**SAMPLE CODE**

**User side views.py**

from django.shortcuts import render, HttpResponse

from .forms import UserRegistrationForm

from django.contrib import messages

from .models import UserRegistrationModel

# Create your views here.

def UserRegisterActions(request):

if request.method == 'POST':

form = UserRegistrationForm(request.POST)

if form.is\_valid():

print('Data is Valid')

form.save()

messages.success(request, 'You have been successfully registered')

form = UserRegistrationForm()

return render(request, 'UserRegistrations.html', {'form': form})

else:

messages.success(request, 'Email or Mobile Already Existed')

print("Invalid form")

else:

form = UserRegistrationForm()

return render(request, 'UserRegistrations.html', {'form': form})

def UserLoginCheck(request):

if request.method == "POST":

loginid = request.POST.get('loginname')

pswd = request.POST.get('pswd')

print("Login ID = ", loginid, ' Password = ', pswd)

try:

check = UserRegistrationModel.objects.get(loginid=loginid, password=pswd)

status = check.status

print('Status is = ', status)

if status == "activated":

request.session['id'] = check.id

request.session['loggeduser'] = check.name

request.session['loginid'] = loginid

request.session['email'] = check.email

print("User id At", check.id, status)

return render(request, 'users/UserHome.html', {})

else:

messages.success(request, 'Your Account has not been activated by Admin.')

return render(request, 'UserLogin.html')

except Exception as e:

print('Exception is ', str(e))

pass

messages.success(request, 'Invalid Login id and password')

return render(request, 'UserLogin.html', {})

def UserHome(request):

return render(request, 'users/UserHome.html', {})

def user\_view\_dataset(request):

from django.conf import settings

import os

import pandas as pd

path = os.path.join(settings.MEDIA\_ROOT,'uci\_heart.csv')

df = pd.read\_csv(path)

df = df.to\_html

return render(request, 'users/user\_data\_view.html',{'data':df})

def user\_machine\_learning(request):

from .utility import MachineLearningUtility

svc\_accuracy, svc\_precision, svc\_recall, svc\_f1score = MachineLearningUtility.calc\_support\_vector\_classifier()

j48\_accuracy, j48\_precision, j48\_recall, j48\_f1score = MachineLearningUtility.calc\_j48\_classifier()

ann\_accuracy, ann\_precision, ann\_recall, ann\_f1score = MachineLearningUtility.calc\_ann\_model()

my\_accuracy, my\_precision, my\_recall, my\_f1score = MachineLearningUtility.calc\_proposed\_model()

j48\_dict = {'j48\_accuracy': j48\_accuracy, 'j48\_precision': j48\_precision, "j48\_recall": j48\_recall,

'j48\_f1score': j48\_f1score}

ann\_dict = {'ann\_accuracy': ann\_accuracy, 'ann\_precision': ann\_precision, 'ann\_recall': ann\_recall,

'ann\_f1score': ann\_f1score}

svc\_dict = {'svc\_accuracy': svc\_accuracy, 'svc\_precision': svc\_precision, 'svc\_recall': svc\_recall,

'svc\_f1score': svc\_f1score}

my\_dict = {'my\_accuracy': my\_accuracy, 'my\_precision': my\_precision, 'my\_recall': my\_recall,

'my\_f1score': my\_f1score}

return render(request, 'users/usermachinelearning.html',

{'j48': j48\_dict, 'ann': ann\_dict, "svc": svc\_dict, 'my': my\_dict})

def user\_hidden\_markov(request):

from .utility import MachineLearningUtility

hmm\_result = MachineLearningUtility.calc\_hmm\_model()

return render(request, 'users/user\_hmm\_result.html', {'hmm': hmm\_result})

def user\_predictions(request):

if request.method=='POST':

age = int(request.POST.get('age'))

sex = int(request.POST.get('sex'))

cp = int(request.POST.get('cp'))

trestbps = int(request.POST.get('trestbps'))

chol = int(request.POST.get('chol'))

fbs = int(request.POST.get('fbs'))

restecg = int(request.POST.get('restecg'))

thalach = int(request.POST.get('thalach'))

exang = int(request.POST.get('exang'))

oldpeak = float(request.POST.get('oldpeak'))

slope = int(request.POST.get('slope'))

ca = int(request.POST.get('ca'))

thal = int(request.POST.get('thal'))

test\_data = [age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang, oldpeak,slope,ca, thal]

from .utility import MachineLearningUtility

test\_pred = MachineLearningUtility.test\_user\_date(test\_data)

if test\_pred[0] == 0:

rslt = False

else:

rslt = True

return render(request, "users/predictions\_form.html", {"test\_data": test\_data, "result": rslt})

else:

return render(request, 'users/predictions\_form.html',{})

models.py

from django.db import models

# Create your models here.

class UserRegistrationModel(models.Model):

name = models.CharField(max\_length=100)

loginid = models.CharField(unique=True, max\_length=100)

password = models.CharField(max\_length=100)

mobile = models.CharField(unique=True, max\_length=100)

email = models.CharField(unique=True, max\_length=100)

locality = models.CharField(max\_length=100)

address = models.CharField(max\_length=1000)

city = models.CharField(max\_length=100)

state = models.CharField(max\_length=100)

status = models.CharField(max\_length=100)

def \_\_str\_\_(self):

return self.loginid

class Meta:

db\_table = 'UserRegistrations'

forms.py

from django import forms

from .models import UserRegistrationModel

class UserRegistrationForm(forms.ModelForm):

name = forms.CharField(widget=forms.TextInput(attrs={'pattern': '[a-zA-Z]+', 'class': 'form-control'}),

required=True, max\_length=100)

loginid = forms.CharField(widget=forms.TextInput(attrs={'pattern': '[a-zA-Z]+', 'class': 'form-control'}),

required=True, max\_length=100)

password = forms.CharField(

widget=forms.PasswordInput(attrs={'pattern': '(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{8,}', 'class': 'form-control',

'title': 'Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters'}),

required=True, max\_length=100)

mobile = forms.CharField(widget=forms.TextInput(attrs={'pattern': '[56789][0-9]{9}', 'class': 'form-control'}),

required=True,

max\_length=100)

email = forms.CharField(

widget=forms.TextInput(attrs={'pattern': '[a-z0-9.\_%+-]+@[a-z0-9.-]+\.[a-z]{2,}$', 'class': 'form-control'}),

required=True, max\_length=100)

locality = forms.CharField(widget=forms.TextInput(attrs={'class': 'form-control'}), required=True, max\_length=100)

address = forms.CharField(widget=forms.Textarea(attrs={'rows': 3, 'cols': 20, 'class': 'form-control'}),

required=True, max\_length=250)

city = forms.CharField(widget=forms.TextInput(

attrs={'autocomplete': 'off', 'pattern': '[A-Za-z ]+', 'title': 'Enter Characters Only ',

'class': 'form-control'}), required=True,

max\_length=100)

state = forms.CharField(widget=forms.TextInput(

attrs={'autocomplete': 'off', 'pattern': '[A-Za-z ]+', 'title': 'Enter Characters Only ',

'class': 'form-control'}), required=True,

max\_length=100)

status = forms.CharField(widget=forms.HiddenInput(), initial='waiting', max\_length=100)

class Meta():

model = UserRegistrationModel

fields = '\_\_all\_\_'

Machine Learning.py

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from django.conf import settings

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix

import numpy as np

path = settings.MEDIA\_ROOT + "//" + "uci\_heart.csv"

df = pd.read\_csv(path)

X = df.iloc[:, :-1].values # indipendent variable

y = df.iloc[:, -1].values # Dependent variable

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.80, random\_state=0)

from hmmlearn.hmm import GaussianHMM

def fitHMM(Q, nSamples):

# fit Gaussian HMM to Q

model = GaussianHMM(n\_components=2, n\_iter=1000).fit(np.reshape(Q, [len(Q), 1]))

# classify each observation as state 0 or 1

hidden\_states = model.predict(np.reshape(Q, [len(Q), 1]))

# find parameters of Gaussian HMM

mus = np.array(model.means\_)

sigmas = np.array(np.sqrt(np.array([np.diag(model.covars\_[0]), np.diag(model.covars\_[1])])))

P = np.array(model.transmat\_)

# find log-likelihood of Gaussian HMM

logProb = model.score(np.reshape(Q, [len(Q), 1]))

# generate nSamples from Gaussian HMM

samples = model.sample(nSamples)

# re-organize mus, sigmas and P so that first row is lower mean (if not already)

if mus[0] > mus[1]:

mus = np.flipud(mus)

sigmas = np.flipud(sigmas)

P = np.fliplr(np.flipud(P))

hidden\_states = 1 - hidden\_states

return hidden\_states, mus, sigmas, P, logProb, samples

def calc\_hmm\_model():

path = settings.MEDIA\_ROOT + "//" + "uci\_heart.csv"

df = pd.read\_csv(path)

X = df.iloc[:, :-1].values # indipendent variable

y = df.iloc[:, -1].values # Dependent variable

X = np.column\_stack([X, y])

print("fitting to HMM and decoding ...", end="")

# Make an HMM instance and execute fit

model = GaussianHMM(n\_components=4, covariance\_type="diag", n\_iter=1000).fit(X)

# Predict the optimal sequence of internal hidden state

hidden\_states = model.predict(X)

print("done")

print("Transition matrix")

print(model.transmat\_)

print()

print("Means and vars of each hidden state")

li = []

for i in range(model.n\_components):

print("{0}th hidden state".format(i))

print("mean = ", model.means\_[i])

print("var = ", np.diag(model.covars\_[i]))

print()

rslt = {str(i)+' hidden state': i, "mean": model.means\_[i], "var": np.diag(model.covars\_[i])}

li.append(rslt)

return li

def calc\_ann\_model():

print("\*" \* 25, "Artificail Neural Network")

import keras

from keras.models import Sequential

from keras.layers import Dense

classifier = Sequential()

classifier.add(Dense(output\_dim=13, init='uniform', activation='relu', input\_dim=13))

classifier.add(Dense(output\_dim=13, init='uniform', activation='relu'))

classifier.add(Dense(output\_dim=1, init='uniform', activation='sigmoid'))

classifier.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

print(classifier.summary())

classifier.fit(X\_train, y\_train, batch\_size=10, nb\_epoch=100)

y\_pred = classifier.predict(X\_test)

y\_pred = (y\_pred > 0.5)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

accuracy = accuracy\_score(y\_test, y\_pred)

print('AI Accuracy:', accuracy)

precision = precision\_score(y\_test, y\_pred)

print('AI Precision Score:', precision)

recall = recall\_score(y\_test, y\_pred)

print('AI Recall Score:', recall)

f1score = f1\_score(y\_test, y\_pred)

print('AI F1-Score:', f1score)

return accuracy, precision, recall, f1score

def calc\_proposed\_model():

print("\*" \* 25, "Random Forest Classification")

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(X\_train, y\_train) # Trained wih 80% Data

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print('RF Accuracy:', accuracy)

cm = confusion\_matrix(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

print('RF Precision Score:', precision)

recall = recall\_score(y\_test, y\_pred)

print('RF Recall Score:', recall)

f1score = f1\_score(y\_test, y\_pred)

print('RF F1-Score:', f1score)

return accuracy, precision, recall, f1score

def calc\_support\_vector\_classifier():

print("\*" \* 25, "SVM Classification")

from sklearn.svm import SVC

model = SVC(kernel='rbf')

model.fit(X\_train, y\_train) # Trained wih 80% Data

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print('SVM Accuracy:', accuracy)

cm = confusion\_matrix(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

print('SVM Precision Score:', precision)

recall = recall\_score(y\_test, y\_pred)

print('SVM Recall Score:', recall)

f1score = f1\_score(y\_test, y\_pred)

print('SVM F1-Score:', f1score)

return accuracy, precision, recall, f1score

def calc\_j48\_classifier():

print("\*" \* 25, "j48")

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(criterion="entropy", max\_depth=3)

model.fit(X\_train, y\_train) # Trained wih 80% Data

y\_pred = model.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print('j48 Accuracy:', accuracy)

cm = confusion\_matrix(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

print('j48 Precision Score:', precision)

recall = recall\_score(y\_test, y\_pred)

print('j48 Recall Score:', recall)

f1score = f1\_score(y\_test, y\_pred)

print('j48 F1-Score:', f1score)

return accuracy, precision, recall, f1score

def test\_user\_date(test\_features):

print(test\_features)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(X\_train, y\_train)

test\_pred = model.predict([test\_features])

return test\_pred

base.html

{% load static %}

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

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

<title>Hybrid Machine Learning Classification</title>

<meta content="" name="description">

<meta content="" name="keywords">

<!-- Favicons -->

<link href="{% static 'assets/img/favicon.png' %}" rel="icon">

<link href="{% static 'assets/img/apple-touch-icon.png' %}" rel="apple-touch-icon">

<!-- Google Fonts -->

<link

href="https://fonts.googleapis.com/css?family=Open+Sans:300,300i,400,400i,600,600i,700,700i|Raleway:300,300i,400,400i,500,500i,600,600i,700,700i|Poppins:300,300i,400,400i,500,500i,600,600i,700,700i"

rel="stylesheet">

<!-- Vendor CSS Files -->

<link href="{% static 'assets/vendor/aos/aos.css' %}" rel="stylesheet">

<link href="{% static 'assets/vendor/bootstrap/css/bootstrap.min.css' %}" rel="stylesheet">

<link href="{% static 'assets/vendor/bootstrap-icons/bootstrap-icons.css' %}" rel="stylesheet">

<link href="{% static 'assets/vendor/boxicons/css/boxicons.min.css' %}" rel="stylesheet">

<link href="{% static 'assets/vendor/glightbox/css/glightbox.min.css' %}" rel="stylesheet">

<link href="{% static 'assets/vendor/remixicon/remixicon.css' %}" rel="stylesheet">

<link href="{% static 'assets/vendor/swiper/swiper-bundle.min.css' %}" rel="stylesheet">

<!-- Template Main CSS File -->

<link href="{% static 'assets/css/style.css' %}" rel="stylesheet">

<link rel="stylesheet" href="{% static 'css/bootstrap.min.css' %}">

</head>

<body>

<!-- ======= Header ======= -->

<header id="header" class="fixed-top bg-black">

<div class="container d-flex align-items-center justify-content-between">

<h1 class="logo"><a href="/">Hybrid Heart Disease</a></h1>

<!-- Uncomment below if you prefer to use an image logo -->

<!-- <a href="index.html" class="logo"><img src="assets/img/logo.png" alt="" class="img-fluid"></a>-->

<nav id="navbar" class="navbar">

<ul>

<li><a class="nav-link scrollto" href="/">Home</a></li>

<li><a class="nav-link scrollto {% block user\_login %}{% endblock %}"

href="{% url 'UserLogin' %}">User</a>

</li>

<li><a class="nav-link scrollto {% block admin\_login %}{% endblock %}" href="{% url 'AdminLogin' %}">Admin</a>

</li>

<li><a class="getstarted scrollto {% block register %} {% endblock %}" href="{% url 'UserRegister' %}">Register</a></li>

</ul>

<i class="bi bi-list mobile-nav-toggle"></i>

</nav><!-- .navbar -->

</div>

</header><!-- End Header -->

{% block content %} {% endblock %}

<!-- ======= Footer ======= -->

<footer id="footer">

<div class="container">

<div class="copyright">

&copy; Copyright <strong><span>Heart Disease</span></strong>. All Rights Reserved

</div>

<div class="credits">

<!-- All the links in the footer should remain intact. -->

<!-- You can delete the links only if you purchased the pro version. -->

<!-- Licensing information: https://bootstrapmade.com/license/ -->

<!-- Purchase the pro version with working PHP/AJAX contact form: https://bootstrapmade.com/dewi-free-multi-purpose-html-template/ -->

Designed by <a href="">Alex Corporation</a>

</div>

</div>

</footer>

<!-- End Footer -->

<div id="preloader"></div>

<a href="#" class="back-to-top d-flex align-items-center justify-content-center"><i

class="bi bi-arrow-up-short"></i></a>

<!-- Vendor JS Files -->

<script src="{% static 'assets/vendor/aos/aos.js' %}"></script>

<script src="{% static 'assets/vendor/bootstrap/js/bootstrap.bundle.min.js' %}"></script>

<script src="{% static 'assets/vendor/glightbox/js/glightbox.min.js' %}"></script>

<script src="{% static 'assets/vendor/isotope-layout/isotope.pkgd.min.js' %}"></script>

<script src="{% static 'assets/vendor/php-email-form/validate.js' %}"></script>

<script src="{% static 'assets/vendor/purecounter/purecounter.js' %}"></script>

<script src="{% static 'assets/vendor/swiper/swiper-bundle.min.js' %}"></script>

<!-- Template Main JS File -->

<script src="{% static 'assets/js/main.js' %}"></script>

</body>

</html>