INDUSTRIAL PROJECT REPORT

HUMAN ACTIVITY RECOGNITION USING MACHINE LEARNING

INDIAN INSTITUTE OF TECHNOLOGY, MANDI

Submitted in partial fulfillment of the requirements of the award of

Degree of Bachelors of Technology in Electronics and Communication Engineering

Submitted By:

Name: Sakshi Sharma	**Roll number: 18BT010443**
Name: Shobhit Sharma	**Roll number: 18BT010449**
Name: Tushar Parmar	**Roll number: 18BT010457**

Training period: 25 March 2022 to 25 June 2022

Submitted To: Prof. Himanshu Monga

Department of Electronics & Communication Engineering

JAWAHARLAL NEHRU GOVT. ENGINEERING.

COLLEGE SUNDERNAGAR, DISTT. MANDI (H.P.)

INDUSTRIAL PROJECT REPORT

HUMAN ACTIVITY RECOGNITION
USING MACHINE LEARNING
INDIAN INSTITUTE OF TECHNOLOGY, MANDI

Submitted in partial fulfillment of the requirements of the award of

Degree of Bachelors of Technology in Electronics and Communication Engineering

Submitted By:

Name: Sakshi Sharma	**Roll number: 18BT010443**
Name: Shobhit Sharma	**Roll number: 18BT010449**
Name: Tushar Parmar	**Roll number: 18BT010457**

Training period: 25 March 2022 to 25 June 2022

Submitted To: Prof. Himanshu Monga

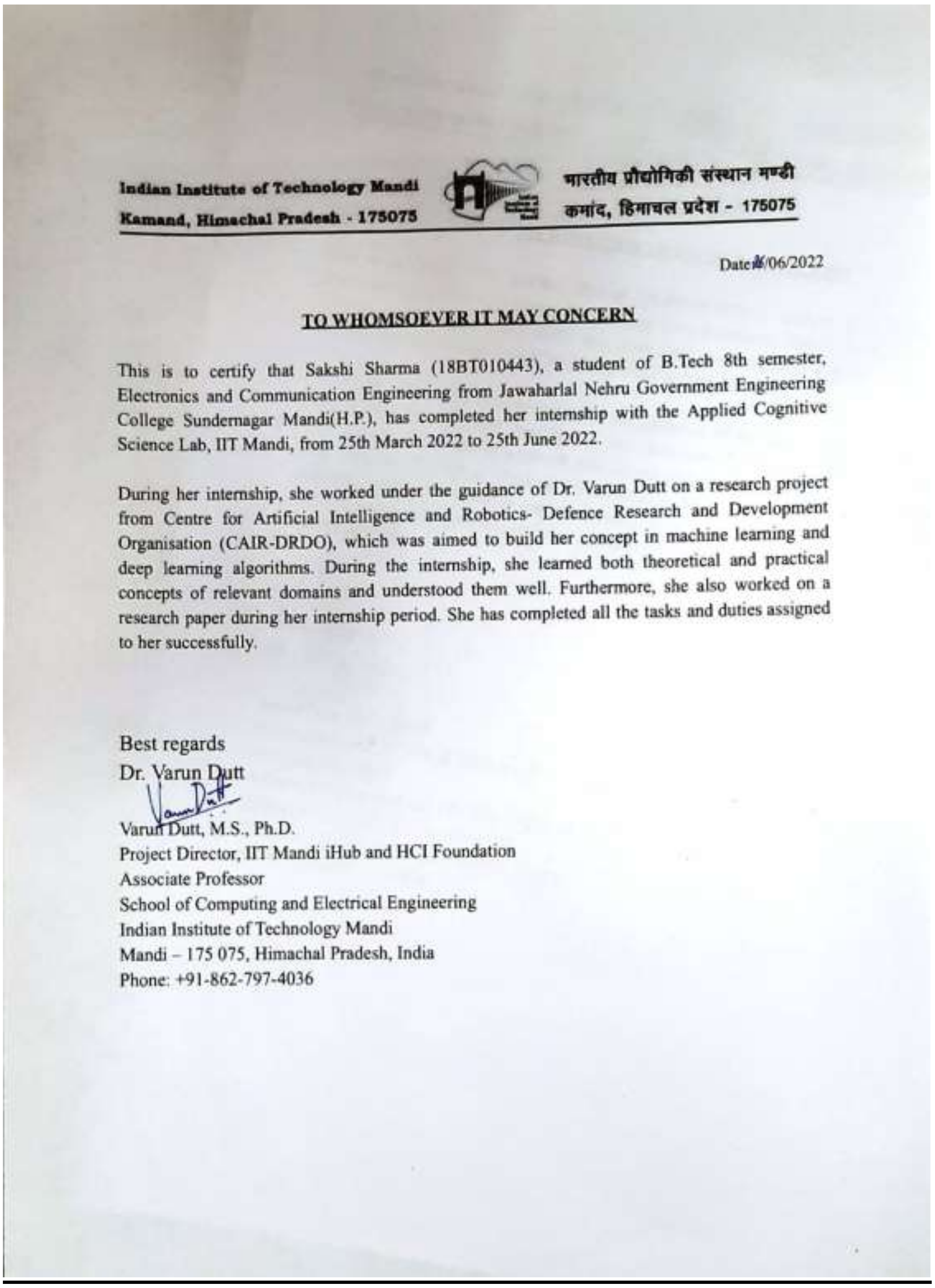
Department of Electronics & Communication Engineering

JAWAHARLAL NEHRU GOVT. ENGINEERING.

COLLEGE SUNDERNAGAR, DISTT. MANDI (H.P.)

TABLE OF CONTENTS

CERTIFICATE BY TRAINING ORGANIZATION



Indian Institute of Technology Mandi
Kamand, Himachal Pradesh - 175075

भारतीय प्रौद्योगिकी संस्थान मण्डी
कमांद, हिमाचल प्रदेश - 175075

Date [illegible]/06/2022

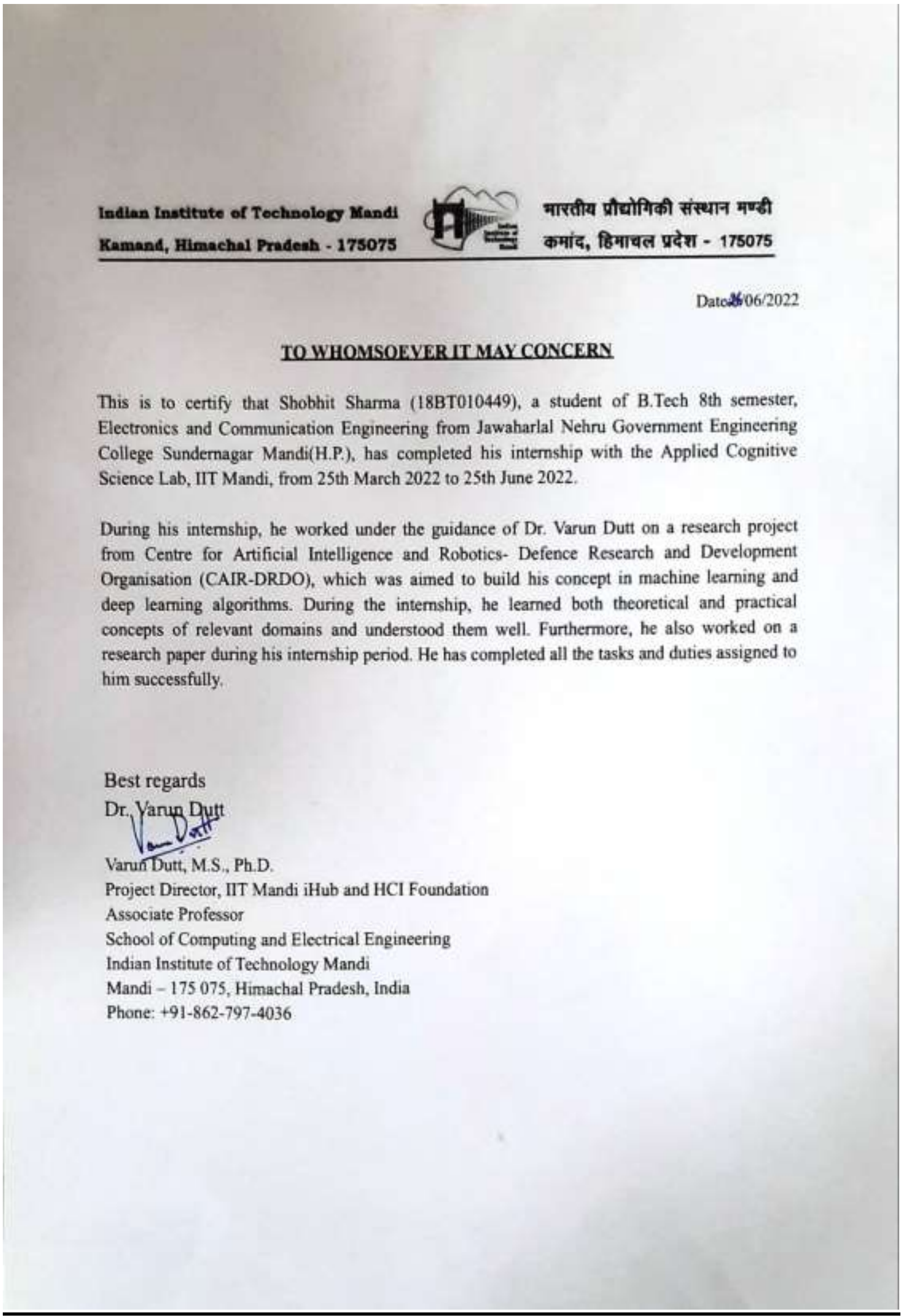
TO WHOMSOEVER IT MAY CONCERN

This is to certify that Sakshi Sharma (18BT010443), a student of B.Tech 8th semester, Electronics and Communication Engineering from Jawaharlal Nehru Government Engineering College Sundernagar Mandi(H.P.), has completed her internship with the Applied Cognitive Science Lab, IIT Mandi, from 25th March 2022 to 25th June 2022.

During her internship, she worked under the guidance of Dr. Varun Dutt on a research project from Centre for Artificial Intelligence and Robotics- Defence Research and Development Organisation (CAIR-DRDO), which was aimed to build her concept in machine learning and deep learning algorithms. During the internship, she learned both theoretical and practical concepts of relevant domains and understood them well. Furthermore, she also worked on a research paper during her internship period. She has completed all the tasks and duties assigned to her successfully.

Best regards
Dr. Varun Dutt

Varun Dutt, M.S., Ph.D.
Project Director, IIT Mandi iHub and HCI Foundation
Associate Professor
School of Computing and Electrical Engineering
Indian Institute of Technology Mandi
Mandi – 175 075, Himachal Pradesh, India
Phone: +91-862-797-4036



Indian Institute of Technology Mandi
Kamand, Himachal Pradesh - 175075

भारतीय प्रौद्योगिकी संस्थान मण्डी
कमांद, हिमाचल प्रदेश - 175075

Date:26/06/2022

TO WHOMSOEVER IT MAY CONCERN

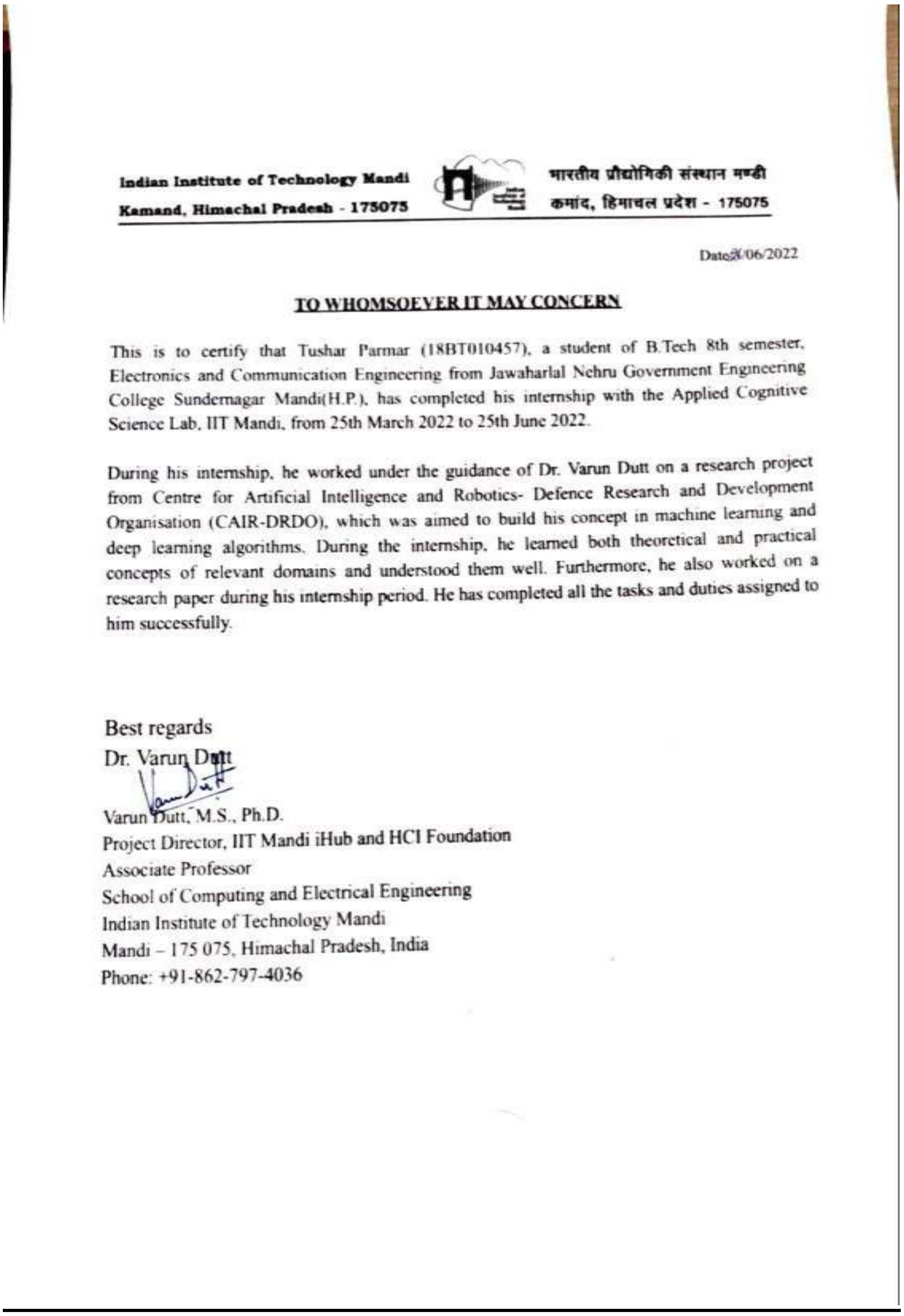
This is to certify that Shobhit Sharma (18BT010449), a student of B.Tech 8th semester, Electronics and Communication Engineering from Jawaharlal Nehru Government Engineering College Sundernagar Mandi(H.P.), has completed his internship with the Applied Cognitive Science Lab, IIT Mandi, from 25th March 2022 to 25th June 2022.

During his internship, he worked under the guidance of Dr. Varun Dutt on a research project from Centre for Artificial Intelligence and Robotics- Defence Research and Development Organisation (CAIR-DRDO), which was aimed to build his concept in machine learning and deep learning algorithms. During the internship, he learned both theoretical and practical concepts of relevant domains and understood them well. Furthermore, he also worked on a research paper during his internship period. He has completed all the tasks and duties assigned to him successfully.

Best regards
Dr. Varun Dutt

[illegible]

Varun Dutt, M.S., Ph.D.
Project Director, IIT Mandi iHub and HCI Foundation
Associate Professor
School of Computing and Electrical Engineering
Indian Institute of Technology Mandi
Mandi – 175 075, Himachal Pradesh, India
Phone: +91-862-797-4036



Indian Institute of Technology Mandi
Kamand, Himachal Pradesh - 175075

भारतीय प्रौद्योगिकी संस्थान मण्डी
कमांद, हिमाचल प्रदेश - 175075

Date[illegible]/06/2022

TO WHOMSOEVER IT MAY CONCERN

This is to certify that Tushar Parmar (18BT010457), a student of B.Tech 8th semester, Electronics and Communication Engineering from Jawaharlal Nehru Government Engineering College Sundernagar Mandi(H.P.), has completed his internship with the Applied Cognitive Science Lab, IIT Mandi, from 25th March 2022 to 25th June 2022.

During his internship, he worked under the guidance of Dr. Varun Dutt on a research project from Centre for Artificial Intelligence and Robotics- Defence Research and Development Organisation (CAIR-DRDO), which was aimed to build his concept in machine learning and deep learning algorithms. During the internship, he learned both theoretical and practical concepts of relevant domains and understood them well. Furthermore, he also worked on a research paper during his internship period. He has completed all the tasks and duties assigned to him successfully.

Best regards
Dr. Varun Dutt
[illegible]
Varun Dutt, M.S., Ph.D.
Project Director, IIT Mandi iHub and HCI Foundation
Associate Professor
School of Computing and Electrical Engineering
Indian Institute of Technology Mandi
Mandi – 175 075, Himachal Pradesh, India
Phone: +91-862-797-4036

DECLARATION BY STUDENT

I hereby declare that the Industrial Training Project Report entitled "Human Activity Recognition using Machine learning" is an authentic record of work carried out by me during my training at the Indian Institute of Technology, Mandi, from 25 - March 2022 to 25 - June 2022 under the supervision of my training supervisor Dr. Varun Dutt, Associate Professor (IIT Mandi) for the award of the degree of B.Tech in Electronics & Communication Engineering.

Signature of Student

Sakshi Sharma (18BT010443)

Shobhit Sharma (18BT010449)

Tushar Parmar (18BT010457)

Date: ___________________________

(Signature of Head of Department)

Examined by:

(Signature of Internal Examiner) (Signature of External Examiner)

ACKNOWLEDGEMENT

The success and final outcome of this project “HUMAN ACTIVITY RECOGNITION USING MACHINE LEARNING” required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

I respect and thank Dr. Varun Dutt, associate professor in the School of Computing and Electrical Engineering, for providing me an opportunity to do the project work in IIT Mandi and giving us all support and guidance which made me complete the project duly. I am extremely thankful to him for providing such a nice support and guidance, although he had busy schedule managing the college affairs. I owe my deep gratitude to our project guide, who took keen interest on our project work and guided us all along, till the completion of our project work by providing all the necessary information for developing a good system.

I am thankful to and fortunate enough to get constant encouragement, support and guidance from all Teaching staffs of School of Computing and Electrical Engineering which helped us in successfully completing our project work. Also, I would like to extend our sincere esteems to all staff in laboratory for their timely support.

Signature of Student

Name: Sakshi Sharma (18BT010443)

Name: Shobhit Sharma (18BT010449)

Name: Tushar Parmar (18BT010457)

Date: ______________________________

Industrial Project Evaluation Sheet from Organization

ABOUT THE INSTITUTE

IIT Mandi (Indian Institute of Technology Mandi) is located in the Shivalik Range of the Himalayas, far from the hustle and bustle of the city. It is located in the Kamand Valley, about 18 kilometers from the historic town of Mandi, on the banks of the Uhl, a tributary of the Beas. IIT Mandi has progressed to impressive heights since its inception in 2009. It now has a fully residential campus with world-class academic and research facilities in such a short period.

In an India marching towards a just, inclusive, and sustainable society, IIT Mandi's vision is to be a leader in science and technology education, knowledge creation, and innovation. IIT Mandi has several areas of focus in order to achieve this vision. The importance of the local mountain region, as well as India, is a common theme. To name a few, there are:

- Research on novel materials is driven by the needs of electrical and electronics engineering at the Advanced Materials Research Centre (AMRC). Over Rs. 30 crores in state-of-the-art instruments have been installed.
- BioX is a project exploring the intersections of life sciences and engineering to meet the demands for affordable health care, agricultural innovations, and environmental protection.
- Energy: multidisciplinary researchers are working to increase energy supply sustainably.
- The Centre for Design and Fabrication of Electronic Devices (C4DFED) conducts research on microelectronic materials and devices with a strong emphasis on industrial relevance.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

HAR	Human Activity Recognition
ML	Machine Learning
NLP	Natural Language Processing
CRM	Customer Relationship Management
BI	Business Intelligence
HRIS	Human resource information systems
API	Access Point Interface
CNN	Convolutional Neural Network
IME	Input Method Editor
SCM	Source Code Management
CNTK	Microsoft Cognitive Toolkit
AI	Artificial Intelligence
PCA	Principal Component Analysis
ACS	Applied Cognitive Science ACS
ID3	Iterative Dichotomiser 3
C4.5	Cervical Segment 4.5
CART	Classification and Regression Tree
CHAID	Chi-square Automatic Interaction Detection
MARS	Multivariate Adaptive Regression Splines
IG	Information gain
MSE	Mean Squared Error
K-NN	K-Nearest Neighbor
SVM	Support Vector Machine
SVC	Support Vector Classifier
MLP	Multilayer Perceptron
ReLU	Rectified Linear Activation Unit
LTU	Linear Threshold Unit
LSTM	Long Short-term Memory
RNN	Recurrent Neural Network
JSON	JavaScript Object Notation
CSV	Comma-Separated Value

CHAPTER I

1. INTRODUCTION

1.1. HUMAN ACTIVITY RECOGNITION

Human activity recognition, or HAR, is a time series classification task in which smartphones' sequenced data is classified into known well-defined movements. There are two traditional approaches for the time series-based data on fixed-sized windows and training machine learning models, such as for ensembles of decision trees.

In this experiment, we will learn about the problem of recognizing human activity and the recurrent neural network and deep learning neural network models that are achieving state-of-the-art performance on it. The HAR project aims to analyze human behavior in a challenging search-and-retrieve scenario through artificial intelligence technology; the notion is that given a system specification; the user can synthesize a system that fulfills his requirements by following the process and using the tools provided to support it.

1.2. MOTIVATION

Understanding human action and their interaction with the surrounding is a key element for developing the aforementioned intelligent system. Human activity recognition deals with the problems generated in integrating sensing and reasoning to provide context-aware data that can confer personalized support across an application.

For example, imagine a smart home equipped with various sensors and devices that can detect the working of all the appliances and the presence of people in the home. It is possible to deduce the various activities performed by a person inside the home based on the sensors signal with other relevant factors like time domain and date (like a person in the morning is supposed to be in the kitchen and the coffee machine suggests that person is making breakfast). That is why the collected HAR can be absorbed to anticipate future people's demands and can be responsible for their purpose (example-automatic temperature set, automatic controlling of light, etc.).

In the Human Activity Recognition system, various issues still need to be addressed, like battery limitation of wearable sensors, difficulty in performing HAR (Human Activity Recognition) in real-time, and lack of entirely ambient systems able to reach users anytime.

1.3. LITERATURE REVIEW

Many researchers have studied human activity recognition, and each has proposed a different solution to tackle the associated problems. The primary goal of this work was to fill a gap in the literature regarding how humans might perform in a complex search and retrieve environment and regarding how the presence of feedback influences human performance in such activities. While previous studies have looked at various psychological and neurological factors that impact human performance in similar tasks requiring lower cognitive work, not much is known about the literature addressed in this experiment.

To recognize the activity or to accomplish the HAR task, the type and quantity of the target and distractors were increased in the test phase, and the performance of the human participants improved steadily. This demonstrates the ability of the human brain to be able to generalize between training and testing. Over time the performance of human participants improved for both the training and test phase (the total scores increased); this boost in human performance is most likely due to participants' growing acquaintance with their surroundings. After getting the dataset, another essential part is data preprocessing. As the quality of input impacts the desired result, quality data needs to be collected to get better accuracy.

Various machine learning problems that are time-consuming and labor- expensive are being solved by active learning techniques. Some applications shown in Fig. 1.3.1 include speech recognition, information extraction or handwriting recognition, etc.

Advantages of HAR:

a. It gives fast and real-time predictions of problems
b. Efficiently utilizes the resources
c. Helps in the automation of different tasks
d. It is used in business, medicine, sports, etc.
e. It helps to interpret the previous behavior of the model.

Applications of HAR:

a. **Surveillance:** Cameras are installed in areas that may need monitoring, such as banks, airports, military installations, and convenience stores. Currently, surveillance systems are mainly for recording. The Aim of activity detection using CCTV‟s is to monitor suspicious activities for real-time reactions like fighting and stealing
b. **Sports play analysis:** analyzing the play and deducing the actions in the sport.
c. **Unmanned Aerial Vehicles (UAVs):** Automated understanding of aerial images. Recognition of military activities like border security, people in bunkers etc.
d. **Monitor Health:** Analyze a person's activity from the information collected by different devices.
e. **Discover activity partners:** Discover which are the variables that determine which activity is doing a person.

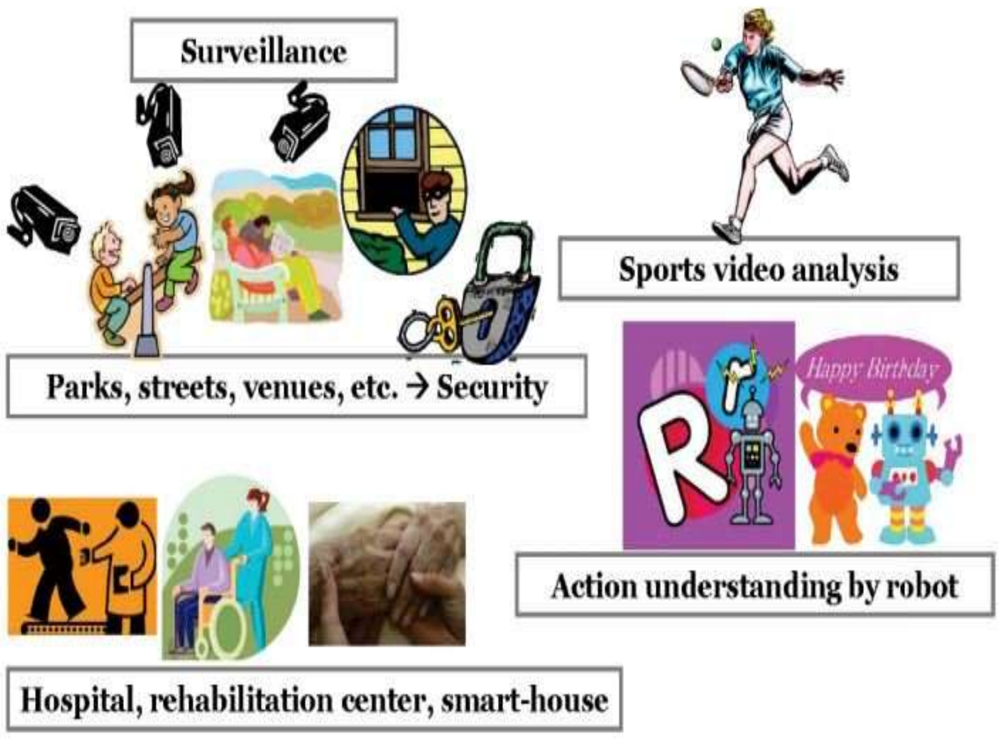

Surveillance
Sports video analysis
Parks, streets, venues, etc. → Security
Happy Birthday
Action understanding by robot
Hospital, rehabilitation center, smart-house

Fig. 1.3.1: HAR Applications. Source: [9]

f. **Detect activity:** Calculate a predictive model that can recognize a person's activity from the signals received by the sensors.
g. **Improve wellbeing:** Design individualized exercise tables to improve the health of a person

CHAPTER – II

MONTHLY PROGRESS REPORT

MONTH	MONTH START DATE	MONTH END DATE	MONTH ONJECTIVE/TASK
1	25/03/2022	25/04/2022	Assignments based on Learning:
2	26/04/2022	26/05/2022	Assignments based on Data Handling: • Choosing a dataset • Uploading the dataset in the drive to work on google colaboratory • Dataset cleaning and data Preprocessing
3	27/05/2022	25/06/2022	Assignments based on Model and result: • Choosing a model and building supervised, and deep learned network model

CHAPTER III

TECHINCAL CONTENT

3.1. INTRODUCTION

3.1.1. Human Activity Recognition using Machine Learning

This project aims to apply machine learning algorithms to real-world data sets to study their accuracy and draw valuable conclusions. It consists of predicting a bot's movement-based coordinates and rewards collected. It traditionally entails deep domain expertise to correctly engineer raw data features to fit a machine learning model. Deep learning methods such as convolutional neural networks and recurrent neural networks like LSTMs have recently demonstrated that they can learn features from raw sensor data and even achieve state-of-the-art results on challenging activity recognition tasks.

Machine learning is vital because it gives enterprises a view of trends in customer behavior and operational business patterns and supports the development of new products. Many of today's leading companies, such as Facebook, Google, and Uber, make machine learning a central part of their operations. Machine learning has become a significant competitive differentiator for many companies.

Here are significant advantages of using Machine Learning technology:

a. **Customer relationship management:** CRM software can use machine learning models to analyze email and prompt sales team members to respond to the most critical messages.
b. **Business intelligence:** BI and analytics vendors use machine learning in their software to identify potentially essential data points, patterns of data points, and anomalies.
c. **Human resource information systems:** HRIS systems can use machine learning models to filter applications and identify the best candidates for an open position.
d. **Self-driving cars:** Machine learning algorithms can make it possible for a semi-autonomous car to recognize a partially visible object and alert the driver.
e. **Virtual assistants:** Smart assistants typically combine supervised and unsupervised machine learning models to interpret natural speech and supply context.

3.1.2. Machine Learning types:

Machine learning involves three different types of tasks given in Fig. 3.1.2.1:

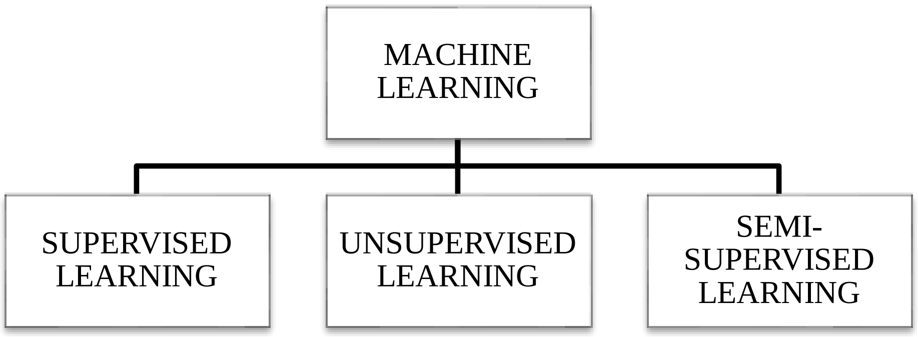


Fig. 3.1.2.1: Machine Learning Types

Supervised machine learning:

Supervised learning is machine learning, where we have the input variables (X) and an output variable (Y), and we use an algorithm for mapping the input variables to the output variable as given in equation(1).

$$Y = f(X) \tag{1}$$

The aim is to increase the accuracy by which the mapping of the function so well that when we have the new input data (x), the program can predict the output variables (Y) for that given data.

This process is called supervised learning because the process of algorithm learning from the training dataset can be related to that of a teacher supervising the learning process.

Supervised learning can be further grouped into:

a. Classification: Here, the output variable will be a category. E.g., Disease/No disease and Male/Female, etc.
b. Regression: Here, the output variable will be a real continuous value. E.g., Price, height, weight, etc.

Unsupervised Learning:

Unsupervised learning is where we have the input data (X) and no output variables. The main goal of unsupervised learning is to model the underlying structure or distribution in the data, which is a process of learning more about the data.

It is called unsupervised learning because there are no correct answers, unlike supervised learning.

Unsupervised learning can be further grouped into:

a. Clustering: A clustering problem is where we want to figure out the inherent groupings in the data given.
b. Association: An association rule learning problem is where we want to find rules that can describe large portions of your data, such as people that buy milk also tend to buy bread.

Semi-supervised Learning:

It is a problem where we have a large sum of input data (X), and only some of the data labeled (Y) are called semi-supervised learning problems. These problems come in between both supervised and unsupervised learning.

3.2. Machine Learning Process:

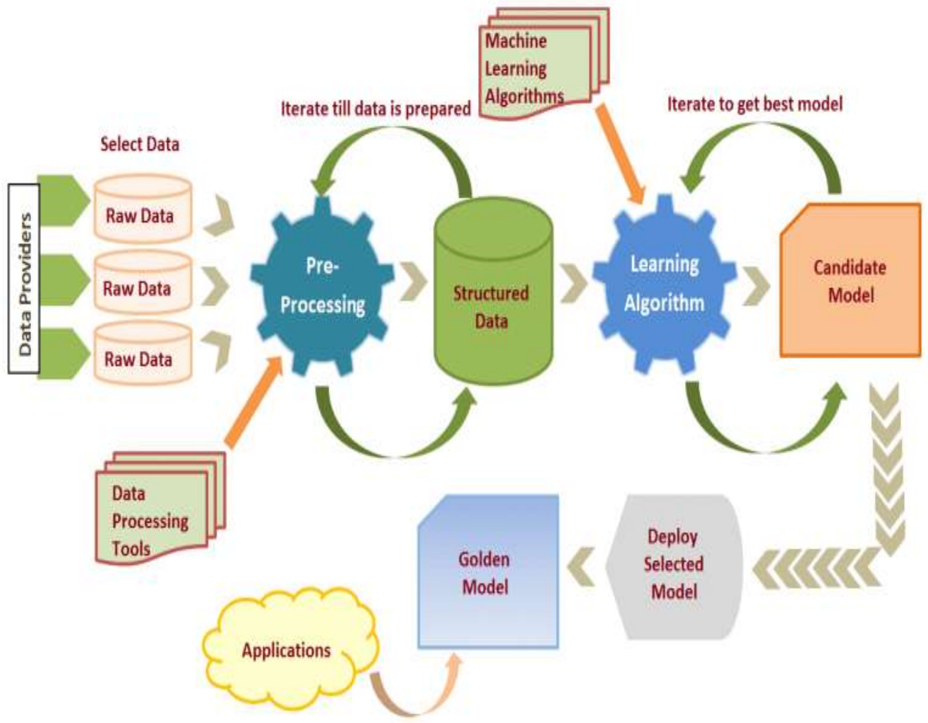


Fig. 3.2.1: Machine Learning Process. Source: [10]

3.2.1. Gather Data

Gathering data is the most important step as shown in Fig. 3.2.1 in solving any machine learning problem. Text classifier can only be as good as the dataset it is built form.

Here are some important things to remember when collecting data:

a. If you are using a public API, understand the limitations of the API before using them. For example, some APIs limit the rate you can make queries.
b. The results are better if you have more training examples (referred to as samples in the rest of this guide). This will help your model to generalize it much better.
c. Ensure the number of samples for every class or topic is not overly imbalanced. You should have a comparable number of samples in each class.
d. Ensure that your samples cover the space of possible inputs, not only the common cases.

Types of data:

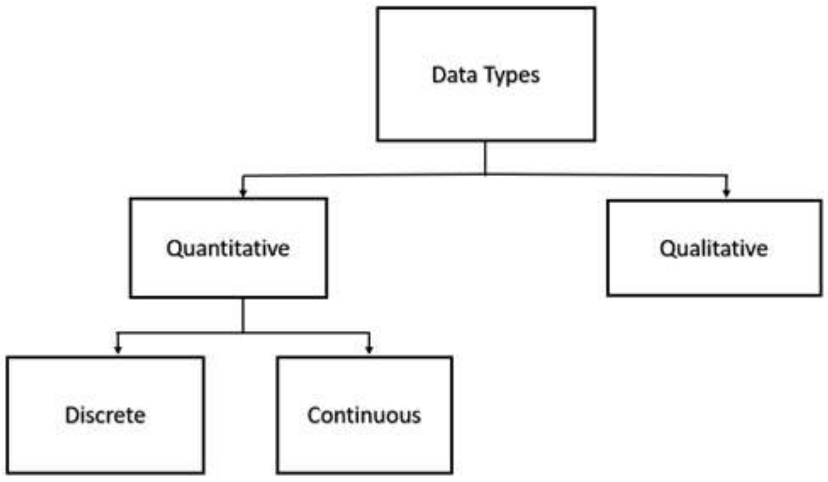


Fig. 3.2.1.1: Types of Data

1. **Numerical data:**

Numerical data is measurable information, and it is, of course, data represented as numbers, not words or text. Numerical data are of two types:

a. **Continuous numbers** are numbers that do not have a logical end to them. Examples include variables that represent money or height.
b. **Discrete numbers** are the opposite; they have a logical end to them. Some examples include variables for days in the month or the number of bugs logged.

2. **Categorical data:**
Categorical data is any data that is not a number, meaning a string of text or a date. These variables can be broken down into nominal and ordinal values, though you will not often see this done. There are three types of categorical data are:

a. **Ordinal values** are values that have a set order to them. Examples of ordinal values include having a priority on a bug such as “Critical” or “Low” or the ranking of a race as “First” or “Third.”
b. **Nominal values** are the opposite of ordinal values, representing values with no set order. Nominal value examples include variables such as “Country” or “Marital Status.”
c. **Binary data types** only have two values – yes or no. This can be represented differently, such as “True” and “False” or 1 and 0. Binary data is used heavily for classification machine learning models.

3.2.2. Explore Your Data

Building and training a model is only one part of the workflow. Understanding the characteristics of your data beforehand will enable you to build a better model. This could simply mean obtaining a higher accuracy. It could also mean requiring less data for training or fewer computational resources.

Load the Dataset: First up, let us load the dataset into Python, as shown in Fig. 3.2.2.1.

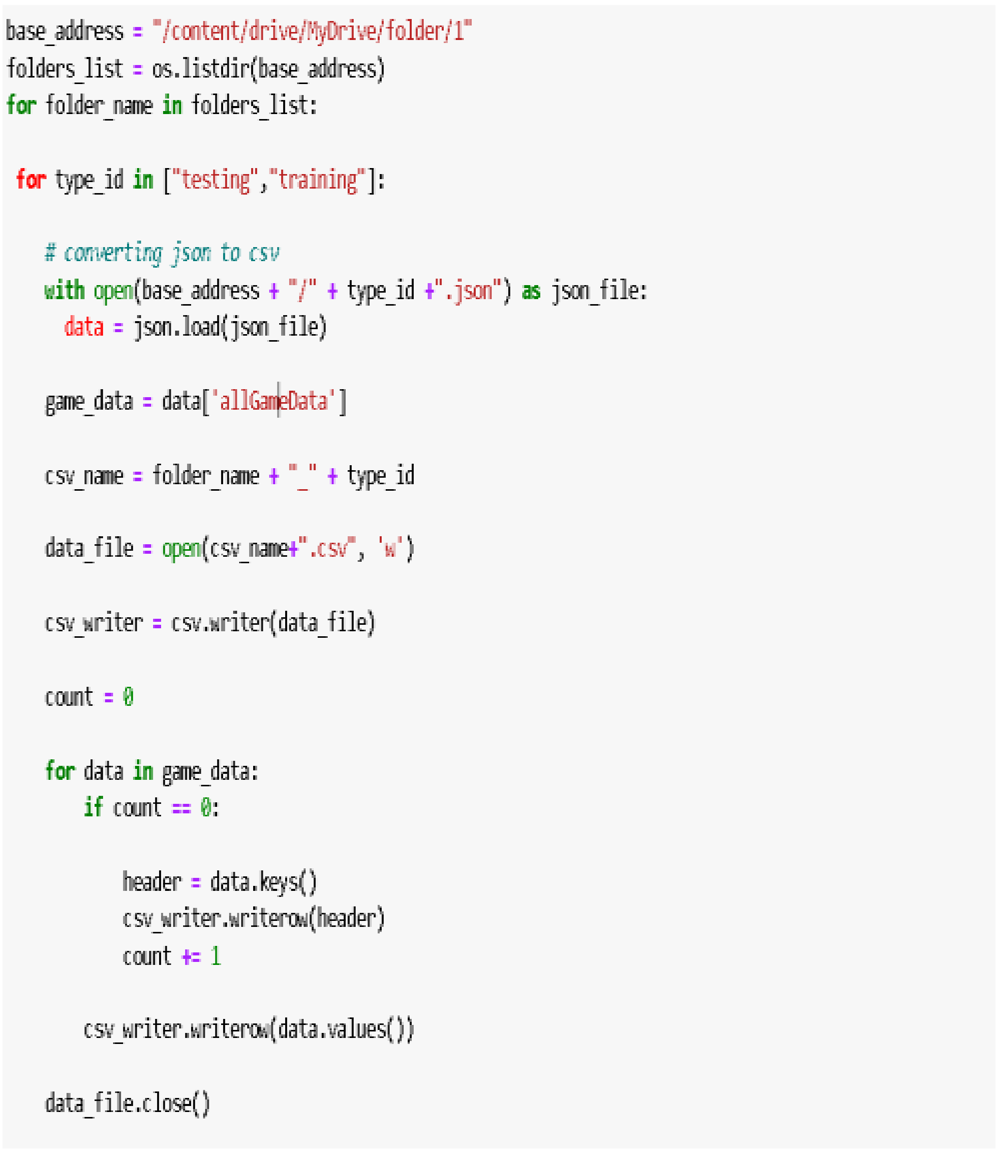
3.2.3. Check the Data

After loading the data, it is good practice to run some checks on it: pick a few samples and manually check if they are consistent with your expectations. For example, print a few random samples to see if the sentiment label corresponds to the review's sentiment. We picked a review randomly from the IMDb dataset: “Ten minutes worth of story stretched out into the better part of two hours. I should have left when nothing significant happened at the halfway point.” The common sentiment (negative) matches the sample’s label.

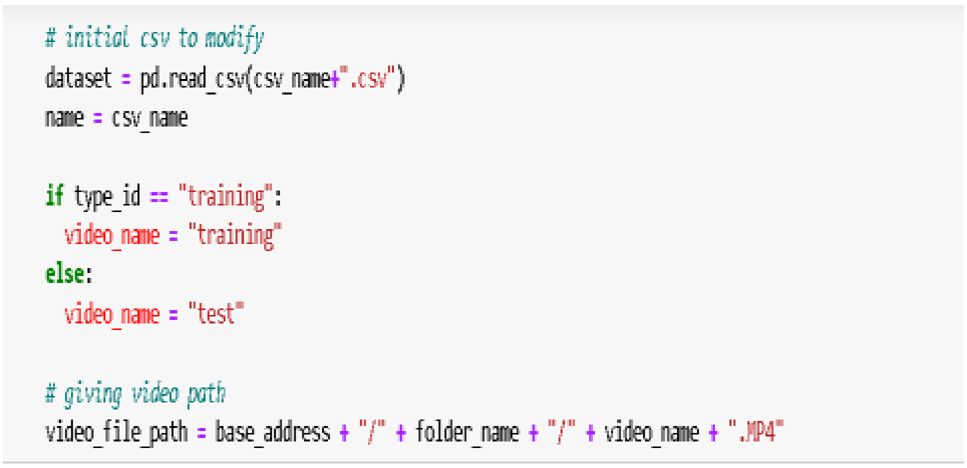
3.2.4. Collect Key Metrics

Once you have verified the data, collect the following important metrics that can help characterize your text classification problem:

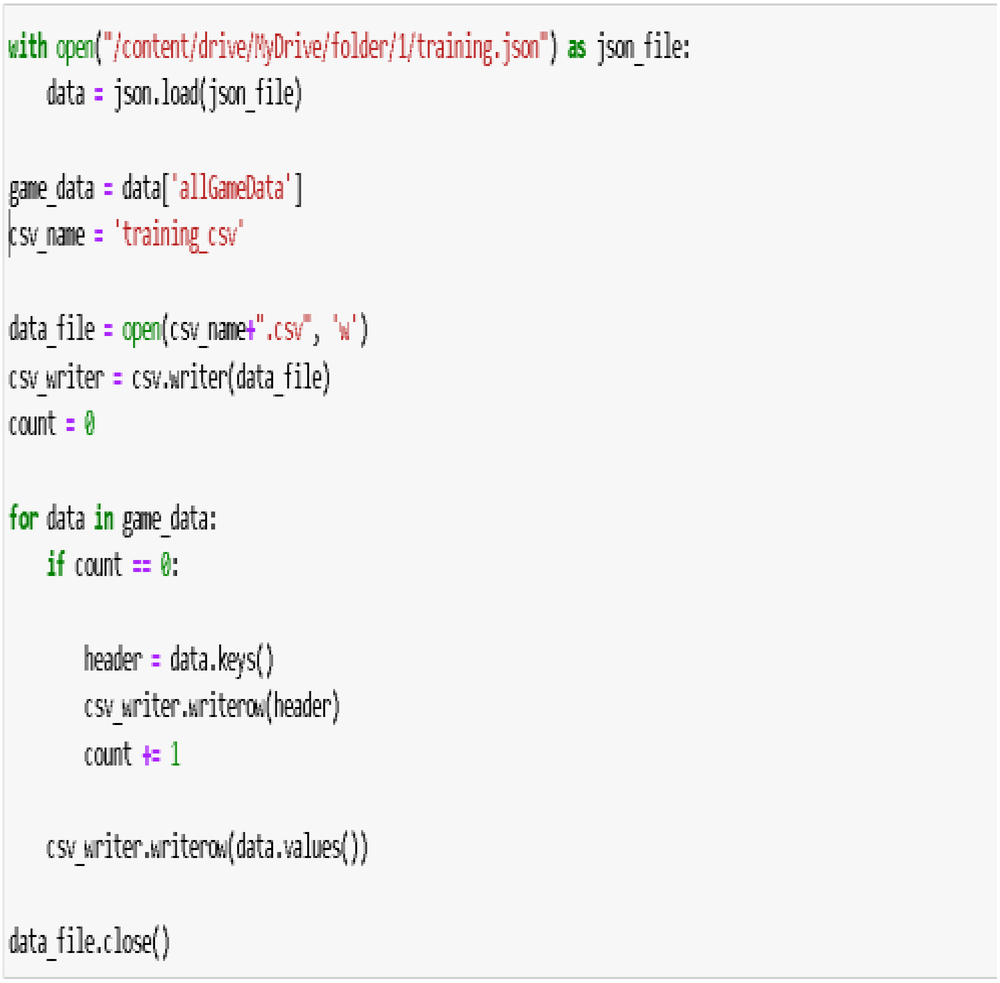
a. Number of samples: Total number of examples you have in the data.
b. Number of classes: Total number of topics or categories in the data.
c. The number of samples per class: Number of samples per class. All classes will have a similar number of samples; in an imbalanced dataset, the number of samples in each class will vary widely.
d. Number of words per sample: Median number of words in one sample.
e. Frequency distribution of words: Frequency is distributed as showing the number of occurrences of each word in the dataset.
f. The distribution of sample length shows the number of words per sample in the dataset.



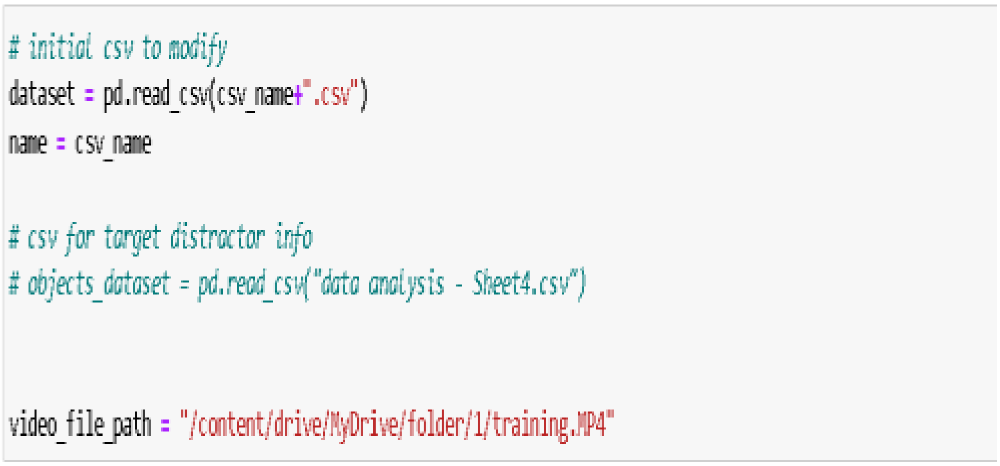
```python
base_address = "/content/drive/MyDrive/folder/1"
folders_list = os.listdir(base_address)
for folder_name in folders_list:

 for type_id in ["testing","training"]:

    # converting json to csv
    with open(base_address + "/" + type_id +".json") as json_file:
      data = json.load(json_file)

    game_data = data['allGameData']

    csv_name = folder_name + "_" + type_id

    data_file = open(csv_name+".csv", 'w')

    csv_writer = csv.writer(data_file)

    count = 0

    for data in game_data:
        if count == 0:

            header = data.keys()
            csv_writer.writerow(header)
            count += 1

        csv_writer.writerow(data.values())

    data_file.close()
```



```python
# initial csv to modify
dataset = pd.read_csv(csv_name+".csv")
name = csv_name

if type_id == "training":
  video_name = "training"
else:
  video_name = "test"

# giving video path
video_file_path = base_address + "/" + folder_name + "/" + video_name + ".MP4"
```



```
with open("/content/drive/MyDrive/folder/1/training.json") as json_file:
    data = json.load(json_file)

game_data = data['allGameData']
csv_name = 'training_csv'

data_file = open(csv_name+".csv", 'w')
csv_writer = csv.writer(data_file)
count = 0

for data in game_data:
    if count == 0:

        header = data.keys()
        csv_writer.writerow(header)
        count += 1

    csv_writer.writerow(data.values())

data_file.close()
```



```
# initial csv to modify
dataset = pd.read_csv(csv_name+".csv")
name = csv_name

# csv for target distractor info
# objects_dataset = pd.read_csv("data analysis - Sheet4.csv")


video_file_path = "/content/drive/MyDrive/folder/1/training.MP4"
```

Fig. 3.2.2.1: Loading the dataset

3.2.5 Choose a Model

At this point, we have assembled our dataset and gained insights into the key characteristics of our data. Next, based on the metrics we gathered, we should consider which classification model we should use. This means asking questions such as, "How do we present the text data to an algorithm that expects numeric input?” (This is called data preprocessing and vectorization), “What type of model should we use?” and “What configuration parameters should we use for our model?” etc. The model selection algorithm and flowchart below in Fig. 3.2.5.1 are a summary.

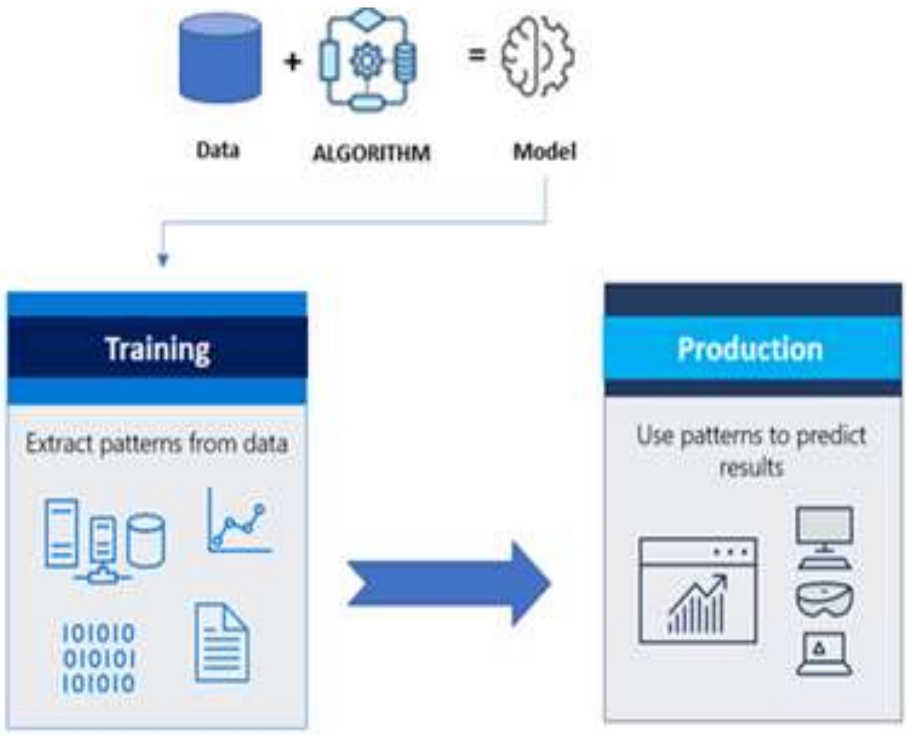


Fig. 3.2.5.1: Model Selection Algorithm. Source: [11]

In the flowchart above, the yellow boxes indicate data and model preparation processes. Grey boxes and green boxes indicate choices we considered for each process. Green boxes indicate our recommended choice.

3.2.6. Prepare the Data

Before our data can be fed to a model, it needs to be transformed into a format the model can understand.

First, the data samples we have gathered may be in a specific order. We do not want any information associated with ordering samples to influence the relationship between texts and labels. For example, if a dataset is sorted by class and then split into training/validation sets, these sets will not represent the overall distribution of data.

A simple best practice to ensure the model is not affected by data order is constantly shuffling the data before doing anything else. If the data is already split into training and validation sets, make sure to transform it into validation data the same way we transform our training data. If we do not already have separate training and validation sets, we can split the samples after shuffling; it is typical to use 80% of the samples for training and 20% for validation.

Second, machine learning algorithms take numbers as inputs. This means that we will need to convert the texts into numerical vectors. There are two steps to this process:

a. Tokenization: Divide the texts into words or smaller sub-texts, enabling good generalization of the relationship between the texts and the labels. This determines the “vocabulary” of the dataset (set of unique tokens present in the data).
b. Vectorization: Define a good numerical measure to characterize these texts.

3.2.7. Train our Model

Now that we have constructed the model architecture, we need to train the model. Training involves making a prediction based on the model's current state, calculating how incorrect the prediction is, and updating the weights or parameters of the network to minimize this error and make the model predict it better. We repeat this process until our model has converged and can no longer learn. There are three key parameters to be chosen for this process.

a. **Metric:** How to measure the performance of our model using a metric? We used accuracy as the metric in our experiments.
b. **Loss function:** A function that is used to calculate a loss value that the training process attempts to minimize by tuning the network weights. For classification problems, cross-entropy loss works well.
c. **Optimizer:** A function that decides how the network weights will be updated based on the output of the loss function. We used the popular Adam optimizer in our experiments.

3.2.8. Tune Hyperparameters

We had to choose some hyperparameters for defining and training the model. We relied on intuition, examples, and best practice recommendations. However, our first choice of hyperparameter values may not yield the best results. It only gives us a good starting point for training. Every problem is different, and tuning these hyperparameters will help refine our model better to represent the particularities of the problem at hand. Let us take a look at some of the hyperparameters we used and what it means to tune them:

a. **The number of layers in the model:** The number of layers in a neural network indicates its complexity. We must be careful in choosing this value. Too many layers will allow the model to learn too much information about the training data, causing overfitting. Too few layers can limit the model's learning ability, causing underfitting.

b. **The number of units per layer:** The units in a layer must hold the information for the transformation that a layer performs. For the first layer, this is driven by the number of features. In subsequent layers, the number of units depends on expanding or contracting the representation from the previous layer.
c. **Dropout rate:** Dropout layers are used in the model for regularization. They define the fraction of input to drop as a precaution for overfitting. Recommended range: 0.2–0.5.
d. **Learning rate:** This is the rate at which the neural network weights change between iterations. A significant learning rate may cause large weight swings, and we may never find their optimal values. A low learning rate is reasonable, but the model will take more iteration to converge.

There are couples of additional hyperparameters we tuned that are specific to our CNN model:

a. **Kernel size:** The size of the convolution window. Recommended values: 3 or 5.
b. **Embedding dimensions:** The number of dimensions we want to use to represent word embedding—i.e., the size of each word vector. Recommended values: 50–300.

3.2.9. Deploy our Model

Following are the key things when deploying the model:

a. Make sure production data follows the same distribution as our training and evaluation data.
b. Regularly re-evaluate by collecting more training data.
c. If our data distribution changes, retrain the model.

3.3. Description of Tools Used

3.3.1 Unity

Tool Description Unity is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Inc.'s Worldwide Developers Conference as a Mac OS X-exclusive game engine. The engine has since been gradually extended to support a variety of desktop, mobile, console, and virtual reality platforms. It is prevalent in iOS and Android mobile game development and is used for games such as Pokémon Go, Monument Valley, Call of Duty: Mobile, Beat Saber, and Cuphead. It is considered easy for beginner developers and is famous for indie game development.

In Fig. 3.3.1.1., the engine can create three-dimensional (3D) and two-dimensional (2D) games, interactive simulations, and other experiences. Industries have adopted the engine outside video gaming, such as film, automotive, architecture, engineering, construction, and the United States Armed Forces.

User Interface

A. **The Toolbar** provides access to the essential working features. The left contains the basic tools for manipulating the Scene view and the GameObject within it. In the center are the play, pause, and step controls. The buttons to the right give you access to Unity Collaborate, Unity Cloud Services, and your Unity Account, followed by a layer visibility menu, and finally, the Editor Layout menu (which provides some alternate layouts for the Editor windows and allows you to save your custom layouts).

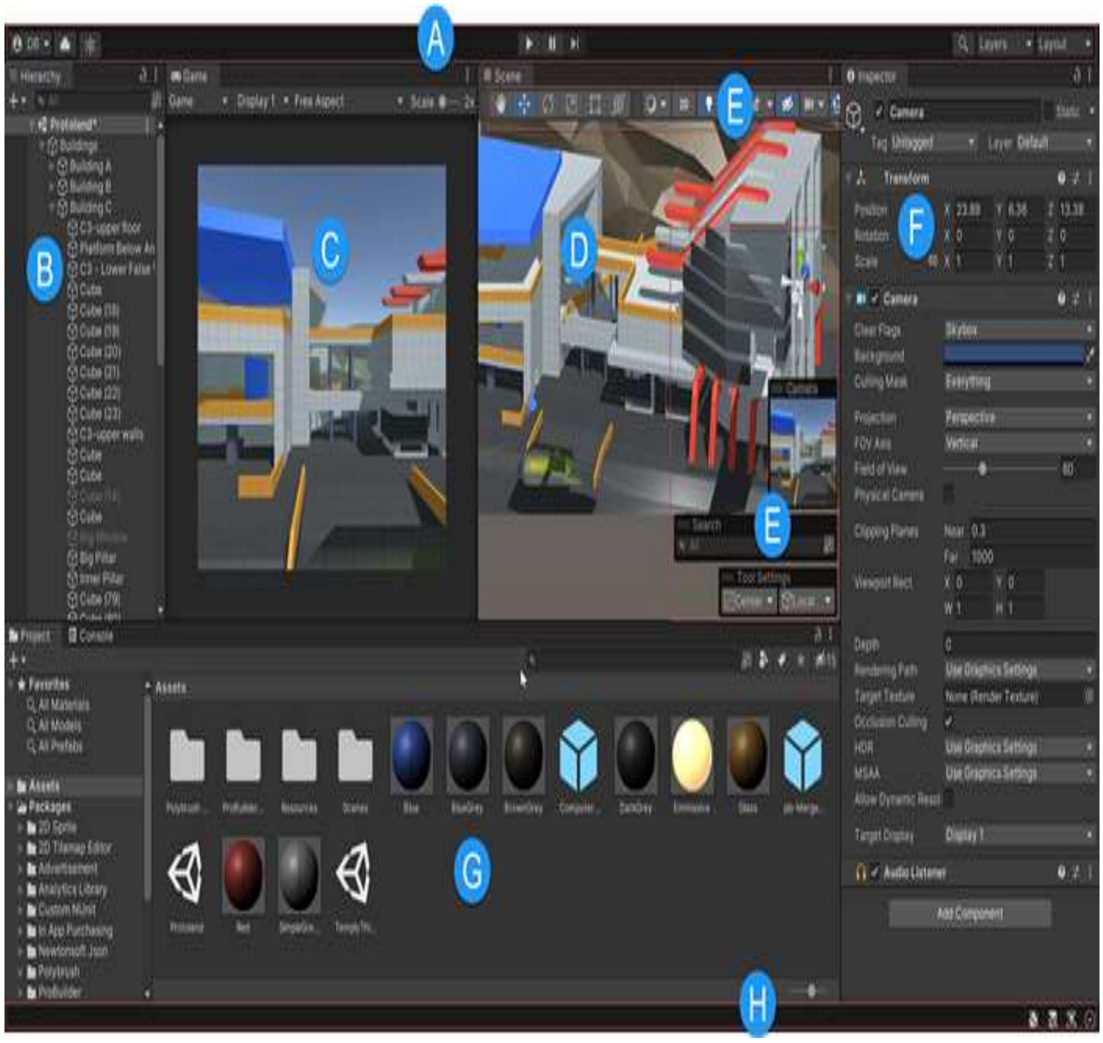


Fig. 3.3.1.1: Unity User Module. Source[]

B. **The Hierarchy** window is a hierarchical text representation of every GameObject in the Scene. Each item in the Scene has an entry in the hierarchy, so the two windows are inherently linked. The hierarchy reveals the structure of how GameObjects attach.
C. **The Game view** simulates what your final rendered game will look like through your Scene Cameras. When you click the Play button, the simulation begins.
D. **The Scene view** allows you to navigate and edit your Scene visually. The Scene view can show a 3D or 2D perspective, depending on the type of project you are working on.
E. **Overlays** contain the basic tools for manipulating the Scene view and the GameObjects within it. You can also add custom Overlays to improve your workflow.
F. **The Inspector Window** allows you to view and edit all the properties of the currently selected GameObject. Because different types of GameObjects have different sets of properties, the layout and contents of the Inspector window change each time you select a different GameObject.
G. **The Project window** displays your library of Assets that are available to use in your Project. When you import Assets into your Project, they appear here.
H. **The status bar** provides notifications about various Unity processes, and quick access to related tools and settings.

Unity Features

a. 2D and 3D mode settings
b. Preferences
c. Shortcuts Manager
d. Build Settings
e. Project Settings
f. Visual Studio C# integration
g. RenderDoc Integration
h. Editor Analytics
i. Check for Updates
j. IME in Unity
k. Version Control
l. Plastic SCM plugin for Unity
m. Safe Mode
n. Command-line arguments
o. Text-Based Scene Files
p. Troubleshooting the Editor

3.3.2 PYTHON

Python is an interpreter, object-oriented, high-level programming language with dynamic semantics. It is a high-level built-in data structure, combined with dynamic typing and dynamic binding, making it very attractive for Rapid Application Development and for use as a scripting or glue language to connect existing components. Python's simple, easy-to-learn syntax emphasizes readability and reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms and can be freely distributed.

REASONS FOR USING PYTHON LANGUAGE FOR ML

It has many libraries and frameworks: The Python language comes with many libraries and frameworks that make coding accessible. This also saves a significant amount of time. The most popular libraries are NumPy, used for scientific calculations; SciPy for more advanced computations; and scikit, for learning data mining and data analysis. These libraries work alongside powerful frameworks like TensorFlow, CNTK, and Apache Spark. These libraries and frameworks are essential for machine and deep learning projects.

Simplicity: Python code is concise and readable even to new developers, which is beneficial to machine and deep learning projects. Due to its simple syntax, the development of Python applications is fast compared to many programming languages. Furthermore, it allows the developer to test algorithms without implementing them. Readable code is also vital for collaborative coding. Many individuals can work together on a complex project. One can easily find a Python developer for the team, as Python is a familiar platform. Therefore, a new developer can quickly get acquainted with Python’s concepts and work on the project instantly.

The massive online support: Python is an open-source programming language and enjoys excellent support from many resources and quality documentation worldwide. It also has a large and active community of developers who provide their assistance at any stage of development. Most scientists have adopted Python for Machine Learning and a Deep Learning project, which means most of the brightest minds worldwide, can be found in Python communities.

a. **Fast development:** Python has a syntax that is easy to understand and friendly. Furthermore, the numerous frameworks and libraries boost software development.
b. **Flexible integrations:** Python projects can be integrated with other systems coded in different programming languages. It is much easier to blend it with other AI projects written in other languages.

c. **Fast code tests:** Python provides many code review and test tools. Developers can quickly check the correctness and quality of the code. AI projects tend to be time-consuming, so a well-structured environment for testing and checking for bugs is needed. Python is the ideal language since it supports these features.
d. **Performance:** It is a superset of Python language designed to achieve code performance like C language. Developers can use Python to code C extensions the same way they code in Python, as its syntax is almost the same. Python increases the language performance significantly.
e. **Visualization tools:** In AI, Machine learning, and deep learning, it is essential to present data in a human-readable format. Some libraries like Matplotlib enable data scientists to generate charts, histograms, and plots to represent data and visualization better. Also, the different APIs that Python supports enhance the visualization process.

3.3.3 Principal Component Analysis

Principal Component Analysis is an unsupervised learning algorithm used for dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation.

Reducing the number of variables of a data set naturally comes at the expense of accuracy. However, the trick in dimensionality reduction is to trade a little accuracy for simplicity. Smaller data sets are easier to explore and visualize, making analyzing data much easier and faster for machine learning algorithms without extraneous variables.

The PCA algorithm is based on some mathematical concepts such as:

a. Variance and Covariance
b. Eigenvalues and Eigen factors

Some standard terms used in the PCA algorithm:

a. **Dimensionality** is the number of features or variables present in the given dataset.
b. **Correlation:** It signifies how strongly two variables are related to each other. Such as, if one changes, the other variable also gets changed.
c. **Orthogonal:** It defines that variables are not correlated to each other. Hence the correlation between the pair of variables is zero.
d. **Eigenvectors:** If there is a square matrix M, a non-zero vector v is given. Then v will be an eigenvector if Av is the scalar multiple of v.
e. **Covariance Matrix:** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

A step-by-step explanation of PCA:

STEP 1: STANDARDIZATION

This step aims to standardize the range of the continuous initial variables so that each one contributes equally to the analysis.

More specifically, it is critical to perform standardization prior to PCA because the latter is quite sensitive regarding the variances of the initial variables. That is, if there are significant differences between the ranges of initial variables, those variables with more extensive ranges will dominate over those with small ranges (For example, a variable that ranges between 0 and 100 will dominate over a variable that ranges between 0 and 1), which will lead to biased results. So, transforming the data to comparable scales can prevent this problem.

Mathematically, this can be done by subtracting the mean and dividing by the standard deviation for each variable's value.

$$\mathrm{Z} = \frac{Value - mean}{standard\ deviation} \tag{2}$$

Once the standardization is done, all the variables will be transformed to the same scale.

STEP 2: COVARIANCE MATRIX COMPUTATION

This step aims to understand how the input data set variables vary from the mean to each other, or in other words, to see if there is any relationship between them. So, in order to identify these correlations, we compute the covariance matrix.

The covariance matrix is a p × p symmetric matrix (where p is the number of dimensions) that has as entries the covariance associated with all possible pairs of the initial variables. For example, for a 3-dimensional data set with three variables x, y, and z, the covariance matrix is a 3×3 matrix of this from:

$$\begin{matrix} \mathrm{Cov}\ (\mathrm{x},\mathrm{x}) & \mathrm{Cov}\ (\mathrm{x},\mathrm{y}) & \mathrm{Cov}\ (\mathrm{x},\mathrm{z}) \\ \mathrm{Cov}\ (\mathrm{y},\mathrm{x}) & \mathrm{Cov}\ (\mathrm{y},\mathrm{y}) & \mathrm{Cov}\ (\mathrm{y},\mathrm{z}) \\ \mathrm{Cov}\ (\mathrm{z},\mathrm{x}) & \mathrm{Cov}\ (\mathrm{z},\mathrm{y}) & \mathrm{Cov}\ (\mathrm{z},\mathrm{z}) \end{matrix} \tag{3}$$

STEP 3: COMPUTE THE EIGENVECTORS AND EIGENVALUES OF THE COVARIANCE MATRIX TO IDENTIFY THE PRINCIPAL COMPONENTS

Eigenvectors and eigenvalues are the linear algebra concepts that we need to compute from the covariance matrix to determine the data's principal components. The Covariance matrix's eigenvectors are the axes' directions where there is the most variance (most information), which we call Principal Components. Moreover, eigenvalues are simply the coefficients attached to eigenvectors, which give the amount of variance carried in each Principal Component. By ranking eigenvectors in their eigenvalues, highest to lowest, we get the principal components in order of significance.

HOW PCA CONSTRUCTS THE PRINCIPAL COMPONENTS

As there are as many principal components as variables in the data, principal components are constructed so that the first principal component accounts for the most significant possible variance in the data set. The second principal component is calculated similarly, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and accounts for the following highest variance that is clearly seen from Fig. 3.3.2.1.

This continues until a total of p principal components have been calculated, equal to the original number of variables.

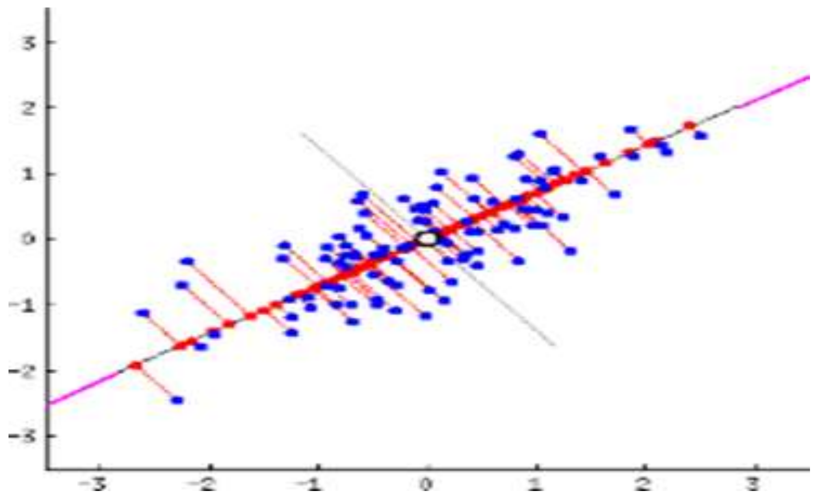


Fig. 3.3.2.1: Principal Components. Source: [12]

STEP 4: FEATURE VECTOR

As we saw in the previous step, computing the eigenvectors and ordering them by their eigenvalues in descending order allow us to find the principal components in order of significance. In this step, we choose whether to keep all these components or discard those of lesser significance (of low eigenvalues) and form with the remaining ones a matrix of vectors that we call Feature vectors.

So, the feature vector is simply a matrix that has the eigenvectors of the components we decide to keep as columns. This makes it the first step towards dimensionality reduction because if we keep only p eigenvectors (components) out of n, the final data set will have only p dimensions.

LAST STEP: RECAST THE DATA ALONG THE PRINCIPAL COMPONENTS AXES

In the previous steps, apart from standardization, you do not make any changes to the data; you just select the principal components and form the feature vector, but the input data set always remains in terms of the original axes (i.e., in terms of the initial variables).

In this step, which is the last one, the aim is to use the feature vector formed using the eigenvectors of the covariance matrix to reorient the data from the original axes to the ones represented by the principal components (hence the name Principal Components Analysis). This can be done by multiplying the original data set's transpose by the feature vector's transpose.

FinalDataSet = FeatureVectorT * StandardizedOriginalDataSetT

3.4. DESCRIPTION OF THE EXPERIMENT

The experiment for this project was conducted in the Applied Cognitive Science (ACS) Lab at the Indian Institute of Technology, Mandi.

3.4.1. Participants

We recruited 50 Indian Institute of Technology Mandi participants to perform this experiment. Out of the total recruited participants, 71% were male and 29% were female. The ages of the male and female participants varied between 22 and 39 years of age. The mean age of the participants was 25.5 years, with a standard deviation of 3.4 years. All the participants were remunerated with a fee of INR 50 for their participation in this experiment. The participant's educational levels varied, with 8.5 percent pursuing undergraduate degrees, 65 percent pursuing graduate degrees, and 26.5 percent pursuing doctoral degrees. Ninety-three percent of the participants had science, technology, engineering, or mathematics degrees. Seven percent had a humanities or social science degree. It is a part of the ACS (Applied Cognitive Science) Lab project in the Computer Science Department, IIT Mandi. It aims the development of a NET framework for providing a common platform for facilitating the use of the methodological approach developed by the ASC team and the integration of various tools developed during the execution of the experiment.

3.4.2. Experimental Design

A Unity 3D gaming environment was developed to simulate a search and retrieve game mission. Four buildings in the simulation, each consisting of targets and distractors present throughout the game. We randomly placed our targets and distractors in all directions around the game. The players’ objective was to get the highest possible score by attempting to collect as many target items as possible while avoiding distractor items. The points awarded to participants were valued at +5 points and -5 points, which could have been accomplished by collecting targets and distractors, respectively.

Our experiment consisted of two phases: the training phase and the test phase. During the training phase, 14 targets and seven distractors were randomly distributed in the game, and feedback was given to any participants. Whereas, during the test phase, 28 targets and 14 distractors were disturbed, with no feedback given to any participants. The total time to play a game is 25 minutes, divided into 15 minutes and 10 minutes for the training and test phase, respectively. In order to generalize the learning environment, various types of targets and distractors were developed in the testing phase.

3.4.3. Procedure

The human activity recognition is performed by generating datasets from Unity 3D game environment simulation data. A Unity 3D game environment simulation was developed as a search and retrieve simulation. To maximize the score, we recruited 50 participants from the IIT Mandi who played the game for a specific duration of the time. The gameplay's total time of 25 minutes is divided into two phases of training and test for 15 and 10 minutes, respectively.

Data preparation

Data Preparation for Recorded Video Dataset using PCA algorithm: The dataset is preprocessed before the PCA algorithm step by applying the following phases:

a. We collected the data by playing the game as a video of the training and test phase.
b. We took the screenshots frame by frame and reshaped them in vectorized form.
c. We applied principal component analysis (PCA) to the vector form of video data and chose 150 major components
d. I was also saving the PCA vector as a separate column and combining it.
e. Flatten and reshape the size of an array to the size of 64*96*3.
f. Thus, the finally prepared data had position coordinates (x, y, z), reward, and PCA variables as independent variables and actions as the dependent variable
g. Save the file using the .csv file extension.
h. Finally, this dataset is being used to develop machine learning models.

Data Preparation for Recorded Video Dataset using feature extraction algorithm: The dataset is preprocessed before the feature extraction step by applying the following phases:

a. Training video frames- selection of nearest frames

b. Find the number of frames and perform feature extraction on the selected training frames
c. Testing video frames- selection of nearest frames
d. Find the number of frames and perform feature extraction on the selected testing frames
e. Saving final user data.

3.5. MACHINE LEARNING ALGORITHMS

Machine learning algorithms used in this project are:

3.5.1. DECISION TREES

The Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too.

In Decision Trees, we start from the root of the tree for predicting a class label for a record. We compare the values of the root attribute with the record's attribute. Based on the comparison, we follow the branch corresponding to that value and jump to the next node

Types of decision trees are based on the type of target variable we have. It can be of two types:

a. **Categorical Variable Decision Tree:** A decision Tree has a categorical target variable called a Categorical variable decision tree.
b. **Continuous Variable Decision Tree:** A decision Tree has a continuous target variable, called a Continuous Variable Decision Tree.

Example:- Let us say we have a problem predicting whether a customer will pay his renewal premium with an insurance company (yes/ no). Here we know that customers' income is a significant variable, but the insurance company does not have income details for all customers. Now that we know this is an important variable, we can build a decision tree to predict customer income based on occupation, product, and various other variables. In this case, we are predicting values for the continuous variables.

How do Decision Trees work?

The decision to do strategic splits heavily affects a tree's accuracy. The decision criteria are different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases to the target variable. The decision tree splits the nodes on all available variables and then selects the split, resulting in the most homogeneous sub-nodes.

The algorithm selection is also based on the type of target variables. Let us look at some algorithms used in Decision Trees:

a. **ID3** → (extension of D3)
b. **C4.5** → (successor of ID3)
c. **CART** → (Classification And Regression Tree)
d. **CHAID** → (Chi-square automatic interaction detection Performs multi-level splits when computing classification trees)
e. **MARS** → (multivariate adaptive regression splines)

The ID3 algorithm builds decision trees using a top-down greedy search approach through the space of possible branches with no backtracking. As the name suggests, a greedy algorithm always makes the choice that seems to be the best at that moment.

Steps in ID3 algorithm:

a. It begins with the original set S as the root node.
b. Each iteration of the algorithm iterates through the very unused attribute of the set S and calculates the Entropy(H) and Information gain(IG) of this attribute.
c. It then selects the attribute with the smallest Entropy or Largest Information gain.
d. The selected attribute then splits the set S to produce a subset of the data.
e. The algorithm continues to recur on each subset, considering only attributes never selected before.

Attribute Selection Measures

If the dataset consists of N attributes, deciding which attribute to place at the root or different levels of the tree as internal nodes is a complicated step. Just randomly selecting any node to be the root cannot solve the issue. If we follow a random approach, it may give us bad results with low accuracy.

To solve this attribute selection problem, researchers worked and devised some solutions. They suggested using criteria like Entropy, Information gain, Gini index, Gain Ratio, Reduction in Variance, and Chi-Square.

Entropy

Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to conclude from that information. Flipping a coin is an example of an action that provides random information.

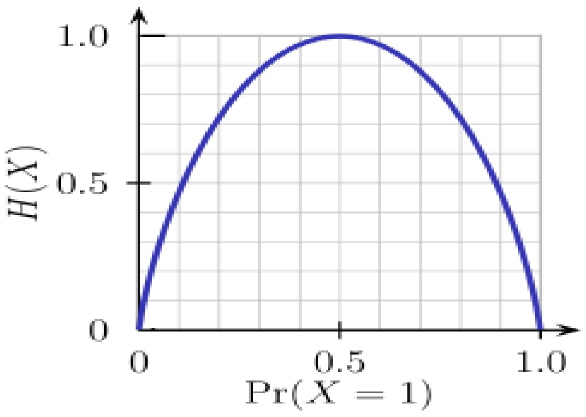


Fig. 3.5.1.1: Entropy vs. Probability graph. Source: [13]

From the above Fig. 3.5.1.1., it is quite evident that the entropy H(X) is zero when the probability is either 0 or 1. The Entropy is maximum when the probability is 0.5 because it projects perfect randomness in the data, and there is no chance of ideally determining the outcome.

Mathematically, Entropy for one attribute is represented as:

$$\mathrm{E(S)} = \sum_{I=1}^{C} - \mathrm{pi} \log_2 pi \tag{4}$$

Mathematically, Entropy for multiple attributes is represented as:

$$\mathrm{E(T,X)} = \sum_{cEX} P(c)E(c) \tag{5}$$

Where S → Current state, Pi → Probability of an event i of state S, T→ Current state, and X → Selected attribute

Information Gain

Information gain or IG is a statistical property that measures how well a given attribute separates the training examples according to their target classification given in equation(5). Constructing a decision tree is about finding an attribute that returns the highest information gain and the smallest entropy.

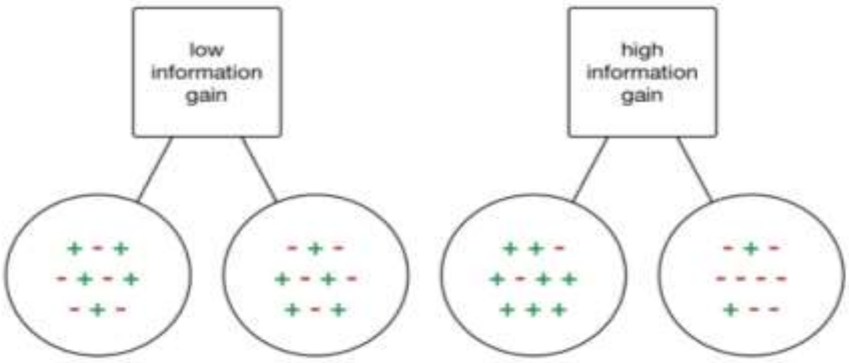


Fig. 3.5.1.2: Low vs. High Information Gain. Source: [14]

Information gain is a decrease in entropy. It computes the difference between entropy before and average entropy, from equation (5), after the split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain.

Mathematically, IG is represented as:

$$\text{Information Gain}(Y,X) = \text{Entropy}(T) - \text{Entropy}(T,X) \qquad (6)$$

In a much simpler way, we can conclude that:

$$\text{Information Gain} = \text{Entropy (before)} - \sum_{j=1}^{K} \text{Entropy}(j, \text{after}) \qquad (5)$$

Where "before" is the dataset before the split, K is the number of subsets generated by the split, and (j, after) is subset j after the split.

Gini Index works with the categorical target variable "Success" or "Failure." It performs only Binary splits.

Advantages of decision trees are:

a. Simple to understand and interpret.
b. Requires little data preparation.
c. The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree
d. Able to handle both numerical and categorical data.
e. Uses a white-box model.
f. Possible to validate a model using statistical tests.

Disadvantages of decision trees are:

a. Unstable nature
b. Less effective in predicting the outcome of a continuous variable.

3.5.2. RANDOM FOREST

Random forests, also known as random decision forests, are a popular ensemble method used to build predictive models for classification and regression problems. Ensemble methods use multiple learning models to gain better predictive results — in the case of a random forest; the model creates an entire forest of random uncorrelated decision trees to arrive at the best possible answer.

The same random forest algorithm or the random forest classifier can use for both classification and the regression task.

a. Random forest classifiers will handle the missing values.
b. The random forest classifier will not overfit the model when we have more trees in the forest.
c. Can model the random forest classifier for categorical values also.

How Random Forest Works?

We can understand the working of the Random Forest algorithm with the help of the following steps and Fig. 3.5.2.1:

Step 1 − First, start with selecting random samples from a given dataset.

Step 2 − Next, this algorithm will construct a decision tree for every sample. Then it will get the prediction result from every decision tree.

Step 3 − In this step, voting will be performed for every predicted result.

Step 4 − Finally, select the most voted prediction result as the final prediction result.

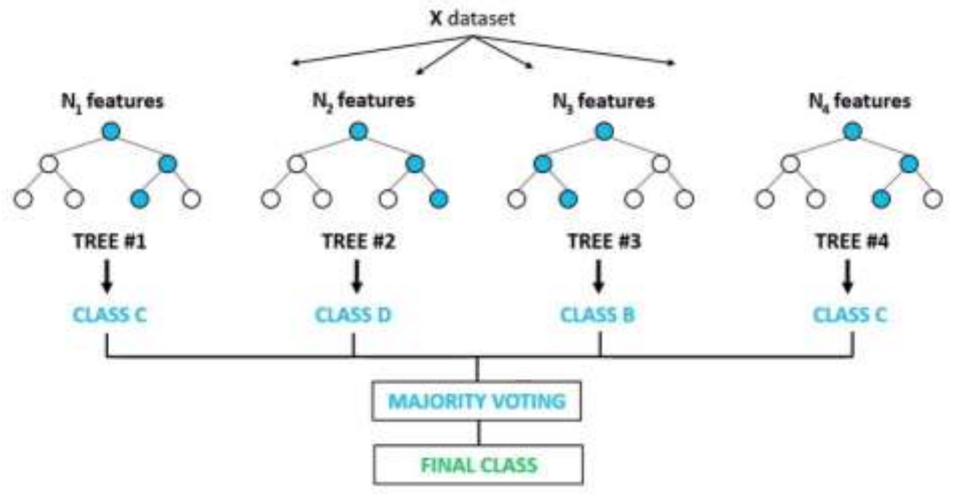


Fig. 3.5.2.1: Working of Random Forest. Source: [15]

The Mathematics Behind Random Forest

Regression Problems

When using the Random Forest Algorithm to solve regression problems, you use the mean squared error (MSE) to determine how your data branches from each node given in equation(6).

$$\mathrm{MSE} = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{N}} (\mathrm{fi} - \mathrm{yi})^2 \tag{6}$$

Where N is the number of data points
Fi is the value returned by the model and
Yi is the actual; value of data point i.

This formula calculates the distance of each node from the predicted actual value, helping to decide which branch is the better decision for your forest. Here, yi is the value of the data point you are testing at a particular node, and fi is the value returned by the decision tree.

Advantages of the random forest:

a. It provides accurate predictions for many types of applications.
b. It can measure the importance of each feature to the training data set.
c. Pairwise proximity between samples can be measured by the training data set.

Disadvantages of the random forest:

a. For data including categorical variables with the different levels, random forests are biased in favor of those attributes with more levels.
b. If the data contain groups of correlated features of similar relevance for the output, then smaller groups are favored over larger groups

3.5.3. NEAREST-NEIGHBOR METHOD

It classifies a sample based on the category of its nearest neighbor. The nearest neighbor-based classifiers use some or all the patterns available in training set to classify a test pattern. The nearest neighbor algorithm assigns the class label of its closest neighbor to a test pattern.

Let there be n training patterns, (X1, θ1), (X2, θ2), (Xn, ѳn), where Xi is of dimension d, and ѳi is the class label of the ith pattern. From equation (7), if P is the test pattern, then

$$d\ (P, \mathbf{X_k}) = \min \{d\ (P, Xi)\} \tag{7}$$

where, i = 1 ...n. Pattern P is assigned to the class θ_k, associated with X_k.

KNN (K nearest neighbor method):

a. K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on the Supervised Learning technique.
b. K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means that when new data appears, it can be easily classified into a well-suited category using the K- NN algorithm.
c. K-NN algorithm can be used for Regression and Classification, but it is mainly used for Classification problems.
d. It is also called a lazy learner algorithm because it does not learn from the training set immediately; instead, it stores the dataset and acts on it at the time of classification.
e. KNN algorithm at the training phase stores the dataset, and when it gets new data, it classifies that data into a category that is much similar to the new data.

Example: Suppose we have an image of a creature that looks similar to a cat and dog, but we want to know whether it is a cat or dog. So, we can use the KNN algorithm for this identification, as it works on a similarity measure. Our KNN model will find similar features of the new data set to the cats and dogs' images and will put it in either cat or dog category based on the most similar features.

<u>Why do we need a K-NN Algorithm?</u>

Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1 so that this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. From Fig. 3.5.3.1, with the help of K-NN, we can quickly identify the category or class of a particular dataset. Consider the below diagram:

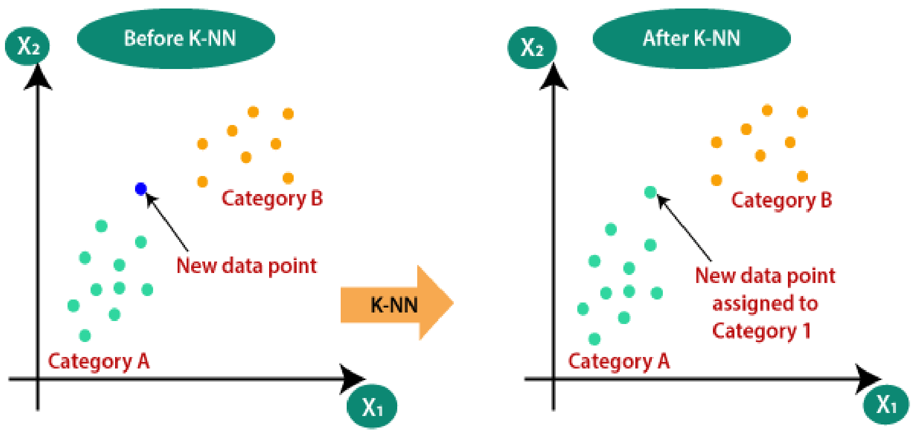


Fig. 3.5.3.1: K-NN. Source: [16]

How does K-NN work?

The K-NN working can be explained based on the below algorithm:

a) Select the number K of the neighbors
b) Calculate the Euclidean distance of K number of neighbors
c) Take the K nearest neighbors as per the calculated Euclidean distance.
d) Among these k neighbors, count the number of the data points in each category.
e) Assign the new data points to that category for which the number of neighbors is maximum.
f) Our model is ready.

Suppose we have a new data point and need to put it in the required category. Consider the below Fig. 3.5.3.2:

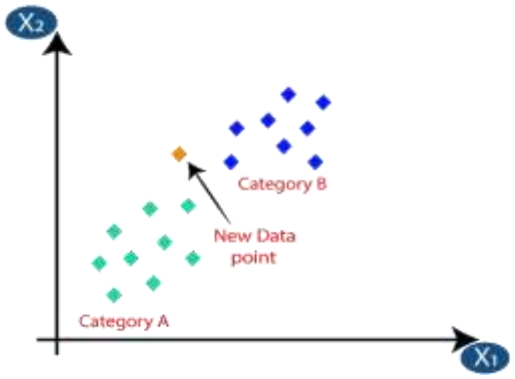


Fig. 3.5.3.2: Before K-NN. Source: [16]

a. Firstly, we will choose the number of neighbors to choose the k=5.
b. Next, we will calculate the Euclidean distance between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as given in following Fig. 3.5.3.3:

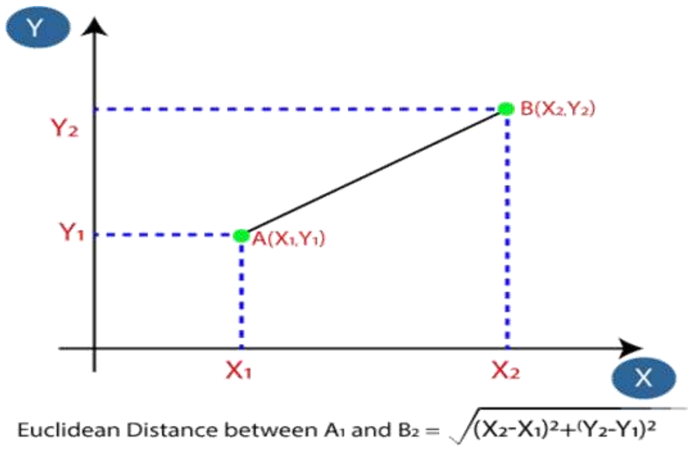

Y
Y_2
Y_1
$B(X_2,Y_2)$
$A(X_1,Y_1)$
X_1
X_2
X
Euclidean Distance between A_1 and B_2 = $\sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

Fig. 3.5.3.3: Calculating Euclidean Distance. Source: [16]

By calculating the Euclidean distance, we got the nearest neighbors as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below Fig. 3.5.4.4:

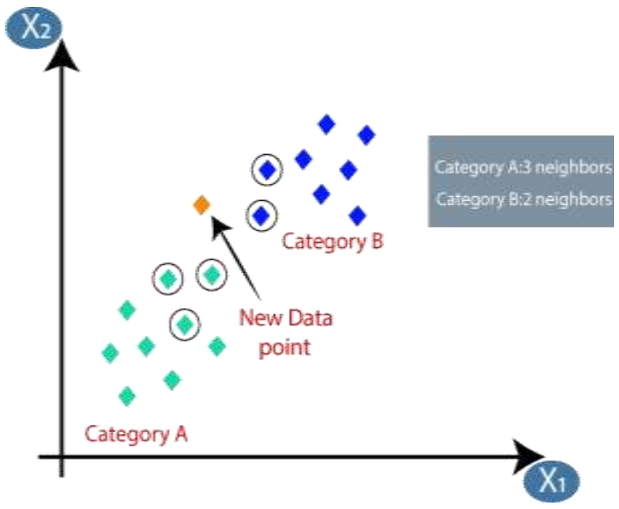


Fig. 3.5.4.4: Nearest Neighbors. Source: [16]

As we can see, the three nearest neighbors are from category A. Hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

a. There is no particular way to determine the best value for "K," so we need to try some values to find the best out. The most preferred value for K is 5.
b. A very low value for K, such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
c. Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

a. It is simple to implement.
b. It is robust to the noisy training data.
c. It can be more effective if the training data is significant.

Disadvantages of KNN Algorithm:

a. Always needs to determine the value of K, which may sometimes be complex.
b. The computation cost is high because of calculating the distance between the data points for all the training samples.

3.5.4. Support Vector Machine (SVM)

SVM is also a supervised learning algorithm that can be used for classification and regression problems. SVM tries to find an optimal hyperplane in N-dimensional space (N refers to the number of features) that help in classifying the different classes. It finds the optimal hyperplane by maximizing the margin distance between the observations of the classes using the Hinge Loss function. The dimension of the hyperplane depends upon the number of input features as shown in Fig. 3.5.4.1. If the number of features is N, then the dimension of the hyperplane is N-1.

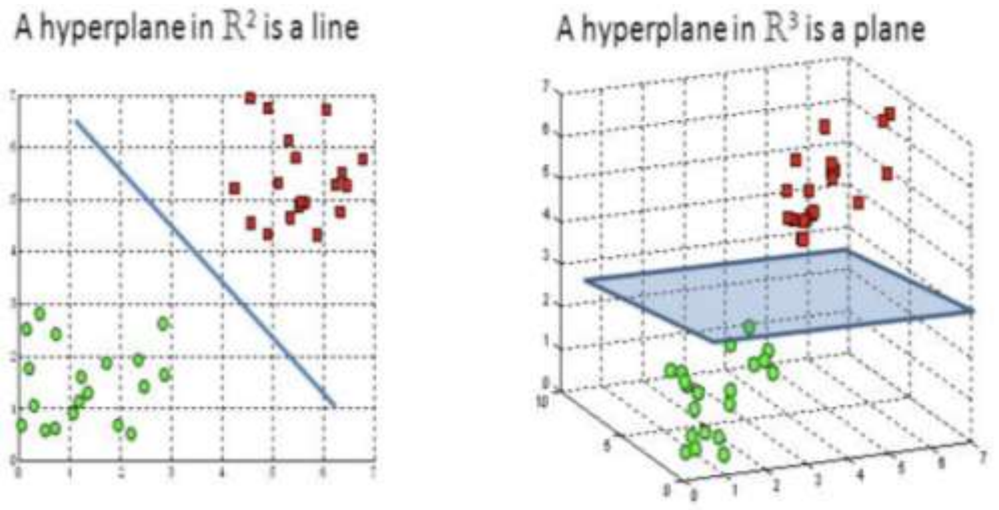


Fig. 3.5.4.1.: 2-D and 3-D Problem Space. Source: [17]

$$l(y) = \max(0, 1 + \max wyx - wtx), \tag{8}$$

Hinge Loss Function, l(y): t → target variable, w →model parameters, x → Input variable (Source: Wikipedia)

Advantages of support vector machine:

a. It works well with a clear margin of separation
b. It is effective in high-dimensional spaces.
c. It is effective in cases where the number of dimensions is greater than the number of samples.
d. It uses a subset of training points in the decision function (called support vectors), making it memory efficient.

Disadvantages of support vector machine:

a. It does not perform well when we have extensive data set because the required training time is higher
b. It also does not perform very well when the data set has more noise, i.e., target classes are overlapping

c. SVM does not directly provide probability estimates, which are calculated using an expensive five-fold cross-validation. It is included in the related SVC method of the Python scikit-learn library.

NEURAL NETWORK MODELS:

We have also applied some Deep Learning Neural Network algorithms to check the accuracy of the predicted action. Deep Learning Algorithms used are:

3.5.5. Multilayer Perceptron (MLP)

The multilayer perceptron is the type of neural network consisting of input layers, output layers, and one or more hidden layers with many neurons layered on top of each other. Neurons in a Perceptron must have an activation function that enforces a threshold, such as ReLU or sigmoid, neurons in a Multilayer Perceptron.

A multilayer perceptron is a feed-forward algorithm like the Perceptron; the inputs are combined with the initial weights in a weighted sum and then subjected to the activation function. On the other hand, each linear combination is propagated to the next layer that results in the computation that is clearly seen from Fig. 3.5.5.1 and applies to all hidden layers as well as the output layer.

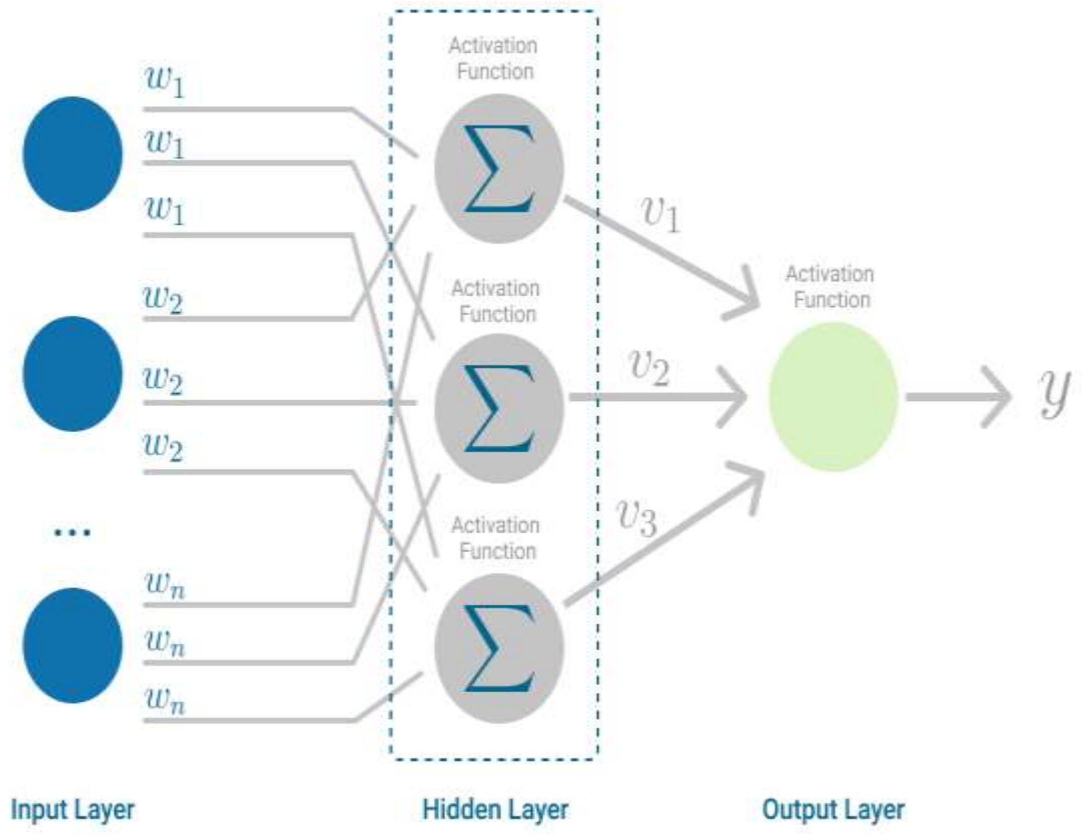


Fig. 3.5.5.1: Multi-Layer Perceptron. Source: [18]

Linear Threshold Unit (LTU)

The linear threshold unit (LTU) consists of one input x with n values, one single-value output y, and in-between mathematical operations to calculate the linear combination of the inputs and their weights (i.e., the weighted sum) to apply an activation function on that calculation. The calculations in MLP in equations (9),(10), are that there exist more layers of LTUs to combine until you reach the output y:

$$h^1 = step(z^1) = step(W^1 . x + b^1) \tag{9}$$

$$y = step\ (z^2) = step\ (W^2 . h^1 + b^2) \tag{10}$$

Input in Batches of k Instances:

ANNs are usually trained in batches of instances (an instance is one input vector x). As a result of this, k instances are drawn from the m instances available in equation(11):

$$x_1 = \begin{pmatrix} x_{1,1} \\ \cdots\cdots \\ x_{1,n} \end{pmatrix}, \ldots\ldots, x_k = \begin{pmatrix} x_{k,1} \\ \cdots\cdots \\ x_{k,n} \end{pmatrix} \tag{11}$$

The k instances can be combined to equation(12):

$$\mathbf{X} = \begin{pmatrix} x_1^T \\ \cdots\cdots \\ x_k^T \end{pmatrix} = \begin{pmatrix} x_{1,1} \ldots\ldots\ldots\ldots x_{1,n}. \\ x_{k,1} \ldots\ldots\ldots\ldots x_{k,n}. \end{pmatrix} \tag{12}$$

By that, the equation(10) to calculate y changes to equation(13):

$$y = \text{step (Z)} = \text{step (X. W+b)} \tag{13}$$

The input X is now a matrix of shape (k, n), where k = number of instances per batch and n = number of input values.

W is a matrix, but the shape changed to (n, u) (W is just the transpose of itself = W^T).

The bias vector b is of shape (u, 1), and the output y changed to a shape (k, u) matrix.

Hereby the heaviside step function is replaced by one of the following activations:

a. Logistic Function (Sigmoid)
b. Rectifying Linear Unit ReLU
c. Hyperbolic Tangent tanh

The activation function at the output layer depends on the task to be solved by the ANN:

a. classification tasks: Softmax activation function
b. regression tasks: no activation function

3.5.6. Long Short-term Memory (LSTM)

Long Short-term memory is the type of the recurrent neural networks; classified under the category of artificial neural network, that has an ability to learn or predict the problem using sequence. It contains four neural networks and cells called memory blocks. The memory block's purpose is to keep its state over time while regulating information flow through non-linear gating units.

The benefit of using LSTMs for sequence classification is that they can learn from the raw time series data directly, and in turn do not require domain expertise to manually engineer input features. The model can learn an internal representation of the time series data and ideally achieve comparable performance to models fit .

The LSTM cell contains the following components

a. Forget Gate “f” (a neural network with sigmoid)
b. Candidate layer “C`"(a NN with Tanh)
c. Input Gate “I” (a NN with sigmoid)
d. Output Gate “O”(a NN with sigmoid)
e. Hidden state “H” (a vector)
f. Memory state “C” (a vector)

Why do we need LSTM if we have RNN?

LSTM can be used to solve problems faced by the RNN model. So, it can be used to solve:

a. Long term dependency problem in RNNs.
b. Vanishing Gradient & Exploding Gradient.

The heart of a LSTM network is it’s cell or say cell state which provides a bit of memory to the LSTM so it can remember the past i.e The cell state may remember the gender of the subject in a given input sequence so that the proper pronoun or verb can be used.

In LSTM we will have 3 gates:

a. Input Gate.
b. Forget Gate.
c. Output Gate.

Gates in LSTM are the sigmoid activation functions i.e they output a value between 0 or 1 and in most of the cases it is either 0 or 1 as shown in Fig. 3.5.6.1.

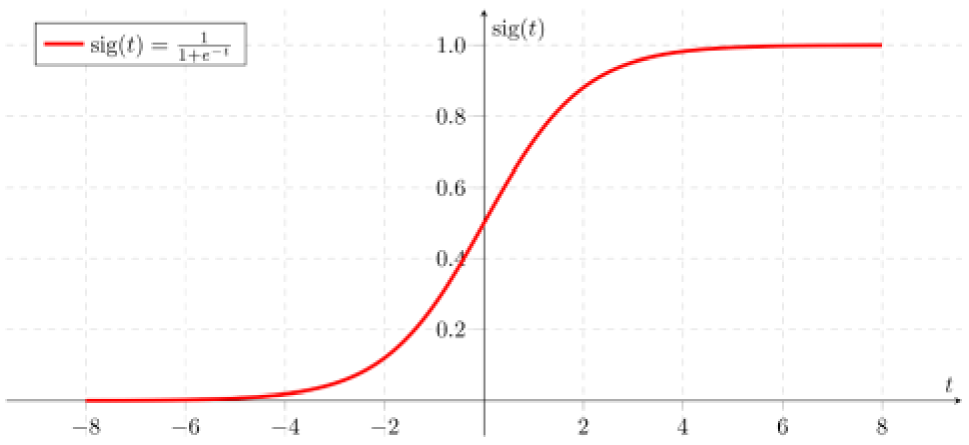


Fig. 3.5.6.1: Sigmoid Function. Source: [19]

We use sigmoid function for gates because, we want a gate to give only positive values and should be able to give us a clear cut answer whether, we need to keep a particular feature or we need to discard that feature.

a. “0” means the gates are blocking everything.
b. “1” means gates are allowing everything to pass through it.

The equation (14),(15), and (16) for the gates in LSTM are:

$$i_t = \sigma(\omega_i[h_{t-1}, x_t] + b_i) \tag{14}$$

$$f_t = \sigma\left(\omega_f[h_{t-1}, x_t] + b_f\right) \tag{15}$$

$$o_t = \sigma(\omega_o[h_{t-1}, x_t] + b_o) \tag{16}$$

i_t – represents input gate
f_t – represents for get gate
o_t – represents output gate
σ – respresents signmoid function
ω_x – weight for the respective gate(x) neurons
h_{t-1} – output of the previous lstm block (at timestamp t -1)
x_t – input at current timestamp
b_x –biases for the respective gates (x)

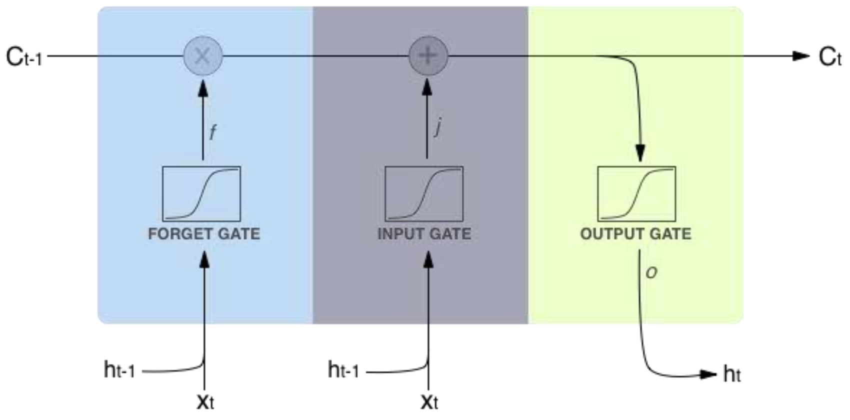


Fig. 3.5.6.2: LSTM Memory Cell. Source: [19]

First equation (17) is for Input Gate which tells us that what new information we're going to store in the cell state(that we will see below).

Second (18) is for the forget gate which tells the information to throw away from the cell state.

Third one (19) is for the output gate which is used to provide the activation to the final output of the lstm block at timestamp 't'.

The equations for the cell state, candidate cell state and the final output:

$$\tilde{c}_t = \tanh\,(\omega_c[h_{t-1}, x_t] + b_c \qquad (17)$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t \qquad (18)$$

$$h_t = o_t * \tanh(c^t) \qquad (19)$$

c_t – cell state (memory) at timestamp(t)
$\tilde{c}_t$ – represents candia=date for cell state at timestamp(t)

To get the memory vector for the current timestamp (c_{t}) the candidate is calculated.

Now, from the above equation we can see that at any timestamp, our cell state knows that what it needs to forget from the previous state(i.e f_{t} * c_{t-1}) and what it needs to consider from the current timestamp (i.e i_{t} * c`_{t}).

Lastly, we filter the cell state and then it is passed through the activation function which predicts what portion should appear as the output of current lstm unit at timestamp t. We can pass this h_{t} the output from current lstm block through the softmax layer to get the predicted output(y_{t}) from the current block.

Let's look at a block of lstm at any timestamp {t} in Fig. 3.5.6.3.

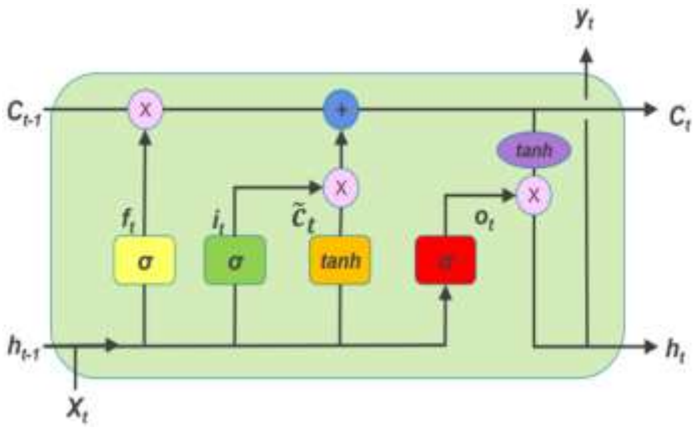


Fig. 3.5.6.3: LSTM Block. Source: [19]

3.6. Results

To explore the impact of the presence of feedback on human performance and the improvement in human performance over time, we used machine learning algorithm to find the accuracy of prediction.

We studied the problem of human activity reorganization and used one recurrent neural network approach, multilayer perceptron, and three deep learning method approaches, Long-Short-Term-Memory, CNN-LSTM, and Conv. LSTM. Both networks and the combination are most suitable for extracting features from raw sensor data and predicting action. Neural Networks are forecasting methods that are based on predictions from the user data. It finds extensive applications in an area where the traditional computer does not go far too well.

In this, machine learning architectures based on both supervised learning and unsupervised learning techniques were used. Decision tree, Random Forest Classifier, Support Vector Machine, and Gaussian Classifier are the models that were trained using the processed training set data and testing set data in the Unity 3D environment using the PCA algorithm. The model completed by training is used to predict the test data, and the real value is evaluated by comparing it to the predicted value to calculate the model's accuracy.

We used the grid search algorithm to select the most accurate hyperparameters. The grid search results for the four models are shown in Table 1

Model	Criterion	Max_depth	Min_sample_split	Accuracy
Decision Tree	Gini	10	2	52.35
Decision Tree	Entropy	5	2	53.27
Decision Tree	Entropy	6	6	60.98

Table 3.5.1: Accuracy of Decision Tree

Model	criterion	n_estimators	Random_state	Accuracy
Random Forest Classifier	Gini	10	1	54.43
Random Forest Classifier	Gini	50	0	59.43
Random Forest Classifier	Gini	256	6	61.28

Table 3.5.2: Accuracy of Random Forest Classifier

Model	kernel	iterations	Gamma	Accuracy
Support Vector Machine	Linear	1000	auto	43.90
Support Vector Machine	Linear	2000	auto	48.20
Support Vector Machine	RBF	1000	auto	45.20

Table 3.5.3: Accuracy of Support Vector Machine

Model	kernel	cv	Accuracy
Gaussian Classifier	default	1	59.80
Gaussian Classifier	1.0 * RBF(1.0)	2	59.80

Table 3.5.4: Accuracy of Gaussian Classifier

Random forest classifier model shown in Table 3.5.2 shows the best performance with criterion 'gini' at random state to be zero with n estimators to be 256. Table 3.5.1, Table 3.5.2, Table 3.5.3, and Table 3.5.4 give us different accuracy of the different model at different hyperparameters. The criterion and random state in Table 3.5.2 indicate the accuracy of predicting the actions of a bot to walk in the left, right, and front direction; random forest classifier accuracy is 61.28%, which is higher in comparison with the other three models. Besides, the results also indicate that the model based on a random forest classifier has classified actions more accurately when the 'gini' criterion is used instead of entropy.

1.2. Feature Extraction

After processing the dataset using feature extraction, we use four neural network models, i.e., one RNN and three CNN models on the dataset obtained by playing the game in the Unity 3D environment that is processed by using 1000 features.

Models	Nearest neighbors	algorithm	Leaf size	Accuracy
KNN	13	auto	5	
KNN	15	ball tree	15	
KNN	49	auto	10	60.60
KNN using SMOTE	49	auto	10	61.43

Table 3.5.5: Accuracy using KNN model

Models	Hidden layers size	Activation function	Random State	Accuracy
MLP	64,64	tanh	1	59.895
MLP	128,128	Relu	1	59.881
MLP	128,128	Relu	0	59.896
MLP	10,30,10	tanh	0	60.017

Table 3.5.6: Accuracy using MLP model

Models		Activation function	Accuracy
LSTM	50	relu	59.90
LSTM	100	Relu	60.02

Table 3.5.7: Accuracy using LSTM model

It can be seen from the table that the accuracy of the LSTM classification model in Table 3.5.7 with Adam optimization is larger than the other models under all datasets. It was quite close to the multilayer perceptron model approach, which had an accuracy of 60.01%. Quick observations were noted that there is no change in accuracy after hyper parameter tuning before and after applying smote analysis on the dataset on the KNN model as shown in Table 3.5.5.

CHAPTER – IV

FINDINGS

After completing Machine Learning training, we were able to:

- **UNDERSTAND:**
    - Acknowledge the concept of machine learning,
    - Acknowledge the importance of machine learning in real life.
- **APPLY:**
  After understanding all of the fundamentals of machine learning and how they can be applied to various machine learning models in various datasets, all set to proceed to the next step.
- **ANALYZE:**
  Analyzing the best machine learning models for various datasets and determining the best-fit model for each dataset.
- **RESULTANT:**
  As a result, we were able to gain a thorough understanding of Machine Learning through lectures and assignments.
- **RESEARCH:**
  We worked on drafting a research paper and submitted it in the conference.

CHAPTER-V

CONCLUSION

In this experiment we studied the problem of human activity reorganization and used one recurrent neural network approach, multilayer perceptron, and three deep learning method approaches, Long-Short-Term-Memory, CNN-LSTM, and Conv. LSTM. Both networks as well as the combination of two are most suitable to extractions features from raw sensor data and predicting action. Neural Networks are forecasting methods that are based on predictions from the user data. It finds extensive applications in area where traditional computer doesn’t go far too well. Problem statement where system learns adapts and changes the results on its own according to the data that we are providing it, rather using pre-programmed outputs. Let us take an example of chess players in the game utilize chess engines to analyze their games, improve their skills, and try new approaches- and it continues where it uses neural network to do so. The search and retrieve environment created using the software Unity3D was made to replicate real world experience, which helps us in concluding that even though the results have been obtained using a simulation in a laboratory, the result would hold true for real world scenarios as well.

This research has prompted lots of new ideas that can be tested in the future. First, based on their diversity and level of required workload, similar simulations can be developed for different cognitive demand tasks. Second, computational cognitive models (such as instance based learning models and machine learning models (such as Soft-Actor Critic and Proximal Policy Optimization can be developed to account for human choices in such tasks. Third, multi agent simulations of such games can be created to see how humans perform in groups with other humans or robots in similar search-and-retrieve tasks. Furthermore, an LSTM-based Recurrent Neural Network with Time Series Classification for human activity recognition can improve the model's accuracy.

REFERENCES

[1] Schulman, S. Levine, P. Moritz, M. I. Jordan, en P. Abbeel, “Trust Region Policy Optimization”, CoRR, vol abs/1502.05477, 2015.

[2] V. Mnih et al., “Asynchronous Methods for Deep Reinforcement Learning”, CoRR, vol abs/1602.01783, 2016.

[3] T. P. Lillicrap et al., “Continuous control with deep reinforcement learning”, arXiv [cs.LG]. 2019

[4] Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

[5] Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off- policy maximum entropy deep reinforcement learning with a stochastic actor." In International conference on machine learning, pp. 1861-1870. PMLR, 2018.

[6] (Anon, 2022)

[7] (Xavier, 2019)

[8] (Brownlee, 2018) (Rani, 2019)

[9] (Mittal, 2017)

[10] ("Introduction | Text classification guide | Google Developers", 2021)

[11] (Kumar, 2018)

[12] ("Binary entropy function - Wikipedia", 2022)

[13] (Chauhan, 2022)

[14] (David, 2020)

[15] ("K-Nearest Neighbor(KNN) Algorithm for Machine Learning - Javatpoint", 2022)

[16] (Gandhi, 2018)

[17] (Bento, 2021)

[18] (Thakur, 2018)

APPENDIX

Assignments Coding:

Assignment- 1

```
## Convert training files from .json extension to .xlsx extension:

In[1]: import json

import pandas as pd

# read json file

In[2]: with open(r'C:\Users\user\Downloads\training.json') as json_file:

        data = json.load(json_file)

data = pd.DataFrame(data['allGameData'])

In[3]: data.to_csv("training.csv",index=False)

# export data in xlsx

In[4]: data.to_excel(r"C:\Users\user\Downloads\training.xlsx",index=False)
```

Assignment- 2

```
## Convert testing files from .json extension to .xlsx extension:

In[1]: import json

import pandas as pd

# read json file

In[2]: with open(r'C:\Users\user\Downloads\testing.json') as json_file:

         data = json.load(json_file)

data = pd.DataFrame(data['allGameData'])

In[3]: data.to_csv("testing.csv",index=False)

# export data in xlsx
```

```
In[4]: data.to_excel(r"C:\Users\user\Downloads\testing.xlsx",index=False)
```

Assignment -3

```
## Convert training files from .xlsx extension to .csv extension::

In[1]: import pandas as pd

import glob

# assume the path

In[2]: excel_files = glob.glob(r'C:\Users\user\Downloads\TRAIN DATA\*.xlsx')

for excel in excel_files:

        out = excel.split('.')[0]+'.csv'

        df = pd.read_excel(excel)

        df.to_csv(out)
```

Assignment- 4

```
## Convert testing files from .xlsx extension to .csv extension:

In[1]: import pandas as pd

import glob

# assume the path

In[2]: excel_files = glob.glob(r'C:\Users\user\Downloads\TEST DATA\*.xlsx')

for excel in excel_files:

        out = excel.split('.')[0]+'.csv'

        df = pd.read_excel(excel)

        df.to_csv(out)
```

Assignment- 5

## Apply PCA algorithm on separate file of 50 user’s data

```
In[1]: from google.colab import drive

drive.mount('/content/drive')

In[2]: import numpy as np

import pandas as pd

import json

import csv

import os

In[3]: base_address = "/content/drive/MyDrive/1"

folders_list = os.listdir(base_address)

for folder_name in folders_list:

for type_id in ["testing","training"]:

# converting json to csv

In[4]: with open(base_address + "/" + type_id +".json") as json_file:

data = json.load(json_file)

game_data = data['allGameData']

csv_name = folder_name + "_" + type_id

data_file = open(csv_name+".csv", 'w')

csv_writer = csv.writer(data_file)

count = 0

for data in game_data:

if count == 0:

header = data.keys()

csv_writer.writerow(header)
```

```
count += 1

csv_writer.writerow(data.values())

data_file.close()

# initial csv to modify

In[5]: dataset = pd.read_csv(csv_name+".csv")

name = csv_name

if type_id == "training":

video_name = "training"

else:

video_name = "test"

# giving video path

In[6]: video_file_path = base_address + "/" + folder_name + "/" + video_name +
".MP4"

In[7]: with open("/content/drive/MyDrive/1/testing.json") as json_file:

data = json.load(json_file)

game_data = data['allGameData']

csv_name = 'testing_csv'

data_file = open(csv_name+".csv", 'w')

csv_writer = csv.writer(data_file)

count = 0

for data in game_data:

   if count == 0:

       header = data.keys()

      csv_writer.writerow(header)

    count += 1
    csv_writer.writerow(data.values())

data_file.close()

# initial csv to modify

In[8]: dataset = pd.read_csv(csv_name+".csv")

name = csv_name

# csv for target distractor info

In[9]: objects_dataset = pd.read_csv("data analysis - Sheet4.csv")

video_file_path = "/content/drive/MyDrive/1/test.MP4"

In[10]: def split_xyz(column,dataset):

            print(column)

# global dataset

In[11]:                    dataset[column]=                    dataset[column].apply(lambda
x:pd.Series(str(x).replace("(","")))

dataset[column]= dataset[column].apply(lambda x:pd.Series(x.replace(")","")))

new = dataset[column].apply(lambda x: pd.Series(x.split(',')))

new.rename(columns = {0: 'x',1:'y',2:'z'}, inplace = True)

dataset = pd.concat([dataset, new], axis=1, join='inner')

return dataset

def split_time(column,dataset):

print(column)

# global dataset

In[12]:                    dataset[column]=                    dataset[column].apply(lambda
x:pd.Series(str(x).replace(":",",")))

    dataset[column]= dataset[column].apply(lambda x:pd.Series(x.replace(".",",")))

    new = dataset[column].apply(lambda x: pd.Series(x.split(',')))
    new.rename(columns = {0: 'Minutes',1:'Second',2:'Miliseconds'}, inplace = True)

In[13]: new['TotalTime']=
(float(new['Minutes'])*60*1000)+(float(new['Second'])*1000)+float(new['Miliseconds'
])

dataset = pd.concat([dataset, new], axis=1, join='inner')

return dataset

In[14]: data_final = dataset.iloc[:,3:7]

data_final = dataset

In[15]: data_final.rename(columns = {'keypress':'Actions', 'timeStamp':'time',

                          'position':'coordinates'}, inplace = True)

In[16]: data_final = split_xyz('coordinates',data_final)

data_final.drop('coordinates',inplace=True,axis = 1)

In[17]: data_final = split_time('time',data_final)

In[18]: def cvt_int(dataset,column):

dataset[column]= dataset[column].apply(lambda x:float(x))

return dataset

In[19]: data_final = cvt_int(data_final,'Minutes')

data_final = cvt_int(data_final,'Second')

data_final = cvt_int(data_final,'Miliseconds')

data_final = cvt_int(data_final,'x')

data_final = cvt_int(data_final,'y')

data_final = cvt_int(data_final,'z')

data_final

In[20]: data_final['Total Time'] = data_final.apply(lambda row:
(row.Minutes*60*1000) + (row.Second*1000) + row.Miliseconds, axis=1)

In[21]: print(data_final[data_final['Total Time'].isnull()])
```

```
data_final = data_final.dropna(subset=['Total Time'])

data_final['Total Time'] = data_final['Total Time'].apply(lambda x:int(x))

# filling the rewards column

In[22]: data_final = data_final.fillna(method='ffill')

In[23]: data_final.drop('time',inplace=True,axis = 1)

data_final

In[24]: time = data_final["Total Time"]

In[25]: if "training" in name:

            type = "training"

elif "testing" in name:

  type = "testing"

data_final['Type']=type

In[26]: id = dataset["uid"].values[0]

In[27]: data_final['Id']=id

In[28]: arrange according to total time

data_final = data_final.sort_values('Total Time')

data_final.reset_index(inplace = True)

data_final.drop('index',inplace=True,axis = 1)

In[29]: data_final.columns

In[30]: data_final["Actions"]

data_final["Actions"].replace('W',0, inplace=True)

data_final["Actions"].replace('A',1, inplace=True)

data_final["Actions"].replace('D',2, inplace=True)

In[31]: final_cols = ['Actions','totalRewards','x','y','z']

# code for taking screenshots
```

```
In[32]: import cv2
import os
cap = cv2.VideoCapture(video_file_path)
frame_timings = []
frame_no = 0
try:
    # creating a folder named data
    if not os.path.exists('image_data_'+id):
        os.makedirs('image_data_'+id)
except OSError:
    print('Error: Creating directory of data')
t = list(data_final["Total Time"])
cursor = 0
while(cap.isOpened()):
    frame_exists, curr_frame = cap.read()
    if (cursor >=dataset.shape[0] ):
        break
    if frame_exists:
        frame_timings.append(cap.get(cv2.CAP_PROP_POS_MSEC))
        for j in frame_timings:
            if t[cursor]<j:
                cursor = cursor+1
                name = './image_data_'+id+'/' + str(cursor) + '.jpg'
                cv2.imwrite(name, curr_frame)
                print(cursor)
            break
    else:
        break
    frame_no += 1
cap.release()
In[33]: frame_no
# saving The PCA vector as separate columns and combined as well
In[34]: images = []
imageNameList=os.listdir('image_data_'+id)
In[35]: for i in imageNameList:
    # print(i)
    # /content/image_data_2aa46b31-0c0d-48a3-ab28-1cc8ef1de65a/118.jpg
    Image=cv2.resize(Image,(int(Image.shape[1]/7.5),int(Image.shape[0]/7.5)))
    images.append(Image)
In[36]: images=np.array(images)
In[37]: images_dim = images.flatten().reshape(len(data_final),64*96*3)
In[38]: import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
pca = PCA(n_components=600)
pca.fit(images_dim)
plt.grid()
plt.plot(np.cumsum(pca.explained_variance_ratio_ * 100))
plt.xlabel('Number of components')
plt.ylabel('Explained variance')
In[39]: def perform_pca(n,i):
```

```
pca = PCA(n_components=n)

images_dim_red = pca.fit(images_dim)

images_dim_recovered = pca.transform(images_dim_red)

   print("Cumulative variance for n=",n," = ",pca.explained_variance_ratio_.cumsum())

In[40]: print(np.cumsum(pca.explained_variance_ratio_ * 100)[-1])

   image_pca = images_dim_recovered[i,:].reshape([64,96,3])

   plt.imshow(image_pca.astype('uint8'))

   plt.title('Compressed image with '+str(n)+' components', fontsize=15, pad=15)

   return images_dim_red

In[41]: red_150=perform_pca(150,112)

red_150=np.array(red_150)

df_150 = pd.DataFrame(red_150)

df_150['PCA(n=150)']= df_150.values.tolist()

In[42]: data_final = pd.concat([data_final, df_150], axis=1, join='inner')

In[43]: data_backup = data_final

data_final.drop(['uid',          'gameMode',         'targetName',

         'targetPosition',        'targetCount',     'disctractorName',

     'disctractorPosition',      'disctractorCount', 'distanceCovered', 'sensitivity','Minutes',
'Second', 'Miliseconds', 'Total Time', 'Type' ],inplace=True,axis = 1)

In[44]: data_final_save = data_final[["Actions","totalReward","x","y","z"]]

data_final_save = pd.concat([data_final_save,data_final.iloc[:,5:]], axis=1, join='inner')

data_final_save = data_final_save.iloc[:,:-1]

data_final_save

In[45]: data_final_save.to_csv('combined.csv',index=False)
```

```
data_final_save.to_csv('/content/drive/MyDrive/Bhavik_backup DATA COLLECT json, imitation/Sakshi Sharma/449fa4ad-c5c6-4feb-80be-beec13384d37.csv',index=False)
```

Assignment- 6

## Apply PCA analysis on combined file of 50 users data

```
In[1]: import numpy as np

import pandas as pd

import json

import csv

import os

from sklearn.decomposition import PCA

In[2]: def split_xyz(column,dataset):

   print(column)

   # global dataset

   dataset[column]= dataset[column].apply(lambda x:pd.Series(str(x).replace("(","")))

   dataset[column]= dataset[column].apply(lambda x:pd.Series(x.replace(")","")))

   new = dataset[column].apply(lambda x: pd.Series(x.split(',')))

   new.rename(columns = {0: 'x',1:'y',2:'z'}, inplace = True)

   dataset = pd.concat([dataset, new], axis=1, join='inner')

   return dataset

def split_time(column,dataset):

   print(column)

   # global dataset

   dataset[column]= dataset[column].apply(lambda x:pd.Series(str(x).replace(":",",")))
```

```
    dataset[column]= dataset[column].apply(lambda x:pd.Series(x.replace(".",",")))

    new = dataset[column].apply(lambda x: pd.Series(x.split(',')))

    new.rename(columns = {0: 'Minutes',1:'Second',2:'Miliseconds'}, inplace = True)

    # new['Total Time'] =
(float(new['Minutes'])*60*1000)+(float(new['Second'])*1000)+float(new['Miliseconds'
])

    dataset = pd.concat([dataset, new], axis=1, join='inner')

    return dataset

In[3]: def cvt_int(dataset,column):

  dataset[column]= dataset[column].apply(lambda x:float(x))

  return dataset

In[4]: def perform_pca(n):

pca = PCA(n_components=n)

images_dim_red = pca.fit(images_dim)

images_dim_recovered = pca.inverse(images_dim_red)

    #         print("Cumulative         variance         for         n=",n,"         =
",pca.explained_variance_ratio_.cumsum())

#    print(np.cumsum(pca.explained_variance_ratio_ * 100)[-1])

    # image_pca = images_dim_recovered[i,:].reshape([64,96,3])

    # plt.imshow(image_pca.astype('uint8'))

    # plt.title('Compressed image with '+str(n)+' components', fontsize=15, pad=15)

    return images_dim_red

In[5]: def perform_pca2(n):

pca = PCA(n_components=n)

images_dim_red = pca.fit_transform(images_dim)

images_dim_recovered = pca.inverse_transform(images_dim_red)
```

```
# print("Cumulative variance for n=",n," = ",pca.explained_variance_ratio_.cumsum())
#     print(np.cumsum(pca.explained_variance_ratio_ * 100)[-1])
# image_pca = images_dim_recovered[i,:].reshape([64,96,3])
# plt.imshow(image_pca.astype('uint8'))
# plt.title('Compressed image with '+str(n)+' components', fontsize=15, pad=15)
        return images_dim_red,pca
In[6]: from google.colab import drive
drive.mount('/content/drive')
In[7]: base_address = "/content/drive/MyDrive/1"
folders_list = os.listdir(base_address)
save_address = "/content/drive/MyDrive/Untitled folder"
for folder_name in folders_list:
            for type_id in ["testing","training"]:
             try:
               # converting json to csv
             with open(base_address + "/" + type_id +".json") as json_file:
                 data = json.load(json_file)
                game_data = data['allGameData']
            csv_name = folder_name + "_" + type_id
                data_file = open(csv_name+".csv", 'w')
                csv_writer = csv.writer(data_file)
                 count = 0
      for data in game_data:
                 if count == 0:
                              header = data.keys()
                 csv_writer.writerow(header)
               count += 1
                  csv_writer.writerow(data.values())
               data_file.close()
      # initial csv to modify
             dataset = pd.read_csv(csv_name+".csv")
                name = csv_name
               if type_id == "training":
            video_name = "training"
                else:
             video_name = "test"
      # giving video path
               video_file_path = base_address + "/" + folder_name + "/" + video_name +
".MP4"
                 data_final = dataset
               data_final.rename(columns = {'keypress':'Actions', 'timeStamp':'time',
                         'position':'coordinates'}, inplace = True)
               #splitting xyz and time
      data_final = split_xyz('coordinates',data_final)
      data_final.drop('coordinates',inplace=True,axis = 1)
      data_final = split_time('time',data_final)
      #convert to integer values
      data_final = cvt_int(data_final,'Minutes')
      data_final = cvt_int(data_final,'Second')
      data_final = cvt_int(data_final,'Miliseconds')
      data_final = cvt_int(data_final,'x')

      data_final = cvt_int(data_final,'y')

      data_final = cvt_int(data_final,'z')

      # data_final

                 # adding total time column to dataset

               data_final['Total Time'] = data_final.apply(lambda row:
(row.Minutes*60*1000) + (row.Second*1000) + row.Miliseconds, axis=1)

               data_final = data_final.dropna(subset=['Total Time'])

               data_final['Total Time'] = data_final['Total Time'].apply(lambda x:int(x))

               data_final = data_final.fillna(method='ffill')

                  #dropping time column

               data_final.drop('time',inplace=True,axis = 1)

      # getting type

      if "training" in name:

        type = "training"

      elif "testing" in name:

        type = "testing"

                  data_final['Type']=type

                 # getting id

               id = dataset["uid"].values[0]

               #arrange according to total time

               data_final = data_final.sort_values('Total Time')

               data_final.reset_index(inplace = True)

               data_final.drop('index',inplace=True,axis = 1)

               # replacing actions column
```

```
data_final["Actions"].replace('W',0, inplace=True)

data_final["Actions"].replace('A',1, inplace=True)

data_final["Actions"].replace('D',2, inplace=True)

 # storing unique time stamp rows

data_final = data_final.drop_duplicates(subset = ["Total Time"], keep="last")

# code for taking screenshots

import cv2

import os

cap = cv2.VideoCapture(video_file_path)

frame_timings = []

frame_no = 0

try:

 # creating a folder named data

if not os.path.exists('image_data_'+ csv_name):

            os.makedirs('image_data_'+ csv_name)

  except OSError:

  print('Error: Creating directory of data')

              t = list(data_final["Total Time"])

  cursor = 0

 while(cap.isOpened()):

  frame_exists, curr_frame = cap.read()

            if (cursor >= data_final.shape[0] ):

   break

               if frame_exists:

            frame_timings.append(cap.get(cv2.CAP_PROP_POS_MSEC))
```

```
          for j in frame_timings:
            if t[cursor]<j:
                   cursor = cursor+1
                name = './image_data_'+csv_name+'/' + str(cursor) + '.jpg'
                 cv2.imwrite(name, curr_frame)
                   # print(cursor)
                    break
        else:
                  break
        frame_no += 1
    cap.release()
    print("screenshots taken")
# saving The PCA vector as seperate columns and combined as well
    images = []
    imageNameList=os.listdir('image_data_'+csv_name)
    for i in imageNameList:
      Image=cv2.imread('./image_data_'+csv_name+'/'+i)
   #Image=cv2.resize(Image,(int(Image.shape[1]/7.5),int(Image.shape[0]/7.5)))
      Image=cv2.resize(Image,(64,96))
      # Image = np
      images.append(Image)
       print(len(imageNameList), len(data_final))
    images=np.array(images)
    images_dim = images.flatten().reshape(len(imageNameList),64*96*3)
    # x = min(150, len(imageNameList))
```

```
      red_150=perform_pca(150)

      red_150=np.array(red_150)

      df_150 = pd.DataFrame(red_150)

      df_150['PCA(n=150)']= df_150.values.tolist()

      data_final = pd.concat([data_final, df_150], axis=1, join='inner')

      data_backup = data_final

      data_final.drop(['uid',           'gameMode',         'targetName',

            'targetPosition',        'targetCount',     'disctractorName',

        'disctractorPosition',     'disctractorCount', 'distanceCovered', 'sensitivity','Minutes',
'Second', 'Miliseconds', 'Total Time', 'Type' ],inplace=True,axis = 1)

      data_final_save = data_final[["Actions","totalReward","x","y","z"]]

      data_final_save = pd.concat([data_final_save,data_final.iloc[:,5:]], axis=1,
join='inner')

      data_final_save = data_final_save.iloc[:,:-1]

      data_final_save

      data_final_save.to_csv(save_address + csv_name+"_"+'combined.csv',index=False,
header=False )

      print(csv_name)

      print("---------------------------")

    except:

      print(csv_name)

      print("NOT DONE")

      print("---------------------------")

      pass
```

Assignment- 7

```
## Apply Feature extraction on 50 user's data

In[1]: from google.colab import drive

drive.mount('/content/drive')

import numpy as np

import pandas as pd

import json

import csv

import os

import cv2

from tqdm import tqdm

In[2]: from tensorflow.keras.applications.resnet import ResNet50

res_model = ResNet50(weights='imagenet', include_top=True, pooling='max', input_shape=(224, 224, 3)) #resnet for feature extraction with max pooling

for layer in res_model.layers:

    layer.trainable = False

# Helper Functions

def get_mseconds(time):

  min= int(time[:2])

  sec= int(time[3:5])

  msec= int(time[6:])



  return (60000*min)+(1000*sec)+(msec)

In[3]: def select_nearest_frame(frame_timings,time):

            return min(frame_timings, key=lambda x:abs(x-time))

# defining Directories
```

```
In[4]: base_address= "/content/drive/MyDrive/Cair Data collected" #base address of the data folder in the drive

In[5]: dir_list = os.listdir(base_address)

user_address = dir_list[9] #defining the focus directory

In[6]: user_training_path=  base_address+"/"+user_address+"/"+"training.xlsx" #path of training data

user_testing_path=  base_address+"/"+user_address+"/"+"testing.xlsx" #path of test data

user_training_video_path=  base_address+"/"+user_address+"/"+"training.MP4" #path of training video

user_testing_video_path=  base_address+"/"+user_address+"/"+"test.MP4"

# path of test video

training_data= pd.read_excel(user_training_path) #reading training data

testing_data= pd.read_excel(user_testing_path) #reading testing data

# reading user id

uid= training_data["uid"][0]

print("------------------USER ID : {}------------------".format(uid))

train_timestamps= training_data["timeStamp"] #training timestamps

test_timestamps= testing_data["timeStamp"] #testing timestamps

train_timestamps= [get_mseconds(x) for x in train_timestamps] #train timestamps in miliseconds

test_timestamps= [get_mseconds(x) for x in test_timestamps] #test timestamps in miliseconds

# training video frames- selection of nearest frames

cap = cv2.VideoCapture(user_training_video_path)

# find the number of frames

video_length = int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) - 1
```

```
frame_timings=[]
frame_exists= 1
count=0
while cap.isOpened():
  # extract the frame
  ret, frame = cap.read()
  if not ret:
      continue
  time= cap.get(cv2.CAP_PROP_POS_MSEC)
  frame_timings.append(time)
  count = count + 1
  # if there are no more frames left
  if (count > (video_length-1)):
      # release the feed
      cap.release()
      # print stats
      break
  # extraction training frames
selected_frames= [select_nearest_frame(frame_timings,x) for x in train_timestamps]
print("Selecting Nearest Train Frames----- Done")
#training video frames- selection of nearest frames
cap = cv2.VideoCapture(user_training_video_path)
# find the number of frames
video_length = int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) - 1
frames=[]
```

```
frame_exists= 1
count=0
while cap.isOpened():
  # extract the frame
  ret, frame = cap.read()
  if not ret:
      continue
    time= cap.get(cv2.CAP_PROP_POS_MSEC)
  if time in selected_frames:
    occurrences= selected_frames.count(time)
    for j in range(occurrences):
      frames.append(frame)
  count = count + 1
  # if there are no more frames left
  if (count > (video_length-1)):
      # release the feed
      print("Extracting Nearest Train Frames----- Done")
      cap.release()
      # print stats
      break
  # feature extraction of selected Train Frames
features=[]
for img in tqdm(frames):
  img = cv2.resize(img, (224,224))
  img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
  img= img/255.0
  feature_x= res_model.predict(np.array([img]))
  feature_x= np.squeeze(feature_x)
  features.append(feature_x)
print("Feature Generation of Train Frames----- Done")
features= np.array(features)
df= pd.DataFrame(features) #training features
training_data = pd.concat([training_data, df], axis=1)
# testing video frames- selection of nearest frames
cap = cv2.VideoCapture(user_testing_video_path)
# find the number of frames
video_length = int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) - 1
frame_timings=[]
frame_exists= 1
count=0
while cap.isOpened():
  # extract the frame
  ret, frame = cap.read()
  if not ret:
      continue
  time= cap.get(cv2.CAP_PROP_POS_MSEC)
  frame_timings.append(time)
  count = count + 1
  # if there are no more frames left
  if (count > (video_length-1)):
      # release the feed
      cap.release()
      # print stats
      break
  # extraction testing frames
selected_frames= [select_nearest_frame(frame_timings,x) for x in test_timestamps]
print("Selecting Nearest Test Frames----- Done")
# training video frames- selection of nearest frames
cap = cv2.VideoCapture(user_testing_video_path)
# find the number of frames
video_length = int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) - 1
frames=[]
frame_exists= 1
count=0
while cap.isOpened():
  # extract the frame
  ret, frame = cap.read()
  if not ret:
      continue
  time= cap.get(cv2.CAP_PROP_POS_MSEC)
  if time in selected_frames:
    occurrences= selected_frames.count(time)
    for j in range(occurrences):
      frames.append(frame)
  count = count + 1
  # if there are no more frames left
  if (count > (video_length-1)):
      # release the feed
      print("Extracting Nearest Test Frames----- Done")
      cap.release()
      # print stats
      break
# feature extraction of selected Test Frames
features=[]
for img in tqdm(frames):
  img = cv2.resize(img, (224,224))
  img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
  img= img/255.0
  feature_x= res_model.predict(np.array([img]))
  feature_x= np.squeeze(feature_x)
  features.append(feature_x)
print("Feature Generation of Test Frames----- Done")
features= np.array(features)
df= pd.DataFrame(features) #training features
testing_data = pd.concat([testing_data, df], axis=1)
# releasing memory
frames=[]
frame_timings=[]
# saving final user data
training_data.to_excel(str(uid)+"_training.xlsx", index=False)
```

```
testing_data.to_excel(str(uid)+"_testing.xlsx", index=False)

print("Process Complete-----Done")
```

Assignment- 8

```
## Train a model using Decision Tree algorithm

In[1]: from google.colab import drive

drive.mount('/content/drive')

# importing the libraries

In[2]: import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.model_selection import ParameterGrid

from sklearn.model_selection import GridSearchCV

from collections import Counter

from sklearn.datasets import make_classification

from imblearn.over_sampling import SMOTE

from imblearn.under_sampling import RandomUnderSampler

from imblearn.pipeline import Pipeline

from matplotlib import pyplot

from numpy import where

# decision tree evaluated on imbalanced dataset

In[3]: from numpy import mean

from sklearn.datasets import make_classification

from sklearn.model_selection import cross_val_score

from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.tree import DecisionTreeClassifier

# importing the train dataset

In[4]: dataset = pd.read_csv('/content/drive/MyDrive/cair/train_combined_50.csv')

X = dataset.iloc[:, 2:].values

y = dataset.iloc[:, 1].values

# training the Decision Tree Classification model on the Training set

In[5]: from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion = 'gini', random_state =
0,splitter='random', max_depth=1)

classifier.fit(X, y)

# evaluate pipeline

In[6]: cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)

scores = cross_val_score(classifier, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)

...

print('Mean ROC AUC: %.3f' % mean(scores))

In[7]: from sklearn.metrics import confusion_matrix, accuracy_score

 y_pred = classifier.predict(X)

 cm = confusion_matrix(y, y_pred)

 print(cm)

 accuracy_score(y, y_pred)

In[8]: parameters ={'max_depth': [5],

             'criterion' : ['gini', 'entropy'],

            'splitter' : ['best','random'],

            'min_samples_leaf': [2],

            'min_samples_split': [2],
            'random_state':[0],
            'min_weight_fraction_leaf':[0.2],
            'max_features':['log2','sqrt','auto'],
            'max_leaf_nodes':[1,2],
            'min_impurity_decrease':[0.0,0.1,0.2],
            'class_weight':['balanced'],
            'ccp_alpha':[0.0,0.1,0.2]}
param_size = ParameterGrid(parameters)
len(param_size)
In[9]: grid_search = DecisionTreeClassifier()
grid_search = GridSearchCV(
    grid_search,
    parameters,
    cv=5,
    scoring='accuracy',n_jobs=-1)
grid_result= grid_search.fit(X,y)
print('Best Params: ', grid_result.best_params_)
print('Best Score: ', grid_result.best_score_)
# importing the test dataset
In[10]: dataset = pd.read_excel('/content/drive/MyDrive/algo try/tested.xlsx')
Xt = dataset.iloc[:, 2:].values
yt = dataset.iloc[:, 1].values
# making the Confusion Matrix
In[11]: from sklearn.metrics import confusion_matrix, accuracy_score
y_pred = classifier.predict(Xt)
```

```
cm = confusion_matrix(yt, y_pred)
print(cm)
accuracy_score(yt, y_pred)
```

Assignment -9

```
## Train a model using Forest tree method
# importing the libraries
In[1]: from google.colab import drive
drive.mount('/content/drive')
In[2]: import matplotlib.pyplot as plt
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from imblearn.pipeline import Pipeline
# importing the dataset
In[3]: dataset = pd.read_csv('/content/drive/MyDrive/cair/7_training.csv')
X = dataset.iloc[:,3:]
y = dataset.iloc[:,2]
In[4]: label_encoder = preprocessing.LabelEncoder()
y = label_encoder.fit_transform(y)
X.targetName=label_encoder.fit_transform(X.targetName)
X.targetPosition=label_encoder.fit_transform(X.targetPosition)
X.disctractorName=label_encoder.fit_transform(X.disctractorName)
```

```
X.disctractorPosition=label_encoder.fit_transform(X.disctractorPosition)

X.timeStamp=label_encoder.fit_transform(X.timeStamp)

X.position=label_encoder.fit_transform(X.position)

# feature Scaling

In[5]: from sklearn.preprocessing import StandardScaler

sc=StandardScaler()

X[:,2:]=sc.fit_transform(X[:,2:])

y[:,1]=sc.transform(y[:,1])

# training the Random Forest Classification model on the Training set

In[6]: classifier = RandomForestClassifier(n_estimators = 300, criterion = 'entropy',
random_state = 0)

classifier.fit(X, y)

In[7]: from sklearn.metrics import confusion_matrix, accuracy_score

X_pred = mlp.predict(X)

cm = confusion_matrix(y, X_pred)

print(cm)

accuracy_score(y, X_pred)

In[8]: dataset = pd.read_csv('/content/drive/MyDrive/cair/7_testing.csv')

Xt = dataset.iloc[:,3:]

yt = dataset.iloc[:,2]

In[9]: param_grid = {  'bootstrap': [True], 'max_depth': [2, None], 'max_features':
['auto', 'log2'], 'n_estimators': [64]}

grid_search = RandomForestRegressor()

rfr = RandomForestRegressor(random_state = 1)

g_search = GridSearchCV(estimator = rfr, param_grid = param_grid, cv = 3, n_jobs =
1, verbose = 0, return_train_score=True)
```

```
g_search.fit(X, y);

print(g_search.best_params_)

print("Best: {0}, using {1}".format(g_search.cv_results_['mean_test_score'],
g_search.best_params_))

In[10]: results_df = pd.DataFrame(g_search.cv_results_)

results_df

# only for mlp

In[11]: reg = MLPClassifier(hidden_layer_sizes=(64,64),activation="relu"
,random_state=0, max_iter=2000).fit(X, y)

# Making the Confusion Matrix

In[12]: from sklearn.metrics import confusion_matrix, accuracy_score

y_pred = mlp.predict(Xt)

cm = confusion_matrix(yt, y_pred)

print(cm)

accuracy_score(yt, y_pred)

In[13]: from sklearn.metrics import classification_report

print(classification_report(yt, y_pred))
```

Assignment- 10

```
## Train a model using Support Vector Machine

# importing the libraries

In[1]: import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# importing the dataset
```

```
In[2]: dataset = pd.read_csv(r'C:\Project\trained.csv')

X = dataset.iloc[:, 2:].values

y = dataset.iloc[:, 1].values

# training the SVM model on the Training set

In[3]: from sklearn.svm import SVC

classifier = SVC(kernel = 'linear', random_state = 0)

classifier.fit(X, y)

# making the Confusion Matrix

In[4]: from sklearn.metrics import confusion_matrix, accuracy_score

y_pred = classifier.predict(X_test)

cm = confusion_matrix(y_test, y_pred)

print(cm)

accuracy_score(y_test, y_pred)

# initialise the Scaler

In[5]: scaler = StandardScaler()

# scale the data

In[6]: scaler.fit(dataset)

In[7]: from sklearn.datasets import load_iris

from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split

from sklearn.model_selection import cross_val_score

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

import pandas as pd

In[8]: dataset = pd.read_csv(r'C:\Project\trained.csv')
```

```
x = dataset.iloc[:, 2:].values

y = dataset.iloc[:, 1].values

In[9]: lsvc = LinearSVC(dual=False)

print(lsvc)

lsvc.fit(x, y)

score = lsvc.score(x, y)

print("Score: ", score)

cv_scores = cross_val_score(lsvc, x, y, cv=10)

print("CV average score: %.2f" % cv_scores.mean())

In[10]: dataset = pd.read_csv(r'C:\Project\tested.csv')

xt = dataset.iloc[:, 2:].values

yt = dataset.iloc[:, 1].values

import numpy as np

xt = np.nan_to_num(xt)

yt = np.nan_to_num(yt)

from sklearn.metrics import confusion_matrix, accuracy_score

ypred = lsvc.predict(xt)

cm = confusion_matrix(yt, ypred)

print(cm)

accuracy_score(yt, ypred)

In[11]: from sklearn.svm import LinearSVC

from sklearn.datasets import load_iris

from sklearn.datasets import make_classification

from sklearn.model_selection import train_test_split

from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

import pandas as pd

In[12]: dataset = pd.read_csv(r'C:\Project\trained.csv')

x = dataset.iloc[:, 2:].values

y = dataset.iloc[:, 1].values

In[13]: svc = svm.SVC(kernel='rbf', C=1,gamma='auto').fit(x, y)

In[14]: dataset = pd.read_csv(r'C:\Project\tested.csv')

xt = dataset.iloc[:, 2:].values

yt = dataset.iloc[:, 1].values

In[15]: xt = np.nan_to_num(xt)

yt = np.nan_to_num(yt)

In[16]: from sklearn.metrics import confusion_matrix, accuracy_score

ypred = lsvc.predict(xt)

cm = confusion_matrix(yt, ypred)

print(cm)

accuracy_score(yt, ypred)

In[17]: from sklearn.model_selection import GridSearchCV

import pandas as pd

import numpy as np

from sklearn.metrics import classification_report, confusion_matrix

from sklearn.svm import SVC

In[18]: dataset = pd.read_csv(r'C:\Users\acslab\Downloads\train_combined_50.csv')

x = dataset.iloc[:, 2:].values

y = dataset.iloc[:, 1].values
```

```
# defining parameter range

In[19]: param_grid = {'C': [0.1, 1, 10, 100, 1000],

              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],

              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search

In[20]: grid.fit(x, y)

# print how our model looks after hyper-parameter tuning

In[21]: print(grid.best_estimator_)

In[22]: dataset = pd.read_csv(r'C:\Project\tested.csv')

xt = dataset.iloc[:, 2:].values

yt = dataset.iloc[:, 1].values

In[23]: xt = np.nan_to_num(xt)

yt = np.nan_to_num(yt)

In[24]: grid_predictions = grid.predict(xt)

print(confusion_matrix(yt,grid_predictions))

print(classification_report(yt,grid_predictions))#Output

accuracy_train = accuracy_score(y,predict_train)

print('accuracy_score on train dataset : ', accuracy_train)

In[25]: predict the target on the test dataset

predict_test = model.predict(xt)

# accuracy Score on test dataset

In[26]: accuracy_test = accuracy_score(yt,predict_test)

print('accuracy_score on test dataset : ', accuracy_test)
```

Assignment- 11

```
## Train a model using Gaussian Classifier

In[1]: import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

In[2]: dataset = pd.read_csv(r'C:\Project\trained.csv')

x = dataset.iloc[:, 2:].values

y = dataset.iloc[:, 1].values

In[3]: print(dataset.dtypes)

df = dataset.infer_objects()

df.dtypes

In[4]: from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()

nb.fit(x, y)

print("Naive Bayes score: ",nb.score(x, y))

In[5]: dataset = pd.read_csv(r'C:\Users\acslab\Downloads\test_combined_50.csv')

xt = dataset.iloc[:, 2:].values

yt = dataset.iloc[:, 1].values

In[6]: xt = np.nan_to_num(xt)

yt = np.nan_to_num(yt)

In[7]: from sklearn.metrics import confusion_matrix, accuracy_score

ypred = nb.predict(xt)

cm = confusion_matrix(yt, ypred)

print(cm)

accuracy_score(yt, ypred)
```

```
# tuning Parameters

# define Grid Search Parameters

In[8]: param_grid_nb = {

    'var_smoothing': np.logspace(0,-9, num=100)

}

from sklearn.naive_bayes import GaussianNB

from sklearn.model_selection import GridSearchCV

nbModel_grid = GridSearchCV(estimator=GaussianNB(), param_grid=param_grid_nb,
verbose=1, cv=2, n_jobs=-1)

In[9]: nbModel_grid.fit(x, y)

In[10]: print(nbModel_grid.best_estimator_)

y_pred = nbModel_grid.predict(xt)

print(y_pred)

In[11]: from sklearn.metrics import confusion_matrix

print(confusion_matrix(yt, y_pred), ": is the confusion matrix")

cm = confusion_matrix(yt, y_pred)

In[12]: from sklearn.metrics import accuracy_score

print(accuracy_score(yt, y_pred), ": is the accuracy score")

In[13]: GridSearchCV(pipeline, param_grid=parameters)

from sklearn.pipeline import Pipeline

pipeline = Pipeline([

    ('clf', GaussianNB())

])

parameters = {

    'clf__priors': [None],
    'clf__var_smoothing': [0.00000001, 0.000000001, 0.00000001]
}
In[14]: cv = GridSearchCV(pipeline, param_grid=parameters)
cv.fit(x, y)
In[15]: from sklearn.metrics import confusion_matrix, accuracy_score
y_pred_gnb = cv.predict(xt)
cm = confusion_matrix(yt, y_pred_gnb)
print(cm)
In[16]: accuracy_score(yt, y_pred_gnb)
# sklearn Pipeline
In[17]: from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
pipe = Pipeline(steps=[
                ('pca', PCA()),
                ('estimator', GaussianNB()),
                ])
parameters = {'estimator__var_smoothing': [1e-11, 1e-10, 1e-9]}
Bayes = GridSearchCV(pipe, parameters, scoring='accuracy', cv=2).fit(x, y)
print(Bayes.best_estimator_)
In[18]: print('best score:')
In[19]: print(Bayes.best_score_)
predictions = Bayes.best_estimator_.predict(xt)
cm = confusion_matrix(yt, predictions)
print(cm)
In[20]: accuracy_score(yt, predictions)
```

```
In[21]: from sklearn.datasets import load_iris

from sklearn.gaussian_process import GaussianProcessClassifier

from sklearn.gaussian_process.kernels import RBF

In[22]: kernel = 1.0 * RBF(1.0)

gpc = GaussianProcessClassifier(kernel=kernel,

        random_state=0).fit(X, y)

gpc.score(X, y)
```

Assignment- 12

```
## Train a model using K-Nearest Neighbor

# import Libaries

In[1]:  import numpy as np

import pandas as pd

from sklearn.model_selection import GridSearchCV

# import Dataset

In[2]: from google.colab import drive

drive.mount('/content/drive')

# read the CSV file

In[3]: dataset = pd.read_csv('/content/drive/MyDrive/Folder/7_training.csv')

# import Label Encoder

In[4]:  from sklearn import preprocessing

from sklearn.preprocessing import LabelEncoder

label_encoder = preprocessing.LabelEncoder()

new_dataset = dataset.apply(LabelEncoder().fit_transform)

# define rows and columns for x and y from train dataset

In[5]: x = new_dataset.iloc[:, 2:].values
```

```
y = new_dataset.iloc[:,1].values

# apply KNN Model

In[6]: from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=49)

knn.fit(x, y)

In[7]: from sklearn.metrics import confusion_matrix, accuracy_score

y_pred = knn.predict(x)

cm = confusion_matrix(y, y_pred)

k_fit = accuracy_score(y, y_pred)

print(k_fit)

# same steps as previous

In[8]: dataset2 = pd.read_csv('/content/drive/MyDrive/Folder/7_testing.csv')

new_dataset_2 = dataset2.apply(LabelEncoder().fit_transform)

xt = new_dataset_2.iloc[:, 2:].values

yt = new_dataset_2.iloc[:,1].values

# predict on dataset

In[9]: from sklearn.metrics import confusion_matrix, accuracy_score

y_pred = knn.predict(xt)

cm = confusion_matrix(yt, y_pred)

k_fit = accuracy_score(yt, y_pred)

print(k_fit)

# hyper parameter tuning and optimization

In[10]: knn = KNeighborsClassifier()

parameters = {'n_neighbors': [49],

              'weights': ['uniform'],
            'algorithm' : ['auto'],
            'leaf_size' : [5],
            'p' : [2]}
# grid Search
# run the grid search
In[11]:grid_obj = GridSearchCV(knn, parameters)
grid_obj = grid_obj.fit(x, y)
In[12]: cm = confusion_matrix(yt, y_pred)
hyperparam_1 = accuracy_score(yt, y_pred)
print(hyperparam_1)
# compare Models parameters
In[13]: models = pd.DataFrame({
   'K-values': ['()', '1', '3', '4', '5', '7',
            '9','15','19','acc_hp'],
   'Score': [k_fit, k1_fit, k3_fit, k4_fit, k5_fit, k7_fit, k9_fit, k15_fit, k19_fit,
hyperparam_1]})
models.sort_values(by='Score', ascending=False)
```

Assignment- 13

```
## Train a model using K-Nearest Neighbor, using SMOTE analysis
# import Libaries
In[1]: import numpy as np
import pandas as pd
import imblearn
# import Dataset
```

```
In[2]: from google.colab import drive

drive.mount('/content/drive')

# read the CSV file

In[3]: dataset = pd.read_csv('/content/drive/MyDrive/Dataset/training/46.csv')

# import Label Encoder

In[4]: from sklearn import preprocessing

from sklearn.preprocessing import LabelEncoder

label_encoder = preprocessing.LabelEncoder()

new_dataset = dataset.astype(str).apply(LabelEncoder().fit_transform)

# define rows and columns for x and y from train dataset

In[5]: x = new_dataset.iloc[:, 2:].values

y = new_dataset.iloc[:,1].values

print(x.shape)

print(y.shape)

# summarize class distribution

In[6]:  from collections import Counter

from sklearn.datasets import make_classification

from matplotlib import pyplot

from numpy import where

counter = Counter(y)

print(counter)

# scatter plot of examples by class label

In[7]:  for label, _ in counter.items():

        row_ix = where(y == label)[0]

        pyplot.scatter(x[row_ix, 0], x[row_ix, 1], label=str(label))
pyplot.legend()

pyplot.show()

from imblearn.over_sampling import SMOTE

oversample = SMOTE()

x, y = oversample.fit_resample(x, y)

In[8]: print

print(x.shape)

print(y.shape)

# summarize the new class distribution

In[9]: counter = Counter(y)

print(counter)

# apply KNN Model

In[10]:  from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=49)

knn.fit(x, y)

# predict on dataset

In[11]: from sklearn.metrics import confusion_matrix, accuracy_score

y_pred = knn.predict(x)

cm = confusion_matrix(y, y_pred)

In[12]: k462_fit = accuracy_score(y, y_pred)

print(k462_fit)

# compares Models parameters

In[13]: models = pd.DataFrame({

    'User Id': ['1','2','4', '5','6','7','8','9','10', '11','12','13',

                '14','15','16','17','18','19','20','21','22',
                '24','25','26','29','30','31','32','33','34','35','36',
                '37','38','39','41','42','43', '44','45','46','462', '48'],
    'Score':      [k1_fit,k2_fit,      k4_fit,      k5_fit,      k6_fit,      k7_fit,      k8_fit,
k9_fit,k10_fit,k11_fit,k12_fit,
               k13_fit, k14_fit,k15_fit,k16_fit,k17_fit,k18_fit, k19_fit, k20_fit,
               k21_fit, k22_fit, k24_fit, k25_fit, k26_fit,k29_fit, k30_fit,k31_fit, k32_fit,
               k33_fit, k34_fit, k35_fit, k36_fit, k37_fit, k38_fit, k39_fit, k41_fit, k42_fit,
               k43_fit, k44_fit, k45_fit, k46_fit, k462_fit, k48_fit]})
models.sort_values(by='Score', ascending=False)
```

Assignment- 14

```
## Train a model using Multilayer Perceptron using sklearn
In[1]: from google.colab import drive
drive.mount('/content/drive')
In[2]: from numpy import sqrt
import keras
from numpy import asarray
import pandas as pd
from pandas import read_csv
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
import tensorflow as tf
In[3]: dataset = pd.read_csv('/content/drive/MyDrive/cair/train_combined_50.csv')
X = dataset.iloc[:,2:].values
```

```
y = dataset.iloc[:,1].values

In[4]: dataset = pd.read_csv('/content/drive/MyDrive/cair/test_combined_50.csv')

Xt = dataset.iloc[:,2:].values

yt = dataset.iloc[:,1].values

# retrieves the values

In[5]: values = dataset.values.astype('float32')

# specify the window size

In[6]: n_steps = 5

# split into samples

In[7]: X, y = split_sequence(values, n_steps)

# reshape into [samples, timesteps, features]

In[8]:  X = X.reshape((X.shape[0], X.shape[1], 1))

# split into train/test

In[9]: n_test = 12

In[10]: X, Xt, y, yt = X[:-n_test], Xt[-n_test:], y[:-n_test], yt[-n_test:]

# define model

In[11]: model = Sequential()

model.add(LSTM(100, activation='relu', kernel_initializer='he_normal', 
input_shape=(None,1), return_sequences=True))

model.add(Dense(50, activation='relu', kernel_initializer='he_normal'))

model.add(Dense(50, activation='relu', kernel_initializer='he_normal'))

model.add(Dense(1))

# compiles the model

In[12]: model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])

# fit the model
```

In[13]: model.fit(X, y, epochs=10, batch_size=512, verbose=2, validation_data=(Xt, yt))

# evaluate the model

In[14]: mse, mae = model.evaluate(Xt, yt, verbose=0)

print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, sqrt(mse), mae))

Assignment- 15

## Train a model using Multilayer Perceptron using Tensorflow

In[1]: import tensorflow

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to_categorical

import pandas as pd

# configuration options

In[2]: feature_vector_length = 784

In[3]: num_classes = 10

dataset = pd.read_csv('/content/drive/MyDrive/cair/7_training.csv')

In[4]: from sklearn.preprocessing import LabelEncoder

dataset = dataset.astype(str).apply(LabelEncoder().fit_transform)

X = dataset.iloc[:,3:]

y = dataset.iloc[:,2]

In[5]: from imblearn.over_sampling import SMOTE

oversample = SMOTE()

X, y = oversample.fit_resample(X, y)

In[6]: print

```
print(X.shape)

print(y.shape)

model = Sequential([

     # dense layer 1

   # Dense(256, activation='tanh'),

   # dense layer 2

   Dense(128, activation='tanh'),

     # output layer

   # Dense(10, activation='tanh'),

])

model.compile(optimizer='adam',

              loss='sparse_categorical_crossentropy',

              metrics=['accuracy'])

In[7]: dataframe = pd.read_csv('/content/drive/MyDrive/cair/7_testing.csv')

dataframe = dataframe.astype(str).apply(LabelEncoder().fit_transform)

In[8]: Xt = dataset.iloc[:,3:]

yt = dataset.iloc[:,2]

Xt, yt = oversample.fit_resample(Xt, yt)

In[9]: model.fit(X, y, epochs=30,

model.summary()

model.get_config()

from keras.utils.vis_utils import plot_model

plot_model(model, to_file='model.png')

results = model.evaluate(Xt,  yt, verbose = 0)

print('test loss, test acc:', results)
```

Assignment- 16

```
## Train a model using Long-Short term memory

In[1]: from google.colab import drive

drive.mount('/content/drive')

from numpy import sqrt

import keras

from numpy import asarray

import pandas as pd

from pandas import read_csv

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import LSTM

import tensorflow as tf

In[2]: dataset = pd.read_csv('/content/drive/MyDrive/cair/train_combined_50.csv')

X = dataset.iloc[:,2:].values

y = dataset.iloc[:,1].values

In[3]: dataset = pd.read_csv('/content/drive/MyDrive/cair/test_combined_50.csv')

Xt = dataset.iloc[:,2:].values

yt = dataset.iloc[:,1].values

# retrieve the values

In[4]: values = dataset.values.astype('float32')

# specify the window size

In[5]: n_steps = 5

# split into samples
```

```
In[6]: X, y = split_sequence(values, n_steps)

# reshape into [samples, timesteps, features]

In[7]: X = X.reshape((X.shape[0], X.shape[1], 1))

# split into train/test

In[8]: n_test = 12

In[9]: X, Xt, y, yt = X[:-n_test], Xt[-n_test:], y[:-n_test], yt[-n_test:]

# define model

In[10]: model = Sequential()

model.add(LSTM(100, activation='relu', kernel_initializer='he_normal',
input_shape=(None,1), return_sequences=True))

model.add(Dense(50, activation='relu', kernel_initializer='he_normal'))

model.add(Dense(50, activation='relu', kernel_initializer='he_normal'))

model.add(Dense(1))

# compile the model

In[11]: model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])

# fit the model

In[12]: model.fit(X, y, epochs=10, batch_size=512, verbose=2, validation_data=(Xt,
yt))

# evaluate the model

In[13]: mse, mae = model.evaluate(Xt, yt, verbose=0)

print('MSE: %.3f, RMSE: %.3f, MAE: %.3f' % (mse, sqrt(mse), mae))
```