Analytics and Systems of Big Data

Exploratory Assignment

CED16I010

Kruttika Bhat

Contents

1 HITS

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

Working: Hyperlink Induced Topic Search (HITS) Algorithm is a Link Analysis Algorithm that rates webpages by providing a ranking relevant for a particular search. HITS uses hubs and authorities to define a recursive relationship between webpages.

- Authorities: Pages containing useful information. It is a page that many hubs link to.
- Hubs: Pages that link to authorities.

HITS identifies good authorities and hubs for a topic by assigning two numbers to a page: an authority and a hub weight/score. These weights are defined recursively.

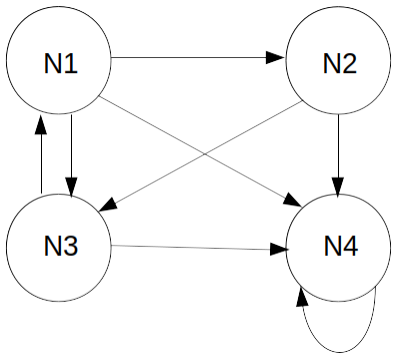
- Authority weight: It estimates the value of the content of the page. A higher authority weight occurs if the page is pointed to by pages with high hub weights.
- Hub weight: It estimates the value of its links to other pages. A higher hub weight occurs if the page points to many pages with high authority weights.

Pseudocode:

1. Let number of iterations be k. Each node is assigned a Hub score $h_i = 1$ and an Authority score $a_i = 1$.
2. Repeat k times:
   - Authority update : Each authority score will be the sum of hub scores of the nodes which are pointing to it.
   $$a_i = \sum_{j \to i} h_j$$
   - Hub update : Each hub score will be the sum of authority scores of the nodes it is pointing to.
   $$h_i = \sum_{i \to j} a_j$$
   - Normalise: Divide each Hub score by square root of the sum of the squares of all Hub scores. Do the same for Authority scores. (This step is optional.)
   $$h_i = \frac{h_i}{\sum_{j=1}^{n} h_j^2}$$

   Now, if you consider that the input is an adjacency matrix A. Then the formulas can be modified as:
   $$h_i = \sum_{i \to j} a_j \Leftrightarrow h_i = \sum_j A_{ij} a_j \Leftrightarrow h = Aa$$
   Similarly, $a = A^T h$

Trace: The input graph is shown below.



Let k=3 (maximum number of iterations). Consider that the algorithm receives the adjacency matrix as input. Initialise hub scores to 1.

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, A^T = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}, h = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

First iteration k=1:

1. Update authority scores.

   $a = A^T h$

   $$a = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 4 \end{bmatrix}$$

2. Update hub scores.

   $h = Aa$

   $$h = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 7 \\ 6 \\ 5 \\ 4 \end{bmatrix}$$

3. Normalization:

   $$a = \begin{bmatrix} \frac{1}{\sqrt{1^2+1^2+2^2+4^2}} \\ \frac{1}{\sqrt{1^2+1^2+2^2+4^2}} \\ \frac{2}{\sqrt{1^2+1^2+2^2+4^2}} \\ \frac{4}{\sqrt{1^2+1^2+2^2+4^2}} \end{bmatrix} = \begin{bmatrix} 0.213 \\ 0.213 \\ 0.426 \\ 0.853 \end{bmatrix}, h = \begin{bmatrix} \frac{7}{\sqrt{7^2+6^2+5^2+4^2}} \\ \frac{6}{\sqrt{7^2+6^2+5^2+4^2}} \\ \frac{5}{\sqrt{7^2+6^2+5^2+4^2}} \\ \frac{4}{\sqrt{7^2+6^2+5^2+4^2}} \end{bmatrix} = \begin{bmatrix} 0.623 \\ 0.535 \\ 0.445 \\ 0.356 \end{bmatrix}$$

Ranking: Authority=$\{4, 3, 2, 1\}$, Hub=$\{1, 2, 3, 4\}$

A rank is given to each node (page) based on it's score.

k=2: The scores before normalisation will be used to get the next set of scores. Normalisation helps to see if we have reached convergence.

1. Update authority scores.

   $a = A^T h$

   $$a = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 7 \\ 6 \\ 5 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 13 \\ 22 \end{bmatrix}$$

2. Update hub scores.

   $h = Aa$

   $$h = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 5 \\ 7 \\ 13 \\ 22 \end{bmatrix} = \begin{bmatrix} 42 \\ 35 \\ 27 \\ 22 \end{bmatrix}$$

3. Normalisation: $a = \begin{bmatrix} 0.185 \\ 0.260 \\ 0.482 \\ 0.816 \end{bmatrix}$, $h = \begin{bmatrix} 0.648 \\ 0.540 \\ 0.417 \\ 0.340 \end{bmatrix}$

Ranking: Authority=$\{4, 3, 2, 1\}$, Hub=$\{1, 2, 3, 4\}$

There was no change in ranking. Although the normalised values showed some change. So, we have not yet reached convergence.

k=3:

1. Update authority scores.

   $a = A^T h$

   $$a = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 42 \\ 35 \\ 27 \\ 22 \end{bmatrix} = \begin{bmatrix} 27 \\ 42 \\ 77 \\ 126 \end{bmatrix}$$

2. Update hub scores.

$h = Aa$

$$h = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 27 \\ 42 \\ 77 \\ 126 \end{bmatrix} = \begin{bmatrix} 245 \\ 203 \\ 153 \\ 126 \end{bmatrix}$$

3. Normalisation: $a = \begin{bmatrix} 0.173 \\ 0.269 \\ 0.494 \\ 0.808 \end{bmatrix}$, $h = \begin{bmatrix} 0.654 \\ 0.542 \\ 0.408 \\ 0.336 \end{bmatrix}$

Ranking: Authority=$\{4, 3, 2, 1\}$, Hub=$\{1, 2, 3, 4\}$

There was no change in ranking. The normalised values are also approximately the same, which means that we are closer to convergence.

The algorithm ends here as we had specified k=3. But if further iterations are done then it would be seen that the normalised values of the hub and authority scores converge and do not change after some point.

2 Clustering algorithms

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

2.1 BIRCH

BIRCH stands for Balanced Iterative Reducing and Clustering using Hierarchies. It is an unsupervised data mining system used to perform hierarchical clustering over big datasets. It integrates the hierarchical clustering (at the initial microclustering stage) and other clustering methods such as iterative partitioning (at the later macroclustering stage). It overcomes the two difficulties of agglomerative clustering methods i.e scalability and the inability to undo what was done in the previous step. BIRCH introduces two concepts which are used to summarise cluster representations:

- Clustering Feature (CF)
- Clustering Feature tree (CF tree)

Given n d-dimensional data objects or points in a cluster, we can define the centroid x_0 , radius R, and diameter D of the cluster as follows:

$$x_0 = \frac{\sum_{i=1}^{n} x_i}{n},\ R = \sqrt{\frac{\sum_{i=1}^{n}(x_i - x_0)^2}{n}},\ D = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2}{n(n-1)}}$$

where R is the average distance from member objects to the centroid and D is the average pairwise distance within a cluster.

Clustering Feature(CF): It is a three-dimensional vector summarising information about clusters of objects. Given n d-dimensional objects or points in a cluster, $\{x_i\}$, then the CF of the cluster is defined as:

$$CF =< n, LS, SS >$$

where,

- n is the number of points in the cluster
- LS is the linear sum of the n points i.e. $\sum_{i=1}^{n} x_i$
- SS is the square sum of the data points i.e. $\sum_{i=1}^{n} x_i^2$

Clustering features are sufficient for calculating all of the measurements that are needed for making clustering decisions in BIRCH. Clustering features are also additive. For example, suppose that we have two disjoint clusters, C_1 and C_2, having the clustering features, CF_1 and CF_2, respectively. The clustering feature for the cluster that is formed by merging C_1 and C_2 is simply $CF_1 + CF_2$.

CF Tree: It is a height-balanced tree that stores the clustering features for a hierarchical clustering. The non-leaf nodes store sums of the CFs of their children, and thus summarise clustering information about their children. A CF tree has two parameters:

- Branching factor (B): Specifies the maximum number of children per non-leaf node.
- Threshold (T): Specifies the maximum diameter of subclusters stored at the leaf nodes of the tree.

These two parameters influence the size of the resulting tree.

Pseudocode:

1. Phase 1: Initial clustering step. BIRCH scans the database to build an initial in-memory CF tree.
   (a) The CF tree is built dynamically as objects are inserted (incremental method).
   (b) An object is inserted into the closest leaf entry (subcluster).
   (c) If the diameter of the subcluster stored in the leaf node after insertion is larger than the threshold value, then the leaf node and possibly other nodes are split.

(d) After the insertion of the new object, information about it is passed toward the root of the tree.

(e) The size of the CF tree can be changed by modifying the threshold.

2. Phase 2 (optional): Condense into desirable range by building a smaller CF tree. The leaf entries in the initial CF tree are scanned. This will remove outliers and group crowded subclusters into larger ones, resulting in a smaller CF tree.

3. Phase 3: Global clustering step. BIRCH applies a (selected) clustering algorithm to cluster the leaf nodes of the CF tree, which removes sparse clusters as outliers and groups dense clusters into larger ones. Any clustering algorithm, such as a typical partitioning algorithm, can be used.

4. Phase 4 (optional): Cluster refining step. This involves additional passes over the data to correct minor and localised inaccuracies. Up to this point the original data has only been scanned once, although the tree information may have been scanned multiple times. Phase 4 uses the centroids of the clusters produced by phase 3 as seeds and redistributes the data points to its closest seed to obtain a set of new clusters.

Trace: Consider the sample dataset given below.

Number	1	2	3	4	5	6	7	8	9	10
x	5	3	7	3	3	4	4	5	7	8
y	1	4	3	8	6	7	5	5	5	5

Phase 1: Insert objects into CF tree. Let's initialise B=3 and T=(2,2) (arbitrarily chosen). Changing these parameters will result in a different tree.

1. Insert (5,1).
   $CF_1 =< 1, (5,1), (25,1) >$

2. Insert (3,4) to CF_1.
   $CF_1 =< 2, (8,5), (34,17) >$
   $x_0 = (4, 2.5)$, $R = (1, 1.5)$, $D = (2, 3)$
   The diameter exceeds the threshold, so the node is split.
   $CF_1 =< 1, (5,1), (25,1) >$, $CF_2 =< 1, (3,4), (9,16) >$

3. Insert (7,3) to CF_1.
   $CF_1 =< 2, (12,4), (74,10) >$
   $x_0 = (6, 2)$, $R = (1, 1)$, $D = (2, 2)$

4. Insert (3,8) to CF_1.

   $CF_1 =< 3, (15, 12), (83, 74) >$

   $x_0 = (5, 4)$, $R = (1.63, 2.94)$, $D = (2.82, 5.09)$

   The diameter exceeds the threshold T. Try inserting in CF_2

   $CF_2 =< 2, (6, 12), (18, 80) >$

   $x_0 = (3, 6)$, $R = (0, 2)$, $D = (0, 4)$

   The diameter exceeds the threshold T along 1 dimension. Split the node or increase the threshold. We will update the threshold to T=(2,4).

5. Insert (3,6) to CF_2, as it has less diameter than inserting in CF_1.

   $CF_2 =< 3, (9, 18), (27, 116) >$

   $x_0 = (3, 6)$, $R = (0, 1.63)$, $D = (0, 2.82)$

6. Insert (4,7) to CF_2, as inserting in CF_1 exceeds the threshold.

   $CF_2 =< 4, (13, 25), (43, 165) >$

   $x_0 = (3.25, 6.25)$, $R = (0.43, 1.47)$, $D = (0.70, 2.41)$

7. Insert (4,5) to CF_2, as it has less diameter than inserting in CF_1

   $CF_2 =< 5, (17, 30), (59, 190) >$

   $x_0 = (3.4, 6)$, $R = (0.48, 1.41)$, $D = (0.77, 2.23)$

8. Insert (5,5) to CF_2, as it has less diameter than inserting in CF_1

   $CF_2 =< 6, (22, 35), (84, 215) >$

   $x_0 = (3.66, 5.83)$, $R = (0.74, 1.34)$, $D = (1.15, 2.08)$

9. Insert (7,5) in CF_1, as inserting in CF_2 exceeds the threshold.

   $CF_1 =< 3, (19, 9), (123, 35) >$

   $x_0 = (6.33, 3)$, $R = (0.94, 1.63)$, $D = (1.63, 2.82)$

10. Insert (8,5) in CF_1, as inserting in CF_2 exceeds the threshold.

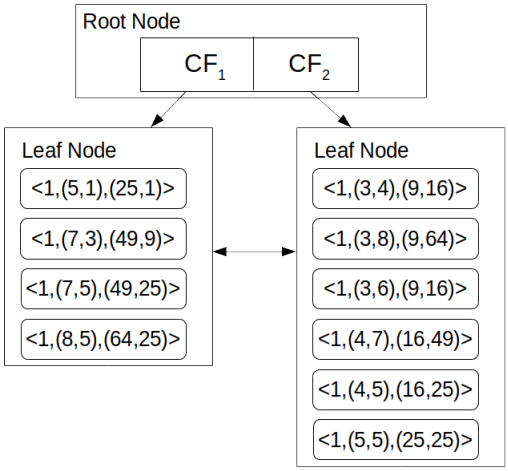
    $CF_1 =< 4, (27, 14), (187, 60) >$

    $x_0 = (6.75, 3.5)$, $R = (1.08, 1.65)$, $D = (1.77, 2.71)$

11. The next step would be to apply a clustering algorithm on the leaves. This would consider each sub cluster as a single point represented by the centroid of the n points belonging to the cluster (leaf node). So, an algorithm, like agglomerative hierarchical clustering, would be applied directly on these sub clusters, where each sub cluster is represented as a single point. Now, we only have 1 level in our tree and there are only 2 clustering features. So, we can't apply any further clustering as it would

be equivalent to clustering 2 points. For larger datasets and a bigger tree, this step would be used to perform global clustering.

12. The algorithm ends with 2 clusters. The final tree is shown below.



2.2 DBSCAN

Density Based Spatial Clustering of Applications with Noise is a clustering algorithm based on the intuitive notion of clusters and noise. The key idea is that for each point of a cluster, the neighbourhood of a given radius has to contain at least a minimum number of points. DBSCAN algorithm requires two parameters.

- Epsilon/eps/ϵ: It defines the neighbourhood around a data point i.e. if the distance between two points is less than or equal to eps then they are considered as neighbours. 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.
- Mininum points or MinPts: Minimum number of neighbours (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 $\geq$ D+1. The minimum value of MinPts is 3.

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

1. Core Point: A point is a core point if it has more than MinPts points within eps.
2. Border Point: A point which has fewer than MinPts within eps but it is in the neighbourhood of a core point.
3. Noise or outlier: A point which is not a core point or border point.

Pseudocode:

1. Find all the neighbour points within eps and identify the core points or visited with more than MinPts neighbours.
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.
4. Iterate through the remaining unvisited points in the dataset.
5. Those points that do not belong to any cluster are noise.

Trace: Consider the sample dataset given below.

Number	1	2	3	4	5	6	7	8	9	10
x	5	3	7	3	3	4	4	5	7	8
y	1	4	3	8	6	7	5	5	5	5

Let us define $\epsilon = 2$ (arbitrarily taken) and MinPts=3 (=D+1). Let us consider Euclidean distance as the distance measure.

1. Get the neighbourhood i.e all data points within epsilon distance, of each point. If there are at least MinPts then label it as a Core point, otherwise label it as Noise.

Sno	Point	Neighbourhood	Label
1	(5,1)	-	Noise
2	(3,4)	(3,6),(4,5)	Noise
3	(7,3)	(7,5)	Noise
4	(3,8)	(3,6),(4,7)	Noise
5	(3,6)	(3,4),(3,8),(4,7),(4,5)	Core
6	(4,7)	(3,8),(3,6),(4,5)	Core
7	(4,5)	(3,4),(3,6),(4,7),(5,5)	Core
8	(5,5)	(4,5),(7,5)	Noise
9	(7,5)	(7,3),(5,5),(8,5)	Core
10	(8,5)	(7,5)	Noise

2. Now visit each core point and assign it to a cluster if it has not been assigned. Recursively check the neighbourhood and update the labels of the points.

3. Consider (3,6). It does not belong to any cluster, so a new cluster is formed.
   $C_1 = \{(3,4), (3,8), (3,6), (4,7), (4,5)\}$
   Now, check the neighbourhood of all the points in the cluster. If it is labelled as a core point, then it remains unchanged. If it is labelled as noise then change the label to border point. Update the cluster when required.

   (a) (3,4): There are only 2 points in its neighbourhood so it is labelled as a border point and C_1 is not updated.

   (b) (3,8): There are only 2 points in its neighbourhood so it is labelled as a border point and C_1 is not updated.

   (c) (4,7): It is already a core point, so label does not change. It's neighbourhood does not give any new points so C_1 is not updated.

   (d) (4,5): It is already a core point, so label does not change. It's neighbourhood gives a new point, so C_1 is updated.
   $C_1 = \{(3,4), (3,8), (3,6), (4,7), (4,5), (5,5)\}$

   (e) (5,5): There are only 2 points in its neighbourhood so it is labelled as a border point. It gives us a new point, so C_1 is updated.
   $C_1 = \{(3,4), (3,8), (3,6), (4,7), (4,5), (5,5), (7,5)\}$

   (f) (7,5): It is already a core point, so label does not change. It's neighborhood gives new points, so C_1 is updated.
   $C_1 = \{(3,4), (3,8), (3,6), (4,7), (4,5), (5,5), (7,5), (7,3), (8,5)\}$

   (g) (7,3): There is only 1 point in its neighbourhood so it is labelled as a border point and C_1 is not updated.

   (h) (8,5): There is only 1 point in its neighbourhood so it is labelled as a border point and C_1 is not updated.

   We cannot go further. Status of visited and unvisited points are:

Sno	Point	Neighbourhood	Label	Visited	Cluster
1	(5,1)	-	Noise	No	-
2	(3,4)	(3,6),(4,5)	Border	Yes	C_1
3	(7,3)	(7,5)	Border	Yes	C_1
4	(3,8)	(3,6),(4,7)	Border	Yes	C_1
5	(3,6)	(3,4),(3,8),(4,7),(4,5)	Core	Yes	C_1
6	(4,7)	(3,8),(3,6),(4,5)	Core	Yes	C_1
7	(4,5)	(3,4),(3,6),(4,7),(5,5)	Core	Yes	C_1
8	(5,5)	(4,5),(7,5)	Border	Yes	C_1
9	(7,5)	(7,3),(5,5),(8,5)	Core	Yes	C_1
10	(8,5)	(7,5)	Border	Yes	C_1

4. Repeat for core points which have not been visited. We notice that there are no core points which have not been visited. I considered the same dataset as given in the previous reading assignment in section 6.1. For a different dataset or for different initialisation of epsilon and MinPts, we would have been able to observe more clusters.

5. Any non-core points which have not been visited will remain as noise. Final clustering:
   $C_1 = \{(3,4),(3,8),(3,6),(4,7),(4,5),(5,5),(7,5),(7,3),(8,5)\}$
   $Noise = \{(5,1)\}$

3 GA Operators

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.

3.1 Selection

1. Fitness Proportionate Selection is one of the most popular ways of parent selection. In this every individual can become a parent with a probability which is proportional to its fitness. Therefore, fitter individuals have a higher chance of mating and propagating their features to the next generation.
   Consider a circular wheel. The wheel is divided into n pies, where n is the number of individuals in the population. Each individual gets a portion of the circle which is proportional to its fitness value. There are two implementations under this.

(a) Roulette Wheel Selection: In roulette wheel selection, the circular wheel is divided as described before. A fixed point is chosen on the wheel circumference 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. A fitter individual has a greater chance of landing in front of the fixed point when the wheel is rotated. Therefore, the probability ($p(i)$) of choosing an individual depends directly on its fitness.

$$p(i) = \frac{f(i)}{\sum_{j=1}^{n} f(j)}$$

To illustrate, consider the given population.

S no	Chromosome	Decoded Integer	Fitness Score	Fitness (%)	Range
1	00101	5	80	0	0
2	11110	30	25230	56	[1,56]
3	01111	15	2940	7	[57,63]
4	01011	11	1100	2	[64,65]
5	10111	23	11132	25	[66,90]
6	10001	17	4352	9	[91,99]

A random number from [0,99] is generated, which is then mapped to a chromosome ($C_1, ..., C_6$) based on the chromosome's range. Let's say this number is 56. Then chromosome 2 is selected as a parent. This procedure is repeated for selecting the other parent.

(b) Stochastic Universal Sampling (SUS): It is quite similar to Roulette wheel selection, however instead of having just one fixed point, we have multiple fixed points. 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.

To illustrate this, assume that there are 2 fixed points. One would be the random number that is generated, say x, and the other would be $(x - 50)\%100$ (100 is the size of the range). So, if the random number generated is 89 then the other number would be $(89 - 50)\%100 = 39$. So, chromosomes 2 and 5 are selected as parents.

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

2. 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.

S no	Chromosome	Decoded Integer	Fitness Score
1	00101	5	80
2	11110	30	25230
3	01111	15	2940
4	01011	11	1100
5	10111	23	11132
6	10001	17	4352

To illustrate, assume k=3 and the randomly selected individuals are 1, 4 and 5. Chromosome 5 has the best fitness score among these, so it is selected as a parent.

3. 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 usually happens at the end of the run). So, no matter how fit they are relative to each other, each individual has an approximately equal probability of getting selected as a parent. This in turn leads to a loss in the selection towards fitter individuals and poor parent selections.

   To overcome this, every individual in the population is ranked according to their fitness. The best individual gets rank n and the worst individual gets rank 1. The probability of selection ($p(i)$) depends on the rank of the individual and not the fitness. The higher ranked individuals are preferred over the lower ranked ones.

$$p(i) = \frac{rank(i)}{n \times (n-1)}$$

S no	Chromosome	Decoded Integer	Fitness Score	Rank	$p(i)$*100	Range
1	00101	5	80	1	3	[0,2]
2	11110	30	25230	6	20	[3,22]
3	01111	15	2940	3	10	[23,32]
4	01011	11	1100	2	7	[33,39]
5	10111	23	11132	5	17	[40,56]
6	10001	17	4352	4	13	[57,69]

After ranking, roulette wheel method can be used to select the parents. To illustrate this, a random number from [0,69] is generated, which is then mapped to a chromosome ($C_1, ..., C_6$) based on the chromosome's range. Chromosome with a higher rank is more likely to be selected as it has a bigger range. Let's say this number is 56. Then chromosome 5 is selected as a parent. This procedure is repeated for selecting the other parent.

3.2 Crossover

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

   To illustrate, assume there are 2 chromosomes 10111 and 00100. The crossover point is selected randomly from a range [1,4]. Say, that the generated point is 3 then,

Parents	Segments	Children
10111	101 11	10100
00100	001 00	00111

2. Multi Point Crossover: This is a generalization of the one-point crossover wherein multiple crossover points are chosen and alternating segments are swapped to get new off-springs.

   To illustrate, let the chromosome length be 10 and 2 crossover points are selected from the range [1,9]. Say that the generated points are 3 and 7 then,

Parents	Segments	Children
1011101110	101 1101 110	1010011110
0010011111	001 0011 111	0011101111

3. 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.

   To illustrate this, assume that a chromosome is of length 5, so a random number in the range [0,1] is generated 5 times. If the number is 0 then the bit does not change, while if it is 1 then the corresponding bits are swapped. Say that the numbers generated are 0, 1, 0, 0, and 1 then the 2nd and last bits are swapped to produce offspring:

Parents	Children
10111	10110
00100	00101

4. Davis 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. In this crossover method, each chromosome of length n consists of the digits 0-(n-1) without repetitions. In case a digit has already been copied then it should be skipped when copying from the second parent.
- Repeat for the second child with the parents role reversed.

To illustrate, let the chromosome length be 10 and 2 crossover points are selected from the range [1,9]. Say that the generated points are 3 and 7 then.

Parents	Segments
0123456789	012 3456 789
8473625190	847 3625 190

Child 1: First copy the segment between the 2 points (middle segment) from the first parent:

_ _ _ 3 4 5 6 _ _ _

Next consider the digits after the second point in the second parent and start copying numbers if they don't already exist in the child.

_ _ _ 3 4 5 6 1 _ _

_ _ _ 3 4 5 6 1 9 _

_ _ _ 3 4 5 6 1 9 0

8 _ _ 3 4 5 6 1 9 0

The next digit is 4 but it already exists in the child, so skip it. 7 is the next digit and it is not in the child, so copy 7.

8 7 _ 3 4 5 6 1 9 0

3 and 6 are already present in the child, so skip them and copy 2.

8 7 2 3 4 5 6 1 9 0

Child 2: Now reverse the roles of the parents. First copy the segment between the 2 points (middle segment) from the first parent:

_ _ _ 3 6 2 5 _ _ _

Next consider the digits after the second point in the second parent and start copying numbers if they don't already exist in the child.

_ _ _ 3 6 2 5 7 _ _

_ _ _ 3 6 2 5 7 8 _

_ _ _ 3 6 2 5 7 8 9

0 _ _ 3 6 2 5 7 8 9

0 1 _ 3 6 2 5 7 8 9

The next digits, 2 and 3, already exist in the child, so skip them. 4 is the next digit and it is not in the child so, copy 4.

0 1 4 3 6 2 5 7 8 9

So, the parents and children are:

Parents	Segments	Children
0123456789	012 3456 789	8723456190
8473625190	847 3625 190	0143625789

3.3 Mutation

1. Bit Flip Mutation: In this, one or more random bits are selected and flipped. This is used for binary encoded GAs.

   To illustrate, consider a chromosome 10111. A random number in the range [1,5] is generated. Say that the number is 4. So, the bit in the 4th position is flipped and the resultant chromosome is 10101.

2. Random Resetting: This 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.

   To illustrate consider the chromosome 12345. A random number in the range [1,5] is generated. Say that the number is 4. Let the set of permissible values be in the range [1,5] and the number selected by 2. So,

Before Mutation	Position	Gene value	After Mutation
12345	4	2	12325

3. Swap Mutation: In swap mutation, two positions on the chromosome are selected at random, and the values are interchanged. This is common in permutation based encodings.

   To illustrate consider the chromosome 10101. Let the generated positions, in the range [1,5], be 1 and 4. Then the resultant chromosome would be 00111.

4. Inversion Mutation: In inversion mutation, a subset of genes is selected and then inverted(reversed).

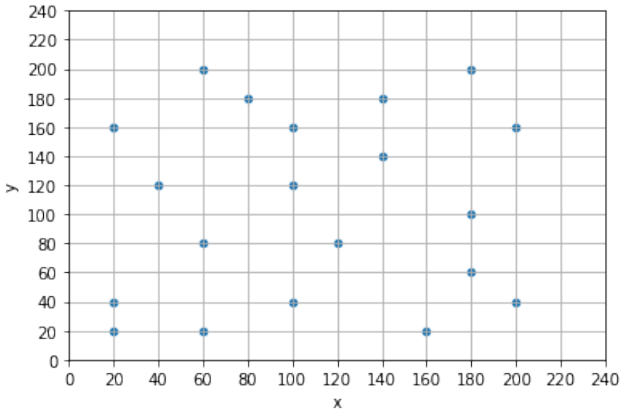
   To illustrate, consider the below example. Generate 2 random points in the range [1,9]. Let it be 3 and 7. Then the subset which should be inverted would be 0101.

Before Mutation	Subsets	After Mutation
0110101110	011 0101 110	011 1010 110 $\equiv$ 0111010110

4 Travelling Salesman Problem

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

City Map : Consider the cities shown in the image below, denoted on a 2D grid as points.



The problem is to find the best route such that we traverse all the cities in the least distance. To solve this problem, genetic algorithm will be used.

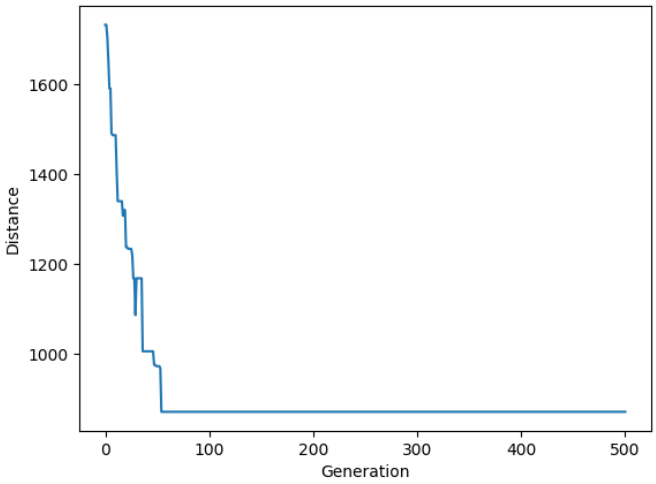
Population Initialisation: Let us consider a population size of 200. Now, an individual or a chromosome would be a possible route i.e the sequence of cities to which we will travel. Each city is an object having an x coordinate and a y coordinate. 200 routes are randomly chosen. Fitness is defined as the total distance (Euclidean) required to travel that route.

Selection: Roulette wheel selection method is used along with elitism. Elitism is used to hold on to the best routes. Let the elite size be 20. So, due to elitism the top 20 routes will definitely be included in the next generation. The remaining 180 parents are chosen using roulette wheel method from the entire population of size 200.

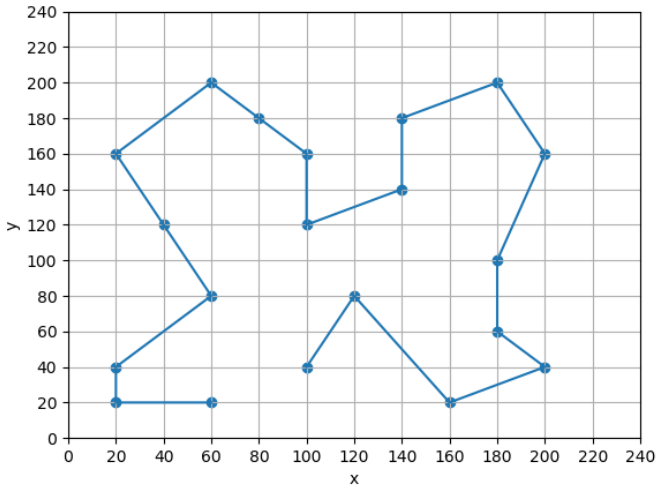
Crossover: Cities should not repeat and all cities should appear in the route. This calls for a permutation based crossover. So we will use Ordered crossover (OX1/Davis Order Crossover).

Mutation: Again we require a mutation suitable for permutations. So, we will use swap mutation method. Here we use a mutation probability of 0.01. So, the probability of a chromosome getting mutated is 1%.

Result: The algorithm is iterated over 500 generations.
Initial Population Best Distance: 1732.5
Final Population Best Distance: 871.1
The progress can be plotted as the individual with the best fitness (i.e the route with minimum distance) for each generation. This is shown below. It can be seen that after ∼60th generation, the best distance does not change.



The output of the algorithm i.e the best route in the 500th generation is shown below. The program is given in the file, tsp.py. Running it may not give the exact same path. Order may change slightly but it will be similar to what is shown.



5 Classifiers

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

5.1 SLIQ

SLIQ is a tree classifier which can handle both numerical and categorical attributes. It pre-sorts the numerical attributes before the tree has been built. It is a breadth first growing strategy as opposed to depth first. It uses gini index as a measure to determine best split. It is an inexpensive tree pruning algorithm based on minimum description length (MDL). For a training set L with n distinct classes:

$$Gini(L) = 1 - \sum_{j=1}^{n} p_j^2 \tag{1}$$

The gini index for a binary split of set L into subsets L_1 and L_2 will be:

$$Gini_{split}(L) = \frac{|L_1|}{|L|} Gini(L_1) + \frac{|L_2|}{|L|} Gini(L_2) \tag{2}$$

Minimum Description Length (MDL): The best model for a given data set minimises the sum of the length of the encoded data by the model plus the length of the model.

$$cost(M, D) = cost(D|M) + cost(M)$$

where,
$cost(M)$ = cost of the model (length of the model)
$cost(D|M)$ = cost to describe the data with the model (classification errors)

Pseudocode:

1. Start Pre-sorting of the samples.
   (a) For each attribute, create an attribute list with columns for the value and an index into the class list (sample-ID).
   (b) Create a class list with columns for the sample-ID, class label and leaf node.
   (c) Iterate over all training samples, for each attribute:
      - Insert its attribute values, sample-ID and class (sorted by attribute value) into the attribute list.
      - Insert the sample-ID, the class and the leaf node (sorted by sample-ID) into the class list.
2. As long as the stop criterion has not been reached:
   (a) For every attribute
      i. Place all nodes into a class histogram.
      ii. Start evaluation of the splits.

A. For each node, and for all attributes construct a histogram (for each class the histogram saves the count of samples before and after the split).

B. For each attribute A, for each value v (traverse the attribute list for A)

- Find the entry in the class list (provides the class and node).
- Update the histogram for the node.
- Assess the split.

(b) Choose a split.

(c) Update the decision tree; for each new node update its class list (nodes).

i. Traverse the attribute list of the attribute used in the node.

ii. For each entry (value, ID) find the matching entry (ID, class, node) in the class list.

iii. Apply the split criterion emitting a new node.

iv. Replace the corresponding class list entry with (ID, class, new node).

Trace: Consider the dataset given below.

Class List Index	Age	Salary	Class
1	30	65	G
2	23	15	B
3	40	75	G
4	55	40	B
5	55	100	G
6	45	60	G

1. Pre-sorting step: There are 2 attributes, Age and Salary. A table is made for each attribute which lists the attribute value in ascending order along with the index of the value. This will make it easier to calculate the gini index when evaluating a split as we don't need to traverse the dataset multiple times to get the count of the values. A class list is also made which lists the class of each entry along with the leaf node it belongs to. Initially all entries will be at the root, so all will be found at the node N1.

Age List		Salary List		Class List		
Age	Class List Index	Salary	Class List Index	Class List Index	Class	Leaf
23	2	15	2	1	G	N1
30	1	40	4	2	B	N1
40	3	60	6	3	G	N1
45	6	65	1	4	B	N1
55	5	75	3	5	G	N1
55	4	100	5	6	G	N1

2. First level split:

   (a) Evaluation step: Scan each attribute list A. For each value v, assess the split if $A \leq v$. Histograms are attached to each leaf node to help in evaluating a split. For each value in the the attribute list, update the histogram and evaluate the split. After scanning the attribute list, choose the split which gave the best gini index result. In the histogram, L refers to values which satisfy the condition, while R refers to values which don't. B and G refer to class label. Before a split, the histogram values will be initialised under R.

   Note: In the example given in the paper (SLIQ: A Fast Scalable Classifier for Data Mining), they have assumed that the data has been initially split using the condition $Age \leq 35$.

   Consider the first possible split, $Age \leq 23$.

N1 Before split		
	B	G
L	0	0
R	2	4

N1 After split		
	B	G
L	1	0
R	1	4

$Gini(L) = 1 - (\frac{LB}{LB+LG}^2 + \frac{LG}{LB+LG}^2)$

$Gini(R) = 1 - (\frac{RB}{RB+RG}^2 + \frac{RG}{RB+RG}^2)$

$Gini(Condition) = \frac{LB+LG}{LB+LG+RB+RG} * Gini(L) + \frac{RB+RG}{LB+LG+RB+RG} * Gini(R)$

Now, LB=1, LG=0, RB=1, RG=4, so:

$Gini(L) = 1 - (1^2 + 0^2) = 0$

$Gini(R) = 1 - (\frac{1}{5}^2 + \frac{4}{5}^2) = 0.32$

$Gini(Age \leq 23) = \frac{1}{6} * 0 + \frac{5}{6} * 0.32 = 0.27$

Repeat this procedure for all remaining possible splits.

Condition	LB	LG	RB	RG	Gini(L)	Gini(R)	Gini(Condition)
$Age \leq 30$	1	1	1	3	0.5	0.375	0.42
$Age \leq 40$	1	2	1	2	0.4	0.4	0.4
$Age \leq 45$	1	3	1	1	0.375	0.5	0.42
$Salary \leq 15$	1	0	1	4	0.375	0.5	0.42
$Salary \leq 40$	2	0	0	4	0	0	0
$Salary \leq 60$	2	1	0	3	0.44	0	0.22
$Salary \leq 65$	2	2	0	2	0.5	0	0.33
$Salary \leq 75$	2	3	0	1	0.48	0	0.4

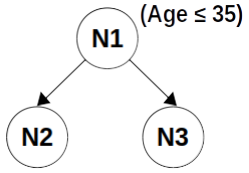
The split with the least Gini index should be selected. So the condition should be $Salary \leq 40$. But, if we select $Salary \leq 40$ then notice that the algorithm would end as it would lead to branches having pure class labels. So, for the sake of further illustration of the example, we'll consider the same assumption given in the paper i.e. $Age \leq 35$. The corresponding histogram of $N1$ is:

N1 Before split		
	B	G
L	0	0
R	2	4

N1 After split		
	B	G
L	1	1
R	1	3

(b) Update class list step: Now that the split has been decided, the class list should be updated to reflect this. $N1$ is split into $N2$ and $N3$. The condition is that $Age \leq 35$. Traverse the Age list and update the class list.

Age List		Class List		
Age	Class List Index	Class List Index	Class	Leaf
23	2	1	G	N2
30	1	2	B	N2
40	3	3	G	N3
45	6	4	B	N3
55	5	5	G	N3
55	4	6	G	N3



3. Second level split: Repeat the above steps for each node.

(a) Evaluation step: Consider $N2$ and the subset of the lists which belong to it (for ease of putting conditions). Since we have already used the Age attribute, now we will look at the remaining attribute i.e. Salary.

Salary List		Class List		
Salary	Class List Index	Class List Index	Class	Leaf
15	2	1	G	N2
65	1	2	B	N2

But, there is only one condition possible i.e. $Salary \leq 15$. The corresponding histogram of $N2$ is:

N2 Before split			N2 After split		
	B	G		B	G
L	0	0	L	1	0
R	1	1	R	0	1

Consider $N3$ and the subset of lists which belong to it. Since we have already used the Age attribute, now we will look at the remaining attribute i.e. Salary.

Salary List		Class List		
Salary	Class List Index	Class List Index	Class	Leaf
40	4	3	G	N3
60	6	4	B	N3
75	3	5	G	N3
100	5	6	G	N3

The possible splits are:

Condition	LB	LG	RB	RG	Gini(L)	Gini(R)	Gini(Condition)
$Salary \leq 40$	1	0	0	3	0	0	0
$Salary \leq 60$	1	1	0	2	0.5	0	0.25
$Salary \leq 75$	1	2	0	1	0.44	0	0.33

The best split would be $Salary \leq 40$ as it has the least Gini index value. The corresponding histogram of $N3$ is:

N3 Before split			N3 After split		
	B	G		B	G
L	0	0	L	1	0
R	1	3	R	0	3

(b) Update class list step: Now, the class list needs to be updated. $N2$ is split into $N4$ and $N5$ with the condition $Salary \leq 15$.

Salary List		Class List		
Salary	Class List Index	Class List Index	Class	Leaf
15	2	1	G	N5
65	1	2	B	N4

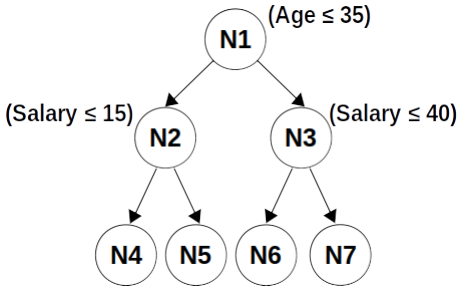
$N3$ is split into $N6$ and $N7$ with the condition $Salary \leq 40$.

Salary List		Class List		
Salary	Class List Index	Class List Index	Class	Leaf
40	4	3	G	N7
60	6	4	B	N6
75	3	5	G	N7
100	5	6	G	N7

4. $N4$, $N5$, $N6$ and $N7$ are nodes with pure class labels and so we cannot split further and the algorithm ends. Final class list is:

Class List		
Class List Index	Class	Leaf
1	G	N5
2	B	N4
3	G	N7
4	B	N6
5	G	N7
6	G	N7

The final tree is:



5.2 ARBC

Association Rule Based Classification or CBA (Classification Based on Associations) is a Classification Association Rule Mining (CARM) algorithm developed by Bing Liu, Wynne Hsu and Yiming Ma (Liu et al.

1998). CBA operates using a two stage approach to generate a classifier:

1. Generating a complete set of CARs (Classification Association Rules).
2. Prune the set of CARs to produce a classifier.

In Liu et al.'s implementation the first stage is implemented using CBA-RG (CBA-Rule Generator). CBA-RG is a multi-pass Apriori style algorithm. Rules, once generated, are pruned using the pessimistic error rate based pruning method described by Quinlan with respect to C4.5 (Quinlan 1992), and placed in a rule list R. The second stage is implemented using CBA-CB (CBA - Classifier Builder). The two stage approach is a fairly common approach used by many CARM algorithms, for example the CMAR (Classification based on Multiple Association Rules) algorithm (Li et al. 2001). The CBA algorithm commences by generating a list of CARS (potential CRs in the target classifier) ordered according to the following schema:

1. Confidence: A rule r1 has priority over a rule r2 if $confidence(r1) > confidence(r2)$.
2. Support: A rule r1 has priority over a rule r2 if $confidence(r1) == confidence(r2)$ && $support(r1) > support(r2)$.
3. Size of antecedent: A rule r1 has priority over a rule r2 if $confidence(r1) == confidence(r2)$ && $support(r1) == support(r2)$ && $|A_{r1}| < |A_{r2}|$.

Pseudocode:

1. Generate association rules using an associative rule mining algorithm such as Apriori or FP-Growth.
2. Order the rules based on the above 3 conditions for confidence, support and size of antecedent.
3. Start the CBA algorithm. First initialise,
   - D = distribution array, array of length equal to the number of classes in the input dataset which records the number of records associated with each class
   - Rerr = 0 (error rate)
4. For each r in rule list R:
   (a) L = Local distribution array recording the number of records, for each class satisfied by r
   (b) Remove from data set all records satisfied by r
   (c) Rerr = Rerr + Number of miss classifications produced by the application of rule r
   (d) $D = D - L$
   (e) r.DefaultClass = The majority class in D

(f) Derr = Number of miss classifications produced by use of the DefaultClass

(g) r.Terr = Rerr+Derr

(h) if no more records then break

5. On completion of the above step, the first rule r in R with the lowest totalError value is identified. All the rules uptil r are added to the classifier while all the rules after r are discarded. A default rule with the class i.e. r.DefaultClass is added to the end of the classifier.

Trace: Consider the sample dataset where e and f are considered as class labels.

Sno	Items
1	a b e
2	a e
3	b e
4	c d f
5	c f
6	d f
7	b c e
8	a d f

1. Generate the frequent itemsets with various support counts. We'll use apriori algorithm to generate frequent itemsets. Notice that there are no itemsets greater than length 1 which have support=4 and for getting association rules, the itemset should at least have length 2. So, we stop at support=3.

   - Support=1:

     $L_1 = \{a, b, c, d, e, f\}$

     $C_1 = \{a, b, c, d, e, f\}$

     $L_2 = \{ab, ac, ad, ae, af, bc, bd, be, bf, cd, ce, cf, de, df, ef\}$

     $C_2 = \{ab, ad, ae, af, bc, be, cd, ce, cf, df\}$

     $L_3 = \{abd, abe, abf, ade, adf, aef, bce, cde, cdf, cef\}$

     $C_3 = \{abe, cdf, bce, adf\}$

   - Support=2:

     $L_1 = \{a, b, c, d, e, f\}$

     $C_1 = \{a, b, c, d, e, f\}$

     $L_2 = \{ab, ac, ad, ae, af, bc, bd, be, bf, cd, ce, cf, de, df, ef\}$

     $C_2 = \{ae, be, cf, df\}$

- Support=3:

  $L_1 = \{a, b, c, d, e, f\}$

  $C_1 = \{a, b, c, d, e, f\}$

  $L_2 = \{ab, ac, ad, ae, af, bc, bd, be, bf, cd, ce, cf, de, df, ef\}$

  $C_2 = \{be, df\}$

2. Now get the confidence for each rule. The rules of the format $X \rightarrow e$ and $X \rightarrow f$ are considered as e and f are the class labels.

   For a rule $X \rightarrow Y$, the confidence $= P(Y|X) = \frac{n(X,Y)}{n(X)} = \frac{count(XY)}{count(X)}$

Rule	Confidence	Support
$a \rightarrow e$	0.66	2
$a \rightarrow f$	0.33	1
$b \rightarrow e$	1	3
$c \rightarrow e$	0.33	1
$c \rightarrow f$	0.66	2
$d \rightarrow f$	1	3
$cd \rightarrow f$	1	1
$ab \rightarrow e$	1	1
$bc \rightarrow e$	1	1
$ad \rightarrow f$	1	1

3. Order the rules based on the 3 conditions specified earlier.

Rule	Confidence	Support
$b \rightarrow e$	1	3
$d \rightarrow f$	1	3
$bc \rightarrow e$	1	1
$ad \rightarrow f$	1	1
$cd \rightarrow f$	1	1
$ab \rightarrow e$	1	1
$a \rightarrow e$	0.66	2
$c \rightarrow f$	0.66	2
$a \rightarrow f$	0.33	1
$c \rightarrow e$	0.33	1

4. Apply each rule to the dataset in the above order. Initialise $Rerr = 0$, $D = [4, 4]$. D is an array which specifies the number of data entries belonging to each class. There are 2 class labels, e and f, and 4 instances of each class. The dataset is given below for ease of reference.

Sno	Items
1	a b e
2	a e
3	b e
4	c d f
5	c f
6	d f
7	b c e
8	a d f

(a) Rule $r = b \rightarrow e$

- $L = [3, 0]$

  This is an array which specifies the number of instances for each class satisfying the rule. Here we have 2 classes, so L will be a 1x2 array. For the data items which satisfy the given rule, there are 3 instances for class e and 0 instances for class f.

- Rerr=Rerr+0=0

  The classification error for the rule will be the number of misclassifications if the rule is applied on the dataset. For this rule, that means if there is an item b then the class label is e. There are no data entries with misclassification.

- $D = D - L = [4, 4] - [3, 0] = [1, 4]$

- r.DefaultClass=f

  This is the majority class in D.

- Derr=1

  This is the number of misclassifications if the default class is applied to D. Since the default class is f and there is 1 value for e in the D array, so the misclassification is 1.

- r.Terr=Rerr+Derr=1

(b) $r = d \rightarrow f$

- $L = [0, 3]$

  For the data items which satisfy the given rule, there are 0 instances for class e and 3 instances for class f.

- Rerr=Rerr+0=0

  For this rule, if there is an item d then the class label is f. There are no data entries with misclassification.

- $D = [1, 4] - [0, 3] = [1, 1]$
- r.DefaultClass=e

  Both are equal, so we take the first class.

- Derr=1

  This is the number of misclassifications if the default class is applied to the remaining dataset. Since the default class is e, out of 2 instances there is 1 misclassification.

- r.Terr=Rerr+Derr=1

(c) $r = bc \rightarrow e$

- $L = [1, 0]$

  For the data items which satisfy the given rule, there is 1 instance for class e and 0 instances for class f.

- Rerr=Rerr+0=0

  For this rule, if there is an item bc then the class label is e. There are no data entries with misclassification.

- $D = [1, 1] - [1, 0] = [0, 1]$
- r.DefaultClass=f

  There is only 1 instance and that is of class f.

- Derr=0

  Since there is only 1 instance left, the misclassification is 0.

- r.Terr=Rerr+Derr=0

(d) $r = ad \rightarrow f$

- $L = [0, 1]$

  For the data items which satisfy the given rule, there is 0 instances for class e and 1 instance for class f.

- Rerr=Rerr+0=0

  For this rule, if there is an item ad then the class label is f. There are no data entries with misclassification.

- $D = [0, 1] - [0, 1] = [0, 0]$
- r.DefaultClass=e

  Both are equal to 0, so take first class as default.

- Derr=0

  Since 0 instances are left, the misclassification is 0.

- r.Terr=Rerr+Derr=0

(e) Since $D = [0, 0]$, we cannot go further and the loop ends. The rules which have been applied are summarised in the table.

Rule r	L	Rerr	D	r.DefaultClass	Derr	r.Terr
$b \rightarrow e$	$[3, 0]$	0	$[1, 4]$	f	1	1
$d \rightarrow f$	$[0, 3]$	0	$[1, 1]$	e	1	1
$bc \rightarrow e$	$[1, 0]$	0	$[0, 1]$	f	0	0
$ad \rightarrow f$	$[0, 1]$	0	$[0, 0]$	e	0	0

5. Now add each rule to the classifier till we reach the first rule with minimum Terr. In this case, the first rule with minimum Terr would be $bc \rightarrow e$. The default class would be the default class corresponding to the last rule added to the classifier, which in this case is f. So, the final classifier would be:
   $b \rightarrow e$
   $d \rightarrow f$
   $bc \rightarrow e$
   Default f

6 Sequence Pattern Mining

Define sequence pattern mining and understand the working of

1. GSP
2. Prefix-Span algorithms.

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

Definitions:

- Data mining: This consists of extracting information from data stored in databases to understand the data and/or take decisions. Some of the most fundamental data mining tasks are clustering, classification, outlier analysis, and pattern mining.
- Pattern mining: This consists of discovering interesting, useful, and unexpected patterns in databases. Various types of patterns can be discovered in databases such as frequent itemsets, associations, subgraphs, sequential rules, and periodic patterns.

- Sequential pattern mining: It is a topic of data mining concerned with finding statistically relevant patterns between data examples where the values are delivered in a sequence.
- More precisely, it consists of discovering interesting subsequences in a set of sequences, where the interestingness of a subsequence can be measured in terms of various criteria such as its occurrence frequency, length, and profit.
- Sequential pattern mining has numerous real-life applications due to the fact that data is naturally encoded as sequences of symbols in many fields such as bioinformatics, e-learning, market basket analysis, texts, and webpage click-stream analysis.
- A sequence database:
  $< a(abc)(ac)d(cf) >$
  $< (ad)c(bc)(ae) >$
  $< (ef)(ab)(df)cb >$
  $< eg(af)cbc >$
- Inside a subsequence (...) items are listed alphabetically. An item can occur at most once in a subsequence.
- Subsequence with one item is written without brackets.

6.1 GSP

Generalised Sequential Pattern mining algorithm (GSP) is an Apriori based approach to sequential pattern mining. It uses a basic property of apriori algorithm which is, if a sequence S is not frequent, then none of the super-sequences of S is frequent. Using the apriori method, frequent itemsets are found which in this case would be frequent sequences. Benefit is the pruning which reduces search space. While some bottlenecks are that it scans the database multiple times and it generates a huge set of candidate sequences.

Pseudocode:

1. Initially, every item in the database is a candidate of length 1.
2. For each level (i.e. sequences of length k) do:
   (a) Scan database to collect support count for each candidate sequence
   (b) Generate candidate length (k+1) sequences from length k frequent sequences.
   (c) Repeat until candidate set cannot be found.

Trace: Consider the dataset given below.

Transaction Date	Customer ID	Items purchased
1	01	a
1	02	b
1	03	b
2	04	f
3	01	b
3	05	a
4	02	g
4	05	bc
5	03	f
6	04	ab
6	02	d
7	01	fg
7	05	g
8	04	c
8	03	g
9	05	f
9	01	c
9	03	ab
10	01	d
10	05	de
10	04	d

The sequence for each customer can be generated as:

Customer ID	Sequence
01	$< ab(fg)cd >$
02	$< bgd >$
03	$< bfg(ab) >$
04	$< f(ab)cd >$
05	$< a(bc)gf(de) >$

1. Let minimum support be 2. Consider items of length 1.

Item	a	b	c	d	e	f	g
Support	4	5	3	4	1	4	4

$L_1 = \{a, b, c, d, e, f, g, h\}$

$C_1 = \{a, b, c, d, e, f\}$

2. Here, order matters and single items might repeat, so we need to consider all combinations. L_2 would be all values from $C_1 \times C_1$. These would be items bought in different transactions.

	a	b	c	d	f	g
a	aa	ab	ac	ad	af	ag
b	ba	bb	bc	bd	bf	bg
c	ca	cb	cc	cd	cf	cg
d	da	db	dc	dd	df	dg
f	ea	eb	ec	ed	ef	eg
g	ga	gb	gc	gd	gf	gg

Along with this we also need to consider the possibility of 2-plus items being bought in the same transaction.

	a	b	c	d	f	g
a		(ab)	(ac)	(ad)	(af)	(ag)
b			(bc)	(bd)	(bf)	(bg)
c				(cd)	(cf)	(cg)
d					(df)	(dg)
f						(fg)
g						

$C_2 = \{$ab, ac, ad, af, ag, bc, bd, bf, bg, cd, fa, fb, fc, fd, gd, (ab)$\}$

3. For length=3:

$L_3 = \{$abc, abd, abf, abg, acd, afb, afc, afd, agd, bcd, bfc, bfd, bgd, fab, fac, fad, f(ab), fbc, fbd, fcd, (ab)c, (ab)d, (ab)f, (ab)g$\}$

$C_3 = \{$abd, abf, abg, acd, afd, agd, bcd, bfd, bgd, f(ab), fcd$\}$

4. For length=4:

$L_4 = \{$abfd, abgd$\}$

$C_4 = \{$abfd, abgd$\}$

5. The algorithm ends here. The frequent sequences are $C_1 \cup C_2 \cup C_3 \cup C_4$.

6.2 Prefix Span

Prefix Span is an algorithm for discovering sequential patterns in sequence databases, proposed by Pei et al. (2001). Prefix Span is like apriori but, uses prefix-projection method to reduce a candidate generation.

- a_i, b_j are items.
- α_i, β_j are itemsets.
- α, β are sequences of itemsets.
- $\alpha = \langle \alpha_1, \alpha_2, ..., \alpha_n \rangle$ and $\beta = \langle \beta_1, \beta_2, ..., \beta_n \rangle$.
- α is prefix of β if $\alpha_1 = \beta_1, ..., \alpha_{m1} = \beta_{m1}$, $\alpha_m \subseteq \beta_m$ and items in $\beta_m - \alpha_m$ are alphabtically after those in α_m.
- a(ab)c is prefix of a(ab)(acd)d(ab).
- Sequence after prefix is postfix. (_ d)d(ab) is postfix in a(ab)(acd)d(ab) after a(ab)c .
- Given $\beta \subseteq \alpha$, $\gamma \subseteq \alpha$, γ is β-project of α if β is prefix of γ and there is no longer subsequence in α so that β is its prefix.
- c -project of a(ab)(cd)cd(ab) is (_ d)cd(ab)

Pseudocode:

1. Find from database all items which have at least support s. Add them to iList.
2. For all items x in iList:
   (a) Form x-project
   (b) Find supported items
   (c) Prune x-project
   (d) If x-project has more than one sequence: nextLevel=PrefixSpan(x-project,Items,s)
   (e) iList=join(x,Items+nextLevel)
3. Return iList;

Trace: Consider the database:

Sno	Sequence
1	$< a(abc)(ac)d(cf) >$
2	$< (ad)c(bc)(ae) >$
3	$< (ef)(ab)(df)cb >$
4	$< eg(af)cbc >$

Let us find sequential patterns starting with $\{ab\}$ and with support 2.

1. First scan all items which have support at least 2. This is $iList = \{a, b, c, d, e, f\}$

2. Since we want sequences starting with ab, we'll start with a-project:

   $< (abc)(ac)d(cf) >$

   $< (\_d)c(bc)(ae) >$

   $< (\_b)(df)cb >$

   $< (\_f)cbc >$

   Items with support at least 2: $\{a, b, c, d, f, (\_b)\}$

   Items with support less than 2: $\{e, (\_e), (\_c), (\_d), (\_f)\}$

3. Prune a-project:

   $< (abc)(ac)d(cf) >$

   $< c(bc)a >$

   $< (\_b)(df)cb >$

   $< cbc >$

   Pruning involves removing the items which aren't having minimum support from the sequences in a-project.

4. Get b-project from pruned a-project:

   $< (\_c)(ac)d(cf) >$

   $< (\_c)a >$

   $< c >$

   Items with support at least 2: $\{(\_c), a, c\}$

   Items with support less than 2: $\{d, f\}$

   We can also do a-project, c-project, d-project, f-project and (_ b)-project from a-project as these are the items with minimum support in a-project. But as we are interested in sequences starting with ab, we'll only consider b-project and then expand that completely.

5. Prune b-project:

   $< (\_c)(ac)c >$

$< (\_c)a >$

$< c >$

We'll check the (_ c)-project, a-project and c-project under this as these are the items with minimum support in b-project.

- Get (_ c)-project from b-project:

  $< (ac)c >$

  $< a >$

  Items with support at least 2: $\{a\}$

  Items with support less than 2: $\{c\}$

  Pruned (_ c)-project:

  $< a >$

  Since $\{a\}$ has minimum support in (_ c), we'll get the a-project from the (_ c)-project.

  $<>$

  This is an empty set and we cannot go further.

- Get a-project from b project:

  $< (\_c)c >$

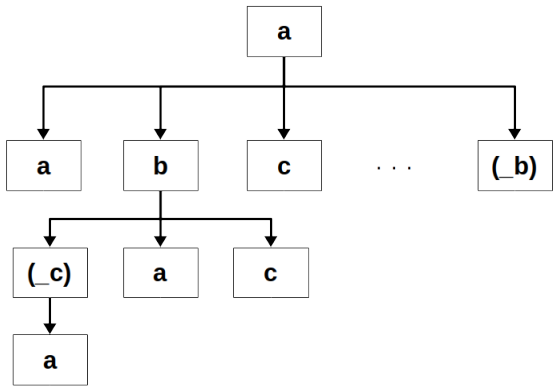
  Only one sequence. We can not get anything with support 2.

- Get c-project from b-project:

  $< c >$

  Only one sequence. We can not get anything with support 2.

6. A tree for a-project can be obtained to denote all the possible sequences starting with a. As we were interested in sequences starting with ab, only b branch was completely expanded. Other branches can similarly be expanded.



Similar to this, trees can be made for the other items in *iList*. From the tree we can see that, the

sequential patterns starting with ab and support 2 are : $\{ab, a(bc), a(bc)a, aba, abc\}$

7 GA Schemata Theorem

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

Schema: A schema is a set of binary strings that match the template for schema H. A template is made up of 1s, 0s, and *s where * is the "don't care" symbol that matches either 0 or 1.
Example: The schema H = 10*1* represents the set of binary strings: 10010, 10011, 10110, 10111, ...
The string 10 of length $l = 2$ belongs to $2^l = 2^2$ different schemas: **, *0, 1*, 10

Order $o(H)$: The order of a schema is the number of its fixed bits, i.e. the number of bits that are not '*' in the schema H.
Example: if H = 10*1* then o(H) = 3

Defining Length $\delta(H)$: The defining length is the distance between its first and the last fixed bits.
Example 1: if H = *1*01 then $\delta(H) = 5 - 2 = 3$
Example 2: if H = 0**** then $\delta(H) = 1 - 1 = 0$

Count: Suppose x is an individual that belongs to the schema H, then we say that x is an instance of H ($x \in H$). m(H, k) denotes the number of instances of H in the kth generation.

Fitness: $f(x)$ denotes fitness value of x. $f(H, k)$ denotes average fitness of H in the k-th generation.

$$f(H,k) = \frac{\sum_{x \in H} f(x)}{m(H,k)}$$

Genetic Algorithms involves 3 operations.

1. Selection
2. Crossover
3. Mutation

By looking at the effect of each of these operations on a schema we will get the schema theorem.

Selection: Selection probability for the individual x:

$$p_s(x) = \frac{f(x)}{\sum_{i=1}^{N} f(x_i)}$$

This is with the assumption that we are performing fitness proportional selection.

Net Effect: The expected number of instances of H in the mating pool $M(H,k)$ is:

$$M(H,k) = \frac{\sum_{x \in H} f(x)}{\overline{f}} = m(H,k)\frac{f(H,k)}{\overline{f}}$$

where $\overline{f}$ is the average fitness of the whole population.

This implies that schemas with fitness greater than the population average are likely to appear more in the next generation.

Crossover: Assuming the method is single point crossover, then schema H survives crossover operation if:

- one of the parents is an instance of the schema H and
- one of the offspring is an instance of the schema H

Given a string with length l and a schema H with the defining length $\delta(H)$, the probability that the crossover occurs within the bits of the defining length is $\frac{\delta(H)}{l-1}$.

Example: Suppose H = *1**0. We gave l = 5 then $\delta(H) = 5 - 2 = 3$. Thus, the probability that the crossover occurs within the defining length is $\frac{3}{4}$.

The upper bound of the probability of the schema H being destroyed is:

$$D_c(H) \leq p_c\frac{\delta(H)}{l-1})$$

where p_c is the crossover probability.

Net Effect: The lower bound on the probability $S_c(H)$ that H survives is:

$$S_c(H) = 1 - D_c(H) \geq 1 - p_c\frac{\delta(H)}{l-1}$$

This implies that schemas with low order are more likely to survive.

Mutation: Assuming that mutation is applied gene by gene. For a schema H to survive, all fixed bits must remain unchanged. Probability of a gene not being changed is $(1-p_m)$ where p_m is the mutation probability of a gene.

Net Effect: The probability a schema H survives under mutation:

$$S_m(H) = (1-p_m)^{o(H)} \approx 1 - o(H)p_m, \text{ since } p_m \ll 1$$

This implies that schemas with low order are more likely to survive.

Schema Theorem: The schema theorem states that the schema with above average fitness, short defining length and lower order is more likely to survive.

Expected number of schema H in next generation > Expected number in Mating Pool * Prob. of Surviving Crossover * Prob. of Surviving Mutation

Mathematically:

$$m(H, k+1) \geq M(H, k) * S_c(H) * S_m(H)$$

On substitution,

$$m(H, k+1) \geq m(H, k) \frac{f(H, k)}{\overline{f}} \left(1 - p_c \frac{\delta(H)}{l-1}\right) (1 - o(H) p_m)$$

Since $p_m \ll 1$,

$$m(H, k+1) \geq m(H, k) \frac{f(H, k)}{\overline{f}} \left(1 - p_c \frac{\delta(H)}{l-1} - o(H) p_m\right)$$

Application: Consider the function $f(x) = x^3 - 2x^2 + x$. The schema theorem will be applied to this function to get the optimal value of $x \in [0, 31]$ which will maximise $f(x)$.

First let us define 3 different schemata $H_1 = 1 * * * *$, $H_2 = *10 * **$ and $H_3 = 1 * * * 0$.

Consider the initial population generated as per the previous reading assignment.

S no	Chromosome	Decoded Integer	Fitness Score	Fitness (%)	Range
1	00101	5	80	0	0
2	11110	30	25230	56	[1,56]
3	01111	15	2940	7	[57,63]
4	01011	11	1100	2	[64,65]
5	10111	23	11132	25	[66,90]
6	10001	17	4352	10	[91,100]

Now, we process the schemas according to the population.

Sno	Schema	String Representatives	Schema Average Fitness $f(H)$
H_1	1****	2,5,6	13571
H_2	*10**	4	1100
H_3	1***0	2	25230

Selection: The expected number of instances in the mating pool is:

$$M(H, k) = m(H, k) \frac{f(H, k)}{\overline{f}}$$

.

Now, $\overline{f} = \dfrac{80 + 25230 + 2940 + 1100 + 11132 + 4352}{6} = 7472$

For each schema:

1. $M(H_1, k) = 3 * \dfrac{13571}{7472} = 5.44$
2. $M(H_2, k) = 1 * \dfrac{1100}{7472} = 0.15$
3. $M(H_2, k) = 1 * \dfrac{25230}{7472} = 3.37$

If we observe the mating pool i.e the chosen parents:

S no	Parent 1	Parent 2	Crossover point	Offspring1	Offspring 2
1	19=C_2=11110	73=C_5=10111	4	11111	10110
2	43=C_2=11110	95=C_6=10001	2	11001	10110
3	18=C_2=11110	25=C_2=11110	1	11110	11110

Schema processing of the mating pool (consider the matching parents):

Sno	Schema	Actual Count	Expected Count
H_1	1****	6	5.44
H_2	*10**	0	0.15
H_3	1***0	4	3.37

We see that the actual count is about the same as the expected count.

Crossover: The expected number of individuals after crossover:

$$m(H, k+1) \geq m(H, k) \frac{f(H, k)}{\overline{f}} \left(1 - p_c \frac{\delta(H)}{l - 1}\right)$$

Now $l = 5$ and $p_c = 1$ since all chosen individuals were mated. So,

1. $m(H_1, k+1) \geq 3 * \dfrac{13571}{7472} \left(1 - \dfrac{0}{4}\right) = 5.44$
2. $m(H_2, k+1) \geq 1 * \dfrac{1100}{7472} \left(1 - \dfrac{1}{4}\right) = 0.11$
3. $m(H_3, k+1) \geq 1 * \dfrac{25230}{7472} \left(1 - \dfrac{4}{4}\right) = 0$

   Due to the long defining length of H_3, it will get destroyed during crossover.

Schema processing of the offspring (refer to above table where selection and crossover occurred):

Sno	Schema	$m(H,k+1)$	$m(H,k)\frac{f(H,k)}{\overline{f}}\left(1-p_c\frac{\delta(H)}{l-1}\right)$
H_1	1****	6	5.44
H_2	*10**	1	0.11
H_3	1***0	4	0

We see that the condition is satisfied for all schemas.

Mutation: The expected number of individuals after mutation:

$$m(H,k+1) \geq m(H,k)\frac{f(H,k)}{\overline{f}}\left(1-p_c\frac{\delta(H)}{l-1}-o(H)p_m\right)$$

Now, in the previous reading assignment, out of the population of 6, 2 were randomly selected and out of the 5 bits, 1 bit was randomly flipped during mutation. So, $p_m = \frac{2}{6} * \frac{1}{5} = 0.067$.

1. $m(H_1,k+1) \geq 3 * \frac{13751}{7472}\left(1-\frac{0}{4}-1*0.067\right) = 5.15$
2. $m(H_2,k+1) \geq 1 * \frac{1100}{7472}\left(1-\frac{1}{4}-1*0.067\right) = 0.1$
3. $m(H_3,k+1) \geq 1 * \frac{25230}{7472}\left(1-\frac{4}{4}-1*0.067\right) = -0.22$
   In the previous step, we saw that it becomes 0 as $\delta(H_3) = 4$, and now accordingly it becomes negative.

If we refer to the population following mutation from the previous reading assignment:

S no	Chromosome	Mutation point	Output	Decoded Integer
1	11111	-	11111	31
2	10110	-	10110	22
3	11001	5	11000	24
4	10110	-	10110	22
5	11110	2	10110	22
6	11110	-	11110	30

Schema processing of the new population:

Sno	Schema	$m(H,k+1)$	$m(H,k)\frac{f(H,k)}{\overline{f}}\left(1-p_c\frac{\delta(H)}{l-1}-o(H)p_m\right)$
H_1	1****	6	5.15
H_2	*10**	1	0.1
H_3	1***0	5	-0.22

The condition is satisfied for all schemas. Hence, the results obtained matches with the theorem.

8 Regression Models

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

8.1 Linear

Linear regression is a statistical approach which quantifies the relationship between one or more predictor (independent) variable(s) and one outcome (dependent) variable. Linear regression is commonly used for predictive analysis and modelling.

Given a data set $\{y_i, x_{i1}, ..., x_{ip}\}_{i=1}^{n}$ 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 that adds "noise" to the linear relationship between the dependent variable and regressors. Thus the model takes the form:

$$y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip} + \epsilon_i, \text{ i=1,...,n}$$

Linear regression with a single predictor variable is known as simple regression. In real-world applications, there is typically more than one predictor variable. Such regressions are called multiple regression.

Pseudocode: This is the algorithm for simple linear regression using least squares technique.

1. Read the data for the 2 variables X and Y.
2. Calculate $\sum_{i=1}^{n} X_i$, $\sum_{i=1}^{n} X_i^2$, $\sum_{i=1}^{n} Y_i$ and $\sum_{i=1}^{n} X_i Y_i$.
3. Calculate a and b of the linear equation, y = a + bx

$$b = \frac{n * \sum_{i=1}^{n} X_i Y_i - \sum_{i=1}^{n} X_i * \sum_{i=1}^{n} Y_i}{n * \sum_{i=1}^{n} X_i^2 - \sum_{i=1}^{n} X_i \sum_{i=1}^{n} X_i}, \; a = \frac{\sum_{i=1}^{n} Y_i - b * \sum_{i=1}^{n} X_i}{n}$$

Trace: Consider the sample dataset.

X	0	1	2	3	4	5	6	7	8	9
Y	1	3	2	5	7	8	8	9	10	12

1. Calculate the required values to get a and b.
   $\sum_{i=1}^{n} X_i = 45$, $\sum_{i=1}^{n} X_i^2 = 285$, $\sum_{i=1}^{n} Y_i = 65$, and $\sum_{i=1}^{n} X_i Y_i = 389$

2. Substitute the values to get a and b. (n=10)

$$b = \frac{n * \sum_{i=1}^{n} X_i Y_i - \sum_{i=1}^{n} X_i * \sum_{i=1}^{n} Y_i}{n * \sum_{i=1}^{n} X_i^2 - \sum_{i=1}^{n} X_i \sum_{i=1}^{n} X_i} = \frac{10 * 389 - 45 * 65}{10 * 285 - 45 * 45} = 1.1697$$

$$a = \frac{\sum_{i=1}^{n} Y_i - b * \sum_{i=1}^{n} X_i}{n} = \frac{65 - 1.1697 * 45}{10} = 1.2364$$

3. So the line fitted to this dataset would be of the form $y = 1.1697x + 1.2364$

8.2 Non-linear

In statistics, nonlinear regression is a form of regression analysis in which observational data are modelled by a function which is a nonlinear combination of the model parameters and depends on one or more independent variables. The data are fitted by a method of successive approximations. The statistical model is of the form:

$$y \sim f(X, \beta)$$

This relates a vector of independent variables, X, and its associated observed dependent variables, y. The function f is nonlinear in the components of the vector of parameters β, but otherwise arbitrary.
Examples of non-linear functions include exponential functions, logarithmic functions, trigonometric functions, power functions, Gaussian function, and Lorenz curves. Some popular algorithms for fitting a non-linear regression include Gauss-Newton algorithm, Gradient descent algorithm and Levenberg-Marquardt algorithm.
In general, there is no closed-form expression for the best-fitting parameters, as there is in linear regression. Usually numerical optimisation algorithms are applied to determine the best-fitting parameters.

Pseudocode: The specifics will depend on the function which you assume. But in general:

1. Initialise variables.
2. Evaluate the parameters based on the assumed function.
3. Evaluate convergence criterion.
4. Exit if the criterion<tolerance or if max iterations is reached.

Trace: Consider the dataset. (Ideal data is assumed for fast convergence.)

t	1	2	4	5	8
y	3.2939	4.2699	7.1749	9.3008	20.259

where, t corresponds to time and and y is the population count. Generally populations are modelled with exponential functions. So we'll assume the model to be:

$$y = x_1 e^{x_2 t}$$

where x_1 and x_2 are the parameters to be estimated. Gauss-Newton method will be used to obtain the parameters.

1. Consider the vector valued function:
$$F(x) = \begin{pmatrix} x_1 e^{x_2 t_1} - y_1 \\ x_1 e^{x_2 t_2} - y_2 \\ x_1 e^{x_2 t_3} - y_3 \\ x_1 e^{x_2 t_4} - y_4 \\ x_1 e^{x_2 t_5} - y_5 \end{pmatrix} = \begin{pmatrix} x_1 e^{1x_2} - 3 \\ x_1 e^{2x_2} - 4 \\ x_1 e^{4x_2} - 6 \\ x_1 e^{5x_2} - 11 \\ x_1 e^{8x_2} - 20 \end{pmatrix}$$
2. In Gauss-Newton method, it computes a search direction using the formula for Newton's method.
$$\nabla^2 f(x) p = -\nabla f(x)$$
$\nabla f(x)$ is gradient of the least-squares objective function:
$$f(x_1, x_2) = \tfrac{1}{2} \sum_{i=1}^{5} (x_1 e^{x_2 t_i} - y_i)^2 = \tfrac{1}{2} F(x)^T F(x)$$
So, $\nabla f(x_1, x_2) = \begin{pmatrix} \sum_{i=1}^{5} (x_1 e^{x_2 t_i} - y_i) e^{x_2 t_i} \\ \sum_{i=1}^{5} (x_1 e^{x_2 t_i} - y_i) x_1 t_i e^{x_2 t_i} \end{pmatrix}$
$$= \begin{pmatrix} e^{x_2 t_1} & e^{x_2 t_2} & e^{x_2 t_3} & e^{x_2 t_4} & e^{x_2 t_5} \\ x_1 t_1 e^{x_2 t_1} & x_1 t_2 e^{x_2 t_2} & x_1 t_3 e^{x_2 t_3} & x_1 t_4 e^{x_2 t_4} & x_1 t_5 e^{x_2 t_5} \end{pmatrix} \begin{pmatrix} x_1 e^{x_2 t_1} - y_1 \\ x_1 e^{x_2 t_2} - y_2 \\ x_1 e^{x_2 t_3} - y_3 \\ x_1 e^{x_2 t_4} - y_4 \\ x_1 e^{x_2 t_5} - y_5 \end{pmatrix} = \nabla F(x) F(x)$$
The Hession, $\nabla^2 f(x)$, is replaced with an approximation.
$$\nabla^2 f(x) = \nabla F(x) \nabla F(x)^T + \sum_{i=1}^{m} f_i(x) \nabla^2 f_i(x) \approx \nabla F(x) \nabla F(x)^T$$
So, now the formula for Newton's method becomes:
$$\nabla F(x) \nabla F(x)^T p = -\nabla F(x) F(x)$$
3. Now, Gauss-Newton method is applied with an initial guess:
$x = \begin{pmatrix} 2.50 \\ 0.25 \end{pmatrix}$

then, $F = \begin{pmatrix} -0.0838 \\ -0.1481 \\ -0.3792 \\ -0.5749 \\ -1.7864 \end{pmatrix}$ and $\nabla F(x)^T = \begin{pmatrix} 1.2840 & 3.2101 \\ 1.6487 & 8.2436 \\ 2.7183 & 27.1828 \\ 3.4903 & 43.6293 \\ 7.3891 & 147.7811 \end{pmatrix}$

So, $\nabla f(x) = \nabla F(x)F(x) = \begin{pmatrix} 16.5888 \\ 300.8722 \end{pmatrix}$, $\nabla^2 f(x) = \nabla F(x)\nabla F(x)^T = \begin{pmatrix} 78.5367 & 1335.8479 \\ 1335.8479 & 24559.9419 \end{pmatrix}$

4. On substituting and solving for p: $p = \begin{pmatrix} 0.0381 \\ 0.0102 \end{pmatrix}$

   and the new estimate of the solution, $x \leftarrow x + p = \begin{pmatrix} 2.5381 \\ 0.2602 \end{pmatrix}$

5. This process is repeated with the updated value of x. The value for the least square objective function at each iteration is shown below.

k	$f(x_k)$	$\|\nabla f(x_k)\|$
0	2×10^{0}	3×10^{2}
1	4×10^{-3}	2×10^{2}
2	2×10^{-9}	3×10^{-2}
3	3×10^{-9}	4×10^{-8}
4	3×10^{-9}	3×10^{-13}

   Since $f(x) \approx 0$, at the 4th iteration with the solution $x = \begin{pmatrix} 2.5411 \\ 0.2595 \end{pmatrix}$, the approximate global solution has been found and the algorithm ends.