```
from __future__ import print_function
from tkinter import messagebox
from tkinter import *
from tkinter import simpledialog
import tkinter
from tkinter import filedialog
import matplotlib.pyplot as plt
from tkinter.filedialog import askopenfilename
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import os
import re
from sklearn.metrics import accuracy_score
import numpy as np
from sklearn import datasets, linear_model
import pandas as pd
from genetic_selection import GeneticSelectionCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
import SwarmPackagePy
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from BAT import BAT
from SwarmPackagePy import testFunctions as tf
from BEE import BEE



main = tkinter.Tk()
main.title("Heart Disease Prediction Using Bio Inspired Algorithms")
main.geometry("1300x1200")

global filename
global train
global ga_acc, bat_acc, bee_acc
global classifier



def upload():
    global filename
    filename = filedialog.askopenfilename(initialdir="heart_dataset")
    pathlabel.config(text=filename)
    text.delete('1.0', END)
    text.insert(END,filename+" loaded\n");




def prediction(X_test, cls):  #prediction done here
    y_pred = cls.predict(X_test)
    for i in range(len(X_test)):
      print("X=%s, Predicted=%s" % (X_test[i], y_pred[i]))
    return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred, details):
    cm = confusion_matrix(y_test, y_pred)
    accuracy = accuracy_score(y_test,y_pred)*100
    text.insert(END,details+"\n\n")
    text.insert(END,"Accuracy : "+str(accuracy)+"\n\n")
    text.insert(END,"Report : "+str(classification_report(y_test, y_pred))+"\n")
    text.insert(END,"Confusion Matrix : "+str(cm)+"\n\n\n\n\n")
    return accuracy



def geneticAlgorithm():
    global classifier
    text.delete('1.0', END)
    global ga_acc
    train = pd.read_csv(filename)
    test = pd.read_csv('heart_dataset/test.txt')
    test_X = test.values[:, 0:12]
    X = train.values[:, 0:12]
    y = train.values[:, 13]

    estimator = linear_model.LogisticRegression(solver="liblinear",
multi_class="ovr")

    selector = GeneticSelectionCV(estimator,
                                  cv=5,
                                  verbose=1,
                                  scoring="accuracy",
                                  max_features=10,
                                  n_population=50,
                                  crossover_proba=0.5,
                                  mutation_proba=0.2,
                                  n_generations=200,
                                  crossover_independent_proba=0.5,
                                  mutation_independent_proba=0.05,
                                  tournament_size=3,
                                  n_gen_no_change=10,
                                  caching=True,
                                  n_jobs=-1)
    selector = selector.fit(X, y)
    y_pred = selector.predict(test_X)
    prediction_data = prediction(test_X, selector)
    ga_acc = cal_accuracy(prediction_data, prediction_data,'GA Algorithm Accuracy,
Classification Report & Confusion Matrix')
    classifier = selector

def runBat():

    text.delete('1.0', END)
    global bat_acc
    train = pd.read_csv(filename)
    alh = BAT(train.values, tf.easom_function, -10, 10, 2, 20)
    data = alh.get_agents()
    X = []
    Y = []
    for i in range(len(data)):
        for j in range(len(data[i])):
            X.append(data[i][j][0:13])
            Y.append(data[i][j][13])

    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.1,
random_state = 0)
    cls =
RandomForestClassifier(n_estimators=50,max_depth=2,random_state=0,class_weig
ht='balanced')
    cls.fit(X_train, y_train)
    prediction_data = prediction(X_test, cls)
    bat_acc = cal_accuracy(y_test, prediction_data,'BAT Algorithm Accuracy,
Classification Report & Confusion Matrix')
    def runBee():
    text.delete('1.0', END)
    global bee_acc
    train = pd.read_csv(filename)
    alh = BEE(train.values, tf.easom_function, -10, 10, 2, 20)
    data = alh.get_agents()
    X = []
    Y = []
    for i in range(len(data)):
        for j in range(len(data[i])):
            X.append(data[i][j][0:13])
            Y.append(data[i][j][13])

    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.1,
random_state = 0)
```

```
    cls                                                                                          =
RandomForestClassifier(n_estimators=30,max_depth=2,random_state=0,class_weig
ht='balanced')
    cls.fit(X_train, y_train)
    prediction_data = prediction(X_test, cls)
    bee_acc  =  cal_accuracy(y_test,  prediction_data,'ABE  Algorithm  Accuracy,
Classification Report & Confusion Matrix')



def predict():
    text.delete('1.0', END)
    filename = filedialog.askopenfilename(initialdir="dataset")
    test = pd.read_csv(filename)
    test = test.values[:, 0:12]
    total = len(test)
    text.insert(END,filename+" test file loaded\n");
    y_pred = classifier.predict(test)
    for i in range(len(test)):
        print(str(y_pred[i]))
        if str(y_pred[i]) == '0.0':
            text.insert(END,"X=%s, Predicted = %s" % (test[i], 'No disease detected')+"\n\n")
        if str(y_pred[i]) == '1.0':
            text.insert(END,"X=%s, Predicted = %s" % (test[i], 'Stage 1 Disease Detected')+"\n\n")
        if str(y_pred[i]) == '2.0':
            text.insert(END,"X=%s, Predicted = %s" % (test[i], 'Stage 2 Disease Detected')+"\n\n")
        if str(y_pred[i]) == '3.0':
            text.insert(END,"X=%s, Predicted = %s" % (test[i], 'Stage 3 Disease Detected')+"\n\n")
        if str(y_pred[i]) == '4.0':
            text.insert(END,"X=%s, Predicted = %s" % (test[i], 'Stage 4 Disease Detected')+"\n\n")




def graph():
    height = [ga_acc,bat_acc,bee_acc]
    bars = ('Genetic Algorithm','Bat Algorithm','Bee Algorithm')
    y_pos = np.arange(len(bars))
    plt.bar(y_pos, height)
    plt.xticks(y_pos, bars)
    plt.show()
def exit():
    main.destroy()
font = ('times', 16, 'bold')
title = Label(main, text='Heart Disease Prediction Using Bio Inspired Algorithms')
title.config(bg='brown', fg='white')
title.config(font=font)
title.config(height=3, width=120)
title.place(x=0,y=5)

font1 = ('times', 14, 'bold')
uploadButton = Button(main, text="Upload Heart Disease", command=upload)
uploadButton.place(x=50,y=100)
uploadButton.config(font=font1)
pathlabel = Label(main)
pathlabel.config(bg='brown', fg='white')
pathlabel.config(font=font1)
pathlabel.place(x=460,y=100)
geneticButton = Button(main, text="Run Genetic Algorithm", command=geneticAlgorithm)
geneticButton.place(x=50,y=150)
geneticButton.config(font=font1)
batButton = Button(main, text="Run BAT Algorithm", command=runBat)
batButton.place(x=330,y=150)
batButton.config(font=font1)

beeButton = Button(main, text="Run BEE Algorithm", command=runBee)
beeButton.place(x=620,y=150)
beeButton.config(font=font1)

predictButton = Button(main, text="Upload & Predict Test Data", command=predict)
```

```
predictButton.place(x=850,y=150)
predictButton.config(font=font1)

graphButton = Button(main, text="Accuracy Graph", command=graph)
graphButton.place(x=50,y=200)
graphButton.config(font=font1)

exitButton = Button(main, text="Exit", command=exit)
exitButton.place(x=330,y=200)
exitButton.config(font=font1)



font1 = ('times', 12, 'bold')
text=Text(main,height=20,width=150)
scroll=Scrollbar(text)
text.configure(yscrollcommand=scroll.set)
text.place(x=10,y=250)
text.config(font=font1)



main.config(bg='brown')
main.mainloop()
```

GA.py:

```
from __future__ import print_function
import numpy as np
from sklearn import datasets, linear_model
import pandas as pd

from genetic_selection import GeneticSelectionCV



def main():
    train = pd.read_csv('heart_dataset/dataset')
    test = pd.read_csv('heart_dataset/test.txt')
    test_X = test.values[:, 0:12]
    X = train.values[:, 0:12]
    y = train.values[:, 13]

    estimator = linear_model.LogisticRegression(solver="liblinear", multi_class="ovr")

    selector = GeneticSelectionCV(estimator,
                                  cv=5,
                                  verbose=1,
                                  scoring="accuracy",
                                  max_features=10,
                                  n_population=50,
                                  crossover_proba=0.5,
                                  mutation_proba=0.2,
                                  n_generations=200,
                                  crossover_independent_proba=0.5,
                                  mutation_independent_proba=0.05,
                                  tournament_size=3,
                                  n_gen_no_change=10,
                                  caching=True,
                                  n_jobs=-1)
    selector = selector.fit(X, y)

    print(selector.support_)
    y_pred = selector.predict(test_X)
    print(y_pred)

if __name__ == "__main__":
    main()
```

ACO.py:

```
import random as rn
import numpy as np
from numpy.random import choice as np_choice
```

```
class ACO(object):

    def __init__(self, distances, n_ants, n_best, n_iterations, decay, alpha=1, beta=1):
        """
        Args:
            distances (2D numpy.array): Square matrix of distances. Diagonal is assumed
to be np.inf.
            n_ants (int): Number of ants running per iteration
            n_best (int): Number of best ants who deposit pheromone
            n_iteration (int): Number of iterations
            decay (float): Rate it which pheromone decays. The pheromone value is
multiplied by decay, so 0.95 will lead to decay, 0.5 to much faster decay.
             alpha (int or float): exponenet on pheromone, higher alpha gives pheromone
  more weight. Default=1
             beta (int or float): exponent on distance, higher beta give distance more
  weight. Default=1
         Example:
            ant_colony = AntColony(german_distances, 100, 20, 2000, 0.95, alpha=1,
beta=2)
        """
        self.distances  = distances
        self.pheromone = np.ones(self.distances.shape) / len(distances)
        self.all_inds = range(len(distances))
        self.n_ants = n_ants
        self.n_best = n_best
        self.n_iterations = n_iterations
        self.decay = decay
        self.alpha = alpha
        self.beta = beta

    def run(self):
        shortest_path = None
        all_time_shortest_path = ("placeholder", np.inf)
        for i in range(self.n_iterations):
            all_paths = self.gen_all_paths()
            self.spread_pheronome(all_paths, self.n_best, shortest_path=shortest_path)
            shortest_path = min(all_paths, key=lambda x: x[1])
            print (shortest_path)
            if shortest_path[1] < all_time_shortest_path[1]:
                all_time_shortest_path = shortest_path
            self.pheromone * self.decay
        return all_time_shortest_path

    def spread_pheronome(self, all_paths, n_best, shortest_path):
        sorted_paths = sorted(all_paths, key=lambda x: x[1])
        for path, dist in sorted_paths[:n_best]:
            for move in path:
                self.pheromone[move] += 1.0 / self.distances[move]

    def gen_path_dist(self, path):
        total_dist = 0
        for ele in path:
            total_dist += self.distances[ele]
        return total_dist

    def gen_all_paths(self):
        all_paths = []
        for i in range(self.n_ants):
            path = self.gen_path(0)
            all_paths.append((path, self.gen_path_dist(path)))
        return all_paths

    def gen_path(self, start):
        path = []
        visited = set()
        visited.add(start)
        prev = start
        for i in range(len(self.distances) - 1):
            move = self.pick_move(self.pheromone[prev], self.distances[prev], visited)
            path.append((prev, move))
            prev = move
            visited.add(move)
        path.append((prev, start)) # going back to where we started
        return path

    def pick_move(self, pheromone, dist, visited):
        pheromone = np.copy(pheromone)
        pheromone[list(visited)] = 0

        row = pheromone ** self.alpha * (( 1.0 / dist) ** self.beta)


        norm_row = row / row.sum()
        move = np_choice(self.all_inds, 1, p=norm_row)[0]
        return move
if __name__ == "__main__":
  distances = np.array([[np.inf, 2, 2, 5, 7],
                    [2, np.inf, 4, 8, 2],
                    [2, 4, np.inf, 1, 3],
                    [5, 8, 1, np.inf, 2],
                    [7, 2, 3, 2, np.inf]])

  ant_colony = ACO(distances, 1, 1, 100, 0.95, alpha=1, beta=1)
  shortest_path = ant_colony.run()
  print ("shorted_path: {}".format(shortest_path))
```