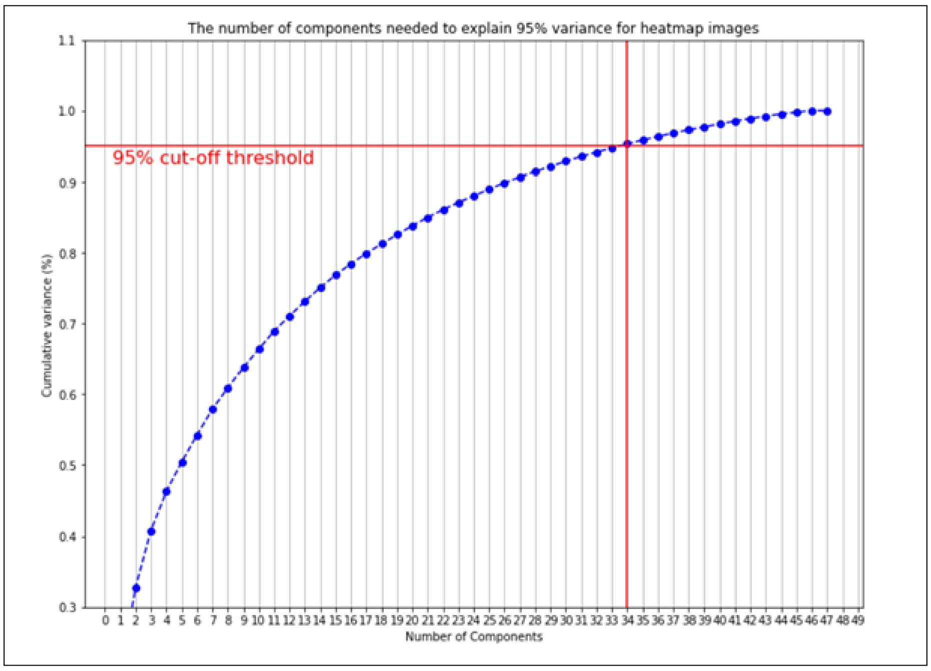
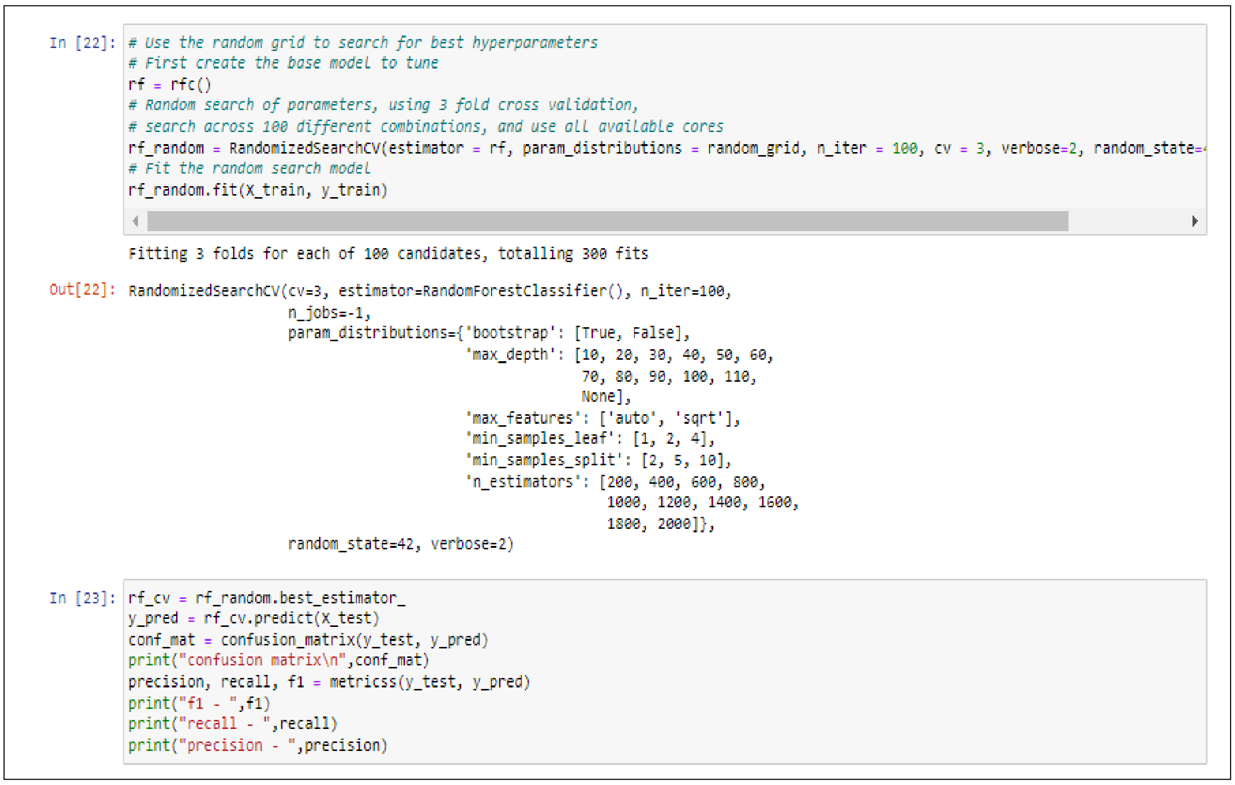


Figure 14: PCA on heat map image's flatten array data shows that 95% of variance was found in combination of 34 features

5.3 Random Forest model

Random Forest algorithm was implemented across all the experiment. The implementation of Random Forest for experiments in research is shown in figure 15. The code includes the search space of hyperparameters for the model. The hyperparameters are tuned. Using the tuned model, the prediction is done on test data.

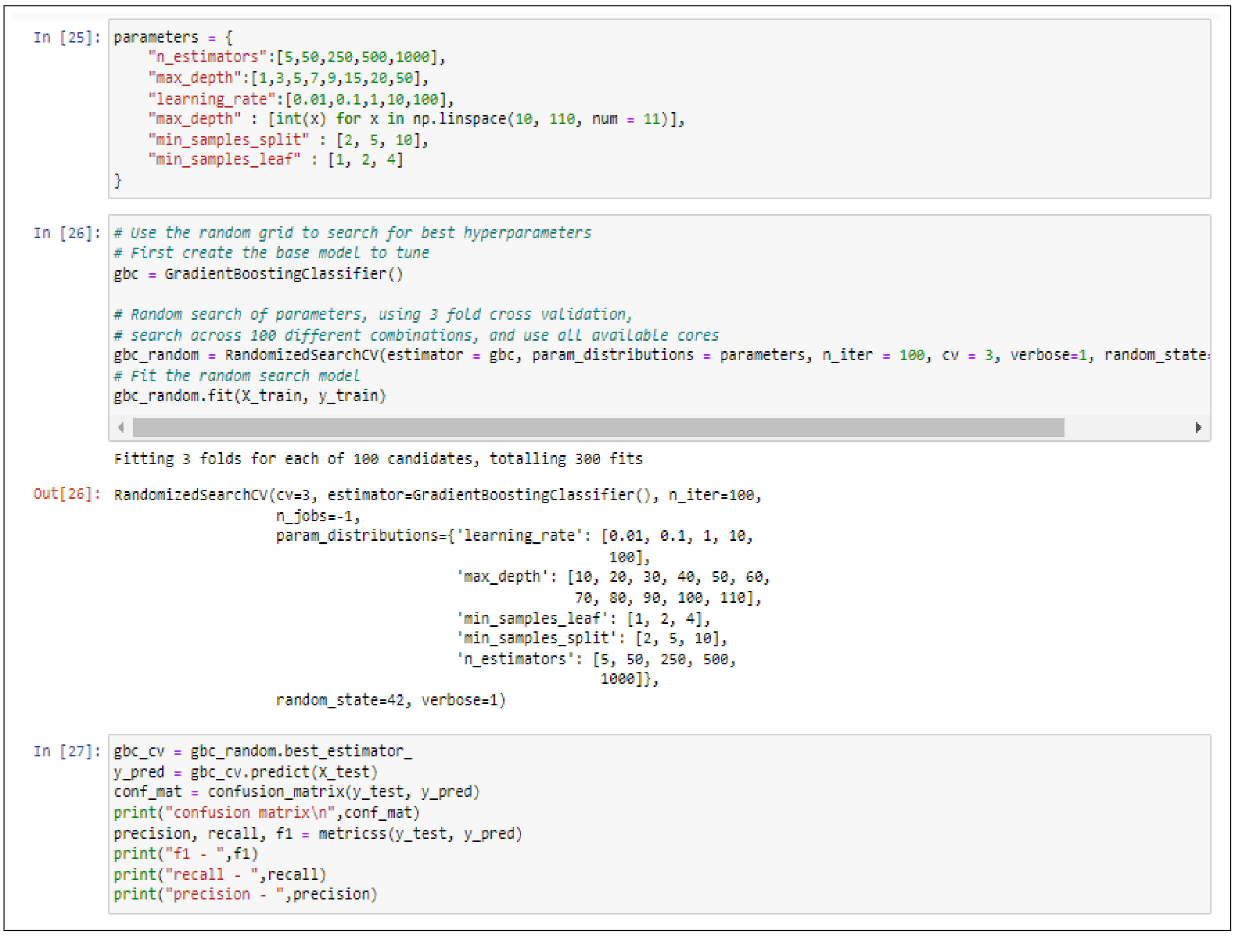


```
In [22]: # Use the random grid to search for best hyperparameters
         # First create the base model to tune
         rf = rfc()
         # Random search of parameters, using 3 fold cross validation,
         # search across 100 different combinations, and use all available cores
         rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_state=
         # Fit the random search model
         rf_random.fit(X_train, y_train)

         Fitting 3 folds for each of 100 candidates, totalling 300 fits

Out[22]: RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                            n_jobs=-1,
                            param_distributions={'bootstrap': [True, False],
                                                 'max_depth': [10, 20, 30, 40, 50, 60,
                                                               70, 80, 90, 100, 110,
                                                               None],
                                                 'max_features': ['auto', 'sqrt'],
                                                 'min_samples_leaf': [1, 2, 4],
                                                 'min_samples_split': [2, 5, 10],
                                                 'n_estimators': [200, 400, 600, 800,
                                                                  1000, 1200, 1400, 1600,
                                                                  1800, 2000]},
                            random_state=42, verbose=2)

In [23]: rf_cv = rf_random.best_estimator_
         y_pred = rf_cv.predict(X_test)
         conf_mat = confusion_matrix(y_test, y_pred)
         print("confusion matrix\n",conf_mat)
         precision, recall, f1 = metricss(y_test, y_pred)
         print("f1 - ",f1)
         print("recall - ",recall)
         print("precision - ",precision)
```

Figure 15: Random Forest implementation with hyper parameters tuned.

5.4 Gradient Boosting model

Gradient boosting 16 was applied. The code includes the search space of hyperparameters for the model. The hyperparameters are tuned. Using the tuned model, the prediction is done on test data. The gradient boosting did not perform well in all the experiments.

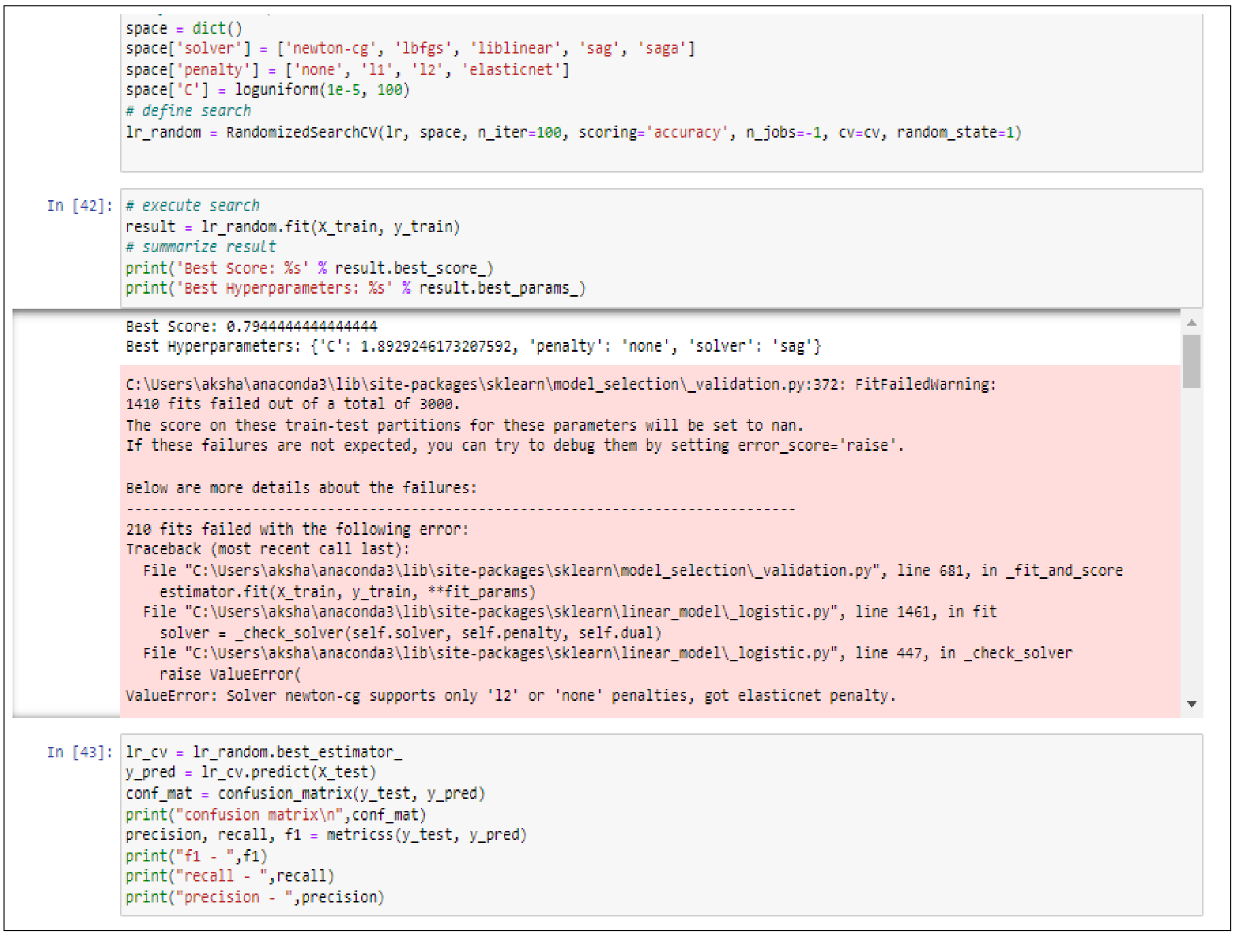


```
In [25]: parameters = {
             "n_estimators":[5,50,250,500,1000],
             "max_depth":[1,3,5,7,9,15,20,50],
             "learning_rate":[0.01,0.1,1,10,100],
             "max_depth" : [int(x) for x in np.linspace(10, 110, num = 11)],
             "min_samples_split" : [2, 5, 10],
             "min_samples_leaf" : [1, 2, 4]
         }

In [26]: # Use the random grid to search for best hyperparameters
         # First create the base model to tune
         gbc = GradientBoostingClassifier()

         # Random search of parameters, using 3 fold cross validation,
         # search across 100 different combinations, and use all available cores
         gbc_random = RandomizedSearchCV(estimator = gbc, param_distributions = parameters, n_iter = 100, cv = 3, verbose=1, random_state=
         # Fit the random search model
         gbc_random.fit(X_train, y_train)

         Fitting 3 folds for each of 100 candidates, totalling 300 fits

Out[26]: RandomizedSearchCV(cv=3, estimator=GradientBoostingClassifier(), n_iter=100,
                            n_jobs=-1,
                            param_distributions={'learning_rate': [0.01, 0.1, 1, 10,
                                                                   100],
                                                 'max_depth': [10, 20, 30, 40, 50, 60,
                                                               70, 80, 90, 100, 110],
                                                 'min_samples_leaf': [1, 2, 4],
                                                 'min_samples_split': [2, 5, 10],
                                                 'n_estimators': [5, 50, 250, 500,
                                                                  1000]},
                            random_state=42, verbose=1)

In [27]: gbc_cv = gbc_random.best_estimator_
         y_pred = gbc_cv.predict(X_test)
         conf_mat = confusion_matrix(y_test, y_pred)
         print("confusion matrix\n",conf_mat)
         precision, recall, f1 = metricss(y_test, y_pred)
         print("f1 - ",f1)
         print("recall - ",recall)
         print("precision - ",precision)
```

Figure 16: Gradient Boosting implementation with hyper parameters tuned.

5.5 Logistic Regression model

Logistic regression model was applied as shown in figure 17. The code includes the search space of hyperparameters for the model. The hyperparameters are tuned. Using the tuned model, the prediction is done on test data. Logistic regression performed the better in all the experiments as compared to the other models.



```
space = dict()
space['solver'] = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
space['penalty'] = ['none', 'l1', 'l2', 'elasticnet']
space['C'] = loguniform(1e-5, 100)
# define search
lr_random = RandomizedSearchCV(lr, space, n_iter=100, scoring='accuracy', n_jobs=-1, cv=cv, random_state=1)
```

```
In [42]: # execute search
         result = lr_random.fit(X_train, y_train)
         # summarize result
         print('Best Score: %s' % result.best_score_)
         print('Best Hyperparameters: %s' % result.best_params_)
```

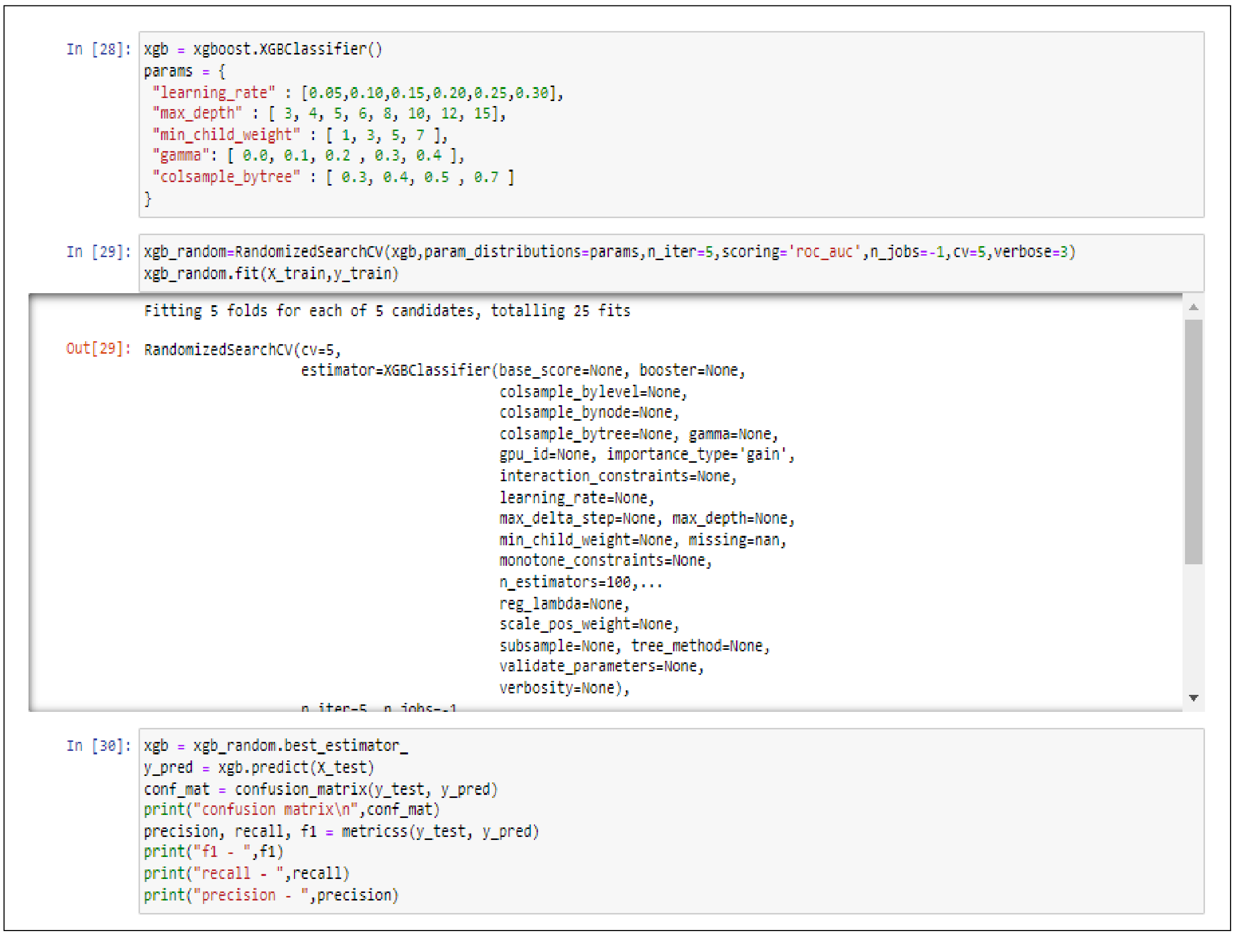
```
Best Score: 0.7944444444444444
Best Hyperparameters: {'C': 1.8929246173207592, 'penalty': 'none', 'solver': 'sag'}

C:\Users\aksha\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:372: FitFailedWarning:
1410 fits failed out of a total of 3000.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
210 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\aksha\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 681, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\aksha\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\aksha\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 447, in _check_solver
    raise ValueError(
ValueError: Solver newton-cg supports only 'l2' or 'none' penalties, got elasticnet penalty.
```

```
In [43]: lr_cv = lr_random.best_estimator_
         y_pred = lr_cv.predict(X_test)
         conf_mat = confusion_matrix(y_test, y_pred)
         print("confusion matrix\n",conf_mat)
         precision, recall, f1 = metricss(y_test, y_pred)
         print("f1 - ",f1)
         print("recall - ",recall)
         print("precision - ",precision)
```

Figure 17: Logistic regression performed the best with 88.89% overall accuracy.

5.5.1 Extreme Gradient Boosting model

XGB Model was applied and figure 18 shows the code snipped of the same. The code includes the search space of hyperparameters for the model. The hyperparameters are tuned. Using the tuned model, the prediction is done on test data. The XGB model did not perform well.



```
In [28]: xgb = xgboost.XGBClassifier()
         params = {
          "learning_rate" : [0.05,0.10,0.15,0.20,0.25,0.30],
          "max_depth" : [ 3, 4, 5, 6, 8, 10, 12, 15],
          "min_child_weight" : [ 1, 3, 5, 7 ],
          "gamma": [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
          "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]
         }

In [29]: xgb_random=RandomizedSearchCV(xgb,param_distributions=params,n_iter=5,scoring='roc_auc',n_jobs=-1,cv=5,verbose=3)
         xgb_random.fit(X_train,y_train)

         Fitting 5 folds for each of 5 candidates, totalling 25 fits

Out[29]: RandomizedSearchCV(cv=5,
                            estimator=XGBClassifier(base_score=None, booster=None,
                                                    colsample_bylevel=None,
                                                    colsample_bynode=None,
                                                    colsample_bytree=None, gamma=None,
                                                    gpu_id=None, importance_type='gain',
                                                    interaction_constraints=None,
                                                    learning_rate=None,
                                                    max_delta_step=None, max_depth=None,
                                                    min_child_weight=None, missing=nan,
                                                    monotone_constraints=None,
                                                    n_estimators=100,...
                                                    reg_lambda=None,
                                                    scale_pos_weight=None,
                                                    subsample=None, tree_method=None,
                                                    validate_parameters=None,
                                                    verbosity=None),
                            n_iter=5, n_jobs=-1

In [30]: xgb = xgb_random.best_estimator_
         y_pred = xgb.predict(X_test)
         conf_mat = confusion_matrix(y_test, y_pred)
         print("confusion matrix\n",conf_mat)
         precision, recall, f1 = metricss(y_test, y_pred)
         print("f1 - ",f1)
         print("recall - ",recall)
         print("precision - ",precision)
```

Figure 18: XGB Implementation

6 Conclusion

The steps are clear and discussed in a way which are easy to replicate the complete project. The conclusion and results of the model are discussed in the thesis report.