Reading Assignment 1

Kausik N – COE17B010

Q I

I) Understand the working of the following classifier algorithms and trace the same for a sample dataset (min 10 records) which involves 2 classes (Binary Classifier) eg: 'Yes' or 'No', 'True' or 'False'.

a) Decision Tree

Give pseudo code and trace decision tree algorithms.

Understand attribute selection measures such as Information gain, gain ratio (use anyone for the trace).

Pseudo Code

S – Samples, A – Attribute List

1. create a node N
2. If all samples are of the same class C then label N with C; Stop;
3. If A is empty then label N with the most common class C in S (majority voting); Stop;
4. Select a $\in$ A, with the highest information gain; Label N with a;
5. For each value v of a:
    a. Grow a branch from N with condition a=y;
    b. Let be the subset of samples in S with a=y;
    c. If is empty then attach a leaf labelled with the most common class in S;
    d. Else attach the node generated for Samples S, Attribute List A-a

Trace for Information Gain

	A	B	C	D	E
1	4.8	3.4	1.9	0.2	positive
2	5	3	1.6	0.2	positive
3	5	3.4	1.6	0.4	positive
4	5.2	3.5	1.5	0.2	positive
5	5.2	3.4	1.4	0.2	positive
6	4.7	3.2	1.6	0.2	positive
7	4.8	3.1	1.6	0.2	positive
8	5.4	3.4	1.5	0.4	positive
9	7	3.2	4.7	1.4	negative
10	6.4	3.2	4.5	1.5	negative
11	6.9	3.1	4.9	1.5	negative
12	5.5	2.3	4	1.3	negative
13	6.5	2.8	4.6	1.5	negative
14	5.7	2.8	4.5	1.3	negative
15	6.3	3.3	4.7	1.6	negative
16	4.9	2.4	3.3	1	negative

A, B, C, D attributes can be considered as predictors

E column class labels can be considered as a target variable.

For constructing a decision tree from this data, we have to convert continuous data into categorical data.

Categorization is done by,

A: >=5 or <5

B: >=3 or <3

C: >=4.2 or <4.2

D: >=1.4 or <1.4

Entropy used for finding IG is given by,

$$H(X) = \mathbb{E}_X[I(x)] = -\sum_{x \in \mathbb{X}} p(x) \log p(x).$$

Now calculating IG for A,

Var A has value >=5 for 12 records out of 16 and 4 records with value <5 value.

(A >= 5 and class is positive): 5/12

(A >= 5 and class is negative): 7/12

$$Entropy\,(5,7) = \; -1 * ((5/12)\,2\,(5/12) + (7/12)\,2\,(7/12)) = 0.9799$$

(A <5 and class is positive):

(A <5 and class is negative):

$$Entropy\,(3,1) = \; -1 * ((3/4)\,2\,(3/4) + (1/4)\,2\,(1/4)) = 0.81128$$

$Entropy\,(Target, A) = P\,(>= 5) * E\,(5,7) + P\,(< 5) * E\,(3,1) = (12/16) * 0.9799 + (4/16) * 0.81128 = 0.93774$

Information Gain for A = E(Target) – E(Target, A) = 1 – 0.937745 = 0.062255

Similarly,

IG(A) = 0.062255

IG(B) = 0.707095

IG(C) = 0.5488

IG(D) = 0.41189

Using IG values in descending order we can construct DT as,

b) Naive Bayesian Classifier (NBC)

Read about Bayes theorem, conditional class independence, prior and posterior probabilities.

How to handle zero probability scenario (Laplacian Estimator)

Give a short pseudo code and trace.

Pseudo Code

Target Variable – Y (true or false – binary)

Predictor Variables – X1 .. Xn

1) From the data, Estimate P(Y = true) and P(Y = false)
2) To do (1), For every value xij of each input attribute Xi,
    a) Find P(Xi = xij | Y = true)
    b) Find P(Xi = xij | Y = false)

Now, Classify a new X' by,

(Let PROD(true) = { P(X1| Y=true). P(X2| Y=true)... P(Xn| Y=true)})

(Let PROD(false) = { P(X1| Y= false). P(X2| Y= false)... P(Xn| Y= false)})

Y' (Predicted Class for X') =

True if P(Y = true)PROD(true) > P(Y = false)PROD(false)

False if P(Y = true)PROD(true) <= P(Y = false)PROD(false)

Trace

INDEX	OUTLOOK	TEMPERATURE	HUMIDITY	WINDY	
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes
9	Sunny	Mild	Normal	False	Yes
10	Rainy	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Sunny	Mild	High	True	No

We assume that all attributes are independent and are of equal importance to outcome.

Then we find P(Xi=xij| Y=Yes) and P(Xi=xij| Y=No)

Outlook

	Yes	No	P(yes)	P(no)
Sunny	2	3	2/9	3/5
Overcast	4	0	4/9	0/5
Rainy	3	2	3/9	2/5
Total	9	5	100%	100%

Temperature

	Yes	No	P(yes)	P(no)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
Total	9	5	100%	100%

Humidity

	Yes	No	P(yes)	P(no)
High	3	4	3/9	4/5
Normal	6	1	6/9	1/5
Total	9	5	100%	100%

Wind

	Yes	No	P(yes)	P(no)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100%	100%

Play		P(Yes)/P(No)
Yes	9	9/14
No	5	5/14
Total	14	100%

Now that we have this data,

Suppose test data is

Today = (Sunny, Hot, Normal, False)

By Bayes Theorem,

$$P\left(Yes|today\right) = \frac{P\left(Sunny|Yes\right)P\left(Hot|Yes\right)P\left(Normal|Yes\right)P\left(False|Yes\right)P\left(Yes\right)}{P\left(today\right)}$$

$$= \frac{\frac{2}{9} \cdot \frac{2}{9} \cdot \frac{6}{9} \cdot \frac{6}{9} \cdot \frac{9}{14}}{P(today)} = \frac{0.0141}{P(today)}$$

$$P\left(No|today\right) = \frac{P\left(Sunny|No\right)P\left(Hot|No\right)P\left(Normal|No\right)P\left(False|No\right)P\left(No\right)}{P\left(today\right)}$$

$= \frac{\frac{3}{5} \cdot \frac{2}{5} \cdot \frac{1}{5} \cdot \frac{2}{5} \cdot \frac{5}{14}}{P(today)} = \frac{0.0068}{P(today)}$

As P(today) $>$ 0,

Clearly P(Yes) $>$ P(No)

So, predicted value for playing golf is YES.

For Problems with Zero Probability, use

Laplace smoothing: By applying this method, prior probability and conditional probability are, K is the number of different values in y and A is the number of different values in aj.

c) Neural Network Classifier (Back Propagation Network(BPN))

Understand basic terminologies - topology of a network, input layer, hidden layer, output layer, activation functions, weight, bias.

Strength and limitations of a neural network.

Take a sample network to learn the basics. For instance, learning the behaviour of OR gate, AND gate.

Take a network involving atleast one hidden layer and trace the BPN algorithm assuming a target class for one epoch.

Trace

Example Dataset

Class	x	y
class 1 (w1)	2	2
class 1 (w1)	-1	2
class 1 (w1)	1	3
class 1 (w1)	-1	-1
class 1 (w1)	0.5	0.5
class 2 (w2)	-1	-3
class 2 (w2)	0	-1
class 2 (w2)	1	-2
class 2 (w2)	-1	-2
class 2 (w2)	0	-2

We take a neural network with 3 layers

1) 1 – input layer (2 nodes + 1 bias)
2) 1 – hidden layer (2 nodes + 1 bias)
3) 1 – output layer (2 nodes)

Hyperparameters are taken as,

1) Learning Rate = 0.5
2) Activation Function – Sigmoid
3) Epochs = 1000
4) Weights = Random

Now, for forward propagation,
Value at a node is given by,

$$N_k^{i+1} = f\left(\sum_{j+1}^{n}\left(W_{jk}^i * N_j^i\right)\right)$$

N_k^{i+1} is the i+1 th node in k th layer (it also includes the bias node).
n is the total number of nodes in i th layer.
f(.) is the activation function.

Therefore, if we use the above formula, we get,

N_1^1 (H1) = f($W^1$11 ∗ x + $W^1$21 ∗ y + $W^1$31 ∗ 1)

N_2^1 (H2) = f($W^1$12 ∗ x + $W^1$22 ∗ y + $W^1$32 ∗ 1)

N_1^2 (O1) = f($W^2$11 ∗ H1 + $W^2$21 ∗ H2 + $W^2$31 ∗ 1)

N_2^2 (O2) = f($W^2$12 ∗ H1 + $W^2$22 ∗ H2 + $W^2$32 ∗ 1)

When we plug in the values for the first value of the dataset i.e. (2, 2), We get,

H1 = f(0.13436424411240122 ∗ 2 + 0.8474337369372327 ∗ 2 + 0.763774618976614 ∗ 1) =f(2.7273705810758817) = 0.9386225302230806

H2 = f(0.2550690257394217 ∗ 2 + 0.49543508709194095 ∗ 2 + 0.4494910647887381 ∗ 1) = f(1.9504992904514635) = 0.8755010741482664

O1 = f(0.651592972722763 ∗ 0.9386225302230806 + 0.7887233511355132 ∗ 0.8755010741482664 + 0.0938595867742349 ∗ 1) = f(1.3959875726318156) = 0.8015464048450159 29 B ANIRUDH

O2 = f(0.02834747652200631 ∗ 0.9386225302230806 + 0.8357651039198697 ∗ 0.8755010741482664 + 0.43276706790505337 ∗ 1) = f(1.1910878942610617) = 0.7669355767878618

Next is backpropagation of the error,

1) For a unit j in the output layer, the error Err_j is computed by,

$$Errj = Outj\,(1 - Outj\,) * (Tj - Outj)$$

Out_j (1 - Out_j) is the derivative of the logistic (sigmoid) function.

T_j is the target value of the O_j node.

2) In hidden layer,

$$Errj = Outj\,(1 - Outj\,) * \sum_k Err_k * W_{jk}$$

W_{jk} is the weight of the connection from unit j to a unit k in the next higher layer.

Err_k is the is the error of unit k.

The weights and biases are updated to reflect the propagated errors. Weights are updated by the following equations, where ΔW_{ij} is the change in weight W_{ij} :

$$\Delta W_{ij} = (l) * Outi$$

$$W_{ij} = W_{ij} + \Delta W_{ij}$$

(l – learning rate)

So, the error for Err1 for the output layer is,

Erri = 0.8015464048450159 ∗ (1 - 0.8015464048450159) ∗ (1 - 0.8015464048450159) = 0.03156796688859639

The change in weight for Hidden Layer is,

Δ W11 = (0.5) ∗ (0.03156796688859639) ∗ (0.9386225302230806) = 0.014815202477486385

W11 = 0.651592972722763 + 0.014815202477486385 = 0.6664081752002493

Similarly, the error for Err1 for the Hidden Layer is,

Erri = 0.9386225302230806 ∗ (1 - 0.9386225302230806) ∗ (0.03156796688859639 ∗ 0.651592972722763 + (0.7669355767878618) ∗ (1 - 0.7669355767878618) ∗ (0 - 0.7669355767878618) ∗ 0.02834747652200631) = 0.0009611362816286504

The change in weight for Hidden Layer is,

Δ W11 = (0.5) ∗ (0.0009611362816286504) ∗ (2) = 0.0009611362816286504

W11 = 0.13436424411240122 + 0.0009611362816286504 = 0.13532538039402986

Now, again forward propagate and backpropagate and update weights and repeat.

d) SVM or Nearest Neighbour Classifier (Anyone)

Understand the algorithm/working and give the pseudocode and the trace

Pseudo Code

k-Nearest Neighbour

Classify (X, Y, x) where X: training data, Y: class labels of X, x: test sample

For i = 1 to m do

    Computer distance d(Xi, x)

End for

Computer set I containing indices for the k smallest distances d(Xi, x)

Return majority label for { Yi where i ∈ I}

Trace

Consider binary Y with attributes X

K = 3

X1	X2	Y
1	2	False
2	3	True
3	1	True
4	3	False
5	5	False
6	1	False
7	2	True

Now for a test x = (2.5, 3.5),

D(X, x) => $\sqrt{(X1 - x1)^2 + (X2 - x2)^2}$

X1	X2	D(X, x)
1	2	2.12
2	3	0.71
3	1	2.55
4	3	1.58
5	5	2.91
6	1	4.30
7	2	4.74

As K = 3, minimum 3 distances is for (2, 3), (4, 3), (1, 2)

And their Y is True, False, False

SO, majority is FALSE = y for x

Q II

II) Understand the working of a simple genetic algorithm involving operators of selection, cross-over, mutation. Apply these operators to an optimisation function such as max f(x)= x3-2x2+x within a range of (0,31).

(Refer David E. Goldberg material for basics of GA)

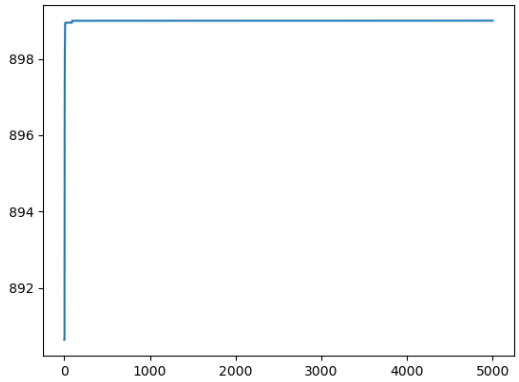
Code in GeneticOptimisation.py under Codes Folder

Run for

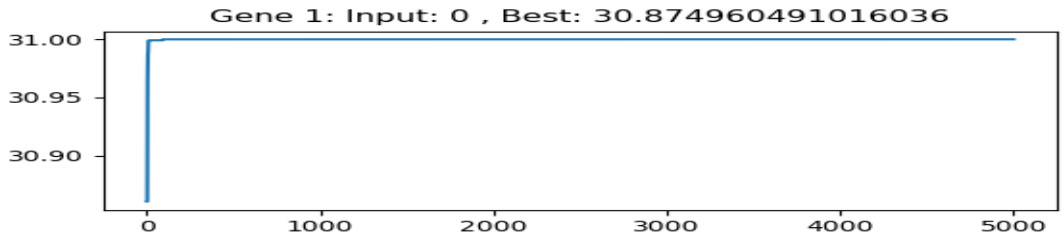
```
sol_per_pop = 200 #  Defining the population size.
num_generations = 5000
num_parents_mating = 100
select_mating_pool = select_mating_pool_bestfitness
crossover = crossover_OnePoint
mutation = mutation_UniformNoise
fitness_func = PolynomialFitness
boundary = (0, 31)
equation_inputs = [0, -2, 1]
num_weights = 1 #  Number of the weights we are looking to optimize.
```

Output:

Fitness Graph



Gene Updation Graph



Best Fitness: 898.9999935912551
Best Chromosome: [30.87496049]

Q III

III) Understand the working of Bucket Brigade Classifier[BBC] (discussed in Goldberg)

Bucket Brigade Classifier

Classifier systems in machine learning is useful to distinguish three levels of activity when looking at learning from the point of view of classifier systems: At the lowest level is the performance system. This is the part of the overall system that interacts directly with the environment. It is much like an expert system, though typically less domain-dependent. The performance systems we will be talking about are rule-based, as are most expert systems, but they are message-passing, highly standardized, and highly parallel. Rules of this kind are called classifiers. These Rules must be evaluated for their performance, and here is where we use BBC.

The third level of activity, the rule discovery system, is required because, even after the system has effectively evaluated millions of rules, it has tested only a minuscule portion of the plausibly useful rules. Selection of the best of that minuscule portion can give little confidence that the system has exhausted its possibilities for improvement; it is even possible that none of the rules it has examined is very good. The system must be able to generate

new rules to replace the least useful rules currently in place. The rules could be generated at random (say by " mutation" operators) or by running through a predetermined enumeration, but such " experience-independent" procedures produce improvements much too slowly to be useful in realistic settings. Somehow the rule discovery procedure must be biased by the system's accumulated experience. In the present context this becomes a matter of using experience to determine useful " building blocks" for rules; then new rules are generated by combining selected building blocks. Under this procedure the new rules are at least plausible in terms of system experience. (Note that a rule may be plausible without necessarily being useful of even correct.) The rule discovery system discussed here employs genetic algorithms.

Terms

1) Credit

   Generally the rules in the performance system are of varying usefulness and some, or even most, of them may be incorrect. Somehow the system must evaluate the rules. This activity is often called credit assignment (or apportionment of credit); accordingly this level of the system will be called the credit assignment system.

2) Strength of a rule

   A classifier system uses groups of rules as the representation. The structure of the concept is modeled by the organization, variability, and distribution of strength among the rules. Because the members of a group compete to become active, the appropriate aspects of the representation are selected only when they are relevant in a given problem solving context. The modularity of the concept thereby makes it easier to use as well as easier to modify. Basically, strength of a rule tells if the rule is applicable to a particular/ given environment.

3) Classifier

The starting point for this approach to machine learning is a set of rulebased systems suited to rule discovery algorithms. The rules must lend themselves to processes that extract and recombine " building blocks" from currently useful rules to form new rules, and the rules must interact simply and in a highly parallel fashion. Classifier systems are parallel, message-passing, rule-based systems wherein all rules have the same simple form. In the simplest version all messages are required to be of a fixed length over a specified alphabet, typically k-bit binary strings. The rules are in the usual condition/ action form. The condition part specifies what kinds of messages satisfy (activate) the rule and the action part specifies what message is to be sent when the rule is satisfied. A classifier system consists of four basic parts.

a. The input interface translates the current state of the environment into standard messages. For example, the input interface may use property detectors to set the bit values (1: the current state has the property, 0: it does not) at given positions in an incoming message.

b. The classifiers, the rules used by the system, define the system's procedures for processing messages.

c. The message list contains all current messages (those generated by the input interface and those generated by satisfied rules).

d. The output interface translates some messages into effects or actions, actions that modify the state of the environment.

A classifier system's basic execution cycle consists of the following steps:
Step 1: Add all messages from the input interface to the message list.
Step 2: Compare all messages on the message list to all conditions of all classifiers and record all matches (satisfied conditions).
Step 3: For each set of matches satisfying the condition part of some classifier, post the message specified by its action part to a list of new messages.
Step 4: Replace all messages on the message list by the list of new messages.
Step 5: Translate messages on the message list to requirements on the output interface, thereby producing the system's current output.
Step 6: Return to Step 1.
Individual classifiers must have a simple, compact definition if they are to serve as appropriate grist for the learning mill; a complex, interpreted definition makes it difficult for the learning algorithm to find and exploit building blocks from which to construct new rules. The major technical hurdle in implementing this definition is that of providing a simple specification of the condition part of the rule. Each condition must specify exactly the set of messages that satisfies it. Though most large sets can be defined only by an explicit listing, there is one class of subsets in the message space that can be specified quite compactly, the hyperplanes in that space. Specifically, let 1, 0 be the set of possible k-bit messages; if we use " " as a " don't care" symbol, then the set of hyperplanes can be designated by the set of all ternary strings of length k over the alphabet 1, 0, . For example, the string 1... designates the set of all messages that start with a 1, while the string 00... 0 specifies the set 00... 01, 00... 00 consisting of exactly two messages, and so on. It is easy to check whether a given message satisfies a condition. The condition and the message are matched position by position, and if the entries at all non- positions are identical, then the message satisfies the condition. The notation is extended by allowing any string c over 1, 0, to be prefixed by a " -" with the intended interpretation that -c is satisfied just in case no message satisfying c is present on the message list.

Credit Appointment using BBC

The first major learning task facing any rule-based system operating in a complex environment is the credit assignment task. Somehow the performance system must determine both the rules responsible for its successes and the representativeness of the conditions encountered in attaining the successes.The task is difficult because overt rewards are rare in complex environments; the system's behavior is mostly " stage-setting" that makes possible later successes. The problem is even more difficult for parallel systems, where only some of the rules active at a given time may be instrumental in attaining later success. An environment exhibiting perpetual novelty adds still another order of complexity. Under such conditions the performance system can never have an absolute assurance that any of its rules is " correct." The perpetual novelty of the environment, combined with an always limited sampling of that environment, leaves a residue to uncertainty. Each rule in effect serves as a hypothesis that has been more or less confirmed. The bucket brigade algorithm is designed to solve the

credit assignment problem for classifier systems. To implement the algorithm, each classifier is assigned a quantity called its strength. The bucket brigade algorithm adjusts the strength to reflect the classifier's overall usefulness to the system. The strength is then used as the basis of a competition. Each time step, each satisfied classifier makes a bid based on its strength, and only the highest bidding classifiers get their messages on the message list for the next time step. It is worth recalling that there are no consistency requirements on posted messages; the message list can hold any set of messages, and any such set can direct further competition. The only point at which consistency enters is at the output interface. Here, different sets of messages may specify conflicting responses. Such conflicts are again resolved by competition. For example, the strengths of the classifiers advocating each response can be summed so that one of the conflicting actions is chosen with a probability proportional to the sum of its advocates.

The bidding process is specified as follows. Let s(C, t) be the strength of classifier C at time t. Two factors clearly bear on the bidding process:

1) relevance to the current situation.

2) past " usefulness" .

Relevance is mostly a matter of the specificity of the rule's condition part–a more specific condition satisfied by the current situation conveys more information about that situation. The rule's strength is supposed to reflect its usefulness. In the simplest versions of the competition the bid is a product of these two factors, being 0 if the rule is irrelevant (condition not satisfied) or useless (strength 0), and being high when the rule is highly specific to the situation (detailed conditions satisfied) and well confirmed as useful (high strength). To implement this bidding procedure, we modify Step 3 of the basic execution cycle,
For each set of matches satisfying the condition part of classifier C, calculate a bid according to the following formula,

$$B\left(C,t\right)=b*R\left(C\right)s\left(C,t\right)$$

R(C) is the specificity, equal to the number of non- in the condition part of C divided by the length thereof.
b is a constant less then one.
The size of the bid determines the probability that the classifier posts its message (specified by the action part) to the new message list. (E.g., the probability that the classifier posts its message might decrease exponentially as the size of the bid decreases.)
The use of probability in the revised step assures that rules of lower strength sometimes get tested, thereby providing for the occasional testing of less favored and newly generated (lower strength) classifiers (" hypotheses").
The operation of the bucket brigade algorithm can be explained informally via an economic analogy. The algorithm treats each rule as a kind of " middleman" in a complex economy. As a " middleman," a rule only deals with its " suppliers" –the rules sending messages satisfying

its conditions–and its " consumers" –the rules with conditions satisfied by the messages the " middleman" sends. Whenever a rule wins a bidding competition, it initiates a transaction wherein it pays out part of its strength to its suppliers. (If the rule does not bid enough to win the competition, it pays nothing.) As one of the winners of the competition, the rule becomes active, serving as a supplier to its consumers, and receiving payments from them in turn. Under this arrangement, the rule's strength is a kind of capital that measures its ability to turn a " profit." If a rule receives more from its consumers than it paid out, it has made a profit; that is, its strength has increased.
More formally, when a winning classifier C places its message on the message list it pays for the privilege by having its strength s(C, t) reduced by the amount of the bid B(C, t),

$$s\left(C, t+1\right) = s\left(C, t\right) - B\left(C, t\right)$$

The Classifiers C' sending messages matched by this winner, the " suppliers," have their strengths increased by the amount of the bid–it is shared among them in the simplest version

$$s\left(C0, t+1\right) = s\left(C0, t\right) + aB\left(C, t\right)$$

a = $\frac{1}{\left(|C'|\right)}$

A rule is likely to be profitable only if its consumers, in their local transactions, are also (on the average) profitable. The consumers, in turn, will be profitable only if their consumers are profitable. The resulting chains of consumers lead to the ultimate consumers, the rules that directly attain goals and receive payoff directly from the environment. (Payoff is added to the strengths of all rules determining responses at the time the payoff occurs.) A rule that regularly attains payoff when activated is of course profitable. The profitability of other rules depends upon their being coupled into sequences leading to these profitable ultimate consumers. The bucket brigade ensures that early acting, " stage-setting" rules eventually receive credit if they are coupled into (correlated with) sequences that (on average) lead to payoff. If a rule sequence is faulty, the final rule in the sequence loses strength, and the sequence will begin to disintegrate, over time, from the final rule backwards through its chain of precursors. As soon as a rule's strength decreases to the point that it loses in the bidding process, some competing rule will get a chance to act as a replacement. If the competing rule is more useful than the one displaced, a revised rule sequence will begin to form using the new rule. The bucket brigade algorithm thus searches out and repairs " weak links" through its pervasive local application. Whenever rules are coupled into larger hierarchical knowledge structures, the bucket brigade algorithm is still more powerful than the description so far would suggest. Consider an abstract rule C∗ of the general form, " if the goal is G, and if the procedure P is executed, then G will be achieved." C∗ will be active throughout the time interval in which the sequence of rules comprising P is executed. If the goal is indeed achieved, this rule serves to activate the response that attains the goal, as well as the stage-setting responses preceding that response. Under the bucket brigade C∗ will be strengthened immediately by the goal attainment. On the very next trial involving P, the earliest rules in P will have their strengths substantially increased under the bucket brigade. This happens because the early rules act as suppliers to the strengthened C∗ (via the condition " if the procedure P is executed"). Normally, the process would have to be

executed on the order of n times to back chain strength through an n-step process P. C∗ circumvents this necessity.

Q V

V) Understand confusion matrix and various performance measures associated with classification tasks such as accuracy, sensitivity, specificity, precision, recall, F1 score etc. Consider a sample binary classifier results (TP, FP, TN, FN) and compute various measures.

Confusion Matrix

A confusion matrix is a summary of predicted results on a classification problem
Terms,

1) Positive (P): outcome/observation is positive (Eg. it is a dog)
2) Negative (N): outcome/observation is not positive (Eg. it is not a dog)
3) True Positive (TP): prediction and observation are positive
4) False Positive (FP): prediction is positive, but true observation is negative
5) False Negative (FN): prediction is negative, but true observation is positive
6) True Negative (TN): prediction and observation are not positive

Formulas,

1) Recall = $\frac{TP}{TP+FN}$ (Also called sensitivity)
   High Recall means the recognized class is correct
   Low Recall means the recognized class is wrong

2) Precision = $\frac{TP}{TP+FP}$
   High Precision means if outcome is positive, in reality also it is positive

3) Specificity = $\frac{TN}{TN+FN}$

High Specificity means if outcome is negative, in reality also it is negative

4) Accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$

5) F1 Score = $\frac{2}{Precision+Recall}$

Q VI

VI) Clustering Algorithms (Euclidean distance may be used)

1) Understand the working of k-means clustering algorithm. Give a pseudo code for the same and trace it for a sample dataset of your choice, clearly showing the centroid updates.

Pseudo Code

a) Choose random K Samples (initial centroid)

b) For specified no of iterations,

(i) Form k clusters by assigning items to their closest mean/centroid

(ii) Update the mean points by taking mean of each cluster

(iii) Repeat

Trace

Dataset

	X	Y
0	3	4
1	7	5
2	2	6
3	3	1
4	8	2
5	7	3
6	4	4
7	6	6
8	7	4
9	6	7

Choose random K = 2 means,

Let they be C1 – (1, 1) and C2 – (7, 7)

Each item is assigned to whichever cluster has least distance

So, C1 – 0, 2, 3, 6, and C2 – 1, 4, 5, 7, 8, 9

Now find new mean of the clusters

C1(new) = ((3+2+3+4) / 4 , (4+6+1+4) / 4) = (3, 3.75)

C2(new) = ((7+8+7+6+7+6) / 6 , (5+2+3+6+4+7) / 6) = (6.83, 4.5)

Thus, means has been updated

Now, Repeat same process

2) Understand the working of k-medoids clustering algorithm. Give a pseudo code for the same and trace it for the sample dataset used for VI-(a), clearly showing the centroid updates.

Pseudo Code

a) Choose random K items from the given dataset (initial medoids)

b) For specified no of iterations,

   (i) Form k clusters by assigning items to their closest medoid

   (ii) Calculate the total dissimilarity

   (iii) Now, select a random non-medoid item as medoid and repeat (a) and (b)

   (iv) If cost/total dissimilarity is more for previous set of medoids than for the new medoid, then replace the old set with new set of medoid and Repeat from (a)

   (v) If cost/total dissimilarity is more for new set of medoids than for the previous medoid, then rollback and keep the old set of medoid and Repeat (c)

Trace

Dataset

	X	Y
0	3	4
1	7	5
2	2	6
3	3	1
4	8	2
5	7	3
6	4	4
7	6	6
8	7	4
9	6	7

Choose random K = 2 medoids,

Let they be C1 – (3, 1) (I3) and C2 – (6, 6) (I7)

Each item is assigned to whichever cluster has least dissimilarity

So, C1 – 0, 4, 6 and C2 – 1, 2, 5, 8, 9

Total Diss = (3 + 6 + 4) + (2 + 4 + 4 + 3 + 1) = 13 + 14 = 27

Now, we choose randomly, (I8) = (7, 4) as medoid

C1 – (3, 1) and C2 – (7, 4)

Now, assignment is, C1 – 0, 2 and C2 – 1, 4, 5, 6, 7, 9

Total Diss = (3 + 6) + (1 + 3 + 1 + 3 + 3 + 4) = 9 + 15 = 24

As 24 $<$ 27, we change medoids to new set – C1 – (3, 1) and C2 – (7, 4)

Repeat above process

3) Understand the working of hierarchical clustering algorithm- Agglomerative, Divisive and trace them for the dataset used in VI-(a). You may trace the algorithm for both the approaches and use the dendrogram to represent the clustering process pictorially as well.

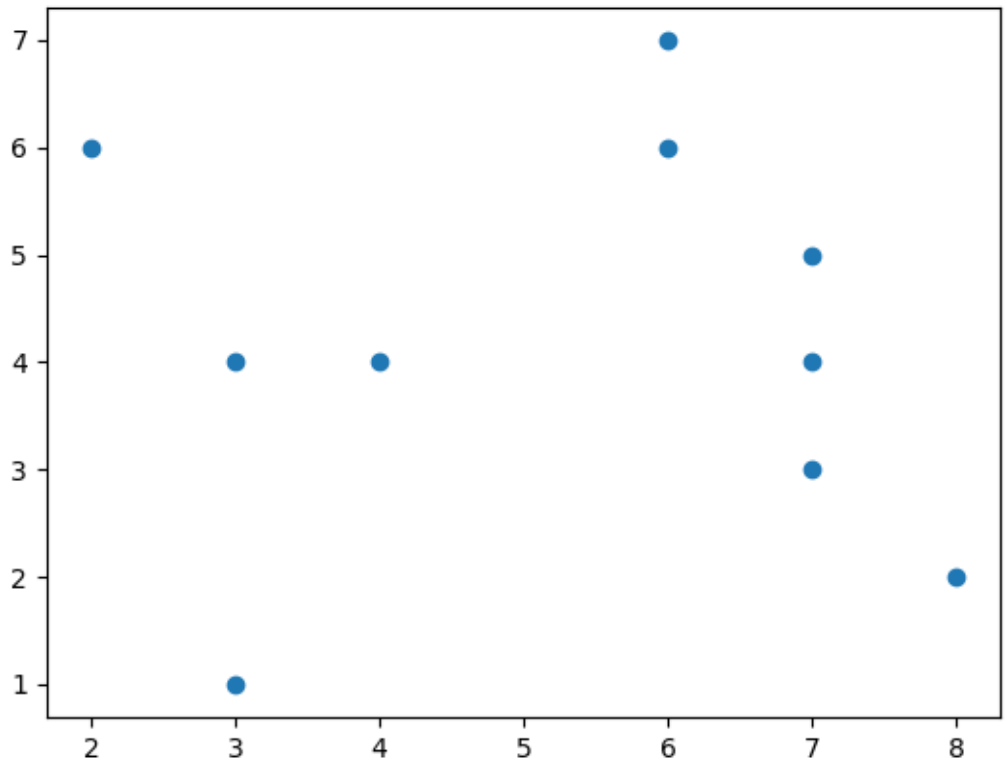
Trace

Agglomerative Clustering

Dataset

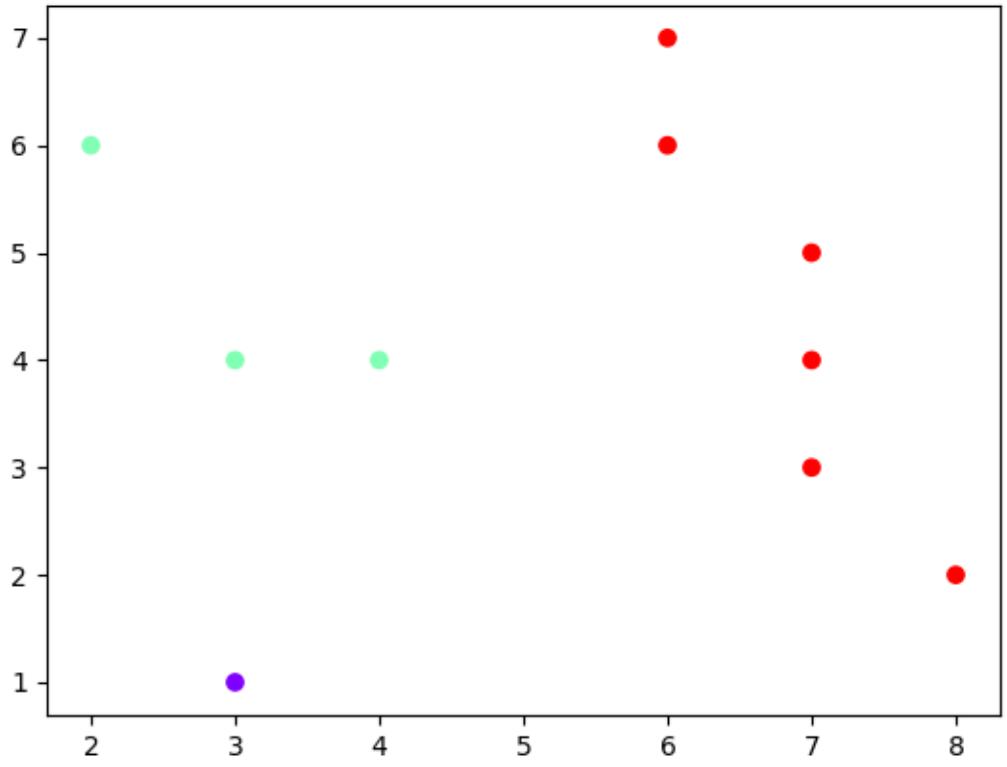
	X	Y
0	3	4
1	7	5
2	2	6
3	3	1
4	8	2
5	7	3
6	4	4
7	6	6
8	7	4
9	6	7

Initially, every item is a cluster – C0, C1, . . . C9



Closest 2 items are (3, 4) and (4, 4) (C0 and C6)

So, Merge them into 1 cluster C06 – (3.5, 4.0)
Next Closest 2 items are (7, 5) and (7, 4) (C1 and C8)
So, Merge them into 1 cluster C18 – (7.0, 4.5)
Simlarly merging,
Finally 3 Clusters



Q VII

VII) Survey the various distance measures used by clustering algorithms eg: Cosine, Jaccard similarity measures etc.

Explore for a minimum of 5 measures (non-Euclidean distance measures) and trace them to measure distance 2 data points.

Cosine Distance

$$s(\mathbf{x},\mathbf{y}) = \frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} = \frac{\sum_{i=0}^{n-1} x_i y_i}{\sqrt{\sum_{i=0}^{n-1}(x_i)^2} \times \sqrt{\sum_{i=0}^{n-1}(y_i)^2}}$$

X = (2, 3)
Y = (4, 1)
Distance = $\frac{2*4+3*1}{\sqrt{2^2+3^2}*\sqrt{4^2+1^2}} = \frac{11}{\sqrt{13}*\sqrt{17}} = \frac{11}{14.87} = 0.739$

Jaccard Distance

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

A = { 1, 2, 3, 4}
B = { 1, 2}
J(A, B) = 2 / 4 = 0.5
Jaccard Distance = 1 – Jaccard Similarity = 1 – 0.5 = 0.5
Manhattan Distance

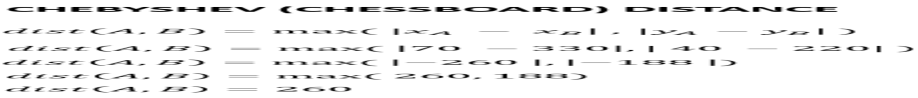
$$distance = \sum_{i=0}^{n-1} |(x[i] - y[i])|$$

X = (4, 3)
Y = (8, 2)
Manhattan Distance = | (4 - 8)| + | (3 - 2)| = 4 + 1 = 5

Bray-Curtis Distance

Bray Curtis	$d(x,y) = \frac{\sum \lvert x_i - y_i \rvert}{\sum x_i + y_i}$

X = (3, 1)
Y = (7, 3)
Bray-Curtis Distance = $\frac{|3-7|+|1-3|}{(3+7)+(1+3)} = \frac{6}{14} = 0.429$
Chebychev Distance



A = (3, 6)
B = (1, 7)
Chebychev Distance = max (| 3-1| , | 6-7|) = max (2, 1) = 2

Q VIII

VIII)
a) Test drive/Implement Decision tree, Naive Bayes, BPN, k-means, hierarchical clustering in a platform of your choice.
Decision Tree

```
Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:    class name  left-weight  left-distance  right-weight  right-distance
0           B           1             1             1              1
1           R           1             1             1              2
2           R           1             1             1              3
3           R           1             1             1              4
4           R           1             1             1              5
```

```
Results Using Gini Index:
Predicted values:
['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:
 [[ 0  6  7]
 [ 0 67 18]
 [ 0 19 71]]
Accuracy:
 73.40425531914893
C:\Users\Kausik N\AppData\Local\Programs\Python\Python38-32\lib\site-packag
 and F-score are ill-defined and being set to 0.0 in labels with no predict
  _warn_prf(average, modifier, msg_start, len(result))
Report:
              precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.73      0.79      0.76        85
           R       0.74      0.79      0.76        90

    accuracy                           0.73       188
   macro avg       0.49      0.53      0.51       188
weighted avg       0.68      0.73      0.71       188
```

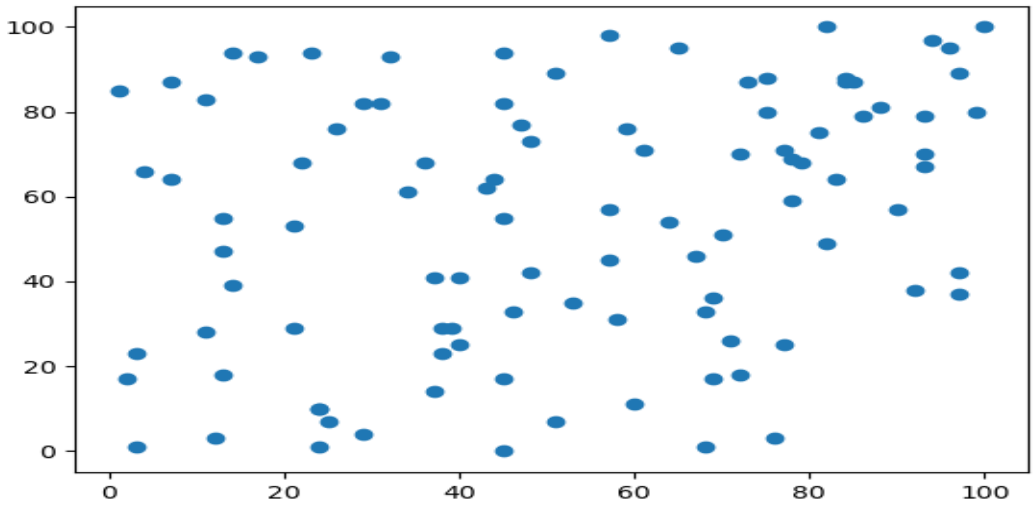
```
Results Using Entropy:
Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:
 [[ 0  6  7]
 [ 0 63 22]
 [ 0 20 70]]
Accuracy:
 70.74468085106383
C:\Users\Kausik N\AppData\Local\Programs\Python\Python38-32\lib\site-package
 and F-score are ill-defined and being set to 0.0 in labels with no predicte
  _warn_prf(average, modifier, msg_start, len(result))
Report:
              precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.71      0.74      0.72        85
           R       0.71      0.78      0.74        90

    accuracy                           0.71       188
   macro avg       0.47      0.51      0.49       188
weighted avg       0.66      0.71      0.68       188
```

Naive Bayes Classifier
Using Iris Dataset

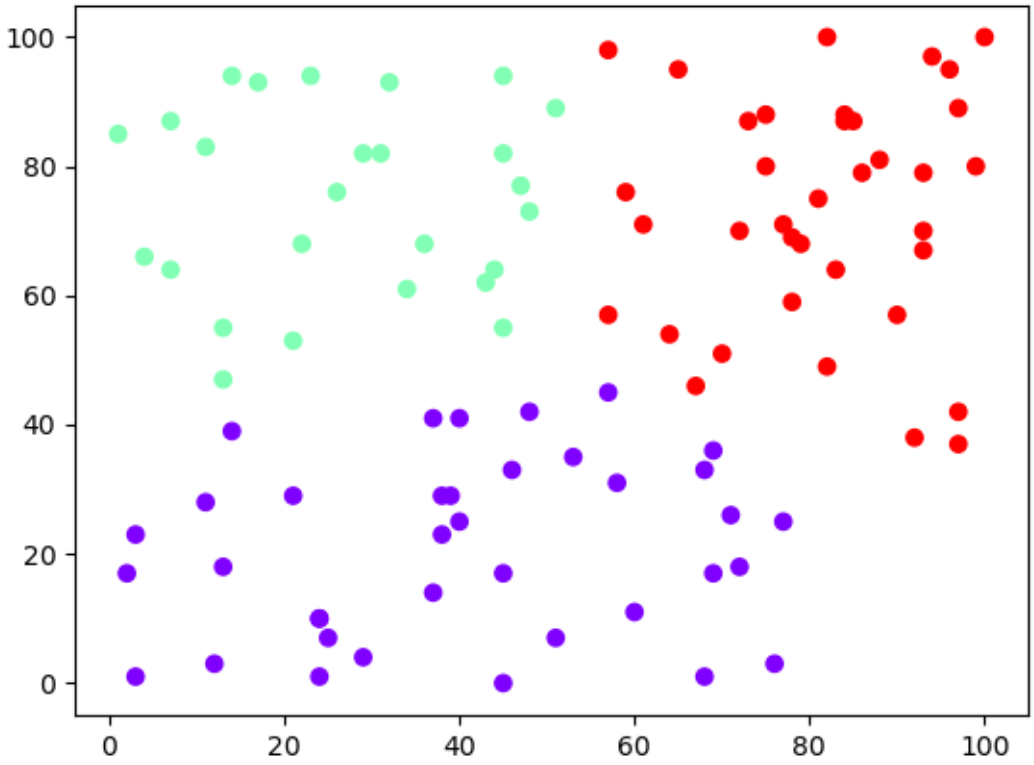
```
Results Using Bayes Classifier:
Predicted values:
[0 1 1 0 2 2 2 0 0 2 1 0 2 1 1 0 1 1 0 0 1 1 2 0 2 1 0 0 1 2 1 2 1 2 2 0 1
 0 1 2 2 0 1 2 1 2 0 0 0 1 0 0 2 2 2 2 2 1 2 1]
Confusion Matrix:
 [[19  0  0]
 [ 0 19  2]
 [ 0  1 19]]
Accuracy:
 95.0
Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.95      0.90      0.93        21
           2       0.90      0.95      0.93        20

    accuracy                           0.95        60
   macro avg       0.95      0.95      0.95        60
weighted avg       0.95      0.95      0.95        60
```

K-Means Clustering

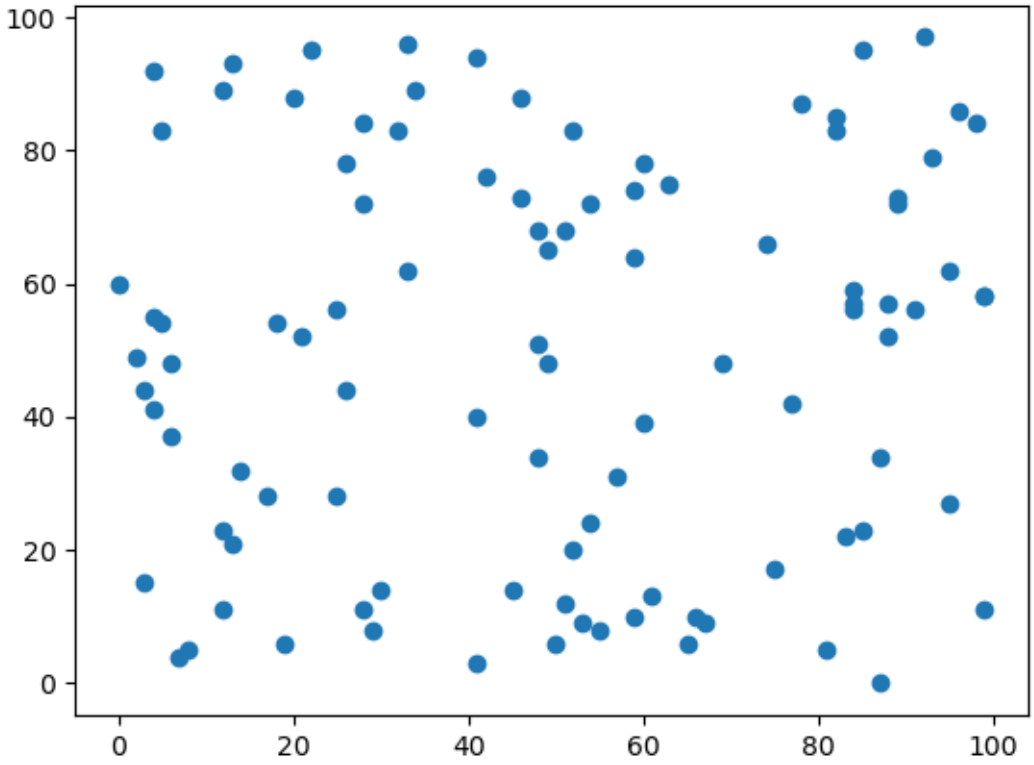
100 Random Points between 0 to 100 values



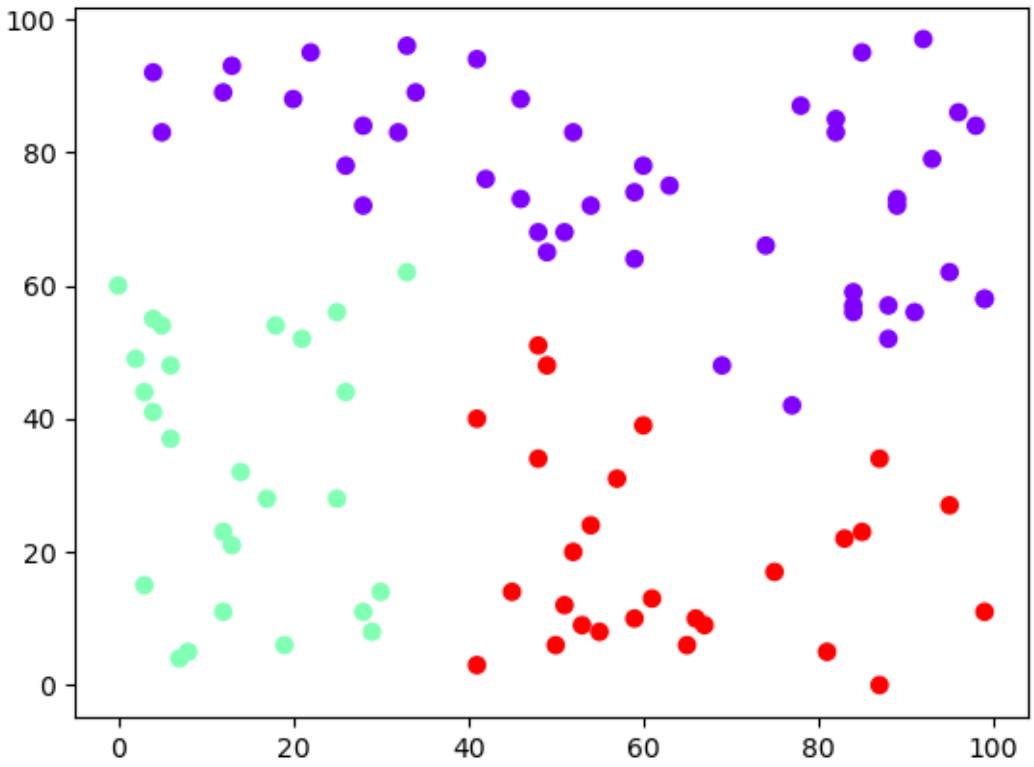
3 Clusters



Agglomerative Clustering
100 Random Points between 0 to 100



3 Clusters



b) Test drive/Implement optimization using GA operator in a platform of your choice (Not mandatory)

Inputs

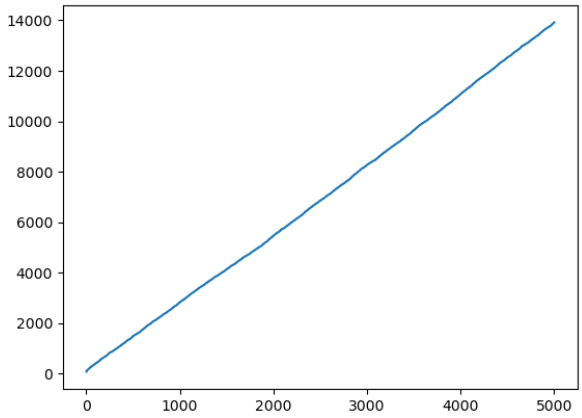
```
sol_per_pop = 200 #  Defining the population size.
num_generations = 5000
num_parents_mating = 100

select_mating_pool = select_mating_pool_bestfitness
crossover = crossover_OnePoint
mutation = mutation_UniformNoise
fitness_func = PolyLinearFitness
boundary = (None, None)
equation_inputs = [4, -2, 3.5, 5, -11, -4.7, 2.5, 0.1]
num_weights = len(equation_inputs) #  Number of the weights we are looking to optimize.
```

After Optimising,

Best Fitness: 13920.720451463068

Best Chromosome: [411.87721227 -240.46700836 384.55563335 444.05949469 -501.41256815 -415.91873926 302.42510916 16.55506078]

Fitness Graph



Gene Updation Graph