Project Athena:

ML-Personalization For Recommendation Engine

Sayan Banerjee

10/11/2019

Clustering

What is the task of “clustering” in a generic sense?

- The task of discovering groups and structure in the data that are ”similar” in some way or other, without using known structure in the data.

How is clustering categorized?

- Clustering is a type of unsupervised learning because there are no class labels denoting a priori partition of the underlying data are known.
- Unlike supervised learning as the “ground truth” is not available for clustering, interpretation of the result of the clustering highly depends on the perspective of the user and context of the problem.

What is our context?

- Provide personalized recommendation to an end-user when an end-user shops for/within a property/entity (who owns or responsible for managing and/or distributing products).

Details of The Problem Statement

- Premise:
  - Recommendation Engine using MAB algorithm.
  - Anonymous end-user (almost always).
  - Properties from all over the world of all possible types and very different from each other with very different customer base on their own.
- What we know so far:
  - MAB produces the ordered list (sub list) of recommended products given a list of products, their respective statistics and end-user “personalization” category (a label given to the end user which is a function of information available by the time of transaction.)
  - This is a proven fact that if a “personalization” category has a similar buying/shopping trend the MAB will produce personalize offers for that category in a fairly short time.
- Satisfying Criteria:
  - Segments/Clusters have to be mutually exclusive and collectively exhaustive.
  - Scalable and fast at inference.
- Key Things to Consider:
  - We need to create “personalization” category based on buying behavior so that we can ensure that within one category people shows similar buying/shopping trend/behavior.
  - We need to assign “personalization” category to an end-user based on the “session”/demographic information.
  - We need to ensure each “personalization” category has enough members for MAB to optimize it’s results and continue to be an adaptable and responsive state so that it can react to any change in environment fairly quickly.

Algorithm of the Whole Process (1)

Step - 1:

- Considering we have “correct” and “complete” representation of traffic data, we will divide the whole customer base into small groups in a meaningful way. We can control what would be the maximum size of these groups so that we don’t loose granularity.
    - We will use CLTree for this purpose. This is the implementation of the approach described here. “Clustering Via Decision Tree Construction” by Bing Liu , Yiyuan Xia, and Philip S. Yu [Ref: http://web.cs.ucla.edu/~wwc/course/cs245a/CLTrees.pdf] (We will go through it later in more details).

Step - 2:

- We will assign a representation vector to each of these small groups. Representation vector can be anything. Some viable choices are,
    - Average Shopping/Buying Behavior of all the members of the underlying group.
    - Average of demographic/”session” based information of all the members of the underlying group.
    - Centroid of the “hyper rectangle” of the group. (We will explain it later in more details).

Algorithm of the Whole Process (2)

Step – 3:

- We will use agglomerative clustering technique on the previously calculated representation vectors in a bottom-up fashion until each cluster reaches an externally provided minimum number of members (Sum of all the members of all the encompassed small groups in each cluster.) and take a snapshot of the current setting and continue the procedure until only 2 clusters left.

Step – 4:

- At the end of the process we will finalize a setting which minimizes,
  - $\forall|\text{cluster}|_i \geq min - members, argmin\ (\frac{Avg.\ Intra-Cluster\ Distance\ in\ Current\ Setting}{Avg.\ Inter-Cluster\ Distance\ in\ Current\ Setting})$.
    - There are externally provided “gradient tolerance” parameter to boost-up the little bigger set of clusters over smaller set of clusters. (We will come back to it later.)

Step – 5:

- Once a setting is finalized, for every cluster in that setting, we will assign an unique id to all small groups within a cluster.
  - Now if you realize, we have divided the the customer base into small groups with one criteria but assigned labels to each group with another criteria.
  - We retrieve the label/id assign to each group with just the “dividing” criteria. Once the label is assigned we don’t need the conquering criteria anymore.

Algorithm of the Whole Process (3)

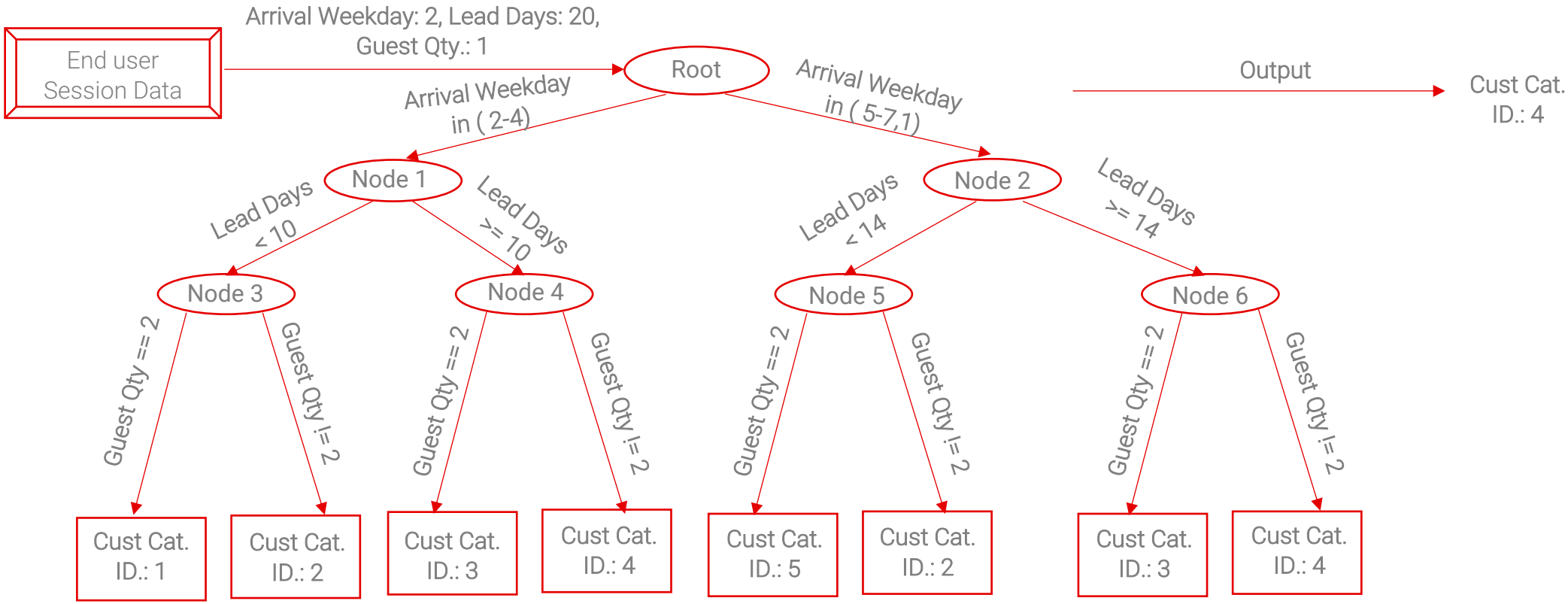
Step – 6:

- At this stage, we don not need to maintain all sorts of complex calculations we have done so far at each stage. So we morph the CLTree into a simple multi-dimensional BST which supports ({<=, >}, {==, !=} and {In, Not In}) to support all kind of data types and attributes.
    - Before morphing/ transferring the CLTree into a into a simple multi-dimensional BST we perform some pruning to make the search space more efficient.

Step – 7:

- We save the final BST in a binary format to get retrieved and use of inference at transaction time.
    - As we were building this final BST, we also make strategies for missing data at inference time.
    - The final model has a very simple API, need only 2 basic python class definitions, ability to read pickle file and input in a dictionary format. That's set.

Persona Segmentation with Multi-dimensional BST



Dividing the Data Into Small Meaningful Groups Using CLTree

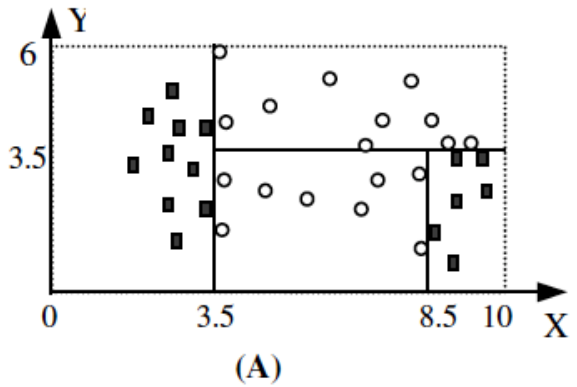
Gentle Introduction of CLTree:

- What is CLTree?
  - CLTree stands for Clustering Based on decision Tree.
  - Uses a mechanism which is more popular in supervised learning problems named “decision tree construction” with some modifications.
- How it is different than typical ML Clustering Techniques?
  - Typical Clustering techniques forms clusters explicitly by grouping data points based on some distance or density measures while CLTree finds cluster implicitly by separating dense regions from sparse regions using a purity function based on concepts from information theory.

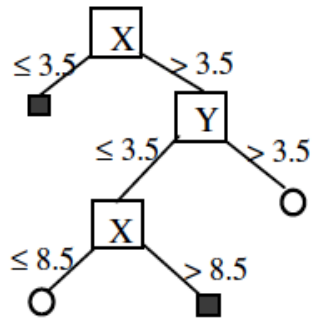
Prerequisite: Gentle Introduction of Decision Tree

- Supervised learning algorithm where there is correct label/real value associated with every data row.
- A Decision Tree has 2 types of nodes, 1. Decision Nodes, 2. Leaf Nodes.
  i. A decision node specifies some test on a single attribute.
  ii. A leaf node indicates a class or some Y-value.
- The algorithm for building a decision tree uses the divide and conquer strategy to recursively partition the data to produce the tree.
- Each successive step greedily chooses the best cut to partition the space into two parts in order to obtain purer regions.
- A commonly used criterion (or purity function) for choosing the best cut is the information gain.
- A serial of tests (or cuts) from the root node to a leaf node represents a hyper-rectangle.

Ex. Let's say, a data is be represented by X > 3.5, Y > 3.5 → Class O and everything else is of Class ■



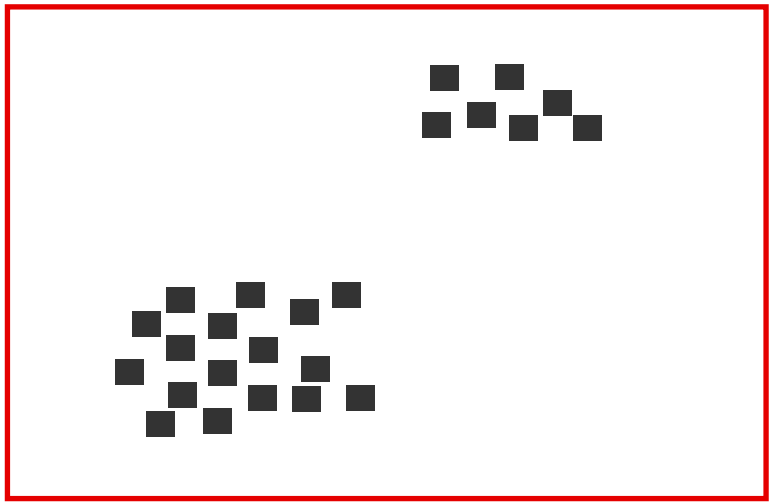
(A)



(B)

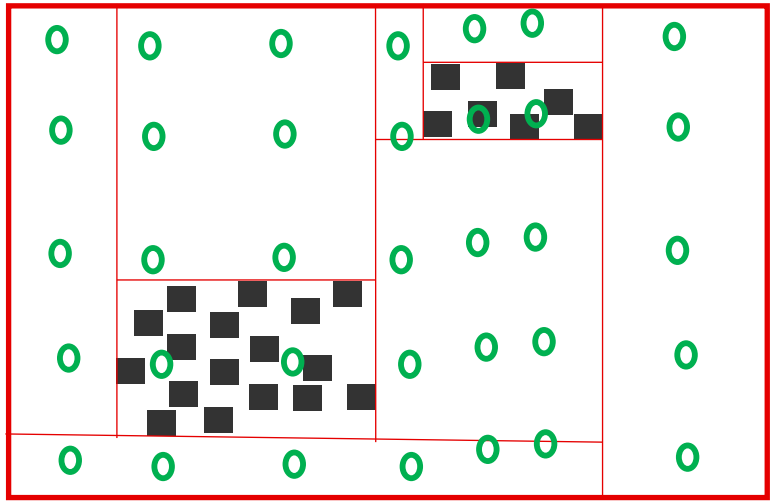
1 **for** each attribute $A_i \in \{A_1, A_2, ..., A_d\}$ of the dataset D **do**
2 **for** each value x of A_i in D **do**
/* each value is considered as a possible cut */
3 Compute the information gain at x
4 **end**
5 Select the test or cut that gives the best information gain to partition the space

Base Intuition Behind CLTree



A

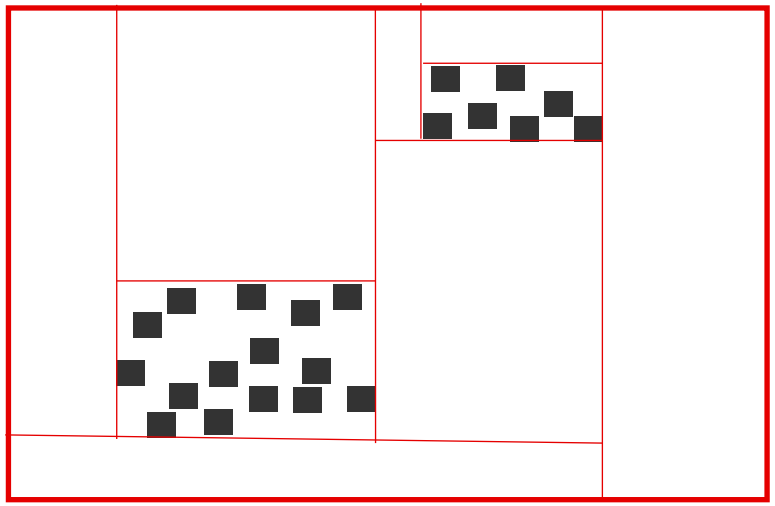
1. Figure-A gives a 2-dimensional space, which has 24 data (Y) points. Two clusters exist in the space.



B

2. We then add some uniformly distributed N points (represented by “o”) to the data space (Figure-B). With the augmented dataset, we can run a decision tree algorithm to obtain a partitioning of the space (Figure-B). The two clusters are identified.

3. CLTree is a technique to solve the problem. This is done by not adding any N point to the space but computing them when needed. Hence, CLTree is able to produce the partition in Figure-C with no N point added to the data.

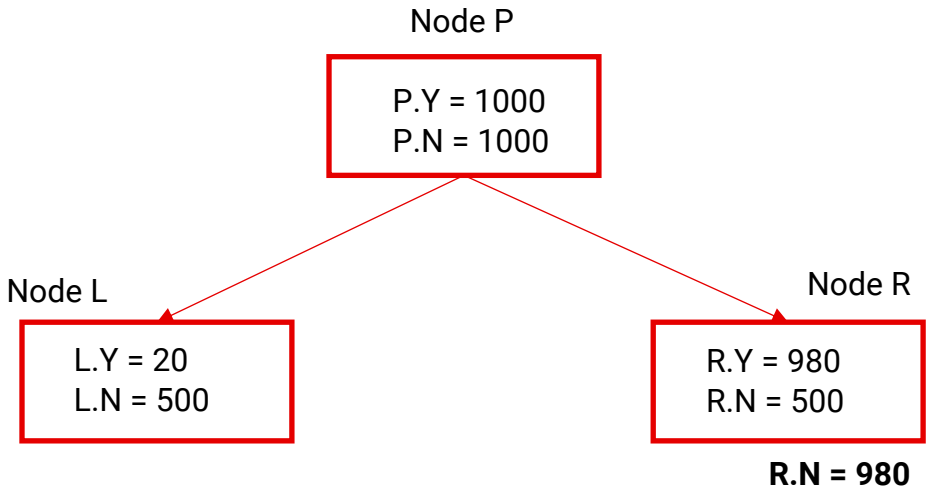


C

Introducing N Points

- Give each data point in the original dataset the class Y.
- Introduce some uniformly distributed “non-existing” N points.(We do not physically add these N points to the original data, but only assume their existence.)
- The number of N points for the current node E is determined by the following rule:
  1. **If** the number of N points inherited from the parent node of E is less than the number of Y points in E then
  2. the number of N points for E is increased to the number of Y points in E
  3. **Else**
  4. the number of inherited N points is used for E
- Ex:

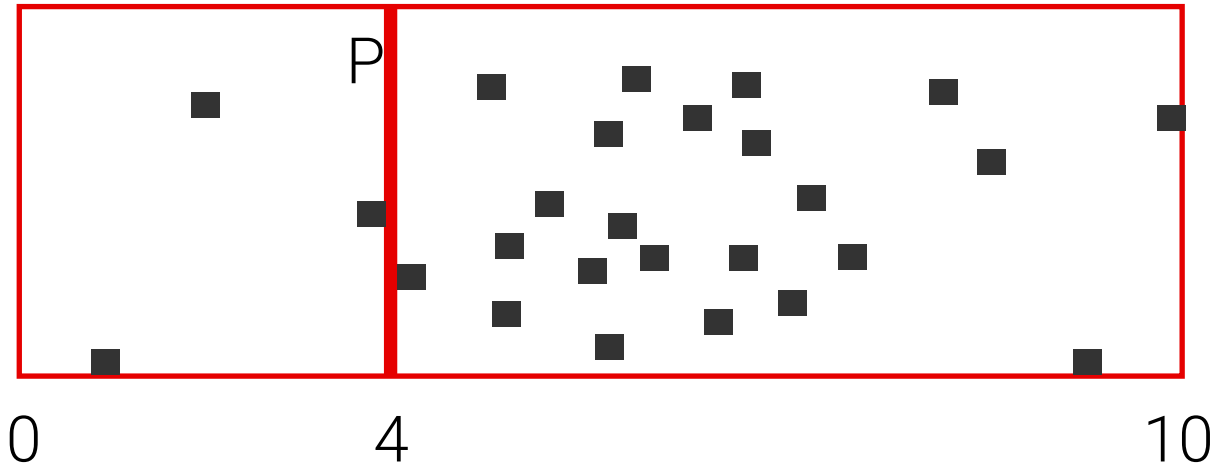
P has two children nodes L and R. Assume P has 1000 Y points and thus 1000 N points, stored in P.Y and P.N respectively. Assume after splitting, L has 20 Y points and 500 N points, and R has 980 Y points and 500 N points. According to the above rule, for subsequent partitioning, we increase the number of N points at R to 980. The number of N points at L is unchanged.



Two Modifications to the Decision Tree Algorithm(1):

- Compute the number of N points on the fly:
  - The information gain evaluation in only needs the frequency or the number of points of each class on each side of a possible cut (or split).

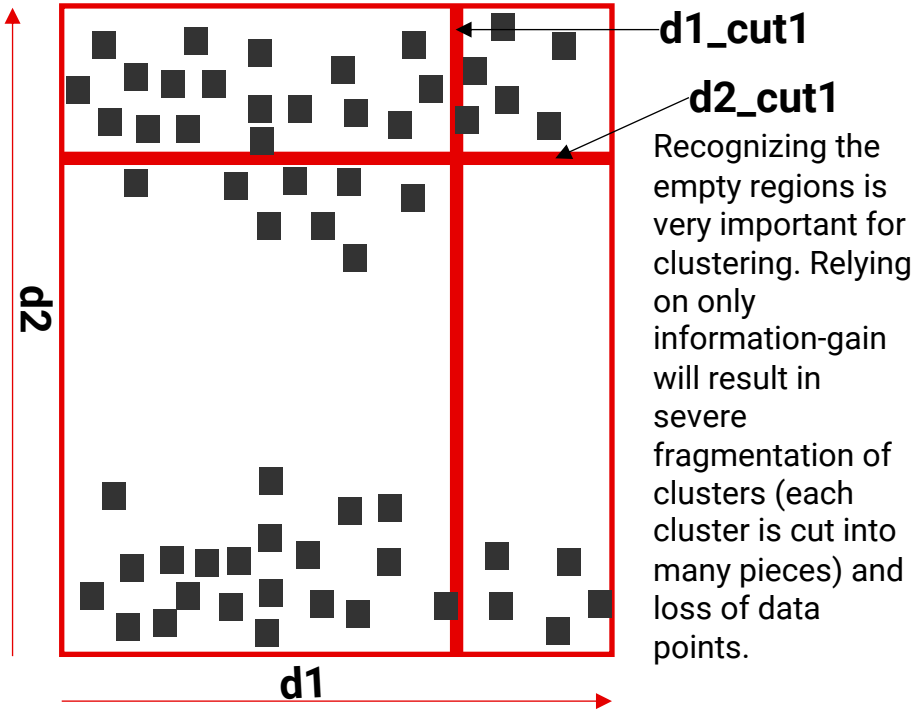
- EX.:
  - The space has 25 data (Y) points and 25 N points. Assume the system is evaluating a possible cut P.
  - The number of N points on the LHS of P is 25 * 4/10 = 10. The number of Y points is 3.
  - Likewise, the number of N points on the RHS of P is 15 (= 25 - 10), and the number of Y points is 22.
  - With these numbers, the information gain at P can be computed. Note that by
  - Computing the number of N points, we essentially guarantee their uniform distribution in the current space.



Two Modifications to the Decision Tree Algorithm(2):

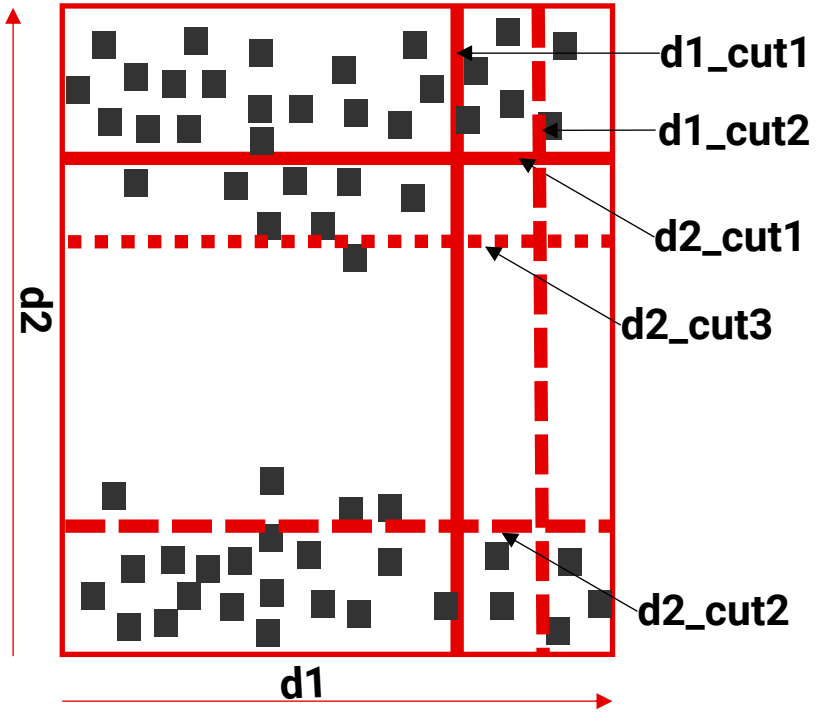
- Evaluate on both sides of data points:
  - In the standard decision tree building, cuts only occur on one side of data points. However, for our purpose this is not adequate.

- Only Information-Gain criterion is not Enough.
  - The cut with the best information gain tends to cut into clusters. This results in severe loss of data points in clusters.
  - The gain criterion does not look ahead in deciding the best cut.



Recognizing the empty regions is very important for clustering. Relying on only information-gain will result in severe fragmentation of clusters (each cluster is cut into many pieces) and loss of data points.

The new criterion made the d1_cut2, d2_cut2 and d2_cut3 possible. The end goal is to, as much as possible, isolate dense region from sparse region at each iteration



The New Criterion for Acquiring a Cut

- The Basic Idea:
  - For each dimension_i, based on the first cut found using the information-gain criterion, we look ahead (at most 2 steps) to find a better cut c_i that cuts less into cluster regions, and to find an associated region r_i that is relatively empty (measured by relative density, see below). c_i of the dimension_i whose r_i has the lowest relative density is selected as the best cut. The intuition behind this modified criterion is clear. It tries to find the emptiest region along each dimension to separate clusters.

- Relative Density:
  - The relative density of a region r is computed with r.Y / r.N, where r.Y and r.N are the number of Y points and the number of N points in r respectively.

- Step By Step:
  1. Find the initial cuts for each dimension_i, we use the information-gain criterion to find the first best cut point di_cut_1. If we cannot find di_cut_1 with any gain for a dimension, we ignore this dimension subsequently.
  2. Look ahead to find better cuts based on the first cut. We find a better cut on each dimension_i by further gain evaluation in respective lower density regions. Our objectives are to find:
     a. a cut that cuts less into clusters (to reduce the number of lost points)
     b. an associated region with a low relative density (relatively empty).
  3. Select the overall best cut by comparing the relative densities (r_density_i) of the low density regions identified in step 2 of all dimensions. The best cut in the dimension that gives the lowest r_density_i value is chosen as the best cut overall. In our example, the relative density between d2_cut2 and d2_cut3 is clearly lower than that between d1_cut2 and the right space boundary, thus d2_cut3 is the overall best cut.

Algorithm to Evaluate Cut at Each Node

for each attribute A_i ∈ {A_1, A_2, …, A_d} do

di_cut1 = the value (cut) of Ai that gives the best gain on dimension_i

di_cut2 = the value (cut) of Ai that gives the best gain in the LRD_i region produced by di_cut1

if the relative density between di_cut1 and di_cut2 is higher than that between di_cut2 and boundary_i then

r_density_i = y_i /n_i, where y_i and n_i are the numbers of Y points and N points between di_cut2 and boundary_i

else

di_cut3 = the value (cut) that gives the best gain in the LRD_i region produced by di_cut2

/* Realize LRD_i here is the region between di_cut1 and di_cut2 */

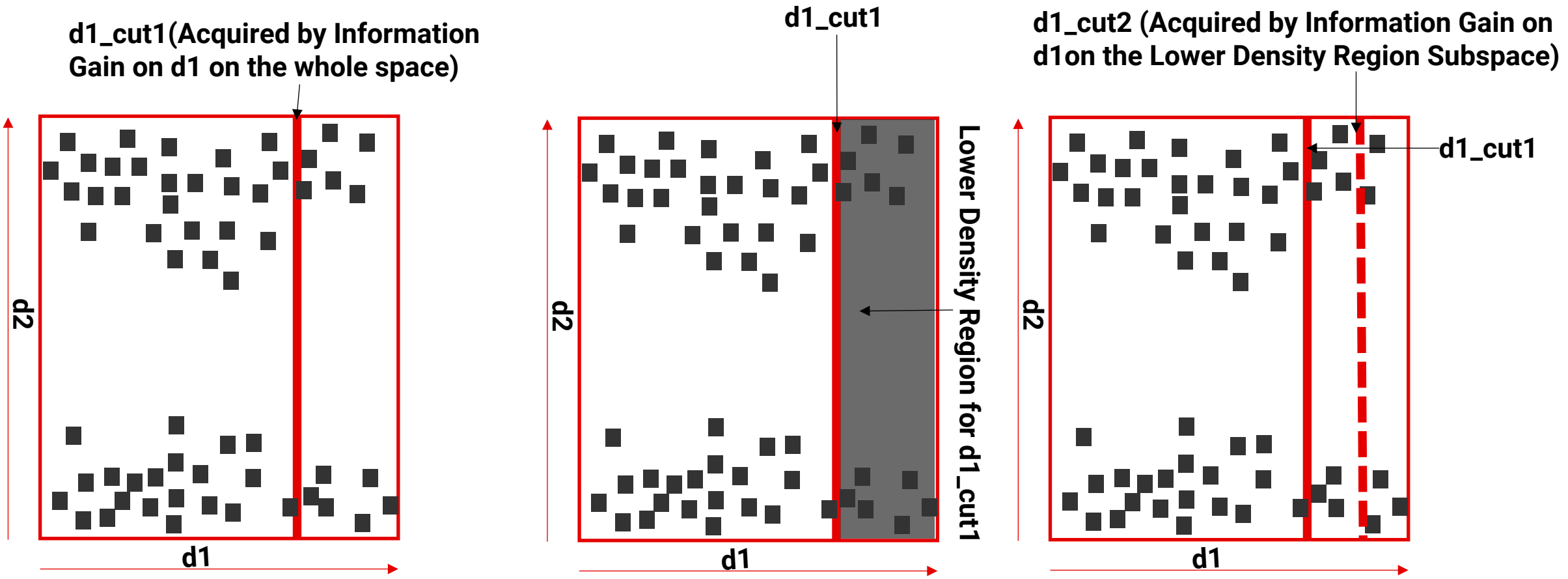
r_density_i = y_i/n_i, where y_i and n_i are the numbers of Y points and N points in the region between di_cut1 and di_cut3 or di_cut2 and di_cut3 that has a lower proportion of Y points (or a lower relative density).

end if

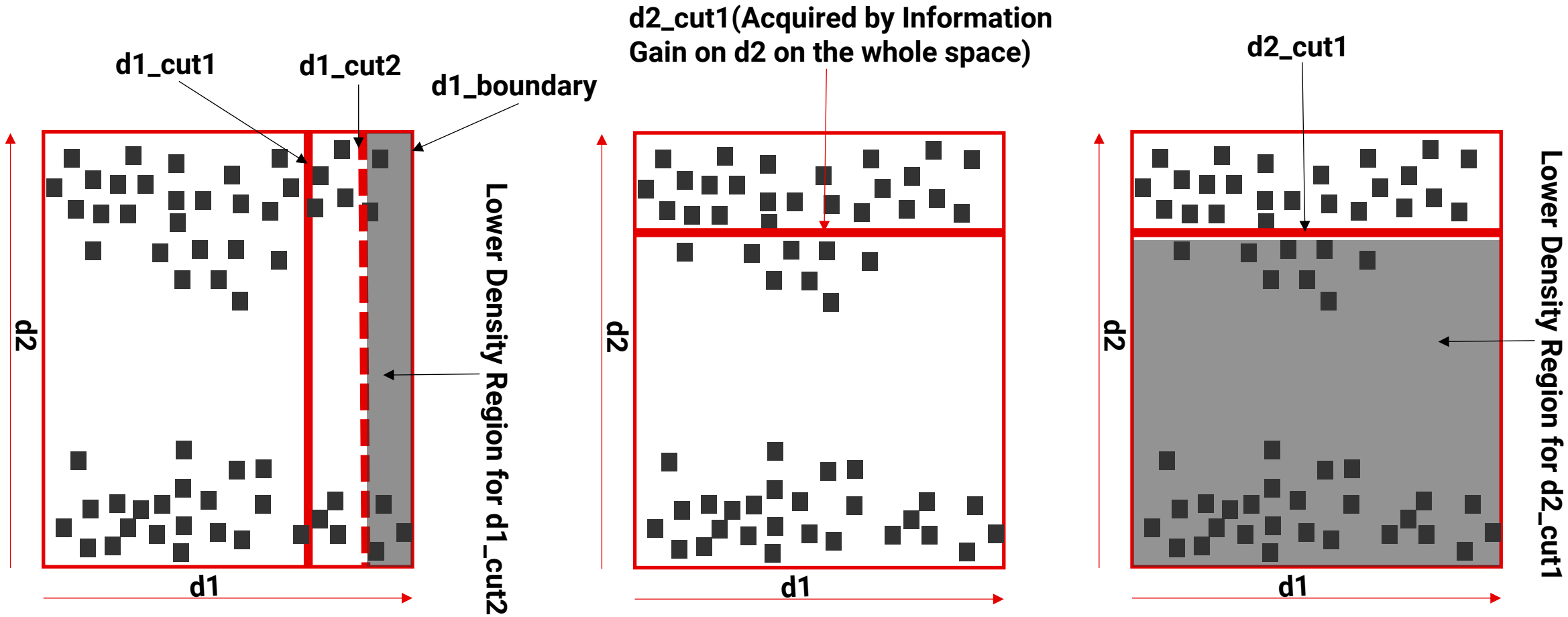
end for

bestCut = di_cut3 (or di_cut2 if there is no di_cut3) of dimension i whose r_density_i is the minimal among the d dimensions.

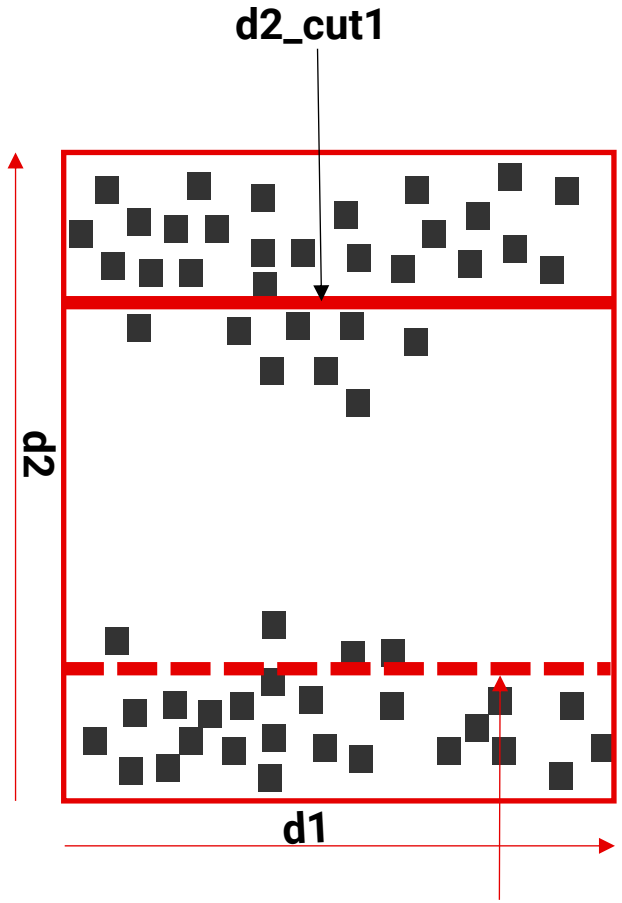
What Happens at Each Node When Evaluating Cuts(1)



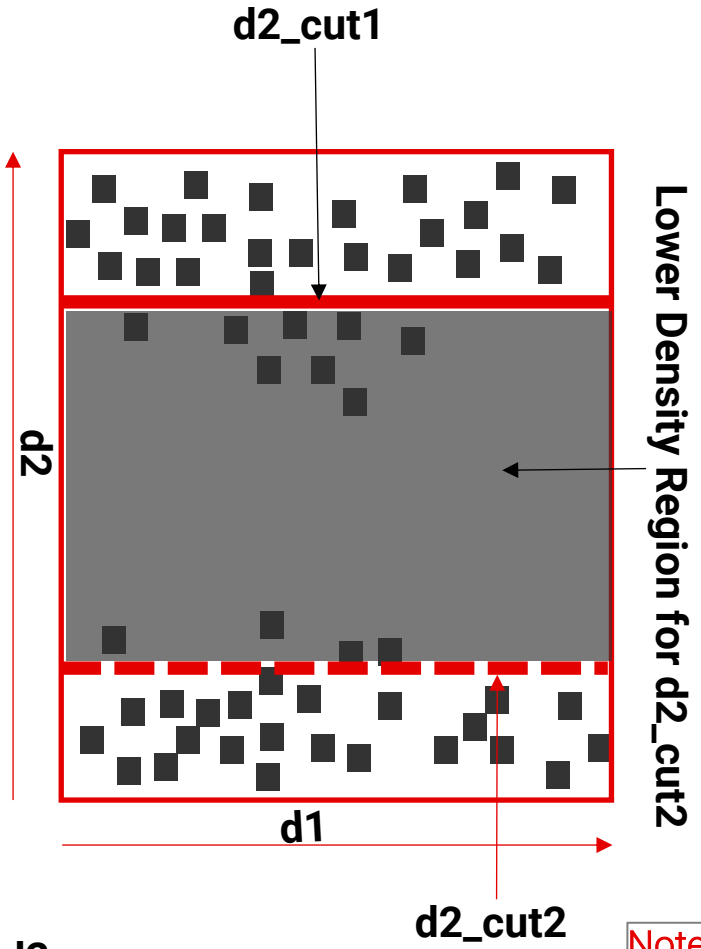
What Happens at Each Node When Evaluating Cuts(2)

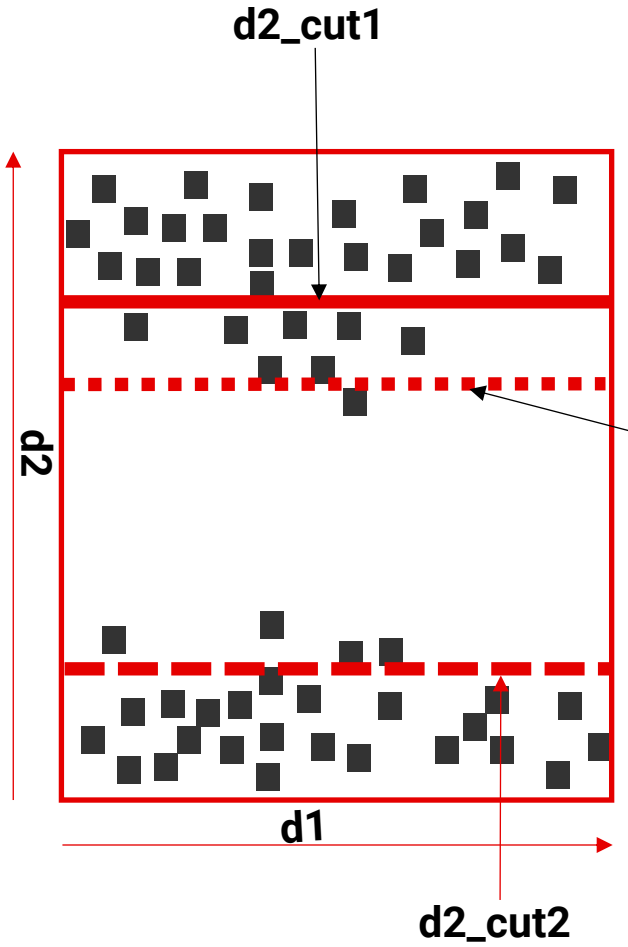


What Happens at Each Node When Evaluating Cuts(3)



d2_cut2 (Acquired by Information Gain on d2 on the Lower Density Region Subspace)





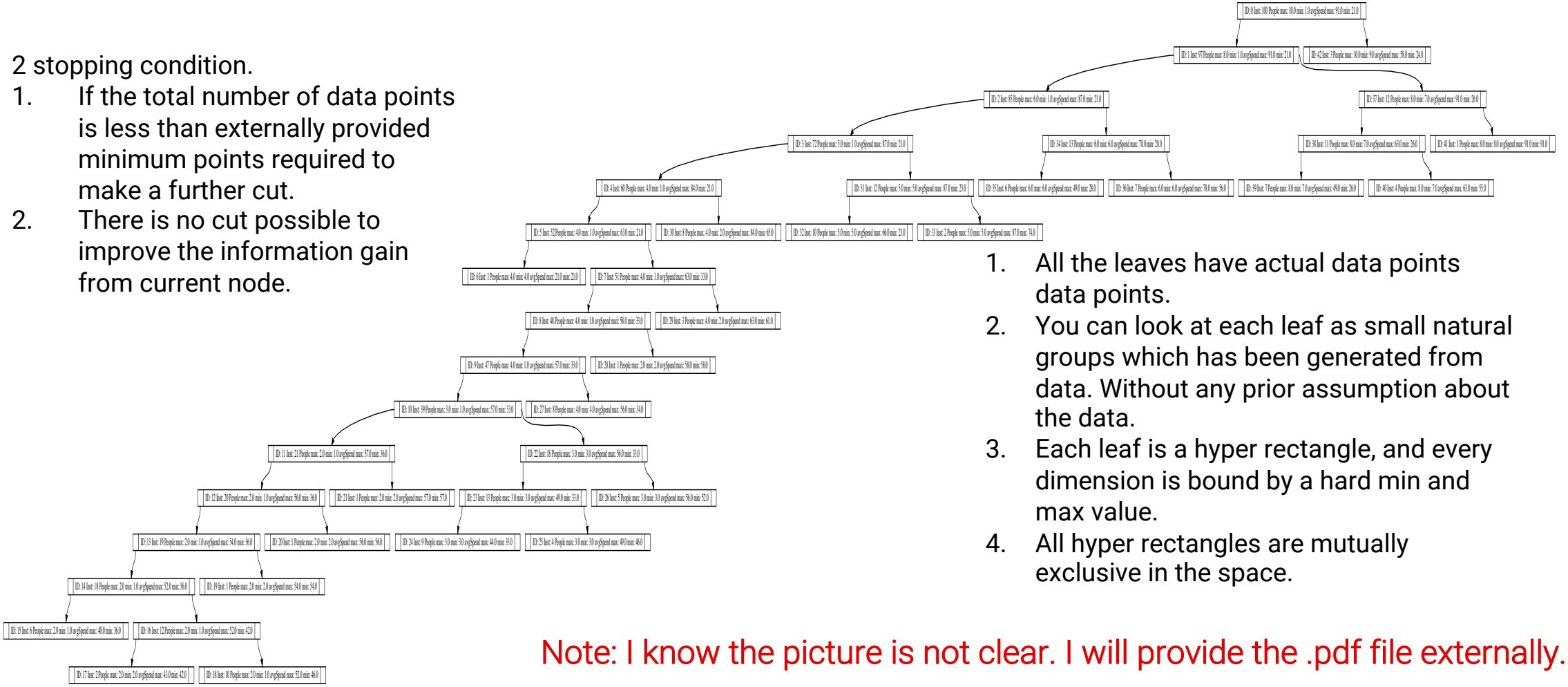
d2_cut3 (Acquired by Information Gain on d2 on the Lower Density Region Subspace)

Note: The final best cut at this level gets decided among [d1_cut2, d2_cut3] by lowest of lower relative density acquired by subsequent cuts.

Initial CLTree on Actual Data

2 stopping condition.

1. If the total number of data points is less than externally provided minimum points required to make a further cut.
2. There is no cut possible to improve the information gain from current node.



1. All the leaves have actual data points data points.
2. You can look at each leaf as small natural groups which has been generated from data. Without any prior assumption about the data.
3. Each leaf is a hyper rectangle, and every dimension is bound by a hard min and max value.
4. All hyper rectangles are mutually exclusive in the space.

Note: I know the picture is not clear. I will provide the .pdf file externally.

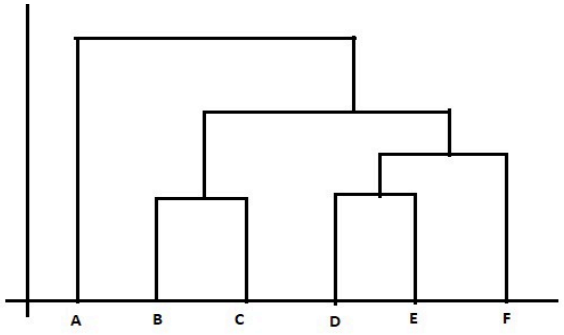
Assigning a Representation Vector to Each Leaf

- There is no restriction or assumption what can be the representation vector of a leaf as long as it represents the data points or the sub-space enclosed by the leaf in the space correctly.
- I have tried few ways and finalized on 3, for different cases.
  - Average Shopping/Buying Behavior of all the members of a underlying group/leaf:
    - Lets revisit one of the most critical reasons for which we started this in the first place. This is a proven fact that if a “personalization” category has a similar buying/shopping trend the MAB will produce personalize offers for that category in a fairly short time.
    - So, if buying/shopping behavior/record available for all the data points/persons/customers at each leaf, we can take “mean”/”Avg.” of those and assign it to each leaf as Representation Vector of the respective Leaf.
  - Average of demographic/”session” based information of all the members of the underlying group.
    - In the absence of the buying/shopping behavior/record we can reuse the data which each leaf/group already has.
    - We can take average of all the attributes within each leaf and assign that “mean”/”Avg.” vector to each leaf as Representation Vector of the respective Leaf.
  - Centroid of the “hyper rectangle” of each leaf/group.
    - Geometrically each leaf of the CLTree is a hyper rectangle, and every dimension is bound by a hard min and max value.
    - As each leaf is a hyper rectangle in the space and we also have max and mean value of all the attributes within each hyper rectangle, we can easily calculate “centroid” of all the leaves and assign that vector to each leaf as Representation Vector of the respective Leaf.

Gentle Introduction To Agglomerative Clustering

- Agglomerative Hierarchical clustering Technique:
  - In this technique, initially each data point is considered as an individual cluster. At each iteration, the similar clusters merge with other clusters until one cluster or K clusters are formed or some other terminating condition is met.

- The basic algorithm of Agglomerative is straight forward.
  1. Compute the proximity matrix
  2. Let each data point be a cluster
  3. Repeat: Merge the two closest clusters and update the proximity matrix
  4. Until only a single cluster remains



- The Hierarchical clustering Technique can be visualized using a Dendrogram.
- We are using “Ward’s Method” to calculate similarity between 2 groups/clusters at each iteration.

Conquering Leaf Nodes to Make Clusters(1)

- At this point we have small enough sized group to protect the granularity of our data as leaves of the CLTree.
- Each leaf has a representation vector.
- We have to merge leaves together to form manageable number of groups where each group which will have certain minimum data points.
- One thing to notice here, our MAB recommendation engine is sensitive towards the amount of traffic each cluster/category gets before a certain point, our goal is to make sure each cluster to have members close to that value.
- The collection of representation vectors from all the leaves has form the new data set for the “Conquering” algorithm. In our case it is Agglomerative Hierarchical clustering.
- Each node in CLTree has an ID and record of how many instances that node encompasses. These are helpful here because, after conquering, we would need to identify which leaves fall under same cluster and to check the “Satisfying Criteria” of min members of each cluster respectively.

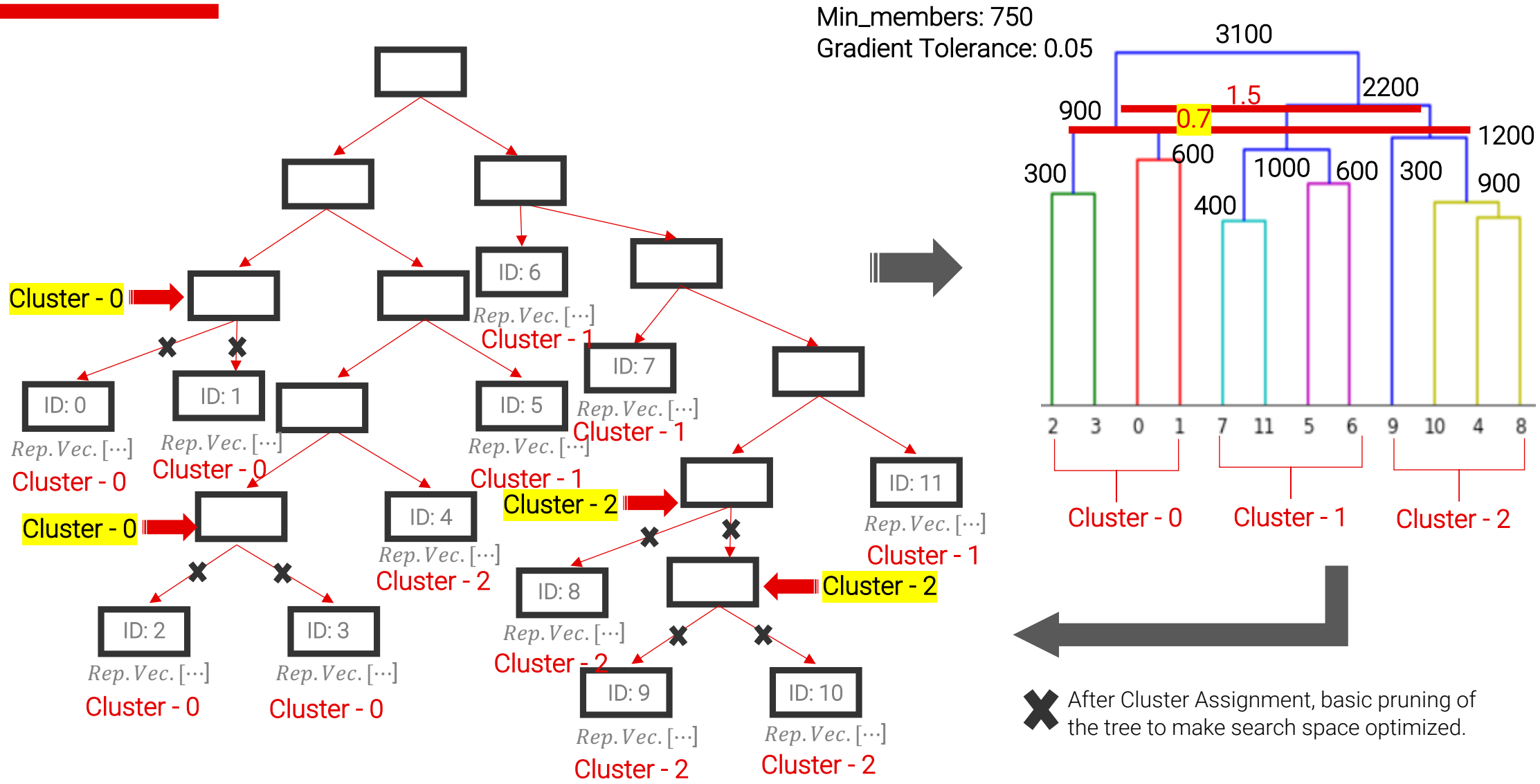
Conquering Leaf Nodes to Make Clusters(2)

- Objective function for our “conquering” algorithm/ Agglomerative Hierarchical clustering to ensure we get the highest quality clusters is,

$$\forall|\text{cluster}|_i \geq min - members, argmin\ (\frac{Avg.\ Intra{-}Cluster\ Distance\ in\ Current\ Setting}{Avg.\ Inter{-}Cluster\ Distance\ in\ Current\ Setting}).$$

- There are externally provided “gradient tolerance” parameter to boost-up the little bigger set of clusters over smaller set of clusters.
- gradient tolerance and $min{-}members$ are hyper parameters of the algorithm.
- Also, as the externally provided “min-members” are an estimate. We do a “Grid Search” on both sides of the estimate to certain extent to ensure get the highest quality clusters. Ex: lets say 1000 is provided as “min-members”, the “Grid-Search” will search from “950-1050” at the space of 10 numbers to get the highest quality clusters.
- Also, remember that the each vector/data point is a representation vector of certain number of data points which are at leaf nodes of CLTree. So, most of the time when 2 data points gets merged, the total size of the the new combined data point is the sum of sizes of both leaves or collection of leaves encompassed by 2 data points at the time of merge.

Assigning Cluster IDs to CLTree Leaves and Pruning



In the Absence of Any Data

- We have to fallback to rule based clustering for personalization.
- We still have to come up with a scalable way to implement variety set of rule depending underlying property.
- Rules can also come from the entity itself or from us or from any other sources.
- The only mandatory criteria for these rules are rules have to be “mutually exclusive and collectively exhaustive” in the space.
- As we ensure all the rules are “mutually exclusive and collectively exhaustive”, we can exploit the functionality of classic “Decision Tree” for classification to create a standardize way to create rule based clustering.
    - We know, how decision tree makes the cut. So, we can automatically curate a data set using the rules for all the “Classes” and use that curated dataset for classification using Decision Tree. The resultant tree we get we can use it anywhere to get the rule based cluster from data.
- We created a library which will create a “Decision Tree” from rule just call of one function. There are validations to ensure all the rules are mutually exclusive and collectively exhaustive. Any type of datatype is acceptable. So far we have standardized format of providing the rules to the function.

Some Other Functionalities Provided

- Balanced Clustering:
  - In dynamic data based clustering, sometimes it is desired to have a set of clusters where the total number of members in each clusters are almost same. Ex.: At the beginning of an experiment and we do not have any prior knowledge.
  - There is a “balancedCluster” flag with the “mergingCentroid” option, if we set this flag to “True” instead of optimizing on $\forall|\text{cluster}|_i \geq min - members, argmin\ (\frac{Avg.\ Intra-Cluster\ Distance\ in\ Current\ Setting}{Avg.\ Inter-Cluster\ Distance\ in\ Current\ Setting})$ he algorithm will optimize on, $\forall|\text{cluster}|_i \geq min - members, argmin(Variance\ among\ total\ number\ of\ cluster - members\ *\ (\frac{Avg.\ Intra-Cluster\ Distance\ in\ Current\ Setting}{Avg.\ Inter-Cluster\ Distance\ in\ Current\ Setting})$.

- Seasonality Detection:
  - What? Grouping together few months together based on some criteria (booking/traffic trend/sales statistics etc.) and use each group as a category rather than each month as a category.
  - Why? If we use some sort of month into the process of dividing the customer base, there is chance any algorithm will divide the data into 12 parts, but it doesn’t give any insight into personalization.
  - It is a collection of stateless algorithms, only assumption is there are 12 months data in 12 rows in a sorted order. Row 0 – January, Row 11: December.
  - Discrete: Group together months based on similarity irrespective of adjacency.
  - Continuous: Group together months only if they are adjacent to each other.
  - Manual: An entity always has a choice to provide us “month groups” externally.
  - It is an optional functionality and adds an additional level at the top of CLTree, based on “keyStringMonth” months.

Inference Model: Multi-Dimensional BST

- For all scenarios and their respective solutions, no matter how it is created get morphed into a simple Binary Search Tree.
- Once the clustering scheme has been built, we don't need this whole complex structure for assigning an end-user to a predefined cluster label.
- At inference, we need something, fast, accurate and light weight.
- There are couple of functions and an "Attribute" class build to support the process of translating the existing CLTree into a simple multi dimensional BST. At each level it checks on an attribute and based on the value the pointer moves to respective branch. Things to remember is, it is a multi-attribute BST, it can support any type of attribute and any type of comparison, {<=, >, op code: 1}, {!=, ==, Op. Code: 2} and {not in, in, !=, Op. Code: 3}.
- We save the final BST in a binary format to get retrieved and use of inference at transaction time.
- As we were building this final BST, we also make strategies for missing data at inference time.
- The final model has a very simple API, need only 2 basic python class definitions, ability to read pickle file and input in a dictionary format. That's set.


Sabre®