DETECTING PARKINSON’S DISEASE USING MACHINE LEARNING

**IBM-Project-10817-1659232384**

**TEAM ID: PNT2022TMID00199**

**Submitted by**

**SYLVIYA J 312319104173**

**SANJANA V 312319104141**

**SOWMYA M 312319104165**

**VIDHYALAKSHMI R G 312319104183**

**In partial fulfilment for the award of the degree of**

**BACHELOR OF ENGINEERING**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

**ST. JOSEPH’S COLLEGE OF ENGINEERING**

**CHENNAI – 600 119**

**INDEX**

# INTRODUCTION

* 1. Project Overview
  2. Purpose

# LITERATURE SURVEY

* 1. Existing solutions
  2. References
  3. Problem Statement Definition

# IDEATION & PROPOSED SOLUTION

* 1. Empathy Map Canvas
  2. Ideation & Brainstorming
  3. Proposed Solution
  4. Problem Solution fit

# REQUIREMENT ANALYSIS

* 1. Functional requirement
  2. Non-Functional requirements

# PROJECT DESIGN

* 1. Data Flow Diagrams
  2. Solution & Technical Architecture
  3. User Stories

# PROJECT PLANNING & SCHEDULING

* 1. Sprint Planning & Estimation
  2. Sprint Delivery Schedule
  3. Reports from JIRA

# CODING & SOLUTIONING (Explain the features added in the project along with code)

* 1. Feature 1
  2. Feature 2
  3. Database Schema

# TESTING

* 1. Test Cases
  2. User Acceptance Testing

# RESULTS

* 1. Performance Metrics

# ADVANTAGES & DISADVANTAGES

1. **CONCLUSION**

# FUTURE SCOPE

1. **APPENDIX**

Source Code

GitHub & Project Demo Link

**DETECTING PARKINSON’S DISEASE USING MACHINE LEARNING**

1. **INTRODUCTION**

# Project Overview

The main reason for Parkinson’s Disease is due to abnormal loss of neurons. The researchers found that the drawing speed was slower and the pen pressure is low among the patients. One of the indications of Parkinson’s is tremors and rigidity in the muscles, making it difficult to draw smooth spirals and waves. It is possible to detect Parkinson’s disease using the drawings alone instead of measuring the speed and pressure of the pen on paper.

This project aims at quantifying the visual appearance using the Histogram of Oriented Gradients method, where the drawings are trained and tested using Machine Learning Model and they are further classified using Convolutional Neural Network. Later, we use the Random Forest classifier to automatically detect Parkinson’s disease in hand-drawn images of spirals and waves. Here the user interacts with the User Interface (UI) to upload the image as input. The uploaded image is analyzed by the model which is integrated. Once the model analyzes the uploaded image, the prediction is showcased on the UI and OpenCV window.

# Purpose

❖ To provide an efficient solution to detect the presence of Parkinson’s Disease in a patient using Deep Learning Architecture.

❖ Develop a Histogram Oriented Gradient (HOG) Architecture to boost the accuracy of prediction.

❖ Reduces the time required for manual classification and eliminates the human error.

❖ Due to Early detection of the disease further medication cost will be reduced.

# LITERATURE SURVEY

* 1. **Existing problem**

Parkinson's disease or PD is among the most common neurodegenerative disorders which, due to loss of neurons, leads to the deterioration of motor and non- motor functions like tremors, shaking, slow movement, impaired balance and depression. It is a slow developing disorder and there is no definitive treatment for it; but its symptoms can be reduced with the help of specific drugs. Various methods have been introduced in order to detect PD in its early stages. Magnetic resonance imaging (MRI) is an effective and efficient method since it is an inexpensive and non-invasive tool which produces high-resolution images. In this paper, early diagnosis of PD has been carried out using different CNN models. Image enhancement has been performed on the magnetic resonance (MR) images in the dataset. After extracting the region of interest (ROI), in this case the substantia nigra (SN) region, a custom CNN model and VGG-16 are implemented for automatically differentiating SN in image and no SN in the image. Then performed binary classification to accurately discriminate between PD and control subjects by implementing deep learning models ResNet-34, VGG-19, Res Net-50 and improved Alex Net. At the end, after comparing the accuracies and losses of different models, it can be concluded that the ResNet-50 model depicted the best performance for this purpose.

# References

1. Anupama Bhan, Sona Kapoor, Manan Gulati," Diagnosing Parkinson's disease in Early Stages using Image Enhancement, ROI Extraction and Deep Learning Algorithms".[2021]

2. Beaumon, P. Onoma, M. Rimlinger, D. Broggio, P. Caldeira Ideias and D.Franck,"Age-specific experimental and computational calibration of thyroid in vivo monitoring",IEEE Transactions on Radiation and Plasma Medical Sciences, Vol: 2829931,2019 3. Chenchen Qin, Yi Wang, Na Wang, Min Xu, Junxiong Yu, Xiao Luo, Xin Yang. Tianfu Wang, Anhua Li, and Dong Ni" Deeply-Supervised Networks with Threshold Loss for Cancer Detection in Automated Breast Ultrasound [Vol 0278-0062,2019]

3. David Russell, Jose M. Anton-Rodriguez, Peter Julyan, Ibrahim Djoukhadar, D. Gareth Evans, Alan Jackson, and Julian C. Matthews," Comparison of a Standard Resolution PET-CT Scanner With an HRRT Brain Scanner for Imaging Small Tumors Within the Head", IEEE Transactions on radiation and plasma medical sciences, vol. 3. no. 4, july 2019.

4. Jin Tae Kwak, Shekoofeh Azizi, Shararch Bayat, Pingkun Yan, Amir Tahmasebi, Sheng Xu, Baris Turkbey, Peter Choyke, Peter Pinto, Bradford Wood, Parvin Mousavi, Purang Abolmaesumi," Deep Recurrent

Neural Networks for Prostate Cancer Detection: Analysis of Temporal Enhanced Ultrasound IEEE Transactions on Medical Imaging "[Vol.no:0278-0062,2018]

# Problem Statement Definition

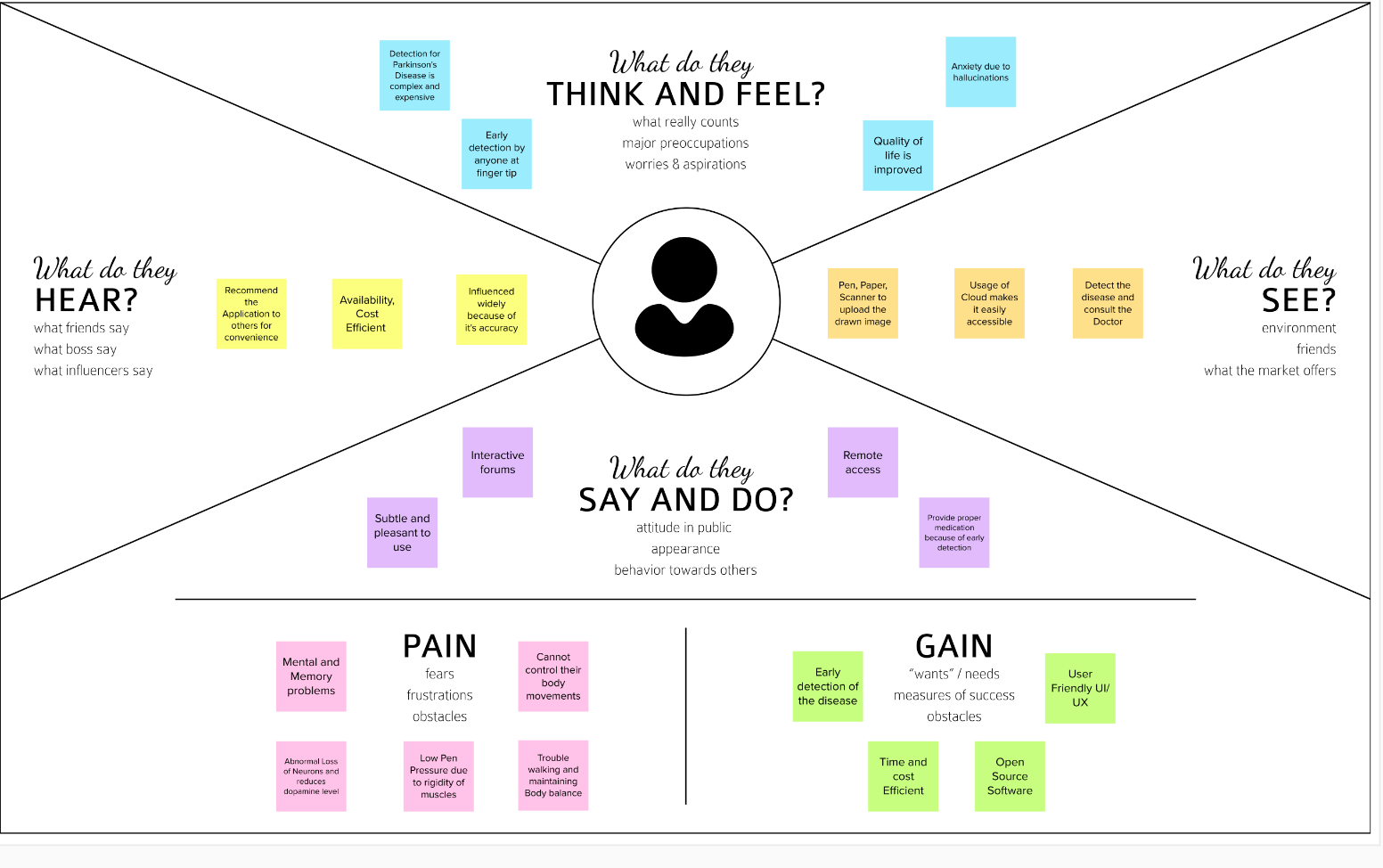
* The problem statement aims at detecting the presence of Parkinson's disease at an

early stage with the help of hand drawings to provide with essential medication to improve the expectancy of life.

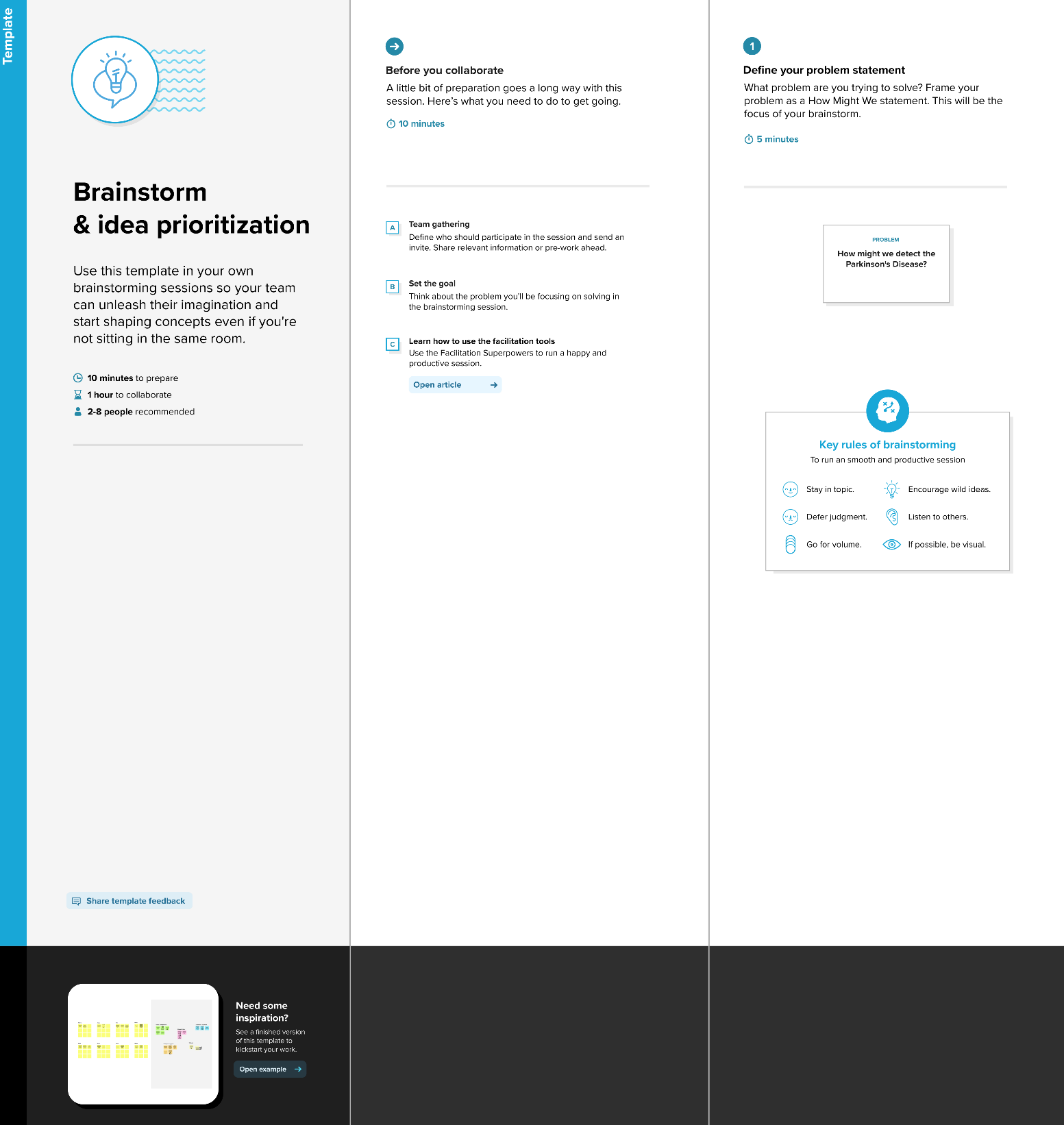
* User needs a way to know whether they are affected by Parkinson's disease so that they can consult for early medication.
* User with symptoms who needs to detect the disease to overcome suffering.

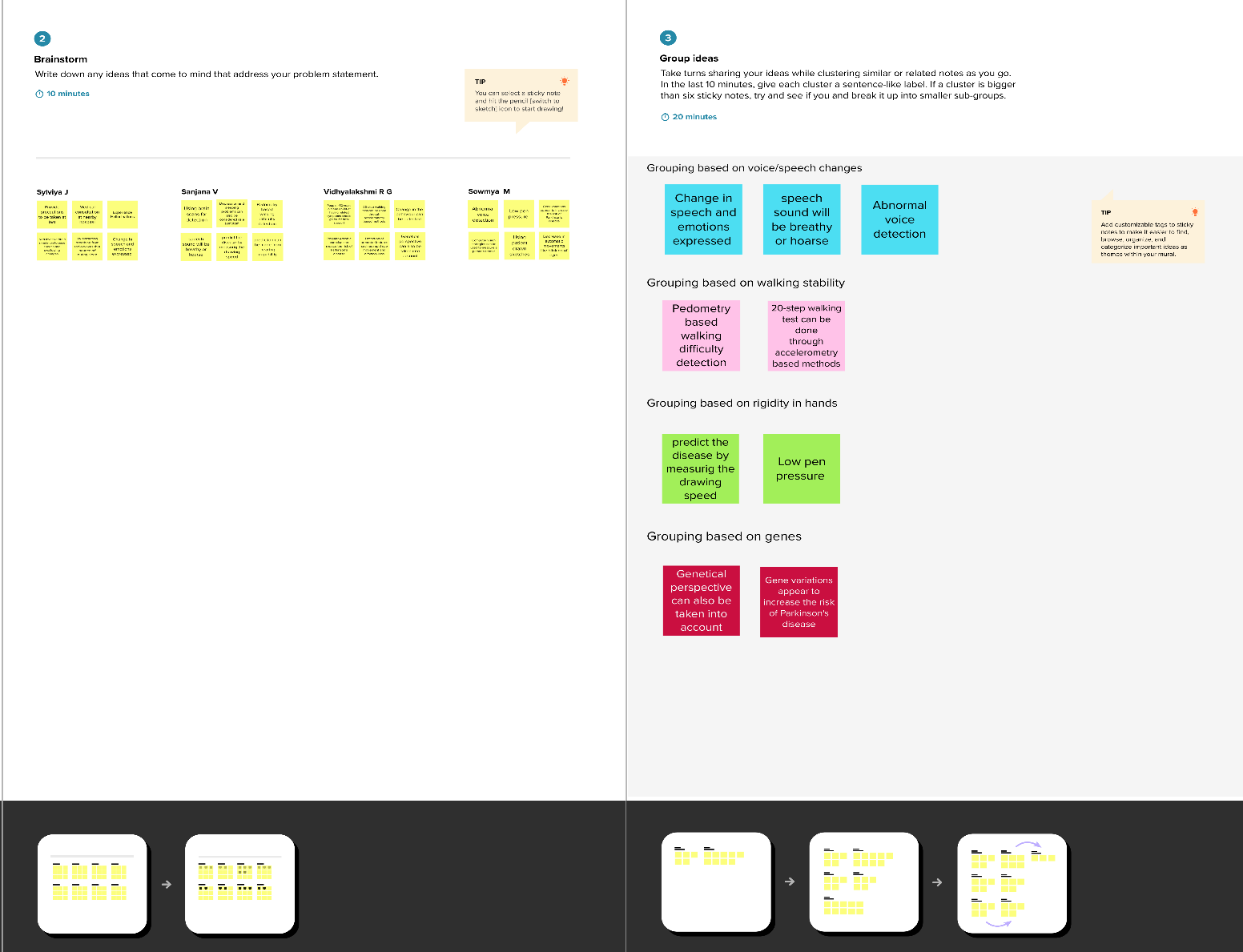
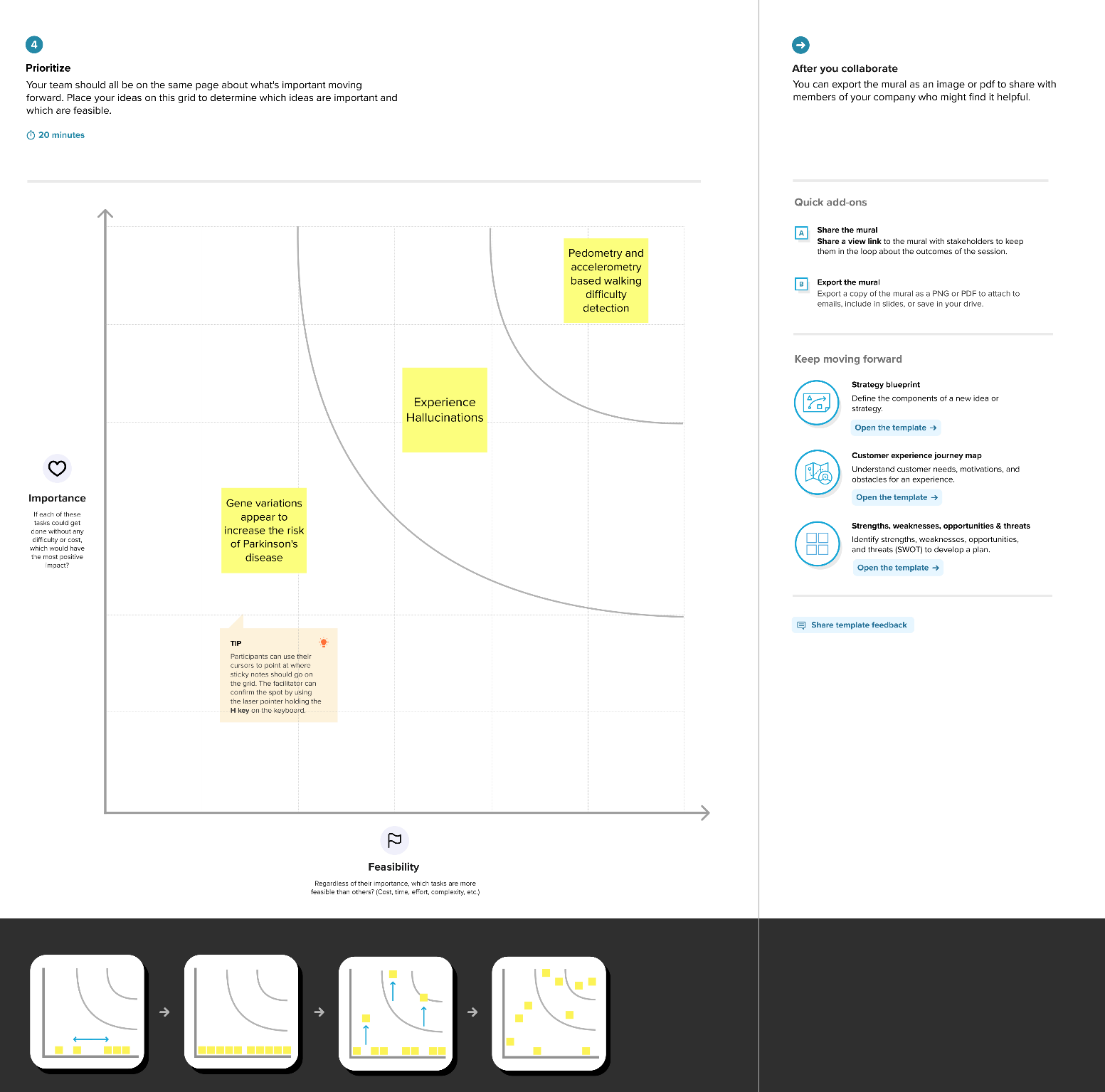
# IDEATION & PROPOSED SOLUTION

* 1. **Empathy Map Canvas**



# Ideation & Brainstorming

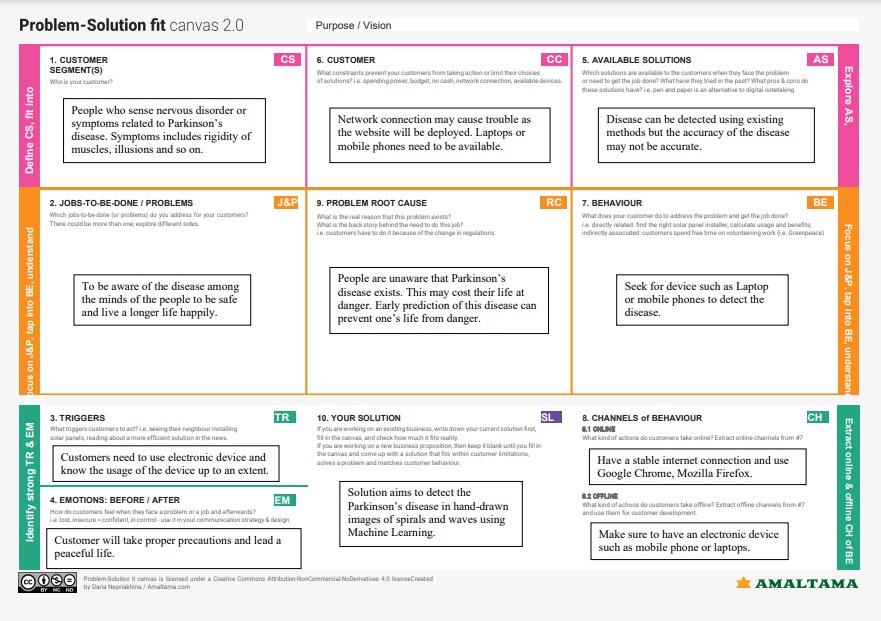


* 1. **Proposed Solution**

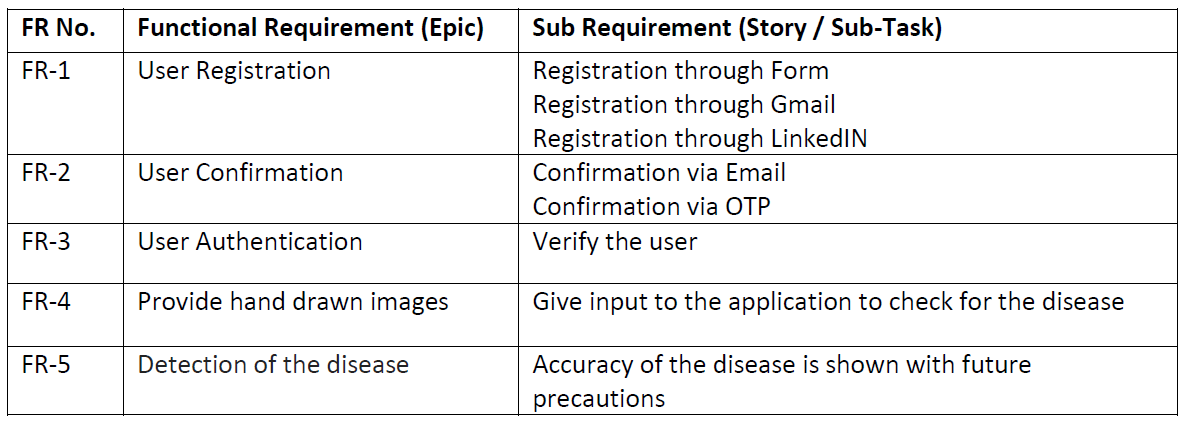
|  |  |  |
| --- | --- | --- |
| **S. No.** | **Parameter** | **Description** |
| 1. | Problem Statement (Problem to be solved) | Early and Automatic detection of Parkinson’s Disease in hand drawn images. |
| 2 | Idea / Solution description | Detection is done by using Histogram of Oriented Gradients (HOG) image descriptor along with a Random Forest classifier. |
| 3 | Novelty / Uniqueness | In this method, the approach to predict Parkinson's disease from hand-drawn wave and spiral images using computer vision and machine learning techniques has been recommended. The previous methods have their constraints. |
| 4 | Social Impact / Customer Satisfaction | People can detect the disease at a very early stage and improve the quality of living. They can take proper precautions and lead a healthy and safe life. |
| 5 | Business Model (Revenue Model) | It is cost efficient as it is a Software as a Service Platform. People need not spend much money to detect the disease. |
| 6 | Scalability of the Solution | Better execution in accuracy, sensitivity, and specificity as well as in system design flexibility. |

# Problem Solution fit

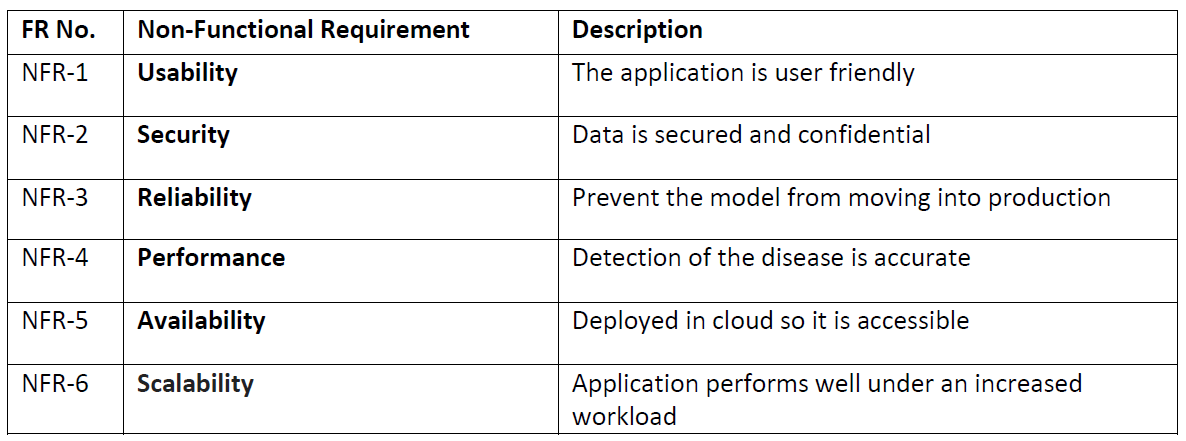


1. **REQUIREMENT ANALYSIS**

# Functional requirement

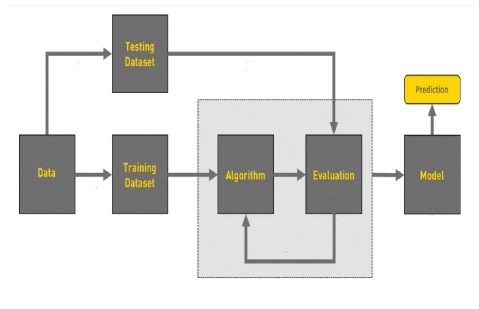


* 1. **Non-Functional requirements**



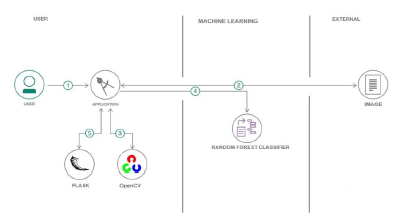
# PROJECT DESIGN

* 1. **Data Flow Diagrams**



# Solution & Technical Architecture

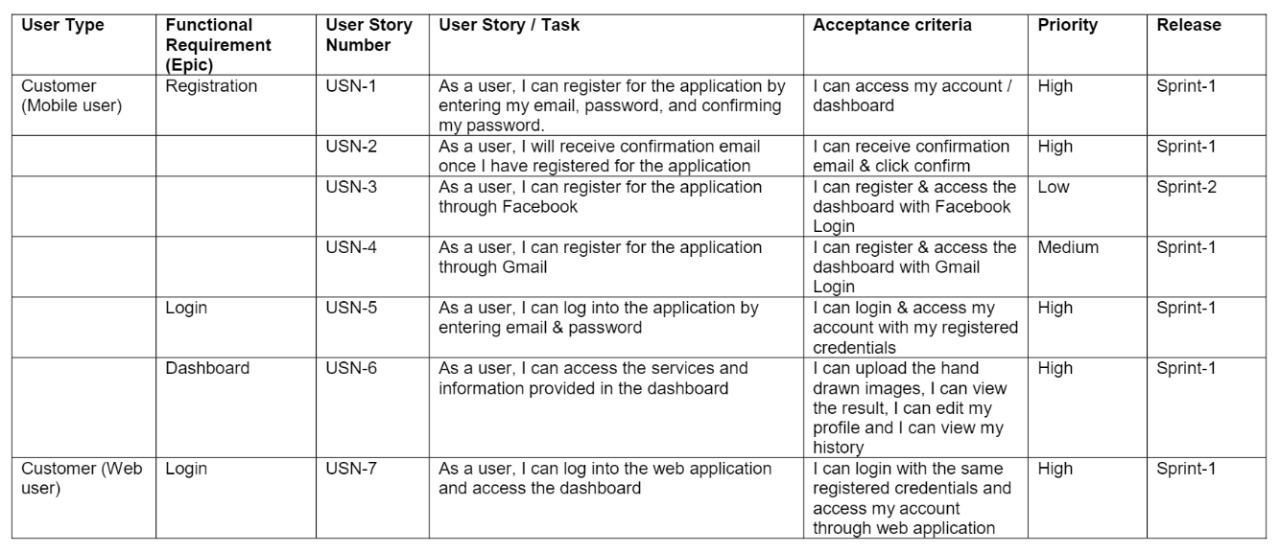
* + 1. **Solution Architecture**

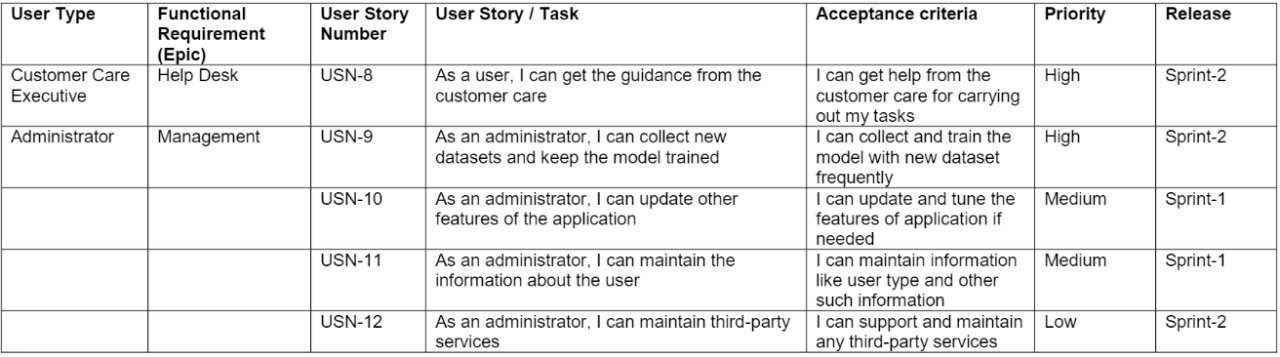


# Technical Architecture



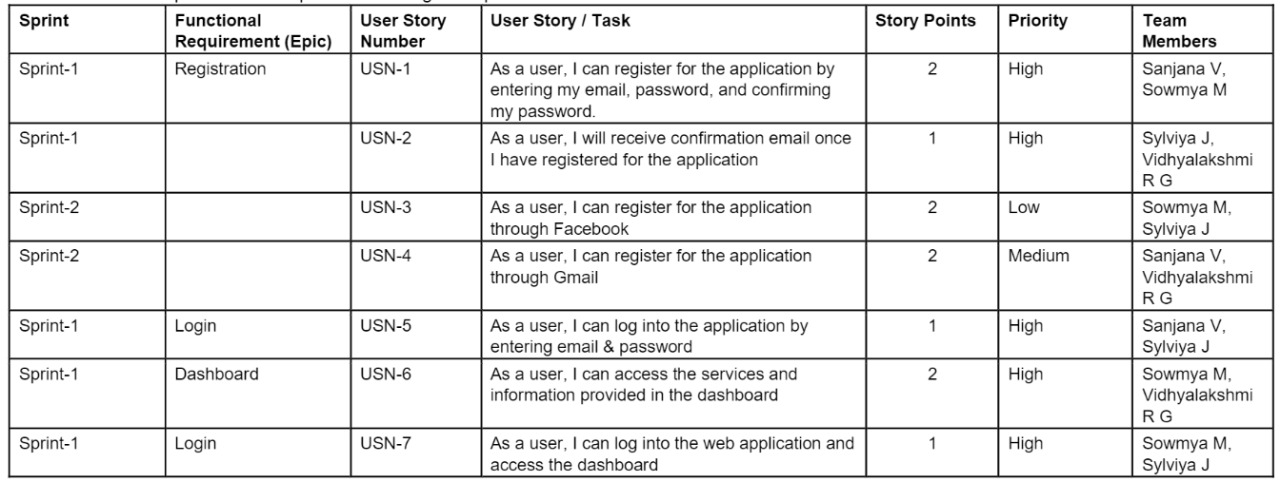
* 1. **User Stories**

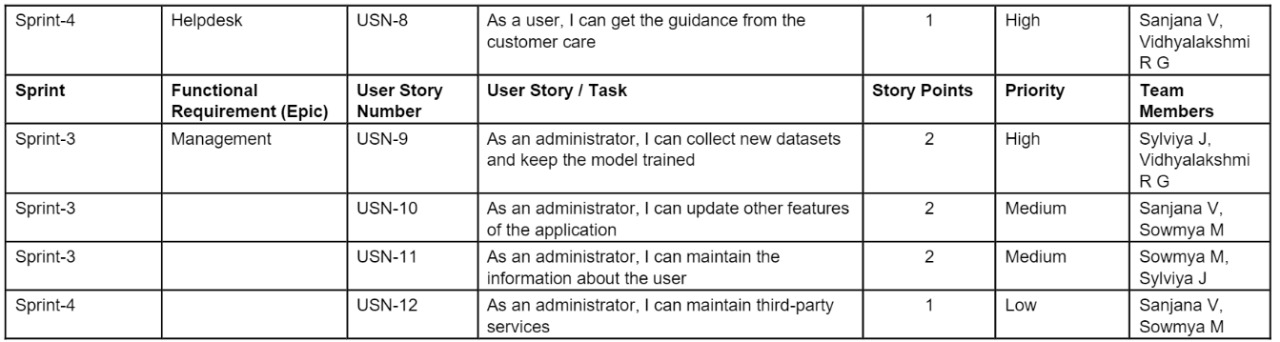




# PROJECT PLANNING & SCHEDULING

* 1. **Sprint Planning & Estimation**

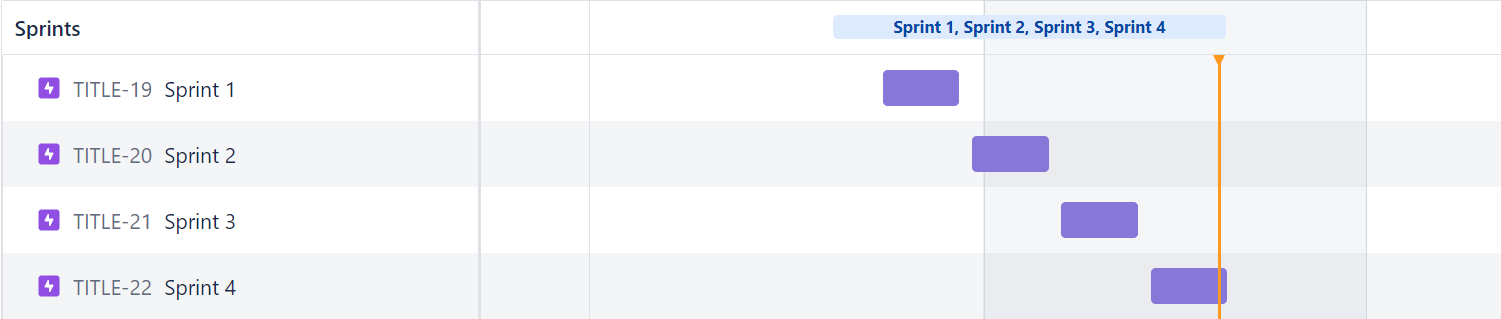




# Sprint Delivery Schedule



* 1. **Reports from JIRA**



# CODING & SOLUTIONING

* 1. **Feature 1 Home page:**

Parkinson’s Detection App is a responsive web application which detects the disease of the uploaded drawn image. The home page contains information about the application. It also contains the home tab to upload the hand drawn image.

# Home.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="static/style1.css">

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.3/css/all.min.css">

<script class="jsbin" src="https://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>

<title>Home</title>

<style>

#log{

float: right;

}

.navbar {

overflow: hidden;

background-color: #333;

padding-top: 0px;

}

.navbar a {

float: left;

font-size: 16px;

color: white;

text-align: center;

padding: 14px 16px;

text-decoration: none;

}

.dropdown {

float: left;

overflow: hidden;

}

.dropdown .dropbtn {

font-size: 16px;

border: none;

outline: none;

color: white;

padding: 14px 16px;

background-color: inherit;

font-family: inherit;

margin: 0;

}

.navbar a:hover, .dropdown:hover .dropbtn {

background-color: teal;

}

.dropdown-content {

display: none;

position: absolute;

background-color: #f9f9f9;

min-width: 160px;

box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);

z-index: 1;

}

.dropdown-content a {

float: none;

color: black;

padding: 12px 16px;

text-decoration: none;

display: block;

text-align: left;

}

.dropdown-content a:hover {

background-color: #ddd;

}

.dropdown:hover .dropdown-content {

display: block;

}

</style>

</head>

<body>

<div class="navbar">

<a href="/index">Home</a>

<a href="/info">Information</a>

<a href="/" id="log">Logout</a>

</div>

<br><br><br><br>

<form action = "/predict" method = "post" enctype="multipart/form-data">

<div class="file-upload">

<button class="file-upload-btn" type="button" onclick="$('.file-upload-input').trigger( 'click' )">Add Image</button>

<div class="image-upload-wrap">

<input class="file-upload-input" type='file' name='file' onchange="readURL(this);" accept="image/\*" />

<div class="drag-text">

<h3>Drag and drop a file or select add Image</h3>

</div>

</div>

<div class="file-upload-content">

<img class="file-upload-image" src="#" alt="your image" />

<div class="image-title-wrap">

<button type="button" onclick="removeUpload()" class="remove-image">Remove <span class="image-title">Uploaded Image</span></button>

</div>

</div>

<br>

<button class="file-upload-btn" type="submit" id="bt">predict</button>

</div>

</form>

<script>

function readURL(input) {

if (input.files && input.files[0]) {

var reader = new FileReader();

reader.onload = function(e) {

$('.image-upload-wrap').hide();

$('.file-upload-image').attr('src', e.target.result);

$('.file-upload-content').show();

$('.image-title').html(input.files[0].name);

};

reader.readAsDataURL(input.files[0]);

}

else {

removeUpload();

}

}

function removeUpload() {

$('.file-upload-input').replaceWith($('.file-upload-input').clone());

$('.file-upload-content').hide();

$('.image-upload-wrap').show();

}

$('.image-upload-wrap').bind('dragover', function () {

$('.image-upload-wrap').addClass('image-dropping');

});

$('.image-upload-wrap').bind('dragleave', function () {

$('.image-upload-wrap').removeClass('image-dropping');

});

</script>

</body>

</html>

# Feature

# Prediction page:

The user will add the hand drawn image in the home page and press the predict button. Then the page will redirect to a new page and provide information whether the person is affected by the disease or not

# app.py

# from flask import Flask,redirect, request, url\_for, render\_template

# import sqlite3 as sql

# import os.path

# import pickle

# import cv2

# from skimage import feature

# import requests

# import csv

# import numpy as np

# items = []

# with open('parkinson\_test\_input.csv', newline='') as csvfile:

# data = list(csv.reader(csvfile))

# print(len(data))

# API\_KEY = "y78YjqQtl67p9cXH197lt6YsP3Fu1w7EZdGs1llpfxEv"

# token\_response = requests.post('https://iam.cloud.ibm.com/identity/token', data={"apikey":

# API\_KEY, "grant\_type": 'urn:ibm:params:oauth:grant-type:apikey'})

# mltoken = token\_response.json()["access\_token"]

# header = {'Content-Type': 'application/json', 'Authorization': 'Bearer ' + mltoken}

# app=Flask(\_\_name\_\_)

# class DB:

# def \_init\_(self, name = 0):

# self.\_name = name

# self.\_lid = 0

# self.\_tot=0

# # getter method

# def get\_name(self):

# return self.\_name

# # setter method

# def set\_name(self, x):

# self.\_name = x

# def get\_lid(self):

# return self.\_lid

# # setter method

# def set\_lid(self, x):

# self.\_lid = x

# def get\_tot(self):

# return self.\_tot

# # setter method

# def set\_tot(self, x):

# self.\_tot = x

# obj=DB()

# @app.route('/')

# def home():

# try:

# lid=obj.get\_lid()

# print(lid)

# return render\_template('login.html',data=lid)

# except Exception as e:

# return render\_template('login.html')

# @app.route('/index')

# def index():

# # obj.\_lid=0

# return render\_template('index.html')

# @app.route('/info')

# def info():

# # obj.\_lid=0

# return render\_template('info.html')

# @app.route('/login')

# def login():

# # obj.\_lid=0

# return render\_template('login.html')

# @app.route('/predict',methods=["GET","POST"])

# def predict():

# if request.method == 'POST':

# f=request.files['file'] #requesting the file

# f.save(f.filename)

# # basepath=os.path.dirname(\_\_file\_\_)#storing the file directory

# # filepath=os.path.join(basepath, "uploads", f.filename)#storing the

# # f.save(filepath)#saving the file #Loading the saved model

# print("[INFO] Loading model...")

# model = pickle.loads (open('model.pkl', "rb").read()) #pre-process the image in the same manner we did earlier

# image=cv2.imread(f.filename)

# output = image.copy() # load the input image, convert it to grayscale, and resize

# output = cv2.resize(output, (128, 128))

# image= cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# image = cv2.resize(image, (200, 200))

# image = cv2.threshold(image, 0, 255,cv2.THRESH\_BINARY\_INV | cv2.THRESH\_OTSU) [1]

# # quantify the image and make predictions based on the extracted #features using the last trained Random Forest

# print('hi')

# features = feature.hog(image, orientations=9, pixels\_per\_cell=(10, 10), cells\_per\_block=(2, 2), transform\_sqrt=True, block\_norm="L1")

# print(features.shape)

# list = features.tolist()

# payload\_scoring = {"input\_data": [{"fields": data, "values": [list]}]}

# response\_scoring = requests.post('https://us-south.ml.cloud.ibm.com/ml/v4/deployments/246e322f-c88b-4818-a537-7ab7463bce8b/predictions?version=2022-11-18', json=payload\_scoring,

# headers={'Authorization': 'Bearer ' + mltoken})

# print("Scoring response")

# print(response\_scoring.json())

# preds=model.predict([features])

# print("dsa",preds)

# 

# pred=response\_scoring.json()

# print(pred)

# result=pred['predictions'][0]['values'][0][0]

# print(result)

# # ls=["healthy","parkinson"]

# # result=ls[preds[0]]

# ls=["healthy","parkinson"]

# res=ls[result]

# # obj.\_lid=0

# return render\_template('predict.html',d=res)

# @app.route("/login", methods = ["GET","POST"])

# def Login():

# l\_id = request.form["logname"]

# l\_pass = request.form["logpass"]

# tab=l\_id+l\_pass

# #print("name")

# if request.method == 'POST':

# print("name")

# l\_id = request.form["logname"]

# l\_pass = request.form["logpass"]

# tab=l\_id+l\_pass

# print(tab)

# obj.set\_name(tab)

# obj.set\_lid(l\_id)

# return redirect(f"/check")

# return render\_template('invalid.html',invalid='Please enter a valid data')

# @app.route('/sign')

# def sign():

# return render\_template('signup.html')

# @app.route("/regis",methods=["GET","POST"])

# def regis():

# u\_id = request.values.get("signu\_id")

# s\_pass = request.values.get("sign\_pass")

# print(u\_id)

# print(s\_pass)

# table\_name=u\_id+s\_pass

# print(table\_name)

# try:

# conn=sql.connect('main.db')

# print("Opened database successfully")

# create="CREATE TABLE "+table\_name+" (detail TEXT, cred TEXT)"

# conn.execute(create)

# #conn.execute("select \* from credential")

# print("Table created successfully")

# conn.close()

# return render\_template("success.html")

# except:

# # print("dsa")

# return render\_template("invalid.html", a="Username and password are already taken. Try another.")

# @app.route("/check",methods = ["GET","POST"])

# def cart():

# tab=obj.get\_name()

# lid=obj.get\_lid()

# print(lid)

# if request.method == 'GET':

# print(f"Your name is {tab}")

# try:

# con = sql.connect("main.db")

# con.row\_factory = sql.Row

# cur = con.cursor()

# a=f"select \* from {tab}"

# print(a)

# cur.execute(a)

# return render\_template('index.html',data=lid)

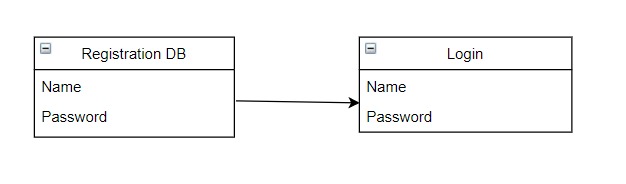
# except :

# return render\_template('invalid.html',a='Please enter a valid data')

# if \_\_name\_\_=="\_\_main\_\_":

# app.run(debug=True)

# Database Schema



1. **TESTING**

# Test Cases

|  |  |  |  |
| --- | --- | --- | --- |
| **Test case ID** | **Test Scenario** | **Expected Result** | **Status** |
| Login\_TC\_001 | Verify whether the user is valid or not. If valid, user enters the home page. | If valid, user enters the dashboard | Pass |
| ImageUpload\_TC\_002 | When the user uploads the image the, page is redirected for prediction | The Hand-drawn image should be processed and predicted by the model. | Pass |
| Prediction\_TC\_003 | Process the uploaded image with high accuracy | Display whether the user is affected or not by the disease. | Pass |
| Information\_TC\_004 | Verify the UI Elements in Information page | Information Page with good look and feel is displayed | Pass |
| Password\_TC\_005 | Categorize Strong/Weak password | Display Strong/Weak Password | Fail |

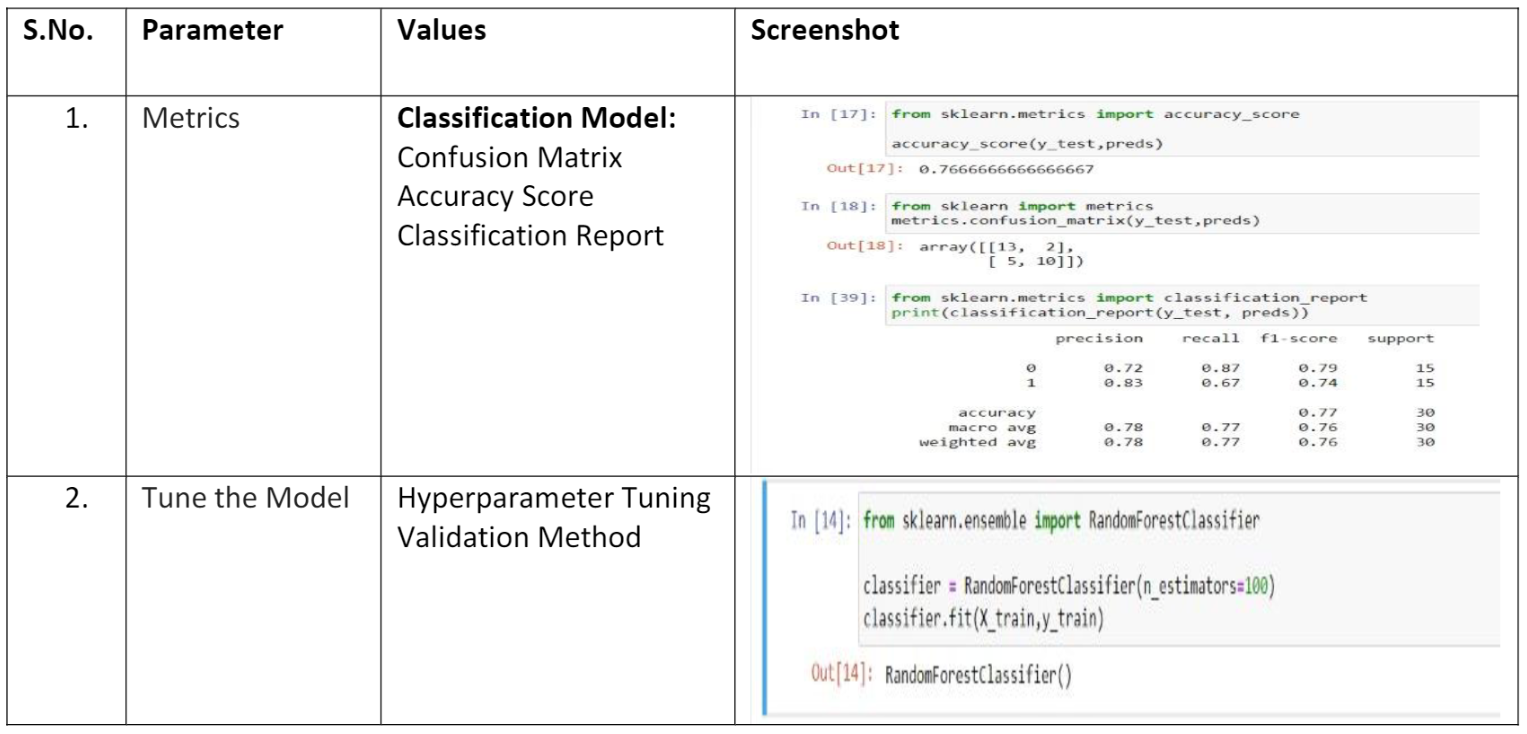
# User Acceptance Testing

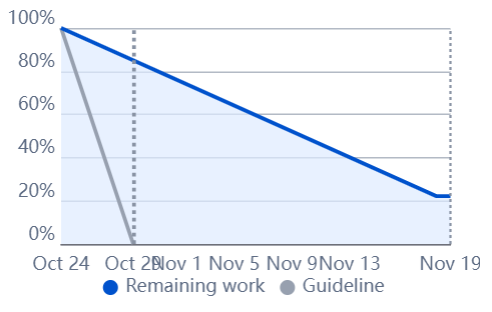
# 

# 

# RESULTS

* 1. **Performance Metrics**

****

****

# ADVANTAGES & DISADVANTAGES

**Advantages:**

* An efficient and accurate solution to detect Parkinson's disease.
* Parkinson's disease effectively.
* Easy and cheap to determine the presence of Parkinson's disease.
* Eliminates the human error rate.

# Disadvantages:

* + - It requires an active internet connection.
    - Better UI
    - The user cannot update their personal details once it has been registered.

# CONCLUSION

This project is used to accurately perform diagnosis for Parkinson's disease in a patient using their brain MRI scans thus, using a web application developed using reactJS. This also help in providing efficient treatment în a most cheap way and eventually reduce the time required for determining the Parkinson's disease in the patient cost effectively. Currently the conventional medical diagnosis is done manually which consumes more time and also involves human error rate. So, reduces the time required for manual classification and eliminates the human error rate by this project.

# FUTURE SCOPE

In future, the application of the Parkinson's disease diagnosis technology in the healthcare field can be reviewed and also it can promote for detecting the stages of the Parkinson's disease with more accuracy. In medical field they are more chance to develop or convert this project in many ways. Thus, this project has an efficient scope in coming future where manual predicting can be converted to computerized production in a cheap way.

# APPENDIX

**Source Code**

# parkinsons.pkl:

{

"cells": [

{

"cell\_type": "code",

"execution\_count": 1,

"metadata": {

"id": "LjgwWni3C1Uz"

},

"outputs": [],

"source": [

"import pandas as pd\n",

"import numpy as np\n",

"import matplotlib.pyplot as plt\n",

"import seaborn as sns\n",

"from imutils import paths\n",

"import os\n",

"import cv2\n",

"from skimage import feature"

]

},

{

"cell\_type": "code",

"execution\_count": 3,

"metadata": {

"id": "YdFXL79pC4AU"

},

"outputs": [],

"source": [

"def split(path):\n",

" imgpaths = list(paths.list\_images(path))\n",

" data=[]\n",

" labels=[]\n",

" for imgpath in imgpaths:\n",

" label=imgpath.split(os.path.sep)[-2]\n",

" labels.append(label)\n",

"\n",

" img=cv2.imread(imgpath)\n",

" img=cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)\n",

" img=cv2.resize(img,(200,200))\n",

"\n",

" img=cv2.threshold(img,0,255,cv2.THRESH\_BINARY\_INV | cv2.THRESH\_OTSU)[1]\n",

"\n",

" features = feature.hog(img,orientations=9,pixels\_per\_cell=(10,10),cells\_per\_block=(2,2),transform\_sqrt=True,block\_norm=\"L1\")\n",

" data.append(features)\n",

" return (np.array(data),np.array(labels))\n"

]

},

{

"cell\_type": "code",

"execution\_count": 4,

"metadata": {

"id": "zOUqj-fHC4Ea"

},

"outputs": [],

"source": [

"(X\_train,y\_train)=split(r'C:\\Users\\Sanjus\\Downloads\\dataset\\dataset\\spiral\\training')\n",

"(X\_test,y\_test)=split(r'C:\\Users\\Sanjus\\Downloads\\dataset\\dataset\\spiral\\testing')"

]

},

{

"cell\_type": "code",

"execution\_count": 5,

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "e\_ppgZLQftOj",

"outputId": "5d8bbdf5-e9a3-43b5-865b-e54fd2336b81"

},

"outputs": [

{

"data": {

"text/plain": [

"(72,)"

]

},

"execution\_count": 5,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"y\_train.shape"

]

},

{

"cell\_type": "code",

"execution\_count": 6,

"metadata": {

"id": "cnlFrICIC4Hi"

},

"outputs": [],

"source": [

"from sklearn.preprocessing import LabelEncoder\n",

"\n",

"enc = LabelEncoder()"

]

},

{

"cell\_type": "code",

"execution\_count": 7,

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "iwOTjC1PC4Lt",

"outputId": "cf7b35b8-faf8-41a1-d323-8be47c4412f0"

},

"outputs": [

{

"data": {

"text/plain": [

"array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,\n",

" 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,\n",

" 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,\n",

" 1, 1, 1, 1, 1, 1], dtype=int64)"

]

},

"execution\_count": 7,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"y\_train=enc.fit\_transform(y\_train)\n",

"\n",

"y\_train"

]

},

{

"cell\_type": "code",

"execution\_count": 8,

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "Mi-x8tPNC4P9",

"outputId": "f0f3ab92-6ccf-4a6b-e899-5c13fdf5c8e8"

},

"outputs": [

{

"data": {

"text/plain": [

"array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,\n",

" 1, 1, 1, 1, 1, 1, 1, 1], dtype=int64)"

]

},

"execution\_count": 8,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"y\_test=enc.fit\_transform(y\_test)\n",

"\n",

"y\_test"

]

},

{

"cell\_type": "code",

"execution\_count": 9,

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "SiU7qxxCC4TE",

"outputId": "0fc38ed0-b755-4d39-bc1f-31643aef7071"

},

"outputs": [

{

"data": {

"text/plain": [

"RandomForestClassifier()"

]

},

"execution\_count": 9,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"from sklearn.ensemble import RandomForestClassifier\n",

"\n",

"classifier = RandomForestClassifier(n\_estimators=100)\n",

"classifier.fit(X\_train,y\_train)"

]

},

{

"cell\_type": "code",

"execution\_count": 10,

"metadata": {

"id": "XAwC0HsFgUfq"

},

"outputs": [],

"source": [

"preds=classifier.predict(X\_test)"

]

},

{

"cell\_type": "code",

"execution\_count": 11,

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "yKLWMYvugmqM",

"outputId": "270e943e-2ef8-45cf-8188-323cecfccff2"

},

"outputs": [

{

"data": {

"text/plain": [

"array([0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,\n",

" 1, 1, 0, 0, 0, 1, 1, 1], dtype=int64)"

]

},

"execution\_count": 11,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"preds"

]

},

{

"cell\_type": "code",

"execution\_count": 12,

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "6aNw4lTGg\_yc",

"outputId": "b57421ad-4172-4ed4-b84e-5f074ff26920"

},

"outputs": [

{

"data": {

"text/plain": [

"0.8"

]

},

"execution\_count": 12,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"from sklearn.metrics import accuracy\_score\n",

"\n",

"accuracy\_score(y\_test,preds)"

]

},

{

"cell\_type": "code",

"execution\_count": 13,

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "yRBFfBw2g1wl",

"outputId": "f75283f2-d321-4fa4-ffae-2d968bd0529c"

},

"outputs": [

{

"data": {

"text/plain": [

"array([[13, 2],\n",

" [ 4, 11]], dtype=int64)"

]

},

"execution\_count": 13,

"metadata": {},

"output\_type": "execute\_result"

}

],

"source": [

"from sklearn import metrics\n",

"metrics.confusion\_matrix(y\_test,preds)"

]

},

{

"cell\_type": "code",

"execution\_count": 14,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

" precision recall f1-score support\n",

"\n",

" 0 0.76 0.87 0.81 15\n",

" 1 0.85 0.73 0.79 15\n",

"\n",

" accuracy 0.80 30\n",

" macro avg 0.81 0.80 0.80 30\n",

"weighted avg 0.81 0.80 0.80 30\n",

"\n"

]

}

],

"source": [

"from sklearn.metrics import classification\_report\n",

"print(classification\_report(y\_test, preds))"

]

},

{

"cell\_type": "code",

"execution\_count": 15,

"metadata": {},

"outputs": [],

"source": [

"import pickle\n",

"pickle.dump(classifier,open('model.pkl','wb'))"

]

}

],

"metadata": {

"colab": {

"provenance": []

},

"kernelspec": {

"display\_name": "Python 3",

"language": "python",

"name": "python3"

},

"language\_info": {

"codemirror\_mode": {

"name": "ipython",

"version": 3

},

"file\_extension": ".py",

"mimetype": "text/x-python",

"name": "python",

"nbconvert\_exporter": "python",

"pygments\_lexer": "ipython3",

"version": "3.8.8"

}

},

"nbformat": 4,

"nbformat\_minor": 1

}

# GitHub & Project Demo Link

**GitHub:** https://github.com/IBM-EPBL/IBM-Project-10817-1659232384

**Project Demo Link:** https://drive.google.com/file/d/1jlh\_XvFks1xCirFBmt3i1TvaLRR9PB3X/view