Module: Conceptual Clustering

Stephanie E. August
saugust@LMU.edu

Andrew Won
andyjwon@gmail.com

Eric Jaso
ericjaso@gmail.com

Michael Fraser
mfraser702@gmail.com

Miguel Vazquez
mvazque93@gmail.com

Poulomi Chatterjee
poulomi.chatterjee22@gmail.com

November 7, 2014

Artificial Intelligence Laboratory

This document is a TAILS module

Loyola Marymount University

1 The Idea

1.1 Purpose

The purpose of this lab is to familiarize the experimenter with machine learning in general and the COBWEB conceptual clustering algorithm specifically. Experimenter will be provided with a visualized simulation tool, which takes in objects defined and inputted by the experimenter and displays the dynamic spanning of the classification based on input.

1.2 Background

1.2.1 Machine Learning

People are able to learn new things. What exactly does it mean about machine learning? Arthur Samuel defined machine learning as a "field of study that gives computers the ability to learn without being explicitly programmed"[1]. In 1997, Tom M. Mitchell gave a more formal definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E"[2]. A possible example to explain E, P, T here could be the email spam filtering program. The email program learns from users which emails are marked as spam and which are not. Based on these data, the program learns to filter spam. In this machine learning case, E is watching user mark emails as spam or as not spam, T is classifying emails as spam or as not spam, and P could be the number of emails correctly classified as spam[3].

1.2.2 Supervised Learning VS Unsupervised Learning

Based on the desired outcome of the algorithm or the type of input available during training the machine, machine learning usually can be divided into two main types: the predictive or supervised learning approach and the descriptive or unsupervised learning approach[4]. The idea of supervised learning is to teach the program to do something, whereas unsupervised learning is going to let the program learn by itself.

In supervised learning, the algorithm is given data in which it is told the "correct answer"for each example[3]. Classification is probably the most widely used form of supervised learning. One example could be the email spam filtering algorithm mentioned above. Whenever a new email is detected, the algorithm will decide whether it is a spam or not and then put it in the right category the algorithm believes.

Unlike supervised learning, we are not told what the desired output is for each input in unsupervised learning. We are just given a bunch of data and will discover any interesting structure in the data. So in unsupervised learning, no one tells you which data is what. All the information lie in the input data. What we need to do is to formalize our task as one of density estimation[4]. The most common form of unsupervised learning is clustering.

1.2.3 Clustering

Clustering is the process of grouping similar objects together, so that objects within each cluster are similar to each other and from different clusters are dissimilar. According to the types of output, there are partitional clustering, where we partition the objects into disjoint sets; and hierarchical clustering, where we create a nested tree of partitions[4]. A simple example of data clustering is shown in Fig1. The input data is clearly distributed in six groups, and those data in the same group have the same label which is the desired output clusters.

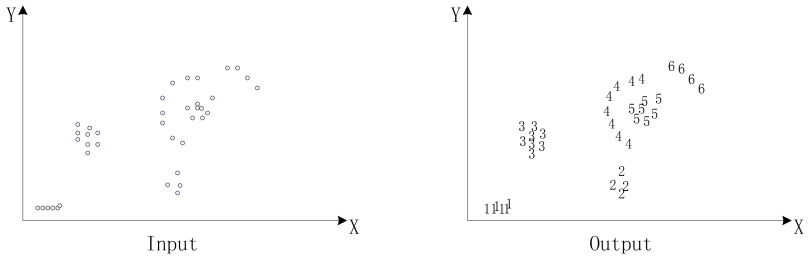


Figure 1: An example of data clustering

Hierarchical clustering algorithms are either top-down or bottom-up. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively merge (or agglomerate) pairs of clusters until all clusters have been merged into a single cluster that contains all data. Bottom-up hierarchical clustering is therefore called hierarchical agglomerative clustering. Top-down clustering requires a method for splitting a cluster. It proceeds by splitting clusters recursively until individual datum is reached[5].

1.2.4 An incremental conceptual clustering algorithm: COBWEB

Conceptual Clustering Conceptual clustering is closely related to data clustering, however, it is distinguished from ordinary data clustering by generating a concept description for each generated class. In conceptual clustering, it is not only the inherent structure of the data that drives cluster formation, but also the description language which is available to the learner. Thus, a statistically strong grouping in the data may fail to be extracted by the learner if the prevailing concept description language is incapable of describing that particular regularity[6].

Conceptual clustering algorithms are generally used in situations where we may not have all the data when we want to start organizing our findings. The main purpose of the conceptual clustering algorithm is to allow for data elements to be organized in relation to each other as more data becomes available.

In most implementations, the description language has been limited to feature conjunction, however, in COBWEB, the feature language is probabilistic.

Incremental Clustering Incremental clustering is based on the assumption that it is possible to consider patterns one at a time and assign them to existing clusters. A typical incremental

clustering algorithm is described as below[7]:

- Assign the first data item to a cluster;
- Consider the next data item. Either assign this item to one of the existing clusters or assign it to a new cluster. This assignment is done based on some criterion, e.g. the distance between the new item and the existing cluster centroids.
- Repeat the second step till all the data items are clustered.

COBWEB The COBWEB conceptual clustering algorithm[8] updates the tree (relation of all existing data elements) in accordance with new data that comes in. The tree this algorithm creates is devised in such a way that any *frontier node* is a data element and any *interior node* is a class which is a general representation of all of its children.

The simplest way of thinking of the COBWEB conceptual clustering algorithm is in terms of a child boy cleaning up his toys (assuming he's only paying attention to the toys he has already organized). Say a child picks up a firetruck first, so he puts the firetruck in its own bin, simply because no bin is in use yet. Then he picks up a bouncy ball. This toy doesn't seem to relate to the firetruck at all, so the child decides to put the bouncy ball in a separate bin from the firetruck. The next toy the child picks up is a racecar. So at this point the child can choose to put the racecar in its own bin, or in the same bin as the firetruck since the racecar and firetruck have a lot in common. The child decides to put the racecar in the same bin as the firetruck because they have enough similarities to share a bin. This is the general concept of the COBWEB conceptual clustering algorithm. As data is brought in, decisions are made.

Category Utility The first step of the COBWEB conceptual clustering algorithm is to determine the best spot for the new data element in the existing tree. That is, to determine which class the new data element fits into best. But one thing to note is that there are situations when the best option is to treat the new data element as its own entity and place it directly under the root node, rather than add it to an existing class.

But once a data element has been added to a class, there may be sub-classes into which it might fit better than the class it is currently assigned to. So in order to determine the best place for a new data element, the algorithm could possibly have to traverse all the way down to a class consisting of *only* frontier nodes.

The next question is, how do we know which spot is "best"for a new element? This is determined by the *categorical utility*[9] of adding the new data element to a certain spot in the tree. Category utility can be viewed as a function that rewards traditional virtues held in clustering generally–similarity of objects within the same class and dissimilarity of objects in different classes[8]:

- Intra-class similarity is reflected by conditional probability: $P(A_i = V_{ij} \,|C_k)$, where A_i denotes an attribute, V_{ij} denotes an attribute value, C_k denotes a class. The larger this probability, the greater proportion of class members sharing the value and the more predictable the value is of class members[8].

- Inter-class similarity is a function of $P(C_k|A_i = V_{ij})$. The larger this probability, the fewer the objects in contrasting classes that share this value and the more predictive the value is of the class[8].

Based on the intra-class and inter-class similarity, for a partition $\{C_1, C_2, ..., C_n\}$ which is a set of mutually exclusive classes, the following equation

$$E_1 = \sum_{k=1}^{n} \sum_{i} \sum_{j} P(A_i = V_{ij}) P(C_k|A_i = V_{ij}) P(A_i = V_{ij}\,|C_k\,)$$

overall measures the partition quality, where the probability $P(A_i = V_{ij})$ weights the importance of the individual attribute value. E_1 indicates that it is more important to increase the class-conditioned predictability and predictiveness of frequently occurring values than for infrequently occurring ones[8].

According to Bayes Rule, $P(A_i = V_{ij})P(C_k|A_i = V_{ij}) = P(C_k)P(A_i = V_{ij}|C_k)$. By substituting to E_1, we will get:

$$E_1 = \sum_{k=1}^{n} P(C_k) \sum_{i} \sum_{j} P(A_i = V_{ij}\,|C_k\,)^2$$

which is the ability to correctly guess the attributes of an object in a certain class.

Finally, category utility is defined by Cluck and Corter as the increase in the expected number of attribute values that can be correctly guessed $P(C_k) \sum_i \sum_j P(A_i = V_{ij}\,|C_k\,)^2$ over the expected number of correct guesses without such knowledge $\sum_i \sum_j P(A_i = V_{ij})^2$[8]:

$$CU = \frac{\sum_{k=1}^{n} P(C_k) \left[\sum_i \sum_j P(A_i = V_{ij}\,|C_k\,)^2 - \sum_i \sum_j P(A_i = V_{ij})^2 \right]}{n}$$

Here is an example: if a class has 8 green balls and 2 red balls and a different class has 4 green squares and 7 blue squares and the new element is a blue ball, the new blue ball will be placed in the first class according to the category utility since the CU value for this partition is the highest.

Table 1: Table of Classes

	Class 1	
A1	**A2**	**Number**
Green	Ball	8
Red	Ball	2

	Class 2	
A1	**A2**	**Number**
Green	Square	4
Blue	Square	7

By the definition above, we can say in this example, there are two classes C_1 and C_2 (shown in Table 1), and two attributes *color* and *shape* with five attribute values which are:

$$A_1 = \{\text{color}|V_{11} = \text{Green}, V_{12} = \text{Red}, V_{13} = \text{Blue}\}$$

$$A_2 = \{\text{shape}|V_{21} = \text{Ball}, V_{22} = \text{Square}\}$$

First, if we place the new element {Blue Ball} in Class 1(shown in Table 2), we can calculate CU for the current partition. The probability for C_1 is

$$P(C_1) = \frac{8+2+1}{8+2+1+4+7} = 0.5$$

and the probability for C_2 is

$$P(C_2) = \frac{4+7}{8+2+1+4+7} = 0.5$$

The expected number of attribute values that can be correctly guessed given C_1 is:

$$\begin{aligned}
&\sum_{i=1}^{2}\sum_{j=1}^{3} P(A_i = V_{ij}\,|C_1)^2 \\
=&(\frac{8}{8+2+1})^2 + (\frac{2}{8+2+1})^2 + (\frac{1}{8+2+1})^2 + (\frac{8+2+1}{8+2+1})^2 + 0 \\
=&1.5702
\end{aligned}$$

and the expected number of attribute values that can be correctly guessed given C_2 is:

$$\sum_{i=1}^{2}\sum_{j=1}^{3} P(A_i = V_{ij}\,|C_2)^2 = (\frac{4}{4+7})^2 + 0 + (\frac{7}{4+7})^2 + 0 + (\frac{4+7}{4+7})^2 = 1.5372$$

The expected number of attribute values that can be correctly guessed without any category knowledge is:

$$\sum_{i=1}^{2}\sum_{j=1}^{3} P(A_i = V_{ij})^2 = (\frac{8+4}{8+2+1+4+7})^2 + (\frac{2}{22})^2 + (\frac{1+7}{22})^2 + (\frac{8+2+1}{22})^2 + (\frac{4+7}{22})^2 = 0.9380$$

Thus, we can get CU for adding the new element to the Class 1:

$$CU_1 = \frac{0.5 \times (1.5702 - 0.9380) + 0.5 \times (1.5372 - 0.9380)}{2} = 0.30785$$

Second, in the same manner, we can also get CU for adding the new element to the Class 2(shown in Table 3), which is:

$$CU_2 = \frac{\frac{10}{22} \times (1.68 - 0.9380) + \frac{12}{22} \times (1.4028 - 0.9380)}{2} = 0.2954$$

Last, besides the possibility that placing the new element in an existing class, there is a chance for the new element generate a new class by itself. In the same manner, we can further calculate CU for generating a new class for the new element(shown in Table 4), which is

$$CU_3 = \frac{\frac{10}{22} \times (1.68 - 0.9380) + \frac{11}{22} \times (1.5372 - 0.9380) + \frac{1}{22} \times (2 - 0.9380)}{3} = 0.2284$$

Table 2: Table of Classes by adding the new element in C_1

C_1		
A1	**A2**	**Number**
Green	Ball	8
Red	Ball	2
Blue	Ball	1

C_2		
A1	**A2**	**Number**
Green	Square	4
Blue	Square	7

Table 3: Table of Classes by adding the new element in C_2

C_1		
A1	**A2**	**Number**
Green	Ball	8
Red	Ball	2

C_2		
A1	**A2**	**Number**
Green	Square	4
Blue	Square	7
Blue	Ball	1

Table 4: Table of Classes by generating a new class with the new element

C_1		
A1	**A2**	**Number**
Green	Ball	8
Red	Ball	2

C_2		
A1	**A2**	**Number**
Green	Square	4
Blue	Square	7

C_3		
A1	**A2**	**Number**
Blue	Ball	1

COBWEB Operations The COBWEB algorithm constructs a classification tree incrementally by inserting the objects into the tree one by one. When inserting an new object into the classification tree, the COBWEB algorithm descend the tree along a path of "best matching nodes"[10] and performing one of the following operations at each level[8]:

- Create the new object as a new class
- Add the new object to an existing class with the highest CU
- Merge two classes into a single class, or
- Split a class into several classes

In the following, we will demonstrate each operation by taking the example of animal descriptions from the reference[8].

Suppose the current classification tree has been built by '*fish*'and '*amphibian*'from Table5. The tree is shown in Figure2. Listed with each box(node) are the probability of the class and the conditional probabilities of attribute values given the current class.

Creating the new object as a new class If the new element '*mammal*'comes, by calculating category utility for classifying '*mammal*'with the existing class '*fish*', CU for classifying '*mammal*'with the other existing class '*amphibian*', and CU for creating a new class by '*mammal*'only, we will know where we should put the new element. It turns out that creating a new singleton class with '*mammal*'yields a better partition than adding the

Table 5: Table of Animal Descriptions

Name	Body Cover	Heart Chamber	Body Temp	Fertilization
Mammal	**Hair**	**4**	**Regulated**	**Internal**
Bird	**Feathers**	**4**	**Regulated**	**Internal**
Reptile	**Cornified-skin**	**Imperfect 4**	**Unregulated**	**Internal**
Amphibian	**Moist-skin**	**3**	**Unregulated**	**External**
Fish	**Scales**	**2**	**Unregulated**	**External**

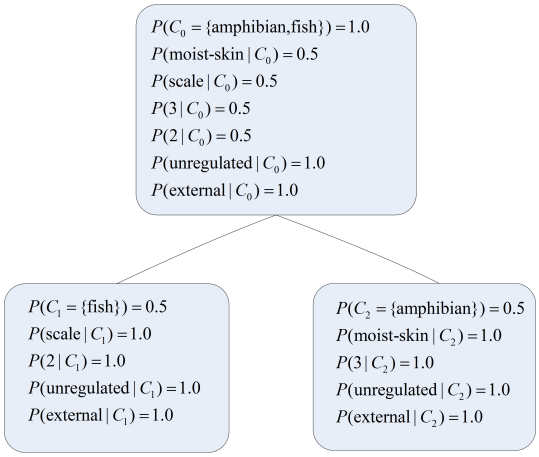


Figure 2: Classification tree built by *fish* and *amphibian*

element to either of the existing classes. Correspondingly, the tree will transit from the above form to the Figure 3.

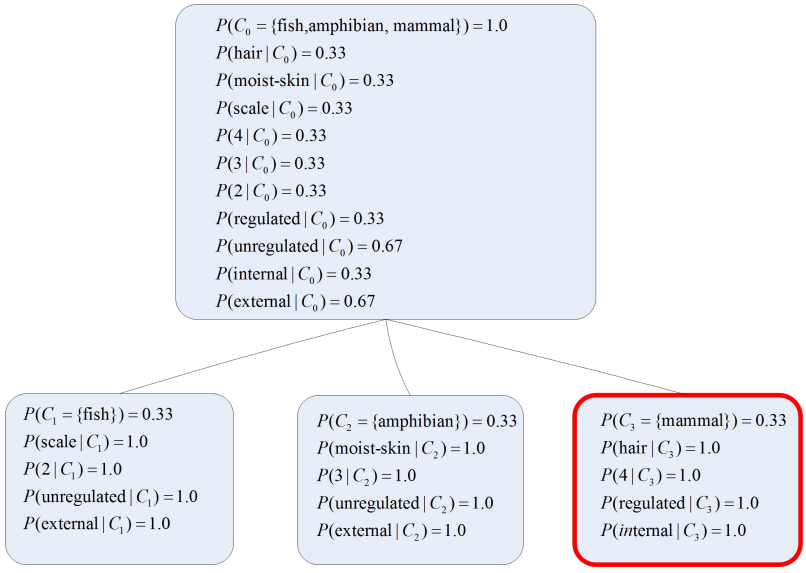


Figure 3: Adding '*mammal*' to the existing classification tree

Adding the new object to an existing class with the highest CU Next, if another new element '*bird*'comes, based on CU, it's easy to know the new element '*bird*'should be added to the existing class '*mammal*'. Since the node '*mammal*'in Figure3 is a leaf, incorporation of '*bird*'involves expanding the leaf to accommodate the new element '*bird*', as well as the previously classified one '*mammal*'. The classification tree after '*bird*'is added is shown in Figure4.

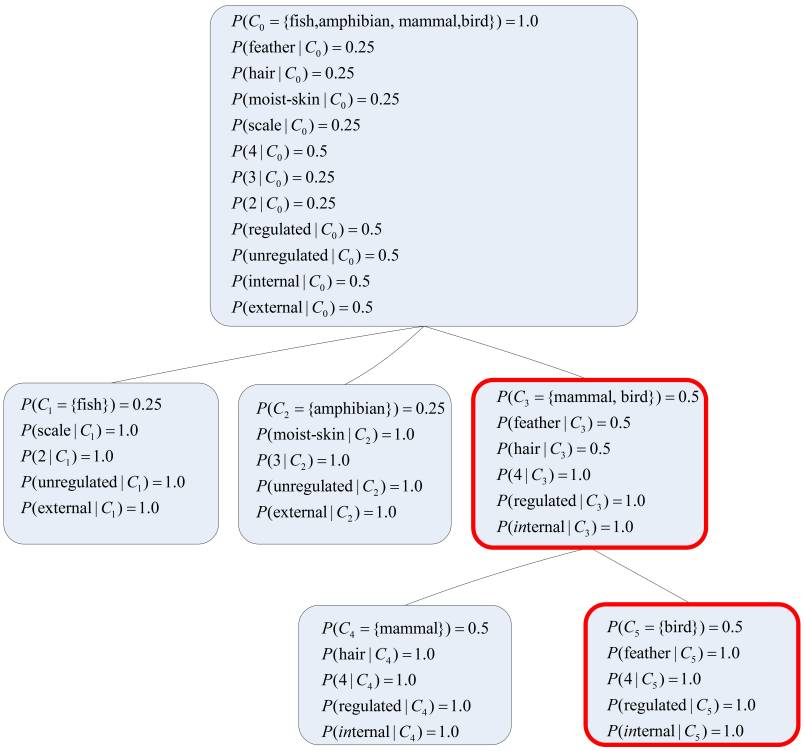


Figure 4: Adding '*bird*'to the existing classification tree

While the two operations, creating the new object as a new class and adding the new object to an existing class, are effective in many cases, by themselves they are very sensitive to the ordering of initial input. To guard against the effects of initially skewed data, COBWEB includes operations for node merging and splitting. When adding a new object, only the two classes with best CU are considered for merging[8].

Merging two classes into a single class Then, if another element '*fish*'comes, classes C_1 and C_2 in the Figure 4 are identified as the best and second best hosts for the new element '*fish*'. Merging C_1, C_2 and the new element '*fish*'results in a partition superior to that obtained by incorporating the new element in the best host C_1. The classification tree spanned from Figure 4 is shown in the Figure 5.

Splitting a class into several classes Splitting is the inverse operation of merging. Splitting is considered only for the children of the best host among the existing categories. A node of a partition (of n nodes) may be deleted and its children promoted, resulting in a partition of n + m - 1 nodes, where the deleted node had m children[8]. An example of

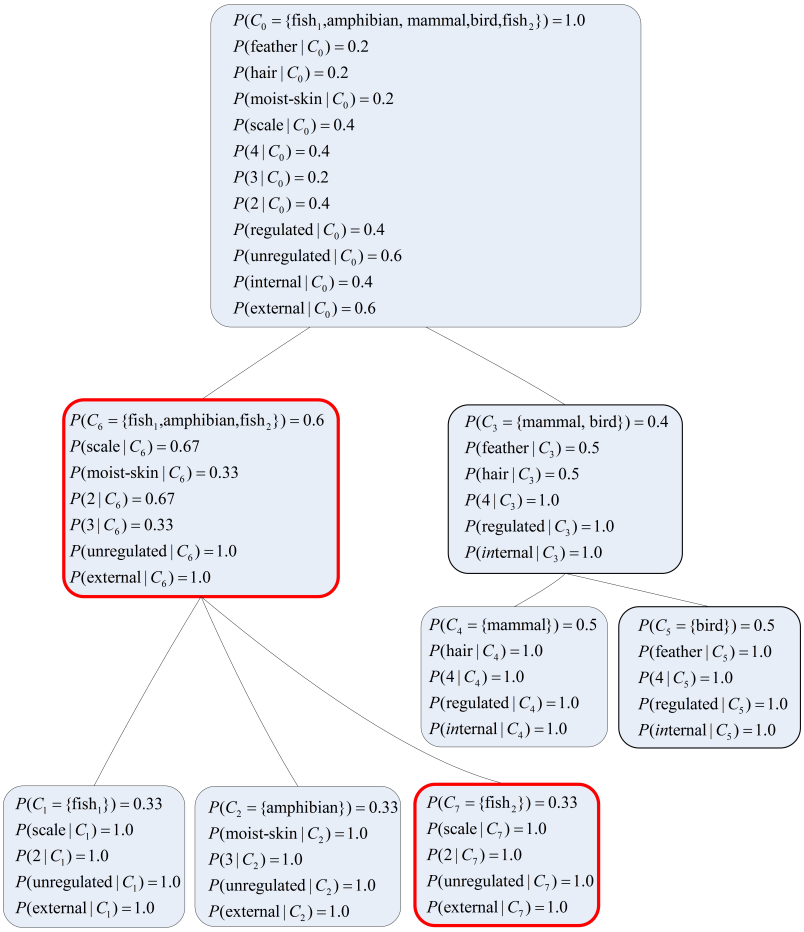


Figure 5: Adding '*fish*'to the existing classification tree

splitting is shown in Figure6.

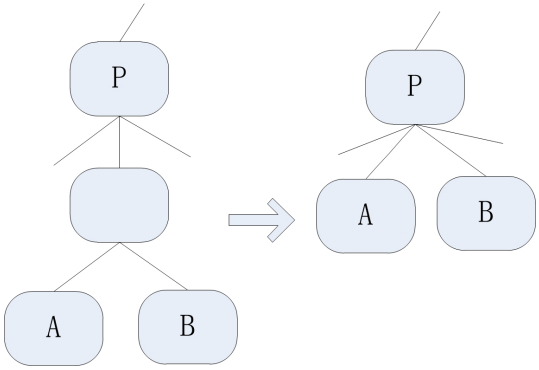


Figure 6: An example of node splitting

COBWEB Algorithm We have detailed the evaluation measure of COBWEB–category utility, and the four operations involved in COBWEB algorithm. The structure of COBWEB algorithm can be summarized as:

- Tentatively add to each existing portion
- Calculate CU of each
- Calculate CU for adding the new object directly to the root
- Add the new object to the child of the root with the highest CU
- As with all hill-climbing, presentation order determines initial tree structure
- MERGE and SPLIT operators minimize the effect of input ordering, and improve partition quality.

2 Applications

Conceptual clustering is widely used in our life. Learning itself is a kind of conceptual clustering. As a child grows up, he may learn to play toy blocks, count, recognize shapes and letters, he may learn maths, drawing, reading and writing, and later he may learn Artificial Intelligence, arts and English literature. Each time when a child learns something new, new knowledge will be stored in his knowledge base and relate to what he has already known or span a new space in his knowledge base. Similar knowledge is grouped together to form a category. As time goes by, the knowledge base broadens just like the classification tree spans. For example, when a student learns geometry in middle school, he may automatically add his previous experience with shapes to the new subject. Later when he learns architecture in college, he may probably reorganize his knowledge by adding geometry as the child node of architecture. Reference to Figure7. Those knowledge categories in mind help a person to respond to the problems from the outside world smoothly. For example, when the student participates a group project about designing a museum, he may need to search the related knowledge category in mind like architecture, recall related experience and finally reacts to the problem.

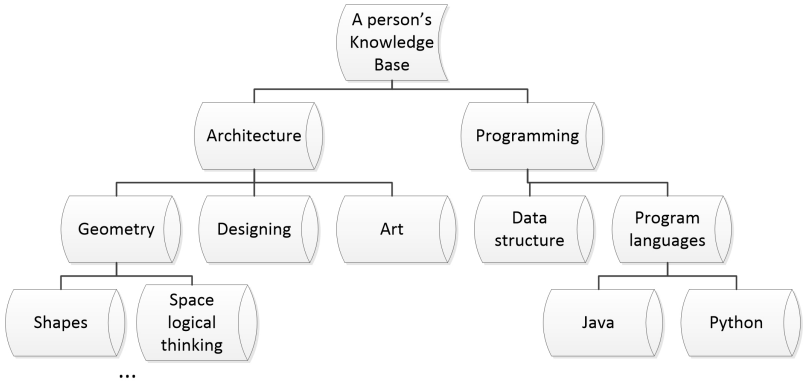


Figure 7: An example of classification tree for a person's knowledge base

Another possible example is that zoologist finds some new species from a isolated island. He may need to examine the new species from the structure or gene to decide if the new species belongs to any known class or it is a new class. The classification tree is shown in Figure 8

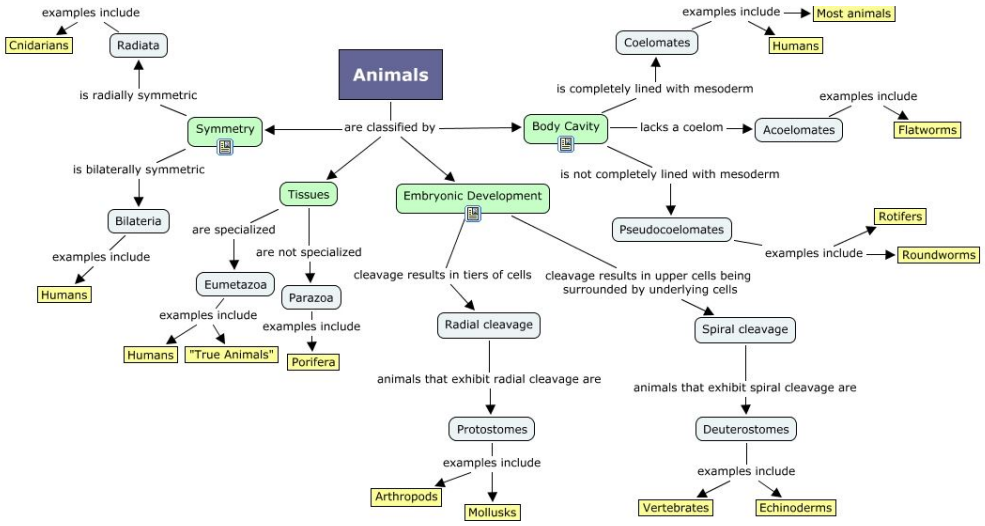


Figure 8: The classification tree for animals[11]

3 Input-Process-Output

3.1 Input

The input to this laboratory includes user-defined objects with various attributes and values. Let's say if we want to cluster different geometric figures, we may describe them in multiple aspects, like shape, color or size. It's possible to have triangle, circle or rectangle. So here, "shape", "color"and "size"are the attributes, and "triangle", "circle"and "rectangle"are the attribute values for the attribute "shape". The inputs for attributes and their values should be in the form of words only and it's case-insensitive. The input interface is shown in Figure 9.

Figure 9: The User Interface of Conceptual Clustering : Users can input custom data either by the browser input or uploading from a file, or users can use the default setting

If users want to input custom data, the tool allows for inputting data by the browser input or loading from a file. Using the browser input, users need to type in attribute names and their values accordingly. An attribute name should be typed in first in the attribute box and then click the "Add Attribute"button. The newly added attribute will be displayed under the input box. Moving the mouse on the attribute, the input box for attribute values will show up. Next, various values for the specific attribute can be typed in with the form of word only. An example is shown in Figure 10.

Users may also choose to load attributes and values from other documents by the box on the right. An example is shown in Figure 11. These attributes are copied from a text file, which are "color,size#orange,blue,red#large,small#". Clicking "Load Attributes"button,

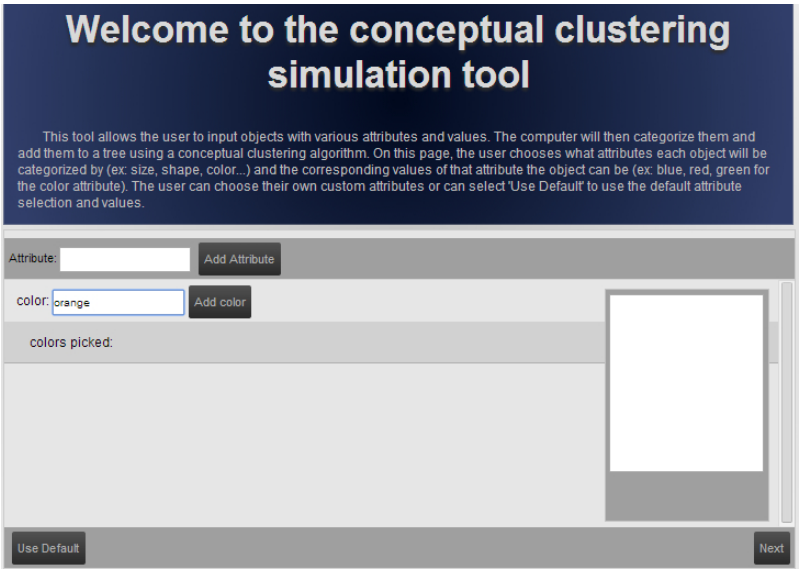


Figure 10: Input custom data by the browser input

the custom data will be inputted. Whatever is loaded, the contents should follow the following rules:

(1) Attribute or values should be in the format of word only;

(2) Any attributes or values should be separated by a comma without a space;

(3) Each line is separated by a pound sign "#";

(4) The contents in first line will be identified as the attributes and the other lines will be recognized as the attribute values for each attribute defined in the first line;

(5) An "Enter"at the end of each line makes no difference.

Except for inputting custom attributes, users can click the "Use Default"button to use the default attribute selections and values.

3.2 Process

The simulation tool will display all the attributes and their values in the second interface and enable users to input different objects from the radio buttons. The tool will categorize the input objects and add them to a classification tree based on the COBWEB conceptual clustering algorithm. The second interface is shown in Figure12.

On the left, one or multiple objects can be added to generate the classification tree by selecting different attribute values and clicking "Add"or "Add Multiple"button. The dynamic graph of generating the classification tree will be displayed, showing how the clustering algorithm works.

Moving the mouse on the node in the tree, the probabilistic concept label for this node will show up. The label includes the conditional probability for each attribute value given

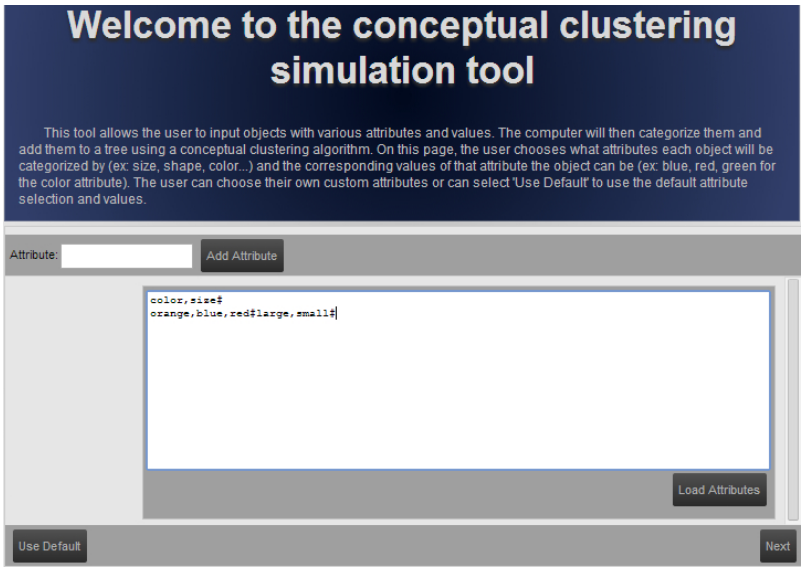


Figure 11: Input custom data by loading a file

the current cluster, which corresponds to $P(A_i = V_{ij} | C_k)$ and the percentage of the current cluster against the entire tree, which corresponds to $P(C_k)$ in the Category Utility function on Page 5.

On the bottom of the visual window, every time a new object is added, there will be an explanation for what specific operation is carried out for the current step. Those operations are denoted as "Added node", "Formed new subtree", "Merged"and "Split"which correspond to the four COBWEB operations listed in Page 7.

On the bottom of the tree display page, there is a input box allowing users to input objects by loading from other files. The format is the same as that for attribute, except that there should be an exclamation mark "!"separating the descriptions from one object to the next one. In the example shown in Figure 12, we copied "blue,small,circle!large,square,red!yellow,medium,green!" a text file. By clicking "Load Vector Values"button, these objects will be added to the tree.

3.3 Output

The classification tree keeps spanning as long as more objects are added. The classification tree indicates the clusters for current input space. All the leaf nodes that share the same parent node is grouped into one cluster.

3.4 Example

In this section, three experiments will be used to illustrate the effect of the input orders for the COBWEB algorithm. As we know, only one datum item is being considered at a time. The classification tree is sensitive to the input orders[8]. The two operations merging and splitting are able to minimize the effect of the input orders but can not remove the effect.

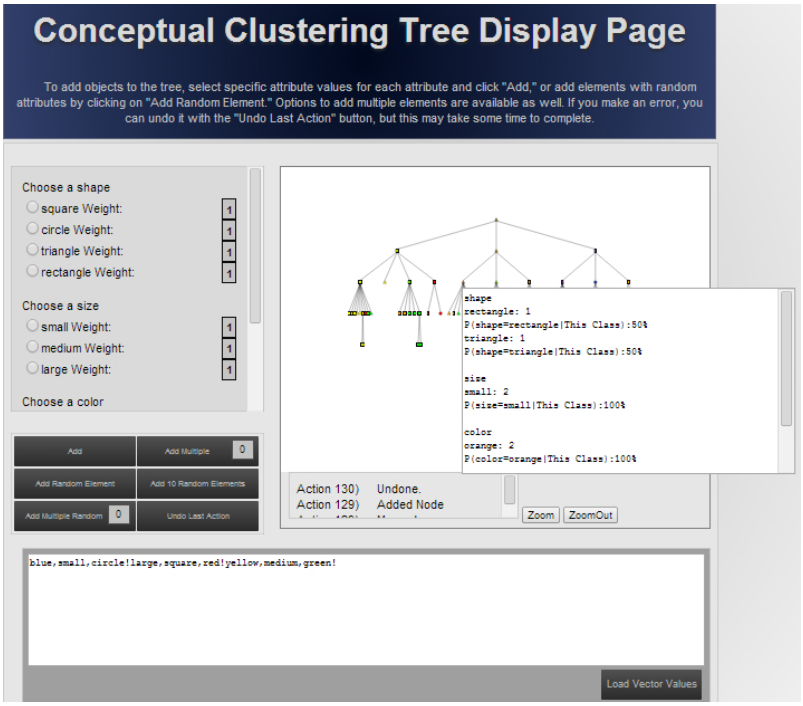


Figure 12: The process of generating the clustering tree

In order to observe the effect of the inputs orders, we carry out three experiments by using the same set of data in Table6.

Table 6: Table of input data

	shape	size	color		shape	size	color
1	**square**	**large**	**blue**	**6**	**circle**	**large**	**blue**
2	**square**	**large**	**yellow**	**7**	**circle**	**large**	**yellow**
3	**square**	**large**	**red**	**8**	**circle**	**large**	**red**
4	**square**	**large**	**orange**	**9**	**circle**	**large**	**orange**
5	**square**	**large**	**green**	**10**	**circle**	**large**	**green**

If we input the data with the sequence shown in the Table6, the tree will be generated in the way shown in Figure13.

In the second experiment, we first input the blue square and blue circle and then followed by the yellow square and yellow circle and so on so forth till all the ten objects have been added. It's obvious that the tree is generated in a different way shown in the Figure14.

In the third experiment, we input the first three circles and then followed by the five squares and then the last two circles. The tree is generated in another way that is different from either of the previous two. The tree generated in the third experiment is shown in Figure15.

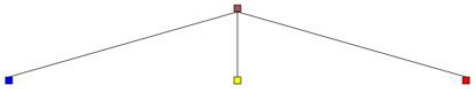
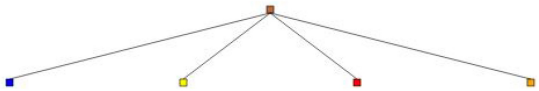
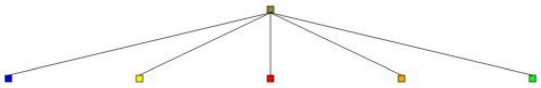
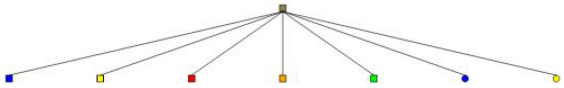
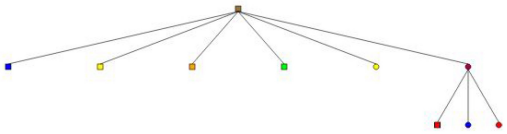
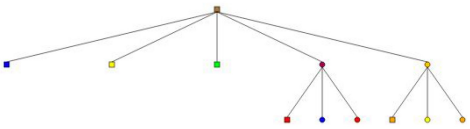
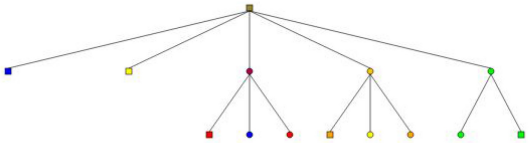
1	square	large	blue		Initialized
2	square	large	yellow		Added Node
3	square	large	red		Added Node
4	square	large	orange		Added Node
5	square	large	green		Added Node
6	circle	large	blue		Added Node
7	circle	large	yellow		Added Node
8	circle	large	red		Merged
9	circle	large	orange		Merged
10	circle	large	green		Formed New Subtree

Figure 13: The effect of input orders

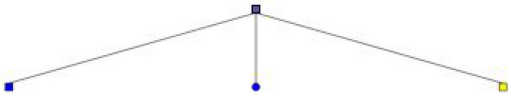
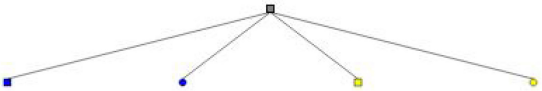
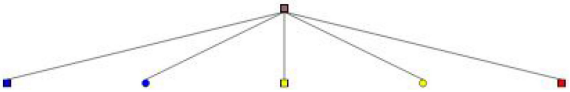
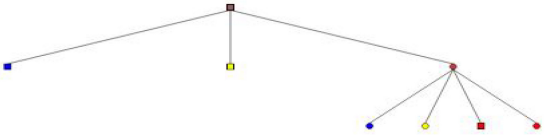
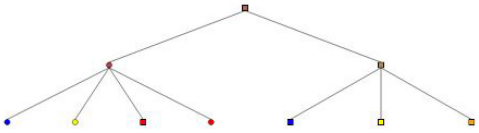
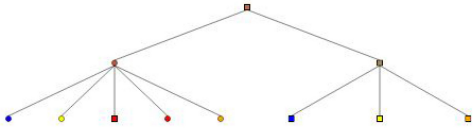
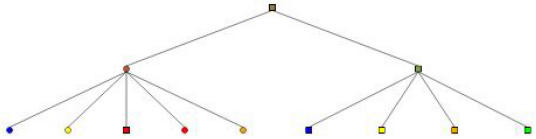
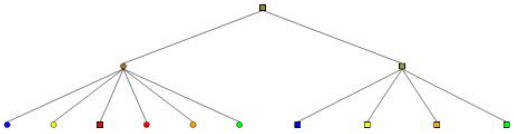
1	square	large	blue		Initialized
2	circle	large	blue		Added Node
3	square	large	yellow		Added Node
4	circle	large	yellow		Added Node
5	square	large	red		Added Node
6	circle	large	red		Merged
7	square	large	orange		Merged
8	circle	large	orange		Added Node
9	square	large	green		Added Node
10	circle	large	green		Added Node

Figure 14: The effect of input orders

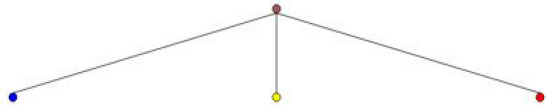
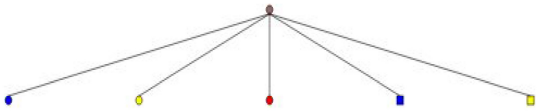
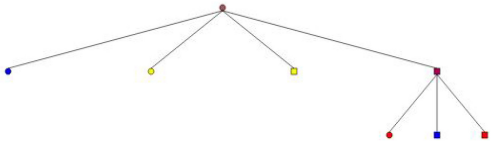
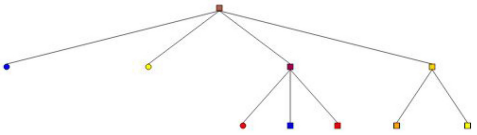
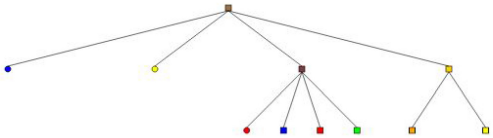
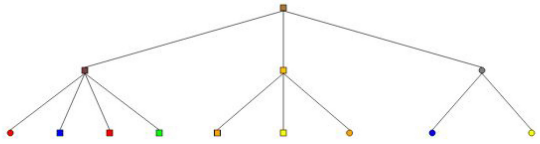
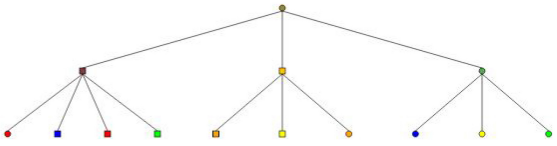
1	circle	large	blue		Initialized
2	circle	large	yellow		Added Node
3	circle	large	red		Added Node
4	square	large	blue		Added Node
5	square	large	yellow		Added Node
6	square	large	red		Merged
7	square	large	orange		Formed New Subtree
8	square	large	green		Added Node
9	circle	large	orange		Merged
10	circle	large	green		Added Node

Figure 15: The effect of input orders

4 Design

4.1 The Interface

Conceptual clustering module introduces COBWEB algorithm by presenting how a classification tree is generated based on user's input data. This simulation tool consists of two interfaces. The first one, corresponding to "index.html", is for user to define or describe the objects that he wants to cluster, and the second one, corresponding to "tree.html", is for user to input the defined object and observe the spanning of the classification tree.

4.2 The Class Diagram

The class digram is shown in Figure 16. This diagram displays the variables and functions defined in each JavaScript file, the connections between each two files and the information transfered between them.

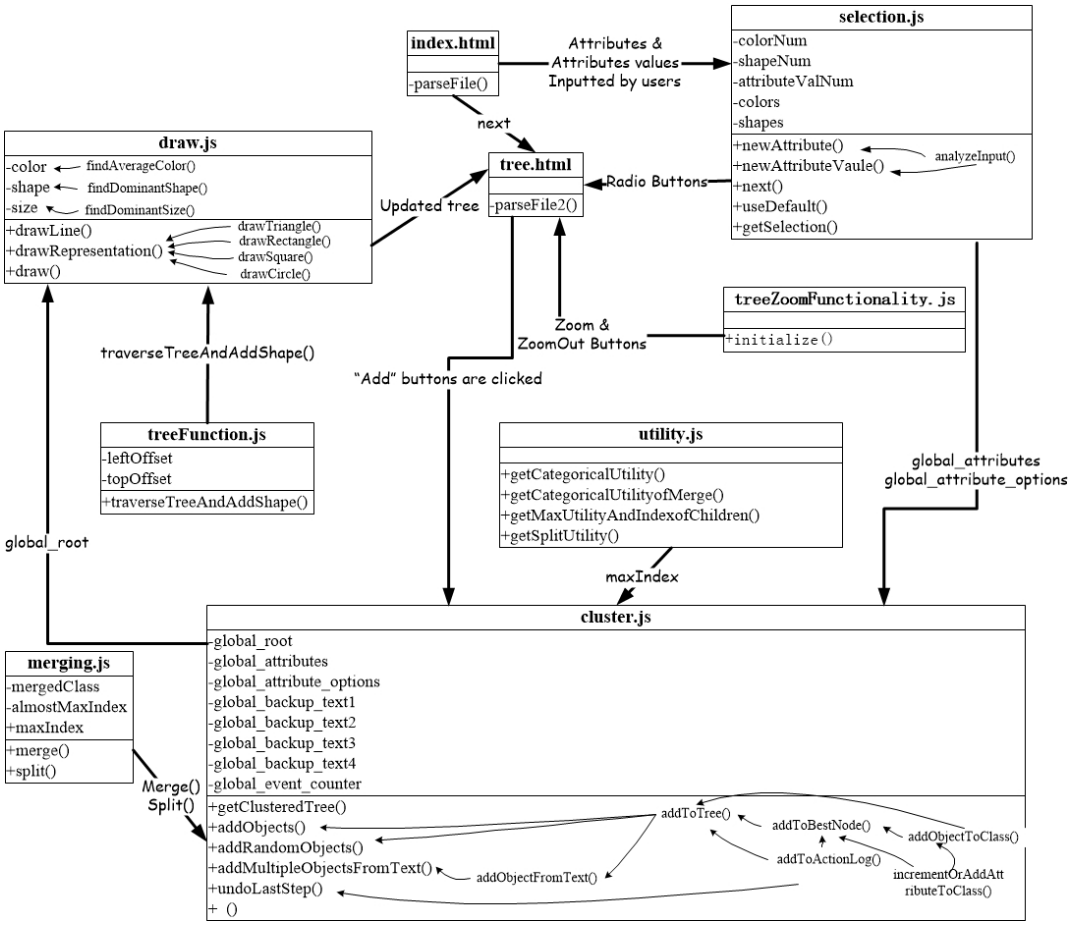


Figure 16: Class Diagram of TAILS Conceptual Clustering Module

4.3 The Sequence Diagram

The sequence diagram is shown in Figure 17. It displays what actions from users can trigger which JavaScript file, the calling sequence of all the files and which function or variable is being transmitted between them.

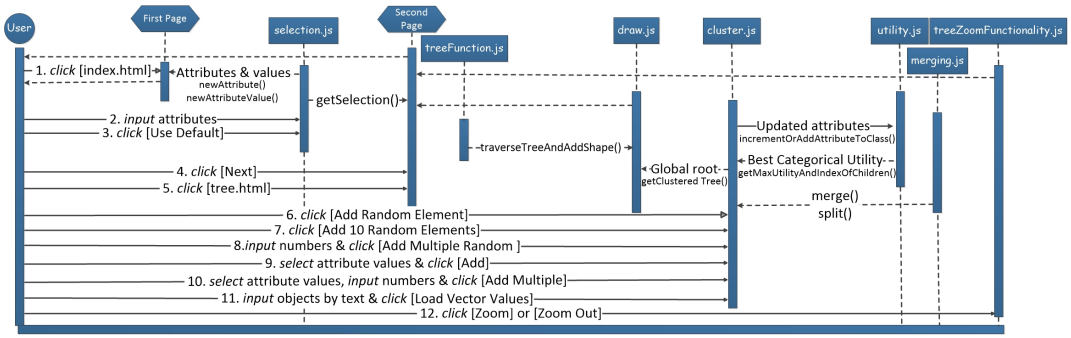


Figure 17: Sequence Diagram of TAILS Conceptual Clustering Module

5 Implementation-Specific Design

5.1 The Code

This program is built with JavaScript files: *selection.js*, *cluster.js*, *utility.js*, *merging.js*, *draw.js*, *treeFunction.js* and *treeZoomFunctionality.js*. *Cluster.js* is acting as the main script in the whole program. To aid with the implementation, the jQuery library and Raphael library are included as well.

index.html :

This file forms the first interface and calls *selection.js*.

selection.js :

This script gets the attributes and attribute values if they are written in the right format. These attributes and values will be kept in the variable *global_attributes* and *global_attribute_options* defined in the function *getSelection()* and will be used by *cluster.js*.

Function *getSelection()* also makes the radio buttons displayed in the second interface.

tree.html :

This file forms the second interface and calls each of the JavaScript files for the clustering algorithm and presents a visual display of the classification tree. In the second interface, *cluster.js* acts as the main function.

cluster.js :

A collection of functions which take in an object and add it to the tree in the proper location determined by a conceptual clustering algorithm. Once the object is added to the tree, the drawing is updated. *cluster.js* is the script that implements the COBWEB algorithm.

When the radio buttons are selected and "Add"or "Add Multiple"button is clicked, the program will go to the function *addObject()*.

When any of the three buttons that adds random object is clicked, the program will go to the function *addRandomObject()*.

When objects are inputted through the text and "Load Vector Values"button is clicked, the program will go to the function *addMultipleObjectsFromText()*.

When the "Undo Last Action"button is clicked, the program will go to the function *undoLastStep()*.

All of the function *addObject()*, *addRandomObject()*, and *addMultipleObjectsFromText()* calls the function *addToTree()*.

addToTree() calls *addToBestNode()* if the tree is not empty; otherwise, it calls the function *addObjectToClass()*.

addToBestNode() is the function that implements the COBWEB algorithm. It finds the best place to put the new node in the tree depending on its categorical utility that is derived

from the function *getMaxUtilityAndIndexOfChildren()* in the file *utility.js*. It also makes sure that if we're adding the object to a terminal node, to turn that node into a class with the two terminal nodes as its children. Once it is determined where the object should be placed, each attribute of the object is parsed through and added to the root node's attribute counts. It calls *addObjectToClass()*.

Variable *loopcheck* makes sure we're not in an infinite loop, that is trying to merge or split infinitely many times. Our stopping point for the infinite loop is trying to merge or split a full two cycles.

addObjectToClass() physically adds the new object to its parent class and the attribute counts for that class are updated. It calls the function *incrementOrAddAttributeToClass()*.

incrementOrAddAttributeToClass() increments each attribute in the class by one. If the attribute doesn't exist in that class yet, it creates it.

getClusteredTree() returns the handle on the root of the tree after the program executes the above functions and knows where to put the new object. *getClusteredTree()* will be used by the function *draw()* in the file *draw.js*.

utility.js :

This script generates the categorical utility of adding a new object to the tree in relation to a specific node. The equation it uses to generate the categorical utility is show in Page 5.

getMaxUtilityAndIndexOfChildren() acts as the main function in the *utility.js*. It creates an object that contains the two classes with the best categorical utilities. Then it decides if it is best to put the new object in an already existing class, merge two classes, or to make it its own class. It transmits the variable *maxIndex* and *maxUtil(utility)* to the function *addToBestNode()* in the file *cluster.js*. It calls *getCategoricalUtility()* and *getCategoricalUtilityOfMerge()*.

getCategoricalUtility() finds the expected number of attribute values that can be correctly guessed and adds this number to the predicted number of correctly guessed attribute values that already exist. *getExpectedNumberOfAttributeToGuess()* is called.

getCategoricalUtilityOfMerge() returns the categorical utility if the algorithm were to decide to merge the two classes with the highest categorical utilities. It calls *getExpectedGuessForMergedClasses()*.

getExpectedGuessForMergedClasses() temporarily adds the object to the hypothetically merged classes to compute the categorical utility of placing the object in this merged class.

getExpectedNumberOfAttributeToGuess() returns the number of expected attributes that would be correctly guessed for the class that is passed in.

getSplitUtility() calculates the categorical utility of splitting an already existing class into separate classes.

merging.js :

merge() takes the two objects/classes with the best categorical utility and first checks to see if they have children. If both of them are just terminal nodes, then it parses through their attributes and adds them to a parent node which will now be their parent class. Then

it adds the new object to this merged class. If one of them is a class, then it just adds the attributes of the other best object to the class. If they're both classes, it takes all of the children of one class and adds them as children to the other class.

split() adds all of the children of a class to the level the parent is at in the tree and then removes the parent class from the tree.

draw.js :

draw() acts as the main function in *draw.js*. It clears the paper if anything is there, then creates the raphael object and draws the tree. *draw()* gets the updated tree from the function *getClusteredTree()* in the *cluster.js* file. It calls the function *traverseTreeAndAddShape()* in the file *treeFunction.js* which takes in a tree and traverses that tree from the root down.

drawShape() and *drawLine()* are acting as the sub-main functions in file *draw.js*. *drawShape()* calls *findAverageColor()*, *findDominantShape()*, *findDominantSize()* and *drawRepresenataion()*.

drawLine() creates a line between the new node and its parent node.

drawRepresenataion calls *drawCircle()*, *drawRectangle()*, *drawTriangle()* and *drawSquare()*. These four functions create a corresponding shape at a specific spot on the canvas.

treeFunctions.js :

traverseTreeAndAddShape() first checks if the root node exists, if it doesn't then it creates the root node as well as the canvas on which the tree will exist. If the root node exists, it finds the placement of each of the child objects of the root node as well as drawing a line between each object and its parent node.

treeZoomFunctionality.js :

This script fulfills the zoom function in the second interface.

5.2 COBWEB Algorithm

The COBWEB Algorithm is:

- Cobweb(Node, Instance)
  begin
- if Node is a leaf then begin
  - create two children of Node, L1 and L2;
  - set the probabilities of L1 to those of Node;
  - initialize the probabilities for L2 to those of instance;

  - add instance of Node, updating Node's probabilities;
    end;

- else if Node is not a leaf
  add Instance to Node, updating Node's probabilities; for each child, C, of Node, compute the category utility of the clustering achieved by placing instance in C;

  - let S1 be the score for the best categorization, C1;

  - let S2 be the score for the second best categorization, C2;

  - let S3 be the score for placing instance in a new category;

  - let S4 be the score for merging C1 and C2 into one category;

  - let S5 be the score for splitting C1(replacing it with its child categories)
    end;

- if S1 is the best score, then cobweb(C1, Instance)

- else if S3 is the best score, then initialize the new category's probabilities to those of instance

- else if S4 is the best score, then cobweb (Cm, Instance), where Cm is the result of merging C1 and C2;

- else if S5 is the best score, then split C1; cobweb(Node, Instance)
  end;

6 Test Suite and Drivers

6.1 Using the default setting

Click *Use Default* in the first interface and then follow these five steps.

1) Choose *circle,large,green*, enter *8* in the *Add Multiple* button and click it.

2) Choose *circle,large,red*, enter *2* in the *Add Multiple* button and click it.

3) Choose *square,small,green*, enter *4* in the *Add Multiple* button and click it.

4) Choose *square,small,blue*, enter *7* in the *Add Multiple* button and click it.

5) Choose *circle,large,blue* and click the *Add* button.

Or you may choose *Use Default* in the first interface and then load the following text to the text box at the bottom of the second interface.
circle,large,green!circle,large,green!circle,large,green!circle,large,green!circle,large,green!
circle,large,green!circle,large,green!circle,large,green!circle,large,red!circle,large,red!
square,small,green!square,small,green!square,small,green!square,small,green!square,small,blue!
square,small,blue!square,small,blue!square,small,blue!square,small,blue!square,small,blue!
square,small,blue!circle,large,blue!

Either way, a classification tree shown in Figure18 will be generated. We can see that the last object, a blue circle is grouped to the class consists of two red circles and eight green circles. The remaining objects, including four green squares and seven blue squares are in the same class. This classification tree corresponds to the example on Page 5.

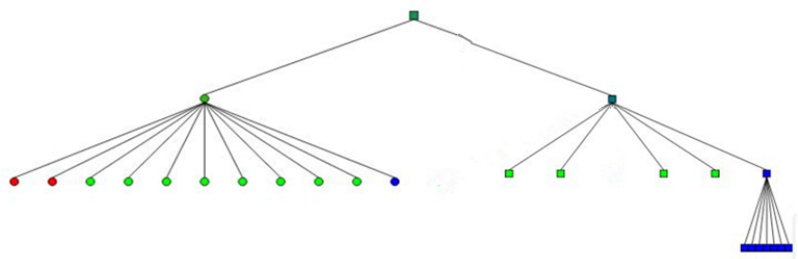


Figure 18

7 Experiments

The purpose of this section is to introduce students to the conceptual clustering simulation tool, clustering algorithms, feature vectors, probability, and the effect of input ordering on the learning that takes place when an application is trained on a set of data. Students with an understanding of Bayes Theorem will be able to study the category utility function used to rank the potential clusters each time a training vector is added to the set.

7.1 Experiment 1

Observe the effect of input ordering on COBWEB The first exercise provides students the opportunity to observe first-hand effect of input ordering on the taxonomy constructed by the COBWEB conceptual clustering algorithm[8, 12, 13].

Objective This exercise demonstrates via experimentation the value of random ordering of input to a learning algorithm. Students use an implementation of the COBWEB conceptual clustering algorithm to visualize the taxonomy of clusters created for a set of feature vectors given as input. The internal or summary nodes of the resulting tree are displayed as objects that reflect the average color and size of the objects in the subtree rooted at the node and the most frequently occurring shape in the subtree.

Experiment Description Start the Conceptual Clustering simulation tool and use the default attributes. Then input the following objects one by one in the order shown in Table 7 and Table 8 and observe the way the tree generated. Understand the concept label of parent nodes in each tree.

Table 7: Experiment1: Input Sequence 1

	shape	**size**	**color**		**shape**	**size**	**color**
1	**square**	**large**	**blue**	**6**	**circle**	**large**	**blue**
2	**square**	**large**	**yellow**	**7**	**circle**	**large**	**yellow**
3	**square**	**large**	**red**	**8**	**circle**	**large**	**red**
4	**square**	**large**	**orange**	**9**	**circle**	**large**	**orange**
5	**square**	**large**	**green**	**10**	**circle**	**large**	**green**

7.2 Experiment 2

Think of any possible application of conceptual clustering algorithm COBWEB in real life This exercise provides students the opportunity to think seriously about COBWEB algorithm and understand relevant concepts.

Table 8: Experiment 1: Input Sequence 2

	shape	size	color		shape	size	color
1	**square**	**large**	**blue**	**6**	**circle**	**large**	**red**
2	**circle**	**large**	**blue**	**7**	**square**	**large**	**orange**
3	**square**	**large**	**yellow**	**8**	**circle**	**large**	**orange**
4	**circle**	**large**	**yellow**	**9**	**square**	**large**	**green**
5	**square**	**large**	**red**	**10**	**circle**	**large**	**green**

Objective Experiment 2 allows students to demonstrate a deep understanding of the conceptual clustering algorithm COBWEB by describing an example in details. Students are expected to be able to tell a clustering problem from a classification problem, and understands concepts of attribute and attribute values by representing objects in their example using feature vectors.

Experiment Description Think of any possible application of conceptual clustering algorithm COBWEB in our life. Try to describe your example in the following aspects: 1)Why it is a clustering problem instead of a classification problem; 2)What might be the *concept label* for a cluster in your example; 3)What might happen when a new item is added in your example, referring to the four operations of COBWEB.

Then try to write down some possible attributes and attribute values of your example and keep it in a text file. Load your attributes and values through the text box in the first interface. Be sure that the attribute should be recorded in the first line and ended with a "#". The next line will be the values for the first attribute and so on so forth. The attributes and attribute values of a possible example about one kind of machines are shown here.
type,rate,weight,pricerange#
modelbased,conceptbased,experiencebased#
excellent,good,average,unknown#
heavy,light#
high,low#
verysmall,small,medium,large,extralarge#

Last, try to write down one possible order for the items in your example and keep it in a text file. Load your sequence of those items through the text box in the second interface. Be sure that each item should be ended with a "!". A possible input sequence for the example above is shown below:
modelbased,excellent,light,high,verysmall!
modelbased,good,light,high,verysmall!
modelbased,unknow,heavy,low,medium!
conceptbased,average,heavy,low,large!
conceptbased,average,heavy,high,verysmall!
conceptbased,excellent,heavy,high,verysmall!

experiencebased,good,light,low,medium!
experiencebased,unknown,light,high,small!

7.3 Experiment 3

Calculate the Category Utility for the following example This exercise is designed for those students who have a basic knowledge of probability, conditional probability and Bayes Theorem.

Objective This exercise demonstrates the ability to understand the evaluation function Category Utility in COBWEB algorithm. Students are expected to have a better idea of Category Utility by doing the calculations.

Experiment Description There are two classes, denoted as C_1 and C_2. Objects contained in C_1 and C_2 are shown in Table9. There are three attributes *color*, *shape* and *size* with seven attribute values which are: $A_1 = \{\text{color}|V_{11} = \text{Green}, V_{12} = \text{Red}, V_{13} = \text{Blue}\}$ $A_2 = \{\text{shape}|V_{21} = \text{Ball}, V_{22} = \text{Square}\}$ $A_3 = \{\text{size}|V_{21} = \text{Small}, V_{22} = \text{Large}\}$. Try to get the value of Category Utility for the current partition.

Hints: You might need to calculate the probability for each class, the expected number of attribute values that can be correctly guessed given each cluster and the expected number of attribute values that can be correctly guessed without any category knowledge, and then substitute those values in the CU function.

$$CU = \frac{\sum_{k=1}^{n} P(C_k) \left[\sum_i \sum_j P(A_i = V_{ij}\,|C_k)^2 - \sum_i \sum_j P(A_i = V_{ij})^2\right]}{n}$$

Table 9: Experiment3: Table of Classes

	Class 1	
A1	**A2**	**A3**
Green	Ball	Small
Green	Ball	Large
Red	Ball	Small

	Class 2	
A1	**A2**	**A3**
Green	Square	Large
Blue	Ball	Large
Blue	Square	Large

7.4 Experiment 4

Modify the codes in *cluster.js* The fourth experiment allows students extend the code to a small extent. It's easy to see the difference directly from the second interface.

Objective This exercise focuses on demonstrating knowledge of both JavaScript and the structure of the COBWEB module. It is a relative easy code exercise for those students who have a basic knowledge of programming.

Experiment Description The current edition only allows undoing four steps at most. Check the function *undoLastStep()* in the *cluster.js* and find out what variables are used to implement undoing function. Modify the codes and make it enable to undo five steps. Please remember to define anything that isn't there in the file *cluster.js*.

8 Source Code

8.1 Conceptual Clustering

The source code for is module can be found in the folder titled clustering working in the Conceptual Clustering Module folder.

9 Solutions for The Experiments of Conceptual Clustering

9.1 Solution for Experiment 1

The tree generated for the input sequence 1 and sequence 2 are shown in the following:

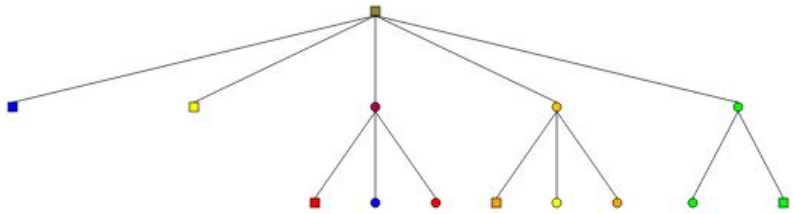


Figure 19: Experiment 1 with sequence 1

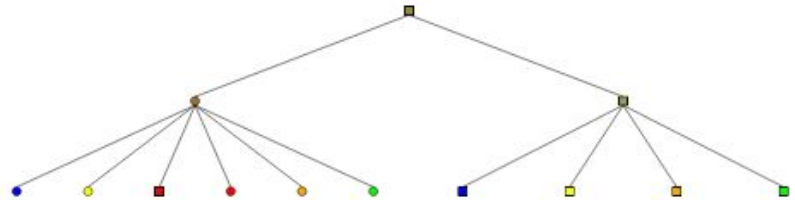


Figure 20: Experiment 1 with sequence 2

9.2 Solution for Experiment 2

Answers for this experiment are open. Here is one possible example. In an organization, people are grouped to different departments. Whenever new people come in, it will be decided which department they should join. Sometimes, several departments are merged to one and in some cases, one department might be broken down into several because of the newly added people. Attributes and values for this specific example could be:
experience,preference,age,gender,advantage#
extensive,relevant,none#
sales,management,logistics,nopreference#
female,male#
under,twenties,thirties,forties,above#
design,writing,communication#

9.3 Solution for Experiment 3

There are usually three key steps to get the category utility of a certain classification. First, get the probability of each class. It is very straightforward. You just need to calculate the

fraction of the number of objects in a certain class against the total number of objects. In this exercise, the number of objects in *Class 1* is three, the same for *class 2* and the total number of objects is six. Thus, the probability for C_1 is

$$P(C_1) = \frac{3}{6} = 0.5$$

and the probability for C_2 is

$$P(C_2) = \frac{3}{6} = 0.5$$

Second, get the expected number of attribute values that can be correctly guessed given a certain class. This is the conditional expectations (See Page 5). For each attribute value given *Class 1*, we can get the following table 10:

Table 10

		Class 1			
A1		**A2**		**A3**	
Green	$(2/3)^2$	**Ball**	$(3/3)^2$	**Small**	$(2/3)^2$
Red	$(1/3)^2$	**Square**	$(0/3)^2$	**Large**	$(1/3)^2$
Blue	$(0/3)^2$	–	–	–	–

The expected number of attribute values that can be correctly guessed given C_1 is:

$$\sum_{i=1}^{3}\sum_{j=1} P(A_i = V_{ij}\,|C_1)^2 = (\frac{2}{3})^2 + (\frac{1}{3})^2 + 0 + 1 + 0 + (\frac{2}{3})^2 + (\frac{1}{3})^2 = 2.11$$

For each attribute value given *Class 2*, we can get the following table 11:

Table 11

		Class 1			
A1		**A2**		**A3**	
Green	$(1/3)^2$	**Ball**	$(1/3)^2$	**Small**	$(0/3)^2$
Red	$(0/3)^2$	**Square**	$(2/3)^2$	**Large**	$(3/3)^2$
Blue	$(2/3)^2$	–	–	–	–

The expected number of attribute values that can be correctly guessed given C_2 is:

$$\sum_{i=1}^{3}\sum_{j=1} P(A_i = V_{ij}\,|C_2)^2 = (\frac{1}{3})^2 + 0 + (\frac{2}{3})^2 + (\frac{1}{3})^2 + (\frac{2}{3})^2 + 0 + 1 = 2.11$$

Last, get the expected number of attribute values that can be correctly guessed without any category knowledge. This is the unconditional expectations.

Table 12

A1		**A2**		**A3**	
Green	$(3/6)^2$	**Ball**	$(4/6)^2$	**Small**	$(2/6)^2$
Red	$(1/6)^2$	**Square**	$(2/6)^2$	**Large**	$(4/6)^2$
Blue	$(2/6)^2$	–	–	–	–

The expected number of attribute values that can be correctly guessed without any category information is:

$$\sum_{i=1}^{3}\sum_{j=1} P(A_i = V_{ij})^2 = (\frac{3}{6})^2 + (\frac{1}{6})^2 + (\frac{2}{6})^2 + (\frac{4}{6})^2 + (\frac{2}{6})^2 + (\frac{2}{6})^2 + (\frac{4}{6})^2 = 1.5$$

Thus, we can get Category Utility for the current classification:

$$CU = \frac{0.5 \times (2.11 - 1.50) + 0.5 \times (2.11 - 1.50)}{2} = 0.305$$

9.4 Solution for Experiment 4

If we go to the function *undoLastStep()* in the file *cluster.js*, we can see that there are four variables that hold the string representation of the cluster. These four variables are *global_backup_text1, global_backup_text2, global_backup_text3 and global_backup_text4*. If you search any of them, you will see that they are defined in the very beginning of the *cluster.js*.

In order to see how the initial value is passed to the *global_backup_text*, we need to search the first variable *global_backup_text1*. Then you will find out that variable *global_backup_text1* gets sting in the function *addObjects()* and function *addRandomObjects()*. Once the variable *global_backup_text1* is updated, its previous string will be passed to the variable *global_backup_text2* and so on so forth.

Then we go back to the function *undoLastStep()* and now it is easy to know that if we want to undo the last step, we need to recover the values for the variables *global_backup_text*. If we want to undo five steps, a new variable *global_backup_text5* needs adding in this file. In order to solve this problem, there are four places that needs modifying in *cluster.js*.

1) Add *var global_backup_text5 = "";* in the very beginning of *cluster.js* to define it;

2) Add *global_backup_text5 = global_backup_text4;* before the code *global_backup_text4 = global_backup_text3;* in the beginning of the function *addObjects()* to make sure that *global_backup_text5* gets the value;

3) Similar to the second. Add *global_backup_text5 = global_backup_text4;* before the code *global_backup_text4 = global_backup_text3;* in the middle of the function *addRandomObjects()*;

4) The last place that needs modifying is in the function *undoLastStep()*. In the *else* clause, add *global_backup_text4 = global_backup_text5;* in front of the code *global_backup_text4 = “”;* and change the code *global_backup_text4 = “”;* to be *global_backup_text5= “”;*

5) Restart the Conceptual Clustering simulation tool and the use the default setting. Add the objects more than five times and then try “Undo Last Action ”button. You will be allowd to undo the last five steps instead of four.

References

[1] Phil Simon, *Too Big to Ignore: The Business Case for Big Data*, John Wiley & Sons, 2013.

[2] Tom M Mitchell, "Machine learning. 1997," *Burr Ridge, IL: McGraw Hill*, vol. 45, 1997.

[3] Andrew Ng, "Introduction, what is machine learning," "http://openclassroom.stanford.edu/MainFolder/VideoPage.php?course=MachineLearning&video=01.2-Introduction-WhatIsMachineLearning&speed=100".

[4] Kevin P Murphy, *Machine learning: a probabilistic perspective*, MIT Press, 2012.

[5] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze, *Introduction to information retrieval*, vol. 1, Cambridge university press Cambridge, 2008.

[6] "Conceptual clustering," "http://en.wikipedia.org/wiki/Conceptual_clustering".

[7] Anil K Jain, M Narasimha Murty, and Patrick J Flynn, "Data clustering: a review," *ACM computing surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[8] Douglas H Fisher, "Knowledge acquisition via incremental conceptual clustering," *Machine learning*, vol. 2, no. 2, pp. 139–172, 1987.

[9] M Gluck, "Information, uncertainty and the utility of categories," in *Proc. of the Seventh Annual Conf. on Cognitive Science Society*, 1985, pp. 283–287.

[10] G.F. Luger, *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*, Pearson Education, 2011.

[11] "Classification map," "http://ci-cmap.oit.umn.edu/rid=1KQ0B0XBH-3J1L9H-P6W/Animal%20Classification.cmap".

[12] Douglas Fisher, "Optimization and simplification of hierarchical clusterings.," in *KDD*, 1995, pp. 118–123.

[13] John H Gennari, Pat Langley, and Doug Fisher, "Models of incremental concept formation," *Artificial intelligence*, vol. 40, no. 1, pp. 11–61, 1989.