

KARNATAKA STATE OPEN UNIVERSITY
MUKTHAGANGOTRI, MYSORE- 570 006

DEPARTMENT OF STUDIES IN INFORMATION TECHNOLOGY



M.Sc IN INFORMATION SCIENCE
II SEMESTER



PRACTICAL 3
INFORMATION ORGANIZATION AND RETRIEVAL

IS 2.5

IS 2.5

Practical 3 :

**LABORATORY MANUAL FOR
INFORMATION ORGANIZATION
AND RETRIEVAL**

Course Design and Editorial Committee

Prof. M.|G.Krishnan

Vice Chancellor & Chairperson
Karnataka State Open University
Manasagangotri, Mysore – 570 006

Prof. Vikram Raj Urs

Dean (Academic) & Convener
Karnataka State Open University
Manasagangotri, Mysore – 570 006

Head of the Department**Rashmi B.S**

Assistant professor & Chairperson
DoS in Information Technology
Karnataka State Open University
Manasagangotri, Mysore – 570 006

Course Co-Ordinator**Mr. Mahesha DM**

Assistant professor in Computer Science
DoS in Computer Science
Karnataka State Open University
Manasagangotri, Mysore – 570 006

Course Editor

Ms. Nandini H.M

Assistant professor of Information Technology
DoS in Information Technology
Karnataka State Open University
Manasagangotri, Mysore – 570 006

Course Writers

Dr. B. H. Shekar

Associate Professor & Chairman
Department of Computer Science
Manasagangotri
Mangalore University

Dr. Manjaiah D H

Associate Professor
Department of Computer Science
Manasagangotri
Mangalore University

Publisher

Registrar

Karnataka State Open University
Manasagangotri, Mysore – 570 006

Developed by Academic Section, KSOU, Mysore

Karnataka State Open University, 2012

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any other means, without permission in writing from the Karnataka State Open University.

Further information on the Karnataka State Open University Programmes may be obtained from the University's Office at Manasagangotri, Mysore – 6.

Printed and Published on behalf of Karnataka State Open University, Mysore-6 by the

Registrar (Administration)

ALGORITHMS FOR INFORMATION ORGANIZATION AND RETRIEVAL INTRODUCTION

1. Purpose:

- *Short introduction to Information Retrieval.*
- *The importance of Information Retrieval Systems.*
- *Short presentation of most common algorithms used for Information Retrieval and Data Mining.*

2. Information Retrieval Introduction

2.1 What is Information Retrieval?

Information retrieval (IR) - finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

Information retrieval is a *problem-oriented* discipline, concerned with the problem of the effective and efficient transfer of desired information between human generator and human user

In other words:

- The indexing and retrieval of textual documents.
- Concerned firstly with retrieving *relevant* documents to a query.
- Concerned secondly with retrieving from *large* sets of documents *efficiently*.

2.2 Why IR? – A Simple Example.

Suppose there is a store of documents and a person (user of the store) formulates a question (request or query) to which the answer is a set of documents satisfying the information need expressed by his question.

Solution: User can read all the documents in the store, retain the relevant documents and discard all the others – *Perfect Retrieval... NOT POSSIBLE !!!*

Alternative: Use a High Speed Computer to *read* entire document collection and extract the *relevant* documents.

Goal = find documents *relevant* to an information need from a large document set.

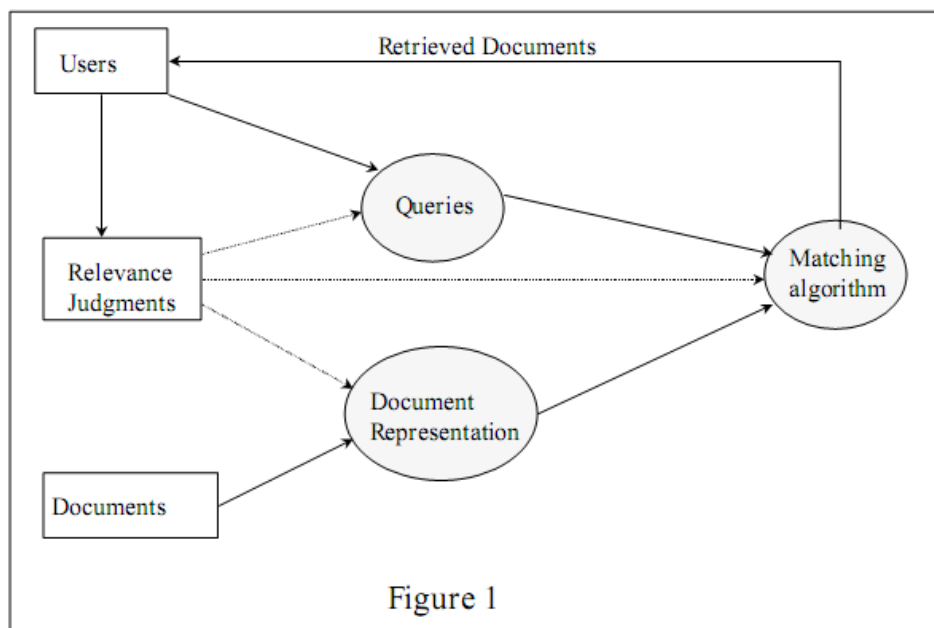
2.3 Role:

Three main areas of Research:

- Content Analysis: Describing the contents of documents in a form suitable for computer processing;
- Information Structures: Exploiting relationships between documents to improve the efficiency and effectiveness of retrieval strategies;
- Evaluation: the measurement of the effectiveness of retrieval.
 - Precision - The ability to retrieve top-ranked documents that are mostly relevant.
 - Recall - The ability of the search to find **all** of the relevant items in the corpus.

3. Information Retrieval Systems

A document based IR system typically consists of three main subsystems: document representation, representation of users' requirements (queries), and the algorithms used to match user requirements (queries) with document representations. The basic architecture is as shown in figure 1.



A document collection consists of many documents containing information about various subjects or topics of interests. Document contents are transformed into a document representation (either manually or automatically). Document representations are done in a way such that matching these with queries is easy. Another consideration in document representation is that such a representation should correctly reflect the author's intention. The

primary concern in representation is how to select proper index terms. Typically representation proceeds by extracting keywords that are considered as content identifiers and organizing them into a given format.

Queries transform the user's information need into a form that correctly represents the user's underlying information requirement and is suitable for the matching process. Query formatting depends on the underlying model of retrieval used. The user rates documents presented as either relevant or non-relevant to his/her information need. The basic problem facing any IR system is how to retrieve only the relevant documents for the user's information requirements, while not retrieving non-relevant ones.

Various system performance criteria like precision and recall have been used to gauge the effectiveness of the system in meeting users' information requirements.

Recall is the ratio of the number of relevant retrieved documents to the total number of relevant documents available in the document collection. Precision is defined as the ratio of the number of relevant retrieved documents to the total number of retrieved documents. Relevance feedback is typically used by the system (dotted arrows in figure 1) to improve document descriptions or queries, with the expectation that the overall performance of the system will improve after such a feedback.

4. Areas of IR application

Information retrieval (IR) systems were originally developed to help manage the huge scientific literature that has developed since the 1940s. Many university, corporate, and public libraries now use IR systems to provide access to books, journals, and other documents. Commercial IR systems offer databases containing millions of documents in myriad subject areas. Dictionary and encyclopaedia databases are now widely available for PCs. IR has been found useful in such disparate areas as office automation and software engineering. Indeed, any discipline that relies on documents to do its work could potentially use and benefit from IR. Information retrieval is used today in many applications. Is used to search for documents, content thereof, document metadata within traditional relational databases or internet documents more conveniently and decrease work to access information. Retrieved documents should be relevant to a user's information need. Obvious examples include search engines as Google, Yahoo or Microsoft Live Search. Many problems in information retrieval can be viewed as a prediction problem, i.e. to predict ranking scores or ratings of web pages, documents, music songs etc. and learning the information desires and interests of users.

4.1 General applications of information retrieval:

4.1.1 Digital Library

A digital library is a library in which collections are stored in digital formats (as opposed to print, microform, or other media) and accessible by computers. The digital content may be stored locally, or accessed remotely via computer networks. A digital library is a type of information retrieval system.

Many academic libraries are actively involved in building institutional repositories of the institution's books, papers, theses, and other works which can be digitized or were 'born digital'.

Many of these repositories are made available to the general public with few restrictions, in accordance with the goals of open access, in contrast to the publication of research in commercial journals, where the publishers often limit access rights. Institutional, truly free, and corporate repositories are sometimes referred to as digital libraries.

4.1.2 Recommender systems

Recommender systems or recommendation engines form or work from a specific type of information filtering system technique that attempts to recommend information items (films, television, video on demand, music, books, news, images, web pages, etc) that are likely to be of interest to the user. Typically, a recommender system compares a user profile to some reference characteristics, and seeks to predict the 'rating' that a user would give to an item they had not yet considered. These characteristics may be from the information item (the content-based approach) or the user's social environment (the collaborative filtering approach). Collaborative filtering is concerned with making recommendation about information items (movies, music, books, news, web pages) to users. Based on the "Word of Mouth" phenomenon, it recommends items that like-minded people liked in the past. Although collaborative filtering is an effective way to alleviate information overload and has been widely adopted in e-commerce websites, collecting user preference data is not trivial because it may raise serious concerns about the privacy of individuals.

4.1.3 Search Engines

A search engine is one of the most the practical applications of information retrieval techniques to large scale text collections. Web search engines are best known examples, but many others searches exist, like: Desktop search, Enterprise search, Federated search, Mobile search, and Social search.

A web search engine is designed to search for information on the World Wide Web. The search results are usually presented in a list of results and are commonly called *hits*. The information may consist of web pages, images, information and other types of files. Some search engines also mine data available in databases or open directories. Unlike Web directories, which are maintained by human editors, search engines operate algorithmically or are a mixture of algorithmic and human input.

Relevance feedback is an important issue of information retrieval found in web searching. Reliability of information is a pre-requisite to get most from research information found onto the web. A frequently encountered issue is that search terms are ambiguous and thus documents from a different non-relevant context are retrieved or you may not know which terms describe your problem properly, especially if you are a non-expert user in this particular domain.

The novel idea of relevance feedback allows users to rate retrieved documents as relevant or less relevant and thus help other users to find documents more quickly. These ideas were adopted from image retrieval. Images are hard to describe using words.

4.1.4 Media search

An image retrieval system is a computer system for browsing, searching and retrieving images from a large database of digital images. Most traditional and common methods of image retrieval utilize some method of adding metadata such as captioning, keywords, or descriptions to the images so that retrieval can be performed over the annotation words. Manual image annotation is time-consuming, laborious and expensive; to address this, there has been a large amount of research done on automatic image annotation. Additionally, the increase in social web applications and the semantic web have inspired the development of several web-based image annotation tools.

5. IR Algorithms

It is hard to classify IR algorithms, and to draw a line between each type of application. However, we can identify three main types of algorithms, which are described below. There are other algorithms used in IR that do not fall within our description, for example, user interface algorithms. The reason that they cannot be considered as IR algorithms is because they are inherent to any computer application. We distinguish three main classes of algorithms. These are retrieval, indexing, and filtering algorithms.

5.1 Retrieval Algorithms

The main class of algorithms in IR is retrieval algorithms, that is, to extract information from a textual database. We can distinguish two types of retrieval algorithms, according to how much extra memory we need:

- Sequential scanning of the text: extra memory is in the worst case a function of the query size, and not of the database size. On the other hand, the running time is at least proportional to the size of the text, for example, string searching.
- Indexed text: an "index" of the text is available, and can be used to speed up the search. The index size is usually proportional to the database size, and the search time is sub-linear on the size of the text, for example, inverted files and signature files.

Formally, we can describe a generic searching problem as follows: Given a string t (the text), a regular expression q (the query), and information (optionally) obtained by pre-processing the pattern and/or the text, the problem consists of finding whether $t \in \Sigma^*q$ (q for short) and obtaining some or all of the following information:

1. The location where an occurrence (or specifically the first, the longest, etc.) of q exists. Formally, if $t \in \Sigma^*q$ find a position $m \geq 0$ such that $t \in \Sigma^m q \Sigma^*$. For example, the first occurrence is defined as the least m that fulfills this condition.
2. The number of occurrences of the pattern in the text. Formally, the number of all possible values of m in the previous category.
3. All the locations where the pattern occurs (the set of all possible values of m).

In general, the complexities of these problems are different.

The efficiency of retrieval algorithms is very important, because we expect them to solve on-line queries with a short answer time. This need has triggered the implementation of retrieval algorithms in many different ways: by hardware, by parallel machines, and so on.

5.2 Filtering Algorithms

This class of algorithms is such that the text is the input and a processed or filtered version of the text is the output. This is a typical transformation in IR, for example to reduce the size of a text, and/or standardize it to simplify searching.

The most common filtering/processing operations are:

- Common words removed using a list of stop words;
- Uppercase letters transformed to lowercase letters;
- Special symbols removed and sequences of multiple spaces reduced to one space;
- Numbers and dates transformed to a standard format;
- Word stemming (removing suffixes and/or prefixes);

- Automatic keyword extraction;
- Word ranking.

Unfortunately, these filtering operations may also have some disadvantages. Any query, before consulting the database, must be filtered as is the text; and, it is not possible to search for common words, special symbols, or uppercase letters, nor to distinguish text fragments that have been mapped to the same internal form.

5.3 Indexing Algorithms

The usual meaning of indexing is to build a data structure that will allow quick searching of the text, as we mentioned previously. There are many classes of indices, based on different retrieval approaches. For example, we have inverted files, signature files, tries, and so on.

Almost all types of indices are based on some kind of tree or hashing. Perhaps the main exceptions are clustered data structures (this kind of indexing is called clustering), which is covered in further laboratories, and the Direct Acyclic Word Graph (DAWG) of the text, which represents all possible sub-words of the text using a linear amount of space and is based on finite automata theory.

Usually, before indexing, the text is filtered. Figure 2 shows the complete process for the text.

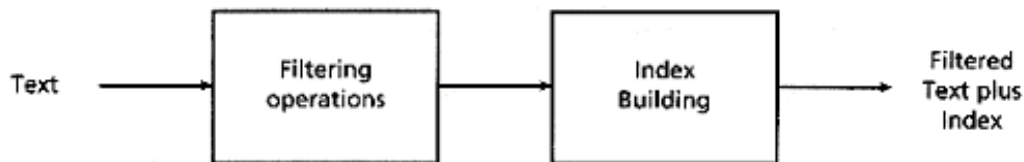


Figure 2: Text pre-processing

The pre-processing time needed to build the index is amortized by using it in searches. For example, if building the index requires $O(n \log n)$ time, we would expect to query the database at least $O(n)$ times to amortize the pre-processing cost. In that case, we add $O(\log n)$ pre-processing time to the total query time (that may also be logarithmic).

6. Data Mining

6.1 Introduction

A topic related to Information Retrieval is Data Mining.

Data mining is the process of extracting patterns from data. Data mining is becoming an increasingly important tool to transform this data into information. It is commonly used in a

wide range of profiling practices, such as marketing, surveillance, fraud detection and scientific discovery.

Data mining can be used to uncover patterns in data but is often carried out only on samples of data. The mining process will be ineffective if the samples are not a good representation of the larger body of data. Data mining cannot discover patterns that may be present in the larger body of data if those patterns are not present in the sample being "mined".

Inability to find patterns may become a cause for some disputes between customers and service providers. Therefore data mining is not foolproof but may be useful if sufficiently representative data samples are collected. The discovery of a particular pattern in a particular set of data does not necessarily mean that a pattern is found elsewhere in the larger data from which that sample was drawn. An important part of the process is the verification and validation of patterns on other samples of data.

Data mining commonly involves the following classes of tasks:

- Classification - Arranges the data into predefined groups. For example, an email program might attempt to classify an email as legitimate or spam. Common algorithms include decision tree learning, nearest neighbour, naive Bayesian classification and neural networks.
- Clustering - Is like classification but the groups are not predefined, so the algorithm will try to group similar items together.
- Regression - Attempts to find a function which models the data with the least error.

6.2 Regression

Regression is the oldest and most well-known statistical technique that the data mining community utilizes. Basically, regression takes a numerical dataset and develops a mathematical formula that fits the data. When you're ready to use the results to predict future behaviour, you simply take your new data, plug it into the developed formula and you've got a prediction! The major limitation of this technique is that it only works well with continuous quantitative data (like weight, speed or age). If you're working with categorical data where order is not significant (like colour, name or gender) you're better off choosing another technique.

6.2.1 Regression algorithms:

- Linear regression involves finding the "best" line to fit two attributes (or variables), so that one attribute can be used to predict the other.

- Multiple linear regression is an extension of linear regression, where more than two attributes are involved and the data are fit to a multidimensional surface.

6.3 Classification

Classification is one of the major data mining tasks. Although this task is accomplished by generating a predictive model of data, interpreting the model frequently provides information for discriminating labelled classes in data

6.3.1 Classification Algorithms

Brief overview of basic classification algorithms

The goal of classification is to build a set of models that can correctly predict the class of the different objects. The input to these methods is a set of objects (i.e., training data), the classes which these objects belong to (i.e., dependent variables), and a set of variables describing different characteristics of the objects (i.e., independent variables). Once such a predictive model is built, it can be used to predict the class of the objects for which class information is not known a priori. The key advantage of supervised learning methods over unsupervised methods (for example, clustering) is that by having an explicit knowledge of the classes the different objects belong to, these algorithms can perform an effective feature selection if that leads to better prediction accuracy.

The followings are brief overview on some classification algorithms that has been used in data mining and machine learning area and used as base algorithms in this course.

6.3.1.1 k-Nearest Neighbour (KNN) Algorithm

KNN classifier is an instance-based learning algorithm that is based on a distance function for pairs of observations, such as the Euclidean distance or Cosine. In this classification paradigm, k nearest neighbours of a training data are computed first. Then the similarities of one sample from testing data to the k nearest neighbours are aggregated according to the class of the neighbors, and the testing sample is assigned to the most similar class. One of advantages of KNN is that it is well suited for multi-modal classes as its classification decision is based on a small neighborhood of similar objects (i.e., the major class). So, even if the target class is multimodal (i.e., consists of objects whose independent variables have different characteristics for different subsets), it can still lead to good accuracy. A major drawback of the similarity measure used in KNN is that it uses all features equally in

computing similarities. This can lead to poor similarity measures and classification errors, when only a small subset of the features is useful for classification.

6.3.1.2 Naive Bayesian (NB) Algorithm

NB algorithm has been widely used for document classification, and shown to produce very good performance. The basic idea is to use the joint probabilities of words and categories to estimate the probabilities of categories given a document. NB algorithm computes the posterior probability that the document belongs to different classes and assigns it to the class with the highest posterior probability. The posterior probability of class is computed using Bayes rule and the testing sample is assigned to the class with the highest posterior probability. The naive part of NB algorithm is the assumption of word independence that the conditional probability of a word given a category is assumed to be independent from the conditional probabilities of other words given that category. There are two versions of NB algorithm. One is the multi-variate Bernoulli event model that only takes into account the presence or absence of a particular term, so it doesn't capture the number of occurrence of each word. The other model is the multinomial model that captures the word frequency information in documents.

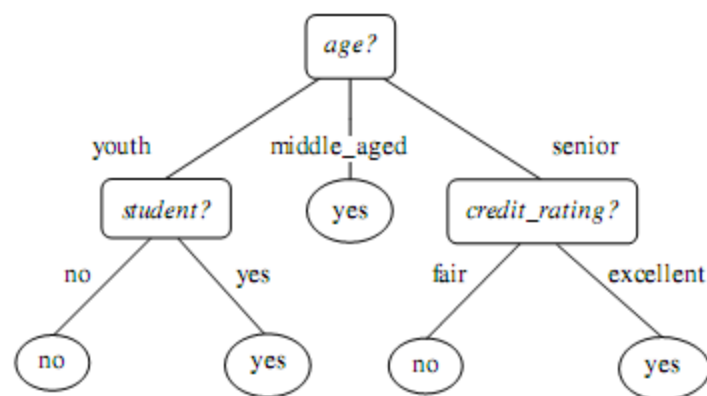
6.3.1.3 Concept Vector-based (CB) Algorithm

In CB classification algorithm, the length of each vector is normalized so that it is of unit length. The idea behind the concept-based classification is extremely simple. For each set of documents belonging to the same class, we compute its concept vector by summing up all vectors in the class and normalize it by its 2-norm. If there are c classes in the training data set, this leads to c concept vectors, where each concept vector for each class. The class of a new sample is determined as follow. First, for a given testing document, which was already normalized by 2-norm so that it has unit length, we compute cosine similarity between this given testing document to all k concept vectors. Then, based on these similarities, we assign a class label so that it corresponds to the most similar concept vector's label. One of the advantages of the CB classification algorithm is that it summarizes the characteristics of each class, in the form of concept vector. So, the advantage of the summarization performed by the concept vectors is that it combines multiple prevalent features together, even if these features are not simultaneously present in a single document. That is, if we look at the prominent dimensions of the concept vector (i.e., highest weight terms), these will correspond to words that appear frequently in the class, but not necessarily all in the same set of documents. This

is particularly important for high dimensional and sparse data sets for which the coverage of any individual feature is often quite low.

6.3.1.4 Decision Tree Induction

Decision tree induction is the learning of decision trees from class-labelled training tuples. A decision tree is a flowchart-like tree structure, where each internal node (non leaf node) denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (or terminal node) holds a class label. The top most node in a tree is the root node.



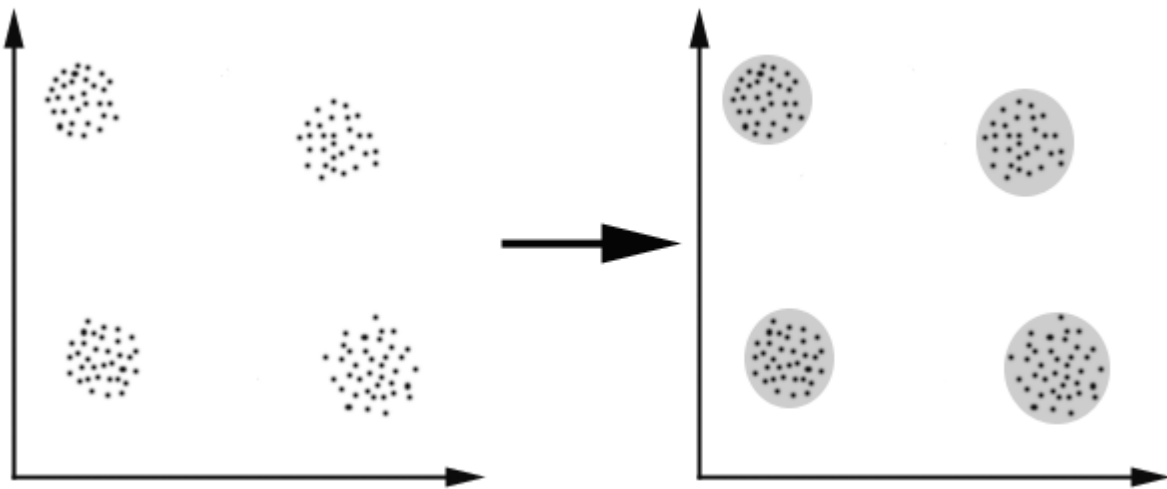
“How are decision trees used for classification?” Given a tuple, X , for which the associated class label is unknown, the attribute values of the tuple are tested against the decision tree. A path is traced from the root to a leaf node, which holds the class prediction for that tuple.

Decision trees can easily be converted to classification rules.

“Why are decision tree classifiers so popular?” The construction of decision tree classifiers does not require any domain knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery. Decision trees can handle high dimensional data. Their representation of acquired knowledge in tree form is intuitive and generally easy to assimilate by humans. The learning and classification steps of decision tree induction are simple and fast. In general, decision tree classifiers have good accuracy. However, successful use may depend on the data at hand. Decision tree induction algorithms have been used for classification in many application areas, such as medicine, manufacturing and production, financial analysis, astronomy, and molecular biology.

6.4 Clustering

Clustering can be considered the most important *unsupervised learning* problem; so, as every other problem of this kind, it deals with finding a *structure* in a collection of unlabeled data. A loose definition of clustering could be “the process of organizing objects into groups whose members are similar in some way”. A *cluster* is therefore a collection of objects which are “similar” between them and are “dissimilar” to the objects belonging to other clusters. We can show this with a simple graphical example:



6.4.1 K-means clustering:

The aim of K-means (or clustering) is: We want to group the items into k clusters such that all items in same cluster are as similar to each other as possible. And items not in same cluster are as different as possible. We use the distance measures to calculate similarity and dissimilarity. One of the important concept in K-means is that of centroid. Each cluster has a centroid. You can consider it as the point that is most representative of the cluster. Equivalently, centroid is point that is the "center" of a cluster.

Algorithm:

1. Randomly choose k items and make them as initial centroids.
2. For each point, find the nearest centroid and assign the point to the cluster associated with the nearest centroid.
3. Update the centroid of each cluster based on the items in that cluster. Typically, the new centroid will be the average of all points in the cluster.
4. Repeats steps 2 and 3, till no point switches clusters.

5. As you can see, the algorithm is extremely simple. After some iterations, we will get k -clusters within which each points are similar.

6.4.2 Hierarchical Clustering Algorithms

Given a set of N items to be clustered, and an $N \times N$ distance (or similarity) matrix, the basic process of hierarchical clustering (defined by S.C. Johnson in 1967) is this:

1. Start by assigning each item to a cluster, so that if you have N items, you now have N clusters, each containing just one item. Let the distances (similarities) between the clusters the same as the distances (similarities) between the items they contain.
2. Find the closest (most similar) pair of clusters and merge them into a single cluster, so that now you have one cluster less.
3. Compute distances (similarities) between the new cluster and each of the old clusters.
4. Repeat steps 2 and 3 until all items are clustered into a single cluster of size N . (*)

References:

- [1] Information Retrieval Data Structures & Algorithms - William B. Frakes and Ricardo Baeza-Yates
- [2] Introduction to Information Retrieval – Christopher D. Manning, Prabhakar Raghavan, Hinrich Schutze
- [3] Data Mining: Concepts and Techniques - Jiawei Han & Micheline Kamber

DESCRIPTION OF WEKA

-A JAVA-IMPLEMENTED MACHINE LEARNING TOOL

Purpose:

- Install and run WEKA
- Experiment environment in GUI version and in command line version

1. Theoretical Aspects

1.1. What is WEKA?

WEKA is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. WEKA contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes. WEKA is used for research, education, and applications. The tool gathers a comprehensive set of data pre-processing tools, learning algorithms and evaluation methods, graphical user interfaces (incl. data visualization) and environment for comparing learning algorithms.

WEKA is open source software issued under the GNU General Public License. "WEKA" stands for the Waikato Environment for Knowledge Analysis, which was developed at the University of Waikato in New Zealand. WEKA is extensible and has become a collection of machine learning algorithms for solving real-world data mining problems. It is written in Java and runs on almost every platform.

WEKA is easy to use and to be applied at several different levels. You can access the WEKA class library from your own Java program, and implement new machine learning algorithms. There are three major implemented schemes in WEKA. (1) Implemented schemes for classification. (2) Implemented schemes for numeric prediction. (3) Implemented "meta-schemes". Besides actual learning schemes, WEKA also contains a large variety of tools that can be used for pre-processing datasets, so that you can focus on your algorithm without considering too much details as reading the data from files, implementing filtering algorithm and providing code to evaluate the results.

Some practical applications that use WEKA:

Acronym identification

This addresses the task of finding acronym-definition pairs in text. Most of the previous work on the topic is about systems that involve manually generated rules or regular expressions. In this manual, we present a supervised learning approach to the acronym identification task. Our approach reduces the search space of the supervised learning system by putting some weak constraints on the kinds of acronym-definition pairs that can be identified. We obtain results comparable to hand-crafted systems

that use stronger constraints. We describe our method for reducing the search space, the features used by our supervised learning system, and our experiments with various learning schemes.

Gene selection from microarray data for cancer classification

A DNA microarray can track the expression levels of thousands of genes simultaneously. Previous research has demonstrated that this technology can be useful in the classification of cancers. Cancer microarray data normally contains a small number of samples which have a large number of gene expression levels as features. To select relevant genes involved in different types of cancer remains a challenge. In order to extract useful gene information from cancer microarray data and reduce dimensionality, feature selection algorithms were systematically investigated in this study.

Using a correlation-based feature selector combined with machine learning algorithms such as decision trees, naive Bayes and support vector machines, we show that classification performance at least as good as published results can be obtained on acute leukemia and diffuse large B-cell lymphoma microarray data sets. We also demonstrate that a combined use of different classification and feature selection approaches makes it possible to select relevant genes with high confidence. This is also the first paper which discusses both computational and biological evidence for the involvement of zyxin in leukaemogenesis.

Benchmarking of Linear and Nonlinear Approaches for Quantitative Structure–Property Relationship Studies of Metal Complexation with Ionophores

A benchmark of several popular methods, Associative Neural Networks (ANN), Support Vector Machines (SVM), k Nearest Neighbors (kNN), Maximal Margin Linear Programming (MMLP), Radial Basis Function Neural Network (RBFNN), and Multiple Linear Regression (MLR), is reported for quantitative–structure property relationships (QSPR) of stability constants $\log K_1$ for the 1:1 (M:L) and \log_2 for 1:2 complexes of metal cations Ag^+ and Eu^{3+} with diverse sets of organic molecules in water at 298 K and ionic strength 0.1 M. The methods were tested on three types of descriptors: molecular descriptors including E-state values, counts of atoms determined for E-state atom types, and substructural molecular fragments (SMF).

1.2. Installing and running WEKA

1.2.1. In lab (this assumes WEKA is already installed)

1.2.2. On your home computer

For installing WEKA on your home computer you must check the following:

There are two stable versions of WEKA. Either you can download the self-extraction executable version that includes the Java Virtual Machine 1.4 (WEKA-3-4jre.exe; 19,543,851 bytes), <http://prdownloads.sourceforge.net/WEKA/WEKA-3-4jre.exe> or the self-extracting executable without Java VM (WEKA-3-4.exe; 6,467,165 bytes).

<http://prdownloads.sourceforge.net/WEKA/WEKA-3-4.exe>

This version comes with the GUI, which provides the user with more flexibility than the command line.

After extracting the files, you will need to set your *classpath* variable to a complete path to WEKA.jar (suppose you extracted WEKA to C:\WEKA, then set your *classpath* variable to C:\WEKA\WEKA.jar, ie add "C:\WEKA\WEKA.jar;" to the list of values that environment variable Path can take when working in Windows)

If you don't have administrator privileges, you can still install WEKA. For that, download the jar archive (WEKA-3-4.jar; 6,322,417 bytes).

<http://prdownloads.sourceforge.net/WEKA/WEKA-3-4.jar>

Make sure that the Java J2SE 1.4 (download from SUN) is installed on your system (which includes the jar utility). Then open a command line console, change into the directory containing WEKA-3-4.jar, and enter

```
jar -xvf WEKA-3-4.jar
```

This will create a new directory called WEKA-3-4. To un-jar (install) the source code, position yourself in the recently created WEKA-3-4 directory and type

```
jar -xvf WEKA-src.jar
```

Which will create a new directory WEKA containing the source code. Since WEKA is open source software issued under the GNU General Public License, you can use and modify the source code as you like.

NOTE: It seems that Windows will not set up your CLASSPATH properly if any of the WEKA directories contains spaces. Therefore, installing WEKA in the Program Files folder is not a good idea.

1.3. Online documentation and further help

From your WEKA-3-4 directory, you will find:

- A jarfile containing the classes only
- A jarfile containing the complete source code
- The tutorial for the experiment environment in the GUI version of WEKA (written by David Scuse), and the README file
- The API documentation
- Some example datasets

The most detailed and up-to-date information could be found in the online documentation on WEKA Web Site. This page has a lot of documentation and guides on installation/usage pages.

http://www.cs.waikato.ac.nz/~ml/WEKA/index_documentation.html

1.4. Launching WEKA

The WEKA GUI Chooser (class `WEKA.gui.GUIChooser`) provides a starting point for launching WEKA's main GUI applications and supporting tools. If one prefers a MDI ("multiple document interface") appearance, then this is provided by an alternative launcher called "Main" (class `WEKA.gui.Main`).

The GUI Chooser consists of four buttons—one for each of the four major WEKA applications—and four menus.

The buttons can be used to start the following applications:

- **Explorer** An environment for exploring data with WEKA (the rest of this documentation deals with this application in more detail).
- **Experimenter** An environment for performing experiments and conducting statistical tests between learning schemes.
- **KnowledgeFlow** This environment supports essentially the same functions as the Explorer but with a drag-and-drop interface. One advantage is that it supports incremental learning.
- **SimpleCLI** Provides a simple command-line interface that allows direct execution of WEKA commands for operating systems that do not provide their own command line interface.

The menu consists of four sections:

1. Program

- **LogWindow** Opens a log window that captures all that is printed to stdout or stderr. Useful for environments like MS Windows, where WEKA is normally not started from a terminal.
- **Exit** Closes WEKA.

2. Tools - Other useful applications.

- **ArffViewer** An MDI application for viewing ARFF files in spread-sheet format.
- **SqlViewer** Represents an SQL worksheet, for querying databases via JDBC.
- **Bayes net editor** An application for editing, visualizing and learning Bayes nets.

3. Visualization - Ways of visualizing data with WEKA.

- **Plot** For plotting a 2D plot of a dataset.
- **ROC** Displays a previously saved ROC curve.
- **TreeVisualizer** For displaying directed graphs, e.g., a decision tree.
- **GraphVisualizer** Visualizes XML BIF or DOT format graphs, e.g., for Bayesian networks.
- **BoundaryVisualizer** Allows the visualization of classifier decision boundaries in two dimensions.

4. Help - Online resources for WEKA can be found here.

- **WEKA homepage** Opens a browser window with WEKA's home page.

- **HOWTOs, code snippets, etc.** The general WEKAWiki [2], containing lots of examples and HOWTOs around the development and use of WEKA.
- **WEKA on Sourceforge** WEKA's project homepage on Sourceforge.net.
- **SystemInfo** Lists some internals about the Java/WEKA environment, e.g., the CLASSPATH.

1.5. Simple CLI

The Simple CLI provides full access to all WEKA classes, i.e., classifiers, filters, clusters, etc., but without the hassle of the CLASSPATH (it facilitates the one, with which WEKA was started). It offers a simple WEKA shell with separated command line and output.

1.5.1. Commands

The following commands are available in the Simple CLI:

- **java <classname> [<args>]** invokes a java class with the given arguments (if any)
- **break** stops the current thread, e.g., a running classifier, in a friendly manner
- **kill** stops the current thread in an unfriendly fashion
- **cls** clears the output area
- **exit** exits the Simple CLI
- **help [<command>]** provides an overview of the available commands if without a command name as argument, otherwise more help on the specified command

1.5.2. Invocation

In order to invoke a WEKA class, one has only to prefix the class with "java". This command tells the Simple CLI to load a class and execute it with any given parameters. E.g., the J48 classifier can be invoked on the iris dataset with the following command:

```
java WEKA.classifiers.trees.J48 -t c:/temp/iris.arff
```

1.5.3. Command Redirection

Starting with this version of WEKA one can perform a basic redirection:

```
java WEKA.classifiers.trees.J48 -t test.arff > j48.txt
```

Note: the > must be preceded and followed by a space, otherwise it is not recognized as redirection, but part of another parameter.

1.5.4. Command completion

Commands starting with java support completion for classnames and filenames via Tab (Alt+BackSpace deletes parts of the command again). In case that there are several matches, WEKA lists all possible matches.

- **package name completion**

- java WEKA.cl<Tab>
- results in the following output of possible matches of package names:
- Possible matches:
- WEKA.classifiers
- WEKA.clusterers
- **classname completion**
 - java WEKA.classifiers.meta.A<Tab>
 - lists the following classes
 - Possible matches:
 - WEKA.classifiers.meta.AdaBoostM1
 - WEKA.classifiers.meta.AdditiveRegression
 - WEKA.classifiers.meta.AttributeSelectedClassifier
- **filename completion**
 - In order for WEKA to determine whether a the string under the cursor
 - is a classname or a filename, filenames need to be absolute (Unix/Linux:
 - /some/path/file;Windows: C:\Some\Path\file) or relative and starting
 - with a dot (Unix/Linux: ./some/other/path/file;Windows: .\Some\Other\Path\file).

1.5.5. The WEKA Explorer

1.5.5.1. Section Tabs

At the very top of the window, just below the title bar, is a row of tabs. When the Explorer is first started only the first tab is active; the others are greyed out. This is because it is necessary to open (and potentially pre-process) a data set before starting to explore the data.

The tabs are as follows:

1. **Preprocess.** Choose and modify the data being acted on.
2. **Classify.** Train and test learning schemes that classify or perform regression.
3. **Cluster.** Learn clusters for the data.
4. **Associate.** Learn association rules for the data.
5. **Select attributes.** Select the most relevant attributes in the data.
6. **Visualize.** View an interactive 2D plot of the data.

Once the tabs are active, clicking on them flicks between different screens, on which the respective actions can be performed. The bottom area of the window (including the status box, the log button, and the WEKA bird) stays visible regardless of which section you are in.

1.5.5.2. Preprocessing

1.5.5.2.1. Opening Files

The first three buttons at the top of the preprocess section enable you to load data into WEKA:

- **Open file....** Brings up a dialog box allowing you to browse for the data file on the local filesystem.
- **Open URL....** Asks for a Uniform Resource Locator address for where the data is stored.
- **Open DB....** Reads data from a database.

1.5.5.2.2. The Current Relation

Once some data has been loaded, the Preprocess panel shows a variety of information. The Current relation box (the “current relation” is the currently loaded data, which can be interpreted as a single relational table in database terminology) has three entries:

- **Relation.** The name of the relation, as given in the file it was loaded from. Filters (described below) modify the name of a relation.
- **Instances.** The number of instances (data points/records) in the data.
- **Attributes.** The number of attributes (features) in the data.

1.5.5.2.3. Working with Attributes

Below the Current relation box is a box titled Attributes. There are three buttons, and beneath them is a list of the attributes in the current relation. The list has three columns:

- **No..** A number that identifies the attribute in the order they are specified in the data file.
- **Selection tick boxes.** These allow you select which attributes are present in the relation.
- **Name.** The name of the attribute, as it was declared in the data file. When you click on different rows in the list of attributes, the fields change in the box to the right titled Selected attribute. This box displays the characteristics of the currently highlighted attribute in the list:
 1. **Name.** The name of the attribute, the same as that given in the attribute list.
 2. **Type.** The type of attribute, most commonly Nominal or Numeric.
 3. **Missing.** The number (and percentage) of instances in the data for which this attribute is missing (unspecified).
 4. **Distinct.** The number of different values that the data contains for this attribute.
 5. **Unique.** The number (and percentage) of instances in the data having a value for this attribute that no other instances have.

Below these statistics is a list showing more information about the values stored in this attribute, which differ depending on its type. If the attribute is **nominal**, the list consists of each possible value for the attribute along with the number of instances that have that value. If the attribute is numeric, the list gives four statistics describing the distribution of values in the data - the minimum, maximum,

mean and standard deviation. And below these statistics there is a **colored histogram**, color-coded according to the attribute chosen as the **Class** using the box above the histogram. (This box will bring up a drop-down list of available selections when clicked.) Note that only nominal **Class** attributes will result in a color-coding. Finally, after pressing the **Visualize All** button, histograms for all the attributes in the data are shown in a separate window.

Returning to the attribute list, to begin with all the tick boxes are **unticked**. They can be toggled **on/off** by clicking on them individually. The three buttons above can also be used to change the selection:

1. **All**. All boxes are ticked.
2. **None**. All boxes are cleared (unticked).
3. **Invert**. Boxes that are ticked become unticked and vice versa.

Once the desired attributes have been selected, they can be removed by clicking the **Remove** button below the list of attributes. Note that this can be undone by clicking the **Undo** button, which is located next to the **Edit** button in the top-right corner of the Preprocess panel.

1.5.5.2.4. Working With Filters

The preprocess section allows filters to be defined that transform the data in various ways. The **Filter** box is used to set up the filters that are required. At the left of the **Filter** box is a **Choose** button. By clicking this button it is possible to select one of the filters in WEKA. Once a filter has been selected, its name and options are shown in the field next to the **Choose** button. Clicking on this box brings up a **GenericObjectEditor** dialog box.

The **GenericObjectEditor** dialog box lets you configure a filter. The same kind of dialog box is used to configure other objects, such as classifiers and clusters (see below). The fields in the window reflect the available options.

Clicking on any of these gives an opportunity to alter **the filters settings**. **For example**, the setting may take a text string, in which case you type the string into the text field provided. Or it may give a drop-down box listing several states to choose from. Or it may do something else, depending on the information required. Information on the options is provided in a tool tip if you let the mouse pointer over of the corresponding field. More information on the filter and its options can be obtained by clicking on the **More** button in the **About** panel at the top of the GenericObjectEditor window. Some objects display a brief description of what they do in an **About** box, along with a **More** button. Clicking on the **More** button brings up a window describing what the different options do.

At the bottom of the **GenericObjectEditor** dialog are four buttons. The first two, **Open...** and **Save...** allow object configurations to be stored for future use. The **Cancel** button backs out without remembering any changes that have been made. Once you are happy with the object and settings you have chosen, click **OK** to return to the main Explorer window.

Applying Filters

Once you have selected and configured a filter, you can apply it to the data by pressing the **Apply** button at the right end of the **Filter** panel in the **Preprocess** panel. The **Preprocess** panel will then show the transformed data. The change can be undone by pressing the **Undo** button. You can also use the **Edit...** button to modify your data manually in a dataset editor. Finally, the **Save...** button at the top right of the **Preprocess** panel saves the current version of the relation in the same formats available for loading data, allowing it to be kept for future use.

Note: Some of the filters behave differently depending on whether a class attribute has been set or not (using the box above the histogram, which will bring up a drop-down list of possible selections when clicked). In particular, the “supervised filters” require a class attribute to be set, and some of the “unsupervised attribute filters” will skip the class attribute if one is set. Note that it is also possible to set **Class** to **None**, in which case no class is set.

1.5.5.3. Classification

1.5.5.3.1. Selecting a Classifier

At the top of the classify section is the **Classifier** box. This box has a text field that gives the name of the currently selected classifier, and its options. Clicking on the text box brings up a **GenericObjectEditor** dialog box, just the same as for filters that you can use to configure the options of the current classifier. The **Choose** button allows you to choose one of the classifiers that are available in **WEKA**.

1.5.5.3.2. Test Options

The result of applying the chosen classifier will be tested according to the options that are set by clicking in the Test options box. There are four test modes:

1. **Use training set.** The classifier is evaluated on how well it predicts the class of the instances it was trained on.
2. **Supplied test set.** The classifier is evaluated on how well it predicts the class of a set of instances loaded from a file. Clicking the Set... button brings up a dialog allowing you to choose the file to test on.
3. **Cross-validation.** The classifier is evaluated by cross-validation, using the number of folds that are entered in the Folds text field.
4. **Percentage split.** The classifier is evaluated on how well it predicts a certain percentage of the data which is held out for testing. The amount of data held out depends on the value entered in the % field.

1.5.5.3.3. The Class Attribute

The classifiers in WEKA are designed to be trained to predict a single ‘class’ attribute, which is the target for prediction. Some classifiers can only learn nominal classes; others can only learn numeric classes (regression problems); still others can learn both. By default, the class is taken to be the last attribute in the data. If you want to train a classifier to predict a different attribute, click on the box below the **Test options** box to bring up a drop-down list of attributes to choose from.

1.5.5.3.4. Training a Classifier

Once the classifier, test options and class have all been set, the learning process is started by clicking on the **Start** button. While the classifier is busy being trained, the little bird moves around. You can stop the training process at any time by clicking on the **Stop** button. When training is complete, several things happen. The **Classifier output** area to the right of the display is filled with text describing the results of training and testing. A new entry appears in the **Result list** box. We look at the result list below; but first we investigate the text that has been output.

1.5.5.3.5. The Classifier Output Text

The text in the Classifier output area has scroll bars allowing you to browse the results. Of course, you can also resize the Explorer window to get a larger display area. The output is split into several sections:

- 1. Run information.** A list of information giving the learning scheme options, relation name, instances, attributes and test mode that were involved in the process.
- 2. Classifier model (full training set).** A textual representation of the classification model that was produced on the full training data.
- 3.** The results of the chosen test mode are broken down thus:
- 4. Summary.** A list of statistics summarizing how accurately the classifier was able to predict the true class of the instances under the chosen test mode.
- 5. Detailed Accuracy By Class.** A more detailed per-class break down of the classifier’s prediction accuracy.
- 6. Confusion Matrix.** Shows how many instances have been assigned to each class. Elements show the number of test examples whose actual class is the row and whose predicted class is the column.

1.5.5.3.6. The Result List

After training several classifiers, the result list will contain several entries. Left-clicking the entries flicks back and forth between the various results that have been generated. Right-clicking an entry invokes a menu containing these items:

- 1. View in main window.** Shows the output in the main window (just like left-clicking the entry).
- 2. View in separate window.** Opens a new independent window for viewing the results.

- 3. Save result buffer.** Brings up a dialog allowing you to save a text file containing the textual output.
- 4. Load model.** Loads a pre-trained model object from a binary file.
- 5. Save model.** Saves a model object to a binary file. Objects are saved in Java ‘serialized object’ form.
- 6. Re-evaluate model on current test set.** Takes the model that has been built and tests its performance on the data set that has been specified with the Set.. button under the Supplied test set option.
- 7. Visualize classifier errors.** Brings up a visualization window that plots the results of classification. Correctly classified instances are represented by crosses, whereas incorrectly classified ones show up as squares.
- 8. Visualize tree or Visualize graph.** Brings up a graphical representation of the structure of the classifier model, if possible (i.e. for decision trees or Bayesian networks). The graph visualization option only appears if a Bayesian network classifier has been built. In the tree visualizer, you can bring up a menu by right-clicking a blank area, pan around by dragging the mouse, and see the training instances at each node by clicking on it. CTRL-clicking zooms the view out, while SHIFT-dragging a box zooms the view in. The graph visualizer should be self-explanatory.
- 9. Visualize margin curve.** Generates a plot illustrating the prediction margin. The margin is defined as the difference between the probability predicted for the actual class and the highest probability predicted for the other classes. For example, boosting algorithms may achieve better performance on test data by increasing the margins on the training data.
- 10. Visualize threshold curve.** Generates a plot illustrating the tradeoffs in prediction that are obtained by varying the threshold value between classes. For example, with the default threshold value of 0.5, the predicted probability of ‘positive’ must be greater than 0.5 for the instance to be predicted as ‘positive’. The plot can be used to visualize the precision/recall tradeoff, for ROC curve analysis (true positive rate vs false positive rate), and for other types of curves.
- 11. Visualize cost curve.** Generates a plot that gives an explicit representation of the expected cost, as described by Drummond and Holte (2000). Options are greyed out if they do not apply to the specific set of results.

1.5.5.4. Clustering

1.5.5.4.1. Selecting a Clusterer

By now you will be familiar with the process of selecting and configuring objects. Clicking on the clustering scheme listed in the **Clusterer** box at the top of the window brings up a **GenericObjectEditor** dialog with which to choose a new clustering scheme.

1.5.5.4.2. Cluster Modes

The **Cluster mode** box is used to choose what to cluster and how to evaluate the results. The first three options are the same as for classification: **Use training set**, **Supplied test set** and **Percentage split** — the data is assigned to clusters instead of trying to predict a specific class. The fourth mode, **Classes to clusters evaluation**, compares how well the chosen clusters match up with a pre-assigned class in the data. The drop-down box below this option selects the class, just as in the **Classify** panel.

An additional option in the **Cluster mode** box, the **Store clusters for visualization** tick box, determines whether or not it will be possible to visualize the clusters once training is complete. When dealing with datasets that are so large that memory becomes a problem it may be helpful to disable this option.

1.5.5.4.3. Ignoring Attributes

Often, some attributes in the data should be ignored when clustering. The **Ignore attributes** button brings up a small window that allows you to select which attributes are ignored. Clicking on an attribute in the window highlights it, holding down the **SHIFT** key selects a range of consecutive attributes, and holding down **CTRL** toggles individual attributes on and off. To cancel the selection, back out with the **Cancel** button. To activate it, click the **Select** button. The next time clustering is invoked, the selected attributes are ignored.

1.5.5.4.4. Learning Clusters

The **Cluster** section, like the **Classify** section, has **Start/Stop** buttons, a result text area and a result list. These all behave just like their classification counterparts. Right-clicking an entry in the result list brings up a similar menu, except that it shows only two visualization options: **Visualize cluster assignments** and **Visualize tree**. The latter is grayed out when it is not applicable.

1.5.5.5. Associating

1.5.5.5.1. Setting Up

This panel contains schemes for learning association rules, and the learners are chosen and configured in the same way as the clusterers, filters, and classifiers in the other panels.

1.5.5.5.2. Learning Associations

Once appropriate parameters for the association rule learner have been set, click the **Start** button. When complete, right-clicking on an entry in the result list allows the results to be viewed or saved.

1.5.5.6. Selecting Attributes

1.5.5.6.1. Searching and Evaluating

Attribute selection involves searching through all possible combinations of attributes in the data to find which subset of attributes works best for prediction. To do this, two objects must be set up: an

attribute evaluator and a search method. The evaluator determines what method is used to assign a worth to each subset of attributes. The search method determines what style of search is performed.

The Attribute Selection Mode box has two options:

1. Use full training set. The worth of the attribute subset is determined using the full set of training data.

2. Cross-validation. The worth of the attribute subset is determined by a process of cross-validation. The Fold and Seed fields set the number of folds to use and the random seed used when shuffling the data. There is a drop-down box that can be used to specify which attribute to treat as the class.

1.5.5.6.2. Performing Selection

Clicking **Start** starts running the attribute selection process. When it is finished, the results are output into the result area, and an entry is added to the result list. Right-clicking on the result list gives several options. The first three, (**View in main window**, **View in separate window** and **Save result buffer**), are the same as for the classify panel. It is also possible to **Visualize reduced data**, or if you have used an attribute transformer such as Principal Components, **Visualize transformed data**.

1.5.5.7. Visualizing

WEKA's visualization section allows you to visualize 2D plots of the current relation.

1.5.5.7.1. The scatter plot matrix

When you select the Visualize panel, it shows a scatter plot matrix for all the attributes, color coded according to the currently selected class. It is possible to change the size of each individual 2D plot and the point size, and to randomly jitter the data (to uncover obscured points). It is also possible to change the attribute used to color the plots, to select only a subset of attributes for inclusion in the scatter plot matrix, and to sub sample the data. Note that changes will only come into effect once the Update button has been pressed.

1.5.5.7.2. Selecting an individual 2D scatter plot

When you click on a cell in the scatter plot matrix, this will bring up a separate window with a visualization of the scatter plot you selected.

1.5.5.7.3. Selecting Instances

A group of data points can be selected in four ways:

- a. **Select Instance.** Clicking on an individual data point brings up a window listing its attributes. If more than one point appears at the same location, more than one set of attributes is shown.
- b. **Rectangle.** You can create a rectangle, by dragging, that selects the points inside it.

- c. **Polygon.** You can build a free-form polygon that selects the points inside it. Left-click to add vertices to the polygon, right-click to complete it. The polygon will always be closed off by connecting the first point to the last.
- d. **Polyline.** You can build a polyline that distinguishes the points on one side from those on the other. Left-click to add vertices to the polyline, right-click to finish. The resulting shape is open (as opposed to a polygon, which is always closed). Once an area of the plot has been selected using Rectangle, Polygon or Polyline, it turns grey. At this point, clicking the Submit button removes all instances from the plot except those within the grey selection area. Clicking on the Clear button erases the selected area without affecting the graph. Once any points have been removed from the graph, the Submit button changes to a Reset button. This button undoes all previous removals and returns you to the original graph with all points included. Finally, clicking the Save button allows you to save the currently visible instances to a new ARFF file.

2. Examples

2.1. Practice WEKA with the classification example about Play Golf

Data format: the Datasets for WEKA are formatted according to the *arff* format. For this example you will use the file *weather.nominal.arff* as a training file to construct a classification model. Save the file in your workspace for example (C:\WEKA_Tutorial), and open it in a text processor to see an example of the arff format; note that the last attribute corresponds to the class.

Run WEKA in the Windows environment:

Find the WEKA directory in your machine (C:\Program Files\WEKA-3-4). Double click in the file "WEKA.jar"; Select the option "Simple CLI". Now you are ready to run WEKA using some commands in this window.

Probe the example with different classifiers, and compare the results obtained with each of the classifiers for example in terms of and number of examples correctly and incorrectly classified:

Decision Trees: In order to probe decision tree you will use the Id3 classifier. Type the following command:

```
java WEKA.classifiers.trees.Id3 -t PATH/weather.nominal.arff
```

(note that the option -t calls the training file according the PATH location of this file in your machine)

Support Vector Machines: In order to probe the SVM classifier, type the following command

```
java WEKA.classifiers.functions.SMO -t PATH/weather.nominal.arff
```

Neural Networks: In order to probe the NNs classifier, type the following command:

```
java WEKA.classifiers.functions.VotedPerceptron -t PATH/weather.nominal.arff
```

Naive Bayes: In order to probe the NB classifier, type the following command:

```
java WEKA.classifiers.bayes.NaiveBayes -t PATH/weather.nominal.arff
```

Save the classification model and then use it to classify new examples: You can save the classification model generated by each one of the above classifiers by using the option `-d` in the following way:

```
java WEKA.classifiers.TYPE.CLASSIFIER_NAME -t PATH/weather.nominal.arff -d  
PATH/modelname.model
```

You should generate a file that contains the model; this can be named for example in the form:

weather_Id3.model

weather_SVM.model

weather_NN.model

weather_NB.model

e.g. by

```
java WEKA.classifiers.trees.Id3 -t PATH/weather.nominal.arff -d PATH/weather_Id3.model
```

In order to use the stored model to classify new examples, use the file "test_weather.arff" (save this file in the same folder than weather.nominal.arff and *.model files). In this file you have two examples without classification. Then classify these examples using the models previously generated in the following way:

```
java WEKA.classifiers.~.classifier_name -T PATH/test_weather.arff -l  
PATH/modelname.model -p 0
```

In this case you use the options: `-T` that calls a test file (test_weather.arff); and `-l` that call the model file to be used. Compare the results obtained using the four models generated.

2.2. Classification of breast cancer examples

Download the file Breast_Cancer.arff that include a set of 699 cases, 9 attributes and the class attribute related to the type of cancer cell (in this dataset class 4 is equivalent to malignant cells and class 2 is equivalent to benign cells). This dataset is from the Wisconsin Breast Cancer Database (January 8, 1991). You can look for this and others examples of dataset in this link.

2.3. Classification of Gene expression data

Download the file ALLAML.arff ((Golub et al 1999)) gene expression data that include 72 examples, 7129 genes (attributes) and 2 classes "acute myeloid leukemia (AML)" and "acute lymphoblastic leukemia (ALL)". For more information you can read the gene list in the file ALLAML.gene_names.txt, and in the paper Golub et al 1999.

Classify the examples in this dataset (ALL or AML class) using the four classifiers mentioned in the exercise 1, and compare the results.

Interpretation: Go to PubMed and search the selected genes, do they have any biological meaning? Can you identify the unknown gene function? (Try using other bioinformatics tools)

3. Assignments

3.1 Become familiar with the *vowel* data set and use it to perform the following experiments:

- a. Remove the first three attributes as well as the class attribute.
- b. Cluster the data using the simple k-Means algorithm, with values of k from 1 to 12. What do you see?
- c. Now add the class attribute back in and repeat the clustering, comparing the clustering with the class. How well does the clustering appear to correlate with class? What might this mean?
- d. Choose several different classifiers and use them to classify the data. How does their performance compare with the clustering's "performance"? Is this something you might expect?
- e. Does adding back in any of the original first three attributes have any effect on either the clustering or the classification performance?

WORKING WITH DATA IN WEKA

Purpose:

- Attribute-Relation File Format (ARFF)
- Managing the data flow using WEKA

1 Preparation Before Lab

Attribute-Relation File Format (ARFF): An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files have two distinct sections. The first section is the **Header** information, which is followed the **Data** information.

The **Header** of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types.

The **Data** of the ARFF file looks like the following:

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
```

Lines that begin with a % are comments. The **@RELATION**, **@ATTRIBUTE** and **@DATA** declarations are case insensitive.

The ARFF Header Section

The ARFF Header section of the file contains the relation declaration and attribute declarations.

The @relation Declaration

The relation name is defined as the first line in the ARFF file. The format is:

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

where <relation-name> is a string. The string must be quoted if the name includes spaces.

The @attribute Declarations

Attribute declarations take the form of an ordered sequence of **@attribute** statements. Each attribute in the data set has its own **@attribute** statement which uniquely defines the name of

that attribute and its data type. The order the attributes are declared indicates the column position in the data section of the file. For example, if an attribute is the third one declared then Weka expects that all that attributes values will be found in the third comma delimited column.

The format for the **@attribute** statement is:

```
@attribute <attribute-name> <datatype>
```

where the <attribute-name> must start with an alphabetic character. If spaces are to be included in the name then the entire name must be quoted.

The <datatype> can be any of the four types currently supported by Weka:

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

where <nominal-specification> and <date-format> are defined below. The keywords **numeric**, **string** and **date** are case insensitive.

Numeric attributes: Numeric attributes can be real or integer numbers.

Nominal attributes: Nominal values are defined by providing an <nominal-specification> listing the possible values: {<nominalname1>, <nominal-name2>, <nominal-name3>, ...}

For example, the class value of the Iris dataset can be defined as follows:

```
@ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}
```

Values that contain spaces must be quoted.

String attributes: String attributes allow us to create attributes containing arbitrary textual values. This is very useful in text-mining applications, as we can create datasets with string attributes, then write Weka Filters to manipulate strings (like `StringToWordVectorFilter`). String attributes are declared as follows:

```
@ATTRIBUTE LCC string
```

Date attributes: Date attribute declarations take the form:

```
@attribute <name> date [<date-format>]
```

where <name> is the name for the attribute and <date-format> is an optional string specifying how date values should be parsed and printed. The default format string accepts the ISO-8601 combined date and time format:

```
"yyyy-MM-dd'T'HH:mm:ss".
```

Dates must be specified in the data section as the corresponding string representations of the date/time

ARFF Data Section

The ARFF Data section of the file contains the data declaration line and the actual instance lines.

The @data Declaration

The **@data** declaration is a single line denoting the start of the data segment in the file. The format is:

```
@data
```

The instance data

Each instance is represented on a single line, with carriage returns denoting the end of the instance. Attribute values for each instance are delimited by commas. They must appear in the order that they were declared in the header section (i.e. the data corresponding to the nth **@attribute** declaration is always the nth field of the attribute).

Missing values are represented by a single question mark, as in:

```
@data  
4.4,?,1.5,?,Iris-setosa
```

Values of string and nominal attributes are case sensitive, and any that contain space must be quoted, as follows:

```
@relation LCCvsLCSH  
@attribute LCC string  
@attribute LCSH string  
@data  
AG5, 'Encyclopedias and dictionaries.;Twentieth century.'  
AS262, 'Science -- Soviet Union -- History.'
```

Dates must be specified in the data section using the string representation specified in the attribute declaration. For example:

```
@RELATION Timestamps  
@ATTRIBUTE timestamp DATE "yyyy-MM-dd HH:mm:ss"  
@DATA  
"2001-04-03 12:12:12"
```

Sparse ARFF files

Sparse ARFF files are very similar to ARFF files, but data with value 0 are not be explicitly represented. Sparse ARFF files have the same header (i.e @relation and @attribute tags) but the data section is different. Instead of representing each value in order, like this:

```
@data
0, X, 0, Y, "class A"
0, 0, W, 0, "class B"
```

the non-zero attributes are explicitly identified by attribute number and their value stated, like this:

```
@data
{1 X, 3 Y, 4 "class A"}
{2 W, 4 "class B" }
```

Each instance is surrounded by curly braces, and the format for each entry is: <index> <space> <value> where index is the attribute index (starting from 0).

Note that the omitted values in a sparse instance are 0, they are not "missing" values! If a value is unknown, you must explicitly represent it with a question mark (?).

3. Weka GUI

3.1. The Command Line Interface

- One can use the command line interface of Weka either through a command prompt or through the SimpleCLI mode
- For example to fire up Weka and run J48 on a ARFF file present in the current working directory, the command is:

```
Java weka.classifiers.trees.J48 -t weather.arff
```

- Weka consists of a hierarchical package system. For example here J48 program is part of the trees package which further resides in the classifier package. Finally the weka package contains the classifiers package
- Each time the Java virtual machine executes J48, it creates an instance of this class by allocating memory for building and storing a decision tree classifier
- The -t option was used in the command line to communicate the name of the training file to the learning algorithm

Weka.filters

The `weka.filters` package is concerned with classes that transform datasets -- by removing or adding attributes, resampling the dataset, removing examples and so on. This package offers useful support for data preprocessing, which is an important step in machine learning.

All filters offer the options `-i` for specifying the input dataset, and `-o` for specifying the output dataset. If any of these parameters is not given, this specifies standard input resp. output for use within pipes. Other parameters are specific to each filter and can be found out via `-h`, as with any other class. The `weka.filters` package is organized into supervised and unsupervised filtering, both of which are again subdivided into instance and attribute filtering. We will discuss each of the four subsection separately.

3.1.1. Weka.filters.supervised

Classes below `weka.filters.supervised` in the class hierarchy are for supervised filtering, i.e. taking advantage of the class information. A class must be assigned via `-c`, for WEKA default behaviour use `-c last`.

3.1.1.1. Attribute

`Discretize` is used to discretize numeric attributes into nominal ones, based on the class information, via Fayyad & Irani's MDL method, or optionally with Kononeko's MDL method. At least some learning schemes or classifiers can only process nominal data, e.g. `rules.Prism`; in some cases discretization may also reduce learning time.

```
java weka.filters.supervised.attribute.Discretize -i data/iris.arff -o iris-nom.arff -c last  
java weka.filters.supervised.attribute.Discretize -i data/cpu.arff -o cpu-classvendor-nom.arff -c first
```

`NominalToBinary` encodes all nominal attributes into binary (two-valued) attributes, which can be used to transform the dataset into a purely numeric representation, e.g. for visualization via multi-dimensional scaling.

```
java weka.filters.supervised.attribute.NominalToBinary -i data/contact-lenses.arff -o contact-lenses-bin.arff -c last
```

Keep in mind that most classifiers in WEKA utilize transformation filters internally, e.g. `Logistic` and `SMO`, so you will usually not have to use these filters explicitly. However, if you plan to run a lot of experiments, pre-applying the filters yourself may improve runtime performance.

3.1.1.2.Instance

Resample creates a stratified subsample of the given dataset. This means that overall class distributions are approximately retained within the sample. A bias towards uniform class distribution can be specified via -B.

```
java weka.filters.supervised.instance.Resample -i data/soybean.arff -o soybean-5%.arff -c last -Z 5
java weka.filters.supervised.instance.Resample -i data/soybean.arff -o soybean-uniform-5%.arff -c last -Z
5 -B 1
```

StratifiedRemoveFolds creates stratified cross-validation folds of the given dataset. This means that per default the class distributions are approximately retained within each fold. The following example splits soybean.arff into stratified training and test datasets, the latter consisting of 25% (=1/4) of the data.

```
java weka.filters.supervised.instance.StratifiedRemoveFolds -i data/soybean.arff -o
soybean-train.arff \
-c last -N 4 -F 1 -V
java weka.filters.supervised.instance.StratifiedRemoveFolds -i data/soybean.arff -o
soybean-test.arff \
-c last -N 4 -F 1
```

3.1.2.Weka.filters.unsupervised

Classes below weka.filters.unsupervised in the class hierarchy are for unsupervised filtering, e.g. the non-stratified version of Resample. A class should not be assigned here.

3.1.2.1.Attribute

StringToWordVector transforms string attributes into a word vectors, i.e. creating one attribute for each word which either encodes presence or word count (-C) within the string. -W can be used to set an approximate limit on the number of words. When a class is assigned, the limit applies to each class separately. This filter is useful for textmining.

Obfuscate renames the dataset name, all attribute names and nominal attribute values. This is intended for exchanging sensitive datasets without giving away restricted information.

Remove is intended for explicit deletion of attributes from a dataset, e.g. for removing attributes of the iris dataset:

```
java weka.filters.unsupervised.attribute.Remove -R 1-2 -i data/iris.arff -o iris-simplified.arff
```

```
java weka.filters.unsupervised.attribute.Remove -V -R 3-last -i data/iris.arff -o iris-simplified.arff
```

3.1.2.2.Instance

Resample creates a non-stratified subsample of the given dataset, i.e. random sampling without regard to the class information. Otherwise it is equivalent to its supervised variant.

```
java weka.filters.unsupervised.instance.Resample -i data/soybean.arff -o soybean-5%.arff -Z 5
```

RemoveFolds creates cross-validation folds of the given dataset. The class distributions are not retained. The following example splits soybean.arff into training and test datasets, the latter consisting of 25% (=1/4) of the data.

```
java weka.filters.unsupervised.instance.RemoveFolds -i data/soybean.arff -o soybean-train.arff -c last -N 4 -F 1 -V
```

```
java weka.filters.unsupervised.instance.RemoveFolds -i data/soybean.arff -o soybean-test.arff -c last -N 4 -F 1
```

RemoveWithValues filters instances according to the value of an attribute.

```
java weka.filters.unsupervised.instance.RemoveWithValues -i data/soybean.arff \-o soybean-without_herbicide_injur
```

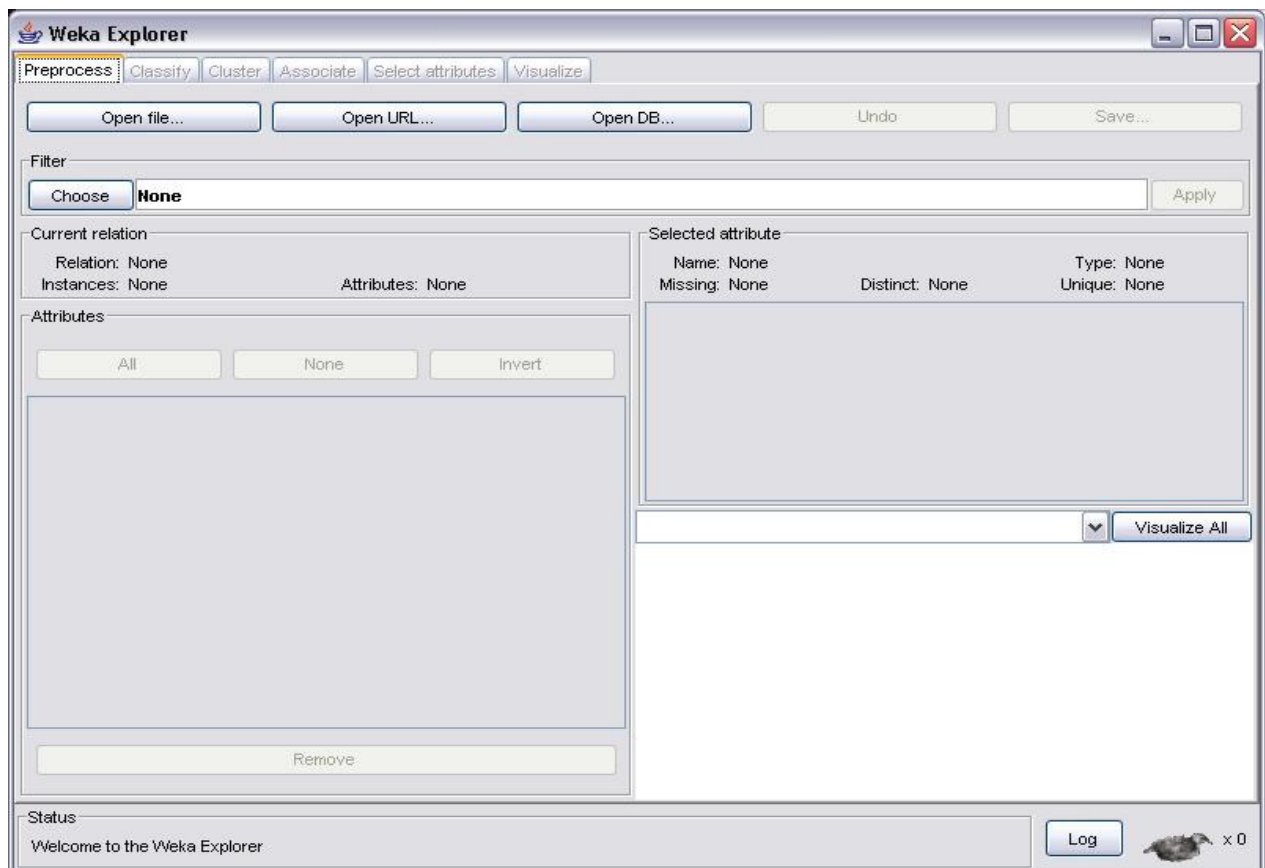
General options

Option	Function
-t <training file>	Specify training file
-T <test file>	Specify test file; if none, a cross-validation is performed on the training data
-c <class index>	Specify index of class attribute
-s <random number stxxi>	Specify random number seed for cross-validation
-x <number of folds>	Specify number of folds for cross-validation
-m<cost matrix file>	Specify file containing cost matrix

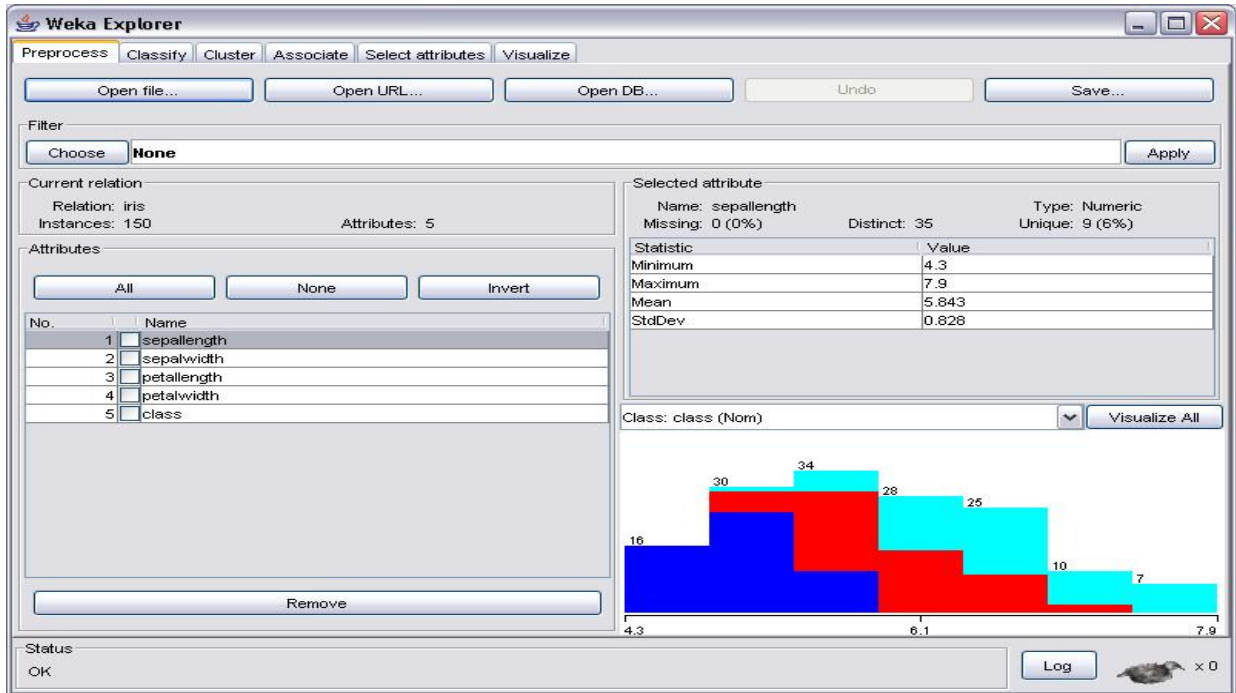
-d <output file>	Specify output file for model
-l <input file>	Specify input file for model
o	Output statistics only, not the classifier
-i	Output information retrieval statistics for two-class problems
-k	Output information-theoretical statistics
-p <attribute range>	Output predictions for test instances
-v	Output no statistics for training data
r	Output cumulative margin distribution
-z <class name>	Output source representation of classifier
-g	Output graph representation of classifier

3.2. Explorer

Start up Weka. You will have a choice between the Command Line Interface, the Experimenter, the Explorer and Knowledge flow. Initially, we'll stick with the Explorer. Once you click on that you'll see the main GUI.



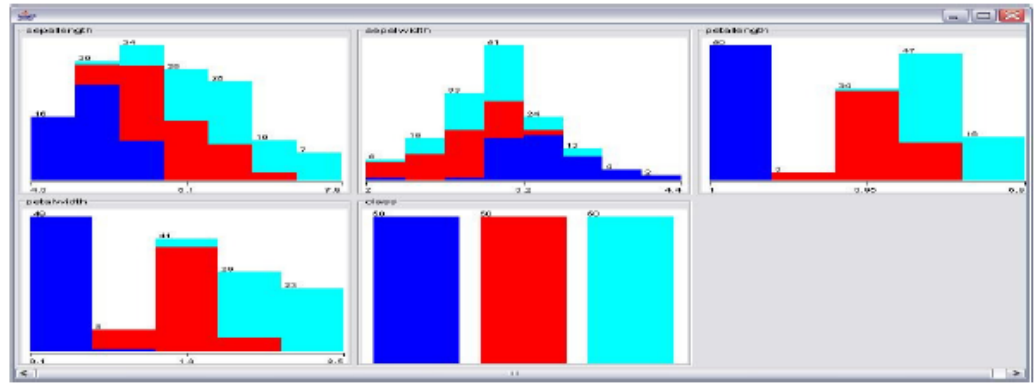
You now have a number of choices, but before you can work with any data, you'll have to load it into Weka. For now, we'll use one of the datasets that are included, but later on you'll have to get any file you'll use into the right format. Open a file from the data subcategory, for example the Iris data to find the following screen.



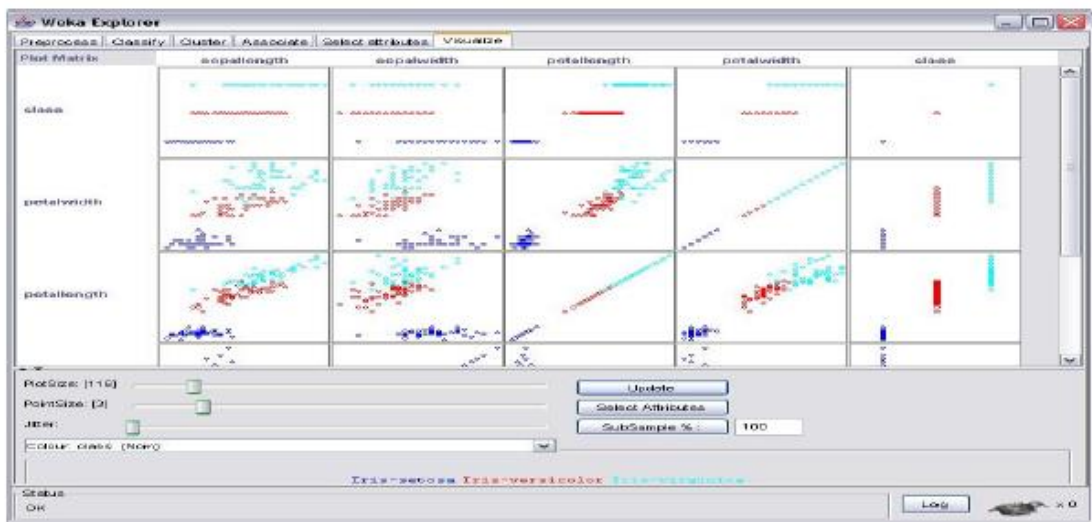
You'll notice that Weka now provides some information about the data, such as for example the number of instances, the number of attributes, and also some statistical information about the attributes one at a time. Figure out how to switch between attributes for which this statistical information is displayed.

Visualization

There are a number of ways in which you can use Weka to visualize your data. The main GUI will show a histogram for the attribute distributions for a single selected attribute at a time, by default this is the class attribute. Note that the individual colors indicate the individual classes (the Iris dataset has 3). If you move the mouse over the histogram, it will show you the ranges and how many samples fall in each range. The button VISUALIZE ALL will let you bring up a screen showing all distributions at once as in the picture below.



There is also a tab called VISUALIZE. Clicking on that will open the scatter plots for all attribute pairs:



From these scatter plots, we can infer a number of interesting things. For example, in the picture above we can see that in some examples the clusters (for now, think of clusters as collections of points that are physically close to each other on the screen) and the different colors correspond to each other such as for example in the plots for class/(any attribute) pairs and the petal width/petal length attribute pair, whereas for other pairs (sepal width/sepal length for example) it's much harder to separate the clusters by color.

By default, the colors indicate the different classes, in this case we used red and two shades of blue. Left clicking on any of the highlighted class names towards the bottom of the screenshot allows you to set your own color for the classes. Also, by default, the color is used in conjunction with the class attribute, but it can be useful to color the other attributes as well.

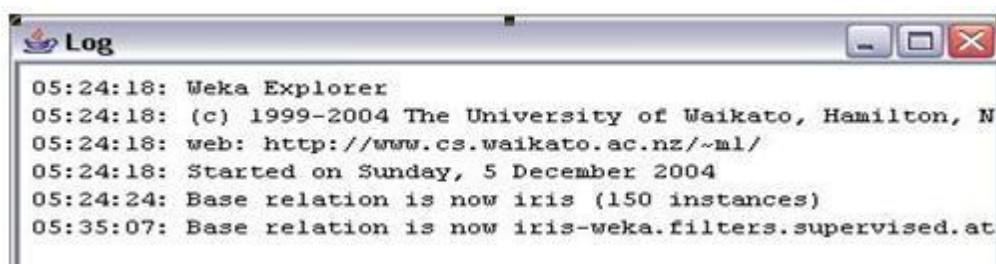
For example, changing the color to the fourth attribute by clicking on the arrow next to the bar that currently reads Color: class (Num) and selecting pedalwidth enables us to observe even more about the data, for example the fact that for the class/sepalwidth attribute pair, which range of attribute values (indicated by different color) tends to go along with which class.

Filters

There are also a number of filters available, which apply different criteria to select either objects (the rows in your data matrix) or attributes (the columns in your data matrix). This allows you to discard parts of your matrix without having to manipulate your original data file. For example, you can look at subsets of attributes, discard the first 20 rows, normalize or discretize attributes and so on. To apply a filter, you first have to select which type of filter you'd like by clicking on the CHOOSE button right underneath Filter in your main GUI. Double clicking on the FILTER folder that appeared will expand the window to show two folders named supervised and unsupervised, both of which you can expand again. Both unsupervised and supervised filters can be applied to objects and attributes. Once you have chosen a filter, the selected option will show up in the bar next to FILTER, but at this stage, nothing has happened to your data yet. You then have to press apply to actually filter your data. There is also a SAVE button which allows you to save any changes you made to your data. Make sure you don't overwrite your original data file!

The log file

The log file is used to keep track of what you did. Clicking on LOG in your main GUI will bring up another window which will show exactly what you did, in this case it shows that we loaded the Iris data set and applied a filter.

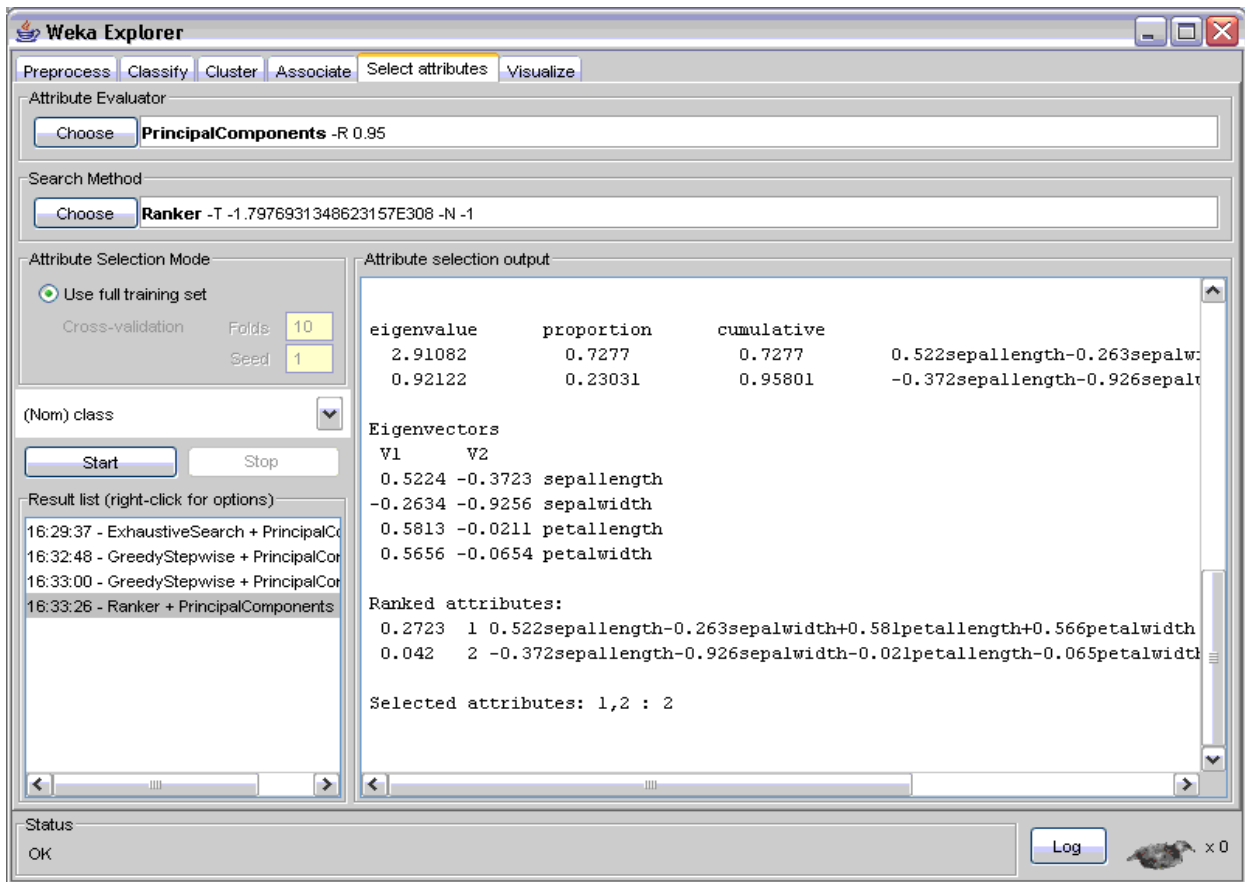


```
05:24:18: Weka Explorer
05:24:18: (c) 1999-2004 The University of Waikato, Hamilton, N
05:24:18: web: http://www.cs.waikato.ac.nz/~ml/
05:24:18: Started on Sunday, 5 December 2004
05:24:24: Base relation is now iris (150 instances)
05:35:07: Base relation is now iris-weka.filters.supervised.at
```

Selecting Attributes

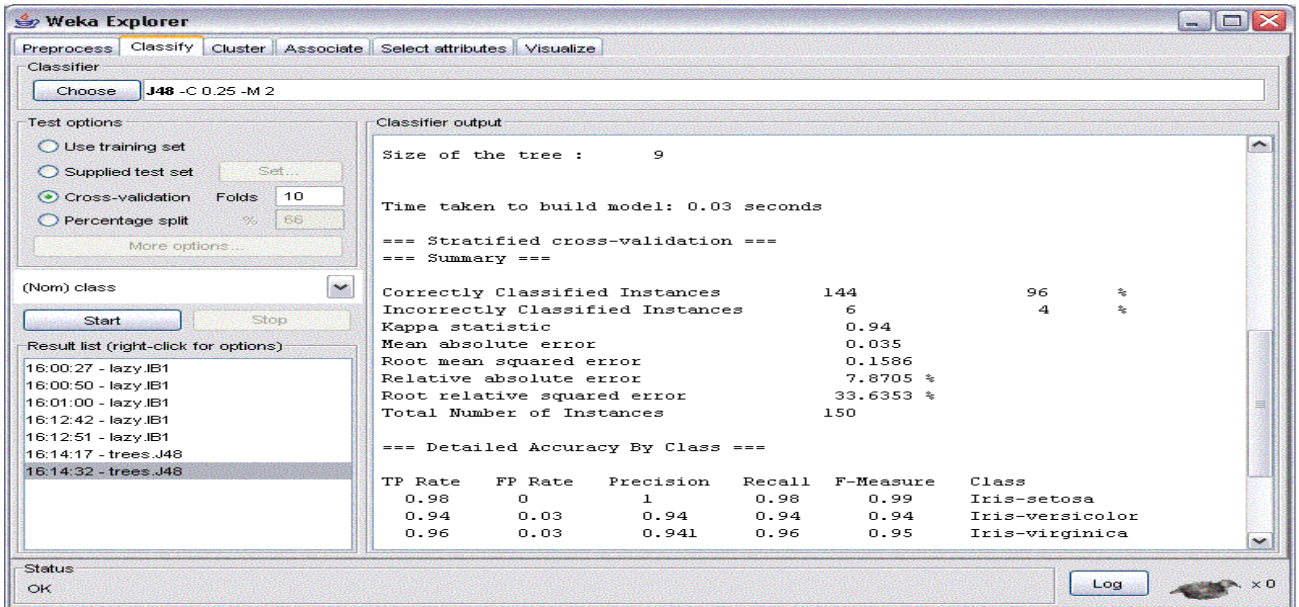
Weka also provides techniques to discard irrelevant attributes or reduce the dimensionality of your dataset. After loading a dataset, click on the select attributes tag to open a GUI which will allow you to choose both the evaluation method (such as Principal Components Analysis for example) and the search method (f. ex. greedy or exhaustive search). Depending on the chosen combination, the actual time spend on selecting attributes can vary substantially and can also be very long, even for small datasets such as the Iris data with only five features (including the class attribute) for each of the 150 samples. The picture below shows the results for a sample application. It is also important to note that not all evaluation/search method combinations are valid, watch out for the error message in the Status bar. There's also a problem using Discretize while in the preprocessing mode, which leads to false results. If you need to use this filter, you can work around this by using the FilteredClassifier option in the classify menu.

Weka also provides techniques to discard irrelevant attributes or reduce the dimensionality of your dataset. After loading a dataset, click on the select attributes tag to open a GUI which will allow you to choose both the evaluation method (such as Principal Components Analysis for example) and the search method (for. ex. greedy or exhaustive search). Depending on the chosen combination, the actual time spend on selecting attributes can vary substantially and can also be very long, even for small datasets such as the Iris data with only five features (including the class attribute) for each of the 150 samples. The picture below shows the results for a sample application. It is also important to note that not all evaluation/search method combinations are valid, watch out for the error message in the Status bar. There's also a problem using Discretize while in the preprocessing mode, which leads to false results. If you need to use this filter, you can work around this by using the FilteredClassifier option in the classify menu.



Classification

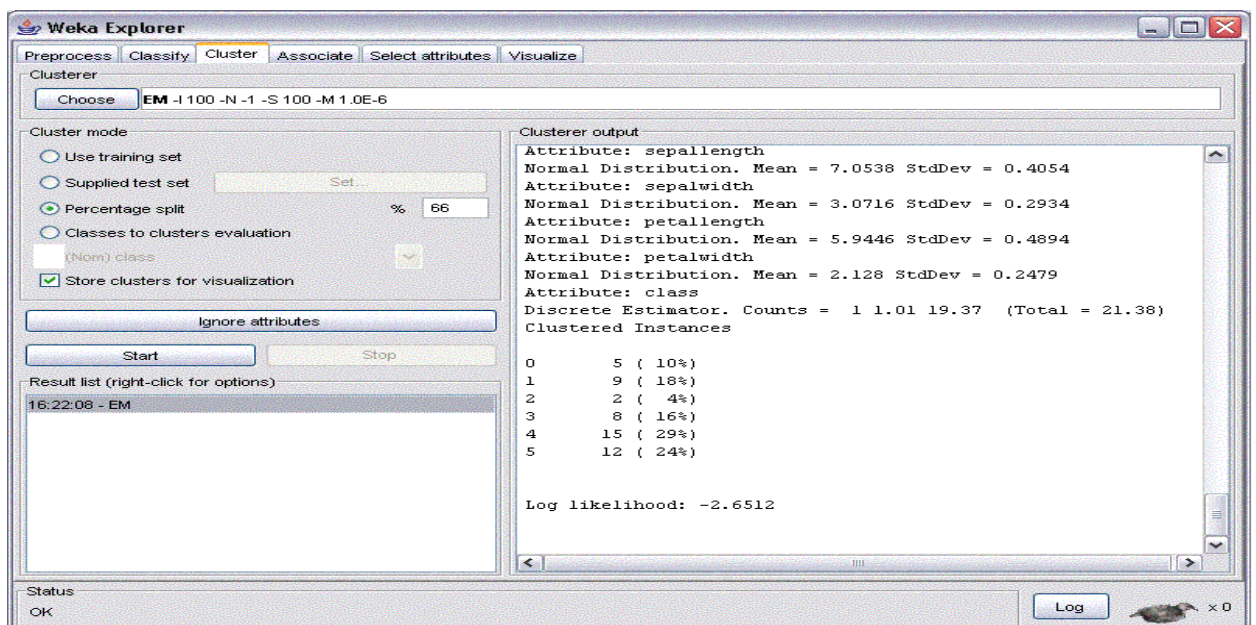
Clicking on the classifier tab after loading a dataset into Weka and selecting the choose tab will bring up a menu with a number of choices for the classifier that is to be applied to the dataset. Note that you have 4 options on how to test the model you're building: Using the test set, a training set (you will need to specify the location of the training set in this case), cross validation and a percentage. The achieved accuracy of your model will vary, depending on the option you select. One pitfall to avoid is to select the training set as a test set, as that will result in an underestimate of the error rate. The resulting model, with a lot of additional information will be displayed after you click on start. What exactly is contained in the output can be determined under options. A sample output for applying the J48 decision tree algorithm to the Iris dataset is shown in the Figure below.



One of the things to watch out for is that the confusion matrix is displayed, as this gives a lot more information than just the prediction accuracy. Other useful things are the options showing up when right clicking the results list on the bottom right. For example, this is where you can load and save the models you built, as well as save the results page. Another fact to keep in mind is that Weka gives hints on how to achieve the same result from the command line: look at what is displayed next to the Choose button and how it changes with the options that you select. This information can also be found towards the top of your results page.

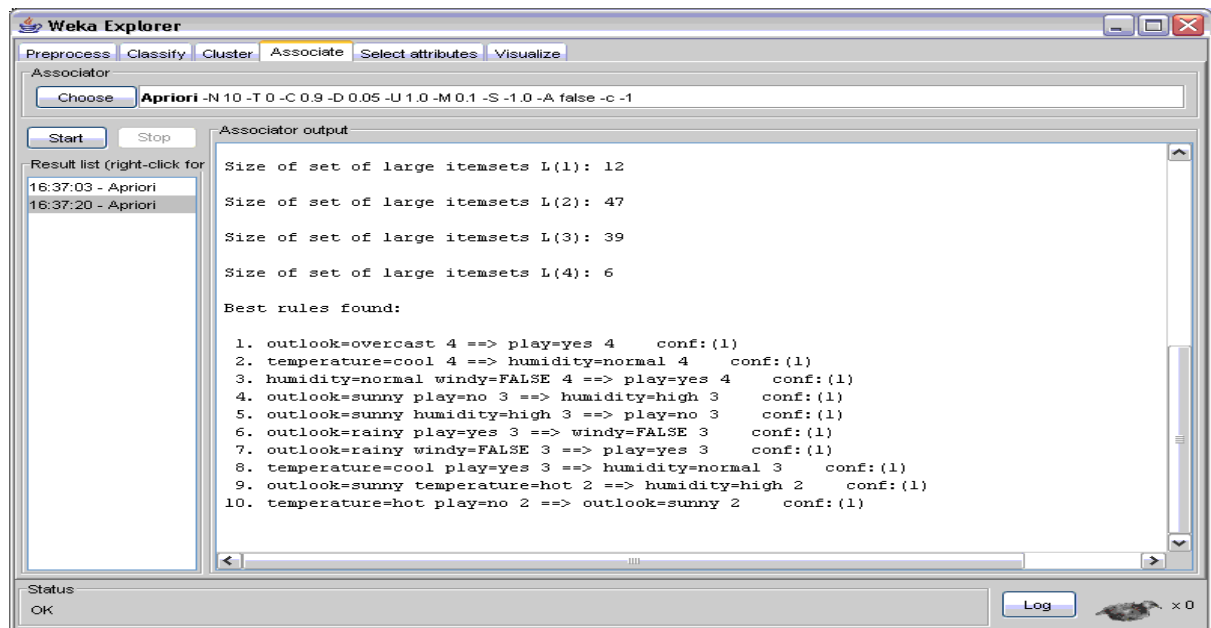
Clustering

The clustering option is very similar to the classification described above, with a few differences regarding the options you select. For instance, there is an easy way to discard undesired attributes.



Association Rules

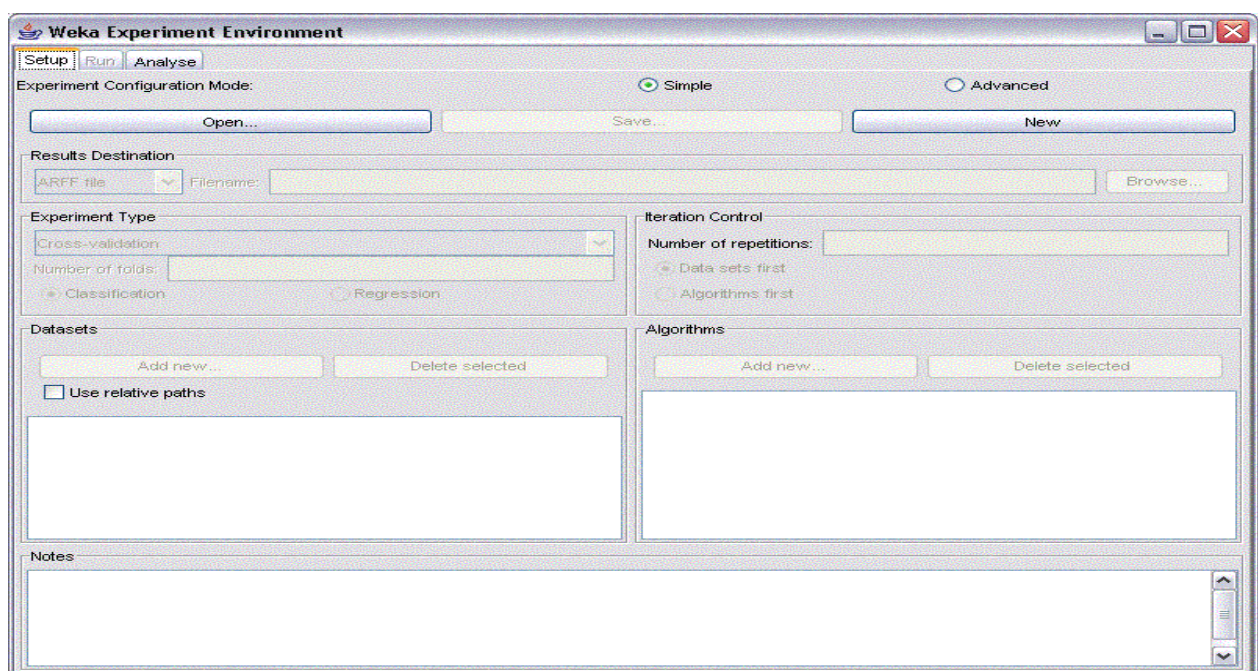
Weka also provides three algorithms to extract association rules from non-numerical data as shown in the picture below.



3.3. Experimenter

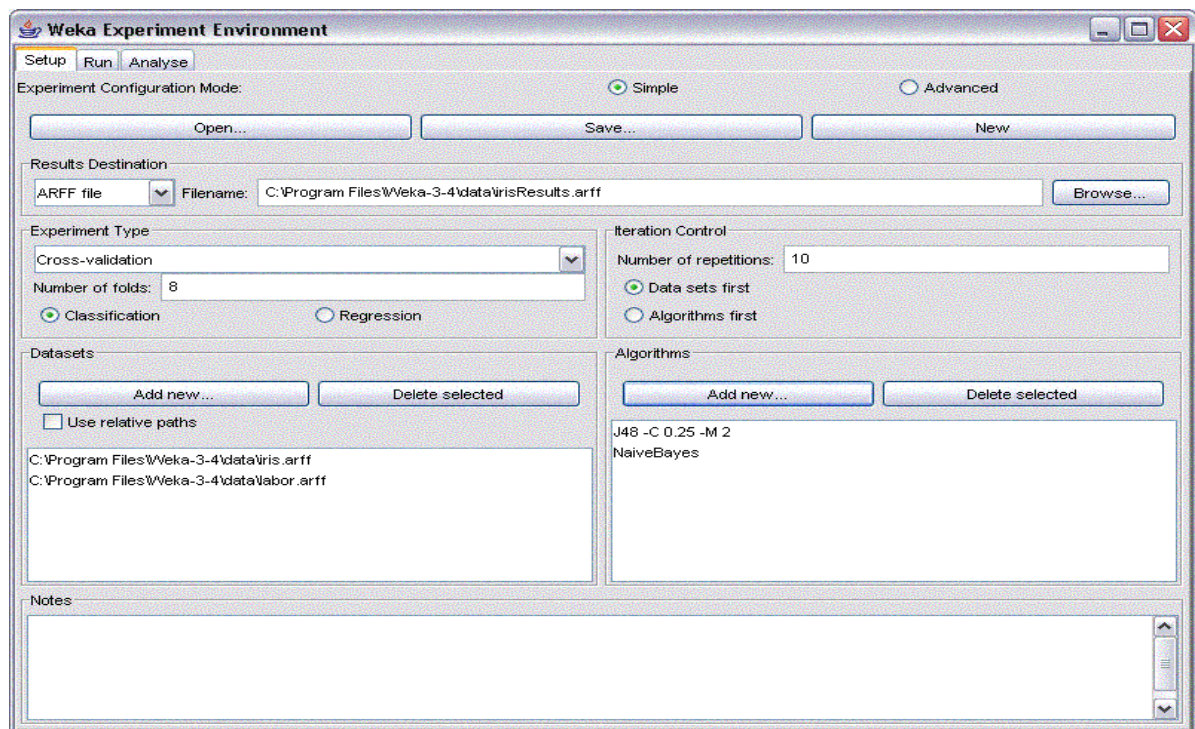
The experimenter, which can be run from both the command line and a GUI, is a tool that allows you to perform more than one experiment at a time, maybe applying different techniques to a datasets, or the same technique repeatedly with different parameters. The Weka homepage provides a link to a tutorial for an earlier version of the Experimenter, which can be downloaded from here.

If you choose the experimenter after starting Weka, you get the following screen.

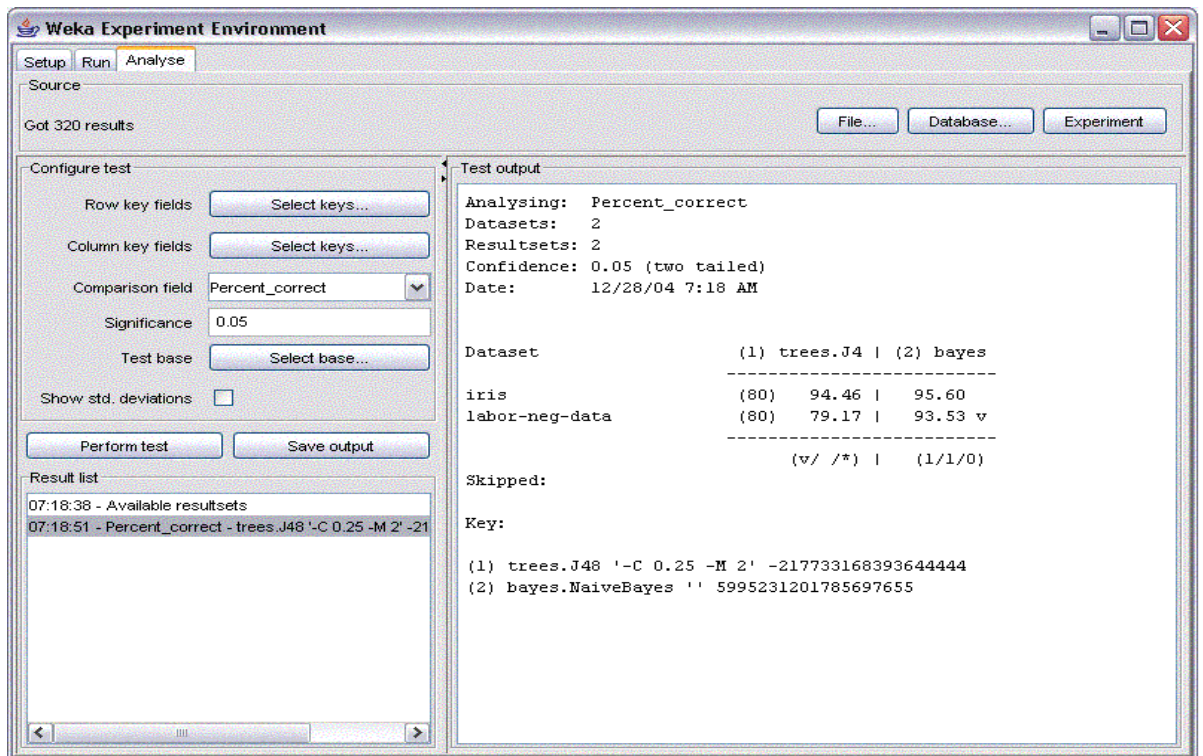


After selecting new, which initializes a new experiment with default parameters, you can select where you want to store the results of your experiment by using browse (there are a number of choices available for the format of your results file). You can then change the default parameters if desired (watch out for the option of selecting classification or regression). For example, you can add more datasets, delete the ones you already selected as well as add and delete algorithms applied to your selected datasets. You can also the type of experiment (cross validation or a percentage split for the training and test set).

The following picture shows the setup for a n 8 fold cross validation, applying a decision tree and Naive Bayes to the iris and labor dataset that are included in the Weka Package. The results are to be stored in an ARFF file called MyResults.arff in the specified subfolder



After running your experiment by selecting Start from the Run tab, your results will be stored in the specified Results file if the run was successful. You then need to load this file into Weka from the Analysis pane to see your results. The picture below shows the Analysis pane after loading the results file for the experiment set up above.



3.4. Knowledge Flow

The knowledge flow is an alternative interface to the functionality provided by the Weka data mining package. WEKA components are selected from a tool bar, positioned a layout canvas, and connected into a directed graph to model a complete system that processes and analyzes data.

Components available in the KnowledgeFlow:

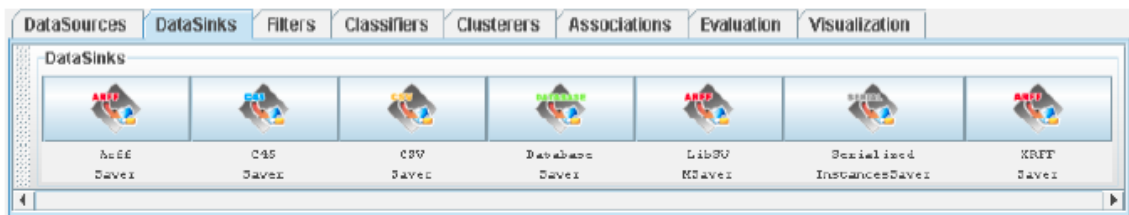
3.4.1. DataSources



- used to indicate where data is coming from
- supports various file types and sources
- configurable for
 - o file name of data source
 - o dataset or instance (incremental) loading

All of WEKA's loaders are available.

3.4.2. DataSinks



- used to indicate where data is going to be written
- supports various file types and sources
- configurable for
 - o file name of data source

All of WEKA's savers are available.

3.4.3.

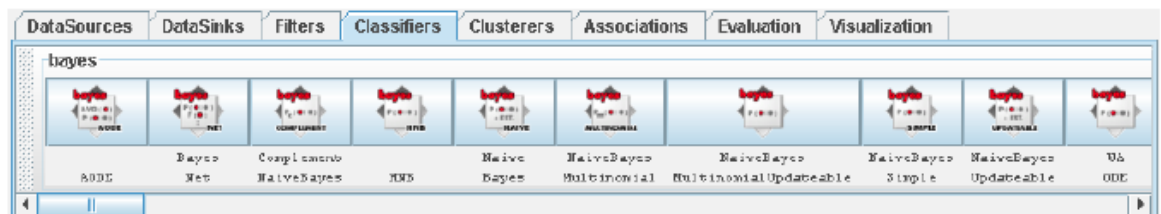
Filters



- used to preprocess data prior to classification or learning
- supports both supervised and unsupervised filters
- configurable depending on filter type

All of WEKA's filters are available.

3.4.4. Classifiers



- supports all classification algorithms presented in the textbook
- parameters are configurable depending on classification algorithm

All of WEKA's classifiers are available.

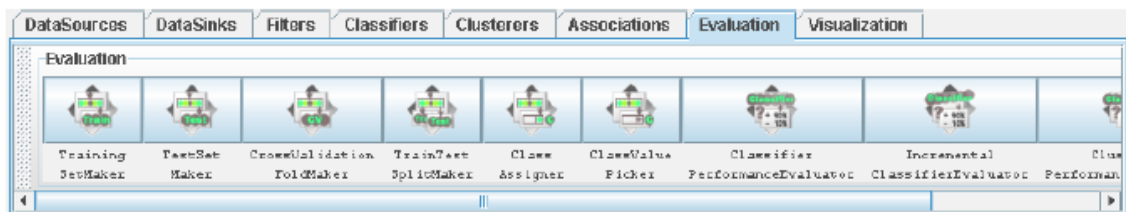
3.4.5. Clusterers



- supports all clustering algorithms presented in the textbook
- parameters are configurable depending on clustering algorithm

All of WEKA's clusterers are available

3.4.6. Evaluation



- used to configure both inputs to and outputs from algorithms
- supports various algorithm performance evaluators
- output format fairly "standardized"
 - TrainingSetMaker - make a data set into a training set
 - TestSetMaker - make a data set into a test set.
 - CrossValidationFoldMaker - split any data set, training set or test set into folds.
 - TrainTestSplitMaker - split any data set, training set or test set into a training set and a test set.
 - ClassAssigner - assign a column to be the class for any data set, training set or test set.
 - ClassValuePicker - choose a class value to be considered as the "positive" class. This is useful when generating data for ROC style curves (see ModelPerformanceChart below and example 4.2).
 - ClassifierPerformanceEvaluator - evaluate the performance of batch trained/tested classifiers.
 - IncrementalClassifierEvaluator - evaluate the performance of incrementally trained classifiers.
 - ClustererPerformanceEvaluator - evaluate the performance of batch trained/tested clusters.

- PredictionAppender - append classifier predictions to a test set. For discrete class problems, can either append predicted class labels or probability distributions.

3.4.7. Visualization

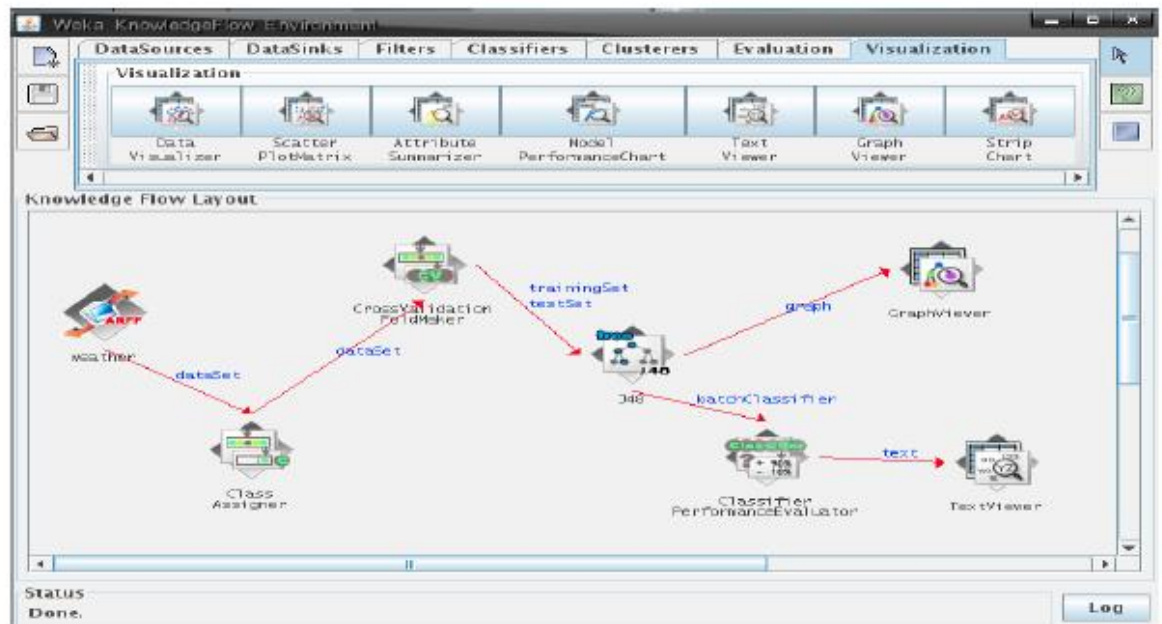


- used to visually display outputs
- supports performance and summaries
- comparable to options from Explorer interface
 - DataVisualizer - component that can pop up a panel for visualizing data in a single large 2D scatter plot.
 - ScatterPlotMatrix - component that can pop up a panel containing a matrix of small scatter plots (clicking on a small plot pops up a large scatter plot).
 - AttributeSummarizer - component that can pop up a panel containing a matrix of histogram plots - one for each of the attributes in the input data.
 - ModelPerformanceChart - component that can pop up a panel for visualizing threshold curves.
 - TextViewer - component for showing textual data. Can show data sets, classification performance statistics
 - GraphViewer - component that can pop up a panel for visualizing tree based models.
 - StripChart - component that can pop up a panel that displays a scrolling plot of data (used for viewing the online performance of incremental classifiers).

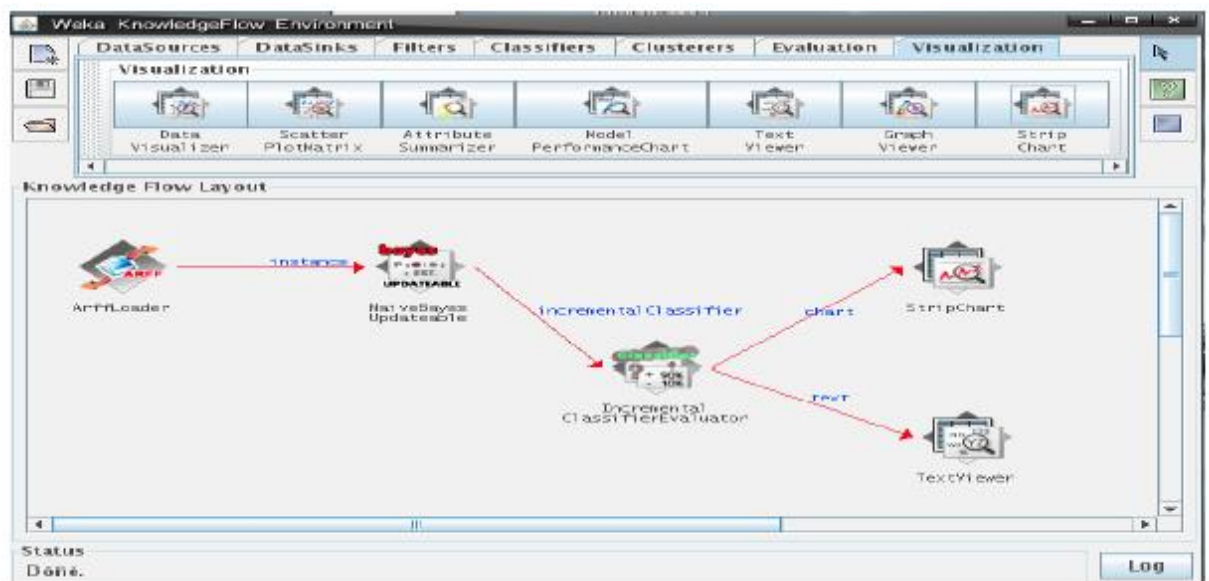
Example1: Decision Tree Classifier

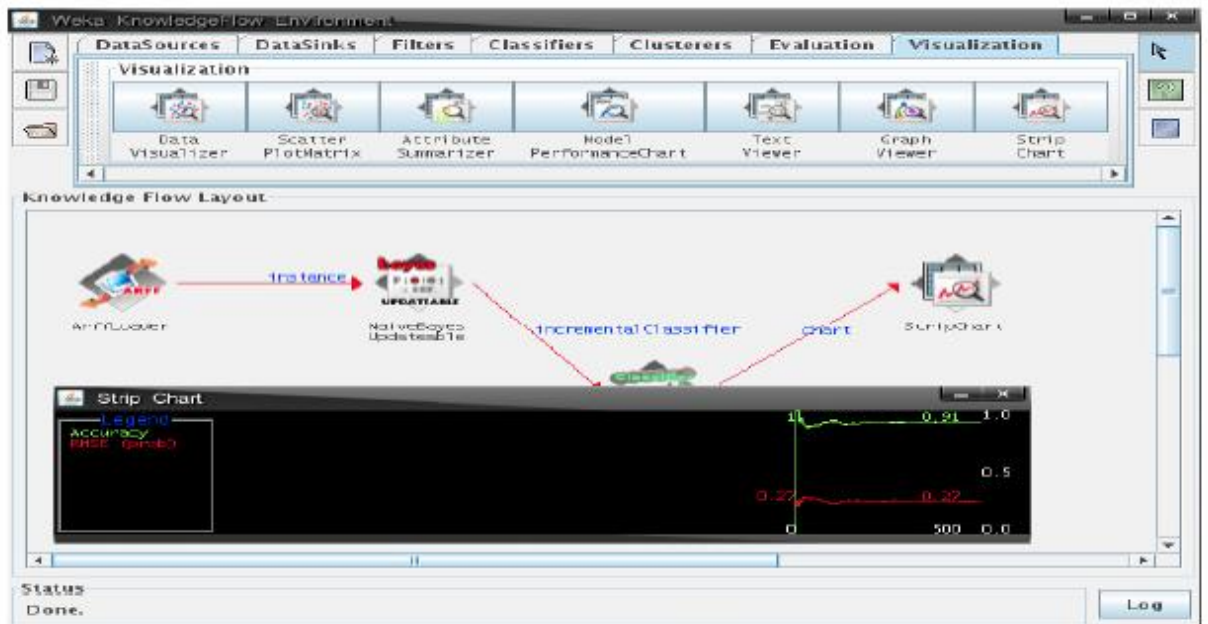
1. Specify a data source
2. Specify which attribute is the class
3. Specify cross validation
4. Specify decision tree algorithm

5. Specify evaluation
6. Specify evaluation output
7. To allow viewing of decision trees per fold
8. Run experiments



Example2: Incremental Learning





Applications

Weka was originally developed for the purpose of processing agricultural data, motivated by the importance of this application area in New Zealand. However, the machine learning methods and data engineering capability item bodies have grown so quickly, and so radically, that the workbench is now commonly used in all forms of data mining applications—from bioinformatics to competition datasets issued by major conferences such as Knowledge Discovery in Databases.

They worked on:

- predicting the internal bruising sustained by different varieties of apple as they make their way through a packing-house on a conveyor belt;
- predicting, in real time, the quality of a mushroom from a photograph in order to provide automatic grading;
- classifying kiwifruit vines into twelve classes, based on visible-NIR spectra, in order to determine which of twelve pre-harvest fruit management treatments has been applied to the vines;

Weka has been used extensively in the field of bioinformatics. Published studies include automated protein annotation, probe selection for gene expression arrays , plant genotype discrimination , and classifying gene expression profiles and extracting rules from them. Text mining is another major field of application, and the workbench has been used to

automatically extract key phrases from text, and for document categorization, and word sense disambiguation. There are many projects that extend or wrap WEKA in some fashion. There are 46 such projects listed on the Related Projects web page of the WEKA site³. Some of these include:

- *Systems for natural language processing.* There are a number of tools that use WEKA for natural language processing: GATE is a NLP workbench ; Balie performs language identification, tokenization, sentence boundary detection and named-entity recognition; Senseval-2 is a system for word sense disambiguation; Kea is a system for automatic keyphrase extraction .
- *Knowledge discovery in biology.* Several tools using or based on WEKA have been developed to aid data analysis in biological applications: BioWEKA is an extension to WEKA for tasks in biology, bioinformatics, and biochemistry; the Epitopes Toolkit (EpiT) is a platform based on WEKA for developing epitope prediction tools; maxdView and Mayday provide visualization and analysis of microarray data.
- *Distributed and parallel data mining.* There are a number of projects that have extended WEKA for distributed data mining; Weka-Parallel provides a distributed cross-validation facility; GridWeka provides distributed scoring and testing as well as cross validation; FAEHIM andWeka4WS make WEKA available as a web service.
- *Open-source data mining systems.* Several well known open-source data mining systems provide plugins to allow access to WEKA's algorithms. The Konstanz Information Miner (KNIME) and RapidMiner are two such systems. The R statistical computing environment also provides an interface toWEKA through the Rweka package.
- *Scientific workflow environment.* The Kepler Weka project integrates all the functionality of WEKA into the Kepler open-source scientific workflow platform.

Many future applications will be developed in an online setting. Recent work on data streams has enabled machine learning algorithms to be used in situations where a potentially infinite source of data is available. These are common in manufacturing industries with 24/7 processing. The challenge is to develop models that constantly monitor data in order to detect changes from the steady state. Such changes may indicate failure in the process, providing operators with warning signals that equipment needs re-calibrating or replacing.

NAIVE-BAYES CLASSIFICATION ALGORITHM

1. Introduction to Bayesian Classification

The Bayesian Classification represents a supervised learning method as well as a statistical method for classification. Assumes an underlying probabilistic model and it allows us to capture uncertainty about the model in a principled way by determining probabilities of the outcomes. It can solve diagnostic and predictive problems. This Classification is named after Thomas Bayes (1702-1761), who proposed the Bayes Theorem.

Bayesian classification provides practical learning algorithms and prior knowledge and observed data can be combined. Bayesian Classification provides a useful perspective for understanding and evaluating many learning algorithms. It calculates explicit probabilities for hypothesis and it is robust to noise in input data.

Uses of Naive Bayes classification:

1. Naive Bayes text classification

(<http://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html>)

The Bayesian classification is used as a probabilistic learning method (Naive Bayes text classification). Naive Bayes classifiers are among the most successful known algorithms for learning to classify text documents.

2. Spam filtering (http://en.wikipedia.org/wiki/Bayesian_spam_filtering)

Spam filtering is the best known use of Naive Bayesian text classification. It makes use of a naive Bayes classifier to identify spam e-mail. Bayesian spam filtering has become a popular mechanism to distinguish illegitimate spam email from legitimate email (sometimes called "ham" or "bacn").[4] Many modern mail clients implement Bayesian spam filtering. Users can also install separate email filtering programs. Server-side email filters, such as DSPAM, SpamAssassin, SpamBayes, Bogofilter and ASSP, make use of Bayesian spam filtering techniques, and the functionality is sometimes embedded within mail server software itself.

3. Hybrid Recommender System Using Naive Bayes Classifier and Collaborative Filtering

(<http://eprints.ecs.soton.ac.uk/18483/>)

Recommender Systems apply machine learning and data mining techniques for filtering unseen information and can predict whether a user would like a given resource. It is proposed a unique switching hybrid recommendation approach by combining a Naïve Bayes classification approach with the collaborative filtering. Experimental results on two different

data sets, show that the proposed algorithm is scalable and provide better performance in terms of accuracy and coverage—than other algorithms while at the same time eliminates some recorded problems with the recommender systems.

4. Online applications (<http://www.convo.co.uk/x02/>)

This online application has been set up as a simple example of supervised machine learning and affective computing. Using a training set of examples which reflect nice, nasty or neutral sentiments, we're training Ditto to distinguish between them. Simple Emotion Modelling, combines a statistically based classifier with a dynamical model. The Naive Bayes classifier employs single words and word pairs as features. It allocates user utterances into nice, nasty and neutral classes, labelled +1, -1 and 0 respectively. This numerical output drives a simple first-order dynamical system, whose state represents the simulated emotional state of the experiment's personification, Ditto the donkey.

1.1. Independence

1.1.1. Example:

Suppose there are two events:

- M: Manuela teaches the class (otherwise it's Andrew)
- S: It is sunny

“The sunshine levels do not depend on and do not influence who is teaching.”

1.1.2 Theory:

From $P(S | M) = P(S)$, the rules of probability imply:

- $P(\sim S | M) = P(\sim S)$
- $P(M | S) = P(M)$
- $P(M \wedge S) = P(M) P(S)$
- $P(\sim M \wedge S) = P(\sim M) P(S)$
- $P(M \wedge \sim S) = P(M) P(\sim S)$
- $P(\sim M \wedge \sim S) = P(\sim M) P(\sim S)$

1.2.3. Theory applied on previous example:

“The sunshine levels do not depend on and do not influence who is teaching.” can be specified

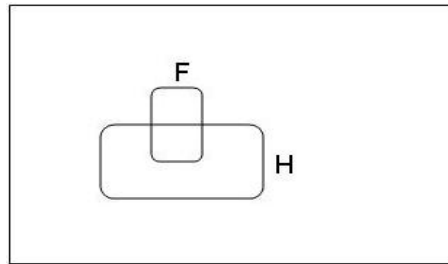
very simply:

$$P(S | M) = P(S)$$

“Two events A and B are statistically independent if the probability of A is the same value when B occurs, when B does not occur or when nothing is known about the occurrence of B”

1.2. Conditional Probability

1.2.1. Simple Example:



H = “Have a headache”

F = “Coming down with Flu”

$$P(H) = 1/10$$

$$P(F) = 1/40$$

$$P(H|F) = 1/2$$

“Headaches are rare and flu is rarer, but if you’re coming down with ‘flu there’s a 50-50 chance

you will have a headache.”

$P(H|F)$ = Fraction of flu-inflicted worlds in which you have a headache =

$$= \frac{\text{\#worlds with flu and headache}}{\text{\#worlds with flu}} = \frac{\text{Area of “H and F” region}}{\text{Area of “F” region}} = \frac{P(H \wedge F)}{P(F)}$$

1.2.2. Theory:

$P(A|B)$ = Fraction of worlds in which B is true that also have A true

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

Corollary:

$$P(A \wedge B) = P(A|B) P(B)$$

$$P(A|B) + P(\neg A|B) = 1$$

$$\sum_{k=1}^n P(A = v_k | B) = 1$$

$$P(A|B) + P(\neg A|B) = 1$$

1.2.3. Detailed Example

- M : Manuela teaches the class
- S : It is sunny
- L : The lecturer arrives slightly late.

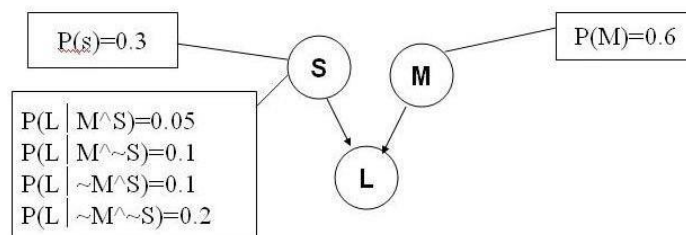
Assume both lecturers are sometimes delayed by bad weather. Andrew is more likely to arrive late than Manuela.

Let us begin with writing down the knowledge:

$$P(S \perp\!\!\!\perp M) = P(S), P(S) = 0.3, P(M) = 0.6$$

Lateness is not independent of the weather and is not independent of the lecturer. Therefore Lateness is *dependant* on both weather and lecturer

$P(S \mid M) = P(S)$	$P(L \mid M \wedge S) = 0.05$
$P(S) = 0.3$	$P(L \mid M \wedge \sim S) = 0.1$
$P(M) = 0.6$	$P(L \mid \sim M \wedge S) = 0.1$
	$P(L \mid \sim M \wedge \sim S) = 0.2$



Read the absence of an arrow between S and M to mean “it would not help me predict M if I knew the value of S”	Read the two arrows into L to mean that if I want to know the value of L it may help me to know M and to know S.
--	--

1.3. Conditional Independence

1.3.1. Example:

Suppose we have these three events:

- M : Lecture taught by Manuela
- L : Lecturer arrives late
- R : Lecture concerns robots

Suppose:

Andrew has a higher chance of being late than Manuela.

Andrew has a higher chance of giving robotics lectures.

Once you know who the lecturer is, then whether they arrive late doesn't affect whether the lecture concerns robots.

1.3.2. Theory:

R and L are conditionally independent given M if for all x,y,z in {T,F}:

$$P(R=x \mid M=y \wedge L=z) = P(R=x \mid M=y)$$

More generally:

Let S1 and S2 and S3 be sets of variables.

Set-of-variables S1 and set-of-variables S2 are conditionally independent given S3 if for all assignments of values to the variables in the sets, $P(S1's \text{ assignments} \mid S2's \text{ assignments} \ \& \ S3's \text{ assignments}) = P(S1's \text{ assignments} \mid S3's \text{ assignments})$

$$P(A|B) = P(A \wedge B)/P(B)$$

Therefore $P(A \wedge B) = P(A|B).P(B)$ – also known as Chain Rule

$$\text{Also } P(A \wedge B) = P(B|A).P(A)$$

$$\text{Therefore } P(A|B) = P(B|A).P(A)/P(B)$$

$$P(A,B|C) = P(A \wedge B \wedge C)/P(C)$$

$$= P(A|B,C).P(B|C)/P(C) \text{ – applying chain rule}$$

$$= P(A|B,C).P(B|C)$$

$$= P(A|C).P(B|C), \text{ If A and B are conditionally independent given C.}$$

This can be extended for n values as $P(A1,A2...An|C) = P(A1|C).P(A2|C)...P(An|C)$ if $A1,A2...An$ are conditionally independent given C.

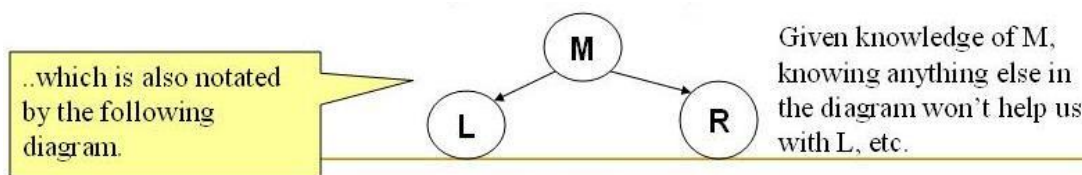
1.3.3. Theory applied on previous example:

For the previous example, we can use the following notations:

$$P(R \mid M,L) = P(R \mid M) \text{ and } P(R \mid \sim M,L) = P(R \mid \sim M)$$

We express this in the following way:

“R and L are conditionally independent given M”



2. Bayes Theorem

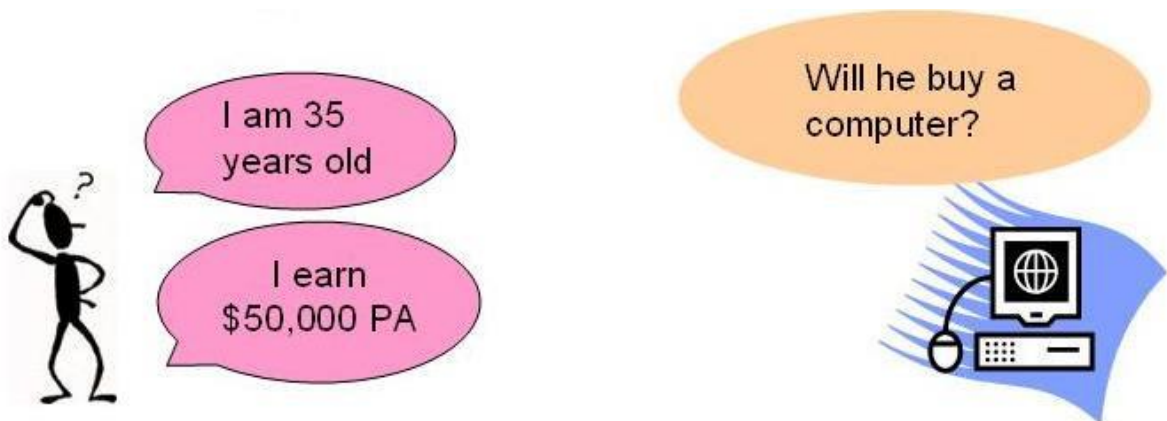
Bayesian reasoning is applied to decision making and inferential statistics that deals with probability inference. It is used the knowledge of prior events to predict future events.

Example: Predicting the color of marbles in a basket

2.1. Example:

rec	Age	Income	Student	Credit_rating	Buys_computer
r1	<=30	High	No	Fair	No
r2	<=30	High	No	Excellent	No
r3	31...40	High	No	Fair	Yes
r4	>40	Medium	No	Fair	Yes
r5	>40	Low	Yes	Fair	Yes
r6	>40	Low	Yes	Excellent	No
r7	31...40	Low	Yes	Excellent	Yes
r8	<=30	Medium	No	Fair	No
r9	<=30	Low	Yes	Fair	Yes
r10	>40	Medium	Yes	Fair	Yes
r11	<=30	Medium	Yes	Excellent	Yes
r12	31...40	Medium	No	Excellent	Yes
r13	31...40	High	Yes	Fair	Yes
r14	>40	Medium	No	Excellent	No

Table1: Data table



2.2. Theory:

The Bayes Theorem:

$$P(h/D) = \frac{P(D/h) P(h)}{P(D)}$$

P(h) : Prior probability of hypothesis h

P(D) : Prior probability of training data D

P(h/D) : Probability of h given D

P(D/h) : Probability of D given h

2.3. Theory applied on previous example:

- D : 35 year old customer with an income of \$50,000 PA
- h : Hypothesis that our customer will buy our computer

$P(h/D)$: Probability that customer D will buy our computer given that we know his age and income

$P(h)$: Probability that any customer will buy our computer regardless of age (Prior Probability)

$P(D/h)$: Probability that the customer is 35 yrs old and earns \$50,000, given that he has bought our computer (Posterior Probability)

$P(D)$: Probability that a person from our set of customers is 35 yrs old and earns \$50,000

2.4. Maximum A Posteriori (MAP) Hypothesis

2.4.1. Example:

h_1 : Customer buys a computer = Yes

h_2 : Customer buys a computer = No

where h_1 and h_2 are subsets of our Hypothesis Space 'H'

$P(h/D)$ (Final Outcome) = $\arg \max \{ P(D/h_1) P(h_1), P(D/h_2) P(h_2) \}$

$P(D)$ can be ignored as it is the same for both the terms

2.4.2. Theory:

Generally we want the most probable hypothesis given the training data $h_{MAP} = \arg \max P(h/D)$ (where h belongs to H and H is the hypothesis space)

$$h_{MAP} = \arg \max \frac{P(D/h) P(h)}{P(D)}$$

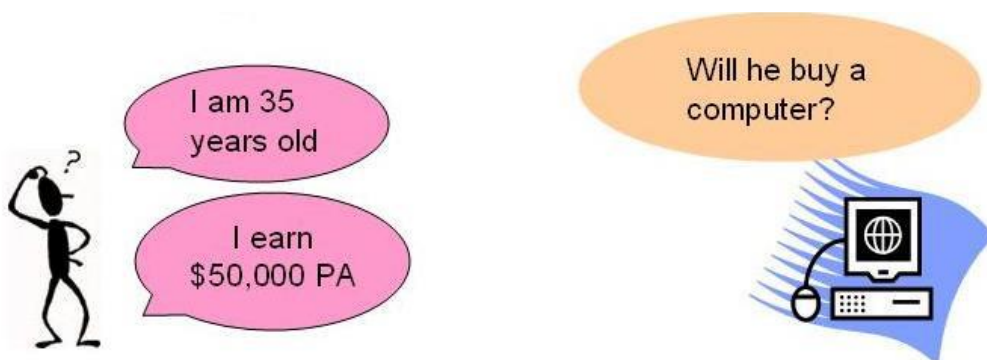
$$h_{MAP} = \arg \max P(D/h) P(h)$$

2.5. Maximum Likelihood (ML) Hypothesis

2.5.1. Example:

Sl No	Age	Income	Student	Credit rating	Buys computer
1	35	Medium	Yes	Fair	Yes
2	30	High	No	Average	No
3	40	Low	Yes	Good	No
4	35	Medium	No	Fair	Yes
5	45	Low	No	Fair	Yes
6	35	High	No	Excellent	Yes
7	35	Medium	No	Good	No
8	25	Low	No	Good	No
9	28	High	No	Average	No
10	35	Medium	Yes	Average	Yes

Table 2



2.5.2. Theory:

If we assume $P(h_i) = P(h_j)$ where the calculated probabilities amount to the same. Further simplification leads to:

$$h_{ML} = \arg \max P(D/h_i) \text{ (where } h_i \text{ belongs to } H)$$

2.5.3. Theory applied on previous example:

$$P(\text{buys computer} = \text{yes}) = 5/10 = 0.5$$

$$P(\text{buys computer} = \text{no}) = 5/10 = 0.5$$

$$P(\text{customer is 35 yrs \& earns } \$50,000) = 4/10 = 0.4$$

$P(\text{customer is 35 yrs \& earns \$50,000} / \text{buys computer} = \text{yes}) = 3/5 = 0.6$

$P(\text{customer is 35 yrs \& earns \$50,000} / \text{buys computer} = \text{no}) = 1/5 = 0.2$

Customer buys a computer $P(h1/D) = P(h1) * P(D/ h1) / P(D) = 0.5 * 0.6 / 0.4$

Customer does not buy a computer $P(h2/D) = P(h2) * P(D/ h2) / P(D) = 0.5 * 0.2 / 0.4$

Final Outcome = $\arg \max \{P(h1/D) , P(h2/D)\} = \max(0.6, 0.2)$

=> Customer buys a computer

3. Naïve Bayesian Classification

It is based on the Bayesian theorem It is particularly suited when the dimensionality of the inputs is high. Parameter estimation for naive Bayes models uses the method of maximum likelihood. In spite over-simplified assumptions, it often performs better in many complex realworld situations. Advantage: Requires a small amount of training data to estimate the parameters

3.1. Example

rec	Age	Income	Student	Credit_rating	Buys_computer
r1	<=30	High	No	Fair	No
r2	<=30	High	No	Excellent	No
r3	31...40	High	No	Fair	Yes
r4	>40	Medium	No	Fair	Yes
r5	>40	Low	Yes	Fair	Yes
r6	>40	Low	Yes	Excellent	No
r7	31...40	Low	Yes	Excellent	Yes
r8	<=30	Medium	No	Fair	No
r9	<=30	Low	Yes	Fair	Yes
r10	>40	Medium	Yes	Fair	Yes
r11	<=30	Medium	Yes	Excellent	Yes
r12	31...40	Medium	No	Excellent	Yes
r13	31...40	High	Yes	Fair	Yes
r14	>40	Medium	No	Excellent	No

$X = (\text{age} = \text{youth}, \text{income} = \text{medium}, \text{student} = \text{yes}, \text{credit_rating} = \text{fair})$

A person belonging to tuple X will buy a computer?

3.2.Theory:

Derivation:

D : Set of tuples

- Each Tuple is an 'n' dimensional attribute vector

- $X : (x_1, x_2, x_3, \dots, x_n)$

Let there be 'm' Classes : $C_1, C_2, C_3 \dots C_m$

Naïve Bayes classifier predicts X belongs to Class C_i iff

- $P(C_i/X) > P(C_j/X)$ for $1 \leq j \leq m, j \neq i$

Maximum Posteriori Hypothesis

- $P(C_i/X) = P(X/C_i) P(C_i) / P(X)$
- Maximize $P(X/C_i) P(C_i)$ as $P(X)$ is constant

With many attributes, it is computationally expensive to evaluate $P(X/C_i)$.

Naïve Assumption of "class conditional independence"

$$P\left(\frac{X}{C_i}\right) = \prod_{k=1}^n P(x_k/C_i)$$

$$P(X/C_i) = P(x_1/C_i) * P(x_2/C_i) * \dots * P(x_n/C_i)$$

3.3. Theory applied on previous example:

$$P(C_1) = P(\text{buys_computer} = \text{yes}) = 9/14 = 0.643$$

$$P(C_2) = P(\text{buys_computer} = \text{no}) = 5/14 = 0.357$$

$$P(\text{age}=\text{youth} / \text{buys_computer} = \text{yes}) = 2/9 = 0.222$$

$$P(\text{age}=\text{youth} / \text{buys_computer} = \text{no}) = 3/5 = 0.600$$

$$P(\text{income}=\text{medium} / \text{buys_computer} = \text{yes}) = 4/9 = 0.444$$

$$P(\text{income}=\text{medium} / \text{buys_computer} = \text{no}) = 2/5 = 0.400$$

$$P(\text{student}=\text{yes} / \text{buys_computer} = \text{yes}) = 6/9 = 0.667$$

$$P(\text{student}=\text{yes} / \text{buys_computer} = \text{no}) = 1/5 = 0.200$$

$$P(\text{credit rating}=\text{fair} / \text{buys_computer} = \text{yes}) = 6/9 = 0.667$$

$$P(\text{credit rating}=\text{fair} / \text{buys_computer} = \text{no}) = 2/5 = 0.400$$

$$P(X/\text{Buys a computer} = \text{yes}) = P(\text{age}=\text{youth} / \text{buys_computer} = \text{yes}) * P(\text{income}=\text{medium} / \text{buys_computer} = \text{yes}) * P(\text{student}=\text{yes} / \text{buys_computer} = \text{yes}) * P(\text{credit rating}=\text{fair} / \text{buys_computer} = \text{yes}) = 0.222 * 0.444 * 0.667 * 0.667 = 0.044$$

$$P(X/\text{Buys a computer} = \text{No}) = 0.600 * 0.400 * 0.200 * 0.400 = 0.019$$

Find class C_i that Maximizes $P(X/C_i) * P(C_i)$

$$\Rightarrow P(X/\text{Buys a computer} = \text{yes}) * P(\text{buys_computer} = \text{yes}) = 0.028$$

$$\Rightarrow P(X/\text{Buys a computer} = \text{No}) * P(\text{buys_computer} = \text{no}) = 0.007$$

Prediction : Buys a computer for Tuple X

4. Sample running example with weka

4.1. Bayesian Network Classifiers in Weka

Let $U = \{x_1, \dots, x_n\}$, $n \geq 1$ be a set of variables. A Bayesian network B over a set of variables U is a network structure BS , which is a directed acyclic graph (DAG) over U and a set of probability tables $BP = \{p(u|pa(u)) | u \in U\}$ where $pa(u)$ is the set of parents of u in BS . A Bayesian network represents a probability distributions $P(U) = \prod_{u \in U} p(u|pa(u))$.

Below, a Bayesian network is shown for the variables in the iris data set. Note that the links between the nodes class, petal length and petal width do not form a directed cycle, so the graph is a proper DAG.

4.2. Conditional independence test based structure learning

Conditional independence tests in Weka are slightly different from the standard tests described

in the literature. To test whether variables x and y are conditionally independent given a set of variables Z , a network structure with arrows $x \rightarrow y$ is compared with one with arrows $\{x \rightarrow y\} \setminus \{x \rightarrow y\}$. A test is performed.

At the moment, only the ICS [9] and CI algorithm are implemented. The ICS algorithm makes two steps, first find a skeleton (the undirected graph with edges iff there is an arrow in network structure) and second direct all the edges in the skeleton to get a DAG.

Starting with a complete undirected graph, we try to find conditional independencies $\langle x, y | Z \rangle$ in the data. For each pair of nodes x, y , we consider sets Z starting with cardinality 0, then 1 up to a user defined maximum. Further-more, the set Z is a subset of nodes that are neighbors of both x and y . If an independency is identified, the edge between x and y is removed from the skeleton.

The first step in directing arrows is to check for every configuration $x - z - y$ where x and y not connected in the skeleton whether z is in the set Z of variables that justified removing the link between x and y (cached in the first step). If z is not in Z , we can assign direction $x \rightarrow y$.

Finally, a set of graphical rules is applied to direct the remaining arrows.

```

Rule 1: i->j--k & i-/-k => j->k
Rule 2: i->j->k & i--k => i->k
Rule 3: m
      /|\
      i | k
      \|/
      j
i->j<-k => m->j

Rule 4: m
      / \
      i --- k
      \| /
      j
i->j => i->m & k->m

Rule 5: if no edges are directed then take a random one (first we can find)

```

The ICS algorithm comes with the following options.

Since the ICS algorithm is focused on recovering causal structure, instead of finding the optimal classifier, the Markov blanket correction can be made afterwards.

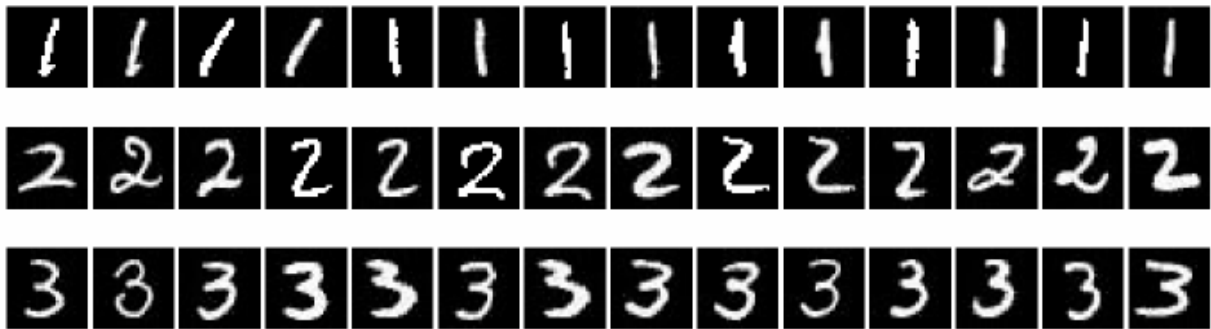
Specific options:

The maxCardinality option determines the largest subset of Z to be considered in conditional independence tests $\langle x, y|Z \rangle$. The scoreType option is used to select the scoring metric.

5. Exercises

Implement, test and interpret the results for Naïve Bayes algorithm for the following problems, using the attached input files

5.1. Being given the following binary files (imagini.zip), which represent the classes for 1, 2 and 3, you must find out the class of a digit in an image.



There will be used as attributes white pixels (value 255) and the positions of their appearance.

Algorithm:

Step 1: It is loaded the image which will be classified as being ONE, TWO or THREE

Step 2: There are loaded the images found in the folder **images**. The name of the files belonging to class ONE are: “image1_*.jpg”, the ones belonging to class TWO are: “image2_*.jpg” and the ones for class THREE are : “image3_*.jpg”.

Step3: It is determined the a priori probability for each class:

$$P(\text{UNU}) = \text{NrTemplateInClassONE} / \text{NumberTotalTemplates}$$

$$P(\text{DOI}) = \text{NrTemplateInClassTWO} / \text{NumberTotalTemplates}$$

$$P(\text{TREI}) = \text{NrTemplateInClassTHREE} / \text{NumberTotalTemplates}$$

Step 4: It is determined the probability that the image from the Step 1 to be in class ONE, TWO or THREE. Let (i,j) be the position of a white pixel in the image. It is calculated the probability that the pixel having the coordinates (i, j) to be white for the class ONE, TWO and THREE.

```

count1i,j = 0
for k = 1,n ; n – the number of images in class ONE
if image1_k(i,j) = 255 then
count1i,j = count1i,j + 1
probability1(i,j) =count1i,j / NrTemplateInClassONE
count2i,j = 0
for k = 1,n ; n- the number of images in class TWO
if image2_k(i,j) = 255 then
count2i,j = count2i,j + 1
probability2(i,j) =count2i,j / NrTemplateInClassTWO
count3i,j = 0
for k = 1,n ; n- the number of images in class THREE
if image3_k(i,j) = 255 then
count3i,j = count3i,j + 1
probability 3(i,j) =count3i,j / NrTemplateInClassTHREE

```

Step 5.

The posteriori probability that the image in Step 1 to be in class ONE is:

$P(T|ONE) = \text{average}(\text{probabilitate1}(i,j))$; (i, j) – the position of the white pixels in the image from Step1

Step 6.

The posteriori probability that the image in Step 1 to be in class TWO is:

$P(T|TWO) = \text{average}(\text{probabilitate1}(i,j))$; (i, j) – the position of the white pixels in the image from Step1

Step 7:

The posteriori probability that the image in Step 1 to be in class THREE is:

$P(T|THREE) = \text{average}(\text{probabilitate1}(i,j))$; (i, j) – the position of the white pixels in the image from Step1

Step 8:

It is determined the probability P for each image class and it is assigned the image from Step1 to the class of images that has the greatest probability.

$$P(ONE|T) = P(T| ONE)*P(ONE)$$

$$P(TWO|T) = P(T| TWO)*P(TWO)$$

$$P(THREE|T) = P(T| THREE)*P(THREE)$$

In order to load an image and to load pixels from an image in an array, you can use the following java code:

```
import java.awt.*;
import java.awt.image.*;
import java.io.*;
import javax.swing.*;
import java.util.*;
public class CImagesLoad {
    Vector<Image> images1 = new Vector<Image>();
    Vector<Image> images2 = new Vector<Image>();
    Vector<Image> images3 = new Vector<Image>();
    public String getFile(boolean isSaveDialog)
    {
        String    currentDirectoryName    =    new    File("").getAbsolutePath()
+File.separator;
        try{
            JFileChooser fc = new JFileChooser(new File(new
File(currentDirectoryName).getParent()));
            int result = 0;
            if(!isSaveDialog)
                result = fc.showOpenDialog(null);
            else
                result = fc.showSaveDialog(null);
            if(result==JFileChooser.CANCEL_OPTION) return null;
            else { //if(result==JFileChooser.APPROVE_OPTION){
                return fc.getSelectedFile().getAbsolutePath();
            }
        }
        catch(Exception e)
        {
            return null;
        }
    }
public void load_images (int template){
```

```

String f = getFile(false);
if (f==null)
{
    return;
}
int k = 1;
while (true)
{
    String curent = new java.io.File (f).getAbsolutePath ();
    int pos = curent.lastIndexOf ("\\");
    curent = curent.substring (0, pos);
    if (k < 10)
    {
        curent += "\\image" + template + "_0" + k + ".jpg";
    }
    else
    {
        curent += "\\image" + template + "_" + k + ".jpg";
    }
    Image img = null;
    img = new javax.swing.ImageIcon(curent).getImage();
    if(img==null || img.getWidth(null)<=0 ||img.getHeight(null)<=0)
    {
        System.out.println("The file \n" + f.toString() + "\nhas an unsupported
image format");
        break;
    }
    else
    {
        k++;
        switch (template)
        {
            case 1:
                images1.add (img);

```

```

        break;
    case 2:
        images2.add (img);
        break;
    case 3:
        images3.add (img);
        break;
    default:
        System.out.println("Other class");
        break;
    }
}
}
}

public void load_pixels (Image image)
{
    int width = image.getWidth(null);
    int height = image.getHeight(null);
    // Allocate buffer to hold the image's pixels
    int pixels[] = new int[width * height];
    // Grab pixels
    PixelGrabber pg = new PixelGrabber (image, 0, 0, width, height,
    pixels, 0, width);
    try
    {
        pg.grabPixels();
    }
    catch (InterruptedException e)
    {
        System.out.println ("Error image loading");
    }
}
}
}

```

5.2. Modify 5.1 in order to classify images that are belonging to class 4.

5.3. Implement the example 3.1 for the following tuple X:

X = (age= youth, income = high, student = no, credit_rating = fair)

Find out if a person belonging to tuple X will buy a computer

6. References

1. http://en.wikipedia.org/wiki/Bayesian_probability
2. http://en.wikipedia.org/wiki/Naive_Bayes_classifier
3. http://www.let.rug.nl/~tiedeman/ml05/03_bayesian_handout.pdf
4. <http://www.statsoft.com/textbook/stnaiveb.html>
5. <http://www.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/mlbook/ch6.pdf>
6. DATA MINING Concepts and Techniques, Jiawei Han, Micheline Kamber Morgan Kaufman Publishers, 2003.

Classification with Decision Trees

Purpose:

- Understand how to build simple baseline models for classification;
- Understand how to build decision trees for classification;
- Understand how different parameters for decision tree algorithms affect their output;
- Assess the accuracy of several models using cross-validation;
- Communicate the information captured in the decision tree model in well written English.

1. Theoretical Aspects

Classification is one of the major data mining tasks. Although this task is accomplished by generating a predictive model of data, interpreting the model frequently provides information for discriminating labeled classes in data. Decision trees provide a predictive model that is easy to interpret to provide a description of data.

In order to believe any predictive model, the accuracy of the model must be estimated. Several methods for evaluating the accuracy of models will be discussed during class lectures. For this assignment, 10-fold cross validation will be used for model assessment. The data mining task you are to perform is to provide descriptions of acceptable and unacceptable labor contracts contained in *labor.arff*. You are to back up you description by the evidence you collect by building decision tree models of the data.

1.1 Entropy

Putting together a decision tree is all a matter of choosing which attribute to test at each node in the tree. We shall define a measure called information gain which will be used to decide which attribute to test at each node. Information gain is itself calculated using a measure called entropy, which we first define for the case of a binary decision problem and then define for the general case.

Given a binary categorization, \mathbf{C} , and a set of examples, \mathbf{S} , for which the proportion of examples categorized as positive by \mathbf{C} is p_+ and the proportion of examples categorized as negative by \mathbf{C} is p_- , then the **entropy** of \mathbf{S} is:

$$Entropy(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-)$$

The reason we defined entropy first for a binary decision problem is because it is easier to get an impression of what it is trying to calculate.

Given an arbitrary categorization, C into categories c_1, \dots, c_n , and a set of examples, S , for which the proportion of examples in c_i is p_i , then the entropy of S is:

$$Entropy(S) = \sum_{i=1}^n -p_i \log_2(p_i)$$

This measure satisfies our criteria, because of the $-p \log_2(p)$ construction: when p gets close to zero (i.e., the category has only a few examples in it), then the $\log(p)$ becomes a big negative number, but the p part dominates the calculation, so the entropy works out to be nearly zero. Remembering that entropy calculates the disorder in the data, this low score is good, as it reflects our desire to reward categories with few examples in. Similarly, if p gets close to 1 (i.e., the category has most of the examples in), then the $\log(p)$ part gets very close to zero, and it is this which dominates the calculation, so the overall value gets close to zero. Hence we see that both when the category is nearly - or completely - empty, or when the category nearly contains - or completely contains - all the examples, the score for the category gets close to zero, which models what we wanted it to. Note that $0 \cdot \ln(0)$ is taken to be zero by convention.

1.2 Information gain

We now return to the problem of trying to determine the best attribute to choose for a particular node in a tree. The following measure calculates a numerical value for a given attribute, A , with respect to a set of examples, S . Note that the values of attribute A will range over a set of possibilities which we call $Values(A)$, and that, for a particular value from that set, v , we write S_v for the set of examples which have value v for attribute A .

The information gain of attribute A , relative to a collection of examples, S , is calculated as:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

The information gain of an attribute can be seen as the expected reduction in entropy caused by knowing the value of attribute A .

1.3 Sample calculation on Entropy and Information gain

Instances:

Weekend	Weather	Parents	Money	Decision (Category)
W1	Sunny	Yes	Rich	Cinema
W2	Sunny	No	Rich	Tennis
W3	Windy	Yes	Rich	Cinema
W4	Rainy	Yes	Poor	Cinema
W5	Rainy	No	Rich	Stay in
W6	Rainy	Yes	Poor	Cinema
W7	Windy	No	Poor	Cinema
W8	Windy	No	Rich	Shopping
W9	Windy	Yes	Rich	Cinema
W10	Sunny	No	Rich	Tennis

The first thing we need to do is work out which attribute will be put into the node at the top of our tree: **weather, parents or money**. To do this, we need to calculate:

$$\begin{aligned}
 \text{Entropy}(S) &= -p_{\text{cinema}} \log_2(p_{\text{cinema}}) - p_{\text{tennis}} \log_2(p_{\text{tennis}}) - p_{\text{shopping}} \log_2(p_{\text{shopping}}) - p_{\text{stay_in}} \log_2(p_{\text{stay_in}}) \\
 &= -(6/10) * \log_2(6/10) - (2/10) * \log_2(2/10) - (1/10) * \log_2(1/10) - (1/10) * \log_2(1/10) \\
 &= -(6/10) * -0.737 - (2/10) * -2.322 - (1/10) * -3.322 - (1/10) * -3.322 \\
 &= 0.4422 + 0.4644 + 0.3322 + 0.3322 = \mathbf{1.571}
 \end{aligned}$$

and we need to determine the best of:

$$\begin{aligned}
 \text{Gain}(S, \text{ weather}) &= 1.571 - (I_{\text{sunny}}/10) * \text{Entropy}(S_{\text{sunny}}) - (I_{\text{windy}}/10) * \text{Entropy}(S_{\text{windy}}) - \\
 & (I_{\text{rainy}}/10) * \text{Entropy}(S_{\text{rainy}}) \\
 &= 1.571 - (0.3) * \text{Entropy}(S_{\text{sunny}}) - (0.4) * \text{Entropy}(S_{\text{windy}}) - (0.3) * \text{Entropy}(S_{\text{rainy}}) \\
 &= 1.571 - (0.3) * (0.918) - (0.4) * (0.81125) - (0.3) * (0.918) = 0.70
 \end{aligned}$$

$$\begin{aligned}
 \text{Gain}(S, \text{ parents}) &= 1.571 - (I_{\text{yes}}/10) * \text{Entropy}(S_{\text{yes}}) - (I_{\text{no}}/10) * \text{Entropy}(S_{\text{no}}) \\
 &= 1.571 - (0.5) * 0 - (0.5) * 1.922 = 1.571 - 0.961 = 0.61
 \end{aligned}$$

$$\begin{aligned}
 \text{Gain}(S, \text{ money}) &= 1.571 - (I_{\text{rich}}/10) * \text{Entropy}(S_{\text{rich}}) - (I_{\text{poor}}/10) * \text{Entropy}(S_{\text{poor}}) \\
 &= 1.571 - (0.7) * (1.842) - (0.3) * 0 = 1.571 - 1.2894 = 0.2816
 \end{aligned}$$

This means that the first node in the decision tree will be the weather attribute. As an exercise, convince yourself why this scored (slightly) higher than the parents attribute - remember what entropy means and look at the way information gain is calculated.

From the weather node, we draw a branch for the values that weather can take: sunny, windy and rainy:



Finishing this tree off is left as a exercise.

2. Decision Tree Induction Algorithm

2.1 ID3 (Iterative Dichotomiser 3)

The ID3 algorithm can be summarized as follows:

1. Take all unused attributes and count their entropy concerning test samples
2. Choose attribute for which entropy is maximum
3. Make node containing that attribute

The actual algorithm is as follows:

ID3 (Examples, Target_Attribute, Attributes)

- Create a root node for the tree
- If all examples are positive, Return the single
- If all examples are negative, Return the single
- If number of predicting attributes is empty, then Return the single node tree

label = most common value of the target attribute in the examples.

- Otherwise Begin
 - A = The Attribute that best classifies examples.
 - Decision Tree attribute for Root = A.
 - For each possible value, v_i , of A
 - + Add a new tree branch below Root, corresponding to the test $A = v_i$.
 - + Let $\text{Examples}(v_i)$, be the subset of examples that have the value v_i for A
 - + If $\text{Examples}(v_i)$ is empty
 - Then below this new branch add a leaf node with label = most common target value in the examples
 - Else below this new branch add the subtree ID3 ($\text{Examples}(v_i)$, Target_Attribute,

Attributes – {A})

* End

* Return Root

2.2 C4.5 Algorithm

C4.5 builds decision trees from a set of training data in the same way as ID3, using the concept of information entropy. The training data is a set $S = s_1, s_2, \dots$ of already classified samples. Each sample $s_i = x_1, x_2, \dots$ is a vector where x_1, x_2, \dots represent attributes or features of the sample. The training data is augmented with a vector $C = c_1, c_2, \dots$ where c_1, c_2, \dots represent the class to which each sample belongs.

At each node of the tree, C4.5 chooses one attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. Its criterion is the normalized information gain (difference in entropy) that results from choosing an attribute for splitting the data. The attribute with the highest normalized information gain is chosen to make the decision.

The C4.5 algorithm then recurses on the smaller sublists.

This algorithm has a few base cases.

- All the samples in the list belong to the same class. When this happens, it simply creates a leaf node for the decision tree saying to choose that class.
- None of the features provide any information gain. In this case, C4.5 creates a decision node higher up the tree using the expected value of the class.
- Instance of previously-unseen class encountered. Again, C4.5 creates a decision node higher up the tree using the expected value.

In pseudocode, the algorithm is:

- Check for base cases
- For each attribute a
 - Find the normalized information gain from splitting on a
- Let a_best be the attribute with the highest normalized information gain
- Create a decision node that splits on a_best
- Recurse on the sublists obtained by splitting on a_best , and add those nodes as children of node

3. Build Baseline Classification Models

A baseline model is one that can be used to evaluate the success of your target model, in this case a decision tree model. Baseline models are typically simple, an inaccurate, but occasionally data is so simple to describe, attempting to use a complex model results in worse behaviour than a simple model.

In this section, you will build and record the accuracy to two baseline modes: ZeroR and OneR. The ZeroR model simply classifies every data item in the same class. For example, a ZeroR model may classify all loan applications as high risk without even considering the attributes of each data instance. The OneR model seeks to generate classification rules using a single attribute only.

1. Start Weka. On studsys, this can be accomplished with the command:

```
java -jar ../weka-3-2-3/weka.jar
```

2. Open Weka's Explorer interface.
3. Open *labor.arff* from the Explorer interface. (You may also want to open this file in a text editor).
4. Typically, at this point, you would record a summary of your data.
5. Click on the Classify tab in the Weka window.
6. The ZeroR classifier should be selected, but if it is not, click on the rectangle underneath the word Classifier and select the ZeroR classifier from the menu that pops up.
7. Select the Cross-validation radio button from the Test options. Use 10 folds.
8. Make sure that the class attribute is selected as the classification label.
9. Click the Start button to build and evaluate the model. Record the results in the output window.
10. Repeat the process, but select the OneR classifier from the Classifier menu using the default minimum bucket size of 6.

4. Generating Decision Trees (example 1)

Once you generate your baseline models and estimate their accuracy, you can create the target model of interest. Even though you will be creating a decision tree model, the algorithm you will use has several parameters. You need to modify the parameters in order to generate a reasonable model. Each time you modify a parameter, you might end up with a different model.

1. Before generating a decision tree, set accuracy goals for you classifier based on the accuracy of your baseline models.
2. Select the J48 algorithm for creating decision trees from the Classifier menu. Don't use the PART version.

3. Generate decision trees for the following combinations of attributes: all false, binarySplits only true, reducedErrorPruning only true, subtreeRaising only true, unpruned only true, useLaplace only true. Record the resulting decision trees and their associated accuracy and error information.
4. Generate decision trees with some combinations of the boolean parameters. Note that if you use reducedErrorPruning, the value of subtreeRaising is ignored. Again, record the results of each trial.
5. Now, for your best model so far, attempt to find a combination of confidenceFactor, minNumObj, numFolds that will improve your results. Use your understanding of the theory and documentation to make good selections. Record the results of each attempt.

5. Generating Decision Trees (example 2)

1. Start Weka. On studsys, this can be accomplished with the command:

```
java -jar ../weka-3-2-3/weka.jar
```
2. Open Weka's Explorer interface.
3. Open *credit.arff* from the Explorer interface. (You may also want to open this file in a text editor).
4. Run a decision tree classifier over the data by selecting classifiers > trees > J48 under the Classify tab.
5. Set a confidenceFactor of 0.2 in the options dialog.
6. Use a test percentage split of 90%.

Observe the output of the classifier. The full decision tree is output for your perusal; you may need to scroll up for this. The tree may also be viewed in graphical form by right-clicking the run in the Result list at the bottom-left and selecting Visualize tree, although it may be very cluttered for large trees.

- How would you assess the performance of the classifier? Hint: check the number of good and bad cases in the test sample (e.g. using the confusion matrix)
- Looking at the decision tree itself, are the rules it applies sensible? Are there any branches which appear absurd?
- What is the effect of the *confidenceFactor* option? Try increasing or decreasing the value of this option and observe the results.

For comparison, we also learn a decision stump. This is a decision tree with only a single node:

- Select the *DecisionStump* classifier.
- Select *Cross-validation* with 10 folds for the test option.

Now build the classifier, and observe the results:

- What single attribute does the algorithm use to make its decision? Do you expect this to be useful in its own right? Hint: visualisation could assist here.
- How do the results compare to that of the J48 tree? Is this what we would expect?
- Is the stump actually discriminating between anything? Hint: return to the Preprocess tab and observe the distribution of the Approve attribute. If
- 70% of applications are approved and 30% are not, how do we make a classifier that is 70% accurate?

6. Assignments

1. What are acceptable and unacceptable labor contracts according to your results?
2. Compare and contrast the pruned and unpruned trees you generated.
3. Do you feel your best tree is overfitting the data? Why or why not?
4. Which class is generally better recognized by the decision trees?
5. Decision trees are limited in the kinds classification problems they can solve
6. Can you find evidence that would lead you to believe other classification techniques would perform worse, better, or equally as well on the labor data?

Flat Clustering – K-Means Algorithm

Purpose:

Clustering algorithms group a set of documents into subsets or clusters. The cluster algorithms' goal is to create clusters that are coherent internally, but clearly different from each other. In other words, documents within a cluster should be as similar as possible; and documents in one cluster should be as dissimilar as possible from documents in other clusters.

Clustering is the most common form of unsupervised learning. No supervision means that there is no human expert who has assigned documents to classes. In clustering, it is the distribution and makeup of the data that will determine cluster membership. An example is in figure 1. It is visually clear that there are three distinct clusters of points. The difference between clustering and classification may not seem great at first. After all, in both cases we have a partition of a set of documents into groups. But as we will see that problems are fundamentally different. Classification is a form of supervised learning. Our goal is to replicate a categorical distinction that a human supervisor imposes on the data. In unsupervised learning, of which clustering is the most important example, we have no such teacher to guide us.

The key input to a clustering algorithm is the distance measure. In Figure 1, the distance measure is distance in the two-dimensional (2D) plane. This measure suggests three different clusters in the figure. In document clustering, the distance measure is often Euclidean distance. Different distance measures give rise to different clusterings. Thus, the distance measure is an important means by which we can influence the outcome of clustering. Flat clustering creates a flat set of clusters without any explicit structure that flat clustering would relate clusters to each other.

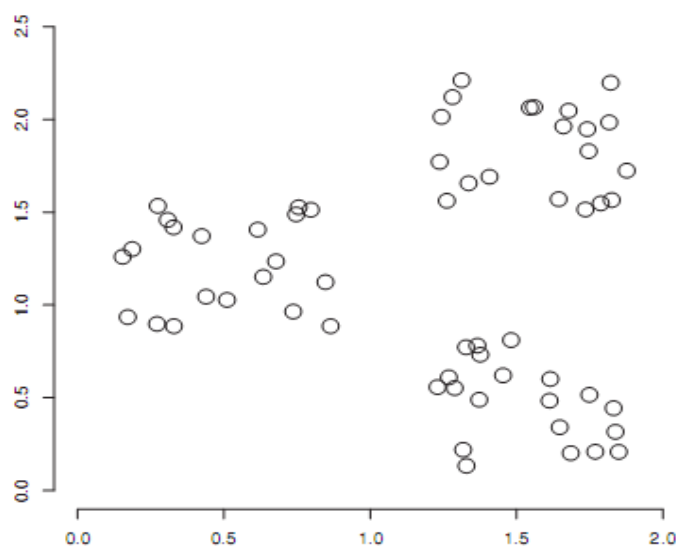


Figure 1: An example of a data set with a clear cluster structure.

A second important distinction can be made between hard and soft clustering algorithms. Hard clustering computes a hard assignment -each document is a member of exactly one cluster. The assignment of soft clustering algorithms is soft - a document's assignment is a distribution over all clusters. In a soft assignment, a document has fractional membership in several clusters.

Latent semantic indexing, a form of dimensionality reduction, is a soft clustering algorithm. This laboratory motivates the use of clustering in information retrieval by introducing a number of applications, defines the problem we are trying to solve in clustering, and discusses measures for evaluating cluster quality. It then describes the K means flat clustering algorithm, and the expectation maximization (or EM) algorithm, a soft clustering algorithm. K-means is perhaps the most widely used flat clustering algorithm because of its simplicity and efficiency. The EM algorithm is a generalization of K-means and can be applied to a large variety of document representations and distributions.

1. Problem statement

We can define the goal in hard flat clustering as follows. Given

(i) A set of documents $D = \{d_1 \dots d_N\}$,

(ii) A desired number of clusters K

(iii) An objective function that evaluates the quality of a clustering we want to compute an assignment $\gamma : D \rightarrow \{1, \dots, K\}$ that minimizes (or, in other cases, maximizes) the objective function. In most cases, we also demand that γ is subjective, that is, that none of the K clusters is empty. The objective function is often defined in terms of similarity or distance between documents. Below, we will see that the objective in K-means clustering is to minimize the average distance between documents and their centroids or, equivalently, to maximize the similarity between documents and their centroids. We use both similarity and distance to talk about relatedness between documents.

For documents, the type of similarity we want is usually topic similarity or high values on the same dimensions in the vector space model. For example, documents about China have high values on dimensions like Chinese, Beijing, and Mao whereas documents about the UK tend to have high values for London, Britain, and Queen. We approximate topic similarity with cosine similarity or Euclidean distance in vector space. If we intend to capture similarity of a type other than topic, for example, similarity of language, then a different representation may be appropriate. When computing topic similarity, stop word scan be safely ignored, but they are important cues for separating clusters of English (in which 'the' occurs frequently and 'la' infrequently) and French documents (in which 'the' occurs infrequently and 'la' frequently).

A difficult issue in clustering is determining the number of clusters or cardinality of a clustering, which we denote by K . Often K is nothing more than a good guess based on experience or domain knowledge. But for K-means, we will also introduce a heuristic method for choosing K and an attempt to incorporate the selection of K into the objective function. Sometimes the

application puts constraints on the range of K . For example, the scatter-gather interface in Figure 16.3 could not display more than about $K = 10$ clusters per layer because of the size and resolution of computer monitors in the early 1990s.

Because our goal is to optimize an objective function, clustering is essentially a search problem. The brute force solution would be to enumerate all possible clusterings and pick the best. However, there are exponentially many partitions, so this approach is not feasible. For this reason, most flat clustering algorithms refine an initial partition iteratively. If the search starts at an unfavorable initial point, we may miss the global optimum. Finding a good starting point is therefore another important problem we have to solve in flat clustering.

2. K-means

K-means is the most important flat clustering algorithm. Its objective is to minimize the average squared Euclidean distance of documents from their cluster centers where a cluster center is defined as the mean or centroid μ of the documents in a cluster ω : centroid

$$\vec{\mu}(\omega) = \frac{1}{|\omega|} \sum_{\vec{x} \in \omega} \vec{x}$$

The definition assumes that documents are represented as length normalized vectors in a real valued space in the familiar way. The ideal clustering K-means is a sphere with the centroid as its center of gravity. Ideally, the clusters should not overlap. Our desiderata for training set in clustering for which we know which documents should be in the same cluster classes in Rocchio classification were the same. The difference is that we have no labelled.

```

K-MEANS( $\{\vec{x}_1, \dots, \vec{x}_N\}$ ,  $K$ )
1   $(\vec{s}_1, \vec{s}_2, \dots, \vec{s}_K) \leftarrow \text{SELECTRANDOMSEEDS}(\{\vec{x}_1, \dots, \vec{x}_N\}, K)$ 
2  for  $k \leftarrow 1$  to  $K$ 
3  do  $\vec{\mu}_k \leftarrow \vec{s}_k$ 
4  while stopping criterion has not been met
5  do for  $k \leftarrow 1$  to  $K$ 
6    do  $\omega_k \leftarrow \{\}$ 
7    for  $n \leftarrow 1$  to  $N$ 
8    do  $j \leftarrow \arg \min_j |\vec{\mu}_j - \vec{x}_n|$ 
9    do  $\omega_j \leftarrow \omega_j \cup \{\vec{x}_n\}$  (reassignment of vectors)
10   for  $k \leftarrow 1$  to  $K$ 
11   do  $\vec{\mu}_k \leftarrow \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} \vec{x}$  (recomputation of centroids)
12  return  $\{\vec{\mu}_1, \dots, \vec{\mu}_K\}$ 

```

A measure of how well the centroids represent the members of their clusters is the residual sum of squares or RSS, the squared distance of each vector residual sum of squares from its centroids summed over all vectors:

$$RSS_k = \sum_{\vec{x} \in \omega_k} |\vec{x} - \vec{\mu}(\omega_k)|^2$$

$$RSS = \sum_{k=1}^K RSS_k$$

RSS is the objective function in K-means and our goal is to minimize it. Because N is fixed, minimizing RSS is equivalent to minimizing the average squared distance, a measure of how well centroids represent their documents.

The first step of K-means is to select as initial cluster centers K randomly selected documents, the seeds. The algorithm then moves the cluster centers seed around in space to minimize RSS. As shown in Figure16.5, this is done iteratively by repeating two steps until a stopping criterion is met: Reassigning documents to the cluster with the closest centroid and recomputing each centroid based on the current members of its cluster. Figure16.6 shows snapshots from nine iterations of the K-means algorithm for a set of points. The “centroid” column of Table17.2 (page364) shows examples of centroids. We can apply one of the following termination conditions.

A fixed number of iterations I has been completed. This condition limits The runtime of the clustering algorithm, but in some cases the quality of the clustering will be poor because of an insufficient number of iterations.

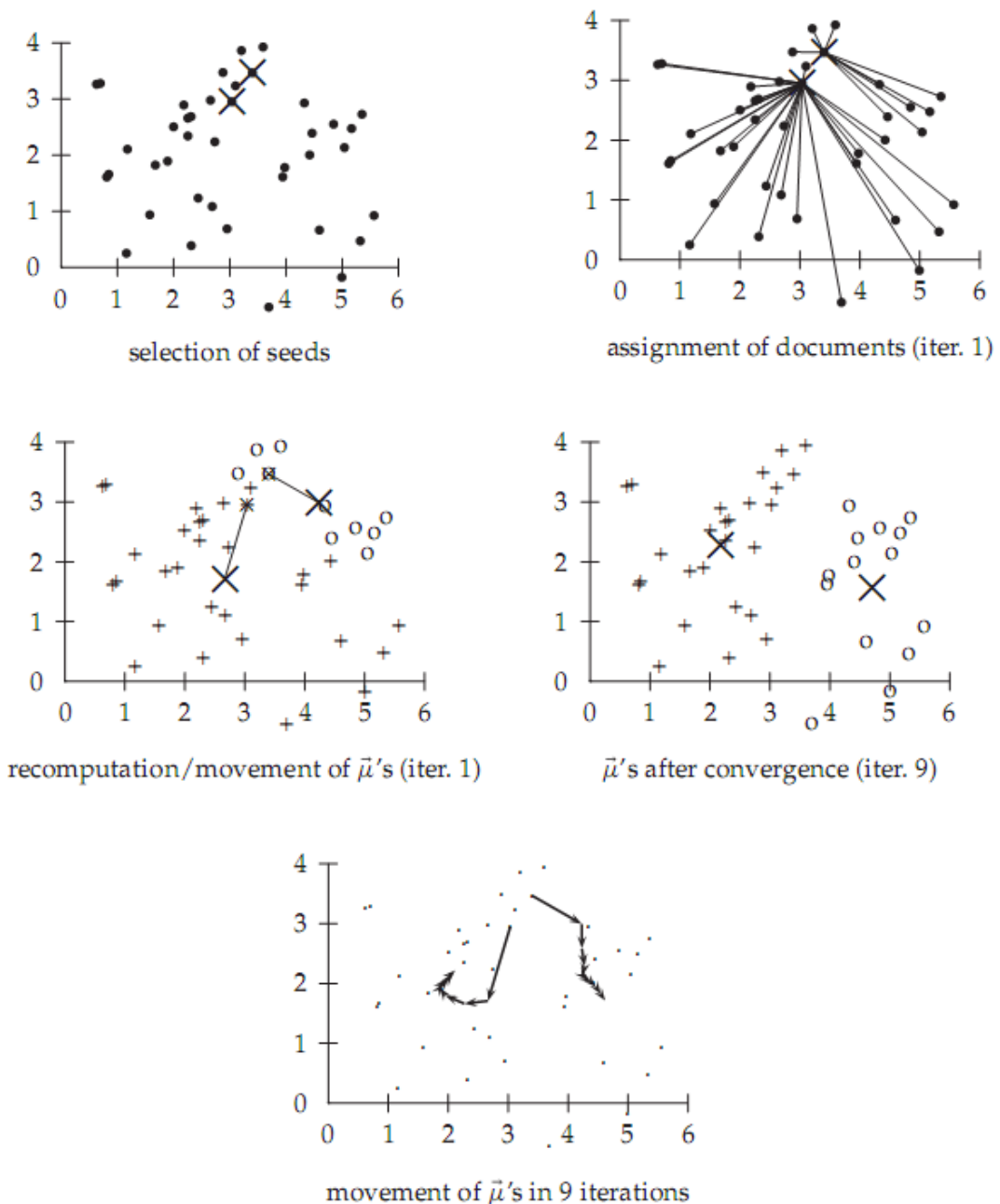


Figure 16.6 A K-means example for $K = 2$ in \mathbb{R}^2 . The position of the two centroids ($\bar{\mu}$'s shown as X's in the top four panels) converges after nine iterations.

Assignment of documents to clusters (the partitioning function ;) does not change between iterations. Except for cases with a bad local minimum, this produces a good clustering, but run-time may be unacceptably long.

Terminate when the decrease in RSS falls below a threshold ϵ . For small ϵ , this indicates that we are close to convergence. Again, we need to combine it with a bound on the number of iterations to prevent very long run-times. We now show that K-means converges by proving that RSS monotonically decreases in each iteration. We will use decrease in the meaning decrease or does not change in this section. First, RSS decreases in the reassignment step; each vector is assigned to the closest centroid, so the distance it contributes to RSS decreases. Second, it

decreases in the re-computation step because the new centroid is the vector \vec{v} for which RSS_k reaches its minimum.

$$RSS_k(\vec{v}) = \sum_{\vec{x} \in \omega_k} |\vec{v} - \vec{x}|^2 = \sum_{\vec{x} \in \omega_k} \sum_{m=1}^M (v_m - x_m)^2$$

$$\frac{\partial RSS_k(\vec{v})}{\partial v_m} = \sum_{\vec{x} \in \omega_k} 2(v_m - x_m)$$

where x_m and v_m are the m^{th} components of their respective vectors. Setting the partial derivative to zero, we get:

$$v_m = \frac{1}{|\omega_k|} \sum_{\vec{x} \in \omega_k} x_m$$

which is the component wise definition of the centroid. Thus, we minimize RSS_k when the old centroid is replaced with the new centroid. RSS , the sum of the RSS_k , must then also decrease during recomputation. Because there is only a finite set of possible clusterings, a monotonically decreasing algorithm will eventually arrive at a (local) minimum. Take care, however, to break ties consistently, for example, by assigning a document to the cluster with the lowest index if there are several equidistant centroids. Otherwise, the algorithm can cycle forever in a loop of clusterings that have the same cost. Although this proves the convergence of K-means, there is unfortunately no guarantee that a global minimum in the objective function will be reached.

This is a particular problem if a document set contains many outliers, documents that are far from any other documents and therefore do not fit well into any cluster. Frequently, if an outlier is chosen as an initial seed, then no other vector is assigned to it during subsequent iterations. Thus, we end up with a singleton cluster (a cluster with only one document) even though there singleton cluster is probably a clustering with lower RSS .

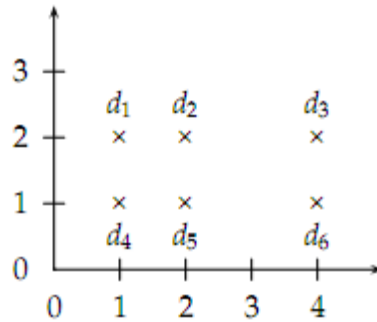


Figure 3: The outcome of clustering in K-means depends on the initial seeds. For seeds d_2 and d_5 , K-means converges to $\{\{d_1, d_2, d_3\}, \{d_4, d_5, d_6\}\}$, a suboptimal clustering. For seeds d_2 and d_3 , it converges to $\{\{d_1, d_2, d_4, d_5\}, \{d_3, d_6\}\}$, the global optimum for $K = 2$.

Effective heuristics for seed selection include (i) excluding outliers from the seed set; (ii) trying out multiple starting points and choosing the clustering with lowest cost; and (iii) obtaining seeds from another method such as hierarchical clustering. Because deterministic hierarchical

clustering methods are more predictable than K-means, a hierarchical clustering of a small random sample of size iK (e.g., for $i = 5$ or $i = 10$) often provides good seeds. Other initialization methods compute seeds that are not selected from the vectors to be clustered. A robust method that works well for a large variety of document distributions is to select i (e.g., $i = 10$) random vectors for each cluster and use their centroid as the seed for this cluster.

What is the time complexity of K-means? Most of the time is spent on computing vector distances. One such operation costs $\#(M)$. The reassignment step computes KN distances, so its overall complexity is $\#(KNM)$. In the re-computation step, each vector gets added to a centroid once, so the complexity of this step is $\#(NM)$. For a fixed number of iterations I , the overall complexity is therefore $\#(IKNM)$. Thus, K-means is linear in all relevant factors: iterations, number of clusters, number of vectors, and dimensionality of the space. This means that K-means is more efficient than the hierarchical algorithms. We had to fix the number of iterations I , which can be tricky in practice. But in most cases, K-means quickly reaches either complete convergence or a clustering that is close to convergence. In the latter case, a few documents would switch membership if further iterations were computed, but this has a small effect on the overall quality of the clustering.

There is one subtlety in the preceding argument. Even a linear algorithm can be quite slow if one of the arguments of $\#(\dots)$ is large, and M usually is large. High dimensionality is not a problem for computing the distance of two documents. Their vectors are sparse, so that only a small fraction of the theoretically possible M component wise differences need to be computed.

Centroids, however, are dense; they pool all terms that occur in any of the documents of their clusters. As a result, distance computations are time consuming in a naïve implementation of K-means. But there are simple and effective heuristics for making centroid–document similarities as fast to compute as document–document similarities. Truncating centroids to the most significant k terms (e.g., $k = 1,000$) hardly decreases cluster quality while achieving a significant speedup of the reassignment step.

The same efficiency problem is addressed by K-medoids, a variant of K-means that computes medoids instead of centroids as cluster centers. We define the medoid of a cluster as the document vector that is closest to the medoid centroid. Since medoids are sparse document vectors, distance computations are fast.

3. Working example using Weka

1. Under the Process tab in Experimenter window, press Open File;
2. Select `<path_to_weka>/data/weather.arff` or any other input data
3. The attributes and their possible values for the relation weather are as follows:
 - @attribute outlook {sunny, overcast, rainy}
 - @attribute temperature real
 - @attribute humidity real

@attribute windy {TRUE, FALSE}

@attribute play {yes, no}

4. Under the Cluster tab, tick off the “Use training set combo box” and press start.

5. The output of the clustering is as follows:

Number of iterations: 3

Within cluster sum of squared errors: 16.237456311387238

Missing values globally replaced with mean/mode

Cluster centroids:

Cluster#

Attribute Full Data 0 1

(14) (9) (5)

=====

outlook sunny sunny overcast

temperature 73.5714 75.8889 69.4

humidity 81.6429 84.1111 77.2

windy FALSE FALSE TRUE

play yes yes yes

Hereafter, we have 2 clusters. The first cluster's centroid is represented by the vector on the (0) column, the second cluster is represented by the vector on the last column, and the mean of all the data is presented on the first column.

4. Real world clustering examples

5.1 Clustering of wines

Being given a database containing categories of wine, described by their properties, we should be able to create some clusters of wine categories, splitting them by major characteristics.

Type	Alcohol	Malic_acid	Ash	Ash_alkalinity	Magnesium	Total_phenols
A	14.23	1.71	2.43	15.6	127	2.8
A	13.2	1.78	2.14	11.2	100	2.65
A	13.16	2.36	2.67	18.6	101	2.8
A	14.37	1.95	2.5	16.8	113	3.85
A	13.24	2.59	2.87	21	118	2.8
A	14.2	1.76	2.45	15.2	112	3.27
A	14.39	1.87	2.45	14.6	96	2.5
A	14.06	2.15	2.61	17.6	121	2.6
A	14.83	1.64	2.17	14	97	2.8
A	13.86	1.35	2.27	16	98	2.98
A	14.1	2.16	2.3	18	105	2.95
A	14.12	1.48	2.32	16.8	95	2.2
A	13.75	1.73	2.41	16	89	2.6

http://www.resample.com/xlminer/help/kMClst/KMClust_ex.htm

5.2 Clustering of students in a group

Being given a database of students from a group, we should be able to create 4 clusters named : weak, Normal, Smart, Outstanding based on their grades on different disciplines. After the clustering is done, we can calculate the distance between a student and a cluster centroid

6. Assignments

1. Being given the following relation: Student(Name, gradeMath, gradeProgramming, gradePhysics, gradeEnglish, gradeOverall), create an arff file containing at least 15 instances, load it into Weka, and apply k-Means clustering to it. Also cluster the instances without Weka, and compare the results. Pick different initial cluster centroids and compare the results.
2. Also create an arff file according to the table with wine instances, load it into Weka and see the results after applying k-Means clustering.
3. Develop a C program that clusters a planar set P of $m=3k$ points into k triangles such that the sum of all triangle circumferences is minimized.
4. Develop a C program that clusters $m = nk$ points on a line into k clusters of equal size, i.e. a balanced clustering, with a minimum sum of all distances between points of the same subset consists of k disjoint segments of the line each containing n points.

Hierarchical Clustering

Purpose:

- Understand theoretical aspects and the most important algorithms used for Hierarchical Clustering;
- See examples of the domains where hierarchical clustering are used in practice;
- Solve practical problems with hierarchical clustering.

1. Theoretical aspects: Assignments

1.1 What is Hierarchical clustering?

Hierarchical clustering is a method of cluster analysis which follows to build a hierarchy of clusters. Hierarchical cluster analysis (or hierarchical clustering) is a general approach to cluster analysis, in which the object is to group together objects or records that are "close" to one another.

A key component of the analysis is repeated calculation of distance measures between objects, and between clusters once objects begin to be grouped into clusters. The outcome is represented graphically as a dendrogram (the dendrogram is a graphical representation of the results of hierarchical cluster analysis).

The initial data for the hierarchical cluster analysis of N objects is a set of $N \times (N - 1) / 2$ object-to-object distances and a linkage function for computation of the cluster-to-cluster distances. A linkage function is an essential feature for hierarchical cluster analysis. Its value is a measure of the "distance" between two groups of objects (i.e. between two clusters).

The two main **categories of methods for hierarchical cluster analysis** are *divisive methods* and *agglomerative methods*. In practice, the agglomerative methods are of wider use. On each step, the pair of clusters with smallest cluster-to-cluster distance is fused into a single cluster.

1.2 Where Hierarchical Clustering is useful?

First example where hierarchical clustering would be useful is a study to predict the cost impact of deregulation. To do the requisite analysis, economists would need to build a detailed cost model of the various utilities. It would save a considerable amount of time and effort if we could cluster similar types of utilities, build detailed cost models for just one typical utility in each cluster, then scale up from these models to estimate results for all utilities.

Second example where hierarchical clustering would be useful is for automatic control of urban road traffic with both adaptive traffic lights and variable message signs. Using hierarchical cluster analysis we can specify the needed number of stationary road traffic sensors and their preferable locations within a given road network.

Third example of using a hierarchical clustering is to take a file that contains nutritional information for a set of breakfast cereals. We have the following information: the cereal name, cereal manufacturer, type (hot or cold), number of calories per serving, grams of protein, grams of fat,

milligrams of sodium, grams of fiber, grams of carbohydrates, grams of sugars, milligrams of potassium, typical percentage of the FDA's RDA of vitamins, the weight of one serving, the number of cups in one serving. Hierarchical Clustering help to find which cereals are the best and worst in a particular category.

1.3 Algorithms for hierarchical clustering:

The most common algorithms for hierarchical clustering are:

Agglomerative methods

An agglomerative hierarchical clustering procedure produces a series of partitions of the data, P_n, P_{n-1}, \dots, P_1 . The first P_n consists of n single object 'clusters', the last P_1 , consists of single group containing all n cases.

At each particular stage the method joins together the two clusters which are closest together (most similar). (At the first stage, of course, this amounts to joining together the two objects that are closest together, since at the initial stage each cluster has one object.)

Differences between methods arise because of the different ways of defining distance (or similarity) between clusters. Several agglomerative techniques will now be described in detail.

Single linkage clustering

One of the simplest agglomerative hierarchical clustering method is single linkage, also known as the nearest neighbor technique. The defining feature of the method is that distance between groups is defined as the distance between the closest pair of objects, where only pairs consisting of one object from each group are considered.

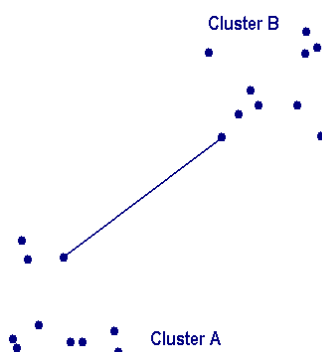
In the single linkage method, $D(r,s)$ is computed as:

$$D(r,s) = \text{Min} \{ d(i,j) : \text{Where object } i \text{ is in cluster } r \text{ and object } j \text{ is cluster } s \}$$

Here the distance between every possible object pair (i,j) is computed, where object i is in cluster r and object j is in cluster s . The minimum value of these distances is said to be the distance between clusters r and s . In other words, the distance between two clusters is given by the value of the shortest link between the clusters.

At each stage of hierarchical clustering, the clusters r and s , for which $D(r,s)$ is minimum, are merged.

This measure of inter-group distance is illustrated in the figure below:



Complete linkage clustering

The complete linkage, also called farthest neighbor, clustering method is the opposite of single linkage. Distance between groups is now defined as the distance between the most distant pair of objects, one from each group.

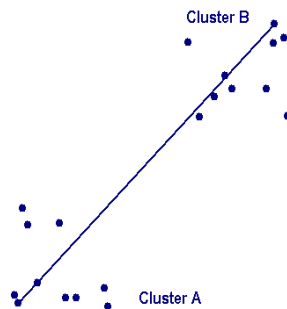
In the complete linkage method, $D(r,s)$ is computed as

$$D(r,s) = \text{Max} \{ d(i,j) : \text{Where object } i \text{ is in cluster } r \text{ and object } j \text{ is cluster } s \}$$

Here the distance between every possible object pair (i,j) is computed, where object i is in cluster r and object j is in cluster s and the maximum value of these distances is said to be the distance between clusters r and s . In other words, the distance between two clusters is given by the value of the longest link between the clusters.

At each stage of hierarchical clustering, the clusters r and s , for which $D(r,s)$ is minimum, are merged.

The measure is illustrated in the figure below:



Average linkage clustering

The distance between two clusters is defined as the average of distances between all pairs of objects, where each pair is made up of one object from each group.

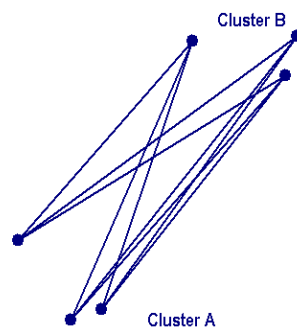
In the average linkage method, $D(r,s)$ is computed as

$$D(r,s) = \text{Trs} / (N_r * N_s)$$

Where Trs is the sum of all pairwise distances between cluster r and cluster s . N_r and N_s are the sizes of the clusters r and s respectively.

At each stage of hierarchical clustering, the clusters r and s , for which $D(r,s)$ is the minimum, are merged.

The figure below illustrates average linkage clustering:



Average group linkage

With this method, groups once formed are represented by their mean values for each variable, that is, their mean vector, and inter-group distance is now defined in terms of distance between two such mean vectors.

In the average group linkage method, the two clusters r and s are merged such that, after merger, the average pairwise distance within the newly formed cluster, is minimum. Suppose we label the new cluster formed by merging clusters r and s , as t . Then $D(r,s)$, the distance between clusters r and s is computed as

$$D(r,s) = \text{Average} \{ d(i,j) : \text{Where observations } i \text{ and } j \text{ are in cluster } t, \text{ the cluster formed by merging clusters } r \text{ and } s \}$$

At each stage of hierarchical clustering, the clusters r and s , for which $D(r,s)$ is minimum, are merged. In this case, those two clusters are merged such that the newly formed cluster, on average, will have minimum pairwise distances between the points in it.

Cobweb

Cobweb generates hierarchical clustering, where clusters are described probabilistically. Below is an example clustering of the weather data (weather.arff). The class attribute (play) is ignored (using the ignore attributes panel) in order to allow later classes to clusters evaluation. Doing this automatically through the "Classes to clusters" option does not make much sense for hierarchical clustering, because of the large number of clusters. Sometimes we need to evaluate particular clusters or levels in the clustering hierarchy.

How Weka represents the Cobweb clusters?

Below is a copy of the output window, showing the run time information and the structure of the clustering tree.

```
Scheme: weka.clusterers.Cobweb -A 1.0 -C 0.234
Relation: weather
Instances: 14
Attributes: 5
outlook
temperature
humidity
windy
Ignored:
play
Test mode: evaluate on training data
Clustering model (full training set)
Number of merges: 2
Number of splits: 1
Number of clusters: 6
node 0 [14]
|   node 1 [8]
```

```

|      | leaf 2 [2]
|      node 1 [8]
|      | leaf 3 [3]
|      node 1 [8]
|      | leaf 4 [3]
node 0 [14]
|      leaf 5 [6]

```

Evaluation on training set

Number of merges: 2

Number of splits: 1

Number of clusters: 6

```

node 0 [14]
|      node 1 [8]
|      | leaf 2 [2]
|      node 1 [8]
|      | leaf 3 [3]
|      node 1 [8]
|      | leaf 4 [3]
node 0 [14]
|      leaf 5 [6]

```

Clustered Instances

2 2 (14%)

3 3 (21%)

4 3 (21%)

5 6 (43%)

Comments on the output above:

- node N or leaf N represents a subcluster, whose parent cluster is N.
- The clustering tree structure is shown as a horizontal tree, where subclusters are aligned at the same column. For example, cluster 1 (referred to in node 1) has three subclusters 2 (leaf 2), 3 (leaf 3) and 4 (leaf 4).
- The root cluster is 0. Each line with node 0 defines a subcluster of the root.
- The number in square brackets after node N represents the number of instances in the parent cluster N.
- Clusters with [1] at the end of the line are instances.
- For example, in the above structure cluster 1 has 8 instances and its subclusters 2, 3 and 4 have 2, 3 and 3 instances correspondingly.
- To view the clustering tree right click on the last line in the result list window and then select Visualize tree.

To evaluate the Cobweb clustering using the classes to clusters approach we need to know the class values of the instances, belonging to the clusters. We can get this information from Weka in the following way: After Weka finishes (with the class attribute ignored), right click on the last line in the result list window. Then choose Visualize cluster assignments - you get the Weka cluster visualize



window. Here you can view the clusters, for example by putting Instance_number on X and Cluster on Y. Then click on Save and choose a file name (*.arff). Weka saves the cluster assignments in an ARFF file. Below is shown the file corresponding to the above Cobweb clustering.

2. Examples

First example:

A hierarchical clustering of distances in kilometres between some Italian cities. The method used is single-linkage.

Input distance matrix ($L = 0$ for all the clusters):

	BA	FI	MI	NA	RM	TO
BA	0	662	877	255	412	996
FI	662	0	295	468	268	400
MI	877	295	0	754	564	138
NA	255	468	754	0	219	869
RM	412	268	564	219	0	669
TO	996	400	138	869	669	0



The nearest pair of cities is MI and TO, at distance 138. These are merged into a single cluster called "MI/TO". The level of the new cluster is $L(\text{MI/TO}) = 138$ and the new sequence number is $m = 1$.

Then we compute the distance from this new compound object to all other objects. In single link clustering the rule is that the distance from the compound object to another object is equal to the shortest distance from any member of the cluster to the outside object. So the distance from "MI/TO" to RM is chosen to be 564, which is the distance from MI to RM, and so on.

After merging MI with TO, we obtain the following matrix:

	BA	FI	MI/TO	NA	RM
BA	0	662	877	255	412
FI	662	0	295	468	268
MI/TO	877	295	0	754	564
NA	255	468	754	0	219
RM	412	268	564	219	0



$\min d(i,j) = d(\text{NA},\text{RM}) = 219 \Rightarrow$ merge NA and RM into a new cluster called NA/RM

$L(\text{NA/RM}) = 219 ,m = 2$

	BA	FI	MI/TO	NA/RM
BA	0	662	877	255
FI	662	0	295	268
MI/TO	877	295	0	564
NA/RM	255	268	564	0



$\min d(i,j) = d(\text{BA}, \text{NA/RM}) = 255 \Rightarrow$ merge BA and NA/RM into a new cluster called

BA/NA/RM

$L(\text{BA/NA/RM}) = 255$

$m = 3$

	BA/NA/RM	FI	MI/TO
BA/NA/RM	0	268	564
FI	268	0	295
MI/TO	564	295	0



$\min d(i,j) = d(\text{BA/NA/RM}, \text{FI}) = 268 \Rightarrow$ merge BA/NA/RM and FI into a new cluster called

BA/FI/NA/RM

$L(\text{BA/FI/NA/RM}) = 268$

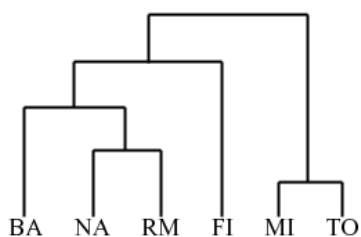
$m = 4$

	BA/FI/NA/RM	MI/TO
BA/FI/NA/RM	0	295
MI/TO	295	0



Finally, we merge the last two clusters at level 295.

The process is summarized by the following hierarchical tree:



Second example:

Coordination Example:

Researchers performed a microarray experiment to generate a gene expression profile data set that indicates relative levels of expression for each of these genes (> 12000) in murine muscle samples. They measured expression levels at 27 time points to find genes that are biologically relevant to the muscle regeneration process. They already know that MyoD is a gene that is the most relevant to muscle regeneration. They run the hierarchical clustering with the data set, and identify a relevant cluster that peaks at day 3. In the parallel coordinates view, they search MyoD using search-by-name query, then make it a model pattern to perform a model-based query. They modify the model pattern to emphasize the peak at day 3 and then adjust the similarity thresholds to get the search result that mostly overlaps with the relevant day 3 cluster (Fig. 1 & Fig. 2). Finally, they confirm through other biological experiments that 2 genes (Cdh15 and Stam) in the overlapped result set are novel downstream targets of MyoD.

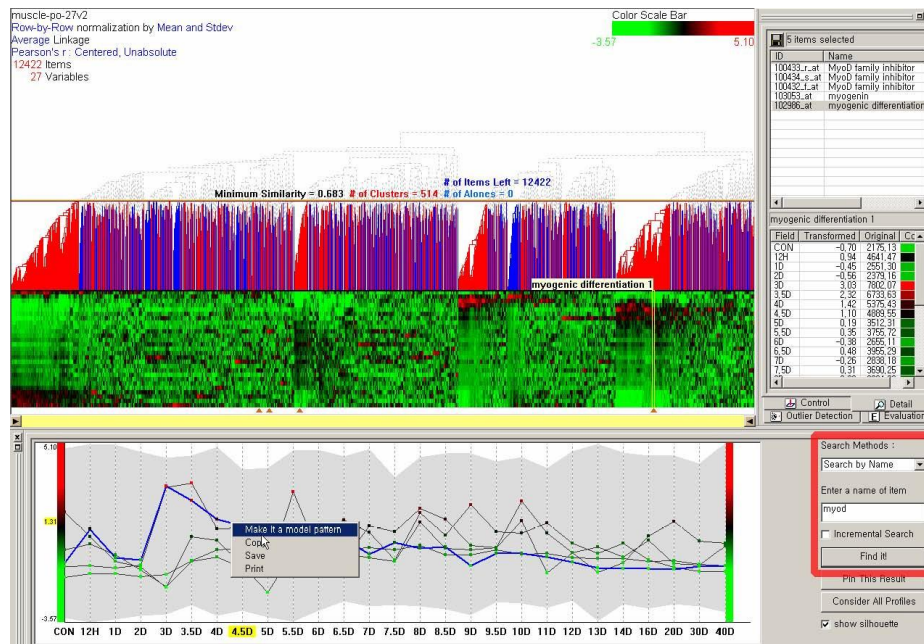


Fig. 1

Run a search-by-name query with MyoD to find 5 genes whose names contain MyoD, and the 5 genes are projected onto the current clustering result visualization shown by triangles under the color mosaic. Select a gene (myogenic differentiation 1) and make it a model pattern for next query.

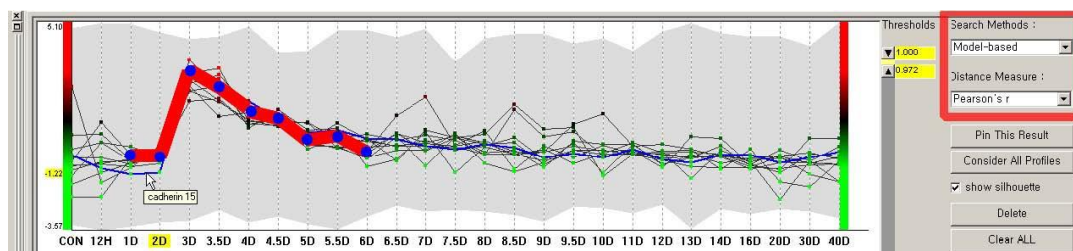


Fig. 2

Modify the model pattern to emphasize the peak at day 3 (notice the bold red line), and run a model-based query to find a small set of candidate genes. The updated search result will be highlighted in the dendrogram view and other views.

3. Assignments

Problem 1:

We have the following data files:

cereal.txt (without 'vitamin' and 'rating' columns) : 77 x 9

here: <http://www.cs.umd.edu/hcil/hce/examples/cereal/cereal.txt>

cereal-updated.txt (with 'vitamin' and 'rating' columns) : 77 x 11

here: <http://www.cs.umd.edu/hcil/hce/examples/cereal/cereal-updated.txt>

The meaning of each column :

1. 1st column : Name of cereal
2. calories: calories per serving
3. protein: grams of protein
4. fat: grams of fat
5. sodium: milligrams of sodium
6. fiber: grams of dietary fiber
7. carbo: grams of complex carbohydrates
8. sugars: grams of sugars
9. potass: milligrams of potassium
10. vitamins: vitamins and minerals - 0, 25, or 100, indicating the typical percentage of FDA recommended
11. shelf: display shelf (1, 2, or 3, counting from the floor)
12. rating: a rating of the cereals (calculated by Consumer Reports)

Requirements:

Use the given data files to find the following using WEKA:

1. Is a strong correlation between dietary fiber and potassium?
2. Are groups of cereals from which we can choose according to our preferences?
3. See other correlation between the data given in the files.

Problem 2:

We have the following data files:

netscan-08-2003.txt (activity log of newsgroups where name contains "windowsxp" for August 2003) : 91x10

here: <http://www.cs.umd.edu/hcil/hce/examples/netscan/netscan-08-2003.txt>

netscan-1year.txt (activity log of newsgroups where name contains "windowsxp" for a year) : 104 x 10

here: <http://www.cs.umd.edu/hcil/hce/examples/netscan/netscan-1year.txt>

The meaning of each column :

1. 1st column : name of newsgroup
2. **Posts** : # of messages that were contributed to the newsgroup
3. **Posters** : # of people who contributed at least on message to the newsgroup
4. **PPRatio**: the ratio of posters to posts
5. **Returns**: # of people who contributed to the newsgroup in the current time period and also contributed a message in the previous time period
6. **Replies**: # of people who contributed at least one message that was a reply to another message
7. **UnRMSGs**: # of messages in the newsgroup that did not receive any reply in the newsgroup
8. **Avg.LineCT**: average # of lines in each message
9. **XPosts**:# of messages that were shared with at least one other newsgroup
10. **XPTgs**:# of newsgroups that shared messages with the selected newsgroups

Requirements:

Use the given data files to find the following using WEKA:

1. What are the most active groups in terms of the number of people involved cluster together?
2. What are the most active communitie?

The COBWEB Conceptual Clustering Algorithm

The COBWEB algorithm was developed by machine learning researchers in the 1980s for clustering objects in a object-attribute data set. The COBWEB algorithm yields a clustering dendrogram called classification tree that characterizes each cluster with a probabilistic description.

Operation of the COBWEB algorithm

The COBWEB algorithm constructs a classification tree incrementally by inserting the objects into the classification tree one by one. When inserting an object into the classification tree, the COBWEB algorithm traverses the tree top-down starting from the root node.

At each node, the COBWEB algorithm considers 4 possible operations and select the one that yields the highest CU function value:

- insert.
- create.
- merge.
- split.

The COBWEB algorithm operates based on the so-called category utility function (CU) that measures clustering quality.

If we partition a set of objects into m clusters, then the CU of this particular partition is

Improvement in probability estimate because of instance cluster assignment

$$C \cup (C_1, C_2, \dots, C_k) = \frac{\sum_l \Pr[C_l] \sum_i \sum_j (\overbrace{\Pr[a_i = v_{ij} | C_l]^2 - \Pr[a_i = v_{ij}]^2})}{k}$$

If each instance in its own cluster:

$$\Pr[a_i = v_{ij} | C_l] = \begin{cases} 1 & v_{ij} = \text{actual value of instance} \\ 0 & \text{otherwise} \end{cases}$$

Category utility function becomes:

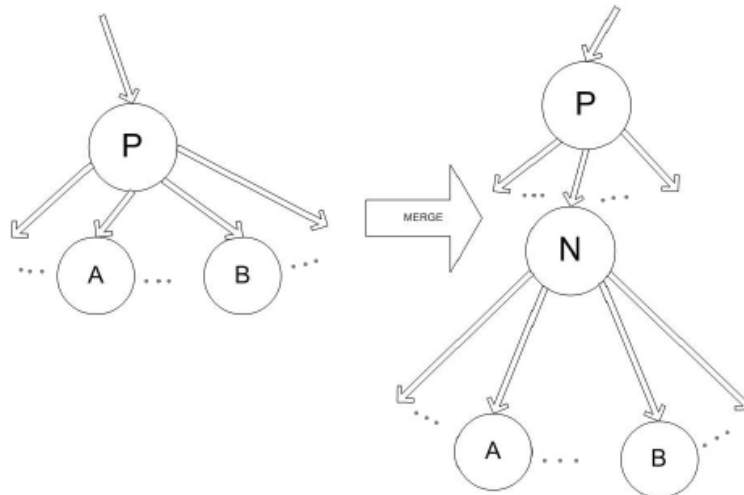
$$C \cup (C_1, C_2, \dots, C_k) = \frac{n - \sum_i \sum_j \Pr[a_i = v_{ij}]^2}{k}$$

Without k it would always be best for each instance to have its own cluster, overfitting!

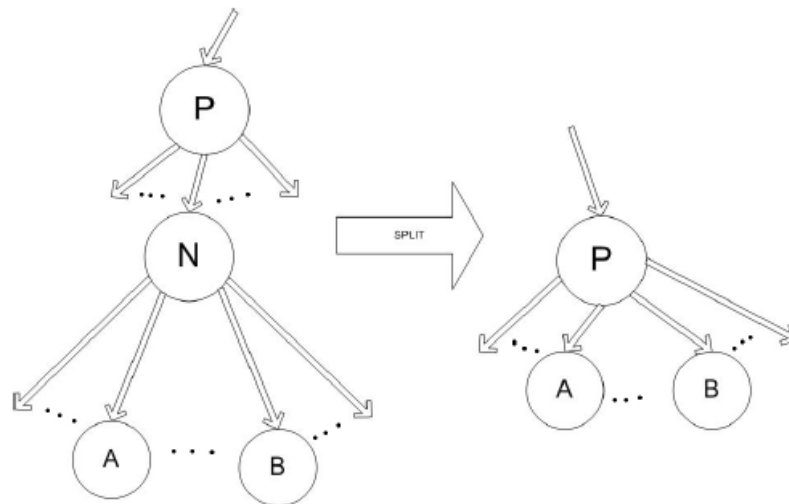
Insertion means that the new object is inserted into one of the existing child nodes. The COBWEB algorithm evaluates the respective CU function value of inserting the new object into each of the existing child nodes and selects the one with the highest score.

The COBWEB algorithm also considers creating a new child node specifically for the new object.

The COBWEB algorithm considers merging the two existing child nodes with the highest and second highest scores.



The COBWEB algorithm considers splitting the existing child node with the highest score.



The COBWEB Algorithm

Input: The current node N in the concept hierarchy.

An unclassified (attribute-value) instance I.

Results: A concept hierarchy that classifies the instance.

Top-level call: Cobweb(Top-node, I).

Variables: C, P, Q, and R are nodes in the hierarchy. U, V, W, and X are clustering (partition) scores.

Cobweb(N, I)

If N is a terminal node,

Then Create-new-terminals(N, I)

Incorporate(N,I).

Else Incorporate(N, I).

For each child C of node N,

Compute the score for placing I in C.

Let P be the node with the highest score W.

Let Q be the node with the second highest score.

Let X be the score for placing I in a new node R.

Let Y be the score for merging P and Q into one node.

Let Z be the score for splitting P into its children.

If W is the best score,

Then Cobweb(P, I) (place I in category P).

Else if X is the best score,

Then initialize R's probabilities using I's values
(place I by itself in the new category R).

Else if Y is the best score,

Then let O be Merge(P, R, N).

Cobweb(O, I).

Else if Z is the best score

Then Split(P, N).

Cobweb(N, I).

Auxiliary COBWEB Operations

Variables: N, O, P, and R are nodes in the hierarchy.

I is an unclassified instance.

A is a nominal attribute.

V is a value of an attribute.

Incorporate(N, I)

update the probability of category N.

For each attribute A in instance I,

For each value V of A,

Update the probability of V given category N.

Create-new-terminals(N, I)

Create a new child M of node N.

Initialize M's probabilities to those for N.

Create a new child O of node N.

Initialize O's probabilities using I's value.

Merge(P, R, N)

Make O a new child of N.

Set O's probabilities to be P and R's average.

Remove P and R as children of node N.

Add P and R as children of node O.

Return O.

Split(P, N)

Remove the child P of node N.

Promote the children of P to be children of N.

An example of using COBWEB Algorithm:

We have the following data:

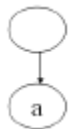
outlook	Temp	Humidity	Windy	Play
sunny	Hot	High	FALSE	No
Sunny	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Rainy	Mild	High	FALSE	Yes
Rainy	Cool	Normal	FALSE	Yes

Rainy	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Sunny	Mild	High	FALSE	No
Sunny	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	FALSE	Yes
Sunny	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Rainy	Mild	High	TRUE	No

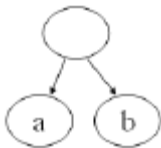
Weather Data (without Play)

Label instances: a,b,...,n

Start by putting the first instance in its own cluster:

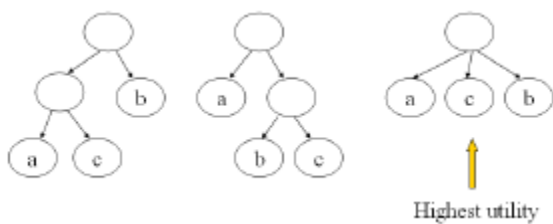


Add another instance in its own cluster:



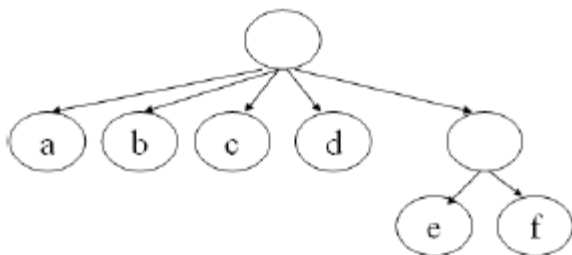
Adding the Third Instance

Evaluate the category utility of adding the instance to one of the two clusters versus adding it as its own cluster:



Adding Instance f

First instance not to get its own cluster:



Look at the instances:

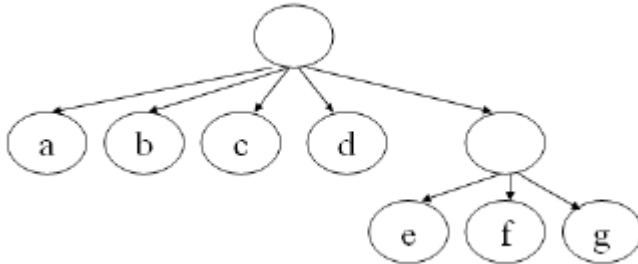
Rainy Cool Normal FALSE
 Rainy Cool Normal TRUE

Quite similar!

Add Instance g

Look at the instances:

- E) Rainy Cool Normal FALSE
- F) Rainy Cool Normal TRUE
- G) Overcast Cool Normal TRUE



Add Instance h

Look at the instances:

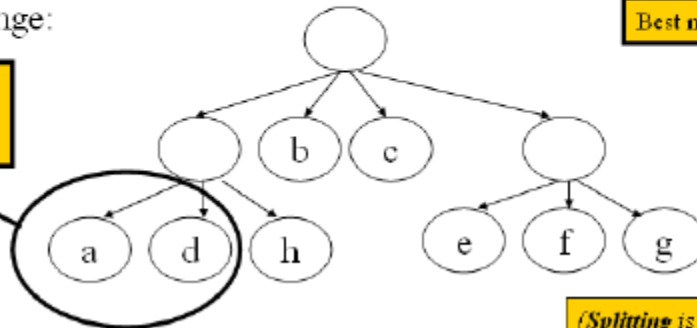
- A) Sunny Hot High FALSE
- D) Rainy Mild High FALSE
- H) Sunny Mild High FALSE

Runner up

Best matching node

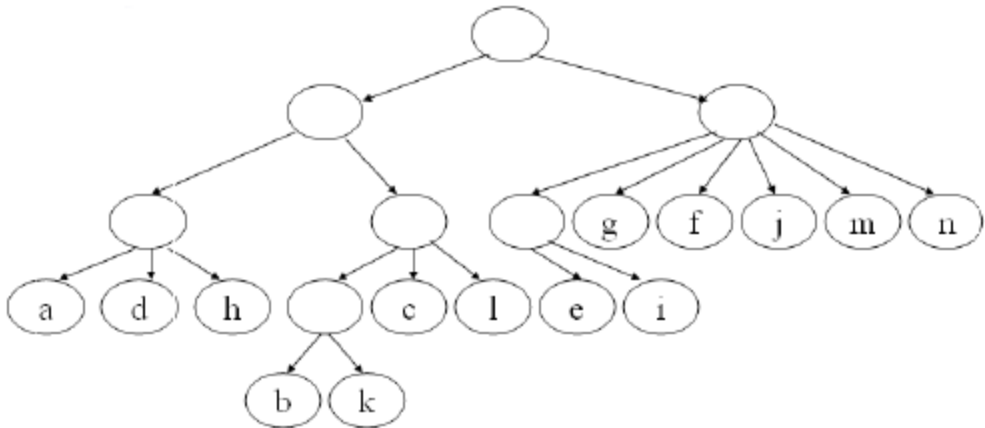
Rearrange:

Merged into a single cluster before h is added



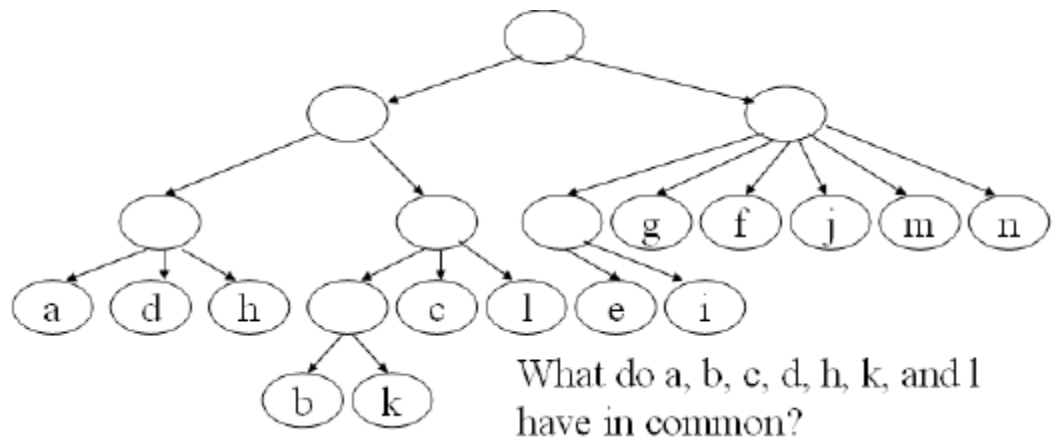
(Splitting is also possible)

Final Hierarchy



What next?

Dendrogram - >clusters



Mining Frequent Itemsets – Apriori Algorithm

Purpose:

- key concepts in mining frequent itemsets
- understand the Apriori algorithm
- run Apriori in Weka GUI and in programatic way

1. Theoretical aspects

In data mining, association rule learning is a popular and well researched method for discovering interesting relations between variables in large databases. Piatetsky-Shapiro describes analyzing and presenting strong rules discovered in databases using different measures of interestingness. Based on the concept of strong rules, Agrawal introduced association rules for discovering regularities between products in large scale transaction data recorded by point-of-sale (POS) systems in supermarkets. For example, the rule {onion,potatoes} \Rightarrow {burger} found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, he or she is likely to also buy burger. Such information can be used as the basis for decisions about marketing activities such as, e.g., promotional pricing or product placements. In addition to the above example from market basket analysis association rules are employed today in many application areas including Web usage mining, intrusion detection and bioinformatics.

In computer science and data mining, Apriori is a classic algorithm for learning association rules. Apriori is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequentation). Other algorithms are designed for finding association rules in data having no transactions (Winepi and Minepi), or having no timestamps (DNA sequencing).

Definition:

Following the original definition by Agrawal the problem of association rule mining is defined as:

Let $\mathbf{I} = \{i_1, i_2, \dots, i_n\}$ be a set of n binary attributes called items. Let $\mathbf{D} = \{t_1, t_2, \dots, t_n\}$ be a set of transactions called the *database*. Each transaction in \mathbf{D} has a unique transaction ID and contains a subset of the items in \mathbf{I} . A rule is defined as an implication of the form $\mathbf{X} \rightarrow \mathbf{Y}$ where $\mathbf{X}, \mathbf{Y} \subseteq \mathbf{I}$ and $\mathbf{X} \cap \mathbf{Y} = \emptyset$. The sets of items (for short *itemsets*) \mathbf{X} and \mathbf{Y} are called *antecedent* (left-hand-side or LHS) and *consequent* (right-hand-side or RHS) of the rule respectively.

To illustrate the concepts, we use a small example from the supermarket domain. The set of items is $\mathbf{I} = \{\text{milk}, \text{bread}, \text{butter}, \text{beer}\}$ and a small database containing the items (1 codes presence and 0 absence of an item in a transaction) is shown in the table below. An example rule for the supermarket could be {milk,bread} \Rightarrow {butter} meaning that if milk and bread is bought, customers also buy butter.

Note: this example is extremely small. In practical applications, a rule needs a support of several hundred transactions before it can be considered statistically significant, and datasets often contain thousands or millions of transactions.

Transaction ID	Milk	Bread	Butter	Beer
1	1	1	0	0
2	0	1	1	0
3	0	0	0	1
4	1	1	1	0
5	0	1	0	0
6	1	0	0	0
7	0	1	1	1
8	1	1	1	1
9	0	1	0	1
10	1	1	0	0
11	1	0	0	0
12	0	0	0	1
13	1	1	1	0
14	1	0	1	0
15	1	1	1	1

Useful Concepts

To select interesting rules from the set of all possible rules, constraints on various measures of significance and interest can be used. The best-known constraints are minimum thresholds on support and confidence.

Support

The support **supp(X)** of an itemset X is defined as the proportion of transactions in the data set which contain the itemset.

supp(X) = no. of transactions which contain the itemset X / total no. of transactions

In the example database, the itemset {milk,bread,butter} has a support of $4 / 15 = 0.26$ since it occurs in 26% of all transactions. To be even more explicit we can point out that 4 is the number of transactions from the database which contain the itemset {milk,bread,butter} while 15 represents the total number of transactions.

Confidence

The confidence of a rule is defined:

$$conf(X \rightarrow Y) = supp(X \cup Y) / supp(X)$$

For the rule {milk,bread} \Rightarrow {butter} we have the following confidence:

$$\text{supp}(\{\text{milk,bread,butter}\}) / \text{supp}(\{\text{milk,bread}\}) = 0.26 / 0.4 = 0.65$$

This means that for 65% of the transactions containing milk and bread the rule is correct. Confidence can be interpreted as an estimate of the probability $P(Y/X)$, the probability of finding the RHS of the rule in transactions under the condition that these transactions also contain the LHS.

Lift

The *lift* of a rule is defined as:

$$\text{lift}(X \rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(Y) * \text{supp}(X)}$$

The rule {milk,bread} \Rightarrow {butter} has the following lift:

$$\text{supp}(\{\text{milk,bread,butter}\}) / \text{supp}(\{\text{butter}\}) \times \text{supp}(\{\text{milk,bread}\}) = 0.26/0.46 \times 0.4 = 1.4$$

Conviction

The conviction of a rule is defined as:

$$\text{conv}(X \rightarrow Y) = \frac{1 - \text{supp}(Y)}{1 - \text{conf}(X \rightarrow Y)}$$

The rule {milk,bread} \Rightarrow {butter} has the following conviction:

$$1 - \text{supp}(\{\text{butter}\}) / 1 - \text{conf}(\{\text{milk,bread}\} \Rightarrow \{\text{butter}\}) = 1 - 0.46 / 1 - 0.65 = 1.54$$

The conviction of the rule $X \Rightarrow Y$ can be interpreted as the ratio of the expected frequency that X occurs without Y (that is to say, the frequency that the rule makes an incorrect prediction) if X and Y were independent divided by the observed frequency of incorrect predictions.

In this example, the conviction value of 1.54 shows that the rule {milk,bread} \Rightarrow {butter} would be incorrect 54% more often (1.54 times as often) if the association between X and Y was purely random chance.

2. Apriori algorithm

General Process

Association rule generation is usually split up into two separate steps:

1. First, minimum support is applied to find all frequent itemsets in a database.
2. Second, these frequent itemsets and the minimum confidence constraint are used to form rules.

While the second step is straight forward, the first step needs more attention.

Finding all frequent itemsets in a database is difficult since it involves searching all possible itemsets (item combinations). The set of possible itemsets is the power set over I and has size $2^n - 1$ (excluding the empty set which is not a valid itemset). Although the size of the powerset grows exponentially in the number of items n in I , efficient search is possible using the *downward-closure property* of support (also called anti-monotonicity) which guarantees that for a frequent itemset, all its subsets are

also frequent and thus for an infrequent itemset, all its supersets must also be infrequent. Exploiting this property, efficient algorithms (e.g., Apriori and Eclat) can find all frequent itemsets.

Apriori Algorithm Pseudocode

```

procedure Apriori (T, minSupport) { //T is the database and minSupport is the minimum support
  L1= {frequent items};
  for (k= 2; Lk-1 !=∅; k++) {
    Ck= candidates generated from Lk-1-1
    //that is cartesian product Lk-1 x Lk-1 and eliminating any k-1 size itemset that is not
    //frequent
    for each transaction t in database do{
      #increment the count of all candidates in Ck that are contained in t
      Lk = candidates in Ck with minSupport
    }//end for each
  }//end for
  return  $\bigcup_k L_k$ ;
}

```

As is common in association rule mining, given a set of *itemsets* (for instance, sets of retail transactions, each listing individual items purchased), the algorithm attempts to find subsets which are common to at least a minimum number C of the itemsets. Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as *candidate generation*), and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found.

Apriori uses breadth-first search and a tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length k - 1. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent k-length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.

Apriori, while historically significant, suffers from a number of inefficiencies or trade-offs, which have spawned other algorithms. Candidate generation generates large numbers of subsets (the algorithm attempts to load up the candidate set with as many as possible before each scan). Bottom-up subset exploration (essentially a breadth-first traversal of the subset lattice) finds any maximal subset S only after all $2^{|S|} - 1$ of its proper subsets.

3. Sample usage of Apriori algorithm

A large supermarket tracks sales data by Stock-keeping unit (SKU) for each item, and thus is able to know what items are typically purchased together. Apriori is a moderately efficient way to build a list

of frequent purchased item pairs from this data. Let the database of transactions consist of the sets {1,2,3,4}, {1,2,3,4,5}, {2,3,4}, {2,3,5}, {1,2,4}, {1,3,4}, {2,3,4,5}, {1,3,4,5}, {3,4,5}, {1,2,3,5}. Each number corresponds to a product such as "butter" or "water". The first step of Apriori is to count up the frequencies, called the supports, of each member item separately:

Item	Support
1	6
2	7
3	9
4	8
5	6

We can define a minimum support level to qualify as "frequent," which depends on the context. For this case, let min support = 4. Therefore, all are frequent. The next step is to generate a list of all 2-pairs of the frequent items. Had any of the above items not been frequent, they wouldn't have been included as a possible member of possible 2-item pairs. In this way, Apriori prunes the tree of all possible sets. In next step we again select only these items (now 2-pairs are items) which are frequent (the pairs written in bold text):

Item	Support
{1,2}	4
{1,3}	5
{1,4}	5
{1,5}	3
{2,3}	6
{2,4}	5
{2,5}	4
{3,4}	7
{3,5}	6
{4,5}	4

We generate the list of all 3-triples of the frequent items (by connecting frequent pair with frequent single item).

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

The algorithm will end here because the pair {2, 3, 4, 5} generated at the next step does not have the desired support.

We will now apply the same algorithm on the same set of data considering that the min support is 5.

We get the following results:

Step 1:

Item	Support
1	6
2	7
3	9
4	8
5	6

Step 2:

Item	Support
{1,2}	4
{1,3}	5
{1,4}	5
{1,5}	3
{2,3}	6
{2,4}	5
{2,5}	4
{3,4}	7
{3,5}	6
{4,5}	4

The algorithm ends here because none of the 3-triples generated at Step 3 have the desired support.

4. Sample usage of Apriori in Weka

For our test we shall consider 15 students that have attended lectures of the Algorithms and Data Structures course. Each student has attended specific lectures. The ARFF file presented below contains information regarding each student's attendance.

```
@relation test_studenti
```

```
@attribute Arbori_binari_de_cautare {TRUE, FALSE}
```

```
@attribute Arbori_optimali {TRUE, FALSE}
```

```

@attribute Arbori_echilibrati_in_inaltime {TRUE, FALSE}
@attribute Arbori_Splay {TRUE, FALSE}
@attribute Arbori_rosu_negru {TRUE, FALSE}
@attribute Arbori_2_3 {TRUE, FALSE}
@attribute Arbori_B {TRUE, FALSE}
@attribute Arbori_TRIE {TRUE, FALSE}
@attribute Sortare_topologica {TRUE, FALSE}
@attribute Algoritmul_Dijkstra {TRUE, FALSE}

```

```
@data
```

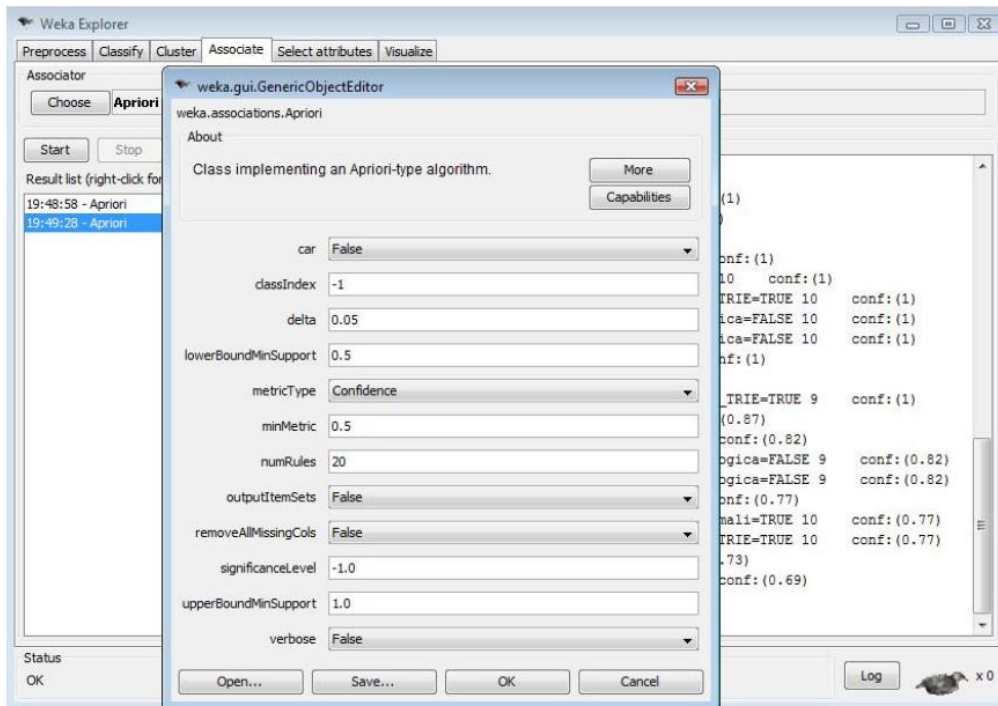
```

TRUE,TRUE,TRUE,TRUE,FALSE,FALSE,TRUE,TRUE,FALSE,FALSE
TRUE,TRUE,TRUE,TRUE,TRUE,TRUE,FALSE,TRUE,FALSE,FALSE
FALSE,TRUE,TRUE,TRUE,FALSE,FALSE,FALSE,TRUE,FALSE,TRUE
FALSE,TRUE,FALSE,FALSE,TRUE,FALSE,TRUE,TRUE,FALSE,TRUE
TRUE,TRUE,FALSE,TRUE,TRUE,FALSE,TRUE,TRUE,FALSE,TRUE
TRUE,FALSE,TRUE,FALSE,FALSE,TRUE,TRUE,TRUE,FALSE,FALSE
FALSE,TRUE,FALSE,TRUE,TRUE,FALSE,TRUE,TRUE,FALSE,TRUE
TRUE,FALSE,TRUE,TRUE,TRUE,FALSE,TRUE,TRUE,TRUE,FALSE
FALSE,TRUE,TRUE,TRUE,TRUE,FALSE,FALSE,TRUE,FALSE,FALSE
TRUE,FALSE,TRUE,FALSE,TRUE,TRUE,FALSE,TRUE,FALSE,TRUE
FALSE,FALSE,TRUE,FALSE,TRUE,FALSE,FALSE,TRUE,TRUE,TRUE
TRUE,FALSE,FALSE,TRUE,TRUE,TRUE,FALSE,TRUE,FALSE,TRUE
FALSE,TRUE,TRUE,FALSE,TRUE,TRUE,FALSE,TRUE,FALSE,TRUE
TRUE,TRUE,TRUE,FALSE,FALSE,TRUE,FALSE,TRUE,FALSE,FALSE
TRUE,TRUE,FALSE,FALSE,TRUE,TRUE,FALSE,TRUE,FALSE,FALSE

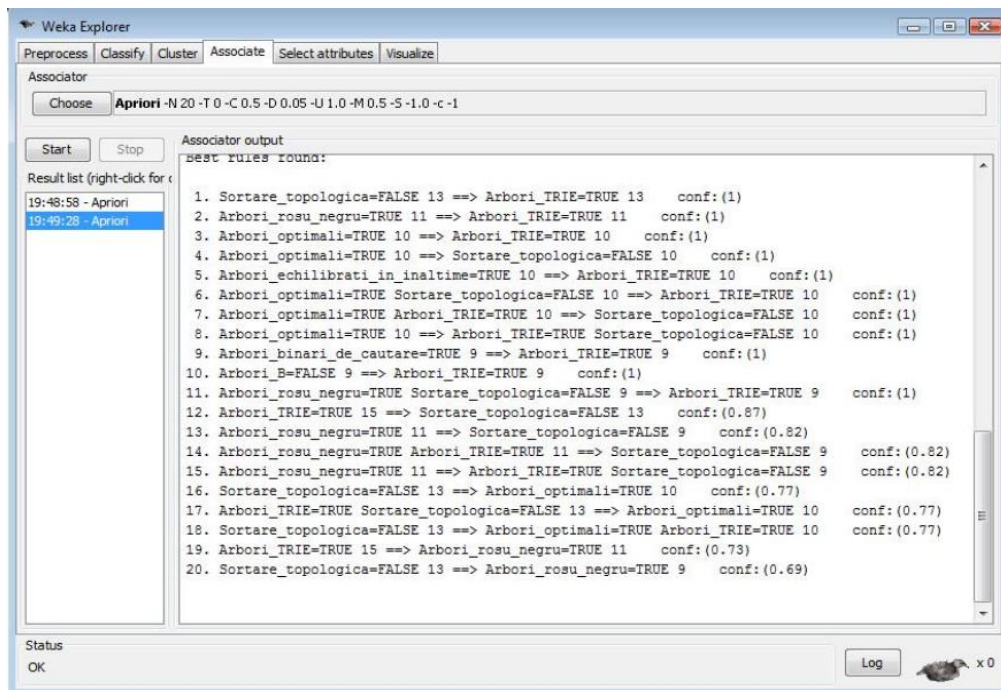
```

Using the Apriori Algorithm we want to find the association rules that have **minSupport=50%** and minimum confidence=50%. We will do this using WEKA GUI.

After we launch the WEKA application and open the *TestStudenti.arff* file, we move to the **Associate** tab and we set up the following configuration:



After the algorithm is finished, we get the following results:



If we look at the first rule we can see that the students who don't attend the *Sortare topologica* lecture have a tendency to attend the *Arbori TRIE* lecture. The confidence of this rule is 100% so it is very believable. Using the same logic we can interpret all the other rules that the algorithm has revealed.

The same results presented above can be obtained by implementing the WEKA Apriori Algorithm in your own Java code. A simple Java program that takes the *TestStudenti.arff* file as input, configures the Apriori class and displays the results of the Apriori algorithm is presented below:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

import weka.associations.Apriori;
import weka.core.Instances;

public class Main
{
    public static void main(String[] args)
    {
        Instances data = null;
        try{
            BufferedReader reader = new BufferedReader( new
                FileReader( "...\\TestStudenti.arff" ));
            data = new Instances(reader);
            reader.close();
            data.setClassIndex(data.numAttributes() - 1);
        }
        catch( IOException e ) {
            e.printStackTrace();
        }

        double deltaValue = 0.05;
        double lowerBoundMinSupportValue = 0.1;
        double minMetricValue = 0.5;
        int numRulesValue = 20;
        double upperBoundMinSupportValue = 1.0;
        String resultapriori;
        Apriori apriori = new Apriori();
        apriori.setDelta(deltaValue);
        apriori.setLowerBoundMinSupport(lowerBoundMinSupportValue);
        apriori.setNumRules(numRulesValue);
        apriori.setUpperBoundMinSupport(upperBoundMinSupportValue);
```

```

apriori.setMinMetric(minMetricValue);

try
{
    apriori.buildAssociations( data );
}
catch ( Exception e ) {
    e.printStackTrace();
}
resultapriori = apriori.toString();
System.out.println(resultapriori);
}
}

```

5. Domains where Apriori is used

Application of the Apriori algorithm for adverse drug reaction detection

The objective is to use the Apriori association analysis algorithm for the detection of adverse drug reactions (ADR) in health care data. The Apriori algorithm is used to perform association analysis on the characteristics of patients, the drugs they are taking, their primary diagnosis, co-morbid conditions, and the ADRs or adverse events (AE) they experience. This analysis produces association rules that indicate what combinations of medications and patient characteristics lead to ADRs.

Application of Apriori Algorithm in Oracle Bone Inscription Explication

Oracle Bone Inscription (OBI) is one of the oldest writing in the world, but of all 6000 words found till now there are only about 1500 words that can be explicated explicitly. So explication for OBI is a key and open problem in this field. Exploring the correlation between the OBI words by Association Rules algorithm can aid in the research of explication for OBI. Firstly the OBI data extracted from the OBI corpus are preprocessed; with these processed data as input for Apriori algorithm we get the frequent itemset. And combined by the interestingness measurement the strong association rules between OBI words are produced. Experimental results on the OBI corpus demonstrate that this proposed method is feasible and effective in finding semantic correlation for OBI.