Configuration Manual for Machine Learning Framework For Predicting Empathy Using Eye Tracking and Facial Expressions

MSc Research Project
Data Analytics

Samarth Krishna Dhawan
Student ID: 20180489

School of Computing
National College of Ireland

Supervisor:	Dr. Anu Sahni
Supervisor:	Dr. Paul Stynes

Student Name:	Samarth Krishna Dhawan
Student ID:	20180489
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Anu Sahni
Supervisor:	Dr. Paul Stynes
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual for Machine Learning Framework For Predicting Empathy Using Eye Tracking and Facial Expressions
Word Count:	1711
Page Count:	8

Signature:	Samarth Krishna Dhawan
Date:	19th September 2022

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

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

Configuration Manual for Machine Learning Framework For Predicting Empathy Using Eye Tracking and Facial Expressions

Samarth Krishna Dhawan
20180489

1 Introduction

This document outlines the step by step process that enabled the implementation of the research project - 'Machine Learning Framework For Predicting Empathy Using Eye Tracking and Facial Expressions'. The research aims to determine to what extent can eye tracking and facial expressions combine in empathy prediction?. For the implementation of this research a YOLOv5 along with machine learning algorithms were used. Below is the structure of this configuration manual describing the stages of implementation of the project:

- Section 2 Hardware and Software requirements - This section will cover the system configuration, tools and technologies used.
- Section 3 Data Collection - This section will describe the data collection process for this research
- Section 4 Data Pre-Processing - This section will describe data extraction, data storage, EDA, data transformation and models
- Section 5 Implementation - This section will describe the implementation of the project
- Section 6 Conclusion - In this section, we will discuss the conclusion of the configuration manual

2 Hardware and Software Requirements

- Eye Tracking - The SMI-A eye tracking glasses provided by National College of Ireland were utilized for eye tracking. The gadget is intended to track eye gazing in real time and is made of sturdy hardware that has been tested by millions of users. With a sampling rate of 60hz, it can catch eye gaze. With the unique SMI Semantic Gaze Mapping (SGM) technology, the SMI BeGaze analysis software enables for exceptionally efficient aggregation of eye tracking data across several participants.
- Facial Expressions - For tracking facial expressions the webcam embedded in the laptop was used which captures videos in 720 Pixel in 16:9 aspect ratio camera with 30 frames per second and 60Hz refresh rate.

- Data Storage - For data storage, Microsoft one drive was used to store all data in a central repository accessbile across devices and platforms
- Machine Learning - For the purpose of building YOLOV5 models, google colab with GPU was used as it was not possible to train such a heavy model on 50 videos on the local machine. Resnet50 was built on the local machine using python and jupyter notebooks.

3 Data Collection

Eye tracking experiment was conducted on 50 participants while they watched a video stimuli and their facial expressions were also recorded using the webcam. The participants had 14 females and 36 males which ranged from 19 to 30 years of age. Before the experiment began, the participants were asked to self report their level of sadness on a scale of 1 to 10. Post the videos, the participants were requested to fill out a memory questionnaire related to the video watched earlier which had 10 questions followed by an empathy questionnaire which consisted of 10 questions on a 7 point likert scale. The last part of the experiment was to self report sadness levels on a scale of 1 to 10 after watching the video. The entire experiment last for about 25-30 minutes depending on the time taken to calibrate the eye tracking and filling out the questionnaire. The list of features and how they were extracted is given below:

- Age - Age of the participant (Self Reported)
- Sex - Gender of the participant (Self Reported)
- Sadness Level Before Experiment - Sadness level before watching the stimuli video (Self Reported)
- Sadness Level After Experiment - Sadness level after watching the stimuli video (Self Reported)
- Difference in Sadness - Differences in self reported sadness level before and after watching the video (Calculated Field)
- Memory test - Test consisting of 10 questions about the video watched by the participant (MCQ questions, score ranging from 1 to 10)
- Average Distance from Right Eye - Average distance from right eye of the narrator in the story. To calculate this, first the corner coordinates of the eyes were identified using the dlib library. Post which the centre points were calculated for both eyes and then average distance of point of gaze from the centre of the eye was calculated (Calculate field) Average Distance from Left Eye - Same calculation as for the right eye
- blink mean - Average duration of blink of the participant (Calculated field from BeGaze Software)
- Blink Std - Standard deviation of blink duration (Calculated field from BeGaze Software)

- Blink Percentage - Blinks as a percentage of the total eye tracking data points captured (Calculated field from BeGaze Software)
- Saccade mean - Average Saccade duration (Calculated field from BeGaze Software). Saccade can be defined as rapid eye movement between two points which gives an indication of distraction of the participant
- Saccade Std - Standard deviation of the saccade duration (Calculated field from BeGaze Software)
- Saccade Percentage - Saccades as a percentage of total eye tracking data points captured (Calculated field from BeGaze Software)
- Heat Maps - Generated heatmaps which depict the area where the participant concentrated the most during the experiment. To generate heatmaps, first a YOLOv5 model was trained to identify the circle depicting the point of gaze along with the laptop screen. Then using numpy arrays and seaborn libraries the heatmpas were generated. Average Emotion - Raw facial expressions were fed into the DeepFace library and the most dominant emotion across the length of watching the video stimuli was extracted Average Emotion Percentage - Percentage count of the most dominant emotion amongst all other emotions depicted

4 Data Pre-Processing

This section will discuss, data pre-processing, Data Extraction and Exploratory Data Analysis

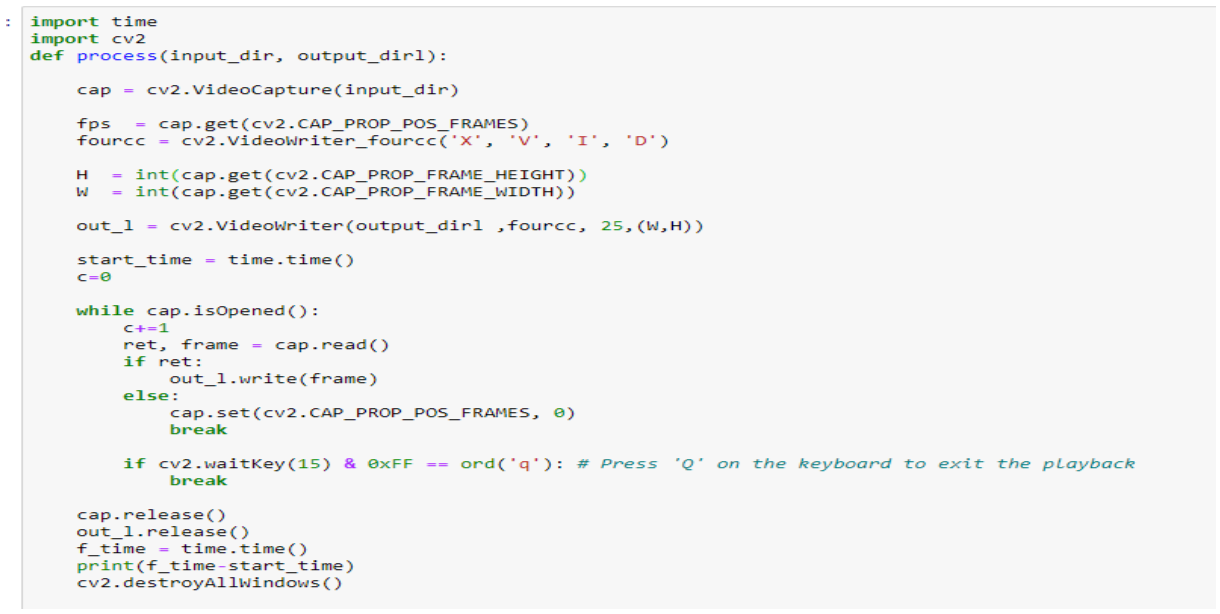
4.1 Data Preparation

The video extracted from the BeGaze software was in 1000 frame per seconds to convert it into 25 frame per seconds the script shown in figure 3 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.

The facial Expressions videos were manually trimmed to match the start and end time of the video stimuli used in the experiment

4.2 Data 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 tomodel$_{w}eights/best_{c}ircle_{w}eight.pt - -img640 - -conf0.25$–$source$video

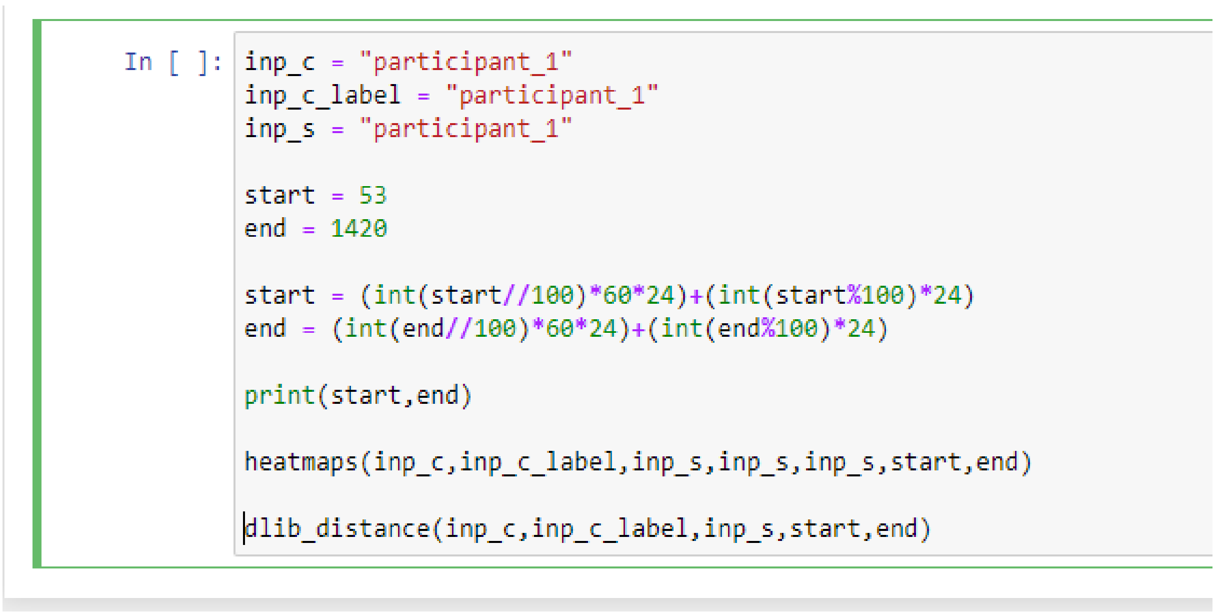


```
: import time
  import cv2
  def process(input_dir, output_dirl):

      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 1: Script to convert 1000fps to 50fps

participant$_1$$.avi - -save - txt - -nosave - -save - conf - -nameparticipant_1 - -projectcircle_or_screen/$

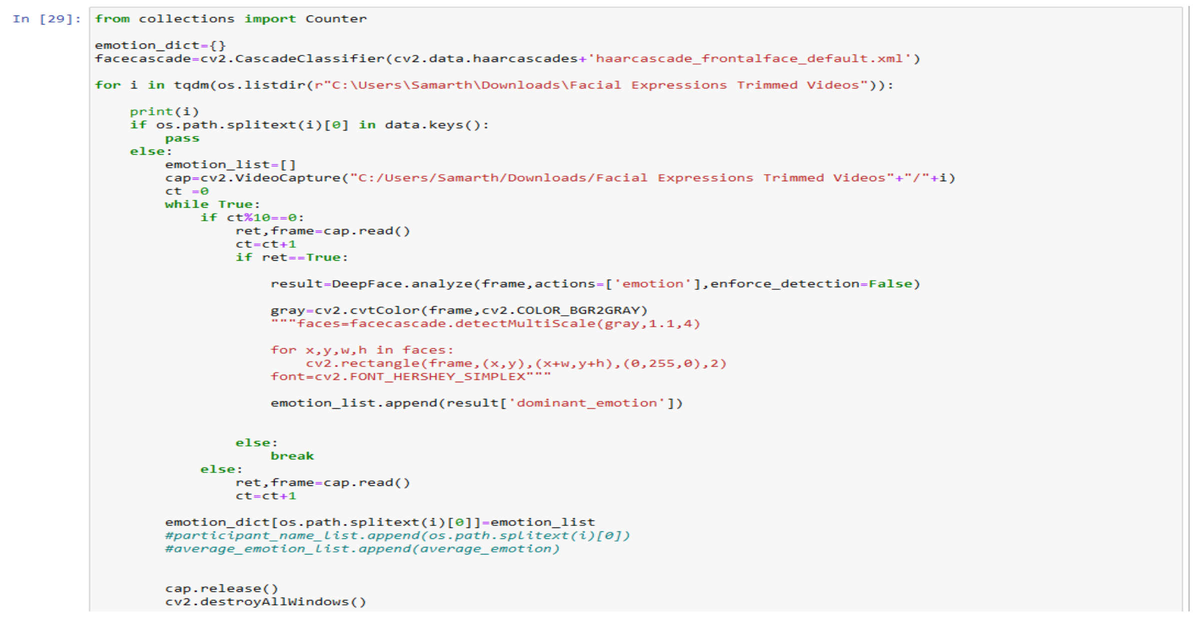
To generate average distance from eye dlib library was used to get the face landmarks of the actors present in the screen of field of gaze. The script used for the same is in jupyter file "get$_face_landmarks_dlib.ipynb" which used weights stored in \shape_predictor_68_face_landmarks.dat".The$



```
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 2: Script to Generate HeatMaps

percentage of occurrence of sad emotion was extracted using the DeepFace library. Deepface was used to extract the dominant emotion in each frame of the video the script for it is shown in Fig.3

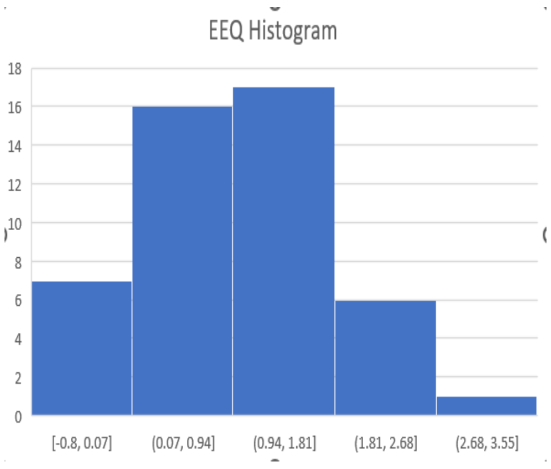


```
In [29]: from collections import Counter

emotion_dict={}
facecascade=cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade_frontalface_default.xml')

for i in tqdm(os.listdir(r"C:\Users\Samarth\Downloads\Facial Expressions Trimmed Videos")):

    print(i)
    if os.path.splitext(i)[0] in data.keys():
        pass
    else:
        emotion_list=[]
        cap=cv2.VideoCapture("C:/Users/Samarth/Downloads/Facial Expressions Trimmed Videos"+"/"+i)
        ct =0
        while True:
            if ct%10==0:
                ret,frame=cap.read()
                ct=ct+1
                if ret==True:

                    result=DeepFace.analyze(frame,actions=['emotion'],enforce_detection=False)

                    gray=cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
                    """faces=facecascade.detectMultiScale(gray,1.1,4)

                    for x,y,w,h in faces:
                        cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
                    font=cv2.FONT_HERSHEY_SIMPLEX"""

                    emotion_list.append(result['dominant_emotion'])


                else:
                    break
            else:
                ret,frame=cap.read()
                ct=ct+1

        emotion_dict[os.path.splitext(i)[0]]=emotion_list
        #participant_name_list.append(os.path.splitext(i)[0])
        #average_emotion_list.append(average_emotion)


        cap.release()
        cv2.destroyAllWindows()
```

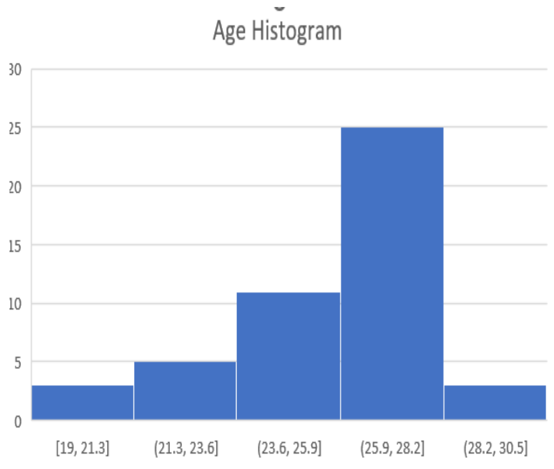
Figure 3: Script to Extract Dominant Facial Expression

4.3 Exploratory Data Analysis

The EDA of the data revealed that participants had 14 females and 36 Males, which ranged from 19 to 30 years old with major population between 25 and 28 years of age. The average sadness level before watching the video was 2.5 while that of after watching the video rose up 4.06. The average memory test score was 8.



(a) Distribution of EEQ Scores



(b) Age Distribution

Figure 4: EDA histograms

5 Implementation

The research was conducted with many iteration of the model building phase. As described in the report the first 3 iterations including building logistic regression models with 3 individual set of features which were Facial, self reported and eye tracking features. Once the best result was achieved by combining all the features, 2 more models were built. All models were built using 3-fold cross validation and hyper-parameters were tuned using randomized-search. Logistic Regression, Random Forest and Gradient Boosting were used as part of this research.

PCA was done to reduce the number of features from the heatmap. it showed that 34 features from heatmap flattened pixels explain 95% of the variance as shown in Fig.

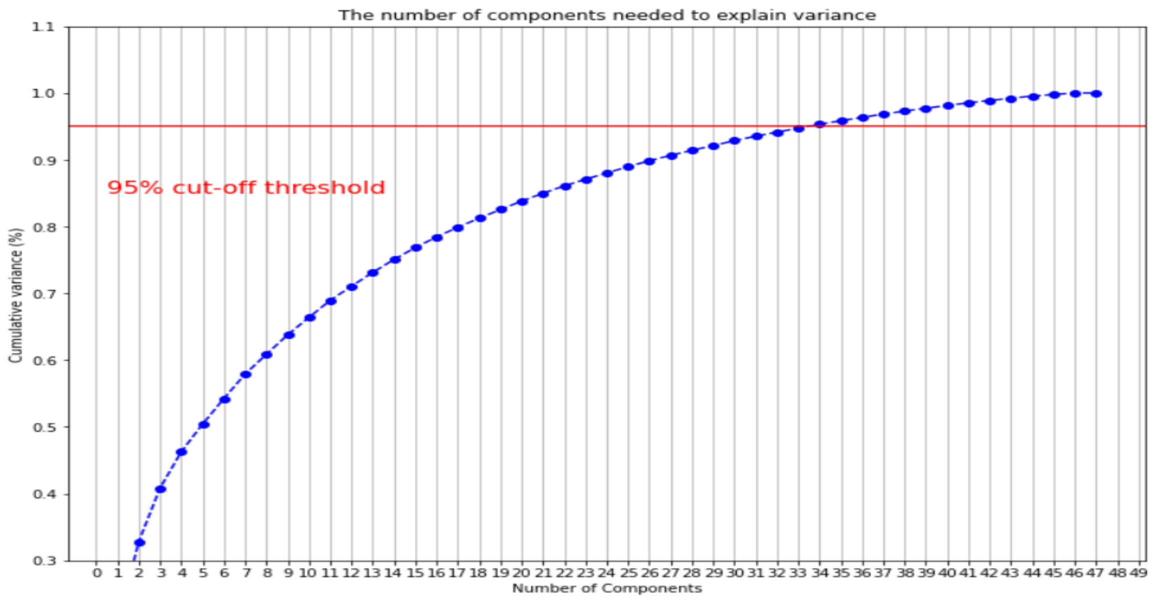


Figure 5: PCA Threshold for HeatMap features

5.1 Model 1 : Random Forest

In this iteration, random forest was built with 3 fold cross validation with n-estimators, max-depth, min-samples-leaf, min-samples-split being the hyperparameters that were tuned using random search

5.2 Model 2 : Gradient Boosting

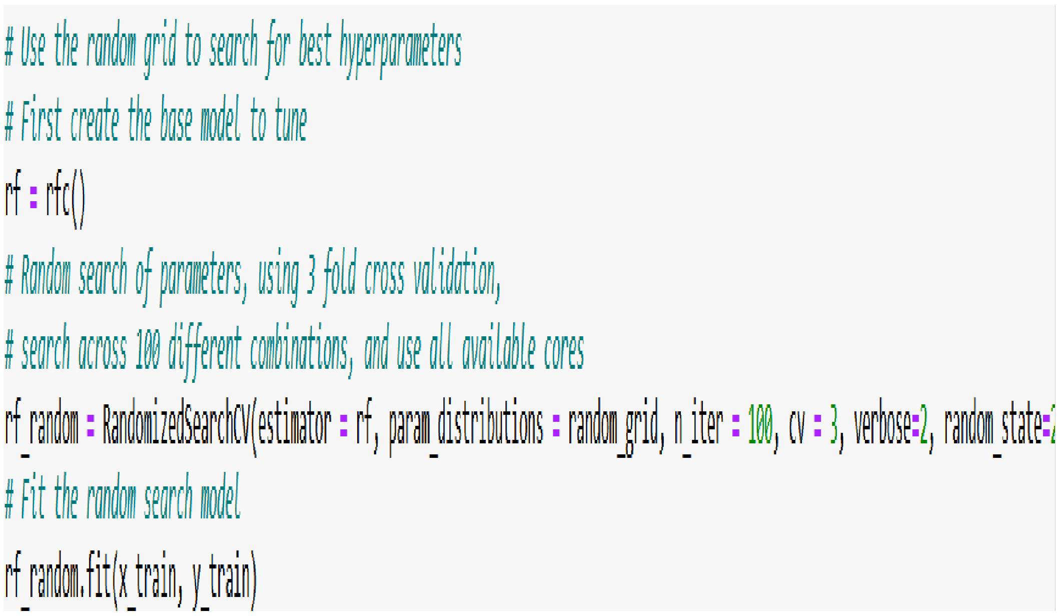
In this iteration, Gradient Boosting was built with 3 fold cross validation with n-estimators, max-depth, learning-rate, min-samples-leaf and min-samples-split being the hyperparameters that were tuned using random search

5.3 Model 3 : Logistic Regression

In this iteration, Logistic Regression was built with 3 fold cross validation with solver, penalty and C being the hyperparameters that were tuned using random search

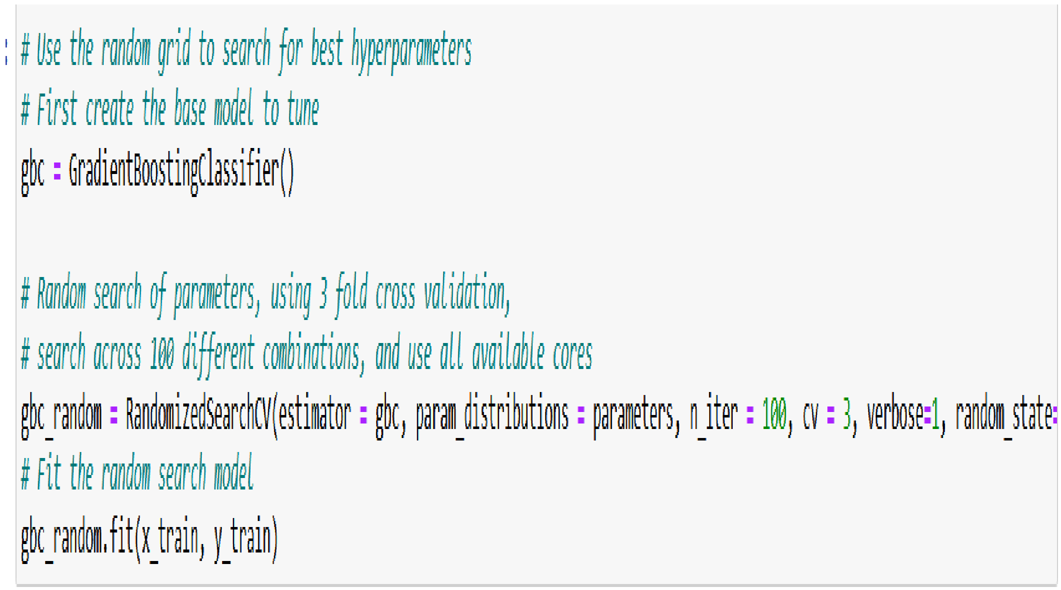
6 Conclusion

In conclusion, the information mentioned in this report, explains the complete step by step implementation of the research. The report is divided into sections and has been explained in detail and sequential form.



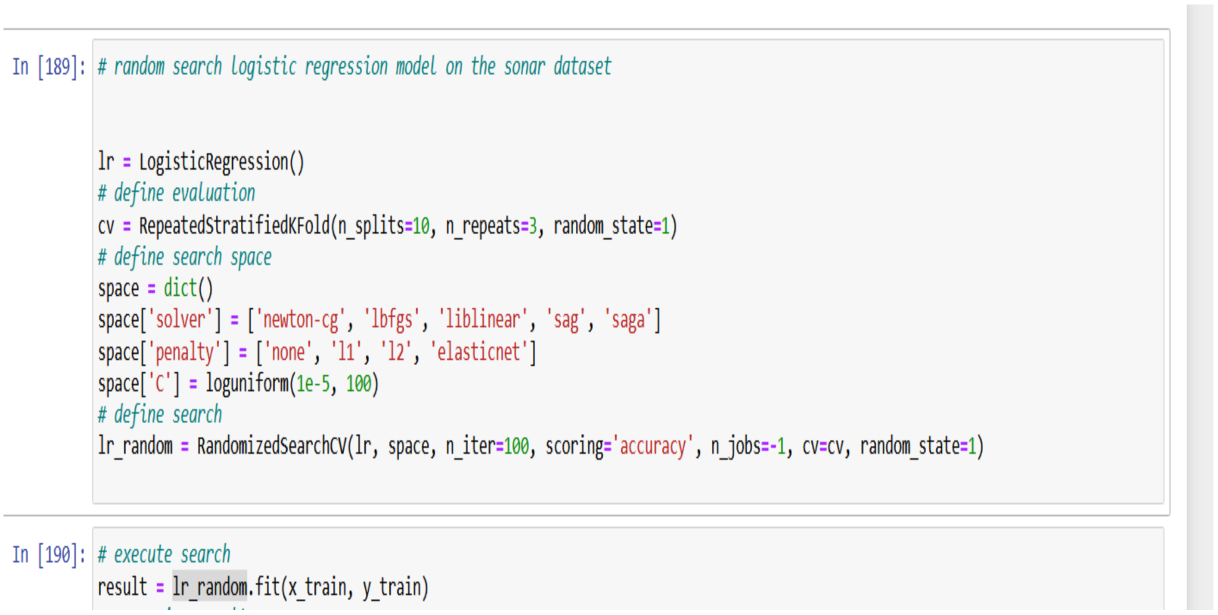
```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = rfc()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions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

Figure 6: Random Forest Implementation



```
# 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)
```

Figure 7: Gradient Boosting Implementation



```
In [189]: # random search logistic regression model on the sonar dataset


          lr = LogisticRegression()
          # define evaluation
          cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
          # define search space
          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 [190]: # execute search
          result = lr_random.fit(x_train, y_train)
```

Figure 8: Logistic Regression Implementation