Analytics and Systems of Big Data
Exploratory Assignment

M.Vinitha (COE16B042)

May, 2020

1. Hyperlink Induced Topic Search (HITS)

- 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.

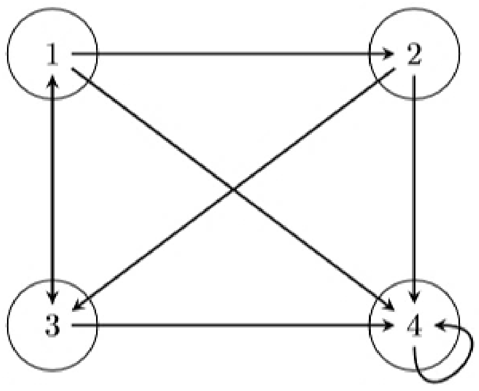
1.1 Pseudo Code

```
1. G := set of pages
2. for each page p in G do
    a. p.auth = 1 // p.auth is the authority score of the page p
    b. p.hub = 1 // p.hub is the hub score of the page p
3. for step from 1 to k do // run the algorithm for k steps
    a. norm = 0
    b. for each page p in G do  // update all authority values first
        i. p.auth = 0
        ii. for each page q in p.incomingNeighbors do
            1. p.auth += q.hub
        iii. norm += square(p.auth)
    c. norm = sqrt(norm)
    d. for each page p in G do  // update the auth scores
        i. p.auth = p.auth / norm  // normalise the auth values
    e. norm = 0
    f. for each page p in G do  // then update all hub values
        i. p.hub = 0
        ii. for each page r in p.outgoingNeighbors do
```

```
            1. p.hub += r.auth
        iii. norm += square(p.hub) // calculate the sum of the squared hub value
    g. norm = sqrt(norm)
    h. for each page p in G do  // then update all hub values
        i. p.hub = p.hub / norm   // normalise the hub values
```

1.2 Trace

The input graph is given below.



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

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

Iteration k=1:

1. Update authority scores.

   $a = A^Th$

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

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.

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

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

Iteration k=2:

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. Normalization.

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.

Iteration k=3:

1. Update authority scores.

a = A^Th

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. Normalization.

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 normalized 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 normalized values of the hub and authority scores converge and do not change after some point.

2. BIRCH and DBSCAN clustering algorithms

2.1 BIRCH Algorithm

- Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) is designed for clustering a large amount of numeric data by integrating 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 in agglomerative clustering methods: 1. scalability and 2. the inability to undo what was done in the previous step.
- BIRCH uses the notions of clustering feature to summarize a cluster, and clustering feature tree (CF-tree) to represent a cluster hierarchy.
- Given n d-dimensional data objects or points in a cluster, we can define the centroid x0 , radius R, and diameter D of the cluster as follows:

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

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

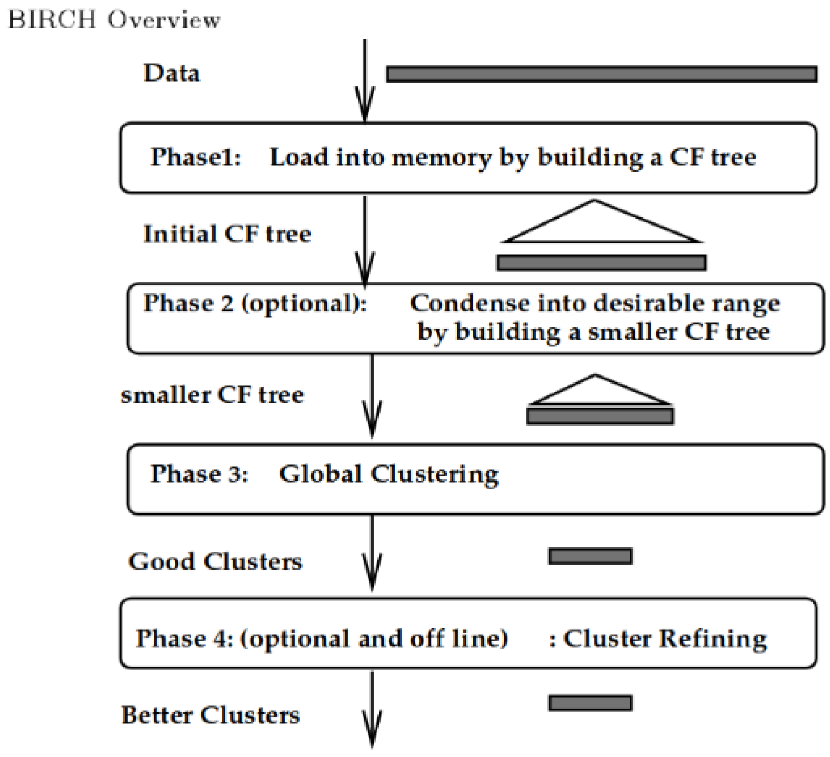
- **CF Definition** : *Given N d-dimensional data points in a cluster:* $\{\vec{X_i}\}$ *where* $i = 1, 2, ..., N$, *the* **Clustering Feature (CF)** *entry of the cluster is defined as a triple:* $\mathbf{CF} = (N, \vec{LS}, SS)$, *where* N *is the number of data points in the cluster,* $\vec{LS}$ *is the linear sum of the* N *data points, i.e.,* $\sum_{i=1}^{N} \vec{X_i}$, *and* SS *is the square sum of the* N *data points, i.e.,* $\sum_{i=1}^{N} \vec{X_i}^2$.

- **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.

2.1.1 Pseudo Code

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 localized 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.


BIRCH Overview
Data
Phase1: Load into memory by building a CF tree
Initial CF tree
Phase 2 (optional): Condense into desirable range by building a smaller CF tree
smaller CF tree
Phase 3: Global Clustering
Good Clusters
Phase 4: (optional and off line) : Cluster Refining
Better Clusters

2.1.2 Trace

- Data Points:

x	y
3	4
2	6
4	5
4	7
3	8

- N = 5
- NS = (16, 30) i.e (3 + 2 + 4 + 4 + 3, 4 + 6 + 5 + 7 + 8)
- SS = (54, 190) i.e ($3^2 + 2^2 + 4^2 + 4^2 + 3^2$, $4^2 + 6^2 + 5^2 + 7^2 + 8^2$)

CF =<5, (16,30), (54,190)>

2.2 DBSCAN Algorithm

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 neighborhood 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 neighborhood 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 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 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.

2.2.1 Pseudo Code

Algorithm: DBSCAN: a density-based clustering algorithm.

Input:

- D: a data set containing n objects,
- ϵ : the radius parameter, and
- MinPts: the neighborhood density threshold.

Output: A set of density-based clusters.

Method:

```
(1) mark all objects as unvisited;
(2) do
(3)         randomly select an unvisited object p;
(4)         mark p as visited;
(5)         if the ϵ -neighborhood of p has at least MinPts objects
(6)                 create a new cluster C, and add p to C;
(7)                 let N be the set of objects in the ϵ-neighborhood of p;
(8)                 for each point p' in N
(9)                         if p' is unvisited
(10)                                mark p 'as visited;
(11)                                if the ϵ-neighborhood of p ' has at least MinPts points,
                                    add those points to N;
(12)                        if p' is not yet a member of any cluster, add p ' to C;
(13)                end for
(14)                output C;
(15)        else mark p as noise;
(16) until no object is unvisited;
```

2.2.2 Trace

Consider the distance matrix for certain data points.

	A	B	C	D	E	F
A	0	0.7	5.7	3.6	4.2	3.2
B	0.7	0	4.9	2.9	3.5	2.5
C	5.7	4.9	0	2.2	1.4	2.5
D	3.6	2.9	2.2	0	1	0.5
E	4.2	3.5	1.4	1	0	1.1
F	3.2	2.5	2.5	0.5	1.1	0

- The above points need to be classified into Boundary point (B), Core point (C) and Noise (N).
- For this trace, min = 3 and eps = 1.5.

- Looking at the table and calculating minimum number of points for each row:
  - A has 2 minimum points which is lesser than min hence it is either B/N
  - B has 2 minimum points which is lesser than min hence it is either B/N
  - C has 2 minimum points which is lesser than min hence it is either B/N
  - D has 3 minimum points which is greater than or equal to min hence it is C
  - E has 3 minimum points which is greater than or equal to min hence it is C
  - F has 3 minimum points which is greater than or equal to min hence it is C
- Now, looking at C, it has a minimum value of 1.4 other than 0 which corresponds to E which is a Core Point. Hence C is near to E which makes it a Boundary point.
- Now, looking at B, it has a minimum value of 0.7 other than 0 which corresponds to A which hasn't been visited yet. Hence B is Noise.
- Now, looking at A, it has a minimum value of 0.7 other than 0 which corresponds to B which is a Noise Point. Hence A is Noise.

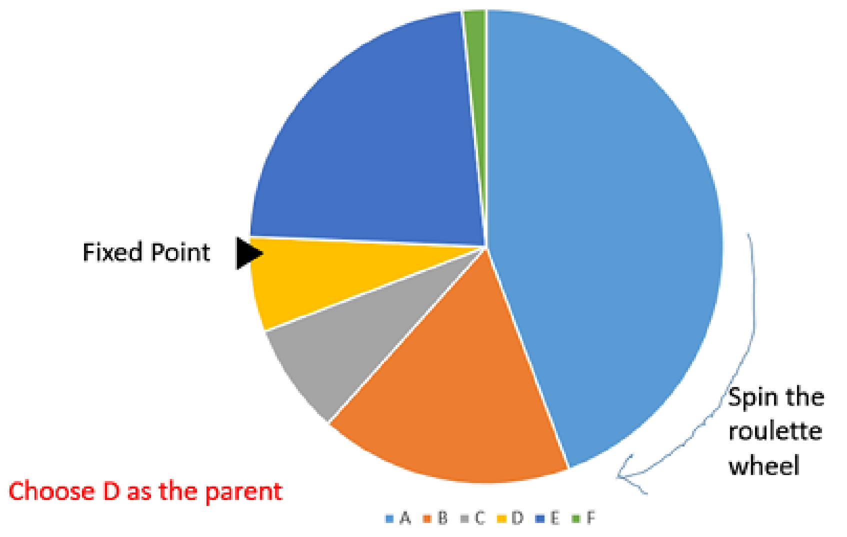
3. GA Operators

3.1 Selection

Selection is the process of choosing two parents from the population for crossing. The primary objective of the selection operator is to emphasize the good solutions and eliminate the bad solutions in a population while keeping the population size constant.

3.1.1 Roulette Wheel Selection

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. 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

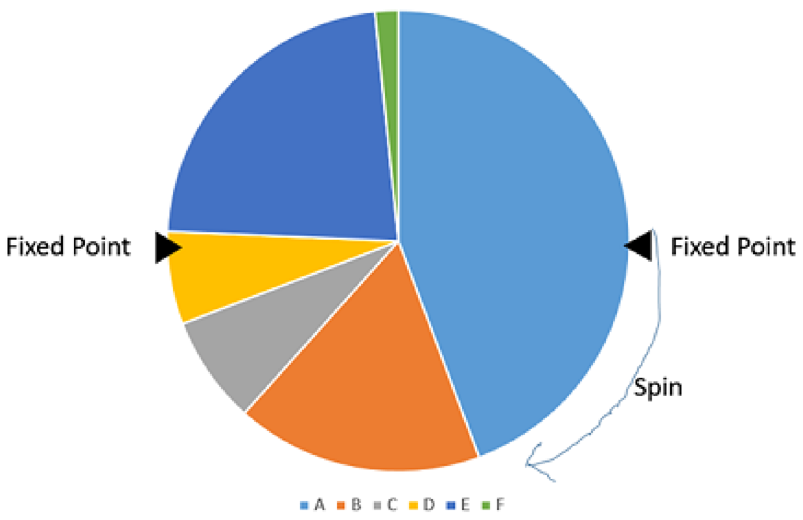
It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated. Therefore, the probability of choosing an individual depends directly on its fitness.

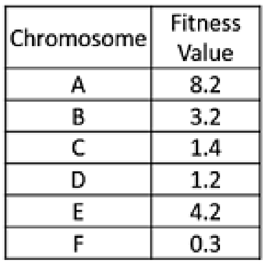
Implementation wise, we use the following steps –

- Calculate S = the sum of a finesses.
- 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.

3.1.2 Stochastic Universal Selection

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.

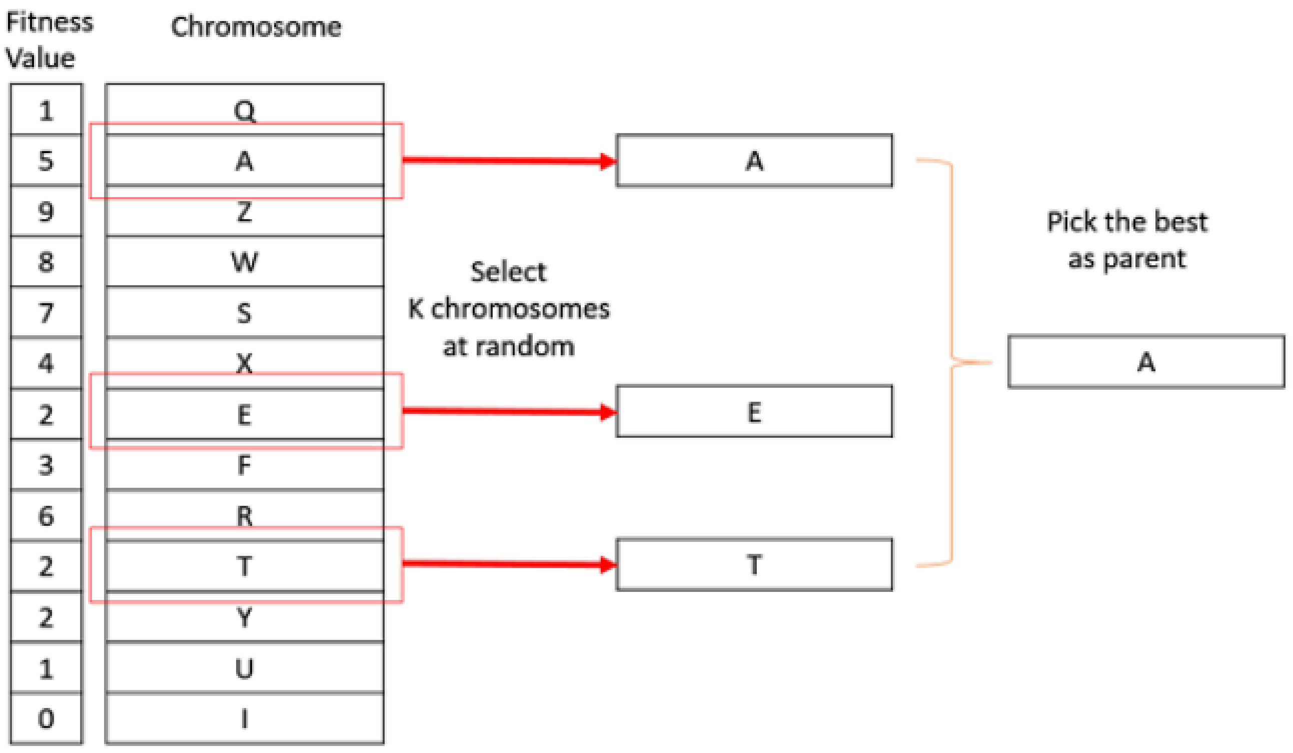




Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

3.1.3 Tournament Selection

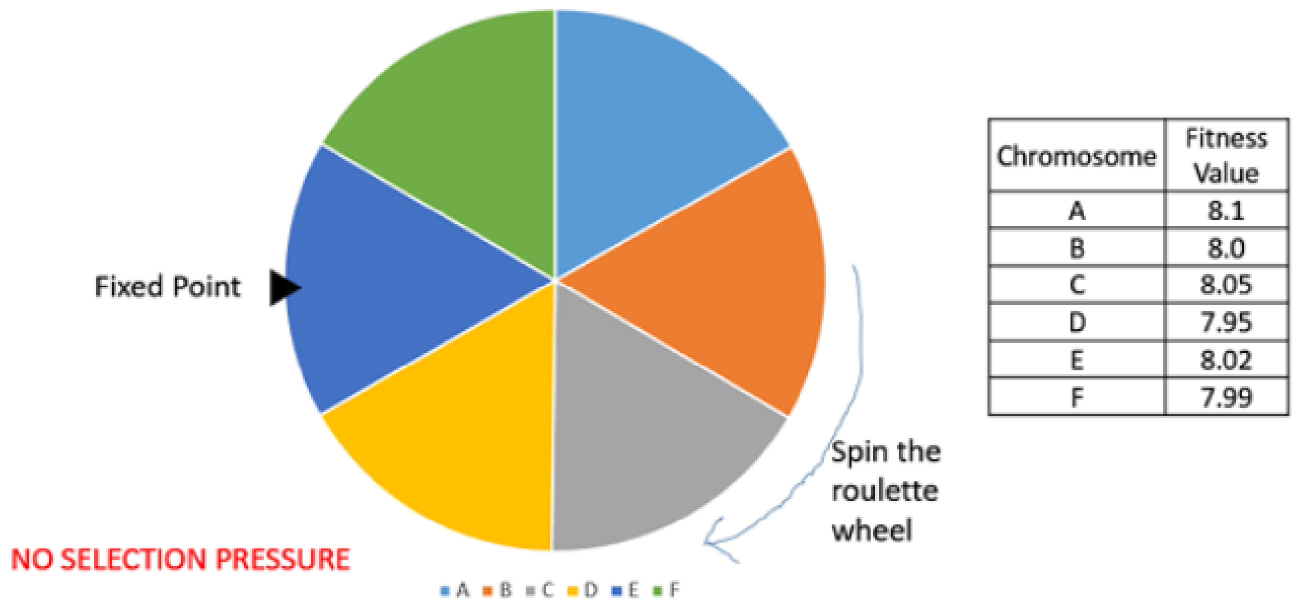
In K-Way tournament selection, we select K individuals from the population at random and select the best out of these to become a parent. The same process is repeated for selecting the next parent. Tournament Selection is also extremely popular in literature as it can even work with negative fitness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



Fitness Value	Chromosome
1	Q
5	A
9	Z
8	W
7	S
4	X
2	E
3	F
6	R
2	T
2	Y
1	U
0	I

3.1.4 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

3.1.5 Steady State Selection

- This is not particular method of selecting parents. Main idea of this selection is that big part of chromosomes should survive to next generation.
- GA then works in a following way. In every generation are selected a few (good - with high fitness) chromosomes for creating a new offspring.
- Then some (bad - with low fitness) chromosomes are removed and the new offspring is placed in their place.
- The rest of population survives to new generation.

3.1.6 Random Selection

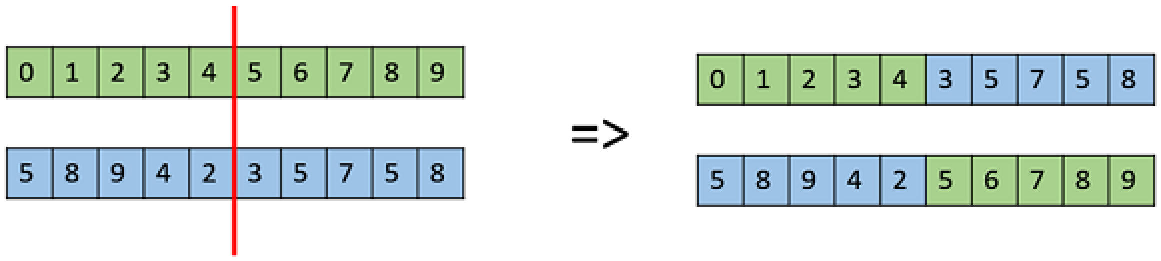
In this strategy we randomly select parents from the existing population. There is no selection pressure towards fitter individuals and therefore this strategy is usually avoided.

3.2 Crossover

The crossover operator is used to create new solutions from the existing solutions available in the mating pool after applying selection operator. This operator exchanges the gene information between the solutions in the mating pool. In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents.

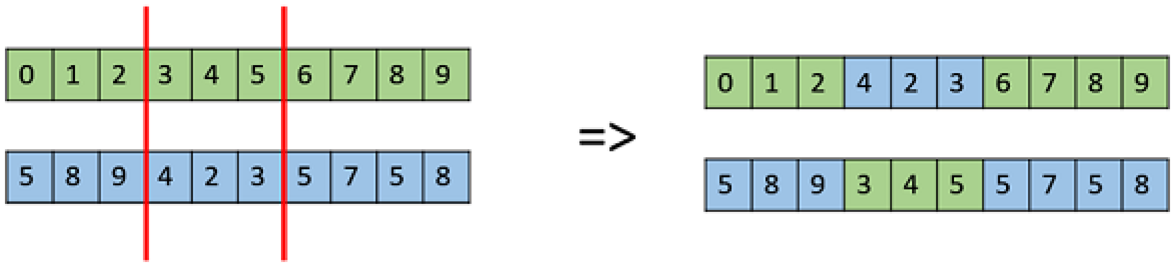
3.2.1 One Point Crossover

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.



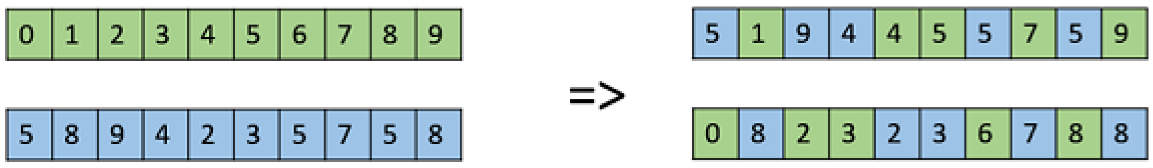
3.2.2 Multi Point Crossover

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



3.2.3 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.

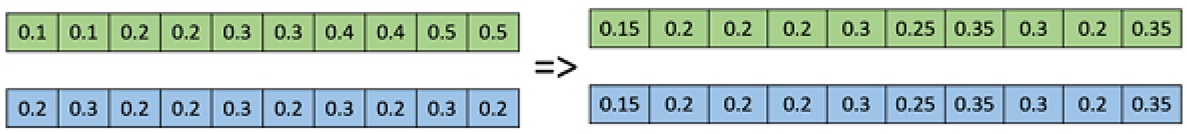


3.2.4 Whole Arithmetic Recombination

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

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

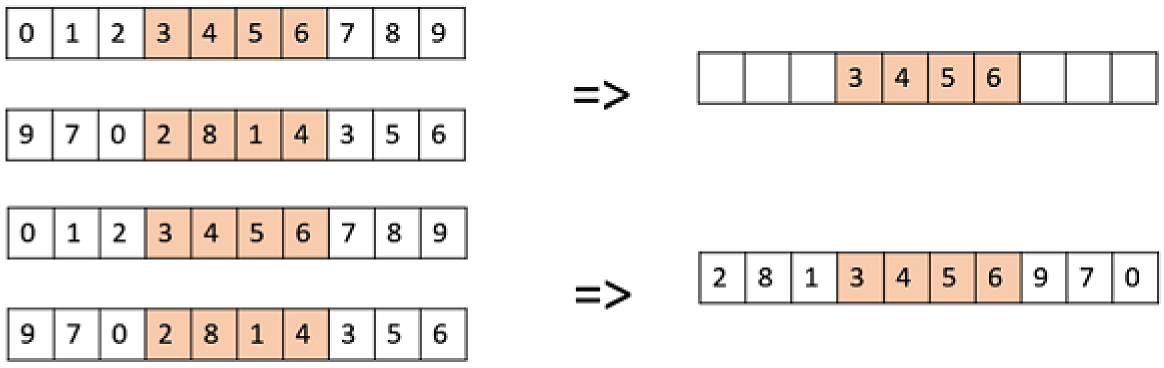
Obviously, if $\alpha = 0.5$, then both the children will be identical as shown in the following image.



3.2.5 Davi'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.



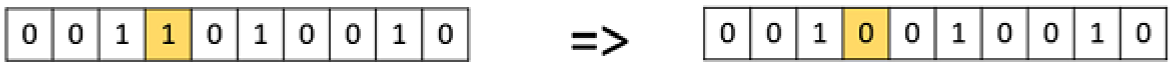
Repeat the same procedure to get the second child

3.3 Mutation

Mutation is the occasional introduction of new features in to the solution strings of the population pool to maintain diversity in the population.

3.3.1 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.

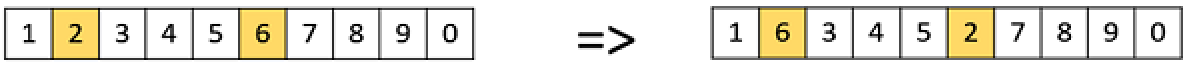


3.3.2 Random Resetting

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.

3.3.3 Swap Mutation

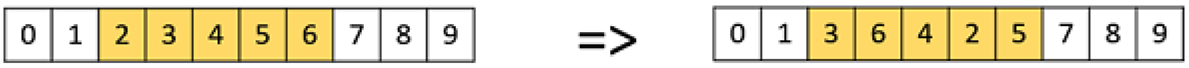
In swap mutation, we select two positions on the chromosome at random, and interchange the values. This is common in permutation-based encodings.



1	**2**	3	4	5	**6**	7	8	9	0	=>	1	**6**	3	4	5	**2**	7	8	9	0

3.3.4 Scramble Mutation

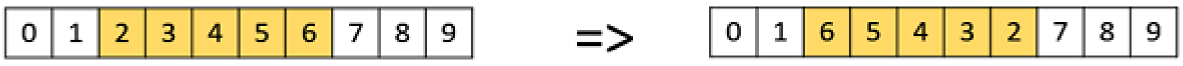
Scramble mutation is also popular with permutation representations. In this, from the entire chromosome, a subset of genes is chosen and their values are scrambled or shuffled randomly.



0	1	**2**	**3**	**4**	**5**	**6**	7	8	9	=>	0	1	**3**	**6**	**4**	**2**	**5**	7	8	9

3.3.5 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.

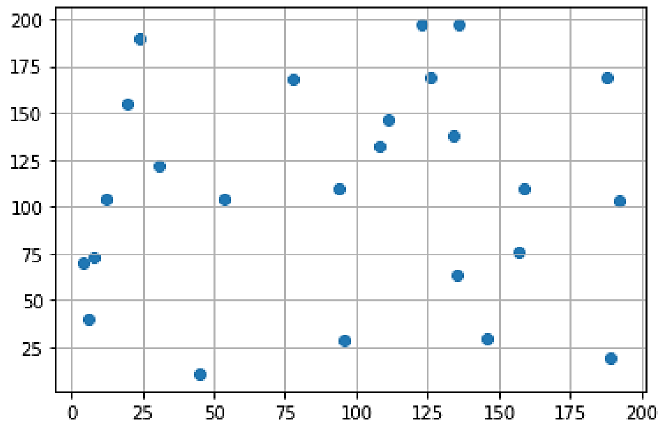


0	1	**2**	**3**	**4**	**5**	**6**	7	8	9	=>	0	1	**6**	**5**	**4**	**3**	**2**	7	8	9

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 Initialization: Let us consider a population size of 100. 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. 100 routes are randomly chosen.

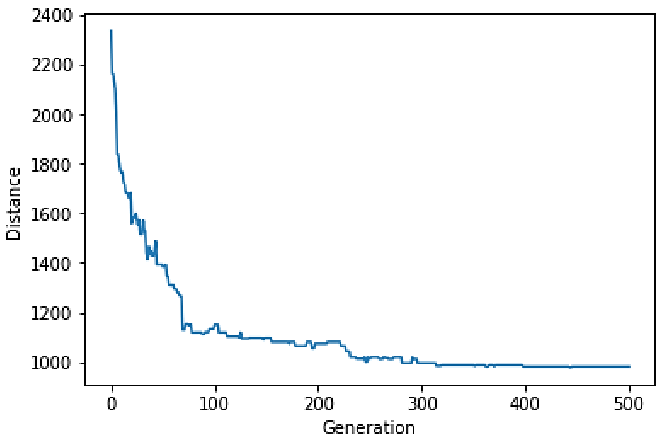
Selection: Roulette wheel selection method is used along with elitism. Elitism is used to hold on to the best routes. Let the elite size be 20. So, due to elitism the top 20 routes will definitely be included in the next generation. The remaining 80 parents are chosen using roulette wheel method from the entire population of size 100.

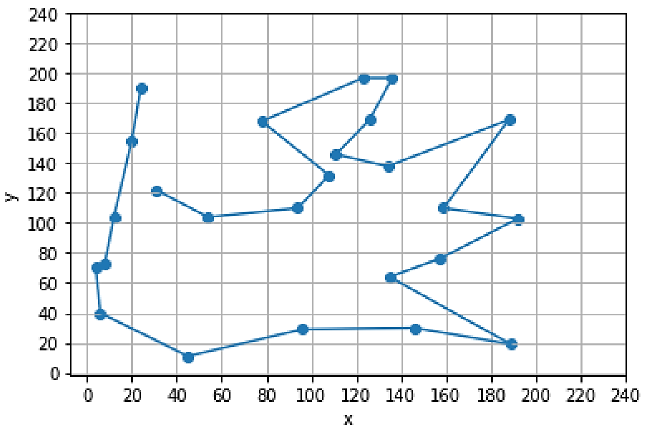
Crossover: Cities should not repeat and all cities should appear in the route. This calls for a permutation-based crossover. So, we will use Ordered crossover (OX1/Davis Order Crossover).

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 Distance: 2250.8162152201808, Final Distance: 1010.580541980844





5. SLIQ and ARBC Classifiers

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:

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

- The gini index for a binary split of set L into subsets L1 and L2 will be:

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

- **Minimum Description Length (MDL):** The best model for a given data set minimizes 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)

5.1.1 Pseudo Code

```
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).
```

5.1.2 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 ≤ v. Histograms are attached to each leaf node to help in evaluating a split. For each value in 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 initialized 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 ≤ 35.
Consider the first possible split, Age ≤ 23.

N1 Before split			N1 After split		
	B	G		B	G
L	0	0	L	1	0
R	2	4	R	1	4

Gini(L) = 1 – ((LB/LB + LG)^2 + (LG/LB + LG)^2)
Gini(R) = 1 – ((RB/RB + RG)^2 + (RG/RB + RG)^2)
Gini(Condition) = (LB + LG / LB + LG + RB + RG) * Gini(L) +
(RB + RG / RB + RG + LB + LG) * Gini(R)
Gini(L) = 1 – (1^2 + 0^2) = 0
Gini(R) = 1 – (0.5^2 + 0.8^2) = 0.32
Gini(Age < 23) = 1/6 * 0 + 5/6 + 0.32 = 0.27

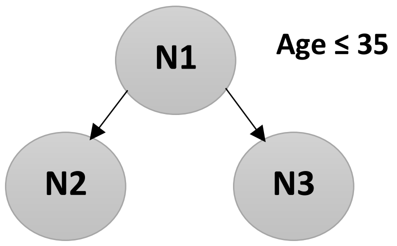
Repeat for all remaining possible splits.

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 ≤ 40. But if we select Salary ≤ 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 will consider the same assumption given in the paper i.e. Age ≤ 35. The corresponding histogram of N1 is:

N1 Before split			N1 After split		
	B	G		B	G
L	0	0	L	1	1
R	2	4	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 ≤ 35. Traverse the Age list and update the class list.



Age List	
Age	Class List Index
23	2
30	1
40	3
45	6
55	5
55	4

Class List		
Class List Index	Class	Leaf
1	G	N2
2	B	N2
3	G	N3
4	B	N3
5	G	N3
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	
Salary	Class List Index
15	2
65	1

Class List		
Class List Index	Class	Leaf
1	G	N2
2	B	N2

But there is only one condition possible i.e. Salary ≤ 15. The corresponding histogram of N2 is:

N2 Before split		
	B	G
L	0	0
R	1	1

N2 After split		
	B	G
L	1	0
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	
Salary	Class List Index
40	4
60	6
75	3
100	5

Class List		
Class List Index	Class	Leaf
3	G	N3
4	B	N3
5	G	N3
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 ≤ 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 ≤ 15.

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

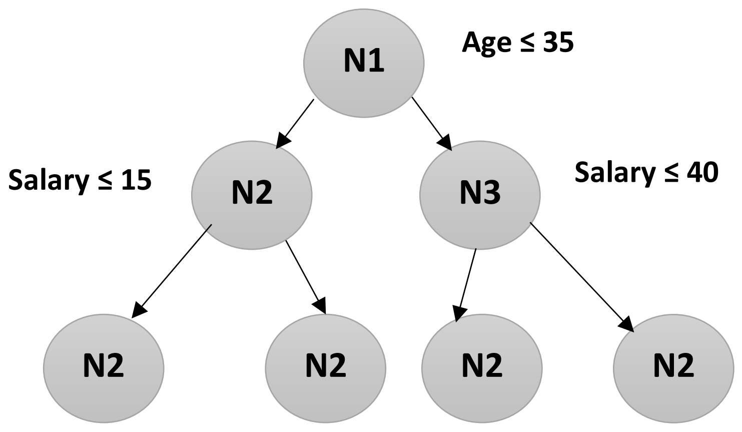
N3 is split into N6 and N7 with the condition Salary ≤ 40.

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 is a rule-based machine learning method for discovering interesting relations between variables in large databases.
- It is intended to identify strong rules discovered in databases using some measures of interestingness.
- Based on the concept of strong rules, Rakesh Agrawal, Tomasz Imielin´ski and Arun Swami introduced association rules for discovering regularities between products in large-scale transaction data recorded by point-of-sale (POS) systems in supermarkets.
- In addition to the above example from market basket analysis association rules are employed today in many application areas including Web usage mining, intrusion detection, continuous production, and bioinformatics.
- In contrast with sequence mining, association rule learning typically does not consider the order of items either within a transaction or across transactions.

5.2.1 Pseudo Code

```
Apriori($T, \epsilon$)
    $L_1 \leftarrow \{\text{large } 1 - \text{itemsets}\}$
    $k \leftarrow 2$
    while $L_{k-1} \neq \emptyset$
        $C_k \leftarrow \{c = a \cup \{b\} \mid a \in L_{k-1} \land b \notin a, \{s \subseteq c \mid |s| = k-1\} \subseteq L_{k-1}\}$
        for transactions $t \in T$
            $D_t \leftarrow \{c \in C_k \mid c \subseteq t\}$
            for candidates $c \in D_t$
                $count[c] \leftarrow count[c] + 1$
        $L_k \leftarrow \{c \in C_k \mid count[c] \geq \epsilon\}$
        $k \leftarrow k + 1$
    return $\bigcup_k L_k$
```

5.2.2 Trace

Assume that a large supermarket tracks sales data by stock-keeping unit (SKU) for each item: each item, such as “butter” or “bread”, is identified by a numerical SKU. The supermarket has a database of transactions where each transaction is a set of SKUs that were bought together.

- Let the database of transactions consist of following item sets:

Itemsets
{1,2,3,4}
{1,2,4}
{1,2}
{2,3,4}
{2,3}
{3,4}
{2,4}

- An item set is frequent if it appears in at least 3 transactions of the database: the value 3 is the support threshold.
- The first step of Apriori is to count up the number of occurrences, called the support, of each member item separately.
- By scanning the database for the first time, the following result is obtained.

Item	Support
{1}	3
{2}	6
{3}	4
{4}	5

- All the itemsets of size 1 have a support of at least 3, so they are all frequent.
- The next step is to generate a list of all pairs of the frequent items.

Item	Support
{1,2}	3
{1,3}	1
{1,4}	2
{2,3}	3
{2,4}	4
{3,4}	3

- The pairs {1,2}, {2,3}, {2,4}, and {3,4} all meet or exceed the minimum support of 3, so they are frequent.
- The pairs {1,3} and {1,4} are not.
- Now, because {1,3} and {1,4} are not frequent, any larger set which contains {1,3} or {1,4} cannot be frequent.

Item	Support
{2,3,4}	2

- In the example, there are no frequent triplets. {2,3,4} is below the minimal threshold, and the other triplets were excluded because they were super sets of pairs that were already below the threshold.

6. Sequence Pattern Mining

- Sequential pattern mining 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.

6.1 GSP

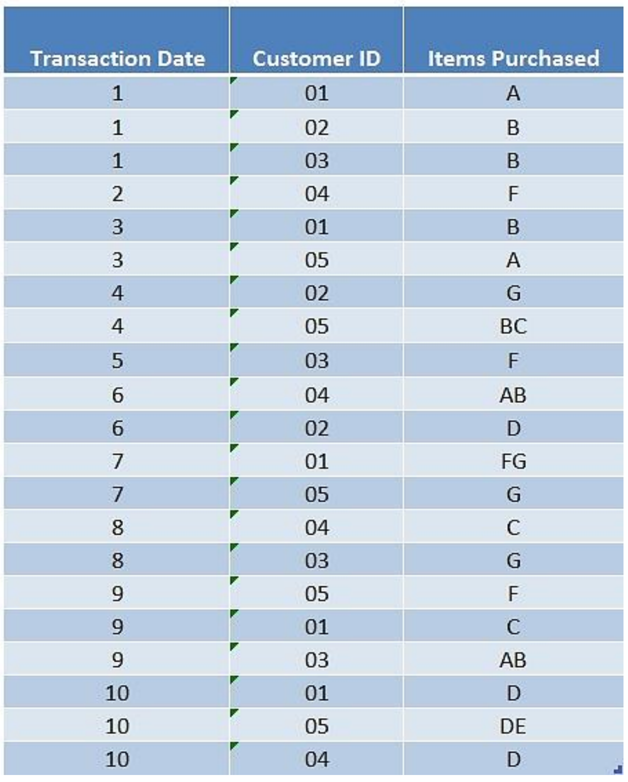
- GSP algorithm (Generalized Sequential Pattern algorithm) is an algorithm used for sequence mining.
- The algorithms for solving sequence mining problems are mostly based on the apriori (level-wise) algorithm.
- One way to use the level-wise paradigm is to first discover all the frequent items in a level-wise fashion.
- It simply means counting the occurrences of all singleton elements in the database. Then, the transactions are filtered by removing the non-frequent items.
- At the end of this step, each transaction consists of only the frequent elements it originally contained.
- This modified database becomes an input to the GSP algorithm. This process requires one pass over the whole database.

6.1.1 Pseudo Code

```
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.
```

6.1.2 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

- Sort the table first by customer ID, and then by transaction time stamp. The sequence for each customer can be generated as:

Transaction Date	Customer ID	Items Purchased	Customer Sequence
1	01	A	<AB(FG)CD>
3	01	B	
7	01	FG	
9	01	C	
10	01	D	
1	02	B	<BGD>
4	02	G	
6	02	D	
1	03	B	<BFG(AB)>
5	03	F	
8	03	G	
9	03	AB	
2	04	F	<F(AB)CD>
6	04	AB	
8	04	C	
10	04	D	
3	05	A	<A(BC)GF(DE)>
4	05	BC	
7	05	G	
9	05	F	
10	05	DE	

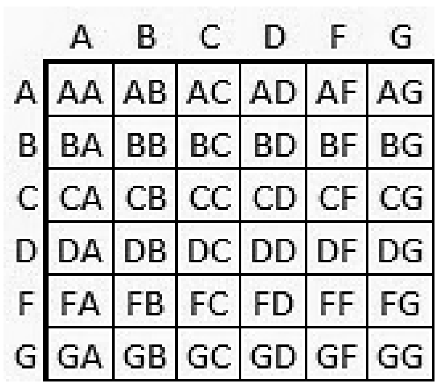
- We use minimum support as 2 for Apriori.

Item	Support
A	4
B	5
C	3
D	4
~~E~~	~~1~~
F	4
G	4

- Item E did not have enough support to meet the threshold, so it gets removed. Based on the Apriori principle, we are able to conclude that because E is not frequent, then any sequence with E in it would not be frequent either. So, we don't need to keep track of it anymore.
- The next tricky is thing is that the support for B is listed at 5, not 6, even though the 3rd customer bought item B twice. This practice is also borrowed from the Apriori algorithm. In the Apriori algorithm, if a customer buys 2 candy bars at once, then we only count 1 candy bar when calculating the support, because we count transactions. The same applies here,

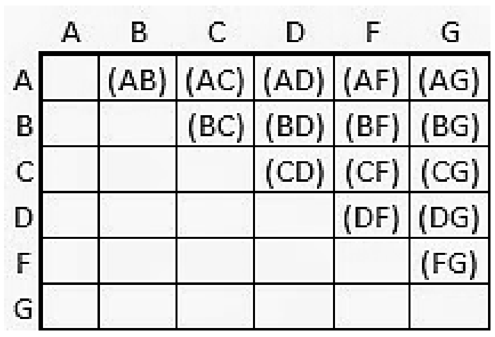
except instead of counting how many transactions contain an item, or itemset, we are counting how many customers have an item/sequence.

- Once we have these items, we start the iterative process of generating larger patterns from these items and checking if they have support. This is where things get really interesting. In Apriori, the total possible combinations would be AB, AC, AD, AF, AG, BC, BD, BF, BG, CD, CF, CG, DF, DG, FG. When generating sequences, that is not nearly sufficient because the order matters, AND we can have sequences where single items repeat (e.g. I bought A, then the next day I bought A again). So, we have to create tables that look like this:



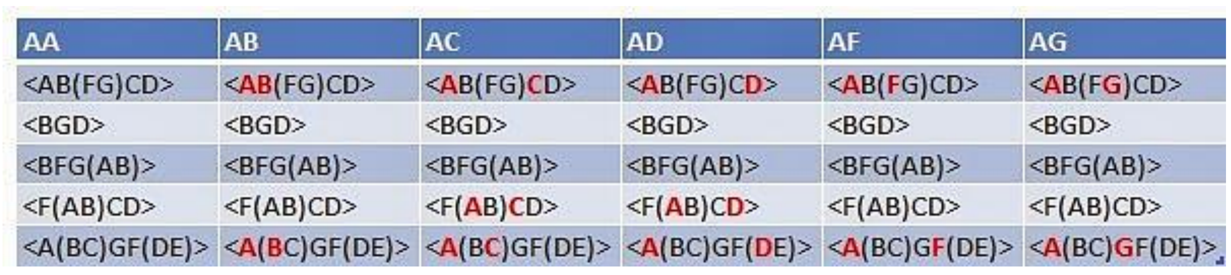
	A	B	C	D	F	G
A	AA	AB	AC	AD	AF	AG
B	BA	BB	BC	BD	BF	BG
C	CA	CB	CC	CD	CF	CG
D	DA	DB	DC	DD	DF	DG
F	FA	FB	FC	FD	FF	FG
G	GA	GB	GC	GD	GF	GG

- The diagonal values are blank here because they're already covered in the first 2-item table. Just to make sure you understand, (AA) isn't there because if A and A were bought in the same transaction it would only be counted once so we never have two of the same items together within a parenthesis. The lower left is also blank, but this is because of the convention I already described of always listing items purchased together in ascending order.



	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						

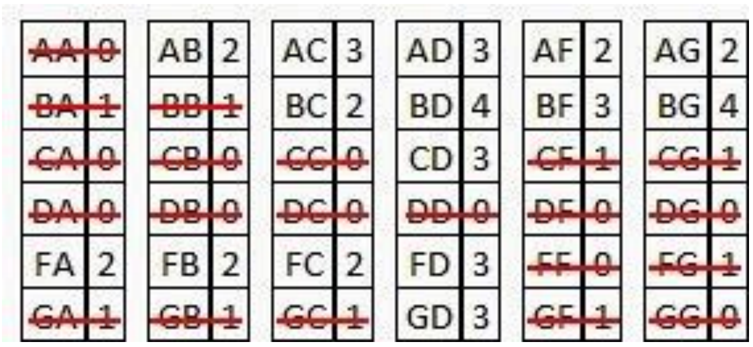
- We can check our 51 possible 2-item sequences against the database to see if they meet the support threshold.



AA	AB	AC	AD	AF	AG
<AB(FG)CD>	<AB(FG)CD>	<AB(FG)CD>	<AB(FG)CD>	<AB(FG)CD>	<AB(FG)CD>
<BGD>	<BGD>	<BGD>	<BGD>	<BGD>	<BGD>
<BFG(AB)>	<BFG(AB)>	<BFG(AB)>	<BFG(AB)>	<BFG(AB)>	<BFG(AB)>
<F(AB)CD>	<F(AB)CD>	<F(AB)CD>	<F(AB)CD>	<F(AB)CD>	<F(AB)CD>
<A(BC)GF(DE)>	<A(BC)GF(DE)>	<A(BC)GF(DE)>	<A(BC)GF(DE)>	<A(BC)GF(DE)>	<A(BC)GF(DE)>

- In the first column, we see that there are no customers that every bought item A on more than 1 occasion, so support there is 0. For <AB>, you can see that customers 1 and 5 bought those items in that sequence. For <AC>, notice that customer 4 bought (AB) together then C, but we get to pick out just A from that combined transaction to get our <AC> sequence. Notice that the items don't have to be right next to each other. It's OK to have other items in between. <AD> follows similar logic. For <AF> we don't count customers 3 or 4 because although they have A and F, they aren't in the right order. To be counted here, they need to be A, then F. When we look for the support of <FA> those will get counted.

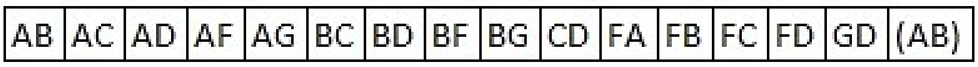
- There is one last rule you need to know for counting support of sequences. If you are looking for support of a sequence like <FG>, then customer 1 doesn't count. It only counts if F is bought before G, NOT at the same time. If you work through the rest of the 2-item sequences that weren't purchased together (i.e. not ones like (AB)), you get support counts that look like this:



~~AA~~	~~0~~	AB	2	AC	3	AD	3	AF	2	AG	2
~~BA~~	~~1~~	~~BB~~	~~1~~	BC	2	BD	4	BF	3	BG	4
~~CA~~	~~0~~	~~CB~~	~~0~~	~~CC~~	~~0~~	CD	3	~~CF~~	~~1~~	~~CG~~	~~1~~
~~DA~~	~~0~~	~~DB~~	~~0~~	~~DC~~	~~0~~	~~DD~~	~~0~~	~~DF~~	~~0~~	~~DG~~	~~0~~
FA	2	FB	2	FC	2	FD	3	~~FF~~	~~0~~	~~FG~~	~~1~~
~~GA~~	~~1~~	~~GB~~	~~1~~	~~GC~~	~~1~~	GD	3	~~GF~~	~~1~~	~~GG~~	~~0~~

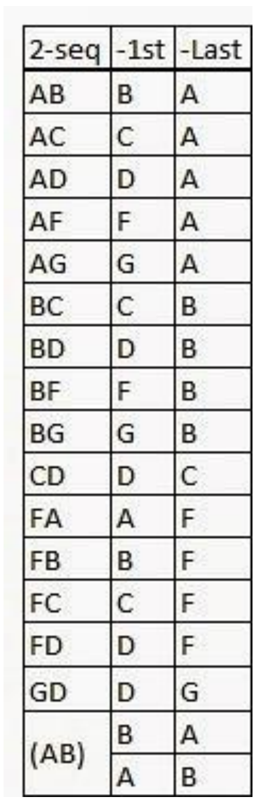
- Now let's take a look at the 2-item sets that could have been purchased together (e.g. (AB)). This one is a little bit easier, because there are only 5 times in our database when 2 items were purchased at the same time. They were (FG), (AB), (AB), (BC) and (DE). Only (AB) shows up twice so we keep it and get rid of the rest.

- Now we have these 2-item sequences left to work with:



AB	AC	AD	AF	AG	BC	BD	BF	BG	CD	FA	FB	FC	FD	GD	(AB)

- You take all of the remaining sequences from the last step (AB, AC, AD, AF, AG, BC...), then you remove the first item from the sequence.

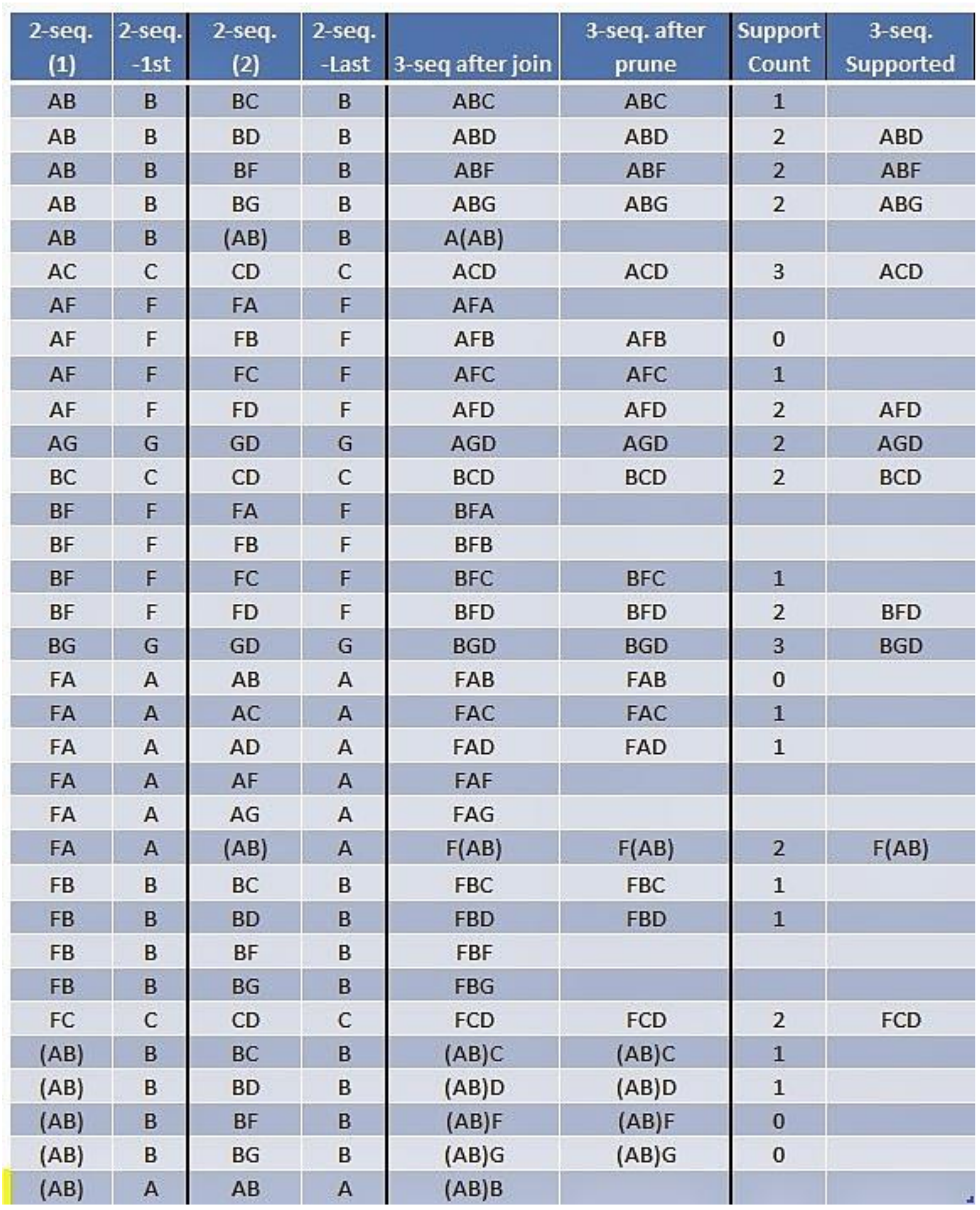


2-seq	-1st	-Last
AB	B	A
AC	C	A
AD	D	A
AF	F	A
AG	G	A
BC	C	B
BD	D	B
BF	F	B
BG	G	B
CD	D	C
FA	A	F
FB	B	F
FC	C	F
FD	D	F
GD	D	G
(AB)	B	A
	A	B

- We combine the sequences together where their "-1st" and "-Last" columns match. So, if we're starting from the top, we see that the 2-item sequence AB matches up with BC, BD, BF, BG and (AB) to create ABC, ABD, ABF, ABG and A(AB).

- The A(AB) one is tricky because you have to remind yourself that the order within the parentheses is just convention and it could easily be written A(BA) which makes it easier to see that the B's match up.

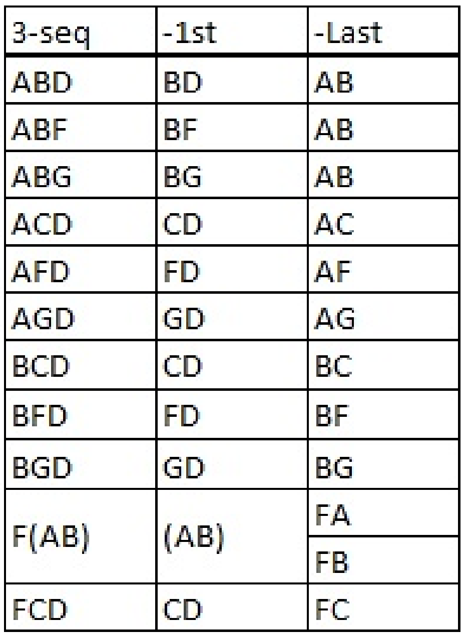
- Working through the rest of the table this way (being careful not to create any duplicate candidates) we populate the "3-seq after join" column in the table below:



2-seq. (1)	2-seq. -1st	2-seq. (2)	2-seq. -Last	3-seq after join	3-seq. after prune	Support Count	3-seq. Supported
AB	B	BC	B	ABC	ABC	1	
AB	B	BD	B	ABD	ABD	2	ABD
AB	B	BF	B	ABF	ABF	2	ABF
AB	B	BG	B	ABG	ABG	2	ABG
AB	B	(AB)	B	A(AB)			
AC	C	CD	C	ACD	ACD	3	ACD
AF	F	FA	F	AFA			
AF	F	FB	F	AFB	AFB	0	
AF	F	FC	F	AFC	AFC	1	
AF	F	FD	F	AFD	AFD	2	AFD
AG	G	GD	G	AGD	AGD	2	AGD
BC	C	CD	C	BCD	BCD	2	BCD
BF	F	FA	F	BFA			
BF	F	FB	F	BFB			
BF	F	FC	F	BFC	BFC	1	
BF	F	FD	F	BFD	BFD	2	BFD
BG	G	GD	G	BGD	BGD	3	BGD
FA	A	AB	A	FAB	FAB	0	
FA	A	AC	A	FAC	FAC	1	
FA	A	AD	A	FAD	FAD	1	
FA	A	AF	A	FAF			
FA	A	AG	A	FAG			
FA	A	(AB)	A	F(AB)	F(AB)	2	F(AB)
FB	B	BC	B	FBC	FBC	1	
FB	B	BD	B	FBD	FBD	1	
FB	B	BF	B	FBF			
FB	B	BG	B	FBG			
FC	C	CD	C	FCD	FCD	2	FCD
(AB)	B	BC	B	(AB)C	(AB)C	1	
(AB)	B	BD	B	(AB)D	(AB)D	1	
(AB)	B	BF	B	(AB)F	(AB)F	0	
(AB)	B	BG	B	(AB)G	(AB)G	0	
(AB)	A	AB	A	(AB)B			

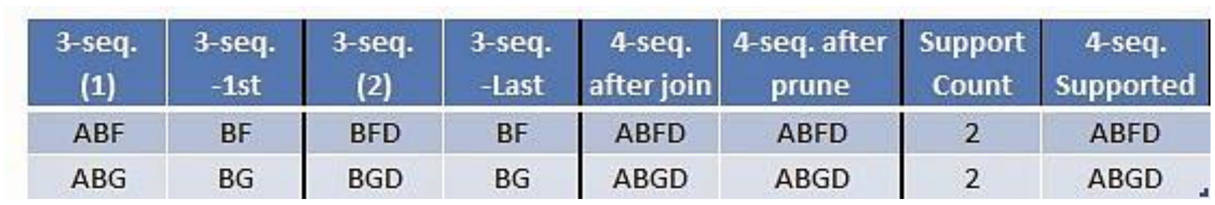
- The next step is to prune these 3-item sequences. This pruning is based on the same Apriori principle that we have used in prior posts. This pruning helps save computing time because we don't want to comb through the database to find the support for a sequence that we know won't meet the support threshold. Now that we have this slimmed down list of 3-item sequences, we scan the database again to determine support just as it was described above for 2-item sequences.

- When that is completed, we end up with the 3-item sequences in the last column of the table above. We remove the first and last items and put them in the "-1st" and "-Last" columns below.



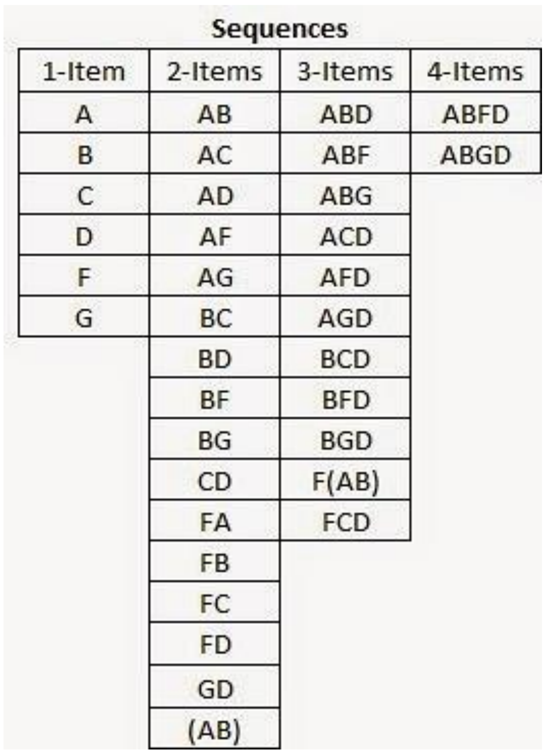
3-seq	-1st	-Last
ABD	BD	AB
ABF	BF	AB
ABG	BG	AB
ACD	CD	AC
AFD	FD	AF
AGD	GD	AG
BCD	CD	BC
BFD	FD	BF
BGD	GD	BG
F(AB)	(AB)	FA
		FB
FCD	CD	FC

- Then we look for sequences that have a "-1st" value that match a "-Last" value of another sequence. In this case we only find matches for "BF" and "BG". You should prune the sequences if they aren't supported by 3-item sequences that are supported/frequent, but in this case they're fine. Then we check the database for support to finalize the list. If you do that you end up with information similar to the table below.



3-seq. (1)	3-seq. -1st	3-seq. (2)	3-seq. -Last	4-seq. after join	4-seq. after prune	Support Count	4-seq. Supported
ABF	BF	BFD	BF	ABFD	ABFD	2	ABFD
ABG	BG	BGD	BG	ABGD	ABGD	2	ABGD

- You can see that, at this point, this example gets pretty simple. If you try the iteration again, the trick of removing the first item and last item yields no matches and you're done. After that you would just output all of the supported/frequent sequences to the user, which in this case would be the following:

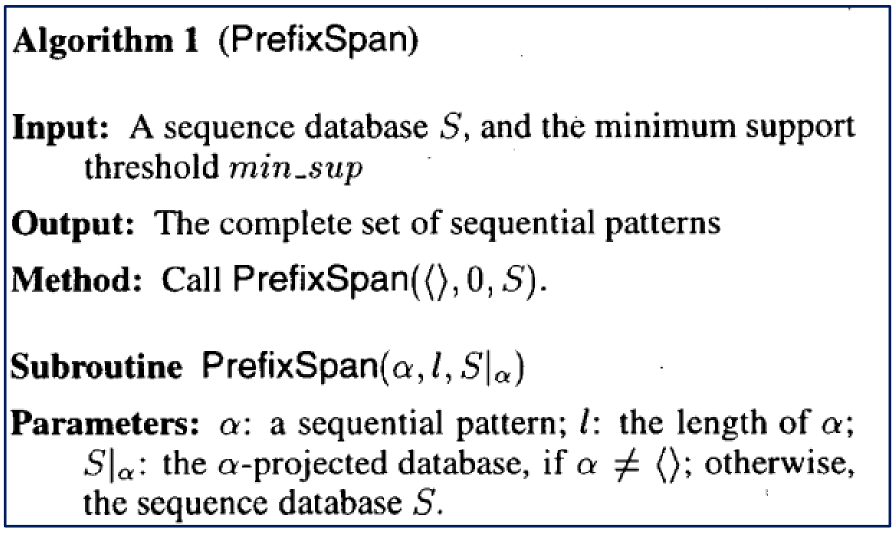


Sequences			
1-Item	2-Items	3-Items	4-Items
A	AB	ABD	ABFD
B	AC	ABF	ABGD
C	AD	ABG	
D	AF	ACD	
F	AG	AFD	
G	BC	AGD	
	BD	BCD	
	BF	BFD	
	BG	BGD	
	CD	F(AB)	
	FA	FCD	
	FB		
	FC		
	FD		
	GD		
	(AB)		

6.2 Prefix-Span Algorithm

- Prefix-Span (Prefix-projected Sequential pattern mining) algorithm is a very well-known algorithm for sequential data mining.
- It extracts the sequential patterns through pattern growth method.
- The algorithm performs very well for small datasets. As the size of datasets increases the overall time for finding the sequential patterns also get increased.

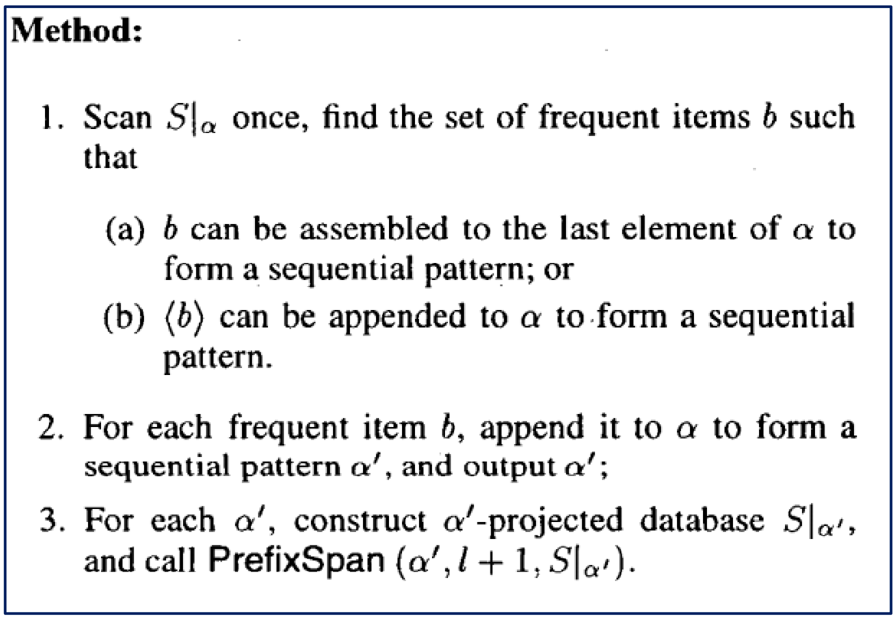
6.2.1 Pseudo Code



```
Algorithm 1 (PrefixSpan)

Input: A sequence database $S$, and the minimum support
    threshold $min\_sup$
Output: The complete set of sequential patterns
Method: Call PrefixSpan($\langle\rangle, 0, S$).

Subroutine PrefixSpan($\alpha, l, S|_\alpha$)
Parameters: $\alpha$: a sequential pattern; $l$: the length of $\alpha$;
    $S|_\alpha$: the $\alpha$-projected database, if $\alpha \neq \langle\rangle$; otherwise,
    the sequence database $S$.
```



Method:

1. Scan $S|_\alpha$ 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) $\langle b \rangle$ 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 $S|_{\alpha'}$, and call PrefixSpan $(\alpha', l+1, S|_{\alpha'})$.

6.2.2 Trace

Consider the database:

SDB	
SID	**Sequence**
10	<a, (abc), (ac)>
20	<(ad), c>
30	<(ef),(ab)>

<a>Proj.SDB	
SID	**Sequence**
10	<(abc), (ac)>
20	<(_d), c>
30	<(_b)>

<b>Proj.SDB	
SID	**Sequence**
10	<(_ac), (ac)>

<c>Proj.SDB	
SID	**Sequence**
10	<(_ab), (ac)>

<ab>Proj.SDB	
SID	**Sequence**
10	<(_c), (ac)>

<ac>Proj.SDB	
SID	**Sequence**
10	<(_ab), (ac)>

7. Schemata Theorem of GA

- Holland's schema theorem, also called the fundamental theorem of genetic algorithms, is an inequality that results from coarse-graining an equation for evolutionary dynamics.
- The Schema Theorem says that short, low-order schemata with above-average fitness increase exponentially in frequency in successive generations.
- The theorem was proposed by John Holland in the 1970s. It was initially widely taken to be the foundation for explanations of the power of genetic algorithms.
- However, this interpretation of its implications has been criticized in several publications reviewed in, where the Schema Theorem is shown to be a special case of the Price equation with the schema indicator function as the macroscopic measurement.
- A schema is a template that identifies a subset of strings with similarities at certain string positions. Schemata are a special case of cylinder sets, and hence form a topological space.
- Consider binary strings of length 6. The schema 1*10*1 describes the set of all strings of length 6 with 1's at positions 1, 3 and 6 and a 0 at position 4. The * is a wildcard symbol, which means that positions 2 and 5 can have a value of either 1 or 0. The order of a schema o(H) is defined as the number of fixed positions in the template, while the defining length δ(H) is the distance between the first and last specific positions. The order of 1*10*1 is 4 and its defining length is 5.
- The fitness of a schema is the average fitness of all strings matching the schema.
- The fitness of a string is a measure of the value of the encoded problem solution, as computed by a problem-specific evaluation function. 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:

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

- Here m(H,t) is the number of strings belonging to schema H at generation t, f(H) is the observed average fitness of schema H and a_t is the observed average fitness at generation t. The probability of disruption p is the probability that crossover or mutation will destroy the schema H. It can be expressed as:

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

- where $o(H)$ is the order of the schema, l is the length of the code, p_m is the probability of mutation and p_c is the probability of crossover. So, a schema with a shorter defining length $\delta(\mathrm{H})$ is less likely to be disrupted.

7.1 Optimization function

The optimization function is,

$$\boldsymbol{x^3 - 2x^2 + x}$$

The constraint on x is $\mathbf{x} \in [\mathbf{0}, \mathbf{31}]$.

- **String Processing**

Sl	Population	x	f(x)	Probability	Expected Count	Actual Count
1	01101	13	1872	0.06	0.24	0
2	11110	30	25230	0.8	3.17	3
3	01000	8	392	0.013	0.05	0
4	10001	17	4352	0.127	0.54	1

$$\sum f(x) = 31846$$
$$avg\ f(x) = 7961.5$$
$$\max f(x) = 25230$$

- **Processing before reproduction**

Sl	Schema	String Representative	Schema Average Fitness
H1	1****	2, 4	14791
H2	*10**	3	392
H3	1***0	2	25230

$$m(H1) = \frac{(4 * 14791)}{7961.5} = 7.43$$
$$m(H2) = \frac{(0 * 392)}{7961.5} = 0$$

$$m(H3) = \frac{(3 * 25230)}{7961.5} = 9.5$$

- **Crossover**

Mating Pool	Mate	Crossover Site	New Population	X Value	f(x)
01101	2	2	11110	30	25230
11110	1	2	11110	30	25230
01000	4	2	11101	29	22736
10001	3	2	10010	18	5202

$$\sum f(x) = 78398$$
$$avg\ f(x) = 19599.5$$
$$\max f(x) = 25230$$

- **Schema processing**

  - **After Reproduction**
    For the element 11110, if mutation is performed in the bit 0, 11111 is obtained which is 31 which gives a fitness value of 27900.

  - **After all operations**

Expected Count	Actual Count	String Representatives
7.43	4	2, 4
0	0	3
9.5	3	2

- Thus, the results obtained match the results obtained earlier in the Reading Assignment.

8. Regression Models

8.1 Linear Regression

- Linear Regression is a statistical approach to modelling the relationship between an input and output as a linear relationship.
- Given a data set $\{y_i, x_{i1}, x_{i2}, \dots, x_{ip}\}$ 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,

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

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

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

8.1.1 Pseudo Code

```
1) Start
2) Read Number of Data (n)
3) For i = 1 to n:
       Read $X_i$ and $Y_i$
       Next i
4) Initialize:
       sumX = 0
       sumX2 = 0
       sumY = 0
       sumXY = 0
5) Calculate Required Sum
   For i=1 to n:
       sumX = sumX + $X_i$
       sumX2 = sumX2 + $X_i$ * $X_i$
       sumY = sumY + $Y_i$
       sumXY = sumXY + $X_i$ * $Y_i$
   Next i
6) Calculate Required Constant a and b of y = a + bx:
   b = (n * sumXY - sumX * sumY) / (n*sumX2 - sumX * sumX)
   a = (sumY - b*sumX)/n
7) Display value of a and b
8) Stop
```

8.1.2 Trace

- Consider the sample dataset.

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

- Now, we calculate sums,

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

sumY = 6 + 7 + 8 + 9 + 10 = 40

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

sumXY = 6 + 14 + 24 + 36 + 50 = 130

- Now, we find parameters,

  b = (n * sumXY - sumX * sumY)/(n*sumX2 - sumX * sumX)

  = (5*130 – 15*40) / (5*55 – 15*15)

  = (50/200)

  = 0.25

  a = (sumY - b*sumX)/n

  = (40 – 0.25*15) / 5

  = 7.25

- The data fits the line, y = 7.25 + 0.25x

8.2 Non-Linear Regression

- 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 nonlinear 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 optimization algorithms are applied to determine the best-fitting parameters.

8.2.1 Pseudo Code

1) Read the known data x_i,y_i
2) Based on the prior experience and visualization of the data(if possible), choose a function(nonlinear) to fit the data.
3) Choose a convex error function, J (like MeanSquaredError, Logloss etc..)
4) Fit the function by estimating its parameters to the given data.
5) To fine tune the parameters of the function use gradient descent method as follows,

   (a) Iterate until convergence,

$$\theta_{k+1} = \theta_k - \alpha \frac{\partial J}{\partial \theta}$$

   where θ_k is the parameter at kth iteration and α is the learning rate or convergence rate.

   (b) This method is called Gradient Descent.

8.2.2 Trace

- Consider the sample dataset.

X	1	2	3	4	5	6	7	8
Y	620	761	900	1040	1270	1590	1850	2155

- Let us try to fit the following function to the above data,

$$y = ae^{bx}$$

- Let a = 500, b = 5, then

$$y = 3e^{5x}$$

- Let the error function, J be MeanSquaredError = $0.5 * (y_i - ae^{bx_i})^2$
- Use Gradient Descent to calculate the values of parameters a,b.
- The procedure should be repeated for all samples until convergence.
- The output obtained after simulating is

$$y = 522.976e^{0.179x}$$