Disease Detection using Machine Learning

A J-Component Project Report

Submitted by

Shrishail Terni - 19BCE0445
Aditya Anavekar – 19BCI0236

Under the guidance of

Dr. Arun Kumar G

Associate Professor, SCOPE,

VIT, Vellore.

in partial fulfillment for the award of the degree of

B.Tech.

in

COMPUTER SCIENCE AND ENGINEERING
NOVEMBER, 2021

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Abstract:

Artificial Intelligence has created a lot of buzz in the past few years. Its applications visualized in the healthcare sector are very astonishing. The present era is overwhelmed with new diseases which leads to an increasing number of patients whereas the number of doctors produced are the same, these situations bring in a need for assistance for the patients to receive proper healthcare. Visiting a hospital or a doctor is difficult at times due to various reasons such as transport, availability, time etc. So, getting instant assistance on the disease you're suffering from along with medication recommendation would be great for such patients and also for the future where you can get the services of a doctor remotely sitting in your house. Disease detection can be achieved with the help of Artificial Intelligence and Machine learning. With the data sets of patients with the diseases and their symptoms the machines can be trained in such a way to predict the disease with a good percentage of accuracy.
The main purpose behind the project is to bring about major reforms in the field of healthcare using Artificial Intelligence. The model shall predict the disease on the basis of the symptoms entered. It will be a major contribution in the field of healthcare which has gained a lot of importance in the current pandemic situation.

Proposed model:

Dataset of disease:

The rows of our dataset correspond to the diseases whereas the columns of our dataset correspond to the symptoms. The
entries of the dataset will be either 0 or 1.
0 - patient does not suffer from that symptom in the corresponding disease
1 - patient does suffer from that symptom in the corresponding disease

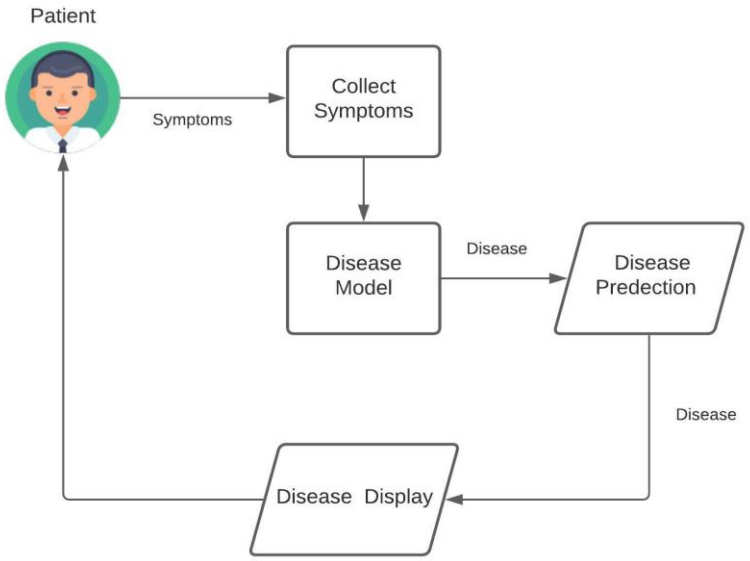
Algorithms:

We will be training our model on 4 algorithms which are:
1. Naive Bayes
2. K-Nearest Neighbor
3. Decision Tree
4. Random Forest
5. SVM

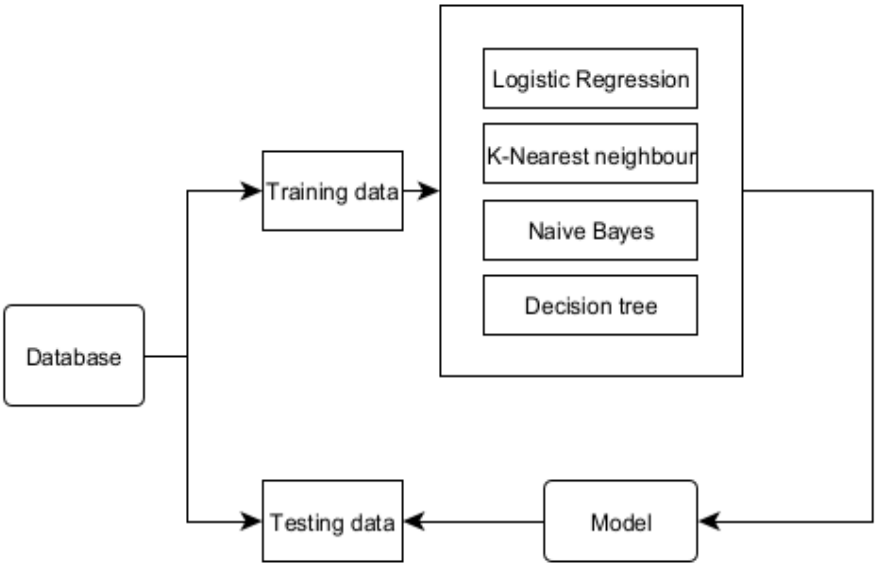
The user can view predictions of all the algorithms and then make an informed choice based on the results.

Design and Architecture diagram:

Design Flow:



Model Architecture:



Code:

```
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from tkinter import *
import numpy as np
import pandas as pd
import os

l1=['back_pain','constipation','abdominal_pain','diarrhoea','mild_fever','yellow_urine',
   'yellowing_of_eyes','acute_liver_failure','fluid_overload','swelling_of_stomach',
   'swelled_lymph_nodes','malaise','blurred_and_distorted_vision','phlegm','throat_irritation',
   'redness_of_eyes','sinus_pressure','runny_nose','congestion','chest_pain','weakness_in_limbs',
   'fast_heart_rate','pain_during_bowel_movements','pain_in_anal_region','bloody_stool',
   'irritation_in_anus','neck_pain','dizziness','cramps','bruising','obesity','swollen_legs',
   'swollen_blood_vessels','puffy_face_and_eyes','enlarged_thyroid','brittle_nails',
   'swollen_extremeties','excessive_hunger','extra_marital_contacts','drying_and_tingling_lips',
   'slurred_speech','knee_pain','hip_joint_pain','muscle_weakness','stiff_neck','swelling_joints',
   'movement_stiffness','spinning_movements','loss_of_balance','unsteadiness',
   'weakness_of_one_body_side','loss_of_smell','bladder_discomfort','foul_smell_of urine',
   'continuous_feel_of_urine','passage_of_gases','internal_itching','toxic_look_(typhos)',
   'depression','irritability','muscle_pain','altered_sensorium','red_spots_over_body','belly_pain',
   'abnormal_menstruation','dischromic
_patches','watering_from_eyes','increased_appetite','polyuria','family_history','mucoid_sputum',
   'rusty_sputum','lack_of_concentration','visual_disturbances','receiving_blood_transfusion',
   'receiving_unsterile_injections','coma','stomach_bleeding','distention_of_abdomen',
   'history_of_alcohol_consumption','fluid_overload','blood_in_sputum','prominent_veins_on_calf',
   'palpitations','painful_walking','pus_filled_pimples','blackheads','scurrying','skin_peeling',
   'silver_like_dusting','small_dents_in_nails','inflammatory_nails','blister','red_sore_around_nose',
   'Yellow_crust_ooze']

disease=['Fungal infection', 'Allergy', 'GERD', 'Chronic cholestasis',
      'Drug Reaction', 'Peptic ulcer disease', 'AIDS', 'Diabetes ',
      'Gastroenteritis', 'Bronchial Asthma', 'Hypertension ', 'Migraine',
      'Cervical spondylosis', 'Paralysis (brain hemorrhage)', 'Jaundice',
      'Malaria', 'Chicken pox', 'Dengue', 'Typhoid', 'hepatitis A',
      'Hepatitis B', 'Hepatitis C', 'Hepatitis D', 'Hepatitis E',
      'Alcoholic hepatitis', 'Tuberculosis', 'Common Cold', 'Pneumonia',
      'Dimorphic hemorrhoids(piles)', 'Heart attack', 'Varicose veins',
      'Hypothyroidism', 'Hyperthyroidism', 'Hypoglycemia',
      'Osteoarthritis', 'Arthritis',
      '(vertigo) Paroxysmal  Positional Vertigo', 'Acne',
      'Urinary tract infection', 'Psoriasis', 'Impetigo']
```

```
def run_fun():
    DecisionTree()
    randomforest()
    KNN()
    NaiveBayes()
    SVM()

datafile = pd.read_csv("Training.csv")
datafile2 = pd.read_csv('Training.csv', index_col='prognosis')

datafile.replace({'prognosis': {'Fungal infection': 0, 'Allergy': 1, 'GERD': 2, 'Chronic cholestasis': 3, 'Drug
Reaction': 4,
                  'Peptic ulcer disease': 5, 'AIDS': 6, 'Diabetes ': 7, 'Gastroenteritis': 8,
                  'Bronchial Asthma': 9, 'Hypertension ': 10,
                  'Migraine': 11, 'Cervical spondylosis': 12,
                  'Paralysis (brain hemorrhage)': 13, 'Jaundice': 14, 'Malaria': 15, 'Chicken pox': 16,
                  'Dengue': 17, 'Typhoid': 18, 'hepatitis A': 19,
                  'Hepatitis B': 20, 'Hepatitis C': 21, 'Hepatitis D': 22, 'Hepatitis E': 23,
                  'Alcoholic hepatitis': 24, 'Tuberculosis': 25,
                  'Common Cold': 26, 'Pneumonia': 27, 'Dimorphic hemorrhoids(piles)': 28, 'Heart attack': 29,
                  'Varicose veins': 30, 'Hypothyroidism': 31,
                  'Hyperthyroidism': 32, 'Hypoglycemia': 33, 'Osteoarthristis': 34, 'Arthritis': 35,
                      '(vertigo) Paroxysmal  Positional Vertigo': 36, 'Acne': 37, 'Urinary tract infection': 38,
                      'Psoriasis': 39,
                      'Impetigo': 40}}, inplace=True)
# df.head()
datafile2.head()

def plotPerColumnDistribution(df1, nGraphShown, nGraphPerRow):
    nunique = df1.nunique()
    df1 = df1[[col for col in datafile if nunique[col] > 1 and nunique[
        col] < 50]]  # For displaying purposes, pick columns that have between 1 and 50 unique values
    nRow, nCol = df1.shape
    columnNames = list(df1)
    nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
    plt.figure(num=None, figsize=(6 * nGraphPerRow, 8 * nGraphRow), dpi=80, facecolor='w', edgecolor='k')
    for i in range(min(nCol, nGraphShown)):
        plt.subplot(nGraphRow, nGraphPerRow, i + 1)
        columnDf = datafile.iloc[:, i]
        if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
            valueCounts = columnDf.value_counts()
            valueCounts.plot.bar()
        else:
            columnDf.hist()
```

```
        plt.ylabel('counts')
        plt.xticks(rotation=90)
        plt.title(f'{columnNames[i]} (column {i})')
    plt.tight_layout(pad=1.0, w_pad=1.0, h_pad=1.0)
    plt.show()

def plotScatterMatrix(df1, plotSize, textSize):
    df1 = df1.select_dtypes(include=[np.number])
    df1 = df1.dropna('columns')
    df1 = df1[[col for col in datafile if datafile[col].nunique() > 1]]
    columnNames = list(datafile)
    if len(columnNames) > 10:  plots
        columnNames = columnNames[:10]
    df1 = df1[columnNames]
    ax = pd.plotting.scatter_matrix(df1, alpha=0.75, figsize=[plotSize, plotSize], diagonal='kde')
    corrs = df1.corr().values
    for i, j in zip(*plt.np.triu_indices_from(ax, k=1)):
        ax[i, j].annotate('Corr. coef = %.3f' % corrs[i, j], (0.8, 0.2), xycoords='axes fraction', ha='center',
                          va='center', size=textSize)
    plt.suptitle('Scatter and Density Plot')
    plt.show()

# plotPerColumnDistribution(df, 10, 5)
# plotScatterMatrix(df, 20, 10)
X = datafile[l1]
y = datafile[["prognosis"]]
np.ravel(y)
# print(X)

tr = pd.read_csv("testing.csv")

tr.replace({'prognosis': {'Fungal infection': 0, 'Allergy': 1, 'GERD': 2, 'Chronic cholestasis': 3, 'Drug Reaction':
4,
                          'Peptic ulcer diseae': 5, 'AIDS': 6, 'Diabetes ': 7, 'Gastroenteritis': 8,
                          'Bronchial Asthma': 9, 'Hypertension ': 10,
                          'Migraine': 11, 'Cervical spondylosis': 12,
                          'Paralysis (brain hemorrhage)': 13, 'Jaundice': 14, 'Malaria': 15, 'Chicken pox': 16,
                          'Dengue': 17, 'Typhoid': 18, 'hepatitis A': 19,
                          'Hepatitis B': 20, 'Hepatitis C': 21, 'Hepatitis D': 22, 'Hepatitis E': 23,
                          'Alcoholic hepatitis': 24, 'Tuberculosis': 25,
                          'Common Cold': 26, 'Pneumonia': 27, 'Dimorphic hemmorhoids(piles)': 28, 'Heart attack': 29,
                          'Varicose veins': 30, 'Hypothyroidism': 31,
                          'Hyperthyroidism': 32, 'Hypoglycemia': 33, 'Osteoarthristis': 34, 'Arthritis': 35,
                          '(vertigo) Paroymsal  Positional Vertigo': 36, 'Acne': 37, 'Urinary tract infection': 38,
                          'Psoriasis': 39,
```

```
                    'Impetigo': 40}}, inplace=True)
tr.head()
X_test = tr[l1]
y_test = tr[["prognosis"]]
np.ravel(y_test)
print(X_test)

def scatterplt(disea):
    x = ((datafile2.loc[disea]).sum())
    x.drop(x[x == 0].index, inplace=True)
    print(x.values)
    y = x.keys()
    print(len(x))
    print(len(y))
    plt.title(disea)
    plt.scatter(y, x.values)
    plt.show()

def scatterinp(sym1, sym2, sym3, sym4, sym5):
    x = [sym1, sym2, sym3, sym4, sym5]  # storing input symptoms in y
    y = [0, 0, 0, 0, 0]  # creating and giving values to the input symptoms
    if sym1 != 'Select Here':
        y[0] = 1
    if sym2 != 'Select Here':
        y[1] = 1
    if sym3 != 'Select Here':
        y[2] = 1
    if sym4 != 'Select Here':
        y[3] = 1
    if sym5 != 'Select Here':
        y[4] = 1
    #print(x)
    #print(y)
    plt.scatter(x, y)
    plt.show()

def DecisionTree():
    l2 = []
    for i in range(0, len(l1)):
        l2.append(0)
    if len(NameEn.get()) == 0:
        pred1.set(" ")
        comp = messagebox.askokcancel("System", "Kindly Fill the Name")
        if comp:
            root.mainloop()
```

```
elif ((Symptom1.get() == "Select Here") or (Symptom2.get() == "Select Here")):
    pred1.set(" ")
    sym = messagebox.askokcancel("System", "Kindly Fill atleast first two Symptoms")
    if sym:
        root.mainloop()
else:
    from sklearn import tree
    clf3 = tree.DecisionTreeClassifier()
    clf3 = clf3.fit(X, y)
    from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
    y_pred1 = clf3.predict(X_test)
    print("Decision Tree")
    print("Accuracy")
    print(accuracy_score(y_test, y_pred1))
    print(accuracy_score(y_test, y_pred1, normalize=False))
    print("Confusion matrix")
    conf_matrix = confusion_matrix(y_test, y_pred1)
    print(conf_matrix)

    psymptoms = [Symptom1.get(), Symptom2.get(), Symptom3.get(), Symptom4.get(), Symptom5.get()]
    for k in range(0, len(l1)):
        for z in psymptoms:
            if (z == l1[k]):
                l2[k] = 1

    inputtest = [l2]
    predict = clf3.predict(inputtest)
    predicted = predict[0]

    h = 'no'
    for a in range(0, len(disease)):
        if (predicted == a):
            h = 'yes'
            break

    if (h == 'yes'):
        pred1.set(" ")
        pred1.set(disease[a])
    else:
        pred1.set(" ")
        pred1.set("Not Found")

    scatterinp(Symptom1.get(), Symptom2.get(), Symptom3.get(), Symptom4.get(), Symptom5.get())
    scatterplt(pred1.get())
```

```
def randomforest():
    l2 = []
    for i in range(0, len(l1)):
        l2.append(0)
    if len(NameEn.get()) == 0:
        pred1.set(" ")
        comp = messagebox.askokcancel("System", "Kindly Fill the Name")
        if comp:
            root.mainloop()
    elif ((Symptom1.get() == "Select Here") or (Symptom2.get() == "Select Here")):
        pred1.set(" ")
        sym = messagebox.askokcancel("System", "Kindly Fill atleast first two Symptoms")
        if sym:
            root.mainloop()
    else:
        from sklearn.ensemble import RandomForestClassifier
        clf4 = RandomForestClassifier(n_estimators=100)
        clf4 = clf4.fit(X, np.ravel(y))

        # calculating accuracy
        from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
        y_pred2 = clf4.predict(X_test)
        print("Random Forest")
        print("Accuracy")
        print(accuracy_score(y_test, y_pred2))
        print(accuracy_score(y_test, y_pred2, normalize=False))
        print("Confusion matrix")
        conf_matrix = confusion_matrix(y_test, y_pred2)
        print(conf_matrix)

        psymptoms = [Symptom1.get(), Symptom2.get(), Symptom3.get(), Symptom4.get(), Symptom5.get()]

        for k in range(0, len(l1)):
            for z in psymptoms:
                if (z == l1[k]):
                    l2[k] = 1

        inputtest = [l2]
        predict = clf4.predict(inputtest)
        predicted = predict[0]

        h = 'no'
        for a in range(0, len(disease)):
            if (predicted == a):
                h = 'yes'
```

```
            break
        if (h == 'yes'):
            pred2.set(" ")
            pred2.set(disease[a])
        else:
            pred2.set(" ")
            pred2.set("Not Found")

        scatterplt(pred2.get())

def KNN():
    l2 = []
    for i in range(0, len(l1)):
        l2.append(0)
    if len(NameEn.get()) == 0:
        pred1.set(" ")
        comp = messagebox.askokcancel("System", "Kindly Fill the Name")
        if comp:
            root.mainloop()
    elif ((Symptom1.get() == "Select Here") or (Symptom2.get() == "Select Here")):
        pred1.set(" ")
        sym = messagebox.askokcancel("System", "Kindly Fill atleast first two Symptoms")
        if sym:
            root.mainloop()
    else:
        from sklearn.neighbors import KNeighborsClassifier
        knn = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
        knn = knn.fit(X, np.ravel(y))

        from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
        y_pred3 = knn.predict(X_test)
        print("kNearest Neighbour")
        print("Accuracy")
        print(accuracy_score(y_test, y_pred3))
        print(accuracy_score(y_test, y_pred3, normalize=False))
        print("Confusion matrix")
        conf_matrix = confusion_matrix(y_test, y_pred3)
        print(conf_matrix)

        psymptoms = [Symptom1.get(), Symptom2.get(), Symptom3.get(), Symptom4.get(), Symptom5.get()]

        for k in range(0, len(l1)):
            for z in psymptoms:
                if (z == l1[k]):
                    l2[k] = 1
```

```
        inputtest = [l2]
        predict = knn.predict(inputtest)
        predicted = predict[0]

        h = 'no'
        for a in range(0, len(disease)):
            if (predicted == a):
                h = 'yes'
                break

        if (h == 'yes'):
            pred4.set(" ")
            pred4.set(disease[a])
        else:
            pred4.set(" ")
            pred4.set("Not Found")
        scatterplt(pred4.get())

def NaiveBayes():
    l2 = []
    for i in range(0, len(l1)):
        l2.append(0)
    if len(NameEn.get()) == 0:
        pred1.set(" ")
        comp = messagebox.askokcancel("System", "Kindly Fill the Name")
        if comp:
            root.mainloop()
    elif ((Symptom1.get() == "Select Here") or (Symptom2.get() == "Select Here")):
        pred1.set(" ")
        sym = messagebox.askokcancel("System", "Kindly Fill atleast first two Symptoms")
        if sym:
            root.mainloop()
    else:
        from sklearn.naive_bayes import GaussianNB
        gnb = GaussianNB()
        gnb = gnb.fit(X, np.ravel(y))

        from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
        y_pred4 = gnb.predict(X_test)
        print("Naive Bayes")
        print("Accuracy")
        print(accuracy_score(y_test, y_pred4))
        print(accuracy_score(y_test, y_pred4, normalize=False))
        print("Confusion matrix")
```

```
        conf_matrix = confusion_matrix(y_test, y_pred4)
        print(conf_matrix)

        psymptoms = [Symptom1.get(), Symptom2.get(), Symptom3.get(), Symptom4.get(), Symptom5.get()]
        for k in range(0, len(l1)):
            for z in psymptoms:
                if (z == l1[k]):
                    l2[k] = 1

        inputtest = [l2]
        predict = gnb.predict(inputtest)
        predicted = predict[0]

        h = 'no'
        for a in range(0, len(disease)):
            if (predicted == a):
                h = 'yes'
                break
        if (h == 'yes'):
            pred3.set(" ")
            pred3.set(disease[a])
        else:
            pred3.set(" ")
            pred3.set("Not Found")

        scatterplt(pred3.get())

    def SVM():
        l2=[]
        for i in range(0,len(l1)):
            l2.append(0)
        if len(NameEn.get()) == 0:
            pred1.set(" ")
            comp=messagebox.askokcancel("System","Kindly Fill the Name")
            if comp:
                root.mainloop()
        elif((Symptom1.get()=="Select Here") or (Symptom2.get()=="Select Here")):
            pred1.set(" ")
            sym=messagebox.askokcancel("System","Kindly Fill atleast first two Symptoms")
            if sym:
                root.mainloop()
        else:
            from sklearn.svm import SVC
            svm_1 = SVC()
            svm_1=svm_1.fit(X,np.ravel(y))
```

```
    from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
    y_pred5=svm_1.predict(X_test)
    print("SVM")
    print("Accuracy")
    print(accuracy_score(y_test, y_pred5))
    print(accuracy_score(y_test, y_pred5,normalize=False))
    print("Confusion matrix")
    conf_matrix=confusion_matrix(y_test,y_pred5)
    print(conf_matrix)

psymptoms = [Symptom1.get(),Symptom2.get(),Symptom3.get(),Symptom4.get(),Symptom5.get()]
    for k in range(0,len(l1)):
        for z in psymptoms:
            if(z==l1[k]):
                l2[k]=1

    inputtest = [l2]
    predict = svm_1.predict(inputtest)
    predicted=predict[0]

    h='no'
    for a in range(0,len(disease)):
        if(predicted == a):
            h='yes'
            break
    if (h=='yes'):
        pred5.set(" ")
        pred5.set(disease[a])
    else:
        pred5.set(" ")
        pred5.set("Not Found")

    scatterplt(pred5.get())

root = Tk()
root.configure(background='Light Blue')
root.title('Disease Predictor System Using AI')
root.resizable(0, 0)

pred1 = StringVar()
pred2 = StringVar()
pred4 = StringVar()
pred3 = StringVar()
pred5 = StringVar()
```

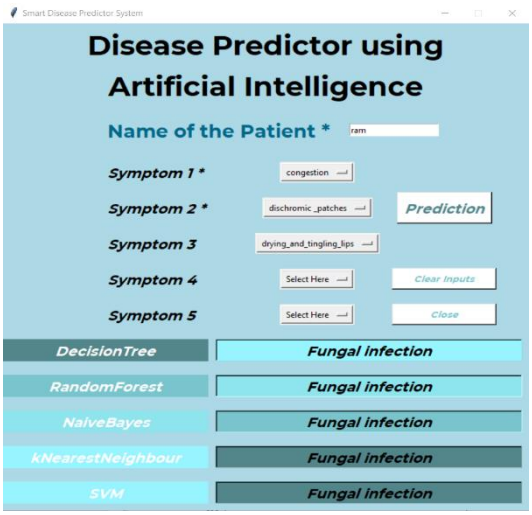
```
Symptom1 = StringVar()
Symptom1.set("Select Here")
Symptom2 = StringVar()
Symptom2.set("Select Here")
Symptom3 = StringVar()
Symptom3.set("Select Here")
Symptom4 = StringVar()
Symptom4.set("Select Here")
Symptom5 = StringVar()
Symptom5.set("Select Here")
Name = StringVar()

prev_win = None
def Reset():
    global prev_win

    Symptom1.set("Select Here")
    Symptom2.set("Select Here")
    Symptom3.set("Select Here")
    Symptom4.set("Select Here")
    Symptom5.set("Select Here")
    NameEn.delete(first=0, last=100)
    pred1.set(" ")
    pred2.set(" ")
    pred3.set(" ")
    pred4.set(" ")
    pred5.set(" ")
    try:
        prev_win.destroy()
        prev_win = None
    except AttributeError:
        pass

from tkinter import messagebox

def Exit():
    qExit = messagebox.askyesno("System", "Do you want to exit?")

    if qExit:
        root.destroy()
        exit()

w2 = Label(root, text="Disease Predictor using", fg="Black", bg="Light Blue")
w2.config(font=("Montserrat", 30, "bold"))
w2.grid(row=1, column=0, columnspan=6, padx=100)
```

```
w2 = Label(root, text="Artificial Intelligence", fg="Black", bg="Light Blue")
w2.config(font=("Montserrat", 30, "bold"))
w2.grid(row=2, column=0, columnspan=6, padx=100)

NameLb = Label(root, text="Name of the Patient *", fg="DeepSkyBlue4", bg="Light Blue")
NameLb.config(font=("Montserrat", 20, "bold"))
NameLb.grid(row=6, column=1, columnspan=4, pady=15, sticky=W)

S1Lb = Label(root, text="Symptom 1 *", fg="Black", bg="Light Blue")
S1Lb.config(font=("Montserrat", 15, "bold italic"))
S1Lb.grid(row=7, column=1, columnspan=4, pady=10, sticky=W)

S2Lb = Label(root, text="Symptom 2 *", fg="Black", bg="Light Blue")
S2Lb.config(font=("Montserrat", 15, "bold italic"))
S2Lb.grid(row=8, column=1, columnspan=4, pady=10, sticky=W)

S3Lb = Label(root, text="Symptom 3", fg="Black", bg="Light Blue")
S3Lb.config(font=("Montserrat", 15, "bold italic"))
S3Lb.grid(row=9, column=1, columnspan=4, pady=10, sticky=W)

S4Lb = Label(root, text="Symptom 4", fg="Black", bg="Light Blue")
S4Lb.config(font=("Montserrat", 15, "bold italic"))
S4Lb.grid(row=10, column=1, columnspan=4, pady=10, sticky=W)

S5Lb = Label(root, text="Symptom 5", fg="Black", bg="Light Blue")
S5Lb.config(font=("Montserrat", 15, "bold italic"))
S5Lb.grid(row=11, column=1, columnspan=4, pady=10, sticky=W)

lrLb = Label(root, text="DecisionTree", fg="white", bg="CadetBlue4", width=20)
lrLb.config(font=("Montserrat", 15, "bold italic"))
lrLb.grid(row=15, column=0, columnspan=2, pady=10)

destreeLb = Label(root, text="RandomForest", fg="White", bg="CadetBlue3", width=20)
destreeLb.config(font=("Montserrat", 15, "bold italic"))
destreeLb.grid(row=17, column=0, columnspan=2, pady=10)

ranfLb = Label(root, text="NaiveBayes", fg="White", bg="CadetBlue2", width=20)
ranfLb.config(font=("Montserrat", 15, "bold italic"))
ranfLb.grid(row=19, column=0, columnspan=2, pady=10)

knnLb = Label(root, text="kNearestNeighbour", fg="White", bg="CadetBlue1", width=20)
knnLb.config(font=("Montserrat", 15, "bold italic"))
knnLb.grid(row=21, column=0, columnspan=2, pady=10)

svmLb = Label(root, text="SVM", fg="White", bg="CadetBlue1", width=20)
```

```
svmLb.config(font=("Montserrat", 15, "bold italic"))
svmLb.grid(row=23, column=0, columnspan=2, pady=10)
OPTIONS = sorted(l1)
NameEn = Entry(root, textvariable=Name)
NameEn.grid(row=6, column=3, columnspan=4)

S1 = OptionMenu(root, Symptom1, *OPTIONS)
S1.grid(row=7, column=1, columnspan=5)

S2 = OptionMenu(root, Symptom2, *OPTIONS)
S2.grid(row=8, column=1, columnspan=5)
S3 = OptionMenu(root, Symptom3, *OPTIONS)
S3.grid(row=9, column=1, columnspan=5)
S4 = OptionMenu(root, Symptom4, *OPTIONS)
S4.grid(row=10, column=1, columnspan=5)
S5 = OptionMenu(root, Symptom5, *OPTIONS)
S5.grid(row=11, column=1, columnspan=5)
dst = Button(root, text="Prediction", command=run_fun, bg="white", fg="CadetBlue4")
dst.config(font=("Montserrat", 15, "bold italic"))
dst.grid(row=8, column=5, padx=10)
rs = Button(root, text="Clear Inputs", command=Reset, bg="white", fg="CadetBlue3", width=15)
rs.config(font=("Montserrat", 10, "bold italic"))
rs.grid(row=10, column=5, padx=10)
ex = Button(root, text="Close", command=Exit, bg="white", fg="CadetBlue3", width=15)
ex.config(font=("Montserrat", 10, "bold italic"))
ex.grid(row=11, column=5, padx=10)
t1 = Label(root, font=("Montserrat", 15, "bold italic"), text="Decision Tree", height=1, bg="CadetBlue1"
        , width=30, fg="Black", textvariable=pred1, relief="sunken").grid(row=15, column=2, columnspan=4,
padx=10)
t2 = Label(root, font=("Montserrat", 15, "bold italic"), text="Random Forest", height=1, bg="CadetBlue2"
        , width=30, fg="Black", textvariable=pred2, relief="sunken").grid(row=17, column=2, columnspan=4,
padx=10)
t3 = Label(root, font=("Montserrat", 15, "bold italic"), text="Naive Bayes", height=1, bg="CadetBlue3"
        , width=30, fg="Black", textvariable=pred3, relief="sunken").grid(row=19, column=2, columnspan=4,
padx=10)
t4 = Label(root, font=("Montserrat", 15, "bold italic"), text="kNearest Neighbour", height=1, bg="CadetBlue4"
        , width=30, fg="Black", textvariable=pred4, relief="sunken").grid(row=21, column=2, columnspan=4,
padx=10)
t5 = Label(root, font=("Montserrat", 15, "bold italic"), text="SVM", height=1, bg="CadetBlue4"
        , width=30, fg="Black", textvariable=pred5, relief="sunken").grid(row=23, column=2, columnspan=4,
padx=10)

root.mainloop()
```

Testing:

The test was performed taking 5 different symptoms combinations to find the predicted disease. The test report of the following test is given below.
Where S1, S2, S3, S4, S5 are the 5 symptoms entered and P1, P2, P3, P4, P5 are the predicted diseases by the respective algorithms.



Test no.		1	2	3	4	5
Symptoms	**S1**	Congestion	Blister	Chest Pain	Mild fever	Continuous fell of urine
	S2	Dischromic patches	Congestion	Fast Heart Rate	Red spots over body	Increase in appetite
	S3	Drying and tingling lips	Constipation	Dizziness	-	obesity
	S4	-	Bloody Stool	-	-	polyuria
	S5	-	-	-	-	Lack of concentration
Predicted Diseases	**P1 Decision tree**	Fungal infection	Dimorphic hemorrhoids (piles)	Hypertension	Drug Reaction	Diabetes
	P2 Random forest	Fungal infection	Dimorphic hemorrhoids (piles)	Hypertension	Chicken Pox	Diabetes
	P3 Naive Bayes	Fungal infection	Dimorphic hemorrhoids (piles)	Hypertension	Chicken Pox	Diabetes
	P4 KNN	Fungal infection	Drug Reaction	Heart Attack	Chicken Pox	Diabetes
	P5 SVM	Fungal infection	Dimorphic hemorrhoids (piles)	Hypertension	Chicken Pox	Diabetes

Results:

After predicting the disease based on the symptoms this is the accuracy of all the algorithms:
KNN - 96 %
Naive Bayes - 89%
Decision Tree - 92%
Random Forest - 93%
Support Vector Classifier - 91%

Conclusion:

The project we proposed has lived up to our expectations. All the 5 algorithms namely KNN, Decision Tree, Naive Bayes, Random Forest and Support Vector Classifier predicted the disease with an accuracy in the range of 88% - 96%, KNN being the highest accuracy. Any changes made in the symptoms was well understood by the algorithms and they gave their results accordingly**.**

Future Enhancements:

We are planning to incorporate the medication component, here we will include a feature which will display the prescribed drug to the user along with the disease. After including this feature, we are planning to host it on a server in the form of a web application, so it is available to all. In order to test the software, we will go to hospitals and test it on patients under the supervision of a doctor.