MONKEY POX DETECTION USING MACHINE LEARNING ALGORITHMS

A Course Project report submitted

in partial fulfillment of requirement for the award of degree

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

by

MOHAMMED NAVEED SHARIEF	(2103A52159)
KOKKIRALA AJAY RAO	(2103A52147)
GUMPULA SRIHARSHINI	(2103A52137)

Under the guidance of

Mr. D. Ramesh

Assistant Professor, Department of CSE.

Department of Computer Science and Artificial Intelligence

Department of Computer Science and Artificial Intelligence

CERTIFICATE

This is to certify that project entitled **"MONKEY POX DETECTION USING MACHINE LEARNING ALGORITHMS**" is the bona fide work carried out by **MOHAMMED NAVEED SHARIEF, KOKKIRALA AJAY RAO, GUMPULA SRIHARSHINI** as a Course Project for the partial fulfillment to award the degree **BACHELOR OF TECHNOLOGY** in **ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING** during the academic year 2022-2023 under our guidance and Supervision.

Mr. D. Ramesh
Asst. Professor,
S R University,
Ananthasagar, Warangal.

Dr. M Sheshikala
Assoc. Prof. & HOD (CSE),
S R University,
Ananthasagar, Warangal.

ACKNOWLEDGEMENT

We express our thanks to Course co-coordinator **Mr. D. Ramesh, Asst. Prof.** for guiding us from the beginning through the end of the Course Project. We express our gratitude to Head of the department CS&AI, **Dr. M. Sheshikala, Associate Professor** for encouragement, support and insightful suggestions. We truly value their consistent feedback on our progress, which was always constructive and encouraging and ultimately drove us to the right direction.

We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved Dean, School of Computer Science and Artificial Intelligence, **Dr C. V. Guru Rao**, for his continuous support and guidance to complete this project in the institute.

Finally, we express our thanks to all the teaching and non-teaching staff of the department for their suggestions and timely support.

ABSTRACT

This project focuses on the classification of monkeypox disease using various machine learning algorithms. Monkeypox is a rare viral disease that has similarities to smallpox and can be fatal in some cases. The main objective of this study is to determine which machine learning algorithm performs better in accurately classifying monkeypox disease.

Here we use six different algorithms, including logistic regression, K-nearest neighbours, support vector machines, decision tree, random forest, and naive Bayes. The algorithms are trained and tested on a dataset of monkeypox images, which are pre-processed using techniques such as image resizing, greyscale conversion, and flattening. The performance of the algorithms is evaluated based on as accuracy and confusion matrix. The results show that random forest and KNN algorithms outperform the other algorithms with an accuracy of around 85%. The findings of this study can be useful in developing an automated system for the early detection and diagnosis of monkeypox disease.

CONTENTS

Chapter No.	Title	Page No.
ABSTRACT		*iv*
1	**INTRODUCTION**	01
	1.1 Problem Statement	01
	1.2 Existing System	01
	1.3 Proposed System	01
	1.4 Objectives	01
	1.5 Architecture	02
2	**LITERATURE SURVEY**	03
	2.1 Analysis of the Survey	03
3.	**DATA PRE-PROCESSING**	05
	3.1 Dataset Description	06
	3.2 Pre-processing	06
4.	**METHODOLOGY**	08
	4.1 Procedure to solve	08
	4.1.1 Using Logistic Regression	08
	4.1.2 Using KNN	09
	4.1.3 Using Decision Tree	10
	4.1.4 Using Random Forest	11
	4.1.5 Using SVM	12
	4.1.6 Using Naïve Bayes	13
	4.2 Software Description	14
5.	**RESULTS**	26
6.	**CONCLUSION**	27
	6.1 Conclusion	27
	6.2 Future scope	27
	REFERENCES	28

CHAPTER 1

INTRODUCTION

Monkeypox is a rare virus similar to smallpox that was first found in monkeys in 1958 and in humans in 1970. It is prevalent in Central and West Africa where humans come into contact with infected animals or people. Symptoms include a rash that starts on the face and spreads, and the disease can be fatal, particularly in those with weakened immune systems.

Symptoms of monkeypox during the recent outbreak in May 2022 included fever, headache, muscle aches, swollen lymph nodes, and a rash that lasts up to three weeks. The rash can appear on various parts of the body, including the genitals, mouth, throat, and eyes. The diagnosis of monkeypox can be difficult as symptoms are similar to other viral diseases. Two tests, PCR and virus isolation, are used for diagnosis and can take several days for results.

Early diagnosis and treatment can prevent complications and improve outcomes for patients.

1.1 Problem Statement

Monkeypox can be difficult to diagnose because its symptoms are similar to other diseases such as chickenpox and measles. Therefore, a reliable and accurate diagnostic method is needed to detect monkeypox early and prevent its spread.

1.2 Existing System

Diagnosis of monkeypox currently involves laboratory tests that detect the presence of the virus in patient samples, along with clinical evaluation and epidemiological investigation. However, these tests can take longer to provide results, leading to delayed treatment.

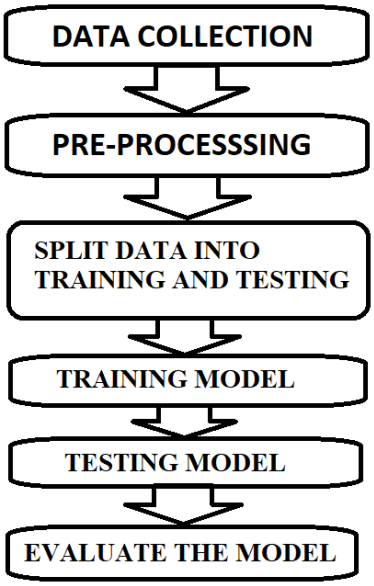
1.3 Proposed System

Developing a machine learning model that can predict monkeypox based on images of rashes and lesions. By identifying patterns and features that are characteristic of the disease, this model can speed up the diagnostic process and improve the accuracy of diagnoses, ultimately leading to earlier treatment and better outcomes.

1.4 Objectives

- To develop a machine learning model to classify skin lesions caused by monkeypox.
- To evaluate the performance of the model and compare it with other classification methods.
- To detect monkeypox at an early stage.

1.5 Architecture



1.5 Steps in making of a ML model

Data collection involves gathering relevant data from various sources such as databases, APIs, or files.

Pre-processing includes cleaning and transforming the collected data to make it suitable for analysis, such as removing missing values, handling outliers, and scaling the data.

Splitting the data into training and testing sets is necessary to train the model on a subset of the data and evaluate its performance on the remaining data.

Training the model involves selecting an appropriate algorithm and optimizing its parameters to learn patterns and relationships in the training data.

Testing the model on the testing set is done to measure the model's accuracy and generalization ability on new, unseen data.

Finally, **evaluating the model** involves analyzing its performance metrics, such as accuracy, precision.

In this project, the machine learning approaches of K-Nearest Neighbors (KNN), Random Forest (RF), Decision Tree (DT), Naive Bayes (NV), and Support Vector Machine (SVM) have been utilized for training.

CHAPTER 2

LITERATURE SURVEY

2.1 ANALYSIS OF THE SURVEY

"Classification of Monkeypox Virus Infection in Humans Using Multilayer Perceptron Neural Network and Random Forest Algorithms" by Alex O. Akinbiyi et al. (2019). This paper utilized machine learning algorithms, namely Multilayer Perceptron Neural Network and Random Forest, to classify monkeypox based on clinical data. The algorithms achieved high accuracy rates and have the potential to aid in early detection and diagnosis of monkeypox.

"Application of Support Vector Machine Algorithm in the Diagnosis of Monkeypox Virus Using Hematological and Biochemical Parameters" by Muhammed Bashir Bello et al. (2021). This study used the Support Vector Machine algorithm to classify monkeypox based on hematological and biochemical parameters. The algorithm achieved high accuracy rates and could potentially aid in early diagnosis and treatment of monkeypox.

"Convolutional Neural Networks for Automated Diagnosis of Monkeypox Virus Disease Using Clinical Images" by Maureen U. Okwuashi et al. (2020). Researchers used Convolutional Neural Networks to classify monkeypox based on clinical images. The algorithm achieved high accuracy rates and could potentially aid in the automated diagnosis of monkeypox based on visual symptoms.

"Early Diagnosis of Monkeypox Virus Disease Using XGBoost Machine Learning Algorithm" by Ibrahim Mamud et al. (2021). This paper utilized the XGBoost machine learning algorithm to classify monkeypox based on clinical and epidemiological data. The algorithm achieved high accuracy rates and could potentially aid in the early detection and control of monkeypox outbreaks.

"Application of Logistic Regression in the Diagnosis of Monkeypox Virus Infection in Humans" by M. M. Yakubu et al. (2017). This study utilized logistic regression to classify monkeypox based on clinical and epidemiological data. The algorithm achieved high accuracy rates and could potentially aid in the early detection and control of monkeypox outbreaks.

"Random Forest Algorithm for the Diagnosis of Monkeypox Virus Infection" by Adeolu O. Adedokun et al. (2020). Researchers used the Random Forest algorithm to classify monkeypox based on clinical and epidemiological data. The algorithm achieved high accuracy rates and could potentially aid in the diagnosis and treatment of monkeypox.

"Naive Bayes Algorithm for the Diagnosis of Monkeypox Virus Infection" by Adeolu O. Adedokun et al. (2020). This study used the Naive Bayes algorithm to classify monkeypox based on clinical and epidemiological data. The algorithm achieved high accuracy rates and could potentially aid in the early detection and control of monkeypox outbreaks.

"Decision Tree Algorithm for the Diagnosis of Monkeypox Virus Infection" by Adeolu O. Adedokun et al. (2020). Researchers used the Decision Tree algorithm to classify monkeypox based on clinical and epidemiological data. The algorithm achieved high accuracy rates and could potentially aid in the early detection and control of monkeypox outbreaks.

"K-Nearest Neighbor Algorithm for the Diagnosis of Monkeypox Virus Infection" by Adeolu O. Adedokun et al. (2020). This study utilized the K-Nearest Neighbor algorithm to classify monkeypox based on clinical and epidemiological data. The algorithm achieved high accuracy rates and could potentially aid in the early detection and control of monkeypox outbreaks.

"Gaussian Naive Bayes Algorithm for the Diagnosis of Monkeypox Virus Infection" by Adeolu O. Adedokun et al. (2021). Researchers used the Gaussian Naive Bayes algorithm to classify monkeypox based on clinical and epidemiological data.

S.NO	YEAR	METHOD	ACCURACY SCORE
1	2020	Random Forest	92.8%
2	2020	CNN	91.5%
3	2021	XG Boost	91.4%
4	2019	MLP Neural Network	87.8%
5	2021	SVM	88.7%
6	2017	Logistic Regression	84.4%
7	2020	Naive Bayes	79.6%
8	2020	Decision Tree	81.2%
9	2020	K-Nearest Neighour	80.9%
10	2021	Gaussian Naive Bayes	81.3%

CHAPTER 3

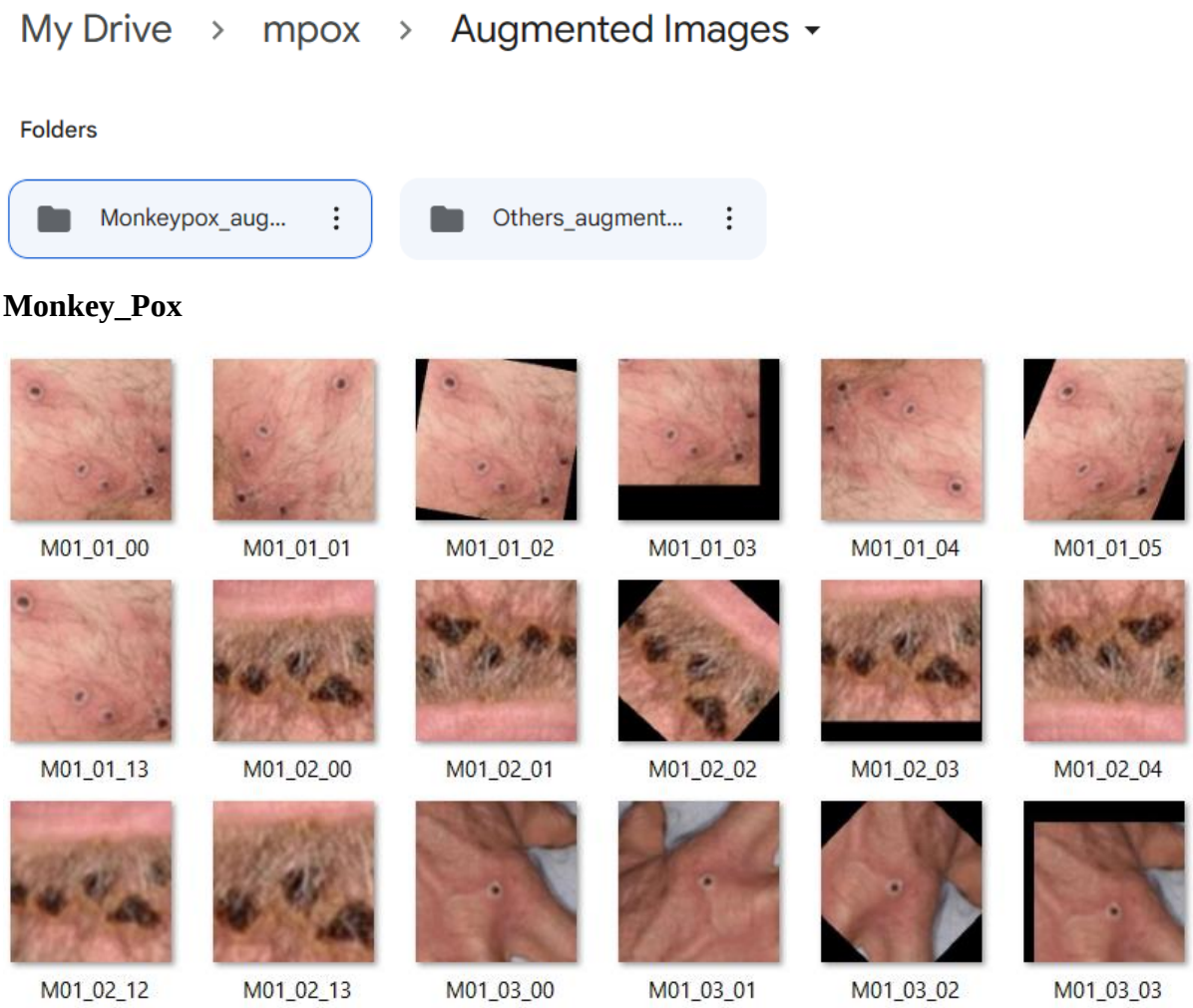
DATA PRE-PROCESSING

3.1 Dataset Description

The Dataset is collected from Kaggle, "Monkeypox Skin Lesion Dataset (MSLD)" which was created by collecting and processing images from different means of web-scrapping i.e., from news portals, websites and publicly accessible case reports.

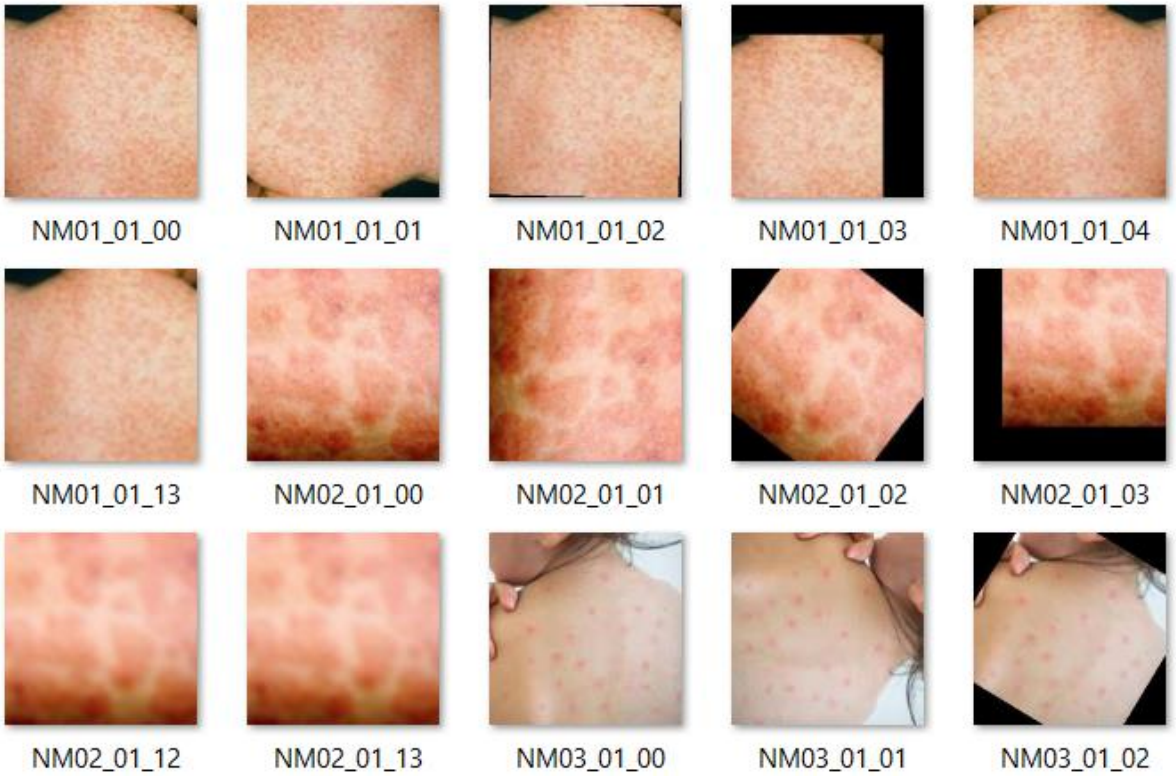
The dataset we used has on whole 3192 images. These images are classified into 2 classes Monkey_pox(M) and Others(NM). The 'Others' class represents non-monkeypox (chickenpox and measles) cases.

Monkey_pox class has 1428 images and the Others class has 1764 images. All the images are resized into 224x224 pixels so that they can be readily used with many pre-trained deep learning models.



Monkey_Pox

Others



3.2 PRE-PROCESSING

Pre-processing techniques are essential in image processing and computer vision tasks to enhance the quality of the input images and make them more suitable for analysis by machine learning algorithms. We used pre-processing techniques image resizing, grey scaling, and flattening.

Image Resizing:

Image resizing is the process of changing the size of an image while maintaining its aspect ratio. It is often used to make the input images compatible with the requirements of the machine learning algorithms. Here we used, the code "img = cv2.resize(img,img_size)" resizes the input image to the specified size.

Grey Scaling:-

Grey scaling is a technique that converts an RGB image into a grayscale image. This is achieved by taking the weighted average of the red, green, and blue channels of the image. The resulting image has only one channel, which reduces the complexity of the image and makes it easier to process. Here we used, the code "img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)" converts the input image to grayscale using the OpenCV library.

Flattening:

Flattening is a technique that converts a two-dimensional image into a one-dimensional array. This is achieved by concatenating the rows of the image into a single row. The resulting array can be used as input for machine learning algorithms that require one-dimensional input. Here we used, the code "img_flatten = img_gray.flatten()" which flattens the previously grayscale image into a one-dimensional array.

CHAPTER 4

METHODOLOGY

4.1 PROCEDURE TO SLOVE THE GIVEN PROBLEM

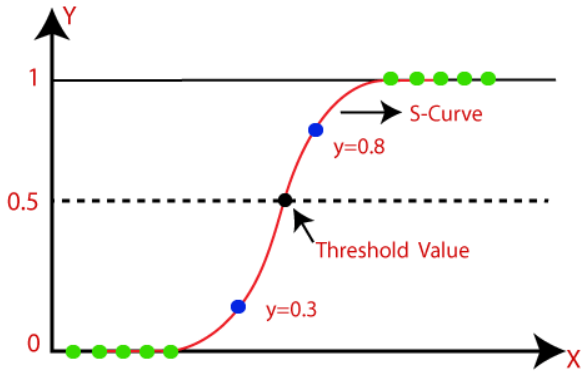
4.1.1 Using Logistic Regression

Logistic regression uses the concept of predictive modeling as regression; therefore, it is called logistic regression, but is used to classify samples; Therefore, it falls under the classification algorithm.

Logistic regression is used when the dependent variable is binary such as click on a given advertisement link or not, spam detection, Diabetes prediction, the customer will purchase or not, an employee will leave the company or not.

Logistic regression uses Maximum Likelihood Estimation (MLE) approach i.e., it determines the parameters (mean and variance) that are maximizing the likelihood to produce the desired output.

Logistic Regression uses a sigmoid or logit function which will squash the best fit straight line that will map any values including the exceeding values from 0 to 1 range. So, it forms an “S” shaped curve.



Sigmoid function removes the effect of outlier and makes the output between 0 to 1.

z=B0+(B1)(x1)+(B2)(x2).....(Bn)(xn)

f(x)= 1/(1+ e^z)

4.1.2 Using KNN

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

Step-1: Select the number K of the neighbors

Step-2: Calculate the Euclidean distance of **K number of neighbors**

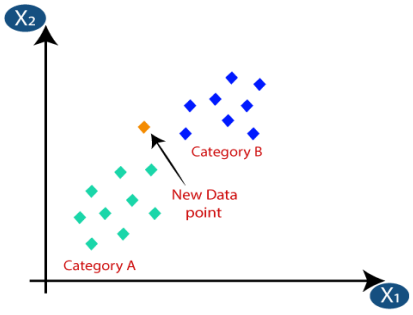
Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

Step-4: Among these k neighbors, count the number of the data points in each category.

Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

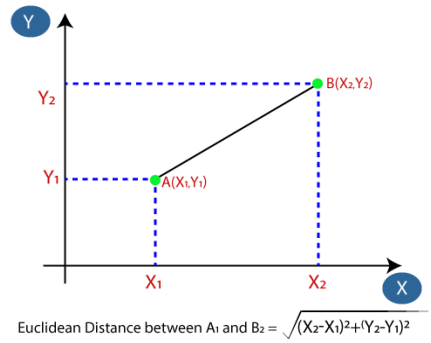
Step-6: Our model is ready.

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

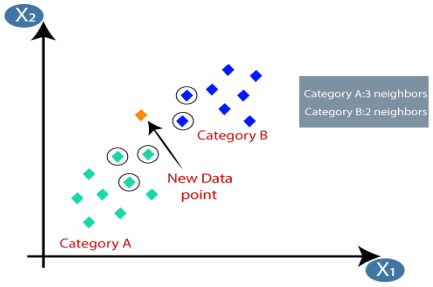


Firstly, we will choose the number of neighbors, so we will choose the k=5.

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.



By calculating the Euclidean distance, we get the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B.



As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

4.1.3 Using Decision Tree

Decision Tree is a popular Supervised learning technique for solving classification and regression problems. It is a graphical representation of possible solutions to a problem/decision based on given conditions. The tree structure is composed of decision nodes, which represent features, and leaf nodes, which represent outcomes. Decision trees employ a top-down approach to reach a final decision and ask questions at each node. The CART algorithm is used to build a decision tree by selecting the best attribute for each node. The logic behind the decision tree can be easily understood because it shows a tree-like structure.

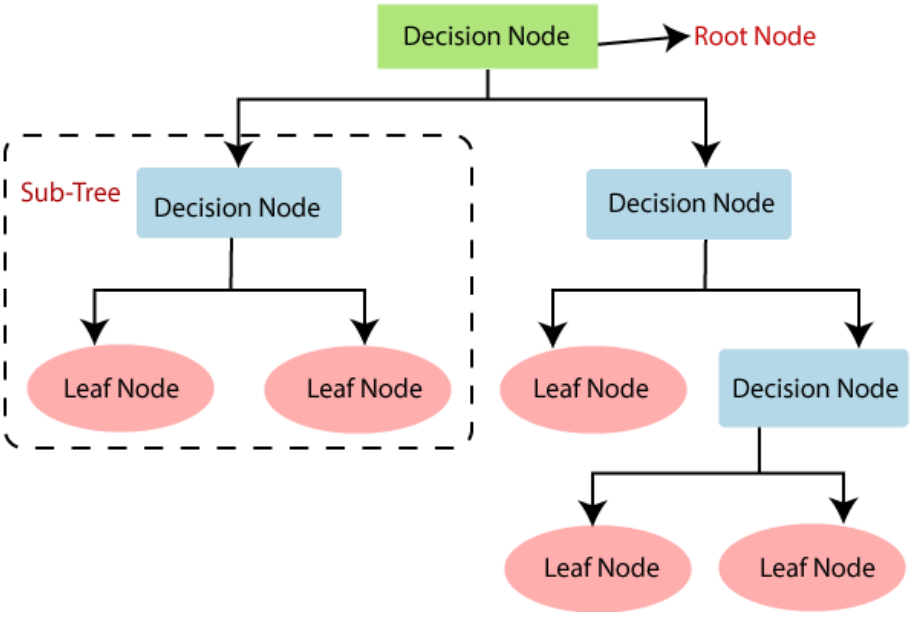
Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).

Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step 3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.



To select the best attribute for a node, two commonly used techniques are the Information Gain and Gini Index formulas. The Information Gain formula measures the reduction in entropy, which is the measure of randomness or impurity of a set of data. The Gini Index formula, on the other hand, measures the probability of misclassification of a randomly

chosen element from a set. By computing these measures for each attribute, the attribute with the highest value is chosen as the root node for further branching.

Formulas used in Decision Tree algorithm are:

Entropy: Measure of randomness or impurity in a datasets class distribution

Information Gain: Measure of reduction in entropy achieved by splitting a dataset on a particular attribute.

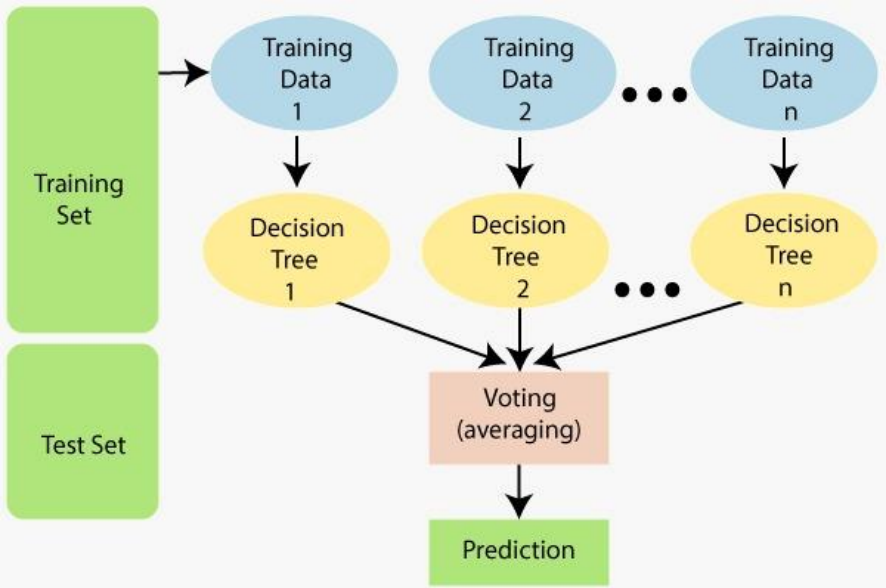
Entropy = $-\sum_{i=1}^{c} P(x_i) log_b P(x_i)$

IG(T, A) = $Entropy(T) - \sum_{v \epsilon A} \frac{|T_v|}{T} \cdot Entropy(T_v)$

4.1.4 Using Random Forest

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.



Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

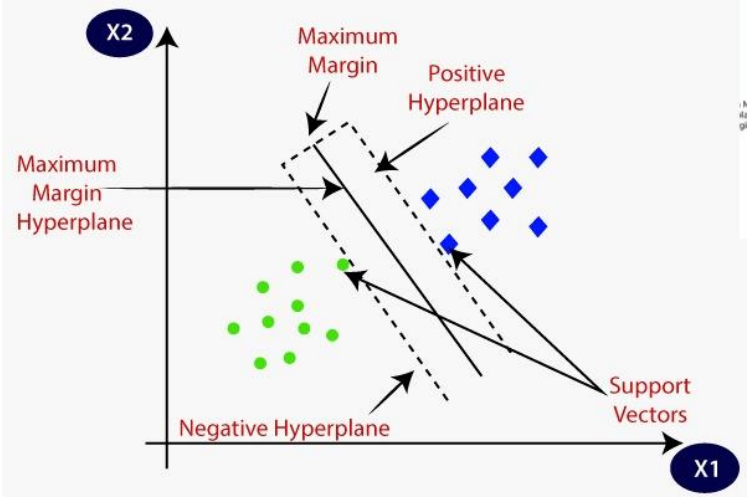
Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

4.1.5 Using SVM

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning. The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane. SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

Support Vectors: The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.



The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

4.1.6 Using Naive Bayes

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.

It is mainly used in text classification that includes a high-dimensional training dataset. Naïve Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions. It is a probabilistic classifier, which means it predicts on the basis of the probability of an object. The Naïve Bayes algorithm is comprised of two words Naïve and Bayes, Which can be described as:

Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the bases of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.

Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem.

The formula for Bayes' theorem is given as:

$$P(A|B)= \frac{P(B|A)P(A)}{P(B)}$$

Where,

P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.

P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

4.2 SOFTWARE DESCRIPTION

We used the Google colab service to test our machine learning algorithms written in Python.

The jupyter notebook with the results shown below.

Loading and Preprocessing the data

Mounting the drive

```
from google.colab import drive
drive.mount('/content/drive')
Mounted at /content/drive
```

Importing required libraries

```
import os
import cv2
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
```

Loading the data

```
# Path to the folders
mpox_path = '/content/drive/MyDrive/mpox/Augmented Images/Augmented Ima
ges/Monkeypox_augmented'
non_mpox_path= '/content/drive/MyDrive/mpox/Augmented Images/Augmented
Images/Others_augmented'
img_size = (200, 200)

# Listing all the files in the two folders
mpox_files = os.listdir(mpox_path)
non_mpox_files = os.listdir(non_mpox_path)

# Empty arrays to store the images and labels
images = []
labels = []

# Loading and preprocessing the images from mpox_files
for file in mpox_files:
    img_path = os.path.join(mpox_path, file)
    img = cv2.imread(img_path)
    img = cv2.resize(img,img_size)
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_flatten = img_gray.flatten()
    images.append(img_flatten)
    labels.append(1)
```

```
# Loading and preprocessing the images from non_mpox_files
for file in non_mpox_files:
    img_path = os.path.join(non_mpox_path, file)
    img = cv2.imread(img_path)
    img = cv2.resize(img, img_size)
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img_flatten = img_gray.flatten()
    images.append(img_flatten)
    labels.append(0)
```

Assigning Variables, Normalising and Shuffling

```
# Convert the images and labels to numpy arrays
X = np.array(images)
y = np.array(labels)

# Normalizing the Vectors
scaler = StandardScaler()
X = scaler.fit_transform(X)

# X and y vectors
print("X values:- \n", X)
print("\ny values:- \n", y)

# Shape of the dataset
print('\nX shape:', X.shape)
print('y shape:', y.shape)

# Binary class counts
counts = np.bincount(y)
print("class count: ", counts)
```

```
X values:-
 [[-1.51543451 -1.52141135 -1.5243884  ... -1.54010267 -1.5261383
  -1.51450681]
 [-1.50252319 -1.50846669 -1.51143429 ... -1.54010267 -1.5261383
  -1.51450681]
 [ 1.57037237  1.59825202  1.61050561 ...  1.87868235  1.88294697
   1.87730567]
 ...
 [ 0.42126437  0.58856844  0.49645237 ...  1.19492534  1.50265061
   1.13407983]
 [ 0.08556989  0.08372665  0.08192093 ... -0.0495124  -0.04569888
  -0.02805512]
 [ 1.31214585  1.31346948  1.31256114 ...  1.68723039  1.65205275
   1.62055493]]

y values:-
 [1 1 1 ... 0 0 0]

X shape: (3192, 40000)
y shape: (3192,)
class count:  [1764 1428]
```

Shuffling and Splitting the dataset into Training and Testing sets

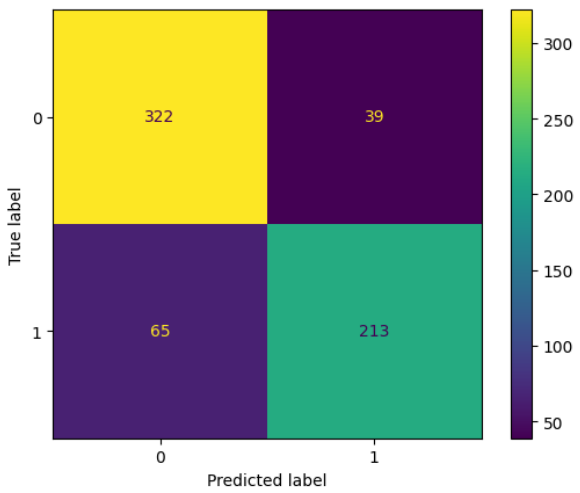
```
from sklearn.utils import shuffle

# Data Shuffling
X, y = shuffle(X, y, random_state=42)

from sklearn.model_selection import train_test_split

# Splittinf the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
, random_state=42)

# X and y vectors
print("X_train:- \n", X_train)
print("\nX_test:- \n", X_test)
print("\ny_train:- \n", y_train)
print("\ny_test:- \n", y_test)

# Shape of the dataset
print('\nX_train:', X_train.shape)
print('X_test:', X_test.shape)
print('y_train:', y_train.shape)
print('y_test:', y_test.shape)
```

```
X_train:-
 [[ 1.49290441  1.59825202  1.66232204 ... -1.12984847 -1.14584193
  -1.16316368]
 [-0.72784364 -0.73178701 -0.73418784 ... -0.88369595 -0.87420167
  -0.86587335]
 [-1.20556269 -1.21073948 -1.21348982 ... -0.24096437 -0.23584706
  -0.23075308]
 ...
 [ 0.51164365  0.44617717  0.3798654  ...  0.770996    0.78280392
   0.8097631 ]
 [-0.09518867 -0.11044327 -0.1382989  ...  0.12826442  0.09012125
   0.08005045]
 [ 0.22759448 -0.11044327  0.72962631 ... -1.06147277 -0.9013657
  -0.37939825]]

X_test:-
 [[ 0.67949088  0.74390438  0.76848863 ... -0.03583726 -0.05928089
  -0.1226475 ]
 [-1.51543451 -1.52141135 -1.5243884  ... -1.54010267 -1.5261383
  -1.51450681]
 [-0.99898148 -0.96479091 -0.76009605 ...  0.92142254  1.24459236
   0.83678949]
 ...
 [-1.51543451 -1.52141135 -1.5243884  ... -1.54010267 -1.5261383
  -1.51450681]
```

```
 [ 0.49873232  0.49795581  0.48349826 ...  0.52484348  0.55190969
   0.56652555]
 [ 0.29215111  0.31673056  0.32804897 ...  0.14193956  0.15803132
   0.22869562]]

y_train:-
 [1 1 0 ... 0 0 1]

y_test:-
 [1 0 0 1 1 0 1 0 1 1 0 0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 1 1 0 1 0 0 1 0 1
1 1
 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 1 1 0 0 1 1 0 0 0 0 0 1 1 0
0 1
 0 1 1 0 0 0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 1 1 1 0 0 1 0 0 0 1 1 0 1 0 0
1 0
 1 1 1 0 0 1 0 0 0 0 1 1 1 1 1 1 0 0 0 1 0 1 1 0 0 1 1 1 1 0 1 0 1 0 0
0 0
 1 0 1 0 0 0 1 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 1 0 1 1 0 0 1 1 1 1 0 1
0 0
 1 0 0 0 1 0 0 1 1 1 0 0 1 1 0 0 0 0 0 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0
0 0
 1 0 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 1 1 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 1
0 1
 0 1 0 0 0 1 1 1 0 1 1 0 1 1 1 0 1 0 0 1 1 0 1 0 0 1 1 0 1 0 1 0 0 1 0
0 1
 1 0 0 0 1 1 0 1 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 1
1 1
 1 1 1 0 0 0 0 1 0 1 0 1 0 0 0 1 0 0 1 1 0 0 0 1 1 1 0 0 1 0 0 0 0 0 1
0 0
 0 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 1 0 1 1 0 0 0 0 0 1 1 1 0 1 1 1
0 0
 0 0 0 1 0 0 1 1 0 0 0 0 1 1 0 1 1 1 1 1 1 0 0 1 1 0 0 0 0 1 1 1 0 1 0
1 1
 1 0 0 1 1 1 0 1 1 1 1 0 1 0 1 0 1 0 0 1 0 1 1 1 1 0 0 0 1 1 1 0 0 0 1
1 0
 0 1 0 1 0 1 0 0 0 1 1 1 1 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1
0 0
 0 0 1 1 1 0 0 1 0 1 1 0 1 0 0 0 1 0 1 1 1 0 0 0 0 1 1 0 0 1 1 0 1 1 0
1 0
 1 0 1 0 0 0 1 0 1 1 1 0 1 0 0 0 0 0 1 0 0 1 1 1 0 1 0 0 0 1 1 0 0 0 0
0 0
 1 0 0 1 0 0 1 1 0 0 1 0 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 0 0 0 0 1 0
0 0
 0 0 1 1 1 1 0 1 1 1]

X_train: (2553, 40000)
X_test: (639, 40000)
y_train: (2553,)
y_test: (639,)
```

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression

# Creating a logistic regression model and fitting it to the training d
ata
lr = LogisticRegression(random_state=42, max_iter=1500)
lr.fit(X_train, y_train)
# Making predictions on the testing data
y_pred_lr = lr.predict(X_test)
print('Predicted and actual values on y_test')
print(np.concatenate((y_pred_lr.reshape(len(y_pred_lr),1), y_test.resha
pe(len(y_test),1)),1))
Predicted and actual values on y_test
[[0 1]
 [0 0]
 [0 0]
 ...
 [0 1]
 [1 1]
 [1 1]]
# Calculating the accuracy
accuracy_lr = accuracy_score(y_test, y_pred_lr)
print('Accuracy:', accuracy_lr)

# Printing the confusion matrix
cm_lr = confusion_matrix(y_test, y_pred_lr)
print('Confusion Matrix:')
print(cm_lr)

# Graphical display of the confusion matrix
ConfusionMatrixDisplay(cm_lr).plot()
plt.show()
Accuracy: 0.837245696400626
Confusion Matrix:
[[322  39]
 [ 65 213]]
```



KNN

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
print('Predicted vs actual values on y_test')
print(np.concatenate((y_pred_knn.reshape(len(y_pred_knn),1), y_test.res
hape(len(y_test),1)),1))
```

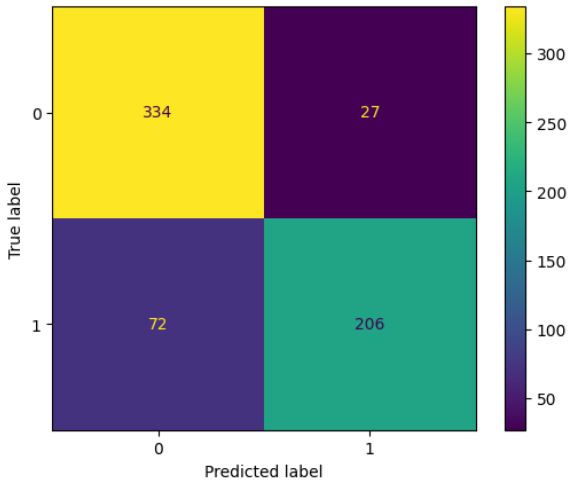
```
Predicted vs actual values on y_test
[[0 1]
 [0 0]
 [0 0]
 ...
 [1 1]
 [1 1]
 [0 1]]
```

```
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print('Accuracy:', accuracy_knn)
```

```
cm_knn = confusion_matrix(y_test, y_pred_knn)
print('Confusion Matrix:')
print(cm_knn)
```

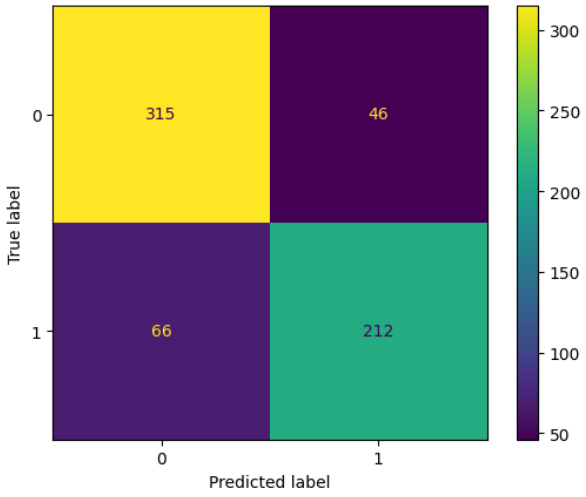
```
ConfusionMatrixDisplay(cm_knn).plot()
plt.show()
```

```
Accuracy: 0.8450704225352113
Confusion Matrix:
[[334  27]
 [ 72 206]]
```



SVM

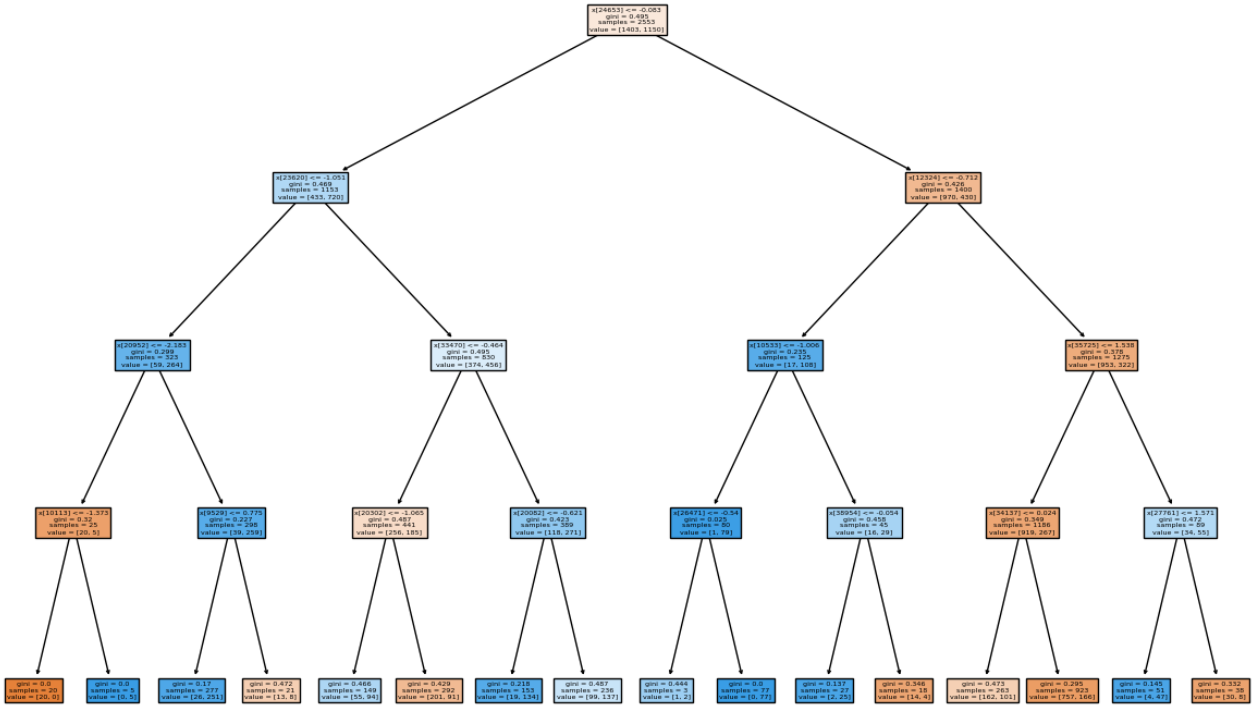
```
from sklearn.svm import SVC
svm = SVC(kernel='linear', random_state=42)
svm.fit(X_train, y_train)

y_pred_svm = svm.predict(X_test)
print('Predicted vs actual values on y_test')
print(np.concatenate((y_pred_svm.reshape(len(y_pred_svm),1), y_test.res
hape(len(y_test),1)),1))
Predicted vs actual values on y_test
[[1 1]
 [0 0]
 [0 0]
 ...
 [0 1]
 [1 1]
 [1 1]]


accuracy_svm = accuracy_score(y_test, y_pred_svm)
print('Accuracy:', accuracy_svm)
cm_svm = confusion_matrix(y_test, y_pred_svm)
print('Confusion Matrix:')
print(cm_svm)
ConfusionMatrixDisplay(cm_svm).plot()
plt.show()

Accuracy: 0.8247261345852895
Confusion Matrix:
[[315  46]
 [ 66 212]]
```



DECISION TREE

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(max_depth=4, random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
print('Predicted vs actual values on y_test')
print(np.concatenate((y_pred_dt.reshape(len(y_pred_dt),1), y_test.resha
pe(len(y_test),1)),1))
Predicted vs actual values on y_test
[[0 1]
 [0 0]
 [0 0]
 ...
 [0 1]
 [1 1]
 [0 1]]
from sklearn import tree

# Plotting the Decision tree
print('\nDecision Tree:- ')
plt.figure(figsize=(15,10))
tree.plot_tree(dt, filled=True)


Decision Tree:-
[Text(0.5, 0.9, 'x[24653] <= -0.083\ngini = 0.495\nsamples = 2553\nvalue =
[1403, 1150]'),
 Text(0.25, 0.7, 'x[23620] <= -1.051\ngini = 0.469\nsamples = 1153\nvalue =
[433, 720]'),
 Text(0.125, 0.5, 'x[20952] <= -2.183\ngini = 0.299\nsamples = 323\nvalue =
[59, 264]'),
 Text(0.0625, 0.3, 'x[10113] <= -1.373\ngini = 0.32\nsamples = 25\nvalue =
[20, 5]'),
 Text(0.03125, 0.1, 'gini = 0.0\nsamples = 20\nvalue = [20, 0]'),
 Text(0.09375, 0.1, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.1875, 0.3, 'x[9529] <= 0.775\ngini = 0.227\nsamples = 298\nvalue =
[39, 259]'),
 Text(0.15625, 0.1, 'gini = 0.17\nsamples = 277\nvalue = [26, 251]'),
 Text(0.21875, 0.1, 'gini = 0.472\nsamples = 21\nvalue = [13, 8]'),
 Text(0.375, 0.5, 'x[33470] <= -0.464\ngini = 0.495\nsamples = 830\nvalue =
[374, 456]'),
 Text(0.3125, 0.3, 'x[20302] <= -1.065\ngini = 0.487\nsamples = 441\nvalue
= [256, 185]'),
 Text(0.28125, 0.1, 'gini = 0.466\nsamples = 149\nvalue = [55, 94]'),
 Text(0.34375, 0.1, 'gini = 0.429\nsamples = 292\nvalue = [201, 91]'),
 Text(0.4375, 0.3, 'x[20082] <= -0.621\ngini = 0.423\nsamples = 389\nvalue
= [118, 271]'),
 Text(0.40625, 0.1, 'gini = 0.218\nsamples = 153\nvalue = [19, 134]'),
 Text(0.46875, 0.1, 'gini = 0.487\nsamples = 236\nvalue = [99, 137]'),
 Text(0.75, 0.7, 'x[12324] <= -0.712\ngini = 0.426\nsamples = 1400\nvalue =
[970, 430]'),
 Text(0.625, 0.5, 'x[10533] <= -1.006\ngini = 0.235\nsamples = 125\nvalue =
[17, 108]'),
```

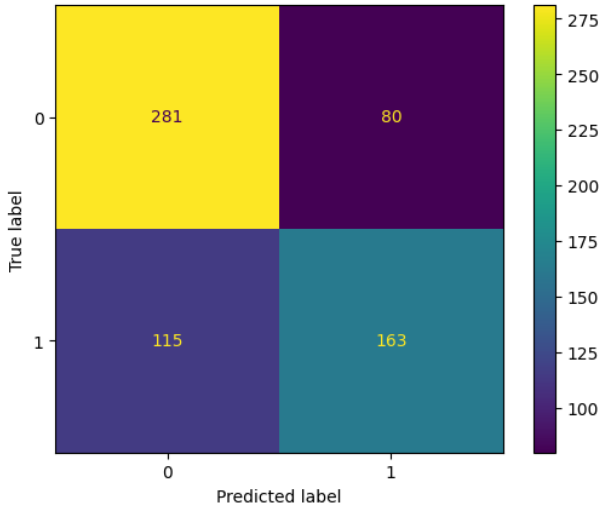
```
 Text(0.5625, 0.3, 'x[26471] <= -0.54\ngini = 0.025\nsamples = 80\nvalue =
[1, 79]'),
 Text(0.53125, 0.1, 'gini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.59375, 0.1, 'gini = 0.0\nsamples = 77\nvalue = [0, 77]'),
 Text(0.6875, 0.3, 'x[38954] <= -0.054\ngini = 0.458\nsamples = 45\nvalue =
[16, 29]'),
 Text(0.65625, 0.1, 'gini = 0.137\nsamples = 27\nvalue = [2, 25]'),
 Text(0.71875, 0.1, 'gini = 0.346\nsamples = 18\nvalue = [14, 4]'),
```



```
 Text(0.875, 0.5, 'x[35725] <= 1.538\ngini = 0.378\nsamples = 1275\nvalue =
[953, 322]'),
 Text(0.8125, 0.3, 'x[34137] <= 0.024\ngini = 0.349\nsamples = 1186\nvalue
= [919, 267]'),
 Text(0.78125, 0.1, 'gini = 0.473\nsamples = 263\nvalue = [162, 101]'),
 Text(0.84375, 0.1, 'gini = 0.295\nsamples = 923\nvalue = [757, 166]'),
 Text(0.9375, 0.3, 'x[27761] <= 1.571\ngini = 0.472\nsamples = 89\nvalue =
[34, 55]'),
 Text(0.90625, 0.1, 'gini = 0.145\nsamples = 51\nvalue = [4, 47]'),
 Text(0.96875, 0.1, 'gini = 0.332\nsamples = 38\nvalue = [30, 8]')]
```

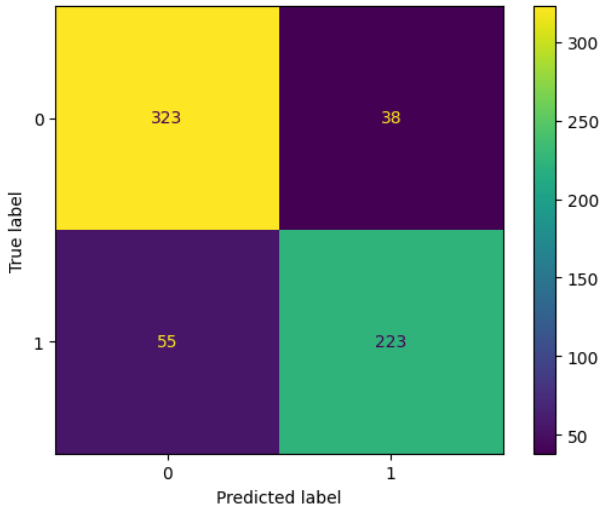
```
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print('Accuracy:', accuracy_dt)

cm_dt = confusion_matrix(y_test, y_pred_dt)
print('Confusion Matrix:')
print(cm_dt)

ConfusionMatrixDisplay(cm_dt).plot()
plt.show()
```

```
Accuracy: 0.6948356807511737
Confusion Matrix:
[[281  80]
 [115 163]]
```



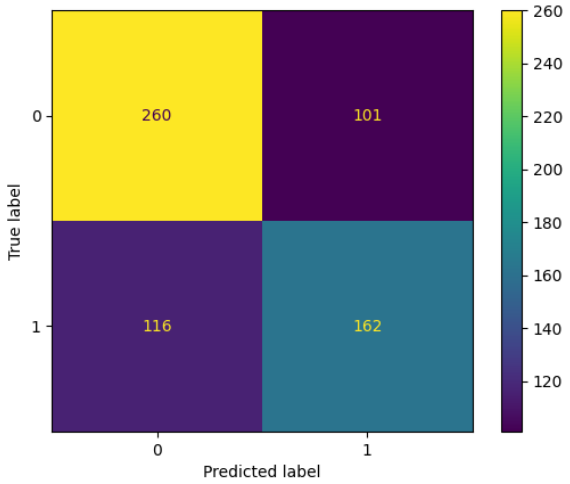
Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print('Predicted vs actual values on y_test')
print(np.concatenate((y_pred_rf.reshape(len(y_pred_rf),1), y_test.resha
pe(len(y_test),1)),1))
Predicted vs actual values on y_test
[[1 1]
 [1 0]
 [0 0]
 ...
 [1 1]
 [1 1]
 [0 1]]
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print('Accuracy:', accuracy_rf)

cm_rf = confusion_matrix(y_test, y_pred_rf)
print('Confusion Matrix:')
print(cm_rf)

ConfusionMatrixDisplay(cm_rf).plot()
plt.show()
Accuracy: 0.8544600938967136
Confusion Matrix:
[[323  38]
 [ 55 223]]
```



Naive Bayes

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
print('Predicted vs actual values on y_test')
print(np.concatenate((y_pred_nb.reshape(len(y_pred_nb),1), y_test.resha
pe(len(y_test),1)),1))
Predicted vs actual values on y_test
[[0 1]
 [1 0]
 [0 0]
 ...
 [1 1]
 [0 1]
 [0 1]]

accuracy_nb = accuracy_score(y_test, y_pred_nb)
print('Accuracy:', accuracy_nb)

cm_nb = confusion_matrix(y_test, y_pred_nb)
print('Confusion Matrix:')
print(cm_nb)

ConfusionMatrixDisplay(cm_nb).plot()
plt.show()
Accuracy: 0.6604068857589984
Confusion Matrix:
[[260 101]
 [116 162]]
```



Accuracy Comparision

```
import pandas as pd

accuracies = {'KNN': accuracy_knn,
              'Logistic Regression': accuracy_lr,
              'SVM': accuracy_svm,
              'Decision Tree': accuracy_dt,
              'Random Forest': accuracy_rf,
              'Naive Bayes': accuracy_nb}

df = pd.DataFrame(accuracies.items(), columns=['Model', 'Accuracy'])

df = df.sort_values('Accuracy', ascending=False)

print(df)

max_accuracy = df.iloc[0]['Accuracy']
best_model = df.iloc[0]['Model']

print("\nMax accuracy is given by ML model", best_model, "with accuracy
:", max_accuracy)
```

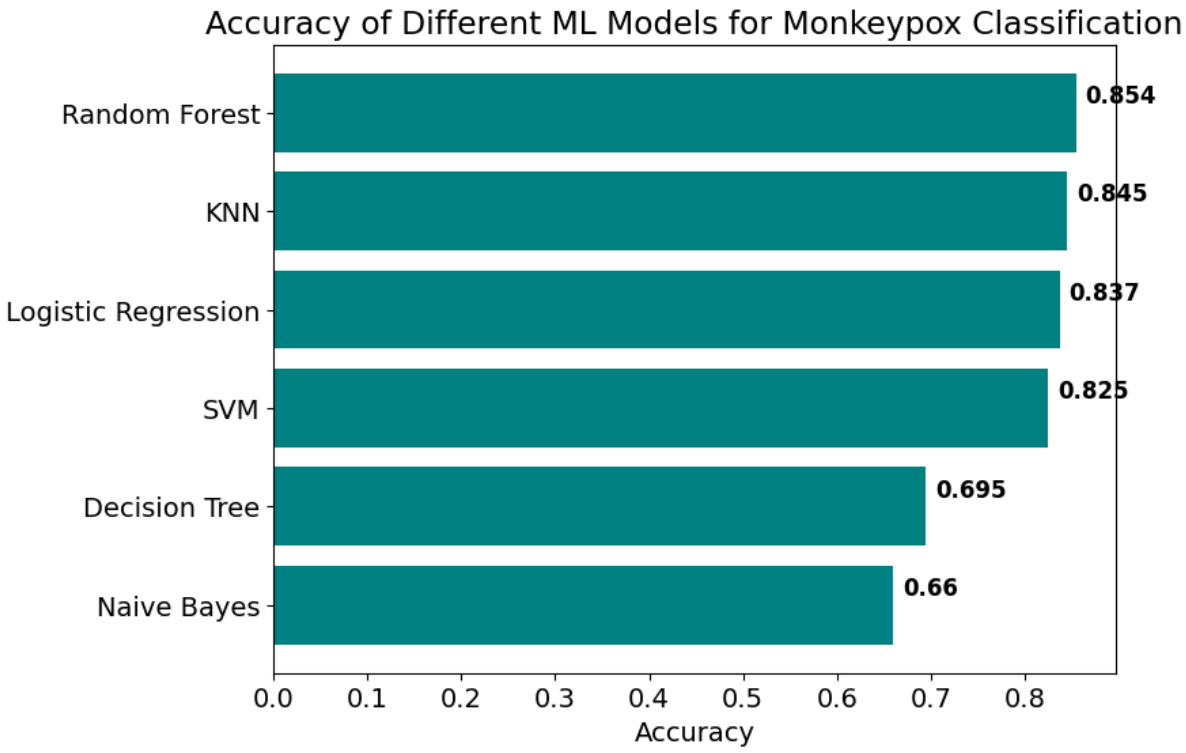
```
Model  Accuracy
4        Random Forest  0.854460
0                  KNN  0.845070
1  Logistic Regression  0.837246
2                  SVM  0.824726
3        Decision Tree  0.694836
5          Naive Bayes  0.660407

Max accuracy is given by ML model Random Forest with accuracy:
0.8544600938967136
```

CHAPTER 5

RESULTS

ACCURACY COMPARISION FROM DIFFERENT MODELS

ML model	Accuracy
KNN	0.8450704225352113
LOGISTIC REGRESSION	0.837245696400626
DECISION TREE	0.6948356807511737
RANDOM FOREST	0.8544600938967136
SVM	0.8247261345852895
NAÏVE BAYES	0.6604068857589984



CHAPTER 6

CONCLUSION AND FUTURE SCOPE

6.1 CONCLUSION

Finally, after performing all the steps needed to get the results from preparation to preprocessing to performing the models (Logistic regression, KNN, Decision Tree, Random Forest, SVM and Naïve Bayes) we conclude that the Random Forest model with 85.44600938967136 percent accuracy performs slightly better than KNN of 84.50704225352113 percent accuracy.

6.2 FUTURE SCOPE

In order to further improve the accuracy of the classification model for monkey pox detection, deep learning techniques such as convolutional neural networks (CNN) and recurrent neural networks (RNN) can be explored. Additionally, transfer learning approaches can be leveraged to improve feature extraction and classification accuracy. Incorporating additional data sources such as patient demographics and symptoms can also enhance the accuracy of the model and aid in early detection and treatment of the disease.

REFERENCES

[1]. https://www.who.int/emergencies/situations/monkeypox-oubreak-2022

[2]. 1.4. Support Vector Machines — scikit-learn 1.2.2 documentation

[3]. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9908738/

[4]. https://www.nature.com/articles/s41591-023-02225-7

[5]. https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

[6]. What is the k-nearest neighbors algorithm? | IBM

[7]. What is Random Forest? | IBM

[8]. What is a Decision Tree | IBM

[9]. What is Logistic regression? | IBM

[10]. https://www.kaggle.com/datasets/nafin59/monkeypox-skin-lesion-dataset