VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Mathematics, Statistics and Computer Science
Te Kura Tatau

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@mcs.vuw.ac.nz

Machine Learning tools for the Weka Workbench: clustering algorithms

Jessica Kampmann

Supervisor: Dr. Peter Andreae

Submitted in partial fulfilment of the requirements for
Master of Computer Science.

Abstract

The goal of this project was to integrate the MAXCCLUS clustering algorithm into the WEKA (Waikato Environment for Knowledge Analysis) toolkit. To integrate the existing implementation of the MAXCCLUS algorithm into the WEKA machine learning toolkit it was necessary to understand clustering algorithms in general, the MAXCCLUS algorithm in particular, and the WEKA toolkit. The integration process involved serveral tasks, such as implementing a data conversion tool and extending the WEKA implementation.

Acknowledgments

I'd like to thank my supervisor Dr. Peter Andreae for his support and suggestion.

Thanks to my parents for tolerating my phone calls early in the morning and their entertaining stories about broken vaccum cleaners. I would also like to thank my sister Sarah for the long phone calls and her support.

Thanks to my flatmates Stacey, Jon, Andre and Rob for the catering service they provided during the last two weeks.

And last but not least thanks to Luke and Johannes for proof reading my report, thanks to Kim for making move to Memphis and all inhabitants and vistors of Memphis for their company and support.

Contents

Figures

Chapter 1

Introduction

The aim of the project is to integrate the MAXCCLUS clustering algorithm into the WEKA (Waikato Environment for Knowledge Analysis) toolkit for machine learning techniques.

MAXCCLUS is not a typical clustering algorithm because, unlike standard clustering algorithms it requires the data to already be classified. MAXCCLUS was designed to make sense of out bioinformatics data by identifying significant clusters of related instances and constructing rules that describe the clusters.

MAXCCLUS processes two data components. The first one contains all instance classifications; the second data input consists of lists of words that describe each instance. These words are also called descriptors. MAXCCLUS produces a set of clusters and rules as results. Each cluster is characterized by a classification and at least one rule. A rule contains a set of descriptors that describes all instances in the cluster and provides information about the instances.

WEKA(Waikato Environment for Knowledge Analysis) is a machine learning toolkit developed by the University of Waikato. It provides a wide selection of machine learning algorithms. As it is an open source project, it is published under the GNU General Public License.

WEKA was designed to be extensible and provides a platform for distribution and evaluation of new machine learning algorithms from three categories: classification algorithms, clustering algorithms, and associating algorithms. Also the WEKA toolkit with its selection of implementations of standard algorithms makes it easy to compare new techniques to existing ones. WEKA is implemented in Java and so it is platform independent. WEKA uses its own data format named ARFF (Attribute-Relation-File-Format). WEKA is a user-friendly tool as it comes with a graphical user interface.

To integrate the existing implementation of the MAXCCLUS algorithm into the WEKA machine learning toolkit it was necessary to understand clustering algorithms in general, the MAXCCLUS in particular, and the WEKA toolkit. The first step of the project involved the understanding of basic clustering algorithms like K-Means, Cobweb and the MAXCCLUS algorithm. The following tasks were to understand the current implementation of MAXCCLUS and to analyze and understand the implementation and the software design of WEKA framework with a special focus on its requirements for integrating a new algorithm.

Given this understanding it was then possible to identify the changes and extensions required to port and re-implement the MAXCCLUS algorithm, and integrate it into the WEKA toolkit.

The next chapter of the report first outlines a theoretical background about clustering algorithms by explaining and describing some standard algorithms, like K-Means clustering, agglomerative clustering and COBWEB clustering then describes the MAXCCLUS algorithm with a focus on its differences from standard clustering algorithms. The final part of the chapter is about the WEKA toolkit and its requirements for additional implementation of a new clustering algorithm.

Chapter 3 describes the problems and issues that arose the integration process and the solutions to solve these problems.

The final chapter will outline some strength and weaknesses of the WEKA framework with a focus on integrating new machine learning algorithms; and it will outline a evaluation the WEKA version of MAXCCLUS.

Chapter 2

Background

2.1 Clustering algorithms

The task of clustering is based on the idea that objects can be classified into different groups. Clustering partitions a data set into subsets (groups). These subsets are called clusters. The goal of the partitioning process is to divide the data into clusters so that the members in a cluster share as many attribute values as possible. Most clustering algorithms measure the level of similarity between two objects (instances) based on a pre-defined distance function or based on probability metrics. The clustering algorithms described in the following subsections are very well know standard clustering algorithms that belong to two different clustering groups partitional clustering and hierarchical clustering.

2.1.1 K-Means clustering

The clustering technique called ″K-Means″ is one of the most basic and popular methods and belongs to the group of partitional clustering techniques. This clustering approach needs to know the final number of clusters in advance to order the data into groups. ″K″ in the name ″K-Means″ refers to the number of clusters into which the instances should be divided. The K-Means method is based on the minimization of the total variance of each cluster and is also related to the machine learning technique called ″Nearest Neighbour″ that is used for data classification.

The general process of the algorithm starts with an assignment of k-cluster centres. Those cluster centres are randomly selected instances out of the data set. Afterwards, each instance is assigned to the cluster whose centre is nearest to the instance. In the next step the algorithm recalculates the centre of each cluster based on the smallest total variance. The new centres have the minimal distance to all instances whose belong to the cluster. If the all attributes have numeric values then the new centre are the mean values of the cluster instances. Otherwise the centres of the clusters have to be represented by the instance so the new centre is instance that gives the smallest total variance. In the final step the algorithm uses the new centres to re-assign all instances to the new clusters. These steps will be repeated until the centres do not change after the re-calculation of the centres.

The algorithm computes the centres of a cluster by evaluating a distance function. The algorithm can easily calculate the distance between numeric attributes. A very common distance metric for two dimensional instances with numeric attributes would be:

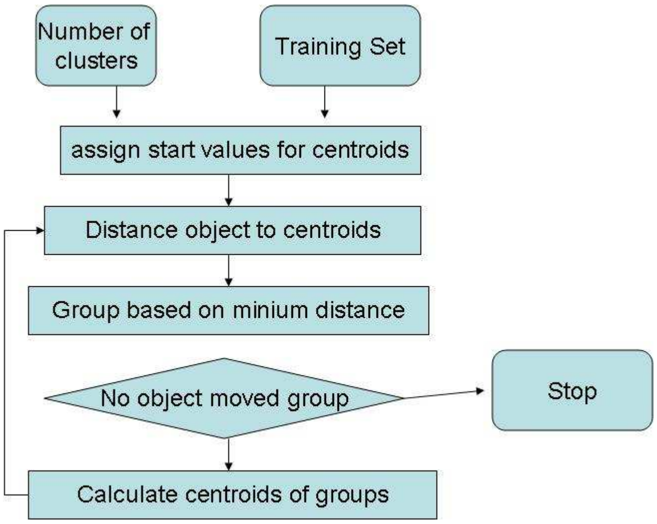

Figure 2.1: Diagram of the K-Means algorithm.

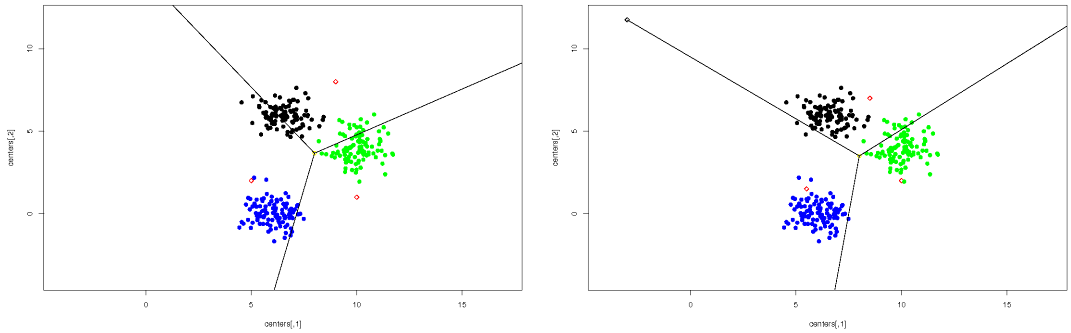

Figure 2.2: Steps K-Means

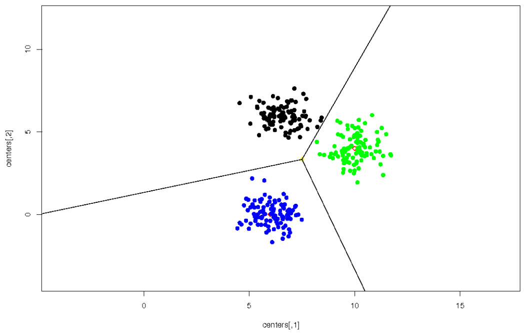

Figure 2.3: Final cluster by K-Means

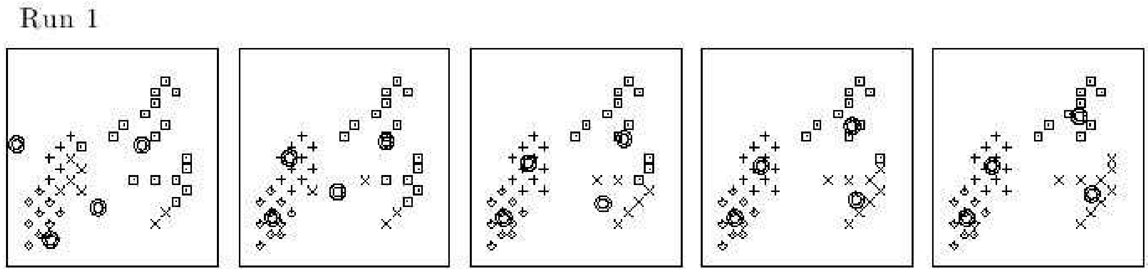

Figure 2.4: First run of K-Means from [8].

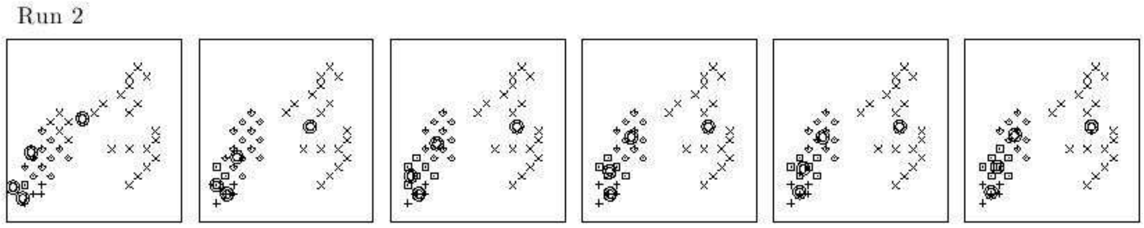

Figure 2.5: Second run of K-Means from [8].

$$d(x,y) = \frac{1}{2}\sum_i (x_i - yi)^2$$

It is possible to define different metrics as a distance functions for example the Euclidean distance $d_e(x,y) = \sqrt{\sum_i (x_i - yi)^2}$. If the attributes are not numeric a definition of a metric to calculate the distance between all possible values of an attribute is necessary. This matching between attribute value and numeric representation makes the recalculation process a lot more complex. It is also necessary to prove if the scale of numeric values represents the real scale of the real attribute values.

The performance of the algorithm depends on the location of the initial randomly selected centres because the number of iterations that are needed until the cluster constellation reaches a stable state is dependent on the position of the initial centres. The positions of the initial centres in relation to all other centres also have an impact on the final result. Both dependencies imply that it is probable that the clustering technique will produce different final clusters among several runs of the algorithm, even if the set of instances each time is the same.

An example of two separate runs of the K-Means algorithm with the same data sets, where the computed clusters and the number of iterations have changed because of the changed initial centres is shown in figures 2.4 and 2.5. The circles show the centres, and the other symbols show the current state of the clusters. These two figures also show that the final clusters depend on the initially assigned centres because the final centres have different locations and the point are partitioned into different clusters.

2.1.2 Agglomerative clustering

Agglomerative clustering belongs to the group of classic clustering methods [7]. The algorithm works with a hierarchical data representation called a dendrogram.(See figure: 2.7) The agglomerative cluster algorithm [5] builds a dendrogram by progressively merging clusters. the selection of cluster that are merged is based on the distances between the clusters, so agglomerative clustering is also a distance-based clustering method.

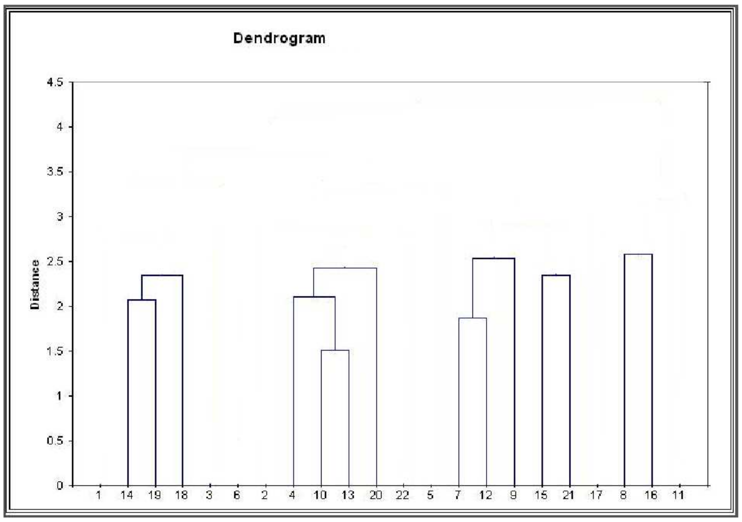

Figure 2.6: An early stage dendrogram.

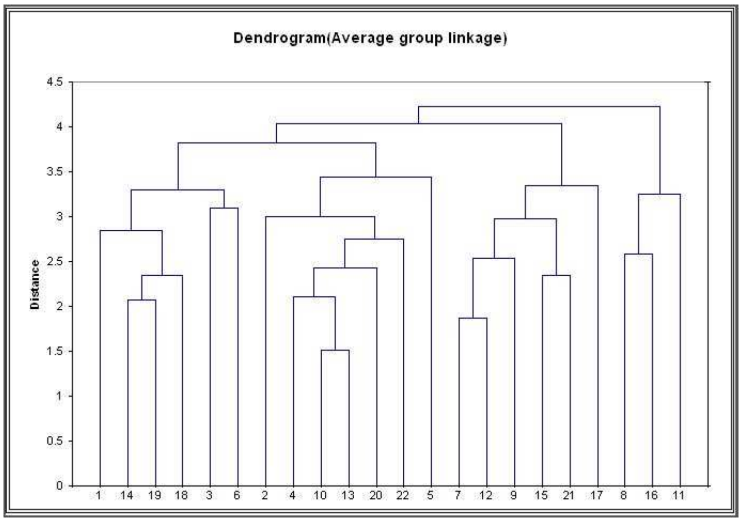

Figure 2.7: The final dendrogram.

The dendrogram is built from the bottom-up. The algorithm starts with a dendrogram that only consists of leaf nodes. Every training instance is represented by a singleton cluster, which represents a leaf node in dendrogram.

To build the dendrogram from the bottom-up the distance between all clusters needs to be calculated based on the defined distance function. The two clusters which have the smallest distance between each other are merged, and distances from new cluster to all the other clusters are calculated. The dendrogram shown in figure 2.6 has started building clusters. This is repeated until the result is one big root cluster that contains all instances, as shown in figure 2.7.

The implementation of the distance function has a strong impact on the final cluster. There are big differences between cluster resulting from a merging of two clusters based on the distances between the nearest instances and clusters based on the distances of the middle or farthest instances.

Some of the standard distance functions between two clusters A and B are the following:

- single linkage clustering: calculate the minimum distance between instances from each cluster see figure 2.8
$$d(A,B) = min_{a \in A, b \in B}\{d(a,b)\}$$
- complete linkage clustering: calculate the maximum distance between instances from each cluster see figure 2.8
$$d(A,B) = max_{a \in A, b \in B}\{d(a,b)\}$$
- average linkage clustering: calculate the average distance between all pair of instances, where one instance is in cluster A and the other one is in cluster B see figure 2.8
$$d(A,B) = \frac{1}{|A||B|} \sum_{a \in A, b \in B} d(a,b)$$
- average group linkage: calculate the average distance between all pair of instances in the union of cluster A and B
$$d(A,B) = \frac{1}{|C|} \sum_{a,b \in C, C = A \cup B} d(a,b)$$
- centroid method: calculate the distance between the cluster means
$$d(A,B) = d(\bar{a}, \bar{b})$$
- Ward's method: measure the increase of the variance after unifying the two clusters. This can also be explained as the difference between the sum of the squared errors of the two single cluster and the union of the cluster A and B
$$d(A,B) = \frac{d(\bar{a}, \bar{b})}{1/|A| + 1/|B|}$$
$$Var(C_i) = \sum_{x \in C_i} (x - m_i)^2$$
$$d(A,B) = Var(A \cap B) - Var(A) - Var(B)$$
where m_i represents the mean of the cluster.

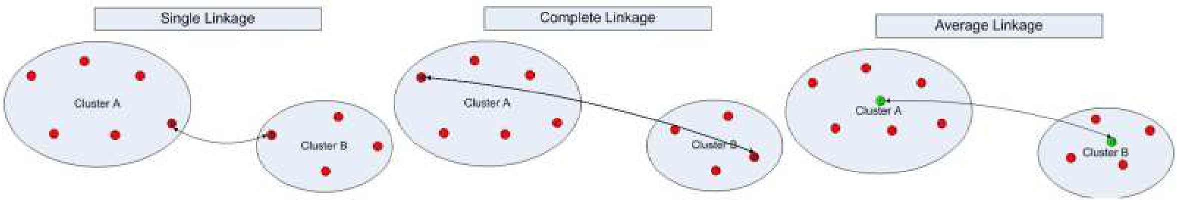

Figure 2.8: Visualization of the three linkage methods.

Clusters that are merged based on the distance of their nearest instances can result in long chain clusters, where only the nearest instances are very close. The occurrence of chain clusters can be avoided by using one the different distance functions explained before. For example the complete linkage method would not lead to clusters that build long chains.

The function that is applied to calculate the distance between two instances has a strong impact on the performance of the clustering algorithm. The algorithm may need to calculate the distance between all possible pair of instances $(0(m * n))$, to calculate the distances between two clusters with m and n instances.

As agglomerative clustering is a distance-based clustering algorithm like K-means (section 2.1.1). The agglomerative clustering algorithm has the same problem in the case that the data instances have attributes with non-numeric values. It is necessary to define a numeric representation for each possible value, which represent the distances between the non-numerical values.

2.1.3 COBWEB clustering

The COBWEB algorithm [4] produces a hierarchy of clusters and works incrementally, which means that the cluster-hierarchy is changed every time we add new instances to a cluster. COBWEB is a top-down approach and starts with one initial root cluster. This root cluster contains all training data instances, which are incrementally placed in the tree. The algorithm performs an incremental search for the best clusters. This means that COBWEB's hierarchical approach attempts to find the best classification-tree, that builds the best conceptual groups of instances (clusters). An example with animal instances and a possible cluster-tree based on this data input:

Name	BodyCover	HeartChamber	Bodytemp	Fertilization
mammal	hair	four	regulated	internal
bird	feathers	four	regulated	internal
reptile	cornified-skin	imperfect-four	unregulated	internal
amphibian	moist-skin	three	unregulated	external
fish	scales	two	unregulated	external

Figure 2.9: Animal descriptions

COBWEB uses a hill-climbing algorithm to find the best classification tree that represents the clusters. To compare possible classification tree candidates, the algorithm uses a heuris-

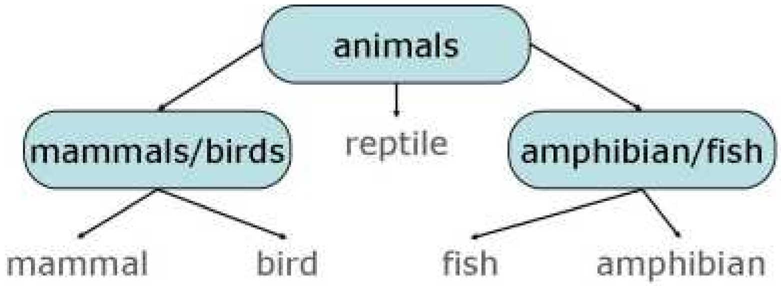

Figure 2.10: COBWEB

tic to measure the improvement of the classification tree after the execution of an optional operation and to decide which operation is the best one and should be performed.

This heuristic is called "category utility", and is defined as a combination of conditional probabilities. The first idea is to reward intra class similarity for an attribute-value pair. This is derived from the fact that an attribute-value pair is more predictive for a class if it occurs more often within this class. the heuristic therefore prefers high values for $P(A_i = V_{ij} \mid C_k)$.

The other idea is to reward inter-class dissimilarity. Attribute-value pairs that also occurs in other classes outside the class are not as likely to describe the class as attribute-value pairs that occur only in the class. The attribute-value pair that occurs outside the cluster should have a lower impact on the prediction. The heuristic therefore prefers high values of $P(C_k \mid A_i = V_{i,j})$.

To calculate the "category utility" for a partition stage in the classification tree it is necessary to sum over all mutually-exclusive classifications $\{C_1, C_2, ..., C_n\}$ and attribute-value pairs:

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

This formula describes the product of all inter and intra class similarities weighted by the attribute-value frequency summed across all attributes(i), values(j) and classes(k). This heuristic increases the class-conditioned predictability and predictiveness of values that occur more often. An increase of this heuristic should indicate an increase in the inference potential of a class partition.

Using the Bayes rule, the formula for above can also be transformed to:

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

Where $\sum_i \sum_j P(A_i = V_{i,j} | C_k)^2$ expresses the *expected* number of attribute values that can be predicted for a randomly chooses instances from the cluster.

Here an example for a probabilistic representation of the class (fish, amphibian, mammal):

Attributes	Values and their probabilities
BodyCover	scales=0.33,moist-skin=0.33,hair=0.33
HeartChamber	two=0.33,three=0.33,four=0.33
Bodytemp	unregulated=0.67,regulated=0.33
Fertilization	external=0.67,internal=0.33

Figure 2.11: Animal descriptions

The calculated clusters are placed in a tree structure in which the root node represents the complete set of instances. As the algorithm works with probabilities each node is labelled with a classification description based on the probability of the attribute-value pairs to summarize all instances in the node and to enable the later classification of new instances.

An example classification tree based on the two instances fish and amphibian:

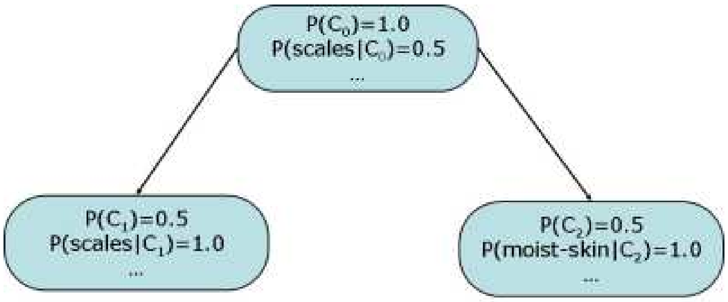

Figure 2.12: A classification tree for fish and amphibian from [4].

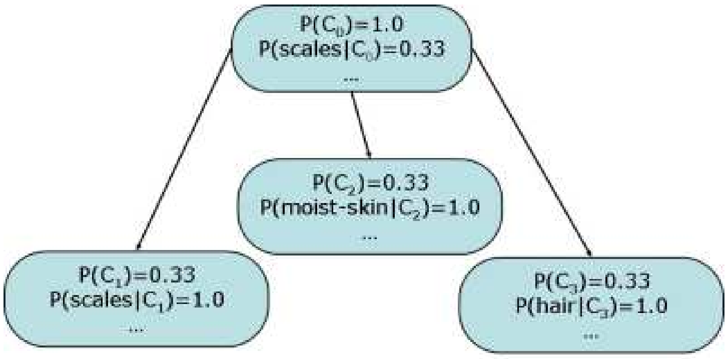

Figure 2.13: A classification tree after adding the mammal from [4].

COBWEB is an incremental clustering algorithm. Therefore several operations are required for adding new instances to the cluster-tree and mutating the cluster-tree. COBWEB makes uses of four operations although two of those operations ”undo” each other. The possibility of manipulating the cluster-tree with complementary operations enables the hill-climbing algorithm to move in two directions through the search space. This improves its ability to overcome local minima during the hill-climbing search. Without these complementary operations, the search algorithm would get trapped in local maxima.

The four operations it considers at each stage are:

- classify a new instance to an existing class
- make a new instance class
- merge two classes
- split two classes

The easiest operation is to place an instance in an existing class. In order to find the most suitable class for the new instance, the instance is tentatively placed in all available classes. Then for each possible constellation, its ”category utility” is calculated. The algorithm selects the class where the added instance caused the highest increase in the ”category utility”.

If the partition does not hold a suitable class for the new instance, the algorithm has to add a new class. The algorithm creates a new class if placing the instance in the best existing class does not increase the ”category utility” as successfully as adding the instance to a new singleton class.

These two operations provide an effective way of building a cluster-tree, but nonetheless, the result would be highly sensitive to the ordering of the input instances. To overcome this problem COBWEB has two further operations. These operations allow the merging and splitting of nodes. Merging and splitting of nodes are complementary actions so the previous action can be undone. This enables the hill climbing algorithm to backtrack in its search for the maximal ”category utility” and ensures that the search is not trapped at local maxima.

The merging operation allows two nodes in a partition to be merged into a single unified node. So the number of clusters in this partition is decreased. One limitation is that the algorithm only consider the best and the second best class-nodes to be merged. The original nodes remain in the tree. The nodes are pushed down within the tree and become children of the merged node. Splitting is exactly the inverse operation of merging. A node is deleted from the cluster-tree and its children are promoted to the high partition level.

COBWEB works with the following steps. First find the node in the class which is the best host for the new instances. Then consider creating a new class, and if this is appropriate, execute the operation. Subsequent do the same with the merging and splitting operations. Although the algorithm makes use of bidirectional search, the resulting clusters depend on the ordering of the instances, whose are incrementally applied to the cluster-tree.

2.2 The clustering algorithm MAXCCLUS

MAXCCLUS is not a typical clustering algorithm because, unlike standard clustering algorithms it requires the data to already be classified. Also the algorithm does not clearly fit into the class of unsupervised learning. MAXCCLUS was designed to analyze bioinformatics gene expression data and their descriptions. The normal goal of a clustering algorithm is to group a fraction of all instances based on their attribute values. Afterwards, each group(cluster) will represent a set of instances with similar properties and can used for classifying new instances. Based on the attribute value pairs that characterize or describe a cluster, the implementation can perform prediction on new instances. If a new instance shares the required attribute values, the algorithm can assign the instance to this cluster. The predicted the class of the instance will based on the class of the cluster.

MAXCCLUS works with data from micro-array experiments, and aims to make sense of this data. The possible outcomes of the experiments are the ”UP” and the ”DOWN”. These two key-words represent the categories or classifications for each instance. For example the following part of one data set:

```
b0002 DN
b0003 DN
b0004 DN
b0006 UP
b0007 UP
b0010 UP
```

The second data input is a set of words. MAXCCLUS also uses data extracted from the SwissProt database. The SwissProt database is a collection of text descriptions of the proteins encoded by genes. MAXCCLUS uses a set of descriptor words for gene extracted from the database.

As the SwissProt data files are natural language files, it is necessary to do some text processing before the text files can be used as descriptor lists. Otherwise it would not be possible to derive rules from these data inputs. For example it is necessary to remove all singleton words and also remove all non significant SwissProt related comments. Another issue with a natural language input is the natural language has a semantics, but MAXCCLUS does not work with the semantics of words. That means every single word represents a attribute value. A An example descriptor data input would look like this:

MAXCCLUS task is to find characterizations of subsets of instances which all belong to one category after a fixed experiment.

The genes occurring in the experiments result are the only instances MAXCCLUS is going to process and it is no necessary to split the data set into training and test data. MAXCCLUS does not have a training or test phase.

The aim of MAXCCLUS is to find rules to describe a subsets of instances. These rules a based on the descriptions of the instances. The descriptions of the instances represent the second data input required by MAXCCLUS. The rules computed by MAXCCLUS should provide some information about the subset of instances related to the performed experiment. From this rules it could be possible to derive an explanation why genes behaved equal under the conditions applied during the experiment.

```
b0006   0-2.4 abundance analysis chromatography
        coli enrichment escherichia genome h.influenzae
        hi0984 hydroxyapatite k12 low min nucleotide
        operon protein proteins region sequence sequencing
        strong systematic termination thrc threonine
```

```
        transcription yaaa
b0007   0-2.4 51.7 analysis belongs coli escherichia
        family genome h.influenzae hi0183 hypothetical inner
        integral intergenic kda membrane min
        orf8 potential protein region saf sequencing
        sodium:alanine strong symport symporter systematic
        thrc-talb transmembrane transport
b0008   0-2.4 3-phosphate 3d-structure 4-phosphate
        6-phosphate 7-phosphate aldolase aldolases
        alpha/beta analysis b balance barrel
        belongs characterization circular
```

The MAXCCLUS algorithm itself has four main stages:

- find the maximal clusters of instances based on the two dimensions, number of words and number of instances
- identify the minimum size for a cluster to be statistically significant
- select a covering of clusters with the required significance and accuracy
- construct the rules for the clusters

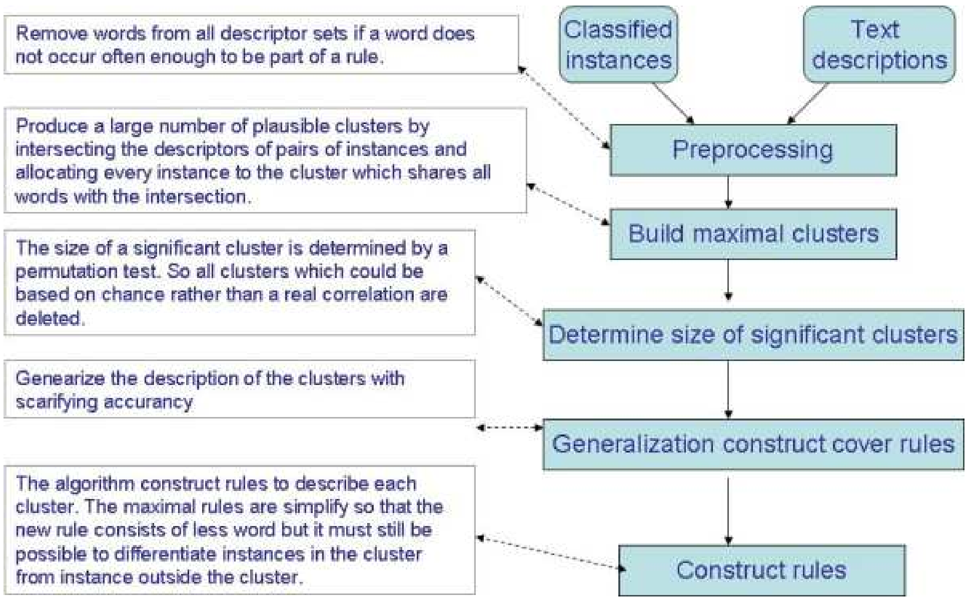

Figure 2.14: A diagram of the MAXCCLUS algorithm.

The initial step is to find the maximal clusters for a given data input. To construct the required maximal cluster, the algorithm starts with the intersection of the descriptions of two instances. For example if the algorithm took the instances `b0006` and `b0007` from the data input example 2.2, the algorithm would start constructing a cluster:

```
instances:  b0006,b0007
words:  0-2.4 analysis coli escherichiagenome h.influenzae
hi0984 min protein region sequencing
strong systematic
```

The cluster with two instances is a initial cluster. This pair of instances represents the core of a new cluster. To extend this cluster, MAXCCLUS compares the word list of the cluster with the word list of each other instance. An instance is allocated to a cluster if only if the word list of the instance contains all the descriptors in the list of the cluster. The result of this process is the largest cluster possible with this most detailed description. This cluster can be called a minimal conjunctive generalization of the instances.

To process all possible combinations, MAXCCLUS compares all instances of the data set pairwise, and calculates the intersection between the descriptions of both instances. Because it is possible to construct the same cluster from different core pairs, so duplicates have to be deleted. The resulting clusters are maximal in two senses. The first sense means that it is impossible add any other instance in the data to a cluster without deleting words from the descriptor list. The second sense means that it is not possible to add any additional word to cluster description without excluding an instance from the cluster.

At this stage the algorithm holds a set of clusters that a maximal in two dimensions. After constructing the maximal clusters the algorithm needs to select the clusters which fulfil the required pre-defined level of accuracy and to calculate the minimal size of a significant cluster.

As MAXCCLUS is designed to find rules to describe the behaviour of group of genes that behave the same under set conditions, it is only possible to uses clusters where a defined percentage of the instances in the cluster have the same class otherwise the resulting rule would not provide any valuable information about the gene's behaviour. An accurate cluster is one in which the fraction of instances in the majority category is as high as the user specified accuracy.

The next problem is to select only the significant clusters. In statistics, a result is significant if it is unlikely to have occurred by chance. Clusters are significant if correlation between the instances is not based on random occurrences. So it is necessary to determine the maximal size of a cluster which could have occurred rather by chance than by a correlation between the instances.

The size of a significant cluster is determined using a permutation test. To perform the test, all instances are assigned a new category, which is randomly assigned in proportion to the distribution the original categories. If the data input has 70% 'UP' genes and 30% 'DOWN' genes the permutation test will generate random categories with the same proportion. Because of the different proportion of each category it is also necessary to determine a significant cluster size for each category: if the total number of 'DOWN' instances is smaller, then the maximum size of a random cluster will be smaller as well. After assigning new categories to each instance the algorithm identifies clusters that would be considered accurate, based on the instances with the new categories. This is repeated many times with changed permutations of categories to derive a representative minimal size for a significant cluster.

After selecting only the accurate and significant ("the good") clusters. The algorithm holds a set of significant and accurate clusters. To construct rules to characterize the instances in each cluster. Each cluster has an associated class (UP or DOWN) and a set of descriptors. It is easy to derive a rule. The algorithm could construct rules by simply listing all the descriptors associated with the cluster. However, these rules are not always useful because some of the words are not helpful for example the word "protein" occurs in 76 percent of all instance-descriptions; this word would not help to differentiate between clusters because it is probably present in a high number of rules regardless of the category of the cluster. To define rules which provide information about the instances, it is required that the words help to distinguish between instances in the cluster and instances out of the cluster. To establish rules that describe the cluster well and mark the unhelpful words, MAXCCLUS classifies all words in a cluster into three different categories:

- necessary: not including this word would allow instances from outside the cluster
- sufficient: this word is in a set of words, at least one of which is required to distinguish
- supplementary: not useful for distinguishing

The algorithm constructs two types of rules. The first type of rules is the minimal conjunctive rules. These minimal conjunctive rules are the minimal subsets of the cluster-description which distinguish the instances in the cluster from all other instances. The words in the description are classified in the categories above.

The second kind of rules is more general. These rules are only minimal subsets of the cluster-description that distinguish the instance in the cluster from all outside instances with the other category. All instance which belongs to the same category are ignored.

Example output MAXCCLUS:

```
cluster 3:   Category: Up  Accuracy: 90.0%
             Coverage: 281  Unique Instances:  5
 reqd words:  operon phosphorylation protein region regulation
              system that phosphoryl phosphotransferase primary
              proteins pts site structure sugar this three
              transfers transmembrane transport two which tetramer
              transfer properties reaction reactions repressed
              resolution revealed sites structural substrate substrates
              subunits succinyl-coa synthesis synthetase synthetases
              thus tricarboxylic via when x-ray well
  necessary:  operon regulation
 sufficient:  phosphorylation phosphoryl phosphotransferase primary pts
supplementary:  protein region system that proteins site structure
               sugar this three transfers transmembrane transport
               two which tetramer transfer properties reaction reactions
               repressed resolution revealed sites structural substrate
               substrates subunits succinyl-coa synthesis synthetase
               synthetases thus tricarboxylic via when x-ray well

cluster 4:   Category: Dn  Accuracy: 93.7%
             Coverage: 16  Unique Instances:  5
 reqd words:  phosphorylation protein that transduction
              sequence site structure this three two which
              tetramer transfer revealed substrate
              subunits succinyl-coa synthesis synthetase
              synthetases thus
```

```
               tricarboxylic via when x-ray well flagellar
  necessary:  flagellar
 sufficient:
supplementary:  phosphorylation protein that transduction
                sequence site structure this three two which
                tetramer transfer revealed substrate subunits
                succinyl-coa synthesis synthetase synthetases
                thus tricarboxylic via when x-ray well
```

The example output shows two rules that were produce by the WEKA version of MAX-CCLUS. The first rules describes a cluster with the category 'UP' and an accuracy of 90% it covers 281 instances. The words tagged `reqd words` are all descriptors found by intersection of two instances and incrementally allocating other matching instances.

These words can be categorized into the three categories `necessary, sufficient, supplementary` described in the list above . The first rule in the example above means the words `operon` and `regulation`, as well one word form the list of sufficient words `phosphorylation`, `phosphoryl`, `phosphotransferase`, `primary`, `pts` are required to describe an instances in the cluster. The second rule for example only requires one word: `flagellar` to distinguish between instances in the cluster and those outside the it.

2.3 The Waikato Environment for Knowledge Analysis (WEKA)

WEKA [9] is a machine learning toolkit that provides a wide selection of machine learning algorithms. It is designed, developed and published by the University of Waikato under the GNU General Public License.

The main purpose of WEKA is to provide a suite of machine learning techniques and make them generally available to the public. WEKA is extensible and provides a comfortable platform for the distribution and the performance analysis of new machine learning algorithms. The wide collection of present implementations makes it also easy to compare new techniques to existing ones.

WEKA contains tools for

- pre-processing of datasets
- conversion of data
- classification (including decision tree, bayes, etc.)
- regression (including linear regression, etc.)
- visualization
- clustering (including Cobweb, EM, FarthestFirst, MakeDensityBasedClusterer)

WEKA is written in Java and runs on nearly every platform. The data that should be applied to the actual learning schemes needs to be stored in a special data format. The ARFF (Attribute-Relation File Format) was especially developed and designed for WEKA. ARFF is based on a ASCII text file, that contains a list of instances sharing a collection of attributes. To simplify and reduce the data pre-processing effort and work, WEKA offers a variety of tools to manipulate, extend or filter existing dataset.

All tools and techniques provided by the WEKA workbench can be used through a graphical user interface. The usability of the GUI has a high standard. The GUI is intuitive to use although it is very packed with functionality because of the high number of different tools. To illustrate all experimental results WEKA has a variety of visualization tools.

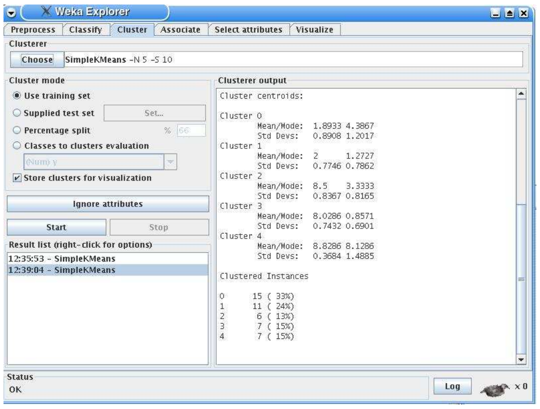

Figure 2.15: The WEKA user interface.

WEKA has three main framework for machine learning algorithms:

1. WEKA's classification framework is designed to train classifiers. The classifiers are able to predict a single class attribute, which is the target for prediction. Some of the classifiers implemented in WEKA can only process nominal attributes. Some algorithms are also able to process numeric attributes. After the training is finished, the final classifier is evaluated. A new test data set is applied to the classifier and WEKA measures how often it predicts the right class.

2. WEKA's clustering implementations cluster instances from the training data set based on a defined distance function. A consequence of this is that clustering algorithms can only process instances with attributes to which a distance function can be applied. The classification of new instances is done by finding the best fitting cluster for each instance and assigning the class of the cluster.

3. WEKA's associating framework finds associations (correlation relationships) among the instances. Association rules are attribute value conditions which occur frequently in instances of the same class. The Association rules have a "if-then" structure to provide information about the dependencies between an attribute value and its category. Based on these rules the associating framework predicts the class of an instance by looking up the rules.

The interface between each framework and the WEKA core is described by an abstract class. The implementation of a new algorithm requires one these abstract classes to be extended.

Chapter 3

Implementation

3.1 The extensibility of WEKA

The WEKA framework is implemented in Java. Java is an object-oriented programming language. Object-oriented programming languages are designed to support the implementation of easily extendible software with high level of reusability. Object-oriented frameworks are designed to provide a suitable basis for new extension. These extensions are supposed to be easy to establish with less effort compared to the creation of a new standalone tool which provides the same functionality but does not make use of the framework. The framework should provide a reusable abstract structure and general functionality to incorporate extensions which can be reused.

Extensibility in Java is realized by class inheritance and implementation of interfaces. Also possible is the definition of a mixture which is called an abstract class. WEKA uses abstract classes to define a structure and a interface for new tools like classifiers, filters or clustering algorithms. Extending a abstract class enables direct interaction between the new implementation and established WEKA features like file processing and the graphical user interface.

3.2 Framework

This project is focused on adding new a clustering algorithm to the WEKA toolkit. WEKA provides various frameworks to add new machine learning algorithms. The interface of each algorithm framework is described by an abstract class [9]. The methods defined in the abstract class are used by WEKA to apply the algorithms to a dataset.

WEKA provides three frameworks to implement new machine learning algorithms:

1. The Classifier framework is used for all supervised classification algorithms
2. The Clustering framework is used for unsupervised clustering algorithms
3. The Association framework is used for algorithms building association rules

A new machine learning algorithm needs to extend one of the provided abstract class definitions. It is possible to create as many additional classes as required. These classes should be located within an algorithm related package to preserve the structured class hierarchy.

The first task was to decide on a suitable framework for the MAXCCLUS algorithm. As MAXCCLUS is a clustering algorithm, the first selected candidate was the clustering framework. Unfortunately, the WEKA clustering interface is designed only for clustering algorithms that perform classification tasks; MAXCCLUS does not perform classification. MAXCCLUS's main aim is to make sense of data by finding rules to describe the clusters. An additional obstacle was a constraint made by the designer of the clustering framework which assumes that all clustering algorithms are going to build their clusters based on a distance measure. MAXCCLUS is not a distance based classification clustering algorithm. So we neither have data to compute a distance between instances nor do we want to perform a classification on new data instances, so the cluster package of WEKA does not provide a suitable interface for MAXCCLUS.

One of the two remaining frameworks is the classification framework. The majority of implemented algorithms are classification algorithms. The classification framework is the most detailed and most narrowly designed framework in WEKA and assumes that the result of the training phase is a classifier. MAXCCLUS does not classify new data so the classification framework is not suitable either.

The last framework is the Association framework. The Association framework is a less strictly defined framework, and it has almost no constraints how the implemented algorithms should deal with the data. The Association framework provides only three methods: one method to execute the algorithm, a copy method, and a method to enable the call-by-name instantiation. The result type is a simple formatted String. As the main goal of MAXCCLUS is to make sense of data by finding description rules for the clusters built before, there is no reason why MAXCCLUS can not be implemented with Association framework[1]. It fits well into Association framework even though it does not produce association rules.

3.3 Data Input

The second task was to convert the data input for MAXCCLUS so that it can be read by the WEKA framework. The stand-alone version of MAXCCLUS reads its data from two different files. The details about the MAXCCLUS data input can be found 3.3.2.

WEKA provides its own file format and data processing implementation. WEKA's data format is called the Attribute-Relation File Format (ARFF). An ARFF file is an ASCII text file that describes a set of instances sharing a set of attributes. The details of this format are given in 3.3.1.

As WEKA works with its own file format it is necessary to convert MAXCCLUS's two data files into the ARFF file format. MAXCCLUS also performs some data pre-processing while reading the data files. This task must be extracted from the original implementation and added to the data converter tool.

I implemented a data-converter application to solve this problem. The data converter is not embedded in the WEKA framework. The data converter reads in two files, the gene expression file and a SwissProt file, and writes a valid ARFF file for WEKA.

The data converter is not restricted to gene data and can handle a more general scenario. The first data file contains all instances represent by a key string and the values of the class attributes where each instances must have its own line. The second file contains with all the instance keys followed by a list of words.

3.3.1 ARFF Data Format

An ARFF file is an ASCII text file that contains three compulsory sections[6]:

1. the relation declaration
2. the attribute declaration
3. the data declarations

The only purpose of the relation declaration is to name the dataset; it has to be the first line in the file.

```
@relation <relation-name>
```

The attribute declaration lists all possible attributes an instance can contain. Each attribute is defined in a new line starting with the `@attribute` keyword followed by the name of the attribute and its data type. The ordering of the attributes is related to the column in the data declaration. Currently WEKA supports four data types:

- `numeric`
- `<nominal-specification>`
- `string`
- `date [<date-format>]`

The data declaration section is the description of the instances. This section starts with a line containing the keyword `@DATA`. The instances are defined in the following lines. Each instance needs to be defined in a single line. All attributes need to be filled in missing values must be denoted by a question mark. The attribute values for each instance are delimited by commas and must appear in the order that they were declared in the attribute section: the data corresponding to the *i*th `@attribute` declaration is always the *i*th field of the instance. The first attribute always represents the class value of the instance. A typical instance would be:

```
sunny,85,85,FALSE,no
```

An example ARFF file would look like this :

```
@relation weather

@attribute outlook
{sunny, overcast, rainy}
@attribute temperature real
@attribute humidity real
@attribute windy {TRUE, FALSE}
```

```
@attribute play {yes, no}

@data
sunny,85,85,FALSE,no
sunny,?,90,TRUE,no
overcast,83,86,?,yes
rainy,70,96,FALSE,yes
```

The above example show that the ARFF file format has the following characteristics:

- ARFF attributes can be nominal, numeric or strings
- a list of attributes needs to be predefined in the data header
- every instance needs to have a value for the predefined attributes
- missing attributes must be represented by a placeholder
- missing attributes can represented by a question mark

3.3.2 MAXCCLUS Data Format

The first file contains the results of the gene experiments. Each line has two items, the name or identifier of the gene and its classification ('UP' or 'DOWN'). The gene expression file only contains genes that changed the behaviour due to the experiment conditions. The second file is a file from the SwissProt database. All genes are listed with their descriptors; each line starts with the name of the gene. The gene name is followed by a list of words that are extracted from scientific essays about the related gene. The words from this list are called descriptors.

A file that contains the classifications:

```
..........
b0014 DN
b0022 UP
b0031 DN
b0035 UP
b0072 DN
.........
```

A file that contains a list of descriptors for each gene:

```
b0001 affecting attenuator biosynthesis characterization codons control...

b0002 0-2.4 12 2 3 4-phospho-l- activities adp aki-hdi allosteric...
```

The two data inputs are the classifications and lists of descriptors. Whereas the ordering of the descriptors in the list does not matter. In contrast the example 3.3.1 shows a data input where the definition of the attributes contains some information. The MAXCCLUS data input has no attribute definition. Another difference is the number of attributes the attribute definition contains a fixed number of attributes like: `outlook temperature humidity windy`. It is easy to see that each instance has the same number of attribute values and that the ordering of the attribute values is important to represent the required information. MAXCCLUS instances have word list as attribute values but those lists can have various lengths and the ordering of the words would not chance the information represent by this instance.

3.3.3 Data Converter

Pre-processing the MAXCCLUS data input involved the implementation of a conversion tool so that WEKA could read the MAXCCLUS data input. This tool is called the data converter and performs the tasks explained below:

First, the data converter reads in the gene names and their classes from the gene expression file. The genes need to be ordered in the same way as the SwissProt database orders the genes. Second the data converter scans the SwissProt file incrementally and adds all descriptors that are related to a gene in the previous file. this is the reason for the required ordering of the genes in the gene expression file. The complete data is stored in a `HashMap`.

The next step involves the data pre-processing to delete all non-significant words, and all instances with no descriptors. The pre-processing creates a dictionary to count the occurrences of every descriptor. The threshold which indicates a significant word can be set via a parameter (the default value is 5). All descriptors which occur fewer times than the threshold are deleted because such words will not occur in a rule that covers a significant cluster. At is point, it is possible that there are instances that have no descriptors; these instances need to be deleted as well.

MAXCCLUS needs the names of the genes to print out the rules and the gene names are also used as identifiers in the MAXCCLUS internal data structure. If they are added as one of the descriptors before the pre-processing, the pre-processing will remove them because they are unique and so they would be classified as non-significant descriptors. Therefore it is necessary to add the gene names to the descriptor list after the pre-processing. To standardize the access of gene names in the WEKA data structure, all gene names are stored at the first position after the instances class. This means it is either possible to access the first attribute via an index or either based on the generated attribute name.

To write the file, some other calculations are required. First we had to decide how we want to store the MAXCCLUS data in an ARFF file. First the ARFF file required a attribute definition section so the data converter has to define a set of attribute names.

One possibility to define the attributes would have been to store each occurring word as an attribute. Each instances would consist of a list of boolean values. If the word is in the descriptor list the value is `true` otherwise it is `false`. This method would have been more coherent with the WEKA data structure but it would have decrease the efficiency and increase the memory required to store the data structure. That was the reason we decided to use another method to store the MAXCCLUS data in the WEKA data structure.

The second method is that data converter generates attributes names. As the words are stored as string attribute values. That means the number of attribute value of the instances is not constant because the number of descriptors can vary between the instances.

To represent the instance with the most descriptors the data converter to generate as many attribute names as descriptors are in the list. As the length of a word list can be less than the length of the longest word list and WEKA requires instances with a fixed number of attributes. The data converter has to deal with the missing attribute values. WEKA allows us to use question marks to represent missing attribute values so the data converter can extend the number of attribute value by appending question marks. The number of question marks appended depends on the difference (`MaxNumOfAttr-instance.numOfAttr`).

The implementation of the data converter was completed and tested with original MAXCCLUS data files.

A generate ARFF file would look like this:

```
@ATTRIBUTE Up_Down string
@ATTRIBUTE word1 string
@ATTRIBUTE word2 string
@ATTRIBUTE word3 string
......

@DATA
'DN','b0331','belongs','family',?,?,?,?
'UP','b1001','coli','escherichia','hypothetical','intergenic','kda','protein'
'UP','b1426','characterization','cloning','coli','escherichia',?,?
'UP','b2787','analysis','belongs','characterization',?,?,?
'DN','b0135','analysis','belongs','cloning','coli','escherichia',?
'UP','b0269','belongs','e.coli','family','hypothetical',?,?
........
```

Figure 3.1: The diagram of the data conveter.

3.4 Data Structure Mapping

The MAXCCLUS algorithm works with its own data structure. This data structure consist of two objects `DescriptorTable` and `InstanceTable`, representing the data in both input file. The `DescriptorTable` stores the gene name as key and the descriptors. The `instance table` stores all gene names a the category of the instance ('UP' or 'DOWN').

The MAXCCLUS data structure was specially designed for the clustering algorithm and it is a complex and efficient data structure. MAXCCLUS works with a numerical representations of the words as well as string representations because string operations are not efficient. The descriptors in the MAXCCLUS data structure need to have a numerical identifier. Therefore MAXCCLUS'S internal data structure stores all words that are descriptors in a dictionary, which matches each word to a index number. The MAXCCLUS clustering algorithm works on the index numbers of the occurring words for efficiency reasons.

Also MAXCCLUS data structure work with instances with different numbers of attributes - MAXCCLUS has no representation for missing values. The attributes are just list of items. for efficiency MAXCCLUS requires list of words of identifiers to be ordered.

WEKA has an internal data structure representation implemented in the class `Instances`. This data structure is created while WEKA reads in the chosen ARFF file. This WEKA data structure stores every instance including their attributes and category. All in WEKA implemented machine learning algorithms are using this data structure. WEKA has only one data structure and all instances have a fixed number of attributes. Missing values are represented by question marks. The data is exactly stored in the same order as it is defined in the ARFF file. As WEKA used distance function in many algorithms like K-Means clustering algorithm 2.1.1 or the agglomerative clustering algorithm 2.1.2.

So the WEKA data structure can be summarized as:

- can not access attribute value directly; all values have a numeric identifier
- attributes value are only accessible via attribute names
- no fixed set of attribute names

The changes required to change MAXCCLUS so that it works on the WEKA based data structure are much more complicated than write a data mapping tool. This data mapping tool will convert the WEKA data structure into the MAXCCLUS data structure. So the data mapping tools kind inverses the results or work of the data converter tool described before.

The WEKA extension needs to map from the WEKA to the MAXCCLUS data structure. That means undoing the changes made to in constructing a valid ARFF file. This involves:

- extracting the gene name and the classification and put into the MAXCCLUS one data structure
- identifying non-empty attributes and put them with the gene name as key into the other MAXCCLUS data structure
- ensuring that the order of the descriptor is preserved. Since MAXCCLUS requires the descriptor lists to be sorted.

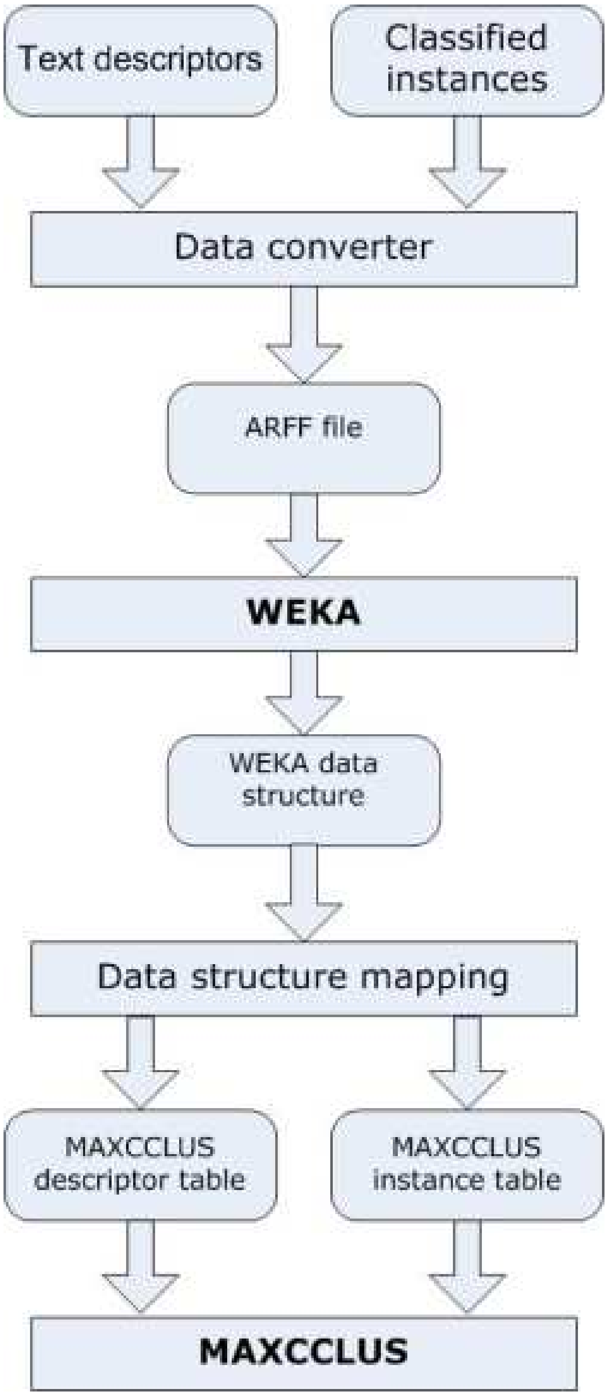

Figure 3.2: The diagram of the complete process.

3.5 The Graphical User Interface

WEKA provides a graphical user interface implemented with Java Swing (for a screenshot see 2.15). All machine learning algorithms implemented in WEKA are executable via the graphical user interface. Adding a new implementation to the WEKA graphical user interface is simple. All classes that implement one of the predefined frameworks are listed in a single file called `GenericEditor.props`. The default file is included in the `WEKA.jar`. This file can be overridden by a similar file which also lists the added algorithm class located in the home directory or in the directory where WEKA is executed.

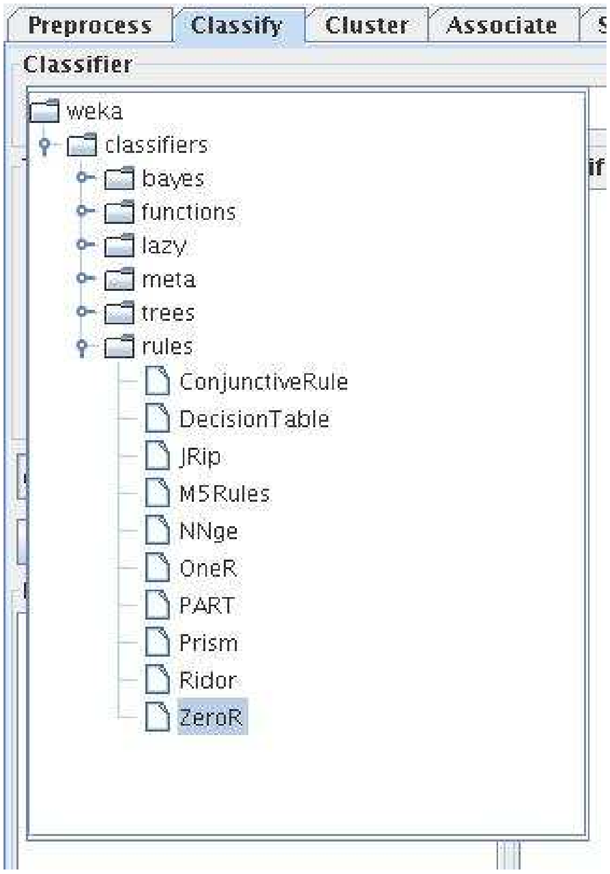

Figure 3.3: Generated list based on GenericEditor.props.

To add MAXCCLUS it is only necessary to override the Association List in the `GenericEditor.props` file and extend the option handling. The user can set the option for the algorithm using a `TextEditField`. The best way to deal with option that are edited via the GUI is for the main class of the algorithm to implement the `OptionHandler` interface. The three required methods are[2]:

- `getOptions`
- `listOptions`
- `setOptions`

Every option needs to provide some specific additional methods, so that they are recognized by WEKA Explorer. These methods should enable the WEKA Explorer to provide help to set and get options. To provide a brief description of the algorithms it is necessary to implement the `public String globalInfo()` method.

Currently the MAXCCLUS algorithm can not be executed via the graphical user interface.

Chapter 4

Evaluation of the WEKA framework

To integrate the MAXCCLUS algorithm into the WEKA machine learning toolkit, it was necessary to analyze the WEKA framework and its design. The WEKA is a complex framework and was based from a certain purpose so it has strength and weaknesses for integrating new algorithms:

Strengths:

WEKA is a machine learning framework that provides a wide selection of implemented machine learning algorithms like decision tree, k-means (for details see 2.3). The toolkit provides a platform for the distribution of new machine learning algorithms. This platform makes it also easy to compare the new algorithms to already implemented algorithms. The wide selection of available datasets for the WEKA framework provides a good basis to evaluate machine learning algorithms under different conditions. WEKA also provides many tools to analyze the performance of algorithms and to visualize their results and data inputs.

The WEKA web page provides many tutorials and very good documentation for the user. The documentation quality is different for the three frameworks. The tutorials provide example datasets and explain especially the classification algorithms very well. The Association framework has nearly no documentation. The Clustering framework is not as well explained as the Classification framework but there is a reasonable amount of documentation about the Clustering framework. The amount of documentation appears to be correlated with the number of implemented algorithms.

Weaknesses:

The design of the WEKA framework caused several problems resulting from assumptions it makes that were not valid for MAXCCLUS.

1. the framework was design for machine learning algorithm that perform classification tasks
2. the framework was built on the assumption that all instances have a fixed number of attributes
3. it assumes that clustering algorithms use a distance function to produce clusters
4. the file format is restricted and obsolete; a data format based on XML would be more flexible

The data structures were designed for machine learning algorithms that perform classification tasks. So the WEKA data structure only provides support of machine learning algorithm that perform classification task and it provides very good support for algorithms that perform classification and distance-based clustering. For example the clustering algorithm K-Means is a distances-based clustering algorithm which needs to calculate the distance between two instances to allocate a instance to a cluster and to recalculate the mean of the cluster. As this method would not work with non-numeric attributes which are a valid attribute type in WEKA. The WEKA data structure needs to generate numeric identifiers for the non-numeric attribute values. This is done based a scalar that needs to be stored in the data structure as well.

Also the complete software design has a lack of consistency. For example the design of the three algorithm frameworks varies a lot in terms of specification of the interface. On the one hand the classification framework is well specified and provides good support for classification algorithms. It would not be possible to implement a algorithm that does not produce a classifier in this framework. On the other hand the Association framework has no detailed specification. It does not define restrictions for implemenation of the algorithm as long as it read the WEKA data structure and return a string as result. Also the associaction framework provides little support for Association algorithms and it is possible to implement other algorithm within this framework.

The data format was also designed under the assumption that all algorithm implemented in WEKA will perform classification and require a data input with a fixed number of attributes. Because of this the WEKA data format is not flexible. This project showed that it can be used for other algorithms as well, but this will always involve a pre-processing stage.

The documentation for developers is of neither high quality nor quantity. Especially the extensibility features are not very well described. As WEKA is written in Java it has an API but there are no examples or tutorials about adding a new algorithm to the WEKA toolkit. The only framework that is well described is the classification algorithm. The classification is narrowly designed so it can not be used for different machine learning algorithms.

To run the WEKA version of MAXCCLUS with a data set that contains 2337 instances it was necesarry to increase the memory for the JVM up to $512MB$. Otherwise the WEKA version of MAXCCLUS could not process this amount of data. Apparently it was not the data reading step or the data structure mapping that caused the unproptional increase in processing time compared to the stand-alone version of MAXCCLUS or a run of the WEKA version of MAXCCLUS with a data set that containing 999 instances. The not linear increase of the MAXCCLUS permutation test by increasing the number of instances linear can cause this unproportinal increase in processing time. That means WEKA must cause this overhead.

The processing time of the WEKA version of MAXCCLUS processing 2337 instances

```
Total time: 14 minutes 16 seconds
Total time: 14 minutes 43 seconds
Total time: 14 minutes 27 seconds
Total time: 14 minutes 28 seconds
```

The processing time of MAXCCLUS stand-alone processing 2337 instances

```
Total time: 2 minutes 6 seconds
```

```
Total time: 3 minutes 24 seconds
Total time: 2 minutes 43 seconds
Total time: 2 minutes 20 seconds
```

The processing time of the WEKA version of MAXCCLUS processing 999 instances

```
Total time: 1 minute 5 seconds
Total time: 1 minute 41 seconds
Total time: 47 seconds
Total time: 1 minute 31 seconds
```

The processing time of MAXCCLUS stand-alone processing 999 instances

```
Total time: 17 seconds
Total time: 10 seconds
Total time: 13 seconds
Total time: 17 seconds
```

Chapter 5

Conclusion

The algorithm fits into the WEKA toolkit so the MAXCCLUS algorithm was successfully integrate into the WEKA toolkit. It was implemented in the association framework as described in 3.2. The WEKA version of MAXCCLUS can produce rules and clusters for any data inputs which consist of a classification, a key and a list of descriptors.

To import the MAXCCLUS algorithm into WEKA it was necessary to understand the general process of clustering. This was done by a writing literature review about the three standard clustering algorithms K-Means, Cobweb and agglomerative clustering. To re-implement he MAXCCLUS algorithm it was required to understand the algorithm and the MAXCCLUS source code. To specify the requirements for the integration of MAXCCLUS into the WEKA framework it was necesarry to understand the design and the implementation of WEKA.

To keep the complete WEKA data processing process consistent WEKA provides its own file format. This format was also defined to support machine learning algorithm that perform classification tasks, and so it only allows instances with a fixed pre-defined number of attributes.

To use the original MAXCCLUS data sets (the gene expression files and the SwissProt gene description file) with the WEKA version of MAXCCLUS. The data converter tool needs to process both data input and convert them into a valid ARFF file because the WEKA toolkit can only read this data format. This data format is unflexible and out of date, it only allows instances with a fixed number of predefined attributes. A data input based on Extensible Markup Language (XML) would have serveral advantages: XML is self-descriptive and new algorithms could use their own data descriptions. To read the data it is only required to implement a new XML parser to process to XML file.

Although the WEKA toolkit is a good framework for machine learning algorithms which classify new instances, it provides no support for other kinds of machine learning algorithms or algorithm requiring data with a different structure. Because the WEKA version of MAXCCLUS needs extra memory to store the data twice and needs to perform several extra pre-processing steps. The process is shown in figure 3.1.

WEKA performs rather poor on data set with a large number of instances the increase in the processing time is not proportial to the increase in processing time of the stand-alone implementation of MAXCCLUS.

Bibliography

[1] ANDREAE, P. Unpublished report on maxcclus.

[2] DELTOUR, A. Tertius extension to weka. Tech. Rep. CSTR-01-001, Department of Computer Science, University of Bristol, September 2001.

[3] E. FALKENAUER, A. M. Using k-means? consider arrayminer.

[4] FISHER, D. H. Knowledge acquisition via incremental conceptual clustering. *Machine Learning 2* (1987), 139–172.

[5] FRIEDMAN, N. Pcluster: Probabilistic agglomerative clustering of gene expression profiles.

[6] GORDON PAYNTER, LEN TRIGG, E. F., AND KIRKBY, R. Documenting the arff data format used by weka. http://www.cs.waikato.ac.nz/ml/weka/, 2006.

[7] JAIN, A. K., MURTY, M. N., AND FLYNN, P. J. Data clustering: a review. *ACM Comput. Surv. 31*, 3 (1999), 264–323.

[8] MACKAY, D. J. C. *An example infernce task: clustering*. Cambridge University Press, 2003, pp. 284–292.

[9] WITTEN, I. H., AND FRANK, E. *Nuts and bolts: Machine learning algorithms in Java*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005, pp. 265–319.