Exploratory Assignment

COE17B010

Q1

Understand the working of Hyperlink Induced Topic Search (HITS). Give a pseudocode and illustrate the same over a sample dataset of your choice.

Introduction:

Hyperlink Induced Topic Search (HITS) Algorithm is a Link Analysis Algorithm that rates webpages, developed by Jon Kleinberg. This algorithm is used to the web link-structures to discover and rank the webpages relevant for a particular search.
HITS uses hubs and authorities to define a recursive relationship between webpages. Before understanding the HITS Algorithm, we first need to know about Hubs and Authorities.

- Given a query to a Search Engine, the set of highly relevant web pages are called **Roots**. They are potential **Authorities**.
- Pages which are not very relevant but point to pages in the Root are called **Hubs**. Thus, an Authority is a page that many hubs link to whereas a Hub is a page that links to many authorities.

Pseudo Code:

Fix iterations: k

1) Each node is assigned a Hub score = 1 and an Authority score = 1.
2) Repeat k times:
   a) **Hub update:**

$$Each node's Hub score = \sum (Authority score of each node it points to)$$

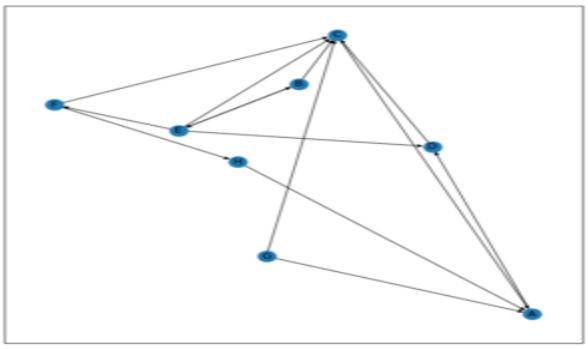
   b) **Authority update:**

$$Each node's Authority score = \sum (Hub score of each node pointing to it)$$

   c) Normalize the scores by dividing each Hub score by square root of the sum of the squares of all Hub scores, and dividing each Authority score by square root of the sum of the squares of all Authority scores. (optional)

Trace:

Input Dataset



Fixing k = 3,

Initially,

Hub Scores:	Authority Scores:
A -> 1	A -> 1
B -> 1	B -> 1
C -> 1	C -> 1
D -> 1	D -> 1
E -> 1	E -> 1
F -> 1	F -> 1
G -> 1	G -> 1
H -> 1	H -> 1

After 1st iteration,

Hub Scores:	Authority Scores:
A -> 1	A -> 3
B -> 2	B -> 2
C -> 1	C -> 4
D -> 2	D -> 2
E -> 4	E -> 1
F -> 1	F -> 1
G -> 2	G -> 0

H -> 1 H -> 1

After 2nd iteration,

Hub Scores: Authority Scores:

A -> 2 A -> 4

B -> 5 B -> 6

C -> 3 C -> 7

D -> 6 D -> 5

E -> 9 E -> 2

F -> 1 F -> 4

G -> 7 G -> 0

H -> 3 H -> 1

After 3rd iteration,

Hub Scores: Authority Scores:

A -> 5 A -> 13

B -> 9 B -> 15

C -> 4 C -> 27

D -> 13 D -> 11

E -> 22 E -> 5

F -> 1 F -> 9

G -> 11 G -> 0

H -> 4 H -> 3

Q2

Understand the working of BIRCH and DBSCAN clustering algorithms. Give a pseudocode and trace the same over a sample dataset of your choice.

1) BIRCH Clustering

Introduction:

Balanced Iterative Reducing and Clustering using Hierarchies, or BIRCH for short, deals with large datasets by first generating a more compact summary that retains as much distribution information as possible, and then clustering the data summary instead of the original dataset. BIRCH actually complements other clustering algorithms by virtue if the fact that different clustering algorithms can be applied to the summary produced by BIRCH. BIRCH can only deal with metric attributes (similar to the kind of features KMEANS can handle). A metric attribute is one whose values can be represented by explicit coordinates in an Euclidean space (no categorical variables).

Pseudo Code:

Take the initial Data,

- **Phase 1:**Load data into memory
  Scan DB and load data into memory by building a CF tree. If memory is exhausted rebuild the tree from the leaf node.
- **Phase 2:**Condense data
  Resize the data set by building a smaller CF tree
  Remove more outliers
  Condensing is optional
- **Phase 3:**Global clustering
  Use existing clustering algorithm (e.g. KMEANS, HC) on CF entries
- **Phase 4:**Cluster refining

  Refining is optional

  Fixes the problem with CF trees where same valued data points may be assigned to different leaf entries.

Trace:

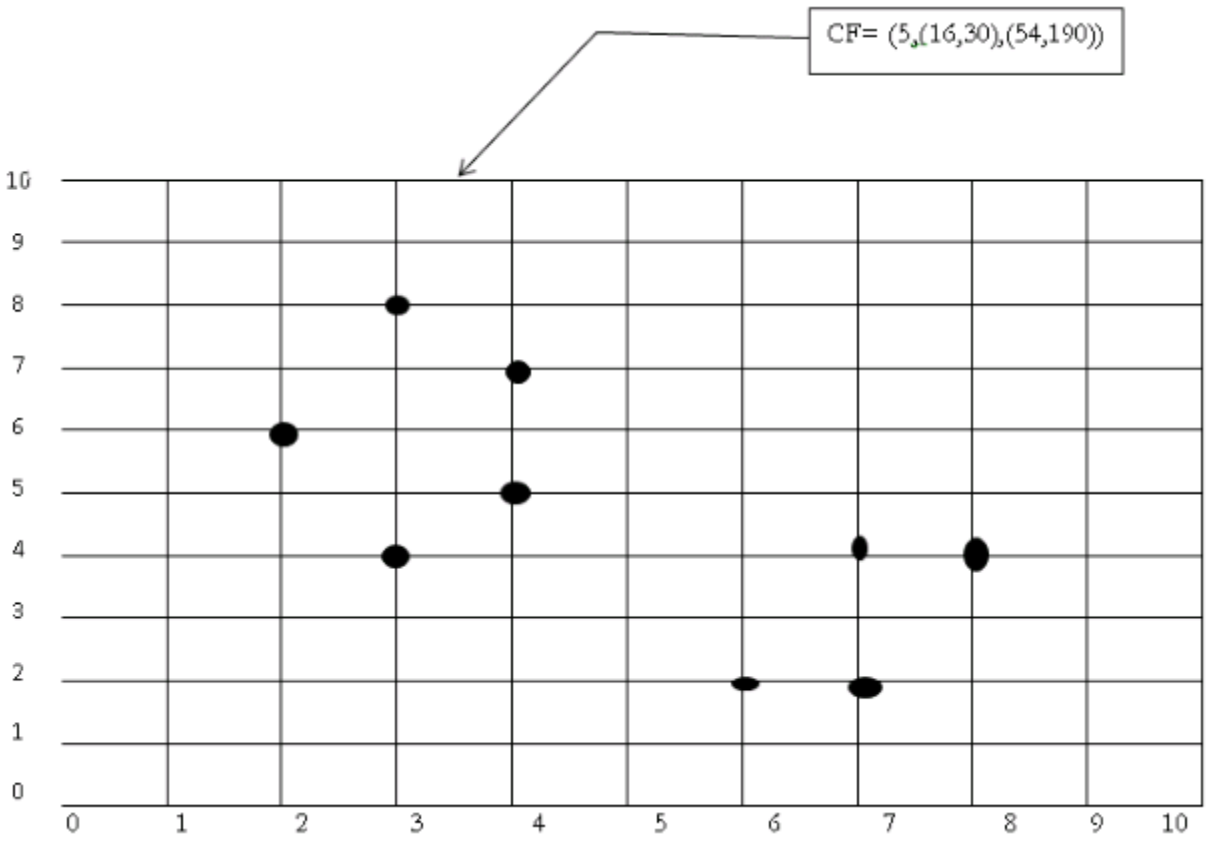
CF= (N, LS, SS)

$$N = No of data points$$

$$LS = \sum_{i=1}^{N} X_i$$

$$SS = \sum_{i=1}^{N} X_i^2$$

Example Trace

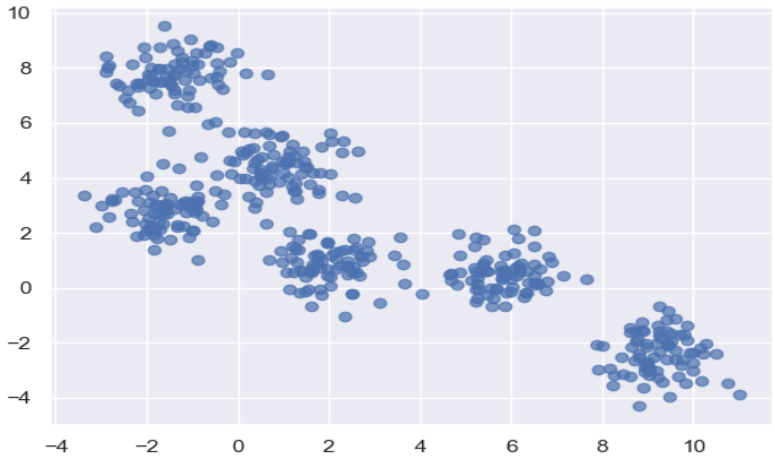


(3,4) (2,6)(4,5)(4,7)(3,8)

N=5

NS= (16, 30) i.e. 3+2+4+4+3=16 and 4+6+5+7+8=30
SS=(54,190)=3^2+2^2+4^2+4^2+3^2=54 and 42+62+52+72+82= 190

Input Data



After Clustering,

2) DBSCAN Clustering

Introduction:

Fundamentally, all clustering methods use the same approach i.e. first we calculate similarities and then we use it to cluster the data points into groups or batches. Here we will focus on **Density-based spatial clustering of applications with noise** (DBSCAN) clustering method.

Clusters are dense regions in the data space, separated by regions of the lower density of points. The ***DBSCAN algorithm*** is based on this intuitive notion of "clusters" and "noise" . The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points.

DBSCAN algorithm requires two parameters –

1. **eps** : It defines the neighborhood around a data point i.e. if the distance between two points is lower or equal to 'eps' then they are considered as neighbors. If the eps value is chosen too small then large part of the data will be considered as outliers. If it is chosen very large then the clusters will merge and majority of the data points will be in the same clusters. One way to find the eps value is based on the ***k-distance graph***.

2. **MinPts**: Minimum number of neighbors (data points) within eps radius. Larger the dataset, the larger value of MinPts must be chosen. As a general rule, the minimum MinPts can be derived from the number of dimensions D in the dataset as, MinPts >= D+1. The minimum value of MinPts must be chosen at least 3.

In this algorithm, we have 3 types of data points.

Core Point*: A point is a core point if it has more than MinPts points within eps.*

Border Point*: A point which has fewer than MinPts within eps but it is in the neighborhood of a core point.*

Noise or outlier*: A point which is not a core point or border point.*

Pseudo Code:

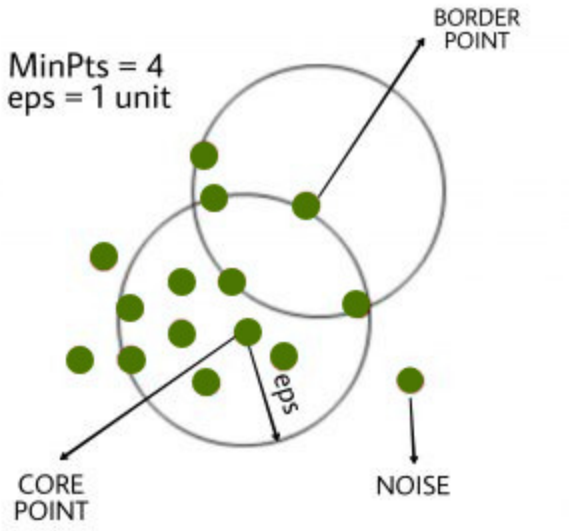
1. Find all the neighbor points within eps and identify the core points or visited with more than MinPts neighbors.

2. For each core point if it is not already assigned to a cluster, create a new cluster.

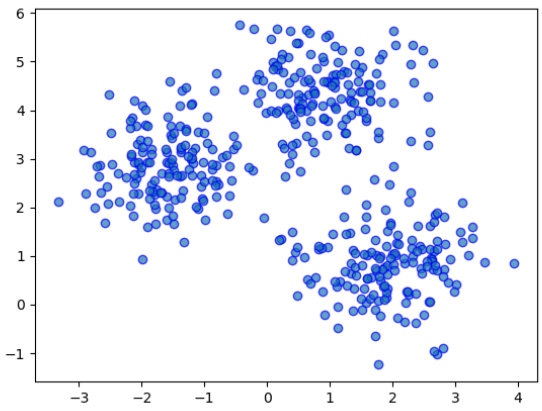
3. Find recursively all its density connected points and assign them to the same cluster as the core point.
   A point a and b are said to be density connected if there exist a point c which has a sufficient number of points in its neighbors and both the points a and b are within the *eps distance*. This is a chaining process. So, if b is neighbor of c, c is neighbor of d, d is neighbor of e, which in turn is neighbor of a implies that b is neighbor of a.

4. Iterate through the remaining unvisited points in the dataset. Those points that do not belong to any cluster are noise.

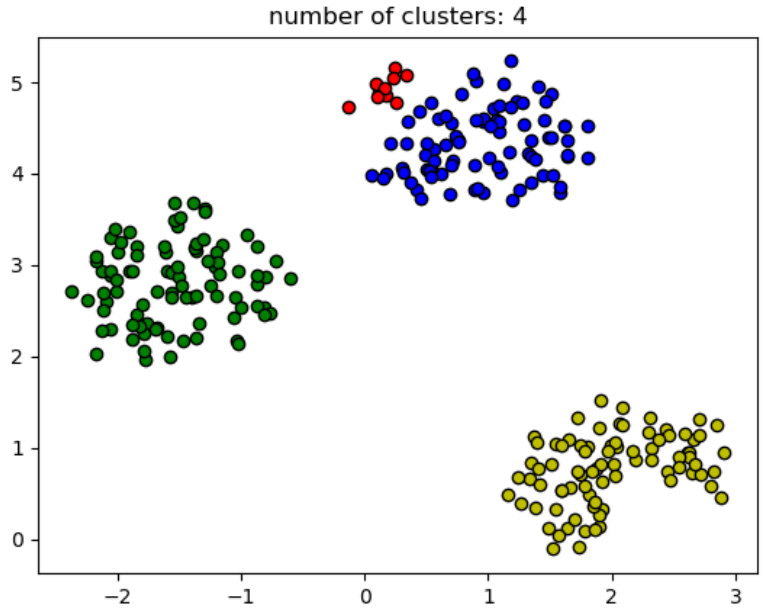
Trace:



Here, first we identify core points and as shown in diagram,
The marked point is a core point as it has > 4 neighbors within eps 1 unit.
Hence the core point is assigned a cluster A.
Similarly, we can identify that all the points are belonging to cluster A itself as neighbors of the central core point also have > 4 neighbors within 1-unit eps thereby connecting their neighbors under same cluster as them, i.e. Cluster A.
Only that 1 point is marked noise as it has no nearby points with > 4 neighbors within 1-unit eps. Hence it does not belong to any cluster, hence noise.
Input Data



Output Clusters



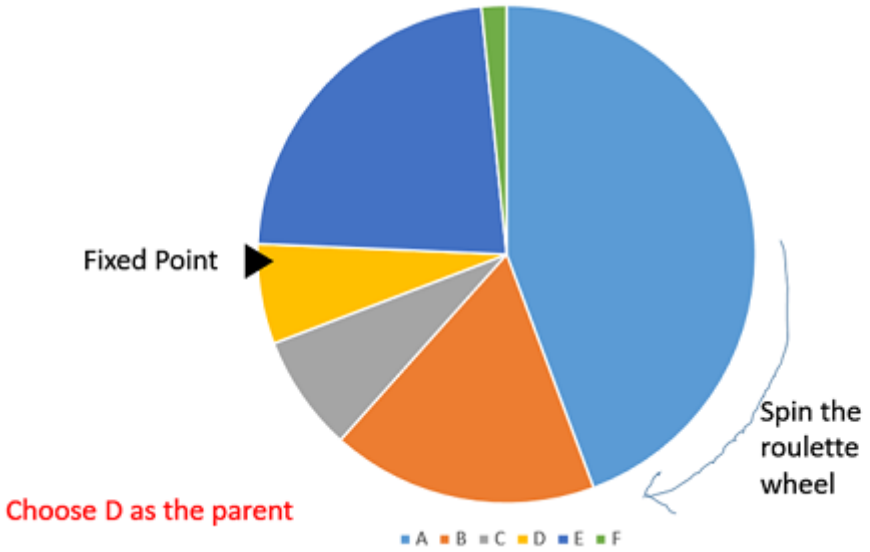
Q3

Understand the different types of selection, cross-over, mutation operation that are employed by Genetic algorithms. Illustrate the working of each over a population of chromosomes (Refer David Goldberg or any other online sources). Explore a minimum of at least 4 to 5 operators under each type.

1) Selection Operators

Roulette Wheel Selection

In a roulette wheel selection, the circular wheel is divided as described before. A fixed point is chosen on the wheel circumference as shown and the wheel is rotated. The region of the wheel which comes in front of the fixed point is chosen as the parent. For the second parent, the same process is repeated.



Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

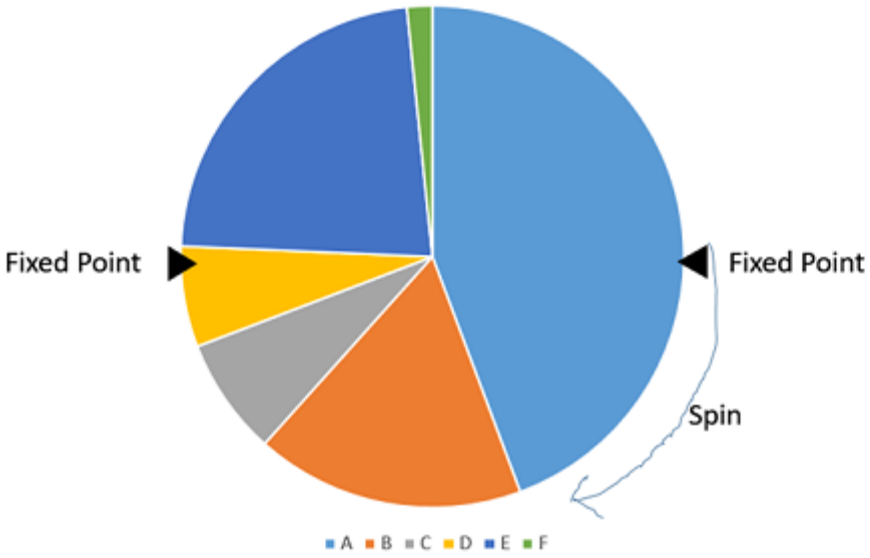
It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated. Therefore, the probability of choosing an individual depends directly on its fitness.

Implementation wise, we use the following steps −

- Calculate S = the sum of a fitnesses.
- Generate a random number between 0 and S.
- Starting from the top of the population, keep adding the finesses to the partial sum P, till P<S.
- The individual for which P exceeds S is the chosen individual.

Stochastic Universal Sampling (SUS)

Stochastic Universal Sampling is quite similar to Roulette wheel selection, however instead of having just one fixed point, we have multiple fixed points as shown in the following image. Therefore, all the parents are chosen in just one spin of the wheel. Also, such a setup encourages the highly fit individuals to be chosen at least once.

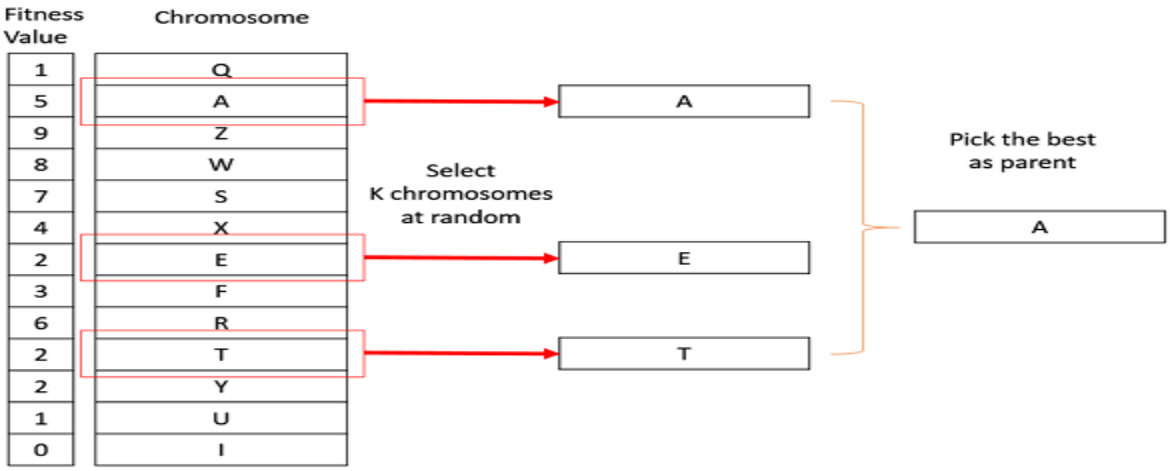


Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

It is to be noted that fitness proportionate selection methods don't work for cases where the fitness can take a negative value.

Tournament Selection

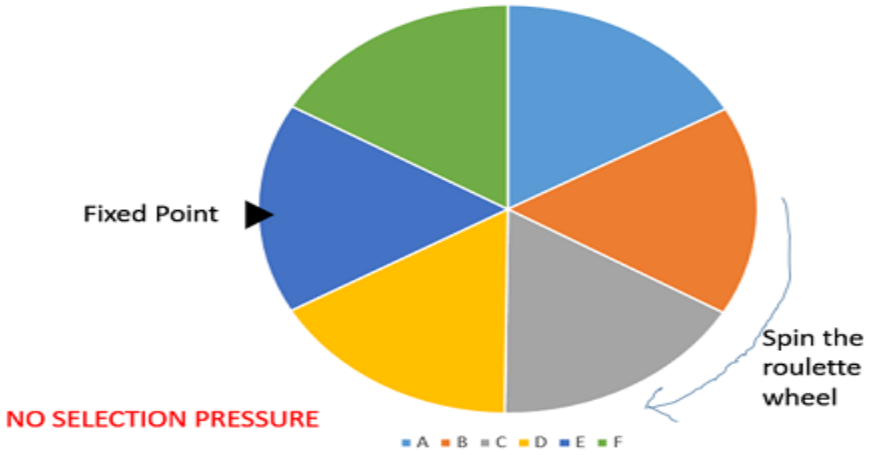
In K-Way tournament selection, we select K individuals from the population at random and select the best out of these to become a parent. The same process is repeated for selecting the next parent. Tournament Selection is also extremely popular in literature as it can even work with negative fitness values.



Rank Selection

Rank Selection also works with negative fitness values and is mostly used when the individuals in the population have very close fitness values (this happens usually at the end of the run).

This leads to each individual having an almost equal share of the pie (like in case of fitness proportionate selection) as shown in the following image and hence each individual no matter how fit relative to each other has an approximately same probability of getting selected as a parent. This in turn leads to a loss in the selection pressure towards fitter individuals, making the GA to make poor parent selections in such situations.



Chromosome	Fitness Value
A	8.1
B	8.0
C	8.05
D	7.95
E	8.02
F	7.99

In this, we remove the concept of a fitness value while selecting a parent. However, every individual in the population is ranked according to their fitness. The selection of the parents depends on the rank of each individual and not the fitness. The higher ranked individuals are preferred more than the lower ranked ones.

Chromosome	Fitness Value	Rank
A	8.1	1
B	8.0	4
C	8.05	2
D	7.95	6
E	8.02	3
F	7.99	5

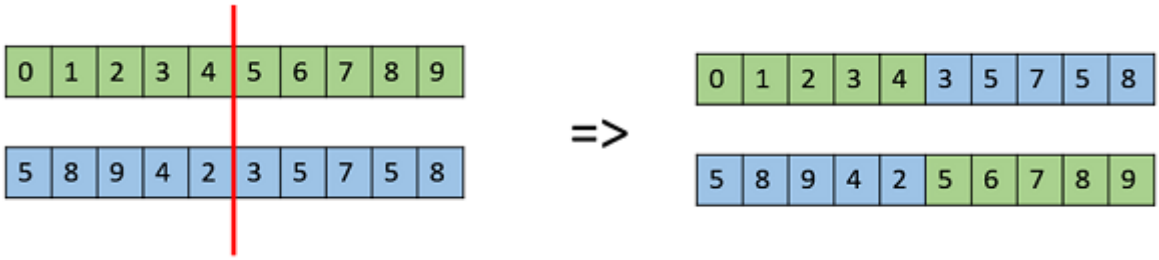
Random Selection

In this strategy we randomly select parents from the existing population. There is no selection pressure towards fitter individuals and therefore this strategy is usually avoided.

2) Crossover Operators

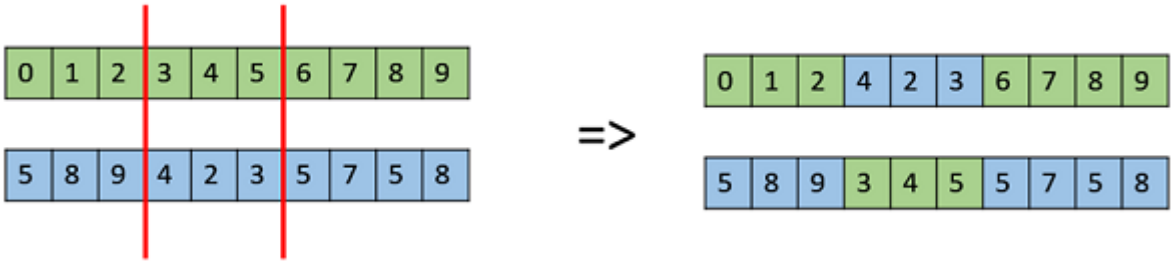
One Point Crossover

In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.



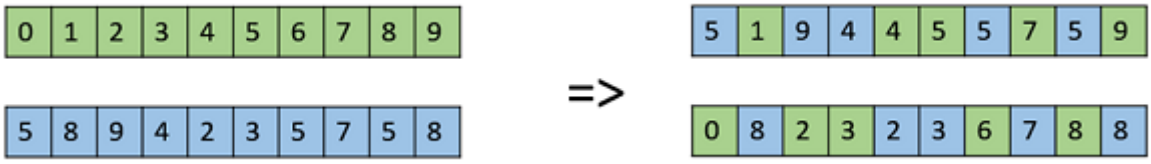
Multi Point Crossover

Multi point crossover is a generalization of the one-point crossover wherein alternating segments are swapped to get new off-springs.



Uniform Crossover

In a uniform crossover, we don't divide the chromosome into segments, rather we treat each gene separately. In this, we essentially flip a coin for each chromosome to decide whether or not it'll be included in the off-spring. We can also bias the coin to one parent, to have more genetic material in the child from that parent.

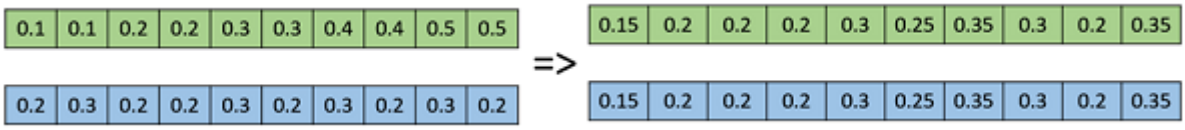


Whole Arithmetic Recombination

This is commonly used for integer representations and works by taking the weighted average of the two parents by using the following formulas,

- $Child1 = \alpha.x_1 + (1 - \alpha).x_2$
- $Child2 = \alpha.x_2 + (1 - \alpha).x_1$

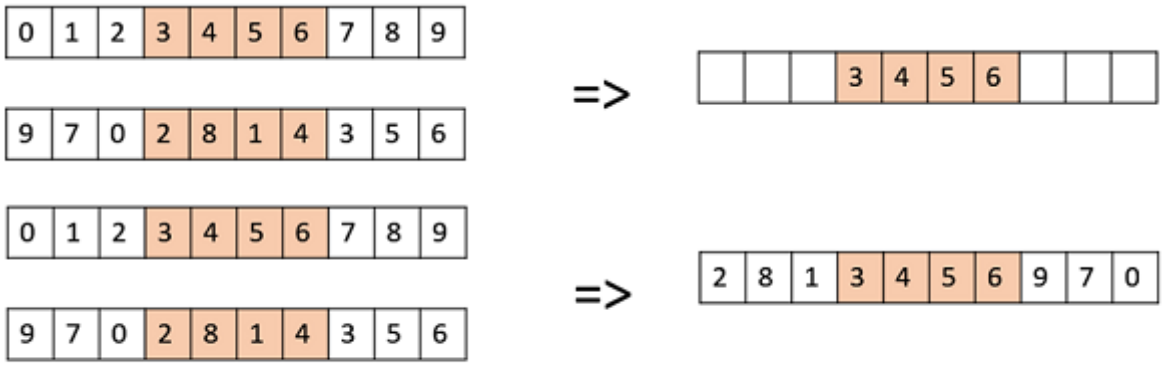
Obviously, if $\alpha = 0.5$, then both the children will be identical as shown in the following image.



Davis's Order Crossover (OX1)

OX1 is used for permutation-based crossovers with the intention of transmitting information about relative ordering to the off-springs. It works as follows −

- Create two random crossover points in the parent and copy the segment between them from the first parent to the first offspring.
- Now, starting from the second crossover point in the second parent, copy the remaining unused numbers from the second parent to the first child, wrapping around the list.
- Repeat for the second child with the parent's role reversed.

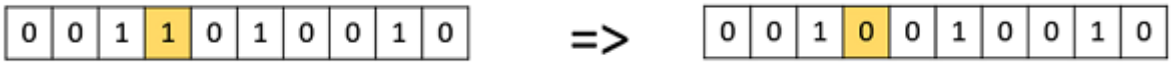


Repeat the same procedure to get the second child

3) Mutation Operators

Bit Flip Mutation

In this bit flip mutation, we select one or more random bits and flip them. This is used for binary encoded GAs.

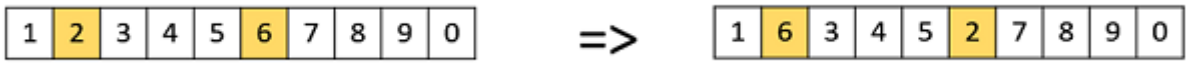


Random Resetting

Random Resetting is an extension of the bit flip for the integer representation. In this, a random value from the set of permissible values is assigned to a randomly chosen gene.

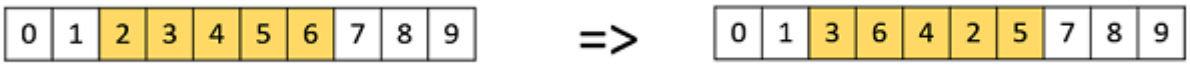
Swap Mutation

In swap mutation, we select two positions on the chromosome at random, and interchange the values. This is common in permutation-based encodings.



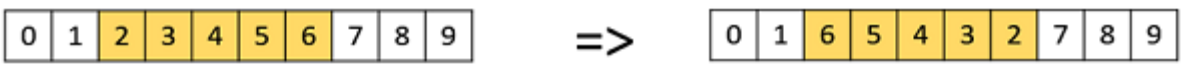
Scramble Mutation

Scramble mutation is also popular with permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.



Inversion Mutation

In inversion mutation, we select a subset of genes like in scramble mutation, but instead of shuffling the subset, we merely invert the entire string in the subset.



Q4

Apply GA based approach to solve an instance of Travelling Salesman problem.

Problem:

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

Pseudo Code:

1) Initialize the population randomly.
2) Determine the fitness of the chromosome.
3) Until done repeat:
    a) Select parents.
    b) Perform crossover and mutation.
    c) Calculate the fitness of the new population.
    d) Append it to the gene pool.

Trace:

Input Dataset,

	0	1	2	3	4
0	0	2	inf	12	5
1	2	0	4	8	Inf
2	inf	4	0	3	3
3	12	8	3	0	10
4	5	inf	3	10	0

Let encoding be in such a way that 012340 means he is travel travelled from 0 to 1 to 2 to 3 to 4 to 0.

And the cost of this path will be fitness. Let the Population size be 10. Initial Temperature be 10000. And before mutating each generation, lets sort based on fitness in ascending order. We are going to run this algo for 5 generations.

New temperature = 0.9 ∗ (Old temperature)

Initial Generation,

For our mutation, let's take a 2 random position of a gene in the current population and swap them.

This new gene will only get accepted if its fitness is less than its parents, or if its Boltzmann probability is greater than 0.5.

So, our current gnome is 043210 with fitness score of 24. The first mutated gene is 043120 with fitness score of infinity.

Clearly, the fitness of the child is greater than the parent, therefore we need to calculate the probability. By formula, we get prob = 0.

Therefore, we reject this child and calculate another, the next mutated child gnome is 013240 with fitness score of 21. As the child is fitter than the parent, we send them to the next gen.

Similarly do for the rest, to show the use of probability, So, our current gnome is 012340 with fitness score of 24. The first mutated gene is 012430 with fitness score of 31.

Clearly, the fitness of the child is greater than the parent, therefore we need to calculate the probability. By formula, we get prob = 0.999305.

Therefore, we accept this child into the next generation. This way, we get,

Gen 1,

T_{new} = 9000 - Gen 2, T_{new} = 8100 - Gen 3,

T_{new} = 7290 - Gen 4, T_{new} = 6561 - Gen 5,

It's clear from the 5th generation that the minimum distance is 21 and 034210 can be one of the possible solutions.

Q5

Understand the working of SLIQ and ARBC classifiers. Give a pseudocode and illustrate the same over a sample dataset of your choice.

1) SLIQ Classifier

Introduction:

Supervised Learning in Quest is a decision tree classifier that can handle both numeric and categorical attributes. SLIQ uses a pre-sorting technique in the tree-growth phase to reduce the cost of evaluating numeric attributes. This sorting procedure is integrated with a breadth-first tree growing strategy to enable SLIQ to classify disk-resident datasets. In addition, SLIQ uses a fast sub setting algorithm for determining splits for categorical attributes. SLIQ also uses a new tree-pruning algorithm based on the Minimum Description Length principle. This algorithm is inexpensive, and results in compact and accurate trees. The combination of these techniques enables SLIQ to scale for large data sets and classify data sets with a large number of classes, attributes, and examples.

Pseudo Code:

Key Features,

a) Tree Classifier, handling numeric and categoric attributes
b) Presorting numeric attributes before tree has been built
c) Breadth first growing strategy
d) Goodness test – Gini Index
e) Inexpensive tree pruning algorithm based on Minimum Description Length (MDL)

In Presorting,

a) Eliminate need for sorting data at each node
b) Create sorted list for each numeric attribute
c) Create class list

Split Evaluation,

EvaluateSplits()

For each attribute A do

Traverse attribute list of A

For each value v in attribute list do

Find the corresponding entry in the class list, and hence the corresponding class and the leaf node l

Update the class histogram in leaf l

If A is a numeric attribute then

Compute splitting index for test (A <= v) for l

If A is a categorical attribute then

For each leaf of the tree do

Find subset of A with best split

Update Class List,

UpdateLabels()

For each attribute A used in a split do

Traverse attribute list of A

For each value v in attribute list do

Find the corresponding entry in the class list e

Find the new class c to which v belongs by applying the splitting test at node referenced

from e

Update the class label for e to c

Update node referenced in e to the child corresponding to the class c

Trace:

Training Dataset,

Pre-Sorting and Breadth-First Growth,

For numeric attributes, sorting time is the dominant factor when finding the best split at a decision tree node. Therefore, the first technique used in SLIQ is to implement a scheme that eliminates the need to sort the data at each node of the decision tree. Instead, the training data are sorted just once for each numeric attribute at the beginning of the tree growth phase. To achieve this pre-sorting, we use the following data structures. We create a separate list for each attribute of the training data. Additionally, a separate list, called class list, is created for the class labels attached to the examples. An entry in an attribute list has two fields: one contains an attribute value, the other an index into the class list. An entry of the class list also has two fields: one contains a class label, the other a reference to a leaf node of the decision tree. The i^{th} entry of the class list corresponds to the i^{th} example in the training data. Each leaf node of the decision tree represents a partition of the training data, the partition being defined by the conjunction of the predicates on the path from the node to the root. Thus, the class list can at any time identify the partition to which an example belongs. We assume that there is enough memory to keep the class list memory-resident. Attribute lists are written to disk if necessary. Initially, the leaf reference fields of all the entries of the class list are set to point to the root of the decision tree. Then a pass is made

over the training data, distributing values of the attributes for each example across all the lists. Each attribute value is also tagged with the corresponding class list index. The attribute lists for the numeric features are then sorted independently.

After Presorting,

Processing Node Splits,

Rather than using a depth-first strategy used in the earlier decision-tree classifiers, we grow trees breadth-first. Consequently, splits for all the leaves of the current tree are simultaneously evaluated in one pass over the data. Figure 4 gives a schematic of the evaluation process. To compute the gini splitting-index for an attribute at a node, we need the frequency distribution of class values in the data partition corresponding to the node. The distribution is accumulated in a class histogram attached with each leaf node. For a numeric attribute, the histogram is a list of pairs of the form ¡class, frequency¿. For a categorical attribute, this histogram is a list of triples of the form ¡attribute value, class, frequency¿. Attribute lists are processed one at a time (recall that the attribute lists can be on disk). For each value IJ in the attribute list for the current attribute A, we find the the corresponding entry in the class list, which yields the corresponding class and the leaf node. We now update the histogram attached with this leaf node. If A is a numeric attribute, we compute at the same time the splitting index for the test A 2 v for this leaf. If A is a categorical attribute, we wait till the attribute list has been completely scanned and then find the subset of A with the best split. Thus, in one traversal of an attribute list, the best split using this attribute is known for all the leaf nodes. Similarly, with one traversal of all of the attribute lists, the best overall split for all of the leaf nodes is known. The best split test is saved with each of the leaf nodes. For our example

This figure illustrates the evaluation of splits on the salary attribute for the second level of the decision tree. The example assumes that the data has been initially split on the age attribute using the split age 2 35. The class histograms reflect the distribution of the points at each leaf node as a result of the split. The L values represent the distributions for examples that satisfy the test and R values represent examples that do not satisfy the test. We show how the class histograms are updated as each split is evaluated. The first value in the salary list belongs to node N2. So the first split evaluated is (salary $\leq$ 15) for N2. After this split, the corresponding example (salary 15, class index 2) which satisfies the predicate belongs to the left branch and the rest belong to the right branch. The class histogram of node N2 is updated to reflect this fact. Next, the split (salary 5 40) is evaluated for node N3. After the split, the corresponding example (salary 40, class index 4) belongs to the left branch and the class histogram of node N3 is updated to reflect this fact.

Updating the class list,

The next step is to create child nodes for each of the leaf nodes and update the class list.

Class list updating,

As an illustration, above figure shows the class list being updated after the nodes N2

and N3 have been split on the salary attribute. The salary attribute list is being traversed and the class list entry (entry 4) corresponding to the salary value of 40 is being updated. First, the leaf reference in the entry 4 of class list is used to find the node to which the example used to belong (N3 in this case). Then, the split selected at N3 is applied to find the new child to which the example belongs (N6 in this case). The leaf reference field of entry 4 in the class list is updated to reflect the new value. This is how we create the decision tree.

2) ARBC Classifier

Introduction:

Association rule mining was introduced as a way to find associative patterns from market basket data. The market basket data consist of transactions where a transaction is a set of items purchased by a customer. The motivation for applying this data mining approach on market basket data was to learn about buying patterns and use that information in catalog design, and store layout design. Since then, association rule mining has been studied and applied in many other domains (e.g. credit card fraud, network intrusion detection, genetic data analysis). In every domain, there is a need to analyze data to identify patterns associating different attributes. Association rule mining addresses this need. Many association rule mining algorithms have been proposed in the data mining literature. Apriori and FP-growth are two of them.

Pseudo Code:

Begin

minConfidence

rules = []

freqItemsets = []

support = UpperBoundSupport

while (support LowerBoundSupport AND rules.size < minNumberOfRules) do

L1 = { 1 – item itemsets}

For (k=2; Lk1 6=) do

Ck = generateCandidates (L k 1)

Lk = evaluateCandidates (Ck)

freqItemsets L(k)

end for

maxFreqItemsets = genMaxFreqItemset (freqItemsets)

rules = GenerateAllRules(maxFreqItemsets, minCondfidence)

support = support – delta

freqItemsets = []

end while

R = rules

R = sort(R)

For each rule r ∈ R in sequence do

Temp = ∅

For each instance d ∈ D do

If d satisfies the conditions of r then

Store d.id in temp and mark r if it correctly classifies d

End if

End for

Find the first rule p in C such that C_p, the rules in C up to p, has lowest number of errors and drop all the rules

Add the default class associated with p to the end of C, and return C

End

Trace:

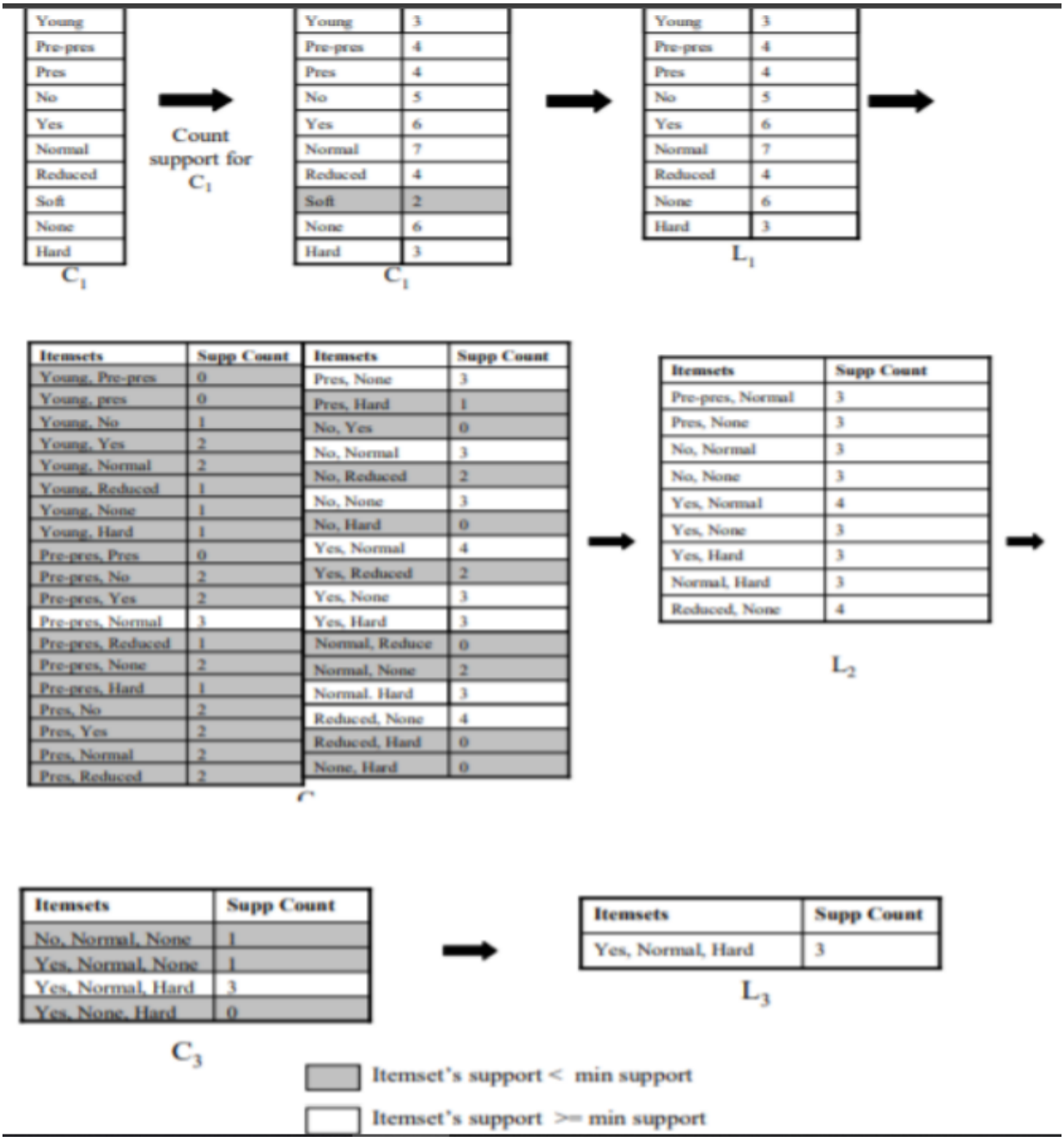
Training Dataset

age	astigmatism	tear-prod-rate	contact-lenses
young	no	normal	soft
young	yes	reduced	none
young	yes	normal	hard
pre-presbyopic	no	reduced	none
pre-presbyopic	no	normal	soft
pre-presbyopic	yes	normal	hard
pre-presbyopic	yes	normal	none
presbyopic	no	reduced	none
presbyopic	no	normal	none
presbyopic	yes	reduced	none
presbyopic	yes	normal	hard

First, Create Associate Rules,

For this, we can use any associate rule mining algorithm to mine the classification rule. In associative classification, the focus is to produce association rules that have only a particular attribute in the consequent. These association rules produced are called class association rules (CARs). Associative classification differs from general association rule mining by introducing a constraint as to the attribute that must appear on the consequent of the rule. The CBA-RG algorithm is an extension of the Apriori algorithm. The goal of this algorithm is to find all rule items of the form $<$ condset, y $>$ where condset is a set of items, and y $\in$ Y, where Y is the set of class labels. The support count of the rule item is the number of instances in the data set D that contain the condset and are labeled with y. Each rule item corresponds to a rule of the form: condset $\rightarrow$ y.

Applying Apriori with support 3,



Classifying based on rules,

Rule items that have support greater than or equal to minsup are called frequent rule items, while the others are called infrequent rule items. For all rule items that have the same condset, the one with the highest confidence is selected as the representative of those rule items. The confidence of rule items is calculated to determine if the rule item meets minconf. The set of rules that is selected after checking for support and confidence is called the classification association rules (CARs). Given a model and a new instance whose class is

unknown, the problem of predicting the instance's class using the model is an interesting problem. There is more than one way to use the model to predict the instance's class. In association rule-based classification models, rules in the model are ordered as follows:

- if rule ri has greater confidence than rj, then ri precedes rj, or
- if ri has the same confidence as rj, then the rule with greater support precede the other, or
- if ri has the same support and the same confidence as rj, then the rule with then smaller number of items in the antecedent will precede, or
- if ri has the same support, confidence and antecedent size as rj, then the order between the two rules is random.

Rules of high confidence are thought to be good for classification. Confidence alone may not make a rule very good. For instance, a rule from an instance that appears only once (high confidence but low support) may not be a good rule for classification. Rules with very high confidence and low support are useful in identifying rare events. After Applying apriori, the rules we get are,

Apriori rules with confidence 0.5,

Tear-prod-rate = reduced ➔ contact-lenses = none [Conf: 1.0, Sup: 0.36]

Contact-lenses = none ➔ tear-prod-rate = reduced [Conf: 0.67, Sup: 0.36]

Astigmatism = yes ➔ tear-prod-rate = normal [Conf: 0.67, Sup: 0.36]

Tear-prod-rate = normal ➔ astigmatism = yes [Conf: 0.57, Sup: 0.36]

Q6

Define sequence pattern mining and understand the working of

i) GSP

ii) Prefix-Span algorithms.

Give a pseudocode and illustrate the same over a sample dataset of your choice.

Sequence Pattern Mining

Introduction:

Sequential Pattern Mining is the mining of frequently occurring ordered events or subsequences as pattern in sequence database.

A Sequence Database stores a number of records, where all records are sequences of ordered events, with or without concrete notions of time

Sequential Patterns are used for targeted marketing and customer retention

1) GSP

Introduction:

The Generalized Sequence Pattern algorithm was created from a simpler algorithm for mining sequences, but it has some extra bells and whistles added so it can be more flexible for different situations.

Pseudo Code:

F_1 = the set of frequent 1-sequence

k=2,

do while Fk-1 != Null;

Generate candidate sets C_k (set of candidate k-sequences);

For all input sequences s in the database D

do

Increment count of all a in C_k if s supports a

End do

F_k = { a $\in$ C_k such that its frequency exceeds the threshold}

k = k+1;

End do

Result = Set of all frequent sequences is the union of all F_k's

Trace:

Input Data

Transaction ID	Customer ID	Items
1	1	A
2	1	B
3	1	A
4	2	B
5	2	A
6	2	B

First we prune Items with less support than threshold

Let threshold = 1

A – 3 > 1

B – 3 > 1

Now, get the sequences for each customer

Customer ID	Sequence
1	ABA
2	BAB

L1 = { A, B}

We generate C2,

C2 = { AA, BB, AB, BA}

Supports are,

AA – 1 – (A)B(A) in CID 1

BB – 1 – (B)A(B) in CID 2

AB – 2 – (AB)A and B(AB) in CID 1 and 2

BA – 2 – A(BA) and (BA)B in CID 1 and 2

All > 1 (Threshold)

L2 = { AA, AB, BA, BB}

We generate C3,

C3 = { AAA, AAB, ABA, ABB, BAA, BAB, BBA, BBB}

Supports are,

AAA – 0

AAB – 0

ABA – 1 >= 1

ABB – 0

BAA – 0

BAB – 1 >= 1

BBA – 0

BBB – 0

So, L3 = { ABA, BAB}

So, frequent sequences are { A, B, AA, AB, BA, BB, ABA, BAB}

2) Prefix Span Algorithm

Introduction:

A pattern-growth method based on projection is used in Prefix Span algorithm for mining sequential patterns. The basic idea behind this method is, rather than projecting sequence databases by evaluating the frequent occurrences of sub-sequences, the projection is made on frequent prefix. This helps to reduce the processing time which ultimately increases the algorithm efficiency.

Pseudo Code:

Input: A sequence database S, and the minimum support threshold min_sup

Output: The complete set of sequential patterns

Parameters:

1) α : sequential pattern,
2) l: the length of α ;
3) S| α : the α -projected database, if $\alpha \neq <>$; otherwise; the sequence database S

Algorithm

PrefixSpan(α , l, S| α)

1) Scan S| α once, find the set of frequent items b such that:

a) b can be assembled to the last element of α to form a sequential pattern; or

b) can be appended to α to form a sequential pattern.

2) For each frequent item b, append it to α to form a sequential pattern α ', and output α ';

3) For each α ', construct α '-projected database S| α ', and call PrefixSpan(α ', l+1, S| α ').

Trace:

Let minsup be 2,

Dataset,

Sequence_id	Sequence
10	$\langle a(abc)(ac)d(cf)\rangle$
20	$\langle (ad)c(bc)(ae)\rangle$
30	$\langle (ef)(ab)(df)cb\rangle$
40	$\langle eg(af)cbc\rangle$

Find length -1 sequential patterns,

Scan the database once to find all frequent items in sequences. Each of these frequent items is a length-1 sequential pattern. They are:

< a >: 4, < b >: 4, < c >: 4, < d >: 3, < e >: 3, < f >: 3

where < prefix >: count.

Divide search space

The complete set of sequential patterns can be partitioned into the following six subsets according to the six prefixes: (1) the ones having prefix < a > ; ... ; and (6) the ones having prefix < f >.

Find subsets of sequential patterns

The subsets of sequential patterns can be mined by constructing corresponding projected databases and mine each recursively. The projected databases as well as sequential patterns found in them are listed in Table 2, while the mining process is explained as follows, First, let us find sequential patterns having prefix < a >. Only the sequences containing < a > should be collected.ted. Moreover, in a sequence containing < a >, only the subsequence

prefixed with the first occurrence of < a >, should be considered. For example, in sequence < (ef)(ab)(df)cb > only the subsequence < (b)(df)cb > should be considered for mining sequential patterns having prefix < a >. Notice that (b) means that the last element in the prefix, which is a, together with b, form one element. As another example, only the subsequence < (abc)(ac)d(cf) > of sequence < a(abc)(ac)d(cf) > is considered. For our example,

Sequences in S containing < a > are projected wrt < a > to form the < a > - projected database, which consists of four postfix sequences : < (abc)(ac)d(cf) >, < (d)c(bc)(ae) >, < (b)(df)cb > and < (f)cbc >. By scanning < a >-projected database once, all the lenght-2 sequential patterns having prefix < a > can be found.

They are: < aa >: 2, < ab >: 4, < (ab) >: 2, < ac >: 4, < ad >: 2, < af >: 2.

Recursively, all sequential having patterns prefix < a > can be partitioned into 6 subsets: (1) those having prefix < aa >, (2) those having prefix < ab >, . . ., and finally, (6) those having prefix < af >. These subsets can be mined by constructing respective projected databases and mining each recursively.

The < aa > -projected database consists of only one non-empty (postfix) subsequences having prefix < aa > : < (bc)(ac)d(cf) >. Since there is no hope to generate any frequent subsequence from a single sequence, the processing of < aa >-projected database terminates.

The < ab > -projected database consists of three postfix sequences: < (c)(ac)d(cf) >, < (c)a > and < c >.

Recursively mining < ab > -projected database returns fours sequential patterns: < (c) >, < (c)a >, < a > and < c > (i.e. < a(bc) >, < a(bc)a >, < aba > and < abc >).

The < (ab) > projected sequence only consist of two sequence, < (c)(ac)d(cf) > and < (df)c >, which leads to the finding of the following sequential patterns having prefix < (ab) > : < c >, < d >, < f > and < dc >.

The < ac > − , < ad > − and < af > − projected databases can be constructed and recursively mined similarly. The sequential patterns found are shown in figure.

Similarly, we can find sequential patterns having prefix < b >, < c >, < d >, < e > and < f > by constructing < b, > − , < c > − , < d > − , < e > − and < f > − projected databases and mining them respectively. The projected databases are shown in figure.

Projected Databases and Sequential Patterns,

Prefix	Projected (postfix) database	Sequential patterns
$\langle a \rangle$	$\langle (abc)(ac)d(cf) \rangle$, $\langle (\_d)c(bc)(ae) \rangle$, $\langle (\_b)(df)cb \rangle$, $\langle (\_f)cbc \rangle$	$\langle a \rangle$, $\langle aa \rangle$, $\langle ab \rangle$, $\langle a(bc) \rangle$, $\langle a(bc)a \rangle$, $\langle aba \rangle$, $\langle abc \rangle$, $\langle (ab) \rangle$, $\langle (ab)c \rangle$, $\langle (ab)d \rangle$, $\langle (ab)f \rangle$, $\langle (ab)dc \rangle$, $\langle ac \rangle$, $\langle aca \rangle$, $\langle acb \rangle$, $\langle acc \rangle$, $\langle ad \rangle$, $\langle adc \rangle$, $\langle af \rangle$
$\langle b \rangle$	$\langle (\_c)(ac)d(cf) \rangle$, $\langle (\_c)(ae) \rangle$, $\langle (df)cb \rangle$, $\langle c \rangle$	$\langle b \rangle$, $\langle ba \rangle$, $\langle bc \rangle$, $\langle (bc) \rangle$, $\langle (bc)a \rangle$, $\langle bd \rangle$, $\langle bdc \rangle$, $\langle bf \rangle$
$\langle c \rangle$	$\langle (ac)d(cf) \rangle$, $\langle (bc)(ae) \rangle$, $\langle b \rangle$, $\langle bc \rangle$	$\langle c \rangle$, $\langle ca \rangle$, $\langle cb \rangle$, $\langle cc \rangle$
$\langle d \rangle$	$\langle (cf) \rangle$, $\langle c(bc)(ae) \rangle$, $\langle (\_f)cb \rangle$	$\langle d \rangle$, $\langle db \rangle$, $\langle dc \rangle$, $\langle dcb \rangle$
$\langle e \rangle$	$\langle (\_f)(ab)(df)cb \rangle$, $\langle (af)cbc \rangle$	$\langle e \rangle$, $\langle ea \rangle$, $\langle eab \rangle$, $\langle eac \rangle$, $\langle eacb \rangle$, $\langle eb \rangle$, $\langle ebc \rangle$, $\langle ec \rangle$, $\langle ecb \rangle$, $\langle ef \rangle$, $\langle efb \rangle$, $\langle efc \rangle$, $\langle efcb \rangle$.
$\langle f \rangle$	$\langle (ab)(df)cb \rangle$, $\langle cbc \rangle$	$\langle f \rangle$, $\langle fb \rangle$, $\langle fbc \rangle$, $\langle fc \rangle$, $\langle fcb \rangle$

Q7

Understand the schemata theorem of GA (relevance of selection/crossover/mutation with the 3 components of the schema theorem. Apply the theorem to the optimization function f(x)= x3-2x2+x. Check whether the empirical results obtained earlier in assignment-I matches with the theorem.

Introduction:

Using the established methods and genetic operators of genetic algorithms, the schema theorem states that short, low-order schemata with above-average fitness increase exponentially in successive generations. Expressed as an equation:

$$E\left(m\left(H, t+1\right)\right) \geq \frac{m\left(\mathrm{H},\ \mathrm{t}\right) f\left(H\right)}{a_t}\left[1-p\right]$$

Here $m\left(\mathrm{H},\ \mathrm{t}\right)$ { \ displaystyle m(H,t)} is the number of strings belonging to schema { \ displaystyle H} H at generation { \ displaystyle t} t, { \ displaystyle f(H)} $f\left(H\right)$ is the *observed* average fitness of schema { \ displaystyle H} H and { \ displaystyle a_{ t} } a_t is the *observed* average fitness at generation { \ displaystyle t} t. The probability of disruption { \ displaystyle p} p is the probability that crossover or mutation will destroy the schema { \ displaystyle H} H. It can be expressed as:

$$p = \frac{\delta\left(H\right)}{l-1} p_c + o\left(H\right) p_m$$

{ \ displaystyle p={ \ delta (H) \ over l-1} p_{ c} +o(H)p_{ m} } where { \ displaystyle o(H)} $o\left(H\right)$ is the order of the schema, { \ displaystyle l} l is the length of the code, { \ displaystyle p_{ m} } p_m is the probability of mutation and { \ displaystyle p_{ c} } p_c is the probability of crossover. So, a schema with a shorter defining length { \ displaystyle \ delta (H)} $\delta\left(H\right)$ is less likely to be disrupted.

Implications of Schema Theorem:

Schemas, like families, need nourishment and encouragement and careful protective management

1) The more bits in your building block family the more likely one is to go off the rails (causing great frustration and heartache).
2) Genes living far apart are prone to breaking up.
3) Conducting a constructive relationship at a distance is hard.
4) Best results achieved by the family unit huddled together in consecutive positions.

Applications of Schema Theorem:

The schema theorem is more applicable at the early stages of a search rather than at the end. Schema theorem indicates that fitter than average schemas are rewarded. The fitter the schema the more it is rewarded.

1) Reward is immediate: you see it in the next generation
2) Should alert us to one danger immediately
3) Premature converge of the population

Trace:

The optimization function is,

$$x^3 - 2x^2 + x$$

The constraint on x is

$$x \in [0, 31]$$

Choose Encoding,

As x max is 31, we use 5-bit binary representation of x

Generate Schema,

The randomly generated schemas are:

['01∗ 0∗ ', '000∗ 0', '0∗ 1∗ ∗ ', '∗ 0∗ 00', '10∗ 11', '01∗ ∗ 1']

Calculate the average fitness of each schema,

The average fitness of a schema is the average of all the instances of that schema. In our case, the fitness of each schema is:

[1073.0, 2147.0, 1680.25, 6074.5, 20793.0, 12018.5]

Select schema for crossover,

Schemas are selected in Russian roulette fashion with probabilities of getting selected being defined as:

$$p\left(x_i\right) = \frac{F\left(x_i\right)}{\sum_{j=1}^{N} F\left(x_i\right)}$$

In our case, the probability of each schema getting selected are:

[0.025, 0.049, 0.038, 0.139, 0.475, 0.274]

And the schema chosen for crossover are:

['10∗ 11' '000∗ 0' '10∗ 11' '10∗ 11' '10∗ 11' '10∗ 11']

Crossover,

Randomly pair up the schemas selected for crossover: In our case:

10∗ 11 and 10∗ 11

000∗ 0 and 10∗ 11

10∗ 11 and 10∗ 11

We are using single point crossover and the offspring generated are:

['01∗ 0∗ ', '00∗ 11', '10∗ 11', '100∗ 0', '10∗ 11', '01∗ ∗ 1']

Mutation,

For mutation, we define a probability of mutation, pm and we go through each gene of the chromosome and mutate them.

In our case, $p_m = 0.02$.

For schema bit with '∗ ', mutation makes no difference, so we may not see any mutation.

After mutation, our schemas are:

['01∗ 0∗ ', '00∗ 11', '10∗ 11', '100∗ 0', '10∗ 11', '01∗ ∗ 1']

Repeat,

Repeat Selection, Crossover and Mutation on the schemas until there is no improvement in the average fitness of the generation.

In our case, that is:

['11∗ ∗ 1', '01∗ 11', '10∗ ∗ 1', '10∗ 11', '00∗ 11', '01∗ ∗ 1']

Perform GA on the schema population,

Now that we have the best schemas, create a population with all the instances of all the schemas in the final generation and apply normal ga to this population.

In our case, after applying ga to the best schema instance population the optimal solution is: '11111'

Q8

Understand the working of linear and non-linear regression models. Give a pseudocode and illustrate the same over a sample dataset of your choice.

1) Linear Regression

Introduction:

Linear Regression is a statistical approach to modelling the relationship between an input and output as a linear relationship.

Given a data set $\{y_i,\ x_{i1},\ x_{i2}, \ldots, x_{ip}\}$ of n statistical units, a linear regression model assumes that the relationship between the dependent variable y and the p-vector of regressors x is linear. This relationship is modelled through a disturbance term or error variable ε -an unobserved random variable that adds 'noise' to the linear relationship between the dependent variable and regressors. Thus, the model takes the form,

$$y_i = \beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip} + \varepsilon_i$$

$$i = 1, \ldots, n$$

Where ^{T} denotes the transpose, so that $x_i^T \beta$ is the inner product between vectors x_i and β .

Pseudo Code:

a) Read Number of Data (n)

b) For i=1 to n:
   Read Xi and Yi
   Next i

c) Initialize:
   sumX = 0
   sumX2 = 0
   sumY = 0
   sumXY = 0

d) Calculate Required Sum
   For i=1 to n:
      sumX = sumX + Xi
      sumX2 = sumX2 + Xi ∗ Xi
      sumY = sumY + Yi
      sumXY = sumXY + Xi ∗ Yi
      Next i

e) Calculate Required Constant a and b of y = a + bx:
   b = (n ∗ sumXY - sumX ∗ sumY)/(n∗ sumX2 - sumX ∗ sumX)
   a = (sumY - b∗ sumX)/n

f) Display value of a and b

Trace:

For sample data,

Suppose actual line parameters are

Y = mX + c

Where m = slope = 5

C = intercept = 10

N = 5

X	Y
1	16
2	19
3	26
4	32
5	36

Now, we calculate sums,

sumX = 1 + 2 + 3 + 4 + 5 = 15

sumY = 16 + 19 + 26 + 32 + 36 = 129

sumX2 = 1 + 4 + 9 + 16 + 25 = 55

sumXY = 16 + 38 + 78 + 128 + 180 = 440

Now, we find parameters,

m = (N ∗ sumXY - sumX ∗ sumY)/(N∗ sumX2 - sumX ∗ sumX) = (5∗ 440 – 15∗ 129) / (5∗ 55 – 15∗ 15)

= 5.3

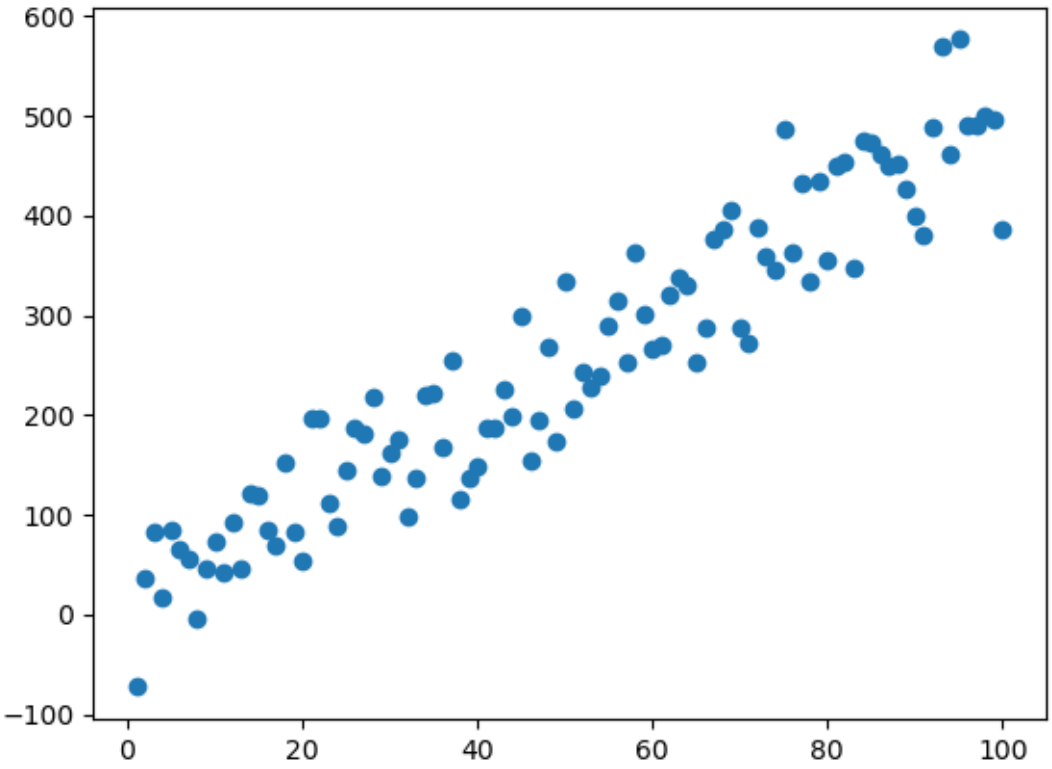
c = (sumY - m∗ sumX)/N = (129 – 5.3∗ 15) / 5 = 9.9

So, Evaluation,

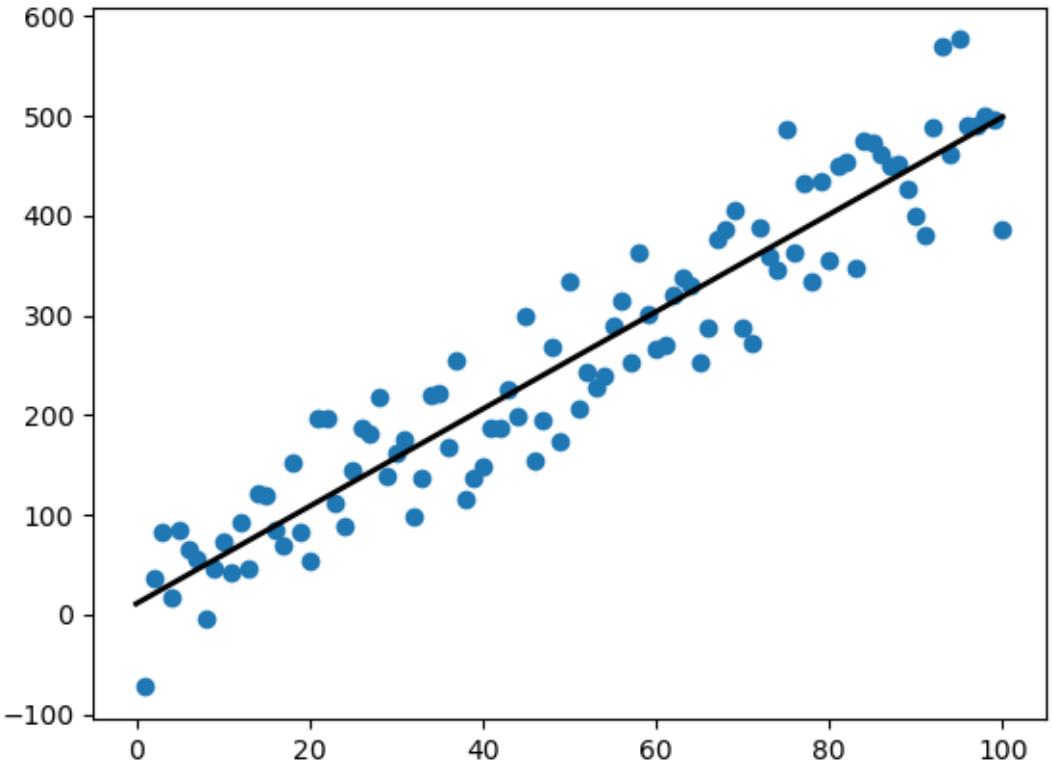
	Actual	Model
Slope	5	5.3
Intercept	10	9.9

Larger Dataset,

Input Data



Output Model



Parameters:

Slope: Actual: 5 - Predicted: 4.880934505955754

Intercept: Actual: 10 - Predicted: 10.804514757031138

Predictions:

X: 12.3 Actual Y: 71.5 - Predicted Y: 70.84000918028691

X: 10.1 Actual Y: 60.5 - Predicted Y: 60.10195326718424

X: 25.2 Actual Y: 136.0 - Predicted Y: 133.80406430711614

2) Logistic Regression

Introduction:

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, **logistic regression** (or **logit regression**) is estimating the parameters of a logistic model (a form of binary regression).

Pseudo Code:

Given Dataset, first we fit the logistic regression model to the train data.

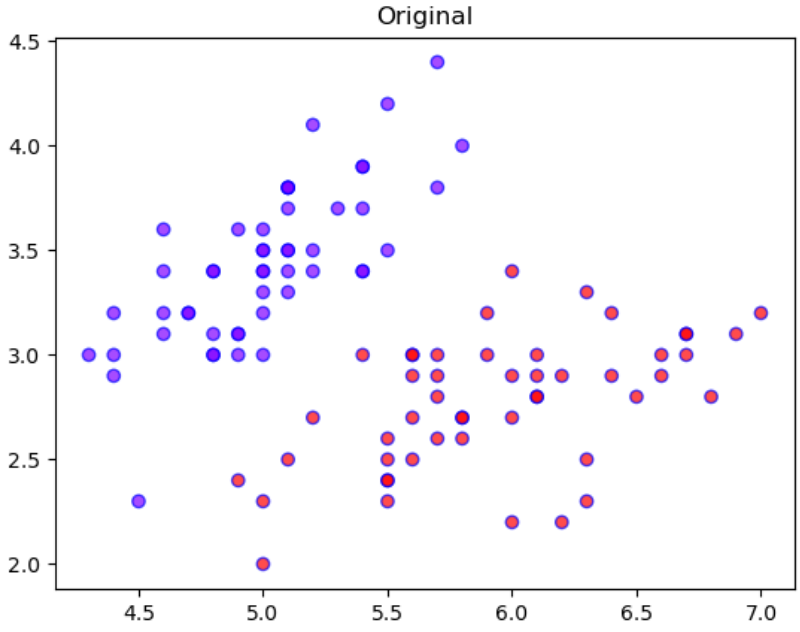
1) Initialize the parameters

2) Repeat {
   Make a prediction on y
   Calculate cost function
   Get gradient for cost function
   Update parameters
   }

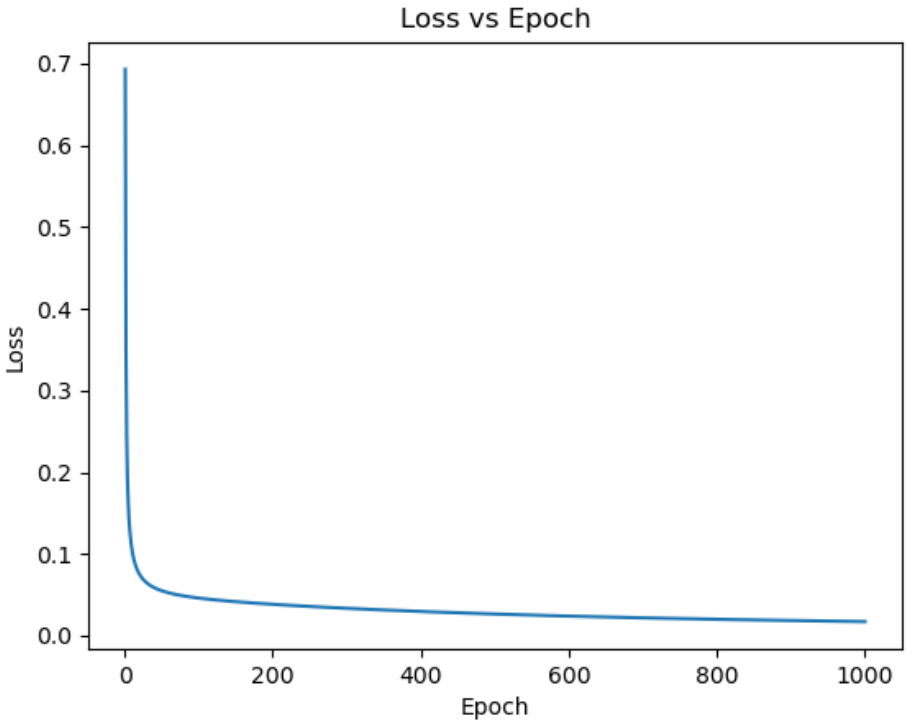
After training, we can use the model parameters to predict for new data points

Trace:

Input Data



Training/Fitting



Output Predictions