

## **1. INTRODUCTION**

### **1.1 OBJECTIVE**

The interplay between optimization and machine learning is one of the most important developments in computational science. Optimization formulations and methods are proving to be vital in designing algorithms to extract essential knowledge from huge volumes of data. Machine learning, however, is not simply a consumer of optimization technology but a rapidly evolving field that is itself generating new optimization ideas. This book captures the state of the art of the interaction between optimization and machine learning in a way that is accessible to researchers in both fields.

Optimization approaches have enjoyed prominence in machine learning because of their wide applicability and attractive theoretical properties. The increasing complexity, size, and variety of today's machine learning models call for the reassessment of existing assumptions. It describes the resurgence in novel contexts of established frameworks such as first-order methods, stochastic approximations, convex relaxations, interior-point methods, and proximal methods. It also devotes attention to newer themes such as regularized optimization, robust optimization, gradient and subgradient methods, splitting techniques, and second-order methods. Many of these techniques draw inspiration from other fields, including operations research, theoretical computer science, and subfields of optimization.

### **1.2 PROBLEM IN EXISTING SYSTEM**

In the existing system different classification algorithms, such as Neural Networks have been used to tell whether the cancer cell is benign or malignant but the accuracy obtained is not satisfactory.

### **1.3 SOLUTION**

So, to overcome those errors and get the result with better accuracy we use genetic algorithms and compare them with classification algorithms for optimization and improve accuracy. We will use genetic algorithms instead of backpropagation to find the optimal initialization weights.

## 1.4 FEATURES

The main feature of this project is improving the accuracy using genetic algorithms. As we are using cancer prediction as an example, here the accuracy is calculated for prediction of cancer using both Neural Networks and Genetic Algorithms and both results are compared.

Genetic Algorithms are very efficient to use for optimization and has many advantages:-

- Does not require any derivative information.
- Is faster and more efficient as compared to the traditional methods.
- Has very good parallel capabilities.
- Optimizes both continuous and discrete functions and also multi-objective problems.
- Provides a list of “good” solutions and not just a single solution.
- Always gets an answer to the problem, which gets better over the time.
- Useful when the search space is very large and there are a large number of parameters involved.

## **2. LITERATURE SURVEY**

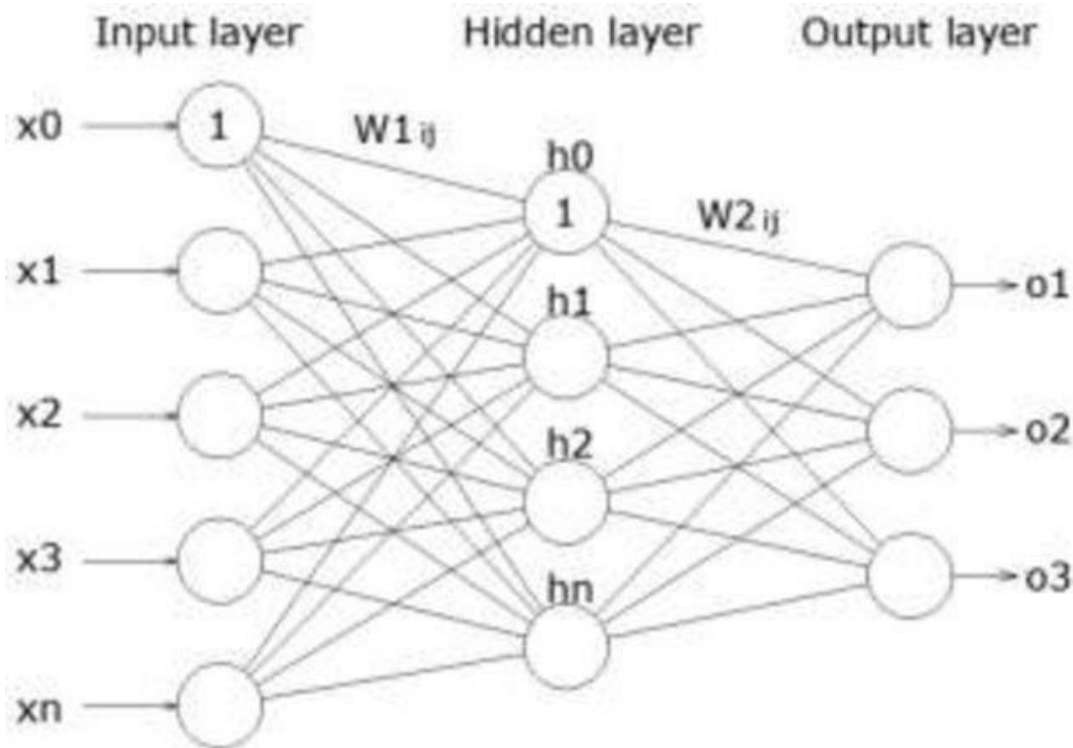
Optimization approaches have enjoyed prominence in machine learning because of their wide applicability and attractive theoretical properties. The increasing complexity, size, and variety of today's machine learning models call for the reassessment of existing assumptions. It describes the resurgence in novel contexts of established frameworks such as first-order methods, stochastic approximations, convex relaxations, interior-point methods, and proximal methods. It also devotes attention to newer themes such as regularized optimization, robust optimization, gradient and sub gradient methods, splitting techniques, and second-order methods. Many of these techniques draw inspiration from other fields, including operations research, theoretical computer science, and subfields of optimization.

In the survey done we learnt about the methods to be used for optimization. An IEEE paper reference has been taken published in 2018 and the process includes the following algorithms. The are:-

### **2.1 NEURAL NETWORK**

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems. A neural network works similarly to the human brain's neural network. A "neuron" in a neural network is a mathematical function that collects and classifies information according to a specific architecture. A neural network

contains layers of interconnected nodes. Each node is a perceptron and is similar to a multiple linear regression. The perceptron feeds the signal produced by a multiple linear regression into an activation function that may be nonlinear. Hidden layers fine-tune the input weightings until the neural network's margin of error is minimal. It is hypothesized that hidden layers extrapolate salient features in the input data that have predictive power regarding the outputs. This describes feature extraction, which accomplishes a utility similar to statistical techniques such as principal component analysis.



**Figure2.1.0 Architechture Of Neural Networks**

## 2.2 GENETIC ALGORITHMS

A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

Genetic algorithms are a subset of the evolutionary algorithm set. The design of evolutionary algorithms is based on processes which occur in living organisms in nature, for example, inheritance, mutation, selection and crossover. In today's world, optimal and intelligent problem

solving approaches are required in every field, regardless of simple or complex problems. Genetic Algorithm is developed to find the most optimized solution for a given problem. Genetic Algorithm consist of five different phases.

- Individual in population compete for resources and mate
- Those individuals who are successful (fittest) then mate to create more offspring than others
- Genes from “fittest” parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent.
- Thus each successive generation is more suited for their environment.

### **2.3 BACKPROPAGATION**

The Backpropagation algorithm is a supervised learning method for multilayer feed-forward networks from the field of Artificial Neural Networks. Feed-forward neural networks are inspired by the information processing of one or more neural cells, called a neuron. A neuron accepts input signals via its dendrites, which pass the electrical signal down to the cell body. The axon carries the signal out to synapses, which are the connections of a cell’s axon to other cell’s dendrites. The principle of the backpropagation approach is to model a given function by modifying internal weightings of input signals to produce an expected output signal. The system is trained using a supervised learning method, where the error between the system’s output and a known expected output is presented to the system and used to modify its internal state.

Technically, the backpropagation algorithm is a method for training the weights in a multilayer feed-forward neural network. As such, it requires a network structure to be defined of one or more layers where one layer is fully connected to the next layer. A standard network structure is one input layer, one hidden layer, and one output layer. It can be used for both classification and regression problems. For classification, backpropagation can be divided into 5 basic steps.

- Initialize Network.

- Forward Propagate.
- Back Propagate Error.
- Train Network.
- Predict.

### **3. REQUIREMENT SPECIFICATIONS**

Software requirements deal with software and hardware deals with resources that need to be installed on a server which provides optimal functioning for the application. These software and hardware requirements need to be installed before the packages are installed. These are the most common set of requirements defined by any operation system. These software and hardware requirements provide a compatible support to the operation system in developing an application.

#### **3.1 SOFTWARE REQUIREMENTS**

The software requirements specify the use of all required software products like data management system. The required software product specifies the numbers and version. Each interface specifies the purpose of the interfacing software as related to this software product.

- Operating system : Windows 7/10
- Coding Language : Python 3,Java
- IDE : Jupyter Notebook, Eclipse

#### **3.2 HARDWARE REQUIREMENTS**

The hardware requirement specifies each interface of the software elements and the hardware elements of the system. These hardware requirements include configuration

characteristics.

- System : Pentium IV 2.4 GHz.
- Hard Disk : 100 GB.
- Monitor : 15 VGA Color.
- RAM : 4 GB.

## 4. DESIGN OF THE SYSTEM

The system is designed to obtain maximum accuracy using genetic algorithms. For that the understanding of genetic algorithms and as it best suits for neural networks, understanding neural networks is also essential. The first step to any machine learning project using dataset is data pre-processing.

### 4.1 PRE-PROCESSING

Data preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format. A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

**Get the Dataset:-** To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the **dataset**.

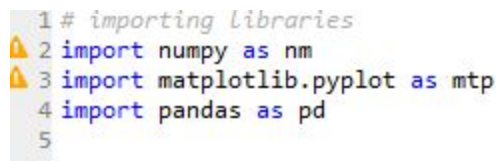
**Importing Libraries:-** In order to perform data preprocessing using Python,

we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:

**Numpy:** Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:

**Matplotlib:** The second library is matplotlib, which is a Python 2D plotting library, and with this library, we need to import a sub-library pyplot. This library is used to plot any type of charts in Python for the code. It will be imported as below:

**Pandas:** The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library.



```
1 # importing libraries
2 import numpy as nm
3 import matplotlib.pyplot as mtp
4 import pandas as pd
5
```

**Figure 4.1.0 Importing libraries**

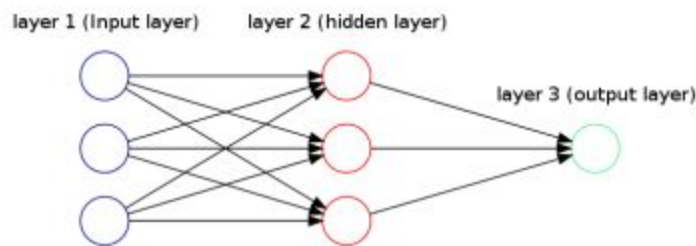
**Technique applied:-** Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. For machine learning, every dataset does not require normalization. It is required only when features have different ranges.

## 4.2 NEURAL NETWORKS

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain

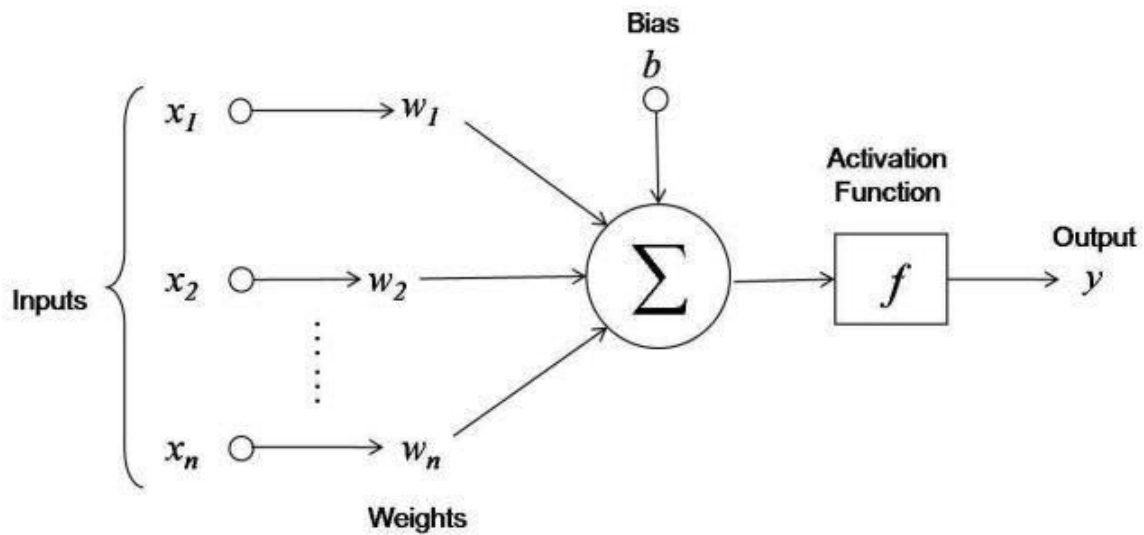


operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems. A neural network works similarly to the human brain's neural network. This describes feature extraction, which accomplishes a utility similar to statistical techniques such as principal component analysis.



**Figure 4.2.0 Simple Structure Of A Neural Network**

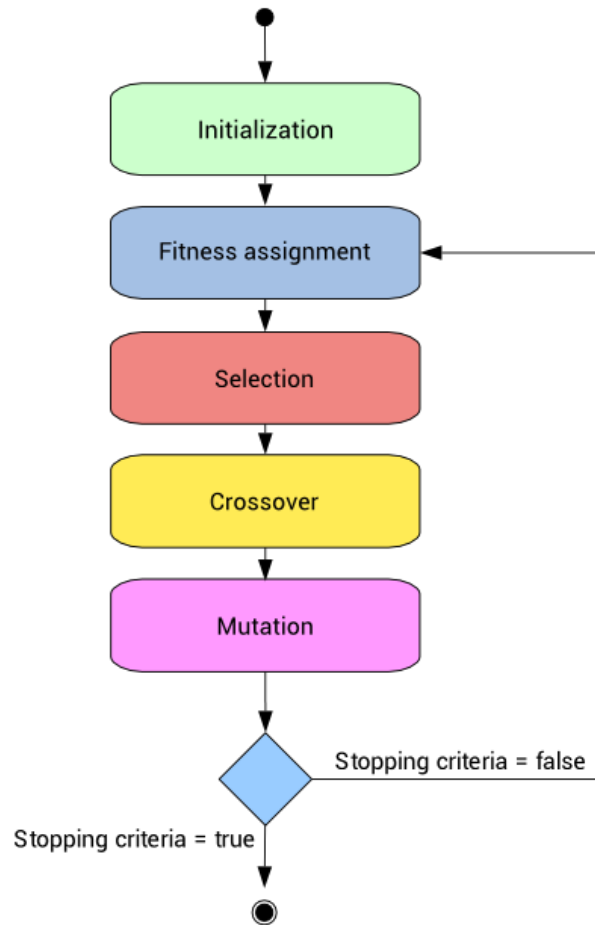
A “neuron” in a neural network is a mathematical function that collects and classifies information according to a specific architecture. A neural network contains layers of interconnected nodes. Each node is a perceptron and is similar to a multiple linear regression. The perceptron feeds the signal produced by a multiple linear regression into an activation function that may be nonlinear. Hidden layers fine-tune the input weightings until the neural network’s margin of error is minimal. It is hypothesized that hidden layers extrapolate salient features in the input data that have predictive power regarding the outputs.



**Figure 4.2.1 Function Of A Neuron**

### 4.3 GENETIC ALGORITHMS

Genetic algorithms are a subset of the evolutionary algorithm set. The design of evolutionary algorithms is based on processes which occur in living organisms in nature, for example, inheritance, mutation, selection and crossover. In today's world, optimal and intelligent problem solving approaches are required in every field, regardless of simple or complex problems. Genetic Algorithm is developed to find the most optimized solution for a given problem. Genetic Algorithm consist of five different phases. They are



**FIGURE 4.3.0 Phases Of Genetic Algorithm**

**Initial Population :-** The process begins with a set of individuals which is called a Population. Each individual is a solution to the problem you want to solve. An individual is characterized by a set of parameters (variables) known as Genes. Genes are joined into a string to form a Chromosome (solution).

In a genetic algorithm, the set of genes of an individual is represented using a string, in terms of an alphabet. Usually, binary values are used (string of 1s and 0s). We say that we encode the genes in a chromosome.

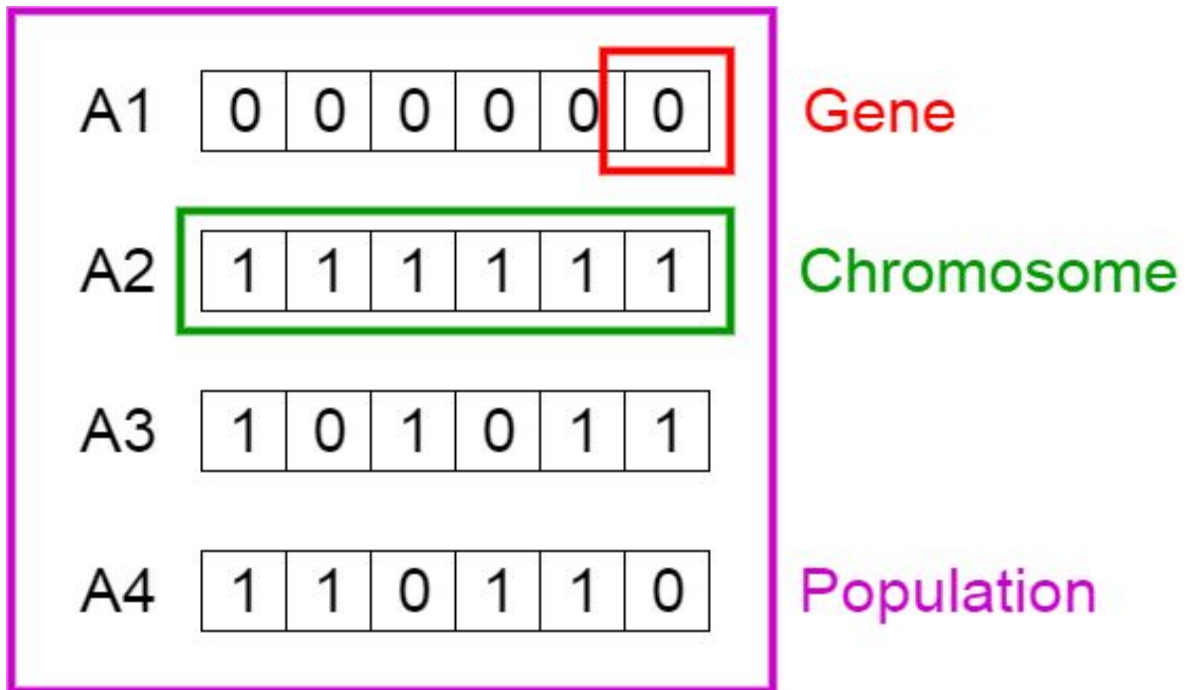
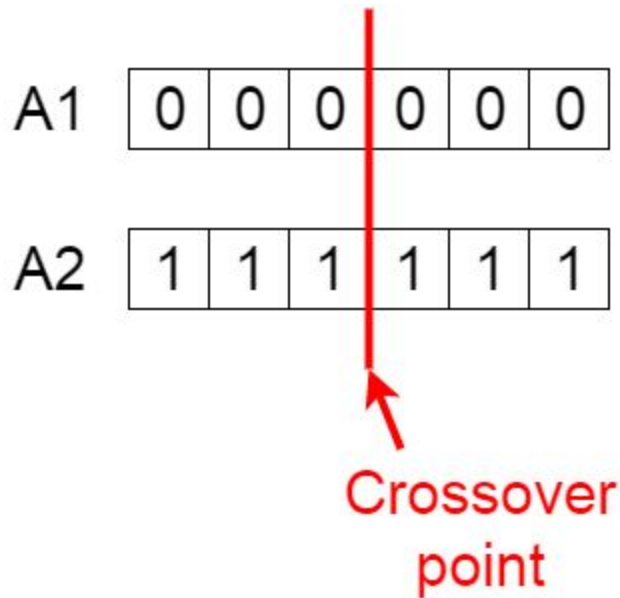


Figure 4.3.1 Initialize Population

**Fitness Function :-** The fitness function determines how fit an individual is the ability of an individual to compete with other individuals. It gives a fitness score to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.

**Selection :-** The idea of selection phase is to select the fittest individuals and let them pass their genes to the next generation. Two pairs of individuals (parents) are selected based on their fitness scores. Individuals with high fitness have more chance to be selected for reproduction.

**Crossover :-** Crossover is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a crossover point is chosen at random from within the genes. Offspring are created by exchanging the genes of parents among themselves until the crossover point is reached.



**Figure 4.3.2 Crossover Point**

**Mutation :-** In certain new offspring formed, some of their genes can be subjected to a mutation with a low random probability. This implies that some of the bits in the bit string can be flipped. Mutation occurs to maintain diversity within the population and prevent premature convergence.

### Before Mutation

A5    

1	1	1	0	0	0
---	---	---	---	---	---

### After Mutation

A5    

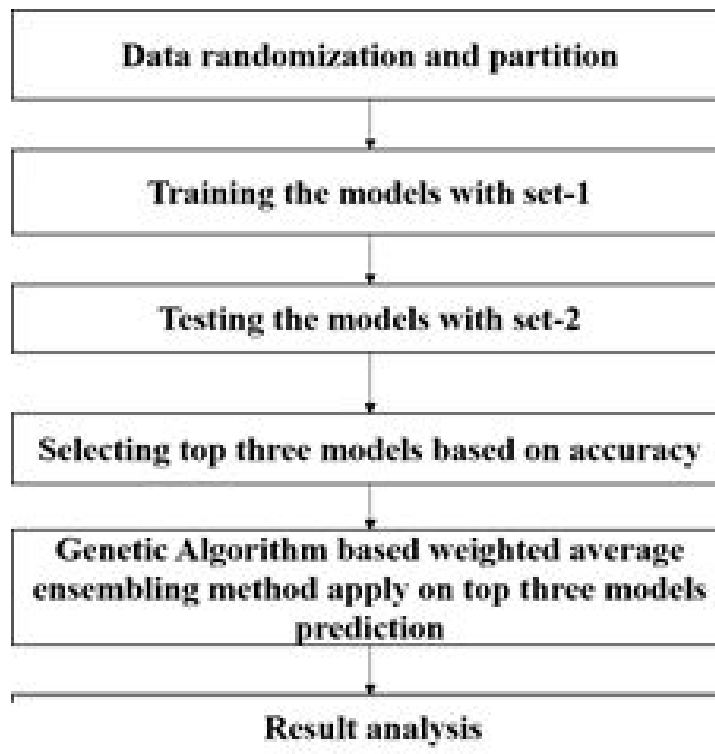
1	1	0	1	1	0
---	---	---	---	---	---

**Figure 4.3.3 Mutation**

**Termination :-** The algorithm terminates if the population has converged (does not produce offspring which are significantly different from the previous generation). Then it is said that the genetic algorithm has provided a set of solutions to our problem.

## 5. MODULES

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 design of the system is completed in six different stages that are explained below.



**Figure 5.0 Architecture Of The Model**

### 5.1 Data randomization and partition

Due to the vast and rapid increase in the size of data, machine learning has become an increasingly more popular approach for the purpose of knowledge discovery and predictive modelling. For both of the above purposes, it is essential to have a data set partitioned into a training set and a test set. In particular, the training set is used towards learning a model

and the test set is then used towards evaluating the performance of the model learned from the training set. The split of the data into the two sets, however, and the influence on model performance, has only been investigated with respect to the optimal proportion for the two sets, with no attention paid to the characteristics of the data within the training and test sets. Thus, the current practice is to randomly split the data into approximately 70% for training and 30% for testing.

Randomization is the process of making something random; in various contexts this involves, generating a random permutation of a sequence such as when shuffling cards)selecting random sample of a population important in statistical sampling allocating experimental units via random assignment to a treatment or control condition generating random numbers see Random number generation or transforming a data stream such as when using a scrambler in telecommunications. Randomization is not haphazard. Instead, a random process is a sequence of random variables describing a process whose outcomes do not follow a deterministic pattern, but follow an evolution described by probability distributions. For example, a random sample of individuals from a population refers to a sample where every individual has a known probability of being sampled. This would be contrasted with nonprobability sampling where arbitrary individuals are selected.

## **5.2 Training the models with set-1**

After partitioning of data the first 70% of the models that are present in ser-1 are trained in this phase. the training set is used towards learning a model

## **5.3 Testing the models with set-2**

After training then the remaining 30% of the models that are present in set-2 are tested and the test set is then used towards evaluating the performance of the model learned from the training set

## **5.4 Selection top three models based on accuracy**

In the second phase by using different models of classification on the same dataset we calculate accuracy. Best models selection is based on accuracy estimation. In this module the best fit model

is selected and then it is trained. After completion of training the trained model is then tested to check whether it is best fit or not.

### **5.5 Genetic Algorithm based weight average ensembling method apply on top three model predictions**

In the final phase, we have to calculate weight by an evolutionary algorithm named as genetic algorithm (GA) and optimize weight by GA applied to a weighted average method of an ensemble. We have taken three nature inspired algorithm (NIA) for the weight optimization out of which GA outperforms because it gives the best fitness function, best chromosome and function evolution is also less in comparison to both algorithms and time taken in weight optimization is also less so, it is used in weighted average ensemble method. This process is used to increase the efficiency of the proposed model.

### **5.6 Result Analysis**

The final result is analyzed after getting the optimized weights and the accuracies of both results are compared.



## 6. IMPLEMENTATION

### 6.1 PREPROCESSING

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import itertools
import timeit
plt.style.use('fivethirtyeight')
%matplotlib inline
cancer=pd.read_csv('breast-cancer-w-diag.csv', header=0)
cancer.isnull().sum()
sns.countplot(x='diagnosis',data=cancer)
plt.show()
cancer.dtypes
cancer.head()
```

```
cancer.drop('id',axis=1,inplace=True)
cancer.drop('Unnamed: 32',axis=1,inplace=True)
# size of the dataframe
len(cancer)
cancer.diagnosis.unique()
cancer['diagnosis'] = cancer['diagnosis'].map({'M':1,'B':0})
cancer.head()
cancer.head(2)
columns=cancer.columns[1:31]
plt.subplots(figsize=(18,15))
length=len(columns)
for i,j in itertools.izip_longest(columns,range(length)):
    plt.subplot((length/2),3,j+1)
    plt.subplots_adjust(wspace=0.2,hspace=0.5)
    cancer[i].hist(bins=20,edgecolor='black')
    plt.title(i)
plt.show()
features_mean=list(cancer.columns[1:11])
# split dataframe into two based on diagnosis
dfM=cancer[cancer['diagnosis']==1]
dfB=cancer[cancer['diagnosis']==0]

#Stack the data
plt.rcParams.update({'font.size': 8})
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(8,10))
axes = axes.ravel()
for idx,ax in enumerate(axes):
    ax.figure
    binwidth= (max(cancer[features_mean[idx]]) - min(cancer[features_mean[idx]]))/50
```

```
ax.hist([dfM[features_mean[idx]],dfB[features_mean[idx]]],
bins=np.arange(min(cancer[features_mean[idx]]), max(cancer[features_mean[idx]]) +
binwidth, binwidth) , alpha=0.5,stacked=True, normed = True,
label=['M','B'],color=['r','g'])
ax.legend(loc='upper right')
ax.set_title(features_mean[idx])
plt.tight_layout()
plt.show()

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.cross_validation import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.learning_curve import validation_curve
from sklearn.learning_curve import learning_curve
from time import time
import warnings
warnings.filterwarnings('ignore')

# set the following figure sizes
plt.rcParams.update({'font.size': 10})
plt.rcParams.update({'figure.figsize': (4,3)})
#outcome=cancer['LeagueIndex']
#data=cancer[cancer.columns[:18]]
from sklearn.model_selection import train_test_split
#train,test=train_test_split(cancer,test_size=0.2,random_state=0,stratify=cancer['diagnosis'
])# stratify the outcome
train,test=train_test_split(cancer,test_size=0.2,random_state=0,stratify=cancer['diagnosis'])
# stratify the outcome
```

```
train_X=train[train.columns[1:31]]
test_X=test[test.columns[1:31]]
train_Y=train['diagnosis']
test_Y=test['diagnosis']
from sklearn.preprocessing import StandardScaler

# fit a standardScaler to normalize all input to zero mean and unit variance
scaler = StandardScaler().fit(train_X)
train_X = pd.DataFrame(scaler.transform(train_X), columns = train_X.columns)
test_X = pd.DataFrame(scaler.transform(test_X), columns = test_X.columns)
type(train_X)
type(train_X)
test_X.head(5)
test_X.head(5)
```

## **6.2 APPLYING NEURAL NETWORK**

```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier()
clf.fit(train_X, train_Y)
train_acc = clf.score(train_X, train_Y)
test_acc = clf.score(test_X, test_Y)
print "Train acc:", train_acc
print "Test acc:", test_acc
```

```
from sklearn.model_selection import validation_curve
degree1 = [(10,),(10,10),(10,10,10)]
degree2 = [(20,),(20,20),(20,20,20)]
degree3 = [(50,),(50,50),(50,50,50)]
degree = np.concatenate([degree1,degree2,degree3])
train_scores, val_scores = validation_curve(MLPClassifier(), train_X, train_Y,
```

```
'hidden_layer_sizes', degree, cv=3)

print(degree)

np.mean(train_scores,1)

np.mean(val_scores,1)

degree = [(2,),(5,),(10,),(20,),(50,),(100,),(200,)]
train_scores, val_scores = validation_curve(MLPClassifier(), train_X, train_Y,
                                             'hidden_layer_sizes', degree, cv=10, n_jobs=3)

plt.plot(degree, np.mean(train_scores, 1), 'o-',color='blue', label='training')
plt.plot(degree, np.mean(val_scores, 1), 'o-', color='red', label='cross validation')

plt.legend(loc='lower right')
plt.ylim(0.85, 1.01)
plt.title("MLP: varying hidden layer nodes \n1 hidden layer (Cancer)")
plt.xlabel('# of hidden layer nodes')
plt.ylabel('Accuracy');
plt.show()

degree = [(20,),(50,),(100,),(50,50),(50,50,50)]

mlps=[]
train_times=[]
test_times=[]
train_accs=[]
test_accs=[]
```

```
for i in range(len(degree)):
    # train
    mlp = MLPClassifier(max_iter=10000,
early_stopping=True,hidden_layer_sizes=degree[i])
    start = timeit.default_timer()
    mlp.fit(train_X, train_Y)
    stop = timeit.default_timer()
    train_time = stop - start
    # train result
    train_Y_predict = clf.predict(train_X)
    train_acc = accuracy_score(train_Y, train_Y_predict)
    # test
    start = timeit.default_timer()
    test_Y_predict = clf.predict(test_X)
    stop = timeit.default_timer()
    test_time = stop - start
    test_acc = accuracy_score(test_Y, test_Y_predict)
    # save results
    mlps.append(mlp)
    train_times.append(train_time)
    test_times.append(test_time)
    train_accs.append(train_acc)
    test_accs.append(test_acc)
    print degree[i], train_acc,test_acc,train_time,test_time

print "degree|  train acc|  test acc|  train time|  test time"
for i in range(len(degree)):
    print degree[i], train_accs[i],test_accs[i],train_times[i],test_times[i]
```

```
params = [{'solver': 'sgd', 'learning_rate': 'constant', 'momentum': 0,
          'learning_rate_init': 0.001},
          {'solver': 'sgd', 'learning_rate': 'constant', 'momentum': .9,
          'learning_rate_init': 0.001},
          {'solver': 'sgd', 'learning_rate': 'constant', 'momentum': 0.5,
          'learning_rate_init': 0.001},
          {'solver': 'sgd', 'learning_rate': 'constant', 'momentum': 0.5,
          'learning_rate_init': 0.01},
          {'solver': 'sgd', 'learning_rate': 'constant', 'momentum': 0.5,
          'learning_rate_init': 0.1},
          ]

mlps=[]

for param in params:
    mlp = MLPClassifier(hidden_layer_sizes=(22,), verbose=0, max_iter=10000, **param)
    mlp.fit(train_X, train_Y)
    mlps.append(mlp)

labels = ["L=0.001, M=0",
          "L=0.001, M=0.9",
          "L=0.001, M=0.5",
          "L=0.01, M=0.5",
          "L=0.1, M=0.5",
          ]

plot_args = [{'c': 'red'},
              {'c': 'purple'},
              {'c': 'blue'},
```

```
{'c': 'green'},
{'c': 'black'}]

for mlp, label, args in zip(mlps, labels, plot_args):
    plt.plot(mlp.loss_curve_, label=label, **args)

plt.legend(loc='upper right')
plt.ylim(-0.1, 0.9)
plt.title("MLP: varying learning_rate (L)\nand momentum (M)\n1 hidden layer, 22 nodes\n(Cancer)")
plt.xlabel('# of epochs')
plt.ylabel('Training Loss');
plt.show()

mlp_temp_model =
MLPClassifier(hidden_layer_sizes=(22,), solver='sgd', learning_rate_init=0.9,
              momentum=0.8, max_iter=10000)

start = timeit.default_timer()
clf = clf.fit(train_X, train_Y)
stop = timeit.default_timer()
train_time = stop - start

train_Y_predict = clf.predict(train_X)
train_acc = accuracy_score(train_Y, train_Y_predict)

start = timeit.default_timer()
test_Y_predict = clf.predict(test_X)
stop = timeit.default_timer()
```



```
test_time = stop - start  
test_acc = accuracy_score(test_Y, test_Y_predict)
```

```
# print result  
print 'test acc: ', test_acc  
print 'train time: ', train_time  
print 'test time: ', test_time
```

### 6.3 APPLYING GENETIC ALGORITHM

```
import dist.*;  
import opt.*;  
import opt.example.*;  
import opt.ga.*;  
import shared.*;  
import func.nn.backprop.*;
```

```
import java.util.*;  
import java.io.*;  
import java.text.*;
```

```
public class GA4BC {  
    private static Instance[] trainSet = initializeInstances("data/btrain.csv", 455);  
    private static Instance[] testSet = initializeInstances("data/btest.csv", 114);  
  
    private static int inputLayer = 30, hiddenLayer = 22, outputLayer = 1,  
trainingIterations = 100;  
    private static BackPropagationNetworkFactory factory = new  
BackPropagationNetworkFactory();
```

```
private static ErrorMeasure measure = new SumOfSquaresError();

private static DataSet set = new DataSet(trainSet);

private static BackPropagationNetwork network = new BackPropagationNetwork();
private static NeuralNetworkOptimizationProblem nnop;

private static OptimizationAlgorithm oa;
private static String oaName = "GA";
private static String results = "";

private static DecimalFormat df = new DecimalFormat("0.0000");

public static void main(String[] args) {
    int repeat = 10;
    double trainingAcc = 0, testingAcc = 0, trainingTime = 0, testingTime = 0;
    for (int i = 0; i < repeat; i++) {
        List<Double> result = oneRun();
        trainingAcc += result.get(0);
        testingAcc += result.get(1);
        trainingTime += result.get(2);
        testingTime += result.get(3);
    }
    trainingAcc /= repeat;
    testingAcc /= repeat;
    trainingTime /= repeat;
    testingTime /= repeat;
}
```

```
System.out.println();

System.out.println("trainingIterations,trainingAcc,testingAcc,trainingTime,testingTime");
    System.out.println(trainingIterations + "," + df.format(trainingAcc) + "," +
df.format(testingAcc) + ","
        + df.format(trainingTime) + "," + df.format(testingTime));
}

private static List<Double> oneRun() {
    List<Double> result = new ArrayList<>();
    network = factory.createClassificationNetwork(new int[] { inputLayer,
hiddenLayer, outputLayer });
    nnop = new NeuralNetworkOptimizationProblem(set, network, measure);

    // change parameters here
    oa = new StandardGeneticAlgorithm(200, 100, 30, nnop);

    double start = System.nanoTime(), end, trainingTime, testingTime, correct =
0, incorrect = 0;
    train(oa, network, oaName); // trainer.train();
    end = System.nanoTime();
    trainingTime = end - start;
    trainingTime /= Math.pow(10, 9);

    Instance optimalInstance = oa.getOptimal();
    network.setWeights(optimalInstance.getData());

    // Calculate Training Set statistics
    double predicted, actual;
```

```
for (int j = 0; j < trainSet.length; j++) {
    network.setInputValues(trainSet[j].getData());
    network.run();

    predicted = Double.parseDouble(trainSet[j].getLabel().toString());
    actual =
Double.parseDouble(network.getOutputValues().toString());

    double trash = Math.abs(predicted - actual) < 0.5 ? correct++ :
incorrect++;

}
double trainingAcc = correct / (correct + incorrect);

// Calculate Test Set Statistics //
start = System.nanoTime();
correct = 0;
incorrect = 0;
for (int j = 0; j < testSet.length; j++) {
    network.setInputValues(testSet[j].getData());
    network.run();

    predicted = Double.parseDouble(testSet[j].getLabel().toString());
    actual =
Double.parseDouble(network.getOutputValues().toString());

    double trash = Math.abs(predicted - actual) < 0.5 ? correct++ :
incorrect++;

}
```

```
double testingAcc = correct / (correct + incorrect);

end = System.nanoTime();
testingTime = end - start;
testingTime /= Math.pow(10, 9);

// add result to the list
result.add(trainingAcc);
result.add(testingAcc);
result.add(trainingTime);
result.add(testingTime);

return result;

}

private static void train(OptimizationAlgorithm oa, BackPropagationNetwork
network, String oaName) {
    //System.out.println("\nError results for " + oaName +
    "\n-----");
    //System.out.println("train mse, test mse");

    for (int i = 0; i < trainingIterations; i++) {
        oa.train();

        double train_error = 0;
        for (int j = 0; j < trainSet.length; j++) {
            network.setInputValues(trainSet[j].getData());
```

```
        network.run();

        Instance output = trainSet[j].getLabel(), example = new
Instance(network.getOutputValues());
        example.setLabel(new
Instance(Double.parseDouble(network.getOutputValues().toString())));
        train_error += measure.value(output, example);
    }

    double test_error = 0;
    for (int j = 0; j < testSet.length; j++) {
        network.setInputValues(testSet[j].getData());
        network.run();

        Instance output = testSet[j].getLabel(), example = new
Instance(network.getOutputValues());
        example.setLabel(new
Instance(Double.parseDouble(network.getOutputValues().toString())));
        test_error += measure.value(output, example);
    }

    //System.out.println(df.format(train_error / trainSet.length) + "," +
df.format(test_error / testSet.length));
    }
}
```

```
private static Instance[] initializeInstances(String filename, int size) {
```

```
double[][][] attributes = new double[size][][];

try {
    BufferedReader br = new BufferedReader(new FileReader(new
File(filename)));

    for (int i = 0; i < attributes.length; i++) {
        Scanner scan = new Scanner(br.readLine());
        scan.useDelimiter(",");

        attributes[i] = new double[2][];
        attributes[i][0] = new double[30]; // 30 attributes
        attributes[i][1] = new double[1];

        for (int j = 0; j < 30; j++)
            attributes[i][0][j] = Double.parseDouble(scan.next());

        attributes[i][1][0] = Double.parseDouble(scan.next());
    }
} catch (Exception e) {
    e.printStackTrace();
}

Instance[] instances = new Instance[attributes.length];

for (int i = 0; i < instances.length; i++) {
    instances[i] = new Instance(attributes[i][0]);
    instances[i].setLabel(new Instance(attributes[i][1][0]));
}
```

```

    return instances;
}

```

## 7. TESTING

### 7.1 INPUT

The input given is the dataset containing 12 columns containing cancer cell features. The dataset has been taken from UCI repository. It has around 3745 records.

	A	B	C	D	E	F	G	H	I	J	K	L
1	id	diagnosis	radius_m	texture_n	perimeter	area_mea	smoothne	compactn	concavity	concave p	symmetry	fractal_di
2	842302	M	17.99	10.38	122.8	1001	0.1184	0.2776	0.3001	0.1471	0.2419	0.07871
3	842517	M	20.57	17.77	132.9	1326	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667
4	84300903	M	19.69	21.25	130	1203	0.1096	0.1599	0.1974	0.1279	0.2069	0.05999
5	84348301	M	11.42	20.38	77.58	386.1	0.1425	0.2839	0.2414	0.1052	0.2597	0.09744
6	84358402	M	20.29	14.34	135.1	1297	0.1003	0.1328	0.198	0.1043	0.1809	0.05883
7	843786	M	12.45	15.7	82.57	477.1	0.1278	0.17	0.1578	0.08089	0.2087	0.07613
8	844359	M	18.25	19.98	119.6	1040	0.09463	0.109	0.1127	0.074	0.1794	0.05742
9	84458202	M	13.71	20.83	90.2	577.9	0.1189	0.1645	0.09366	0.05985	0.2196	0.07451
10	844981	M	13	21.82	87.5	519.8	0.1273	0.1932	0.1859	0.09353	0.235	0.07389
11	84501001	M	12.46	24.04	83.97	475.9	0.1186	0.2396	0.2273	0.08543	0.203	0.08243
12	845636	M	16.02	23.24	102.7	797.8	0.08206	0.06669	0.03299	0.03323	0.1528	0.05697

Figure 7.1.0 Dataset

### 7.2 OUTPUT

When it comes to testing for neural networks testing is done through changing the weights of the nodes and the lambda and momentum values. The highest accuracy is obtained for lambda=0.8 and momentum = 0.9.

```

train acc: 0.997802197802
test acc: 0.938596491228
train time: 2.18189205849
test time: 0.00120024898524

```

Figure 7.2.0 Accuracy Obtained For Neural Networks

Then the genetic algorithms are applied and the no of generation is 100 then the output obtained is as follows:



```
training time:310325.0  
trainacc:98.740000  
testacc:99.653333
```

**Figure 7.2.1 Accuracy Obtained For Genetic Alorithms**

## **8. CONCLUSION AND FUTURE SCOPE**

### **8.1 CONCLUSION**

In this project, we have compared the accuracies of classification algorithms, such as Neural Networks with the genetic lgorithms for optimization and tell whether the cancer is benign or malignant and the accuracy for genetic algorithm is the best one. In conclusion, Genetic Algorithm (GA) is the best application to solve various common problems using fitness functions. Genetic Algorithm is an exhaustive approach which can be applied in the Artificial intelligence field to find optimal solutions in complex search spaces. It is a heuristic search algorithm that will exploit the historical information for the best solution. Genetic Algorithm works on a state space of potential solutions and selects maximum or optimal solution based on the fitness value of candidate solution. ... Genetic Algorithms avoid the problem of getting stuck at local maxima which is usually faced by Traditional Search techniques.

### **8.2 FUTURE SCOPE**

The future scope of the project is that, to try and implement the other algorithms like SVM and Decision Trees and also develop user interface for the whole model.

## 9. REFERENCES

1. Implementing Genetic Algorithm- <https://ieeexplore.ieee.org/document/8493927>
2. [https://www.researchgate.net/publication/328767222\\_Breast\\_Cancer\\_Prediction\\_Using\\_Gen](https://www.researchgate.net/publication/328767222_Breast_Cancer_Prediction_Using_Gen)
3. For understanding neural networks-  
[https://www.researchgate.net/publication/319284812\\_Classification\\_of\\_breast\\_cancer\\_dataset](https://www.researchgate.net/publication/319284812_Classification_of_breast_cancer_dataset)
4. <https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3>
5. To understand using weka for JAVA API- <https://sourceforge.net/projects/weka/>
6. Randomization-  
[https://www.researchgate.net/publication/228563200\\_Data\\_Randomization](https://www.researchgate.net/publication/228563200_Data_Randomization)