CARDIOVASCULAR DISEASE PREDICTION USING GENETIC ALGORITHM

Ghanashyam Vagale, Nikhil Gupta, Siddharth Mittal, Deepti Botlaguduru, Hardique Dasore, Ashee Kanungo

---***---

ABSTRACT:

Cardiovascular diseases are those diseases that related to heart . Heart diseases are notshort term diseases like fever or cold . They take years of time to diagnose and are hardto detect and predict based on symptoms . It is a major cause of morbidity and transience in the modern society . Diagnosis of cardiovascular disease using various medical tests is an important but complicated task which should be performed accurately. If there are any errors or mistakes in those predictions , the life of patient might be in danger. Hence a Powerful tool in the prediction of heart disease with lowercost has Become the need of time. Detection of such cardiovascular i.e heart diseases might be done with the help of some common symptoms like regular illness or even be predicted using risk factors such as age, family history diabetes ,hypertension ,high cholesterol, tobacco smoking, alcohol intake ,obesity or physical in-activity ,etc.

A very scarce number of the systems predict heart diseases based on these risk factors.Heart disease patients have lot of these visible risk Factors in common which can be used very effectively for diagnosis. System based on such risk factors would not only help medical Professionals but it would give patients a warning about the probable Presence of heart disease even before he visits a hospital. In this, we will Apply ANNand binary classification to the dataset which is nothing but the risk factors , for Prediction and training of network .

Key words: Cardiovascular diseases, genetic algorithm, neuro adaptive capability,ANN , Binary classification

1. INTRODUCTION:

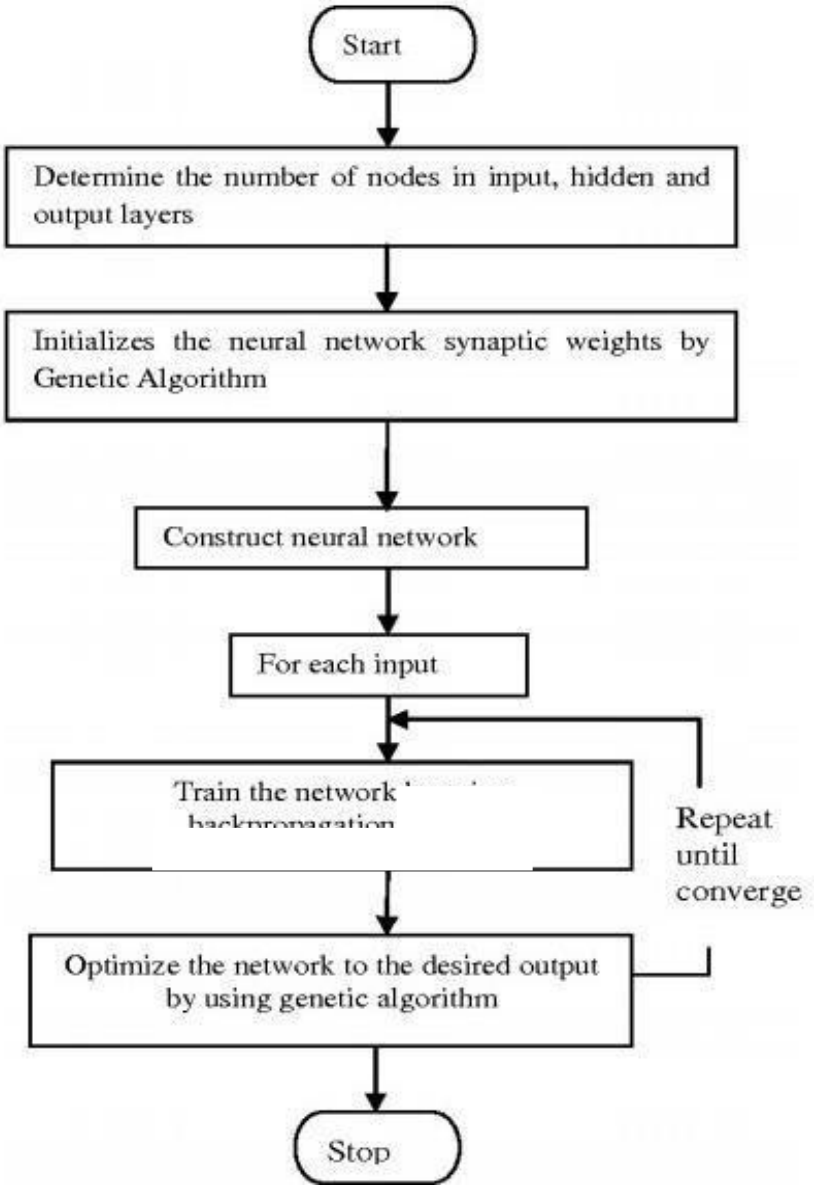
In medical diagnosis, the information provided by the patients may Include redundant and interrelated symptoms and signs especially when the patients suffer from more than one type of disease of same category. The physicians may not able to diagnose it correctly. So it is necessary to identify the important diagnostic features of a disease and this may facilitate the physicians to diagnose the disease early and correctly. Genetic algorithms are commonly used for better solution due to its operators like selection, crossover and mutation .Accurate and reliable decision making in cardiological prognosis can help in the planning of suitable surgery and therapy, and generally, improve patient management through the different stages of the disease.

Prediction of diseases isn't an easy task to perform. We might even need more than one soft computing and machine learning, data mining techniques to understand the situation and predict.

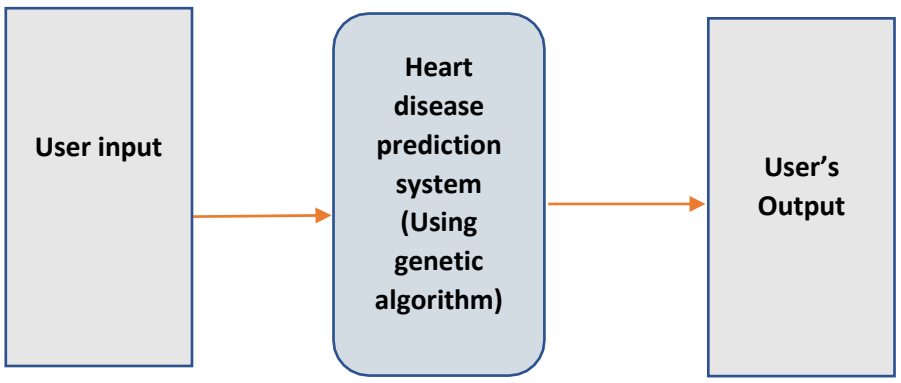
The proposed problem thus obviously is related to unawareness among people and their resulting disregard for proper medical care especially related to cardio-logical problems. Thus this system aims to spread awareness among people by accurately predicting if they are at a potential risk of contracting a heart disease and thereby make them pro-active In making healthier life choices and follow regular check ups .Our main aim in this review is to develop a heart disease prediction system, check its accuracy and verify if it is optimal using genetic algorithm and compare with an ANNto verify if the solution provided by Genetic algorithm is ok .

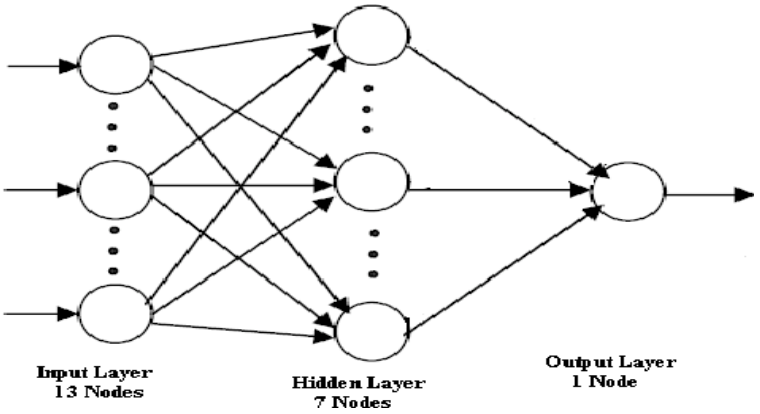
2) General Architecture

a) For genetic algorithm:



b) Genetic algorithm general architecture :





3) Dataset specification:

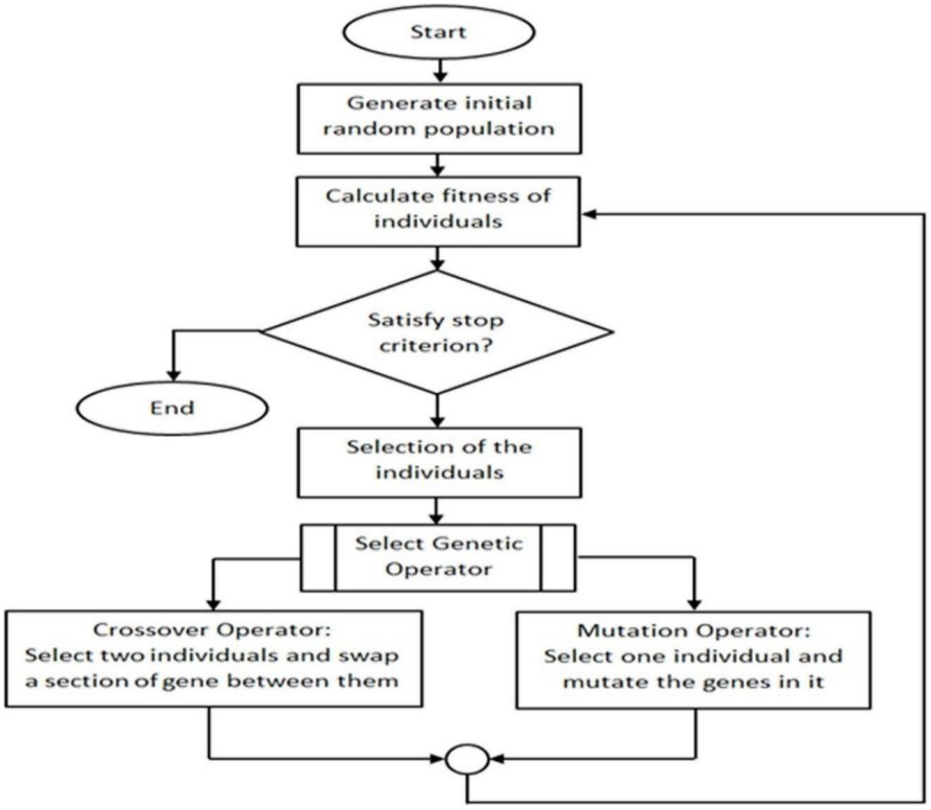
Heart dataset:

About dataset:

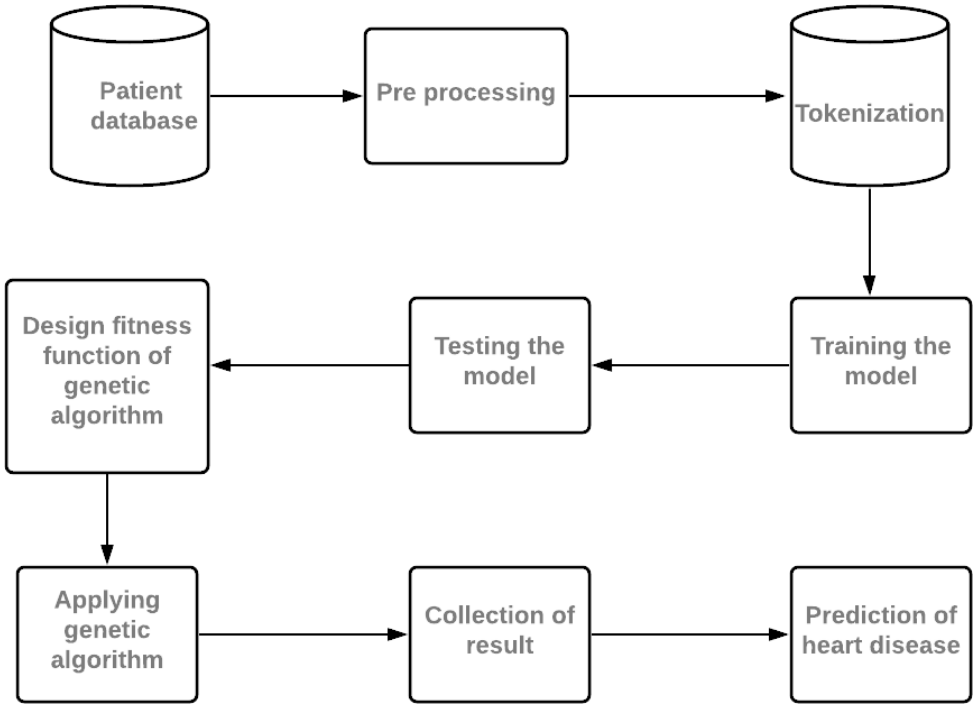
Data Set Characteristics:	Multivariate	**Number of Instances:**	303	**Area:**	Life
Attribute Characteristics:	Categorical, Integer, Real	**Number of Attributes:**	75	**Date Donated**	1988-07-01
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number ofWeb Hits:**	1469955

1. SYSTEM DESIGN

1.1. Architecture Diagram / Flow Diagram / Flowchart



System architecture:



5.2 Detailed Description of Modules

Patient database: Generally in hospitals certain patient databases are maintained for future use. We are using that data as our dataset from Kaggle website. This data is themain source of our project and with the help of database we will perform all the otheroperations in the flow chart.

Pre-processing: We have two types of attributes in the database; primary attributes(more important) and secondary attributes(less important). Through pre- processing we will refine the data by separating more important attributes from less one.

Tokenization: It is the process of turning sensitive data into non-sensitive data called "tokens" that can be used in a database or internal system without bringing it into scope. Tokenization can be used to secure sensitive data by replacing the original datawith an unrelated value of the same length and format. We will replace the fuzzy valuesof the data as crisp values and change the data into bit strings so that the data can be easily used in genetic algorithm.

Training the model: Training of the model is done by artificial neural network in which we will perform updation of weights with the help of old weights present in database. Then by using threshold value and activation function according to the dataobtained we will compare and provide the output and updated weights as results.

Testing the model: Testing the gained results provide the accuracy of the model. We are performing testing through genetic algorithm as the best fitted chromosomes survives and the least fitted will be dead. This mechanism gives the performance of the model. The decision variable 'x' is coded into finite length string and initial population is selected randomly.

Designing fitness of genetic algorithm: Fitness Function (also known as the Evaluation Function) evaluates how close a given solution is to the optimum solution of the desired problem. It determines how fit a solution is. Then 'x' values aredecoded for initial population.

Applying genetic algorithm: Here genetic algorithm comes into action. Genetic Algorithm (GA) is a search-based optimization technique based on the principles of Genetics and Natural Selection. It is frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve. The sub tasks of genetic algorithm like producing child chromosomes from parent chromosomes is done by "crossover" and "mutation" techniques.

Collection of results: After crossover and mutation we will get best score of the child chromosomes and matched against their respected parent fitness score. If the child's score is greater than parents then child is best fitted and it can proceed for further survival, otherwise we have to repeat from testing module again till we get the best score.

Prediction of heart disease: With the help of artificial neural network and genetic algorithm we can predict the accuracy of the model. Genetic based neural network is used for training the system. The final weights of the neural network are stored in theweight base and are used for predicting the risk of cardiovascular disease. Theclassification accuracy obtained using this approach is 81.3%.

2. SOFTWARE REQUIREMENTS SPECIFICATION

The code of genetic algorithm and artificial neural networks is in python programming language. We have used Jupyter notebook platform for writing and executing the code. Installing the latest Jupyter notebook on updated Windows 10 will help us importing new libraries. Jupyter is a project and community whose goal is to "develop open- source software, open-standards, and services for interactive computing across dozensof programming languages".

2.1. Output :

1. Data import and pre-processing :

[1]:
```
import sys
import pandas as pd
import numpy as np
import sklearn
import matplotlib
import keras
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
import seaborn as sns
cleveland = pd.read_csv('heart.csv')
cleveland.loc[0:]
```

[1]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

Fig1: Importing the data from the Kaggle website with 303 rows x 14 columns

```
[2]: # remove missing data (indicated with a "?")
data = cleveland[~cleveland.isin(['?'])]
data.loc[0:]
```

[2]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

Fig2: removing the missing data from the table

```
[3]: # drop rows with NaN values from DataFrame
data = data.dropna(axis=0)
data.loc[0:]
```

[3]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

303 rows × 14 columns

Fig3: Dropping the rows with NaN values from the table

[4]:
```
# drop rows with NaN values from DataFrame
data = data.dropna(axis=0)
data.loc[0:]
# print the shape and data type of the dataframe
print(data.shape)
print(data.dtypes)
# print data characteristics, usings pandas built-in describe() function
data.describe()
```

```
(303, 14)
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

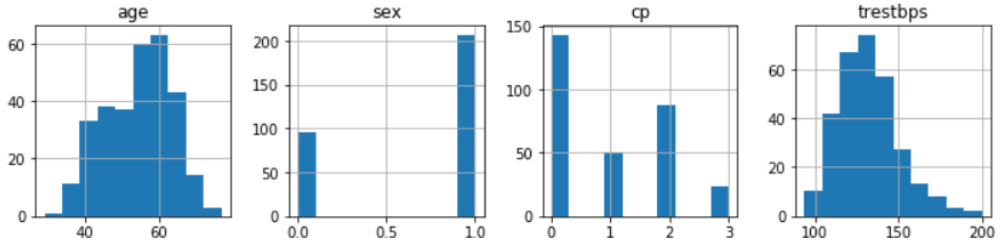
[4]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0.544554
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	0.498835
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

Fig4: After removing missing and NaN values from the table

2. Data visualization :

[5]:
```
# plot histograms for each variable
data.hist(figsize = (12, 12))
plt.show()
pd.crosstab(data.age,data.target).plot(kind="bar",figsize=(20,6))
plt.title('Heart Disease Frequency for Ages')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
plt.figure(figsize=(10,10))
sns.heatmap(data.corr(),annot=True,fmt='.1f')
plt.show()
```



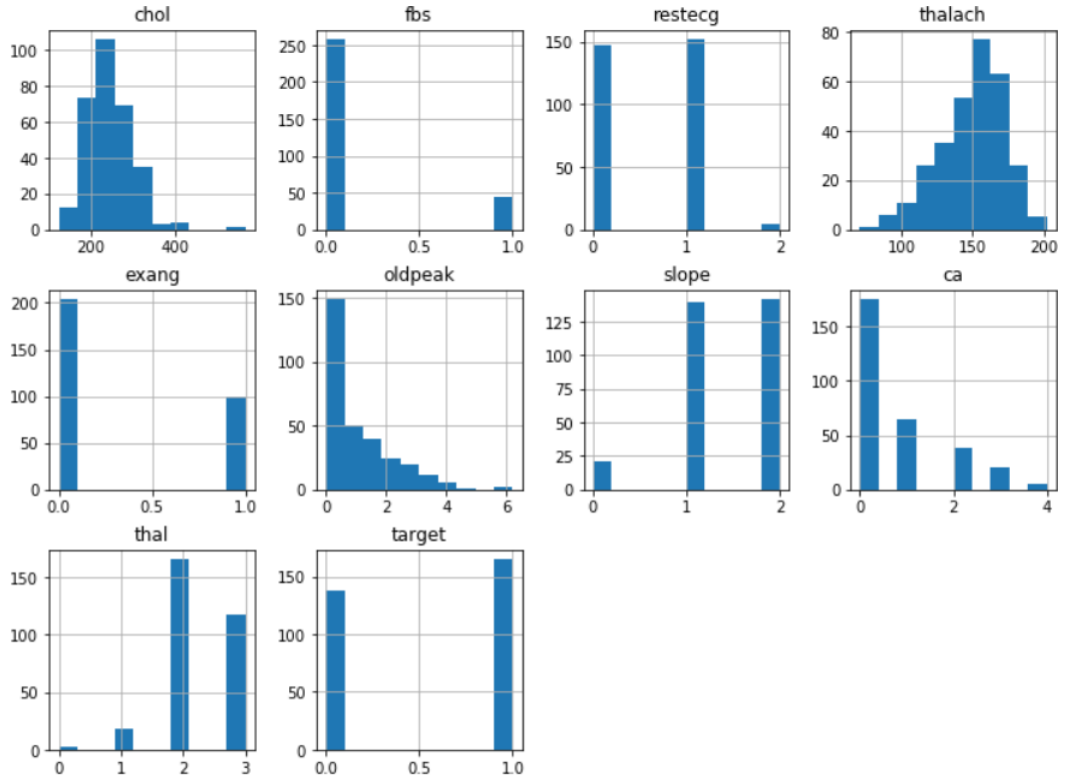


Fig5: Histograms of every attribute in the data

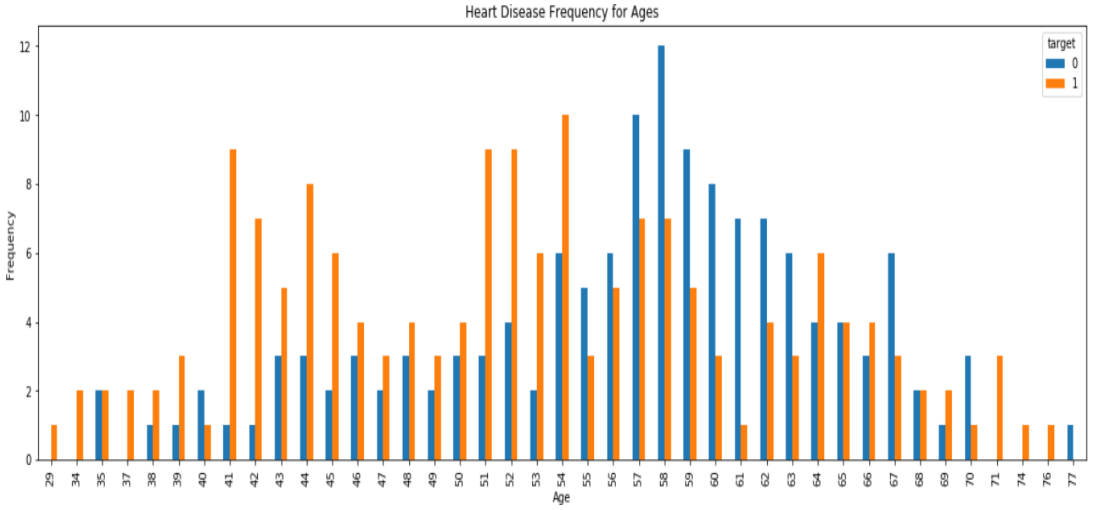


Fig6: Heart disease frequency for ages with targets-0,1

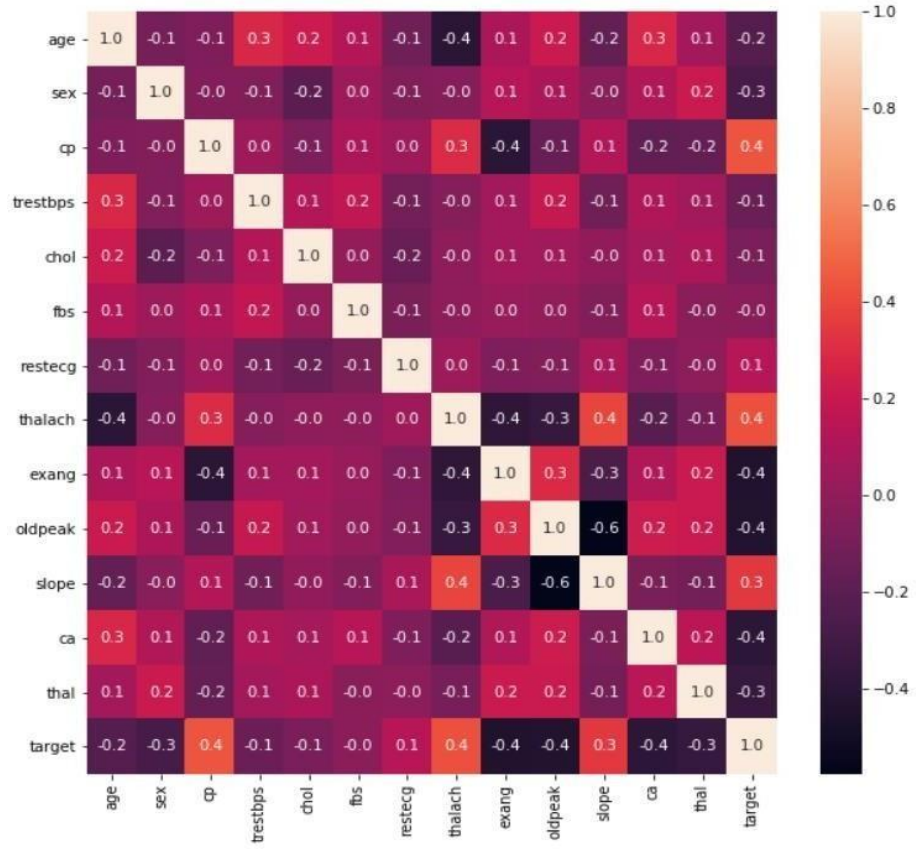


Fig7: Matrix representation of linear data with its own attributes

3. Training and Testing

```
[7]: X = np.array(data.drop(['target'], 1))
     y = np.array(data['target'])
     mean = X.mean(axis=0)
     X -= mean
     std = X.std(axis=0)
     X /= std
     # create X and Y datasets for training
     from sklearn import model_selection
     X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, stratify=y, random_state=42, test_size = 0.2)
     from keras.utils.np_utils import to_categorical
     Y_train = to_categorical(y_train, num_classes=None)
     Y_test = to_categorical(y_test, num_classes=None)
     print (Y_train.shape)
     print (Y_train[:10])
     X_train[0]

     (242, 2)
     [[0. 1.]
      [1. 0.]
      [1. 0.]
      [1. 0.]
      [0. 1.]
      [0. 1.]
      [0. 1.]
      [0. 1.]
      [1. 0.]
      [0. 1.]]
[7]: array([ 1.61392956, -1.46841752,  1.97312292,  0.47839125, -0.14038081,
            -0.41763453,  0.89896224,  0.05917329, -0.69663055,  0.65599028,
             0.97635214,  1.24459328, -0.51292188])
```

Fig8: Training the data using ANN algorithm-only 80% data is used

```
[8]: from keras.models import Sequential
     from keras.layers import Dense
     from tensorflow.keras.optimizers import Adam
     from keras.layers import Dropout
     from keras import regularizers
     # define a function to build the keras model
     def create_model():
         # create model
         model = Sequential()
         model.add(Dense(16, input_dim=13, kernel_initializer='normal', kernel_regularizer=regularizers.l2(0.001), activation='relu'))
         model.add(Dropout(0.25))
         model.add(Dense(8, kernel_initializer='normal', kernel_regularizer=regularizers.l2(0.001), activation='relu'))
         model.add(Dropout(0.25))
         model.add(Dense(2, activation='softmax'))
         # compile model
         adam = Adam(learning_rate=0.001)
         model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
         return model
     model = create_model()
     print(model.summary())
     # fit the model to the training data
     history=model.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=50, batch_size=10)
     # fit the model to the training data
     history=model.fit(X_train, Y_train, validation_data=(X_test, Y_test),epochs=50, batch_size=10)
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 16)                224
_________________________________________________________________
dropout (Dropout)            (None, 16)                0
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 136
_________________________________________________________________
dropout_1 (Dropout)          (None, 8)                 0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 18
=================================================================
Total params: 378
Trainable params: 378
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/50
25/25 [==============================] - 33s 43ms/step - loss: 0.6834 - accuracy: 0.6699 - val_loss: 0.6656 - val_accuracy: 0.7049
Epoch 2/50
25/25 [==============================] - 0s 6ms/step - loss: 0.6440 - accuracy: 0.7283 - val_loss: 0.6278 - val_accuracy: 0.7541
Epoch 3/50
25/25 [==============================] - 0s 4ms/step - loss: 0.6224 - accuracy: 0.7941 - val_loss: 0.5852 - val_accuracy: 0.7705
Epoch 4/50
25/25 [==============================] - 0s 4ms/step - loss: 0.5372 - accuracy: 0.8433 - val_loss: 0.5439 - val_accuracy: 0.7705
Epoch 5/50
25/25 [==============================] - 0s 5ms/step - loss: 0.4845 - accuracy: 0.8457 - val_loss: 0.5075 - val_accuracy: 0.7705
Epoch 6/50
25/25 [==============================] - 0s 6ms/step - loss: 0.4766 - accuracy: 0.8498 - val_loss: 0.4794 - val_accuracy: 0.7869
Epoch 7/50
25/25 [==============================] - 0s 4ms/step - loss: 0.4287 - accuracy: 0.8215 - val_loss: 0.4545 - val_accuracy: 0.7869
Epoch 8/50
25/25 [==============================] - 0s 4ms/step - loss: 0.4186 - accuracy: 0.8475 - val_loss: 0.4402 - val_accuracy: 0.7869
Epoch 9/50


Epoch 45/50
25/25 [==============================] - 0s 4ms/step - loss: 0.2620 - accuracy: 0.9215 - val_loss: 0.4782 - val_accuracy: 0.7869
Epoch 46/50
25/25 [==============================] - 0s 5ms/step - loss: 0.2807 - accuracy: 0.8926 - val_loss: 0.4870 - val_accuracy: 0.7869
Epoch 47/50
25/25 [==============================] - 0s 6ms/step - loss: 0.2732 - accuracy: 0.9008 - val_loss: 0.4855 - val_accuracy: 0.7869
Epoch 48/50
25/25 [==============================] - 0s 5ms/step - loss: 0.2512 - accuracy: 0.9215 - val_loss: 0.4908 - val_accuracy: 0.7869
Epoch 49/50
25/25 [==============================] - 0s 4ms/step - loss: 0.2938 - accuracy: 0.9050 - val_loss: 0.4902 - val_accuracy: 0.7869
Epoch 50/50
25/25 [==============================] - 0s 5ms/step - loss: 0.2818 - accuracy: 0.9050 - val_loss: 0.4976 - val_accuracy: 0.7869
```

Fig9: Testing the data with accuracy and loss-20% of data is used

```
[11]: plt.plot(history.history['loss'])
      plt.plot(history.history['val_loss'])
      plt.title('Model Loss')
      plt.ylabel('loss')
      plt.xlabel('epoch')
      plt.legend(['train', 'test'])
      plt.show()
```

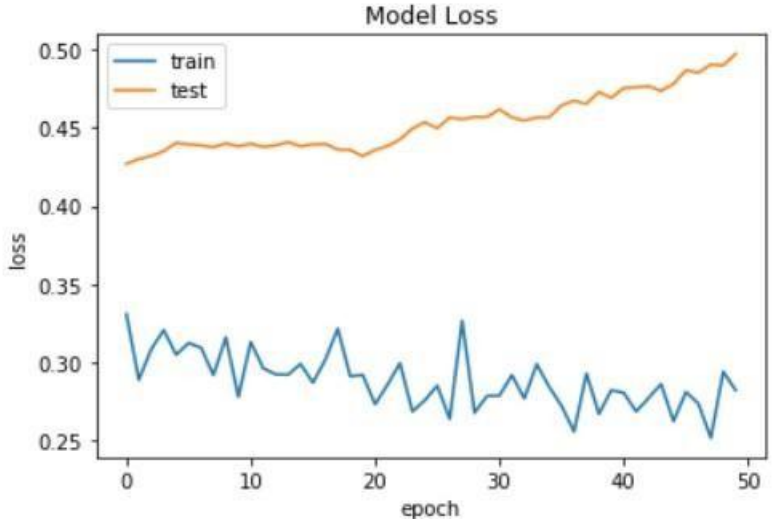


Fig10: Model loss graph forANN

```
[14]: import matplotlib.pyplot as plt
      %matplotlib inline
      # Model accuracy
      plt.plot(history.history['accuracy'])
      plt.plot(history.history['val_accuracy'])
      plt.title('Model Accuracy')
      plt.ylabel('accuracy')
      plt.xlabel('epoch')
      plt.legend(['train', 'test'])
      plt.show()
```

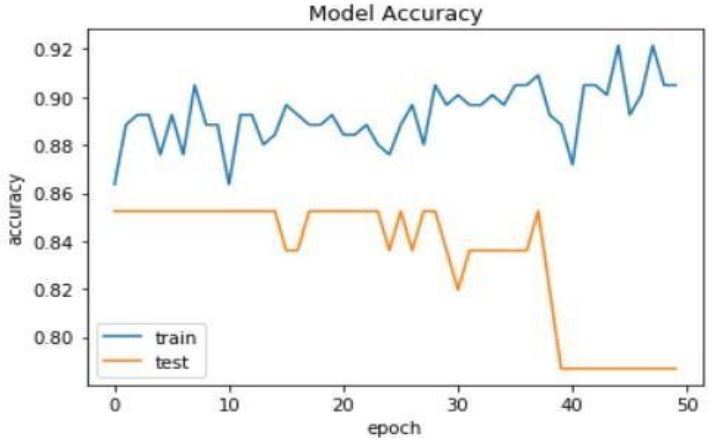


Fig11: Model accuracy graph for ANN

```
[15]: Y_train_binary = y_train.copy()
Y_test_binary = y_test.copy()
Y_train_binary[Y_train_binary > 0] = 1
Y_test_binary[Y_test_binary > 0] = 1
print(Y_train_binary[:20])
# define a new keras model for binary classification
def create_binary_model():
    # create model
    model = Sequential()
    model.add(Dense(16, input_dim=13, kernel_initializer='normal',  kernel_regularizer=regularizers.l2(0.001),activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(8, kernel_initializer='normal',  kernel_regularizer=regularizers.l2(0.001),activation='relu'))
    model.add(Dropout(0.25))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    adam = Adam(learning_rate=0.001)
    model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
    return model
binary_model = create_binary_model()
print(binary_model.summary())
# fit the binary model on the training data
history=binary_model.fit(X_train, Y_train_binary, validation_data=(X_test, Y_test_binary), epochs=50, batch_size=10)
```

```
[1 0 0 0 1 1 1 1 0 1 0 1 0 0 0 1 0 0 1 1]
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 16)                224
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_4 (Dense)              (None, 8)                 136
_________________________________________________________________
dropout_3 (Dropout)          (None, 8)                 0
_________________________________________________________________
```

Fig12: Training of data with binary classification algorithm

```
[1 0 0 0 1 1 1 1 0 1 0 1 0 0 0 1 0 0 1 1]
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 16)                224
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_4 (Dense)              (None, 8)                 136
_________________________________________________________________
dropout_3 (Dropout)          (None, 8)                 0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 9
=================================================================
Total params: 369
Trainable params: 369
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/50
25/25 [==============================] - 2s 26ms/step - loss: 0.6891 - accuracy: 0.6683 - val_loss: 0.6775 - val_accuracy: 0.7869
Epoch 2/50
25/25 [==============================] - 0s 4ms/step - loss: 0.6662 - accuracy: 0.7868 - val_loss: 0.6542 - val_accuracy: 0.7705
Epoch 3/50
25/25 [==============================] - 0s 4ms/step - loss: 0.6405 - accuracy: 0.8233 - val_loss: 0.6184 - val_accuracy: 0.8033
Epoch 4/50
25/25 [==============================] - 0s 4ms/step - loss: 0.6096 - accuracy: 0.7832 - val_loss: 0.5805 - val_accuracy: 0.7869
Epoch 5/50
25/25 [==============================] - 0s 5ms/step - loss: 0.5501 - accuracy: 0.8331 - val_loss: 0.5444 - val_accuracy: 0.7705
Epoch 6/50
25/25 [==============================] - 0s 4ms/step - loss: 0.5132 - accuracy: 0.8236 - val_loss: 0.5099 - val_accuracy: 0.7869
Epoch 7/50
25/25 [==============================] - 0s 6ms/step - loss: 0.4890 - accuracy: 0.8562 - val_loss: 0.4833 - val_accuracy: 0.8033
Epoch 8/50
25/25 [==============================] - 0s 5ms/step - loss: 0.4638 - accuracy: 0.8302 - val_loss: 0.4585 - val_accuracy: 0.7869
Epoch 9/50

Epoch 45/50
25/25 [==============================] - 0s 5ms/step - loss: 0.3298 - accuracy: 0.9016 - val_loss: 0.3945 - val_accuracy: 0.8197
Epoch 46/50
25/25 [==============================] - 0s 4ms/step - loss: 0.3274 - accuracy: 0.8525 - val_loss: 0.3936 - val_accuracy: 0.8197
Epoch 47/50
25/25 [==============================] - 0s 4ms/step - loss: 0.3076 - accuracy: 0.8889 - val_loss: 0.3959 - val_accuracy: 0.8361
Epoch 48/50
25/25 [==============================] - 0s 4ms/step - loss: 0.2897 - accuracy: 0.9086 - val_loss: 0.3970 - val_accuracy: 0.8361
Epoch 49/50
25/25 [==============================] - 0s 5ms/step - loss: 0.3703 - accuracy: 0.8635 - val_loss: 0.3961 - val_accuracy: 0.8361
Epoch 50/50
25/25 [==============================] - 0s 5ms/step - loss: 0.3307 - accuracy: 0.8624 - val_loss: 0.3967 - val_accuracy: 0.8197
```

Fig13: Testing of data with binary classification algorithm

```
[18]: # Model Losss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()
```

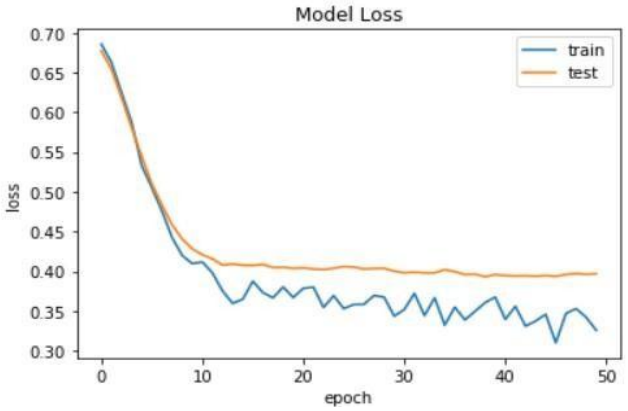


Fig14: Model accuracy graph for binary classification

```
[17]: import matplotlib.pyplot as plt
%matplotlib inline
# Model accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'])
plt.show()
```

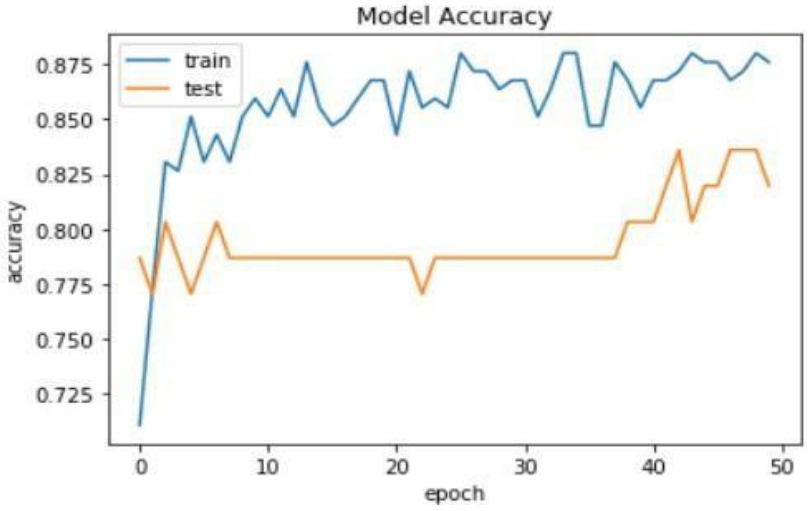


Fig15: Model loss graph for binary classification

4. Genetic Algorithm (GA)

```
[131]: #initialize population
import random
best=-100000
populations =[[1, 0, 0, 0,1],[1, 1, 1, 0,1], [0, 1, 0, 0, 0],[1, 0, 0, 1, 1]]
parents=[]
new_populations = []
print(populations)
```

```
<class 'list'>
[[1, 0, 0, 0, 1], [1, 1, 1, 0, 1], [0, 1, 0, 0, 0], [1, 0, 0, 1, 1]]
```

```
[101]: #fitness score calculation .....
def fitness_score() :
    global populations,best
    fit_value = []
    fit_score=[]
    for i in range(4) :
        chromosome_value=0
        for j in range(4,0,-1) :
            chromosome_value += populations[i][j]*(2**(4-j))
        chromosome_value = -1*chromosome_value if populations[i][0]==1 else chromosome_value
        print(chromosome_value)
        fit_value.append(-(chromosome_value**2) + 5 )
    print(fit_value)
    fit_value, populations = zip(*sorted(zip(fit_value, populations) , reverse = True))
    best= fit_value[0]
fitness_score()
```

```
-1
-13
8
-3
[4, -164, -59, -4]
```

Fig16: Initializing population and calculating fitness score of parent chromosomes

```
[102]: #print(type(populations))
#selecting parents....
def selectparent():
    global parents
    #global populations , parents
    parents=populations[0:2]
    print(type(parents))
    print(parents)
selectparent()
```

```
<class 'tuple'>
([1, 0, 0, 0, 1], [1, 0, 0, 1, 1])
```

```
[103]: #single-point crossover .........
def crossover() :
    global parents
    cross_point = random.randint(0,5)
    parents=parents + tuple([(parents[0][0:cross_point +1] +parents[1][cross_point+1:6])])
    parents =parents+ tuple([(parents[1][0:cross_point +1] +parents[0][cross_point+1:6])])
    print(parents)
crossover()
```

```
([1, 0, 0, 0, 1], [1, 0, 0, 1, 1], [1, 0, 0, 0, 1], [1, 0, 0, 1, 1])
```

Fig17: Selecting parent chromosomes using fitness score and performing crossover

[104]:
```
def mutation() :
    global populations, parents
    mute = random.randint(0,49)
    if mute == 20 :
        x=random.randint(0,3)
        y = random.randint(0,4)
        parents[x][y] = 1-parents[x][y]
    populations = parents
    print(populations)
mutation()
```

```
([1, 0, 0, 0, 1], [1, 0, 0, 1, 1], [1, 0, 0, 0, 1], [1, 0, 0, 1, 1])
```

Fig18: Performing mutation and getting children chromosomes

[105]:
```
for i in range(1000) :
    fitness_score()
    selectparent()
    crossover()
    mutation()
print("best score :")
print(best)
print("sequence........")
print(populations[0])
```

```
-1
-3
-1
-3
[4, -4, 4, -4]
<class 'tuple'>
([1, 0, 0, 0, 1], [1, 0, 0, 0, 1])
([1, 0, 0, 0, 1], [1, 0, 0, 0, 1], [1, 0, 0, 0, 1], [1, 0, 0, 0, 1])
([1, 0, 0, 0, 1], [1, 0, 0, 0, 1], [1, 0, 0, 0, 1], [1, 0, 0, 0, 1])
-1
-1
-1
-1
[4, 4, 4, 4]
<class 'tuple'>
([1, 0, 0, 0, 1], [1, 0, 0, 0, 1])
([1, 0, 0, 0, 1], [1, 0, 0, 0, 1], [1, 0, 0, 0, 1], [1, 0, 0, 0, 1])
([1, 0, 0, 0, 1], [1, 0, 0, 0, 1], [1, 0, 0, 0, 1], [1, 0, 0, 0, 1])
-1
-1
-1
```

```
0
0
0
0
[5, 5, 5, 5]
<class 'tuple'>
([1, 0, 0, 0, 0], [1, 0, 0, 0, 0])
([1, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0, 0, 0])
([1, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0, 0, 0])
0
0
0
0
[5, 5, 5, 5]
<class 'tuple'>
([1, 0, 0, 0, 0], [1, 0, 0, 0, 0])
([1, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0, 0, 0])
([1, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0, 0, 0])
0
0
0
0
[5, 5, 5, 5]
<class 'tuple'>
([1, 0, 0, 0, 0], [1, 0, 0, 0, 0])
([1, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0, 0, 0])
([1, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0, 0, 0], [1, 0, 0, 0, 0])
best score :
5
sequence........
[1, 0, 0, 0, 0]
```

Fig19: Calculating best score of children chromosome

5. Final Results :

```
[26]: #Results:
# generate classification report using predictions for categorical model
from sklearn.metrics import classification_report, accuracy_score

categorical_pred = np.argmax(model.predict(X_test), axis=1)

print('Results for Categorical Model')
print(accuracy_score(y_test, categorical_pred))
print(classification_report(y_test, categorical_pred))
```

```
Results for Categorical Model
0.7868852459016393
              precision    recall  f1-score   support

           0       0.83      0.68      0.75        28
           1       0.76      0.88      0.82        33

    accuracy                           0.79        61
   macro avg       0.79      0.78      0.78        61
weighted avg       0.79      0.79      0.78        61
```

Fig20: Metrics of ANN algorithm for predicting heart disease

```
[27]: # generate classification report using predictions for binary model
from sklearn.metrics import classification_report, accuracy_score
# generate classification report using predictions for binary model
binary_pred = np.round(binary_model.predict(X_test)).astype(int)

print('Results for Binary Model')
print(accuracy_score(Y_test_binary, binary_pred))
print(classification_report(Y_test_binary, binary_pred))
```

```
Results for Binary Model
0.819672131147541
              precision    recall  f1-score   support

           0       0.90      0.68      0.78        28
           1       0.78      0.94      0.85        33

    accuracy                           0.82        61
   macro avg       0.84      0.81      0.81        61
weighted avg       0.83      0.82      0.82        61
```

Fig21: Metrics of binary classification algorithm for predicting heart disease

8. REFERENCES

[1] Reddi, Sivaranjani & srinivasa naresh, Vankamamidi & Murthy, Nistala V.E.S.. (2019). Coronary Heart Disease prediction using genetic algorithm based decision tree. 10.1515/9783110621105-004.

[2] Gadekallu, Thippa & Reddy, Praveen & Lakshman, Kuruva & Rajput, Dharmendra & Kaluri, Rajesh & Srivastava, Gautam. (2020). Hybrid genetic algorithm and a fuzzy logic classifier for heart disease diagnosis. Evolutionary Intelligence. 13. 10.1007/s12065-019-00327-1.

[3] Bano, Shaikh. (2019). Heart Disease Prediction System using Genetic Algorithm.International Journal for Research in Applied Science and Engineering Technology. 7.2178-2182. 10.22214/ijraset.2019.6366.

[4] Kumkum Chaudhary, Radhika Naidu, Rhea Rai, Narendra Gawai. (2019). Hybrid Architecture of Heart Disease Prediction System using Genetic Neural Network.
V6/i5/IRJET-V6I5857

[5] Uyar, Kaan & Ilhan, Ahmet. (2017). Diagnosis of heart disease using genetic algorithm based trained recurrent fuzzy neural networks. Procedia Computer Science.120. 588-593. 10.1016/j.procs.2017.11.283.

[6] Awan, Shahid & Riaz, Muhammad & Khan, Abdul. (2018). PREDICTION OF HEART DISEASE USING ARTIFICIAL NEURAL NETWORK. 13. 102-112.

[7] Prasadgouda B Patil, Dr. P Mallikarjun Shastry, Dr Ashokumar P S. (2020). A Novel Approach for Prediction of Cardio Vascular Disease: An Improved Genetic Algorithm Approach Using Classifiers. International Journal of Advanced Science andTechnology, 29(7s), 4493 – 4504.

[8] Srikanth Meda1 , Raveendra Babu Bhogapathi 2 1 Research scholar, Acharya Nagarjuna University, Guntur & Associate Professor in the Department of Computer Science and Engineering at R.V.R. & J.C. College of Engineering, Guntur 522019, India 2 Professor, Department of Computer Science and Engineering, R.V.R & J.C College of Engineering, Guntur 552019, India

[9] Akruti Dave, Prof. Gayatri Pandi , Master of Engineering Student, Head of Department Department of Computer Engineering ,L. J Institute of Engineering&Technology (Gujarat Technological University), Ahmedabad, India

[10] Kasbe, Tanmay & Pippal, Ravi. (2017). Design of heart disease diagnosis systemusing fuzzy logic. 3183-3187. 10.1109/ICECDS.2017.8390044.

[11] Zeinab Arabasadi , Roohallah Alizadehsani , Mohamad Roshanzamir , Hossein Moosaei , Ali Asghar Yarifard , Computer aided decision making for heart disease detection using hybrid neural network-Genetic algorithm , Computer Methods and Programs in Bio medicine , Volume 141,2017,Pages 19-26,ISSN 0169-2607.

[12] Zabeen, Ashiya & Utsav, Ankur & Lal, Kanhaiya. (2018). Detection of Heart Disease Applying Fuzzy Logics and Its Comparison with Neural Networks. 461-467.10.1109/RTEICT42901.2018.9012315.

[13] Kumar, S., & Sahoo, G. (2018). Enhanced decision tree algorithm using genetic algorithm for heart disease prediction. International Journal of Bioinformatics Research and Applications, 14(1-2), 49-69.

[14] Nikam, S., Shukla, P., & Shah, M. (2017). Cardiovascular disease prediction using genetic algorithm and neuro-fuzzy system.

[15] http://www.ijareeie.com/upload/2016/rapideet/20_PP_RE1145_NEW.pdf

[16] Kumar, P. S., Anand, D., Kumar, V. U., Bhattacharyya, D., & Kim, T. H. (2016).A computational intelligence method for effective diagnosis of heart disease using genetic algorithm. International Journal of Bio-Science and Bio-Technology, 8(2), 363-372.

[17] Salem, T. (2018). Study and analysis of prediction model for heart disease: an optimization approach using genetic algorithm. International Journal of Pure and Applied Mathematics, 119(16), 5323-5336.

[18] Santhanam, T., & Ephzibah, E. P. (2015). Heart disease prediction using hybrid genetic fuzzy model. Indian Journal of Science and Technology, 8(9), 797.