Machine Learning Algorithm

B ANIRUDH SRINIVSAN
COE17B019

1. Hyperlink Induced Topic Search (HITS)

1.1 Introduction

Hyperlink-Induced Topic Search (HITS; also known as hubs and authorities) is a link analysis algorithm that rates Web pages. The idea behind Hubs and Authorities stemmed from a particular insight into the creation of web pages when the Internet was originally forming; that is, certain web pages, known as hubs, served as large directories that were not actually authoritative in the information that they held, but were used as compilations of a broad catalog of information that led users direct to other authoritative pages. In other words, a good hub represents a page that pointed to many other pages, while a good authority represents a page that is linked by many different hubs.

1.2 Important Terms

Authorities

These are the page which contain useful information or the information we need.

Root set

Set of all authorities for our query is known as Root set.

Hubs

Hubs are web pages which don't carry the information for our query, but point to the page (authority) which contains the information. We need to calculate/ consider hubs as they help to define the relevance of the authority.

Base set

Set of all Hubs and Authorities for our query is called Base set.

1.3 Ranking web pages with an example

Step 1: Define the Base set

We need to define the base set with all the directed edge.

If A points to B then a directed edge should be drawn from A to B.

For our example, we take this graph.

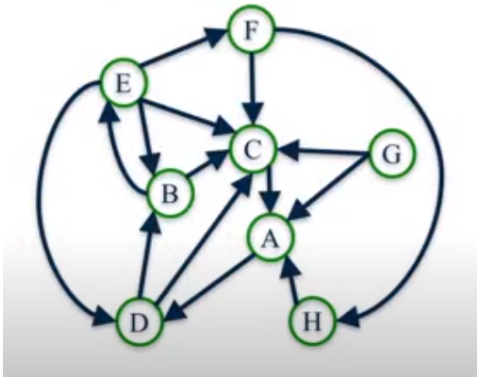

Figure 1: Example Graph

Step 2: Initialise the Hub and Authority Score for each node

We initialise the hub and authority score to 1 for every node in the graph.

	Old Auth	Old Hub
A	1	1
B	1	1
C	1	1
D	1	1
E	1	1
F	1	1
G	1	1
H	1	1

Figure 2: Inital scores

Step 3: Update Hub and Authority score

We need to update the Authority and Hub score based on the following rule :

$$new\_auth(p) = \sum_{q \in p\_hub} hub(q) \tag{1}$$

where,

- p ∈ Base set.
- p_hub is the set of all hub pointing to p.

Similarly,

$$new\_hub(p) = \sum_{q \in p\_auth} auth(q) \tag{2}$$

where,

- p ∈ Base set.
- p_auth is the set of all authorities pointed by p.

We use the old authority and hub values to update.

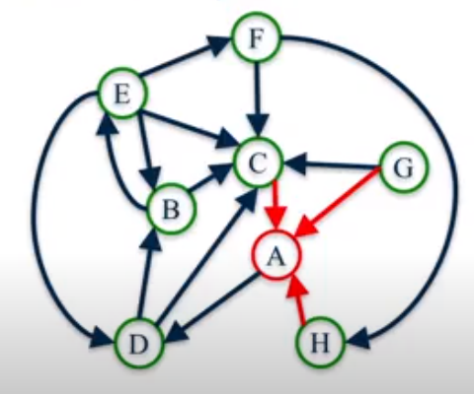

Figure 3: Calculation of Authority Value of node A.

$$\begin{aligned} auth(A) &= hub(H) + hub(G) + hub(C) \\ auth(A) &= 1 + 1 + 1 \\ auth(A) &= 3 \end{aligned}$$

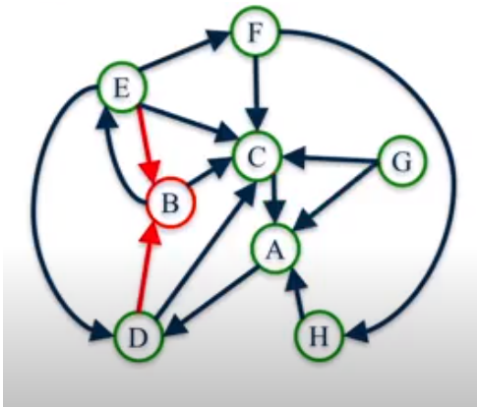

Figure 4: Calculation of Authority Value of node B.

$$auth(B) = hub(E) + hub(D) + hub(C)$$
$$auth(B) = 1 + 1$$
$$auth(B) = 2$$

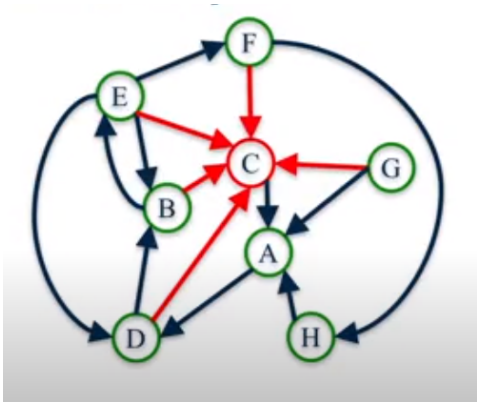

Figure 5: Calculation of Authority Value of node C.

$$auth(C) = hub(E) + hub(B) + hub(D) + hub(F) + hub(G)$$
$$auth(C) = 1 + 1 + 1 + 1 + 1$$
$$auth(C) = 5$$

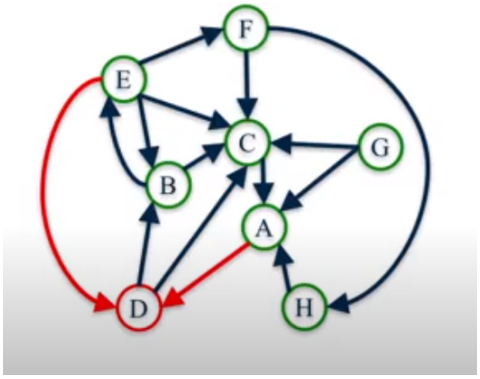

Figure 6: Calculation of Authority Value of node D.

$$auth(D) = hub(E) + hub(A)$$
$$auth(D) = 1 + 1$$
$$auth(D) = 3$$

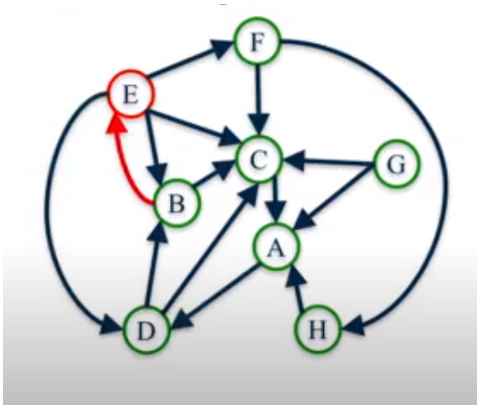

Figure 7: Calculation of Authority Value of node E.

$$auth(E) = hub(B)$$
$$auth(E) = 1$$
$$auth(E) = 1$$

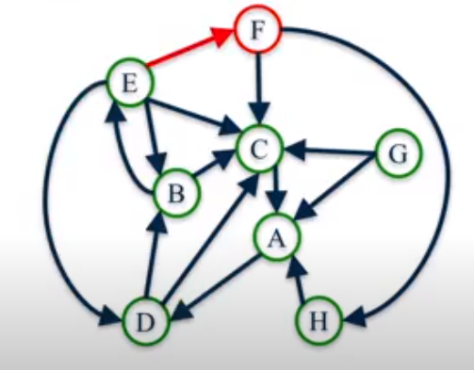

Figure 8: Calculation of Authority Value of node F.

$$auth(F) = hub(E)$$
$$auth(F) = 1$$
$$auth(F) = 1$$

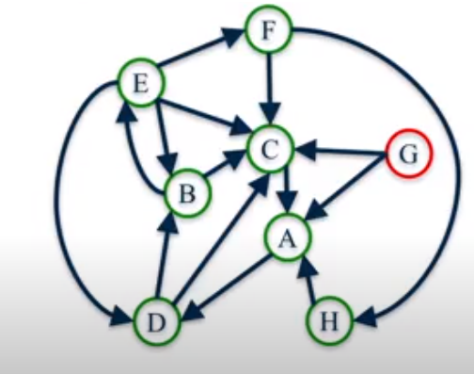

Figure 9: Calculation of Authority Value of node G.

$$auth(G) = 0$$

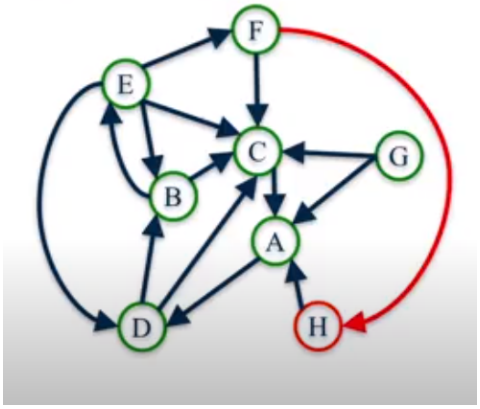

Figure 10: Calculation of Authority Value of node H.

$$auth(H) = hub(F)$$
$$auth(H) = 1$$
$$auth(H) = 1$$

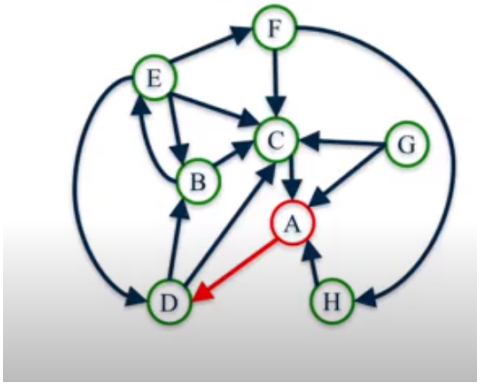

Figure 11: Calculation of Hub Value of node A.

$$hub(A) = auth(D)$$
$$hub(A) = 1$$
$$hub(A) = 1$$

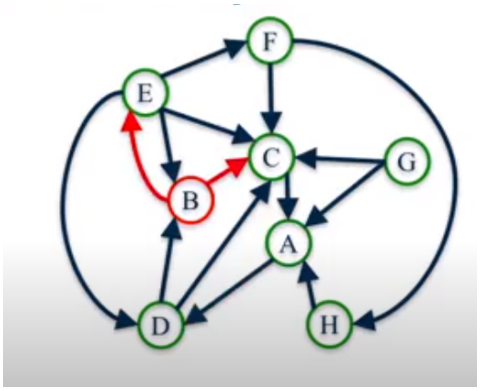

Figure 12: Calculation of Hub Value of node B.

$$hub(B) = auth(C) + auth(E)$$
$$hub(B) = 1 + 1$$
$$hub(B) = 2$$

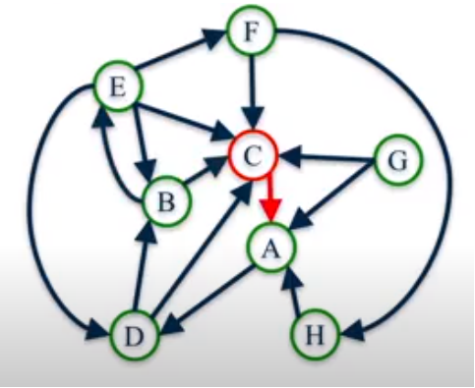

Figure 13: Calculation of Hub Value of node C.

$$hub(C) = auth(A)$$
$$hub(C) = 1$$
$$hub(C) = 1$$

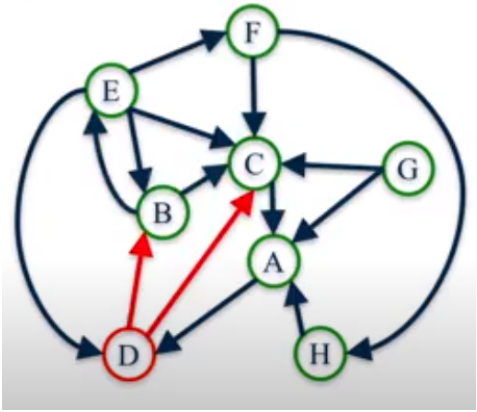

Figure 14: Calculation of Hub Value of node D.

$$hub(D) = auth(B) + auth(C)$$
$$hub(D) = 1 + 1$$
$$hub(D) = 2$$

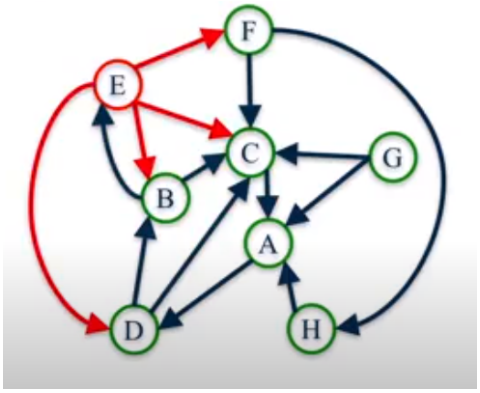

Figure 15: Calculation of Hub Value of node E.

$$hub(E) = auth(F) + auth(C) + auth(B) + auth(D)$$
$$hub(E) = 1 + 1 + 1 + 1$$
$$hub(E) = 4$$

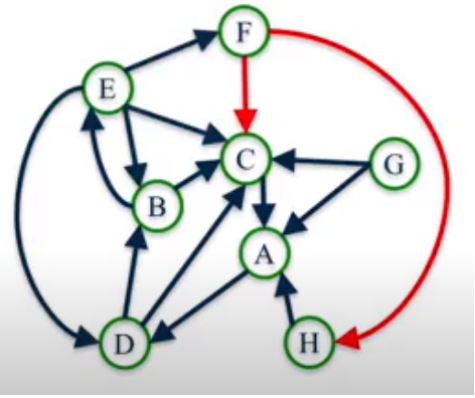

Figure 16: Calculation of Hub Value of node F.

$$hub(F) = auth(H) + auth(C)$$
$$hub(F) = 1 + 1$$
$$hub(F) = 2$$

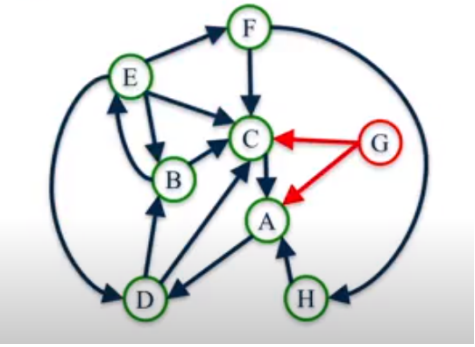

Figure 17: Calculation of Hub Value of node G.

$$hub(G) = auth(A) + auth(C)$$
$$hub(G) = 1 + 1$$
$$hub(G) = 2$$

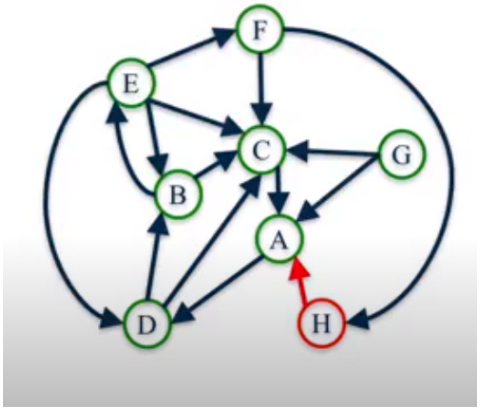

Figure 18: Calculation of Hub Value of node H.

$$hub(H) = auth(A)$$
$$hub(H) = 1$$
$$hub(H) = 1$$

To summarise the calculations,

	Old Auth	Old Hub	New Auth	New Hub
A	1	1	3	1
B	1	1	2	2
C	1	1	5	1
D	1	1	2	2
E	1	1	1	4
F	1	1	1	2
G	1	1	0	2
H	1	1	1	1

Figure 19: Table summarising the above calculations.

Step 4: Normalise

We need to normalise the updated hub and authority value with the formula,

$$auth(p) = \frac{auth(p)}{\sum_{q \in Base\ set} auth(q)} \tag{3}$$

$$hub(p) = \frac{hub(p)}{\sum_{q \in Base\ set} hub(q)} \tag{4}$$

Normalised values for our hubs and authorities are,

	Old Auth	Old Hub	New Auth	New Hub
A	1	1	3/15	1/15
B	1	1	2/15	2/15
C	1	1	5/15	1/15
D	1	1	2/15	2/15
E	1	1	1/15	4/15
F	1	1	1/15	2/15
G	1	1	0/15	2/15
H	1	1	1/15	1/15

Figure 20: Normalised values.

Step 5: Repeat for k times

After some k iterations, the value of hub and authorities converge to a point and they wont change after that. We can stop once that point is reached.

We can do that way, or we can pre-define k as some constant and run k times.

Here we define k as 2. So after 2 iterations,

	Old Auth	Old Hub	New Auth	New Hub
A	1/5	1/15	4/35	2/45
B	2/15	2/15	6/35	2/15
C	1/3	1/15	12/35	1/15
D	2/15	2/15	1/7	7/45
E	1/15	4/15	2/35	2/9
F	1/15	2/15	4/35	2/15
G	0	2/15	0	8/45
H	1/15	1/15	2/35	1/15

Figure 21: Hubs and Authority after 2 iterations

By taking k value as 2, 4, 6 we get these values for hubs and authority.

	k	A	B	C	D	E	F	G	H
Auth	2	.11	.17	.34	.14	.06	.11	0	.06
	4	.10	.18	.36	.13	.06	.11	0	.06
	6	.09	.19	.37	.13	.06	.11	0	.06
Hub	2	.04	.13	.07	.16	.22	.13	.18	.07
	4	.04	.14	.05	.18	.25	.14	.17	.04
	6	.04	.14	.04	.18	.26	.14	.16	.04

Figure 22: Comparing values when k = 2, 4 and 6

We can see that, the value of hubs and authorities are converging.
The converged values of hubs and authorities are :

	A	B	C	D	E	F	G	H
Auth	.08	**.19**	**.40**	.13	.06	.11	0	.06
Hub	.04	.14	.03	**.19**	**.27**	.14	.15	.03

Figure 23: Converged values of hubs and authorities

The hubs and authorities with the maximum value is the best one. In our example, for authority its node C and for hub it is node E.

1.4 Pseudo-Code for HITS

```
Begin
G := set of pages
for each page p in G do
    p.auth = 1 // p.auth is the authority score of the page p
    p.hub = 1 // p.hub is the hub score of the page p
    for step from 1 to k do // run the algorithm for k steps
        norm = 0
        for each page p in G do  // update all authority values first
            p.auth = 0
            for each page q in p.incomingNeighbors do // p.incomingNeighbors
    is the set of pages that link to p
                p.auth += q.hub
            norm += (p.auth) // calculate the sum of the squared auth values
    to normalise
          for each page p in G do  // update the auth scores
              p.auth = p.auth / norm   // normalise the auth values
        norm = 0
        for each page p in G do  // then update all hub values
            p.hub = 0
            for each page r in p.outgoingNeighbors do // p.outgoingNeighbors
    is the set of pages that p links to
                p.hub += r.auth
            norm += p.hub // calculate the sum of the squared hub values to
    normalise
          for each page p in G do  // then update all hub values
              p.hub = p.hub / norm    // normalise the hub values
```

2. BIRCH Algorithm

2.1 Introduction

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is an unsupervised data mining algorithm used to perform hierarchical clustering over particularly large data-sets. An advantage of BIRCH is its ability to incrementally and dynamically cluster incoming, multi-dimensional metric data points in an attempt to produce the best quality clustering for a given set of resources (memory and time constraints). In most cases, BIRCH only requires a single scan of the database.

Previous clustering algorithms performed less effectively over very large databases and did not adequately consider the case wherein a data-set was too large to fit in main memory. As a result, there was a lot of overhead maintaining high clustering quality while minimizing the cost of additional IO (input/output) operations. Furthermore, most of BIRCH's predecessors inspect all data points (or all currently existing clusters) equally for each 'clustering decision' and do not perform heuristic weighting based on the distance between these data points.

2.2 Important Terms

Centroid

Centroid is the average of all points present in a cluster. The foumula for centroid for a cluster C with N points,

$$centroid = \frac{\sum_{x \in C} x}{N} \tag{5}$$

Radius

Radius is the square root of average distance of all points in the cluster to that of the centroid of the cluster.

Therefore, for a cluster C with N points and x_i as centroid,

$$radius = \sqrt{\frac{\sum_{x \in C}(x_i - x)^2}{N}} \tag{6}$$

Diameter

Diameter is the square root of average mean square distance between all pairs of points in the cluster. So, for a cluster C with N points,

$$diameter = \sqrt{\frac{\sum_{x_i \in C} \sum_{x_j \in C}(x_i - x_j)^2}{N(N-1)}} \tag{7}$$

Cluster Feature

It is a tuple of size 3. The tuple consist of three information about the cluster its representing. The normally, CF = (N, LS, SS), where, N is the number of nodes in the cluster, LS is the sum of all points in the cluster and SS is the sum of all squared point vector in the cluster.

Note that, centroid of a cluster would be $\frac{LS}{N}$

CF-Tree

This is a Tree. Each non leaf node atmost B entries (where B is the branching factor). Each node is a collection of CF. Note that each leaf node is a sub cluster and not a datapoint. Leaf node each have a pointer prev and next, connecting them all together.

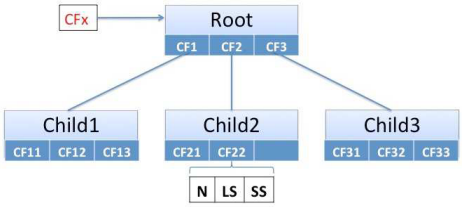

Figure 24: Structure of CF Tree.

2.3 BIRCH Algorithm with example

Phase 1: Build the CF tree

We need to follow simple steps to create the CF tree. We build it using incremental design. We take points from the data set one by one, and build the tree.

We define a threshold, such a way that, if a point is at a distance greater than the threshold from all the cluster, it forms its own cluster.

Else, we put it into the best suitable cluster and update the CF of that cluster.

As mentioned above, each node is a collection of CF. BIRCH algo is meant to deal with large dataset, thus we also define a threshold(T) (maximum number of CF in each node, normally affected by the page size), branching factor(B) (number of branches per node) and leafnode limit(L) (max number of CF in the leaf node).

We initially start with an empty node, and then keep on adding data points. When the number of clusters in the root node exceeds the leafnode limit, we break the node into B nodes, each $\frac{L}{B}$ CF features.

Currently the parent node is empty, so will also find the clustering feature of all the clusters present in that node and add it to the parent node of the child node.

Now while adding new recursively descending the CF tree and choosing the closest child node according to a chosen distance metric. Test whether the leaf can absorb the node without violating the threshold. If there is no room, then split the node. update CF information up the path.

This way, construct a CF-tree.

For our example, lets take points : (3, 3), (4, 3), (6, 3), (7, 4), (7, 5) Let the Branching factor be 2, with max 2 clusters in the leaf node and a thresh hold distance of 1.5.

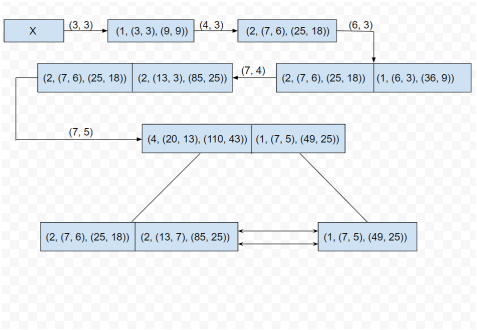

Figure 25: CF Tree for our example problem.

Phase 2: Removal of outliers

Remove all the outliers present in the leaf node. Note that this is an optional phase. This is done to prepare the tree for the next phase.

As it is, optional, we arent performing it to our example.

Phase 3: Cluster the leafnode

The problem addressed by the BIRCH is to cluster datasets too large to be fitted into the main memory.

Therefore, we apply a known clustering algo to the centroids of the clusters in the leafnode, as these are something that can be fit into the main memory.

In our example, the centroid for leaf node is, (3.5, 3), (6.5, 3.5), (7, 5)

Now, after performing k-means algo to our centroids, we get,

(3.5, 3), (6.75, 4.25)

Phase 4: Cluster the dataset.

Now that we have the centroids of all the clusters formed using the previous step, we can call all datapoints one by one, and group them into the cluster which is closest to it.

This way, we can obtain all the clusters.

In our example,
we cluster, (3, 3), (4, 3) into the same cluster and (6, 3), (7, 4) and (7, 5) into the same cluster.

2.4 Pseudo-Code for BIRCH

```
Begin
CFT(D):
    data <- regularizate(D)
    root <- initCF()
    for i from 0 to size(data):
        leaf <- closestLeaf(root, data_i)
        m <-closestCluster(root, data_i)
        d <-distance(m, data_i)
        if d < T:
            insert(data, m)
        Else:
            c <-createMinCluster(data_i)
            insert(leaf, c)
            if size(leaf) > L:
                split(leaf)
            update(leaf.parent)

clust(CFT):
    data = CFT.leaf.centroids
    centroids = kmeans(data).centroids

final\_clustering(centroids, D):
    data <- regularized(D)
    for i from 0 to size(data):
        cluster(data_i, closest(centroids, data_i))
End
```

3. Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

3.1 Introduction

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm. It is a density-based clustering non-parametric algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.

Let's think in a practical use of DBSCAN. Suppose we have an e-commerce and we want to improve our sales by recommending relevant products to our customers. We don't know exactly what our customers are looking for but based on a data set we can predict and recommend a relevant product to a specific customer. We can apply the DBSCAN to our data set (based on the e-commerce database) and find clusters based on the products that the users have bought. Using this clusters we can find similarities between customers, for example, the customer A have bought 1 pen, 1 book and 1 scissors and the customer B have bought 1 book and 1 scissors, then we can recommend 1 pen to the customer B. This is just a little example of use of DBSCAN, but it can be used in a lot of applications in several areas.

3.2 Important Terms

- **Eps**

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

- **MinPts**

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

- **Core Point**

  A point is a core point if it has more than MinPts points within eps.

- **Border Point**

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

- **Noise or outlier**

  A point which is not a core point or border point.

- **Reachability**

  In terms of density establishes a point to be reachable from another if it lies within a particular distance (eps) from it.

- **Connectivity**

  It involves a transitivity based chaining-approach to determine whether points are located in a particular cluster. For example, p and q points could be connected if $p->r->s->t->q$, *where* $a->b$ means b is in the neighborhood of a.

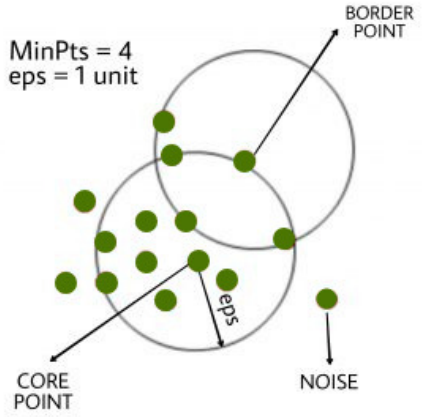

Figure 26: Figure illustrating the various terms discussed above.

3.3 DBSCAN Algorithm with example

Example Dataset

Table 1: Example dataset, note that, this is a distance matrix.

	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	2.5	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	2.5	1.4	1	0	1.1
F	3.2	2.5	2.5	0.5	1.1	0

Phase 1: Find the Core Points

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

Before this, decide upon eps and minpts.

For our example, lets take, eps as 1.5 and minpts as 3.

By looking at our example distance matrix, we get that, D, E and F are core points as they have more than 3 values in their respective column less than 1.5.

Phase 2: Create Clusters

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

In our example, distance between point D and E is 1, D and F is 0.5. Thus they all belong to the same cluster.

Phase 3: Assign points to the cluster

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

In our example, C is close (within eps distance) to corepoint D, thus making it a part of the cluster.

Phase 4: Final iteration

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

In our example, the unvisited points are A and B. But as both A and B have have no core points in their vicinity, they become noise points.

Thus our cluster comprises of D, E, F and C.

3.4 Pseudo-Code for DBSCAN

```
Begin
DBSCAN(DB, distFunc, eps, minPts) {
    C = 0                                                   /* Cluster counter
    */
    for each point P in database DB {
        if label(P)     undefined then continue             /* Previously
    processed in inner loop */
        Neighbors N = RangeQuery(DB, distFunc, P, eps)      /* Find neighbors
    */
        if |N| < minPts then {                              /* Density check */
            label(P) = Noise                                /* Label as Noise
    */
            continue
        }
        C = C + 1                                           /* next cluster
    label */
        label(P) = C                                        /* Label initial
    point */
        Seed set S = N/P                                    /* Neighbors to expand */
        for each point Q in S {                             /* Process every
    seed point */
            if label(Q) = Noise then label(Q) = C           /* Change Noise to
    border point */
            if label(Q)     undefined then continue         /* Previously
    processed */
            label(Q) = C                                    /* Label neighbor
    */
            Neighbors N = RangeQuery(DB, distFunc, Q, eps) /* Find neighbors
    */
            if |N|     minPts then {                        /* Density check
    */
                S = S     N                                 /* Add new
    neighbors to seed set */
            }
        }
    }
}
RangeQuery(DB, distFunc, Q, eps) {
    Neighbors = empty list
    for each point P in database DB {                       /* Scan all points
    in the database */
        if distFunc(Q, P)     eps then {                    /* Compute
    distance and check epsilon */
            Neighbors = Neighbors     {P}                   /* Add to result
    */
        }
    }
    return Neighbors
}
End
```

4. SLIQ

4.1 Introduction

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

4.2 Classification using SLIQ using example

Example Dataset

Age	Salary	Class
30	65	G
23	15	B
40	75	G
55	40	B
55	100	G
45	60	G

Figure 27: Training Dataset

Step 1: Pre-Sorting and Breadth-First Growth

For numeric attributes, sorting time is the dominant factor when finding the best split at a decision tree node. Therefore, the first technique used in SLIQ is to implement a scheme that eliminates the need to sort the data at each node of the decision tree. Instead, the training data are sorted just once for each numeric attribute at the beginning of the tree growth phase. To achieve this pre-sorting, we use the following data structures. We create a separate list for each attribute of the training data. Additionally, a separate list, called class list, is created for the class labels attached to the examples. An entry in an attribute list has two fields: one contains an attribute

value, the other an index into the class list. An entry of the class list also has two fields: one contains a class label, the other a reference to a leaf node of the decision tree. The ith entry of the class list corresponds to the ith example in the training data. Each leaf node of the decision tree represents a partition of the training data, the partition being defined by the conjunction of the predicates on the path from the node to the root. Thus, the class list can at any time identify the partition to which an example belongs. We assume that there is enough memory to keep the class list memory-resident. Attribute lists are written to disk if necessary.

Initially, the leaf reference fields of all the entries of the class list are set to point to the root of the decision tree. Then a pass is made over the training data, distributing values of the attributes for each example across all the lists. Each attribute value is also tagged with the corresponding class list index. The attribute lists for the numeric features are then sorted independently.

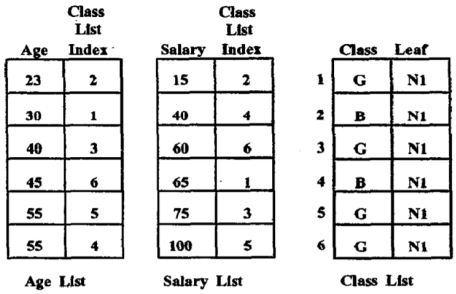

Figure 28: After Pre-sorting.

Step 2: Processing Node Splits

Rather than using a depth-first strategy used in the earlier decision-tree classifiers, we grow trees breadth-first. Consequently, splits for all the leaves of the current tree are simultaneously evaluated in one pass over the data. Figure 4 gives a schematic of the evaluation process.

To compute the gini splitting-index for an attribute at a node, we need the frequency distribution of class values in the data partition corresponding to the node. The distribution is accumulated in a class histogram attached with each leaf node. For a numeric attribute, the histogram is a list of pairs of the form ¡class, frequency¿. For a categorical attribute, this histogram is a list of triples of the form ¡attribute value, class, frequency¿. Attribute lists are processed one at a time (recall that the attribute lists can be on disk). For each value IJ in the attribute list for the current attribute A, we find the the corresponding entry in the class list, which yields the corresponding class and the leaf node. We now update the histogram attached with this leaf node. If A is a numeric attribute, we compute at the same time the splitting index for the test A 2 v for this leaf. If A is a categorical attribute, we wait till the attribute list has been completely scanned and then find the subset of A with the best split. Thus, in one traversal of an attribute list, the best split using this attribute is known for

all the leaf nodes. Similarly, with one traversal of all of the attribute lists, the best overall split for all of the leaf nodes is known. The best split test is saved with each of the leaf nodes. For our example

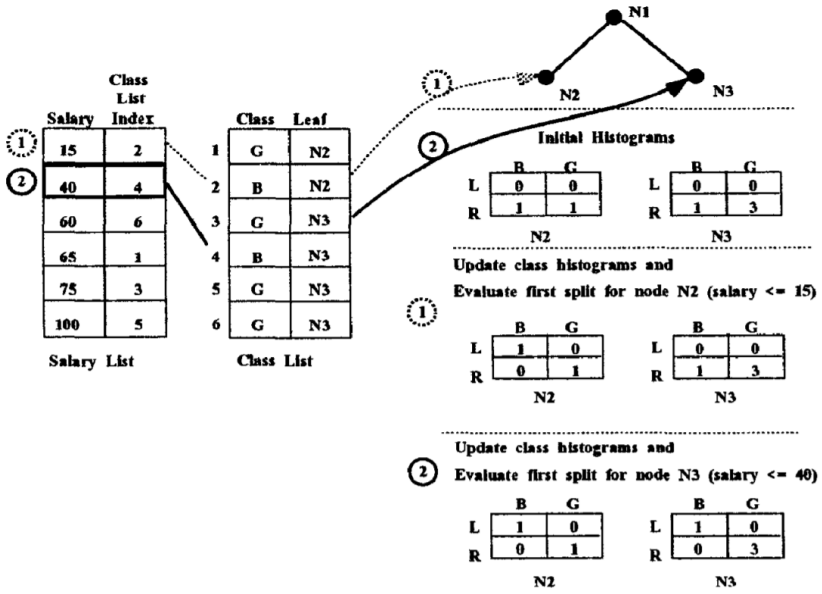

Figure 29: Node Splitting.

Figure 29 illustrates the evaluation of splits on the salary attribute for the second level of the decision tree. The example assumes that the data has been initially split on the age attribute using the split age 2 35. The class histograms reflect the distribution of the points at each leaf node as a result of the split. The L values represent the distributions for examples that satisfy the test and R values represent examples that do not satisfy the test. We show how the class histograms are updated as each split is evaluated. The first value in the salary list belongs to node N2. So the first split evaluated is (salary $\leq$ 15) for N2. After this split, the corresponding example (salary 15, class index 2) which satisfies the predicate belongs to the left branch and the rest belong to the right branch. The class histogram of node N2 is updated to reflect this fact. Next, the split (salary 5 40) is evaluated for node N3. After the split, the corresponding example (salary 40, class index 4) belongs to the left branch and the class histogram of node N3 is updated to reflect this fact.

Step 3: Updating the Class List

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

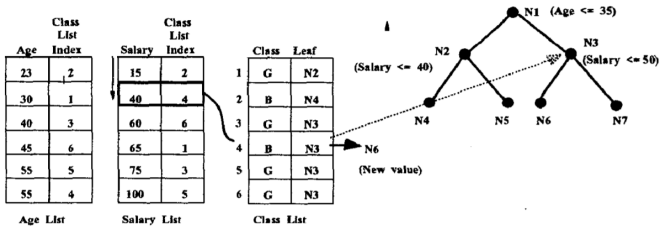

Figure 30: Class List Updation.

As an illustration, Figure 30 shows the class list being updated after the nodes N2 and N3 have been split on the salary attribute. The salary attribute list is being traversed and the class list entry (entry 4) corresponding to the salary value of 40 is being updated. First, the leaf reference in the entry 4 of class list is used to find the node to which the example used to belong (N3 in this case).

Then, the split selected at N3 is applied to find the new child to which the example belongs (N6 in this case). The leaf reference field of entry 4 in the class list is updated to reflect the new value.

This is how we create the decision tree.

4.3 Pseudo-Code for SLIQ

```
Begin
EvaluateSplits()
    for each attribute A do
        traverse attribute list of A
        for each value TV in the attribute list do
            find the corresponding entry in the class list, and
                hence the corresponding class and the leaf node (say I)
            update the class histogram in the leaf I
            if A is a numeric attribute then
                compute splitting index for test (A ≤ v) for leaf 1
            if A is a categorical attribute then
                for each leaf of the tree do
                    find subset of A with best split


UpdateLabels()
    for each attribute A used in a split do
        traverse attribute list of A
        for each value u in the attribute list do
            find the corresponding entry in the class list (say e)
            find the new class c to which v belongs by applying
                the splitting test at node referenced from e
            update the class label for e to c
            update node referenced in e to the child corresponding to the
    class c

End
```

5. Associate Rule Based Classification(ARBC)

5.1 Introduction

Association rule mining was introduced as a way to find associative patterns from market basket data. The market basket data consist of transactions where a transaction is a set of items purchased by a customer. The motivation for applying this data mining approach on market basket data was to learn about buying patterns and use that information in catalog design, and store layout design.

Since then, association rule mining has been studied and applied in many other domains (e.g. credit card fraud, network intrusion detection, genetic data analysis).

In every domain, there is a need to analyze data to identify patterns associating different attributes. Association rule mining addresses this need. Many association rule mining algorithms have been proposed in the data mining literature. Apriori and FP-growth are two of them.

5.2 Classification using ARBC using example

Example Dataset

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

Figure 31: Training Dataset

Step 1: Create Associate rules

For this, we can use any associate rule mining algorithm to mine the cassfication rule.In associative classification, the focus is to produce association rules that have only a particular attribute in the consequent. These association rules produced are called class association rules(CARs).

Associative classification differs from general association rule mining by introducing a constraint as to the attribute that must appear on the consequent of the rule.

The CBA-RG algorithm is an extension of the Apriori algorithm. The goal of this algorithm is to find all rule items of the form $< condset, y >$ where condset is a set of items, and y $\in$ Y, where Y is the set of class labels. The support count of the rule item is the number of instances in the data set D that contain the condset and are labeled with y. Each rule item corresponds to a rule of the form: condset $\rightarrow$ y.

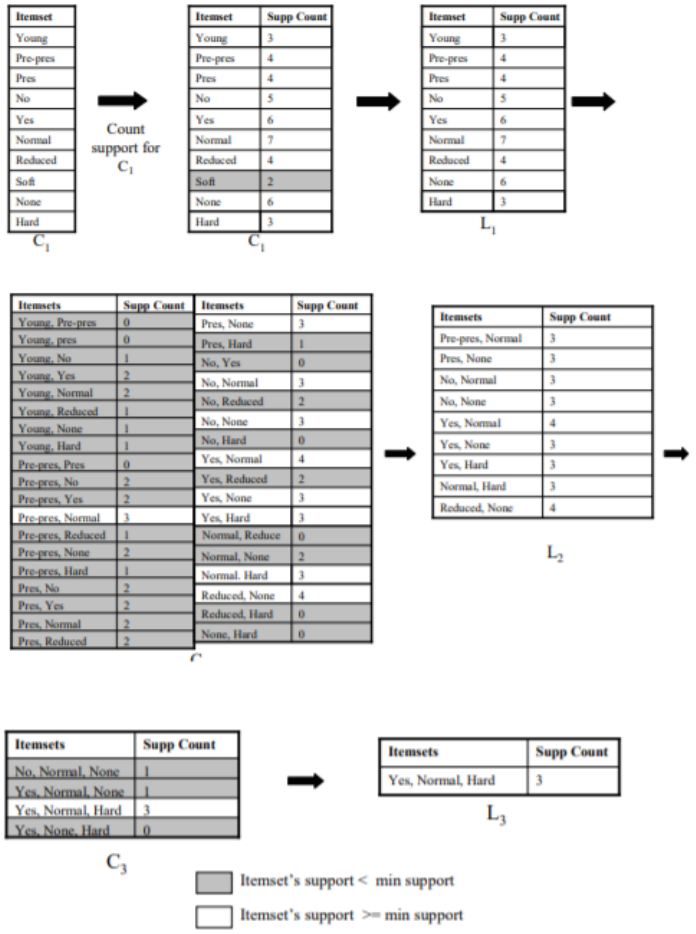

Figure 32: Applying Apriori with min support 3.

Step 2: Classifying Based on the rules

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

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

- if rule ri has greater confidence than rj, then ri precedes rj, or

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

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

After Applying apriori, the rules we get are,

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

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

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

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

Figure 33: Apriori rules with confidence 0.5.

5.3 Pseudo-Code for ARBC

```
Begin
minConfidence
rules =    ;
freqItemsets =    ;
support = UpperBoundSupport;
while (support     LowerBoundSupport AND rules.size < minNumberOfRules) do
    L1 = {1-item itemsets};
    for (k = 2;  L k 1  6=    ) do
        Ck = generateCandidates( L k 1 );
        Lk = evaluateCandidates(Ck);
        freqItemsets     L(k);
    end for
    maxFreqItemsets = genMaxFreqItemset(freqItemsets);
    rules = GenerateAllRules(maxFreqItemsets, minConfidence);
    support = support - delta;
    freqItemsets =    ;
end while

R = rules

R = sort(R);
for each rule r ∈ R in sequence do
    temp = ∅
    for each instance d ∈ D do
        if d satisfies the conditions of r then
            store d.id in temp and mark r if it correctly classifies d;
        end if
    end for
    if r is marked then
        insert r at the end of C;
        delete all the cases with the ids in temp from D;
        select the default class for the current C;
        compute the total number of errors of C;
    end if
end for
 Find the first rule p in C such that Cp, the list of rules in C up to p, has
    lowest
al number of errors. and drop all the rules.
 Add the default class associated with p to the end of C, and return C
End
```

6. Linear Regression

6.1 Introduction

In statistics, linear regression is a linear approach to modeling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression. This term is distinct from multivariate linear regression, where multiple correlated dependent variables are predicted, rather than a single scalar variable.

In linear regression, the relationships are modeled using linear predictor functions whose unknown model parameters are estimated from the data. Such models are called linear models. Most commonly, the conditional mean of the response given the values of the explanatory variables (or predictors) is assumed to be an affine function of those values; less commonly, the conditional median or some other quantile is used. Like all forms of regression analysis, linear regression focuses on the conditional probability distribution of the response given the values of the predictors, rather than on the joint probability distribution of all of these variables, which is the domain of multivariate analysis.

Linear regression was the first type of regression analysis to be studied rigorously, and to be used extensively in practical applications. This is because models which depend linearly on their unknown parameters are easier to fit than models which are non-linearly related to their parameters and because the statistical properties of the resulting estimators are easier to determine.

Linear regression has many practical uses. Most applications fall into one of the following two broad categories:

- If the goal is prediction, forecasting, linear regression can be used to fit a predictive model to an observed data set of values of the response and explanatory variables.

- If the goal is to explain variation in the response variable that can be attributed to variation in the explanatory variables, linear regression analysis can be applied to quantify the strength of the relationship between the response and the explanatory variables, and in particular to determine whether some explanatory variables may have no linear relationship with the response at all, or to identify which subsets of explanatory variables may contain redundant information about the response.

6.2 Math behind Linear Regression

For a response variable Y, and explanatory variable $X_1,\ X_2,\ ...,\ X_n$, the equation of Y is,

$$Y = \alpha_0 + \alpha_1 X_1 + ... + \alpha_n X_n \tag{8}$$

where,

- $\alpha_0,\ ...,\ \alpha_n$ are coefficient of the respective explanatory variable. With them we can tell how much of an effect each explanatory variable has on response variable.
- α is the coefficient vector, i.e., $[\alpha_0,\ \alpha_1,\ ...,\ \alpha_n]^T$

If we have m datapoints, the error/cost function is,

$$J(\alpha) = \frac{1}{2m} * (\sum_{i=1}^{m} \hat{Y_i} - Y_i)^2 \tag{9}$$

where

- Y_i is the output of i^th point in the dataset.
- is our predicted output of i^th point in the dataset.

We need to reduce the cross function. For a learning rate l, we update the coefficients as,

$$\alpha_i = \alpha_i - l * \frac{\partial J(\alpha)}{\partial \alpha_i} \tag{10}$$

6.3 Linear Regression on an example dataset

Example Dataset

Table 2: Dataset

X	Y
6.2	26.3
6.5	26.65
5.48	25.03
6.54	26.01
7.18	27.9
7.93	30.47

Here is the plot of the above datapoints.

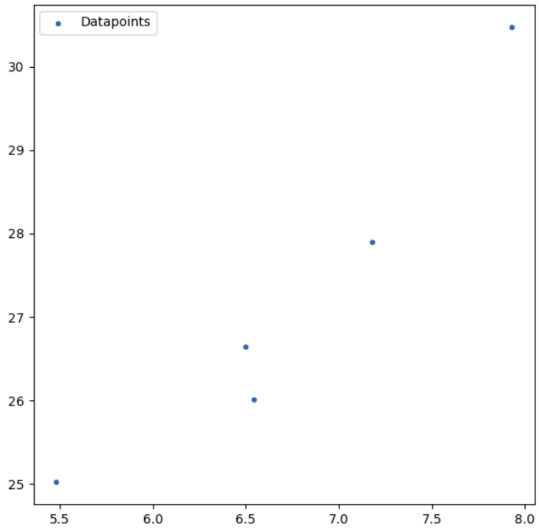

Figure 34: Plot of the example dataset.

Step 1: Set initial values

We need to set the learning rate and initial value pf α_i.

For our dataset, we set the learning rate as 0.01 and α_0 as 1, and α_1 as 1.

Step 2: Find the cost

We need to calculate the cost generated by our model.

In our example, by eq(12), the cost we get is 190.48

Step 3: Update the coefficient

Update the coefficient according to the equation.

For our example,

$$\frac{\partial J(Y)}{\partial \alpha_0} = -19.48$$
$$\frac{\partial J(Y)}{\partial \alpha_1} = -131.86$$

Therefore, by eq(13)

$$\alpha_0 = 1.195$$
$$\alpha_1 = 2.319$$

Step 5: Repeat

Find the cost and update the coefficient until the cost is negligible or the change in the coefficients are negligible.

In our example, the algo will stop when the difference between coefficient is less than 0.01.

The final values of $\alpha_1 = 3.8$ and $\alpha_0 = 1.425$ with cost 0.997.

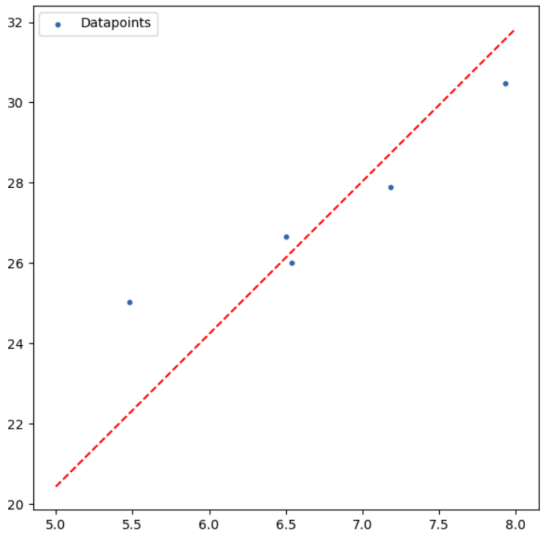

Figure 35: Our regression line.

6.4 Pseudo-Code for Linear Regression

```
Begin
initialise learning rate l
n = len(X)
Initialise coefficient α - [1, 1, 1, ..., 1]
while J or α converges:
    J = 1/2m * (∑_{i=1}^{n} Targetvalue_i − Predictedvalue_i)^2
    for i in [0, n]:
        α_i = α_i − l * ∂J(α)/∂(α_i)
End
```

7. Non Linear Regression

7.1 Introduction

In statistics, nonlinear regression is a form of regression analysis in which observational data are modeled 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.

7.2 Math behind Non Linear Regression

In nonlinear regression, a statistical model of the form,

$$Y \sim f(X, \beta) \tag{11}$$

where

- f(.) is a non linear function.
- is the weight vector.
- X is the explanatory variable and Y is the response variable.

Some popular non linear models are

- Exponential Model

$$f(X) = ae^{bX}$$

- Power Model

$$f(X) = aX^b$$

- Saturation Growth Model

$$f(X) = \frac{aX}{b+X}$$

- Polynomial Model

$$f(X) = a_0 + a_1X + a_2X^2 + ... + a_mX^m, m \geq 2$$

The assumption underlying this procedure is that the model can be approximated by a linear function, namely a first-order Taylor series:

$$f(x_i, \beta) \approx f(x_i, 0) + \sum_j J_{ij}\beta_j \tag{12}$$

where,

- $J_{ij} = \dfrac{\partial f(x_i, \beta)}{\partial \beta_j}$

The final value of β will be :

$$\hat{\beta} \approx (JJ^T)^{-1}J^TY \tag{13}$$

The nonlinear regression statistics are computed and used as in linear regression statistics, but using J in place of X in the formulas. The linear approximation introduces bias into the statistics. Therefore, more caution than usual is required in interpreting statistics derived from a nonlinear model.

For example, for a quadratic model

$$Y \sim \beta_0 + \beta_1 X + \beta_2 X^2$$

$$\begin{bmatrix} 1 & X_1 & X_1^2 \\ 1 & X_2 & X_2^2 \\ & \cdot & \\ & \cdot & \\ & \cdot & \\ 1 & X_m & X_m^2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \\ \cdot \\ \cdot \\ \cdot \\ Y_n \end{bmatrix}$$

$$X\beta = Y$$

$$\beta = (X^TX)^{-1}X^TY$$

7.3 Non Linear Regression (Polynomial Model) on an example dataset

Example Dataset

Table 3: Dataset

X	Y
1	7
2	2
3	1
4	3

Here is the plot of the above datapoints.

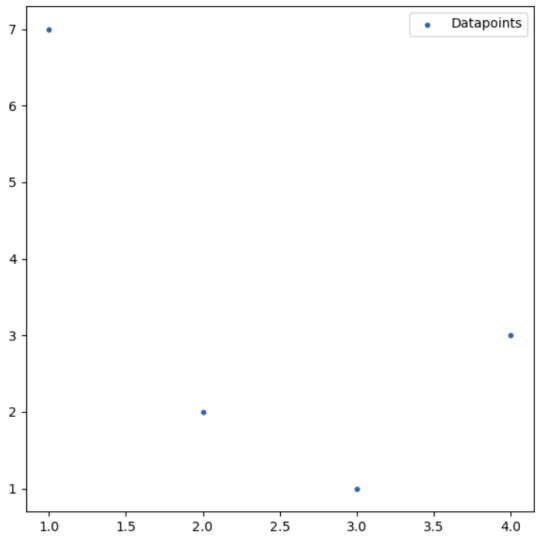

Figure 36: Plot of the example dataset.

Step 1: Set the X matrix

Set the X matrix as given in the example above.

In our example,

$$X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{bmatrix} \tag{14}$$

Step 2: Find the pseudo inverse

Find the pseudo inverse of the X matrix.

In our example,

$$(X^T X)^{-1} X^T = \begin{bmatrix} 2.2500 & -0.7500 & -1.2500 & 0.7500 \\ -1.5500 & 1.1500 & 1.3500 & -0.9500 \\ 0.2500 & -0.2500 & -0.2500 & 0.2500 \end{bmatrix} \tag{15}$$

Step 3: Find weight vector

Calculate β as per the example above.

In our example,

$$\beta = \begin{bmatrix} 2.2500 & -0.7500 & -1.2500 & 0.7500 \\ -1.5500 & 1.1500 & 1.3500 & -0.9500 \\ 0.2500 & -0.2500 & -0.2500 & 0.2500 \end{bmatrix} \begin{bmatrix} 7 \\ 2 \\ 1 \\ 3 \end{bmatrix}$$

$$\beta = \begin{bmatrix} 15.25 \\ -10.05 \\ 1.75 \end{bmatrix}$$

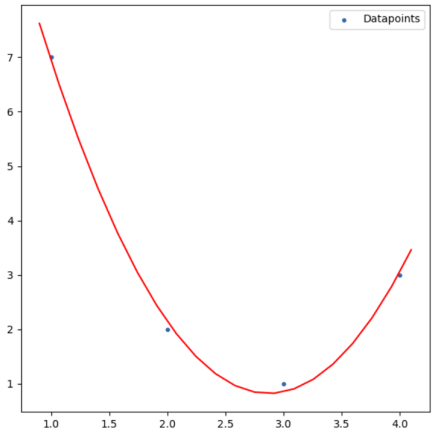

Figure 37: Our Polynomial line.

7.4 Pseudo-Code for Polynomial Regression

```
Begin
Construct the X matrix and Y matrix respectively
Find the pseudo inverse of X matrix
β = X⁺Y
End
```