

**THE IMPLEMENTATION OF A MULTIMEDIA DATA MINING
TOOL**

by

Ayşe Nur TAŞLIPINAR

A thesis submitted to
the Graduate Institute of Sciences and Engineering

of

Fatih University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

July 2007
Istanbul, Turkey

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Bekir KARLIK
Head of Department

This is to certify that I have read this thesis and that in my opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Atakan KURT
Supervisor

Examining Committee Members

Assist. Prof. Dr. Atakan KURT

Assoc. Prof. Dr. Onur TOKER

Assist. Prof. Dr. Zeynep ORHAN

It is approved that this thesis has been written in compliance with the formatting rules laid down by the Graduate Institute of Sciences and Engineering.

Assist. Prof. Dr. Nurullah ARSLAN
Director

Date
July 2007

THE IMPLEMENTATION OF A MULTIMEDIA DATA MINING TOOL

Ayşe Nur TAŞLIPINAR

M. S. Thesis - Computer Engineering
July 2007

Supervisor: Assist. Prof. Dr. Atakan KURT

ABSTRACT

For beginner users like students of computer science current data mining software are very sophisticated. A lightweight tool that provides an easy to use environment for the students who are learning data mining and multimedia is beneficial for their education.

In this thesis, we developed a tool that provides a simple interface for image processing and a way to running data mining algorithms on image files which will be quite valuable in data mining and image processing courses. This tool provides the functionalities of manipulation and filtering of image files, conversion of image files to a format suitable for data mining, and application of several data mining algorithms on these groups of files.

Keywords: Multimedia, data mining, image processing, JAI, Weka

MULTİMEDYA VERİ MADENCİLİĞİ ARACI

Ayşe Nur TAŞLIPINAR

Yüksek Lisans Tezi – Bilgisayar Mühendisliği
Temmuz 2007

Tez Yöneticisi: Yrd. Doç Dr. Atakan KURT

ÖZ

Bilgisayar bilimi öğrencileri gibi yeni başlayanlar için günümüzde mevcut olan veri madenciliği yazılımları oldukça kompleks bir yapıya sahiptir. Veri madenciliği ve multimedya öğrencilerine kullanımı kolay bir ortam sağlayan sade bir araç eğitimleri için faydalıdır.

Bu tez çalışmasında, veri madenciliği ve görüntü işleme derslerinde oldukça faydalı olacak, görüntü dosyaları üzerinde görüntü işleme ve veri madenciliği algoritmalarını çalıştırma yöntemi sağlayan bir araç geliştirildi. Bu araç; görüntü dosyalarının filtre edilmesi ve işlenmesi, görüntü dosyalarının veri madenciliği için uygun formata dönüştürülmesi ve bu dosya grupları üzerinde çeşitli veri madenciliği algoritmalarının uygulanması işlevlerini sağlamaktadır.

Anahtar Kelimeler: Multimedya, veri madenciliği, görüntü işleme, JAI, Weka

ACKNOWLEDGEMENT

I express sincere appreciation to Assist. Prof. Dr. Atakan KURT and for his guidance and insight throughout the research.

My sincere thanks to the committee member Assist. Prof. Dr. Zeynep Orhan for her support and motivation.

I am also grateful to my family and my friends for their understanding, motivation and patience.

TABLE OF CONTENTS

ABSTRACT.....	iii
ACKNOWLEDGEMENT	v
TABLE OF CONTENTS	vi
LIST OF TABLES	viii
LIST OF FIGURES.....	ix
LIST OF SYMBOLS AND ABBREVIATIONS	x
CHAPTER 1 INTRODUCTION	1
CHAPTER 2 BACKGROUND	3
2.1 PREVIOUS WORK.....	3
2.2 IMAGE PROCESSING WITH JAVA.....	6
2.2.1 Java AWT Model	7
2.2.2 Java 2D Imaging Model.....	8
2.2.3 Java Advanced Imaging Model.....	11
2.3 DATA MINING WITH WEKA	12
2.3.1 Introducing WEKA.....	12
2.3.2 ARFF file format	13
CHAPTER 3 PROGRAM USAGE.....	16
3.1 TABLE FORMAT OF MDT.....	17
3.2 XML FILE FORMAT OF MDT	19
3.3 USAGE SCENARIOS	21

3.3.1 Image Dataset Preparation	21
3.3.2 Table Creation	23
3.3.3 Image Processing Example	25
3.3.4 Data Mining Example	27
CHAPTER 4 IMPLEMENTATION	34
4.1 INTRODUCTION TO MDT CLASS STRUCTURE	34
4.2 MAINWINDOW CLASS	36
CHAPTER 5 CONCLUSION	41
REFERENCES	43

LIST OF TABLES

Table 2.1 Basic imaging classes of java AWT	7
Table 2.2 Basic imaging classes in Java 2D	9
Table 2.3 Filtering classes in Java 2D	10
Table 2.4 Basic JAI imaging classes	11
Table 3.1 Columns of MDT table	18
Table 4.1 Basic classes of MDT	35
Table 4.2 Methods of MainWindow class	37

LIST OF FIGURES

Figure 2.1 Screen from WEKA.....	13
Figure 2.2 Sample ARFF file.....	14
Figure 3.1 XML file format of MDT	20
Figure 3.2 Grayscale images of subjects in different poses.....	22
Figure 3.3 Creating a table.....	23
Figure 3.4 Adding row using menu.....	24
Figure 3.5 Importing image from explorer.....	25
Figure 3.6 Applying filter to an instance	26
Figure 3.7 Result of filtering operation	26
Figure 3.8 Preparing the dataset for data mining	27
Figure 3.9 Choosing classification algorithm	28
Figure 3.10 Generic Object Editor of J48 tree.....	29
Figure 3.11 Data mining window for classification.....	30
Figure 3.12 Results of the classification.....	30
Figure 3.13 Selecting clustering algorithm.....	31
Figure 3.14 Generic Object Editor of EM algorithm	31
Figure 3.15 Data mining window for clustering.....	32
Figure 3.16 Results of the clustering.....	33
Figure 4.1 Function loadAtable().....	38
Figure 4.2 Function addColumn().....	39
Figure 4.3 Function deleteColumn()	39
Figure 4.4 Function deleteRow()	40
Figure 4.5 Function saveTable()	40

LIST OF SYMBOLS AND ABBREVIATIONS

SYMBOL/ABBREVIATION

ARFF	Attribute Relation File Format
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
AWT	Abstract Window Toolkit
CLI	Command Line Interpreter
JAI	Java Advanced Imaging
JPEG	Joint Photographic Experts Group
GIF	Graphics Interchange Format
MDT	Multimedia Data Mining Tool
PNG	Portable Network Graphics
SAX	Simple API for XML
WEKA	Waikato Environment for Knowledge Analysis
XML	Extensible Markup Language
2D	Two dimensional

CHAPTER 1

INTRODUCTION

Demand of knowledge discovery from image files has rapidly grown in past few years. Current data mining software are very sophisticated and not all of them provide features for working on image files. For beginner users like computer science students it can be very complicated to figure out how to use these tools during their learning periods. A lightweight tool is beneficial for data mining and image processing courses that enable users to manage and apply data mining on image files easily. In this manner, an application that provides a simple interface for image processing and running data mining algorithms on image files will be quite valuable for data mining and image processing education.

“Multimedia data mining is a subfield of data mining that deals with the extraction of implicit knowledge, multimedia data relationships, or other patterns not explicitly stored in multimedia databases.” (Zaiane et al., 1998) Multimedia file types include text, image, audio and video. Since gathering knowledge from image data is mostly in demand and easier to handle than video, most of the multimedia data mining research are being done on image data. For these purposes together with the vision of simplicity, our study will only focus on image file type.

In this study a computer application called Multimedia Data Mining Tool (MDT) which provides simple filtering and data mining of image files is implemented. The tool has a plain but practical user interface which allows creating and managing image datasets. Besides, it provides the datasets to be stored in and retrieved from specially designed document format which is based on the Extensible Markup Language (XML).

One of the prominent features of the MDT is its plain table data structure. Based on this structure, converters for the two different file formats, XML and Attribute Relation File Format (ARFF) are developed in order to use table data. Besides, MDT provides simple filters and operators for image processing. Previewing is supported in this step. Further, MDT forms a link to the data mining algorithms in Waikato Environment for Knowledge Analysis (WEKA). By the way, “Weka is a collection of state-of-the-art machine learning algorithms and data preprocessing tools” (Witten and Frank, 2005). In the data mining step, classification and clustering algorithms can be applied and the results are output to the screen. “Image classification and clustering are the supervised and unsupervised classification of images into groups.” (Hsu et al., 2002). However, the image classification and clustering methods we used are very different from the advanced ones. What we used here is pixel based classification.

This thesis is organized as follows: Chapter 2 gives information about the previously implemented systems that handles multimedia data, and then it presents the key points of the two technologies, Java Advanced Imaging Application Programming Interface (JAI API) and Weka Data Mining Tool, which are used to develop the application. Chapter 3 illustrates a usage scenario of MDT. Chapter 4 presents detailed information of MDT’s basic classes. Chapter 5 discusses future work and concludes the study.

CHAPTER 2

BACKGROUND

The two most important capabilities that The Multimedia Data Mining Tool (MDT) provides are the ability to do image processing and data mining on image files. This chapter includes general background information by first presenting previous work done in the area of multimedia data mining and then the two basic technologies that provides means for these capabilities in several sections. In section 2.1, some significant works in the interested area are summarised. In section 2.2, the Java Media API which is used for handling image processing part is explained. In section 2.3, the Weka data mining tool which provides algorithms for data mining process is explained.

2.1 PREVIOUS WORK

“Data mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner” (Hand et al., 2001). Simply, it is often described as finding hidden information from data.

Multimedia is referred to any type of information medium that can be captured, represented, processed, communicated and stored in digital form. Multimedia data means any type of text, image, video, audio, etc files. From this point of view, multimedia mining can be defined as a subfield of data mining that deals with the extraction of implicit patterns, relationships, or facts from the multimedia data.

Grosky and Tao identify multimedia data mining and discuss a representation design for multimedia objects used for mining (Grosky and Tao, 1998).

They make a classification of multimedia data mining and present a data model over existing models (Gudivada et. al., 1996).

Although image mining and video mining are considered as different subfields of multimedia data mining, same type of input data -image- is being considered for similar operations and mining techniques. The most important research areas in image and video mining are listed as pattern recognition, content-based image retrieval, video retrieval, video sequence analysis, change detection, training for object recognition, and image model learning. All of these areas have two common problems that processing of large samples of images and extraction of semantically meaningful information for the user (Missaoui and Palenichka, 2005).

The emergence of multimedia data mining concept influenced the related areas of databases and information systems. It led to many applications of knowledge discovery, information analysis and retrieval, content based searching of multimedia information. This triggered the implementations of many multimedia data mining systems. Some of the most significant systems will be explained briefly. For instance, not all of these systems can be considered as data mining systems but they are specific examples for drawing a picture about the usage and retrieval of multimedia data.

One of the leading researches is the MultiMediaMiner system. (Zaiane et al., 1998) which is defined as a prototype for multimedia data mining system designed for mining information from large multimedia databases. It provided an interactive mining interface and display for image and video data. In implementation, DBMiner (Han et. al., 1997) and content based image retrieval (CBIRD) (Gudivada and Raghavan, 1995) systems are extended to handle multimedia data for knowledge discovery purposes. One feature of the MultiMediaMiner is that it includes the multimedia data cubes which provide a multi dimensional analysis for multimedia data. The underlying CBIRD system is composed of four major components:

- (a) an image excavator for gathering images and videos from the multimedia database;
- (b) a preprocessor that extracts features of image and stores precomputed data;
- (c) a search kernel that matches queries with image and video features;

- (d) some discovery modules (characterizer, classifier and associator) that applies image mining methods to extract knowledge and patterns.

Another feature of MultimediaMiner is that it includes some data mining modules which act as, characterizer, classifier, and associator, in image and video databases. These are named as MM-Characterizer, MM-Classifer, and MM-Associator respectively.

There are many systems implemented to mine scientific data. Diamond Eye (Roden et al, 1999) is a scientific image mining system that enables users to gather images from large image databases. Diamond Eye system uses Java client-server architecture. The server maintains an object oriented database that stores client data, query models, and results and provide query processing. Inside the system there are several image processing and object recognition algorithms that enables searching and retrieving image content. The client side tools enables image search queries and analysis of query results. Query results are displayed as thumbnail images. Metadata elements can be added to Image class which enables metadata operations for developers.

Another scientific toolkit is Algorithm Development and Mining (ADaM) (Rushing et. al, 2005) which provides data mining and image processing capabilities together. ADaM toolkit provides pattern recognition, image processing, optimization and data mining together with many supporting data mining tools. It provides common data mining methods which are classification, clustering, association rule mining, and data preprocessing. There are tools to extract features from images, conversion of image from image data pattern vector form and vice versa. The toolkit is a combination of many independent components. ADaM is interoperable with WEKA at the data level because it uses the same data format of ARFF.

There are also example researches that focus on video retrieval. An interactive system called VideoQ is defined as the first on-line advanced content-based video searching system (Chang et al, 1998). Its significant features are listed as:

- (a) automatic segmentation and tracking of video objects;
- (b) a rich library of visual features ;

- (c) making queries with multiple objects;
- (d) making queries with the spatio-temporal constraints.

The system allows users to do video search using a large set of visual features and spatio-temporal relationships. The query is formulated by so called animated sketch in which each object is assigned with motion and temporal duration attributes in addition to the usual shape, color and texture attributes. Segmentation and tracking is done automatically and different features of the object like color, shape, texture, etc. are stored in the feature libraries. When query is processed, it is matched with these features and then candidate video shots are listed.

Another interesting implementation is The Informedia Digital Video Library project (Wactlar et. al, 1996) developed in Carnegie Mellon University. Informedia is also an on-line digital video library that supports content based searching and retrieval. The system includes a speech recognizer that transcribes video soundtracks, and a structure that understands and stores the transcript in a text database. The database allows making queries in which the user can specify the words in the soundtrack.

Image mining trends are rapidly advancing. Hsu et. al. examines research issues in this area and presents some future directions for image mining.(Hsu et al, 2002). Hsu et. al. classified image mining reseaches to two dimensions .First one is extraction of most relevant features(Fayyad et al.,1996; Hsu et al.,2000 ;Kitamoto, 2001), second is generating image patterns for understanding.(Ordonez and Omiecinzki, 1999; Zaiane et al.,1998).

2.2 IMAGE PROCESSING WITH JAVA

MDT provides simple image processing features. The image processing operations are implemented using the latest Java Imaging classes which are known as Java Advanced Imaging Application Programming Interface (JAI API).

In order to understand the structure of the Java Media API, essential steps in java image processing should be examined. This section will provide the progressive

development of image processing in Java until the introduction of Java Advanced Imaging Model. Only the image processing parts of the APIs will be focused on.

The evolution of image processing in Java is considered as three main steps: First is the imaging part of Abstract Window Toolkit (AWT) API, which provides the basic simple rendering package. Second is the Java 2D API, which is an extension to the early AWT and added support for many more graphics and rendering operations. Third is the JAI API which advanced image processing by providing elaborate classes and high performance processing capabilities.

2.2.1 Java AWT Model

AWT supports image processing by providing the `java.awt` and `java.awt.image` class packages. Basic AWT imaging interfaces and classes with their definitions are listed in Table 2.1. All class declarations are official from Sun Java website.

Table 2.1 Basic imaging classes of java AWT

Class	Description
Image	The abstract class Image is the superclass of all classes that represent graphical images.
ImageConsumer	The interface for objects expressing interest in image data through the ImageProducer interfaces.
ImageObserver	An asynchronous update interface for receiving notifications about Image information as the Image is constructed.
ImageProducer	The interface for objects which can produce the image data for Images.
ColorModel	The ColorModel abstract class encapsulates the methods for translating a pixel value to color components (for example, red, green, and blue) and an alpha component.
PixelGrabber	The PixelGrabber class implements an ImageConsumer which can be attached to an Image or ImageProducer object to retrieve a subset of the pixels in that image.

Image processing in AWT is based on the concept of filtering pipeline of image producers and consumers. An image object of `java.awt.Image` abstract class may be created either by loading from an image source or being drawn by an AWT component. The Image object does not provide the actual image data; it only provides a means for processing it. To process an image, an object implementing the ImageProducer interface

should send image data to an object implementing the ImageConsumer interface. Filter objects implement both interfaces. The AWT imaging model is called as Push Model because the consumer cannot request the Image, image data is pushed into the image processing pipeline by the ImageProducer object when it is needed.

AWT provides a few simple filters, for cropping and color manipulation operations. Basic filters class is ImageFilter base class. After the filtering operation, the resulting consumer image which is also an AWT Image object can be drawn upon the screen by obtaining a Graphics object.

In AWT, image data is stored as array of bytes. When an image is loaded, the Image object is obtained with encapsulated pixel data inside. There is not a mechanism for reusable persistent memory storage of image pixels. Pixel data can be extracted by using the PixelGrabber class. The ColorModel accompanying the image data describes the layout and interpretation of the pixels.

AWT supports width and height properties of image and a very limited number of image input file types, such as GIF, JPEG, and later added PNG.

There are many deficiencies of the AWT imaging model that makes it insufficient for high level image processing. These may be listed as the absence of permanent image data, the abstractions in the push model, poor filtering capabilities, the insufficiency of image data formats, and the lack of some common concepts for extensive image processing.

2.2.2 Java 2D Imaging Model

The Java 2D API introduced special classes that extend the Java AWT classes to support for two-dimensional imaging operations. These classes are merged into AWT and became a part of the Java Core with the Java Platform 1.2 release. They also formed the foundation of the Java Advanced Imaging API.

Basic image-handling interfaces and classes which are part of Java 2D are listed in Table 2.2. All class declarations are official from Sun Java website.

Table 2.2 Basic imaging classes in Java 2D

Class/ Interface	Description
RenderedImage	RenderedImage is a common interface for objects which contain or can produce image data in the form of Rasters.
WritableRenderedImage	WritableRenderedImage is a common interface for objects which contain or can produce image data in the form of Rasters and which can be modified and/or written over.
BufferedImage	The BufferedImage subclass describes an Image with an accessible buffer of image data.
ComponentColorModel	A ColorModel class that works with pixel values that represent color and alpha information as separate samples and that store each sample in a separate data element.
ComponentSampleModel	This class represents image data which is stored such that each sample of a pixel occupies one data element of the DataBuffer.
DataBuffer	This class exists to wrap one or more data arrays.
FilteredImageSource	This class is an implementation of the ImageProducer interface which takes an existing image and a filter object and uses them to produce image data for a new filtered version of the original image.
ImageFilter	This class implements a filter for the set of interface methods that are used to deliver data from an ImageProducer to an ImageConsumer.
Kernel	The Kernel class defines a matrix that describes how a specified pixel and its surrounding pixels affect the value computed for the pixel's position in the output image of a filtering operation.
Raster	A class representing a rectangular array of pixels.
SampleModel	This abstract class defines an interface for extracting samples of pixels in an image.
WritableRaster	This class extends Raster to provide pixel writing capabilities.
RenderableImage	A RenderableImage is a common interface for rendering-independent images (a notion which subsumes resolution independence).
ParameterBlock	A ParameterBlock encapsulates all the information about sources and parameters (Objects) required by a RenderableImageOp, or other classes that process images.
RenderableImageOp	This class handles the renderable aspects of an operation with help from its associated instance of a ContextualRenderedImageFactory.
RenderableImageProducer	An adapter class that implements ImageProducer to allow the asynchronous production of a RenderableImage.
RenderContext	A RenderContext encapsulates the information needed to produce a specific rendering from a RenderableImage.

The imaging model of Java 2D API is based on the producer/consumer model of AWT, additionally, it provides the permanent image data in memory. The image processing model of Java 2D is called the immediate mode imaging model, since it makes the entire image data available in memory immediately after each step in the imaging pipeline.

Primary image class is `java.awt.image.BufferedImage`, which represents an area of memory containing pixel data. A `BufferedImage` consists of a `DataBuffer`, which stores actual pixel data as one or more arrays of primitive datatypes, and associated with it a `SampleModel`, and a `ColorModel` to read and write the data. `SampleModel` deals with grouping the numbers into pixels and `ColorModel` deals with conversion of pixels to colors.

Filtering classes provided by Java 2D supports many image processing operations like blurring, sharpening, geometric transformation, rotating, scaling, thresholding, etc. Imaging operations can be done directly to `BufferedImage` using these filters. Some of these are listed in Table 2.3. All class declarations are official from Sun Java website.

Table 2.3 Filtering classes in Java 2D

Class/ Interface	Description
<code>AffineTransformOp</code>	This class uses an affine transform to perform a linear mapping from 2D coordinates in the source image or <code>Raster</code> to 2D coordinates in the destination image or <code>Raster</code> .
<code>BandCombineOp</code>	This class performs an arbitrary linear combination of the bands in a <code>Raster</code> , using a specified matrix.
<code>ColorConvertOp</code>	This class performs a pixel-by-pixel color conversion of the data in the source image.
<code>ConvolveOp</code>	This class implements a convolution from the source to the destination.
<code>RescaleOp</code>	This class performs a pixel-by-pixel rescaling of the data in the source image by multiplying the sample values for each pixel by a scale factor and then adding an offset.

The Java 2D API provides device-independent rendering by introducing `Renderable` and `Rendered` interfaces. By the way, rendering an image means to transform it to an appropriate format for an output device. Since rendering independence concept is essential to JAI, these interfaces will be briefly explained.

Renderable and Rendered layers are two integrated imaging layers. The renderable layer is rendering-independent layer that provides reusable image sources for different contexts and operators that take rendering-independent parameters. In contrast, the Rendered Layer provides context-specific image sources and operators that take context-dependent parameters.

2.2.3 Java Advanced Imaging Model

“The Java Advanced Imaging (JAI) API further extends the Java platform (including the Java 2D API) by allowing sophisticated, high-performance image processing to be incorporated into Java programming language applets and applications” (Prasad, 2002). JAI provides many image operators and an extension mechanism for developing additional operators. Basic JAI classes are listed in Table 2.4. All class declarations are official from Sun Java website.

Table 2.4 Basic JAI imaging classes

Class	Description
CollectionImage	An abstract superclass for classes representing a Collection of images.
JAI	A convenience class for instantiating operations.
OpImage	This is the base class for all image operations.
PlanarImage	A RenderedImage is expressed as a collection of pixels.
RenderableOp	A node in a renderable imaging chain.
RenderedOp	A node in a rendered imaging chain.
TiledImage	A concrete implementation of WritableRenderedImage.

Imaging model of JAI, called as pull model, allows image processing when needed. The image object waits for a request to pass the pixel data. The model is based on Renderable and Rendered layer concepts introduced in Java 2D. Both the images and

operators are objects. The operator object defined with image source(s) and parameters. Then, they can be linked to form chains.

The top level image class is `javax.media.jai.PlanarImage`. Besides, previous `SampleModel`, `ColorModel`, `DataBuffer`, and `Raster` classes are extended.

In JAI, an image processing operation is progressed through four steps: firstly, gathering the image either by loading from a file or a data source, or by creating internally, secondly describing the imaging graph by defining the operators and their relationships, thirdly, evaluating the result through using one of the rendered, renderable or remote execution models, and fourthly, processing the resultant image either by saving it to a file or sending to a device or API.

2.3 DATA MINING WITH WEKA

“The Weka Workbench is a collection of state-of-the-art machine learning algorithms and data preprocessing tools” (Witten and Frank, 2005). It provides various methods for common data mining problem areas which are classification, clustering, regression, association rule mining, and attribute selection together with a powerful user interface. “The philosophy behind WEKA is to move away from supporting a computer science or machine learning researcher, and towards supporting the end user of machine learning” (Holmes et. al., 1995).

2.3.1 Introducing WEKA

Weka provides a fully comprehensive environment for the total data mining process. In addition to various data mining algorithms, it provides capabilities for data preprocessing i.e. preparation, evaluation and visualization. Its powerful user interface combines all of its capabilities.

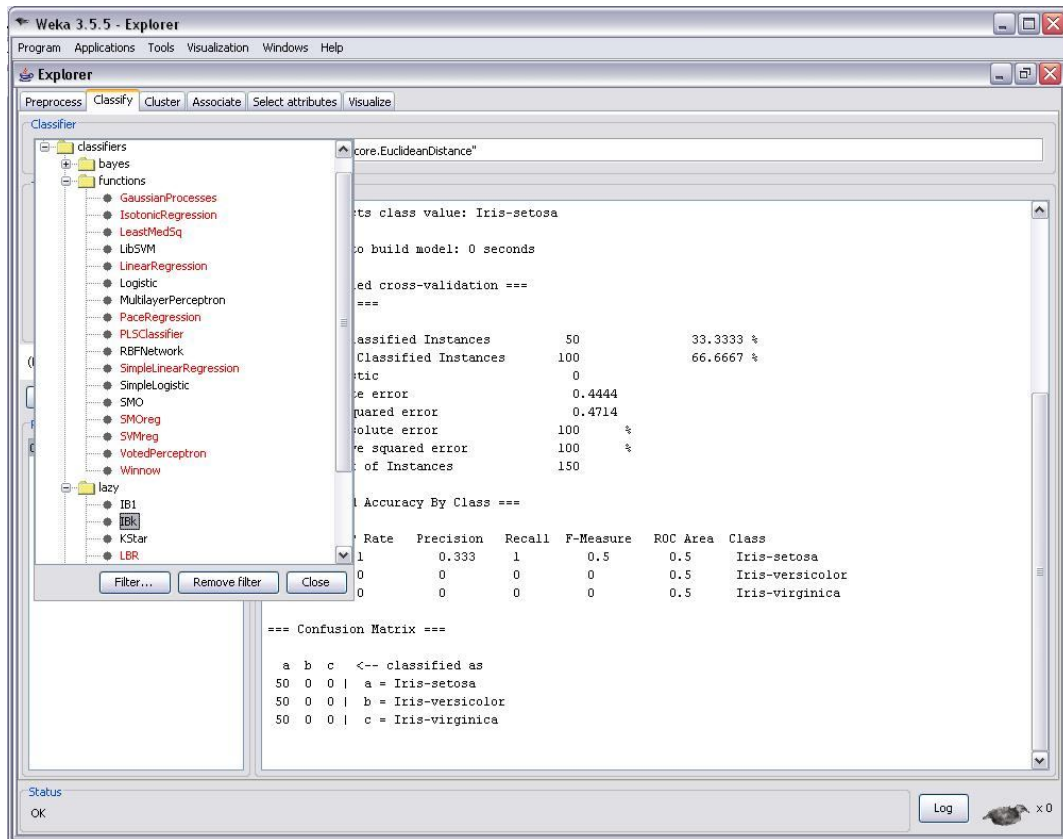


Figure 2.1 Screen from WEKA

The graphical user interface presents four interfaces according to usage objective. First and the mostly used one is the Explorer. Figure 2.1 shows a screen view of the Explorer. Second is the Knowledge Flow interface that allows the user to design configurations for incremental algorithms. The third one is the Experimenter for advanced users who need to make comparison of various learning techniques and different parameter settings. The last one is a simple command-line interface that allows to run data mining commands from the command line. In MDT, we used the concept of command line interface. We focused on generating the appropriate command that will be sent as input to the classes of command line interface.

2.3.2 ARFF file format

Tables stored in XML format is also converted to Weka's ARFF file format for data mining process. Data Mining is done on ARFF set files.

In Weka homepage, the definition is “An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes.” A sample which is taken from our tool output arff file is shown in the figure Figure 2.2

```

% Here comes comments
%

@RELATION myTable

@ATTRIBUTE column1 NUMERIC
@ATTRIBUTE column2 NUMERIC
@ATTRIBUTE column3 NUMERIC
@ATTRIBUTE column4 { yes, no }
...
@ATTRIBUTE CLASS { class1, class2, class3 }

@DATA
7,7,7,yes,.....,75,75,78,78,78,class1
6,6,6,no,.....,87,87,79,79,79,class2
0,0,0,no,.....,5,5,7,7,7,class3
3,3,3,no,.....,2,2,3,3,3,class2
0,0,0,yes,.....,3,3,7,7,7,class3
7,7,7,no,.....,76,76,79,79,79,class1
.....

```

Figure 2.2 Sample ARFF file

An ARFF file consists of two sections. The header section is the information between @RELATION and @DATA declarations. This part is the Header information, which contains the declarations of the relation, and the attributes with their types. The data section starts with @DATA declaration until the end of the document. This part includes the actual instances as each line.

The percent sign indicates the beginning of a comment line. If spaces will be used in the user specified strings like relation name, attribute name, data values, etc, then the entire string must be quoted.

The name of the relation (table) is written just next to the @relation declaration. This forms the first line in the document.

Attribute declarations consists of an ordered sequence of @attribute statements. Each attribute declaration is given by its name and data type. The order of the attribute

declarations is maintained in the data section as column positions. The attribute name must start with an alphabetic character. The data type of the attribute can be any of the numeric, nominal, string, or date types. For MDT implementation, only numeric and nominal types are used. Numeric attributes can be real or integer numbers. Nominal values are written as a comma separated list inside curly braces, as shown in the figure.

The @data declaration is a single line denoting the start of the data segment. Each instance is listed below the @data declaration, on a single line with carriage returns denoting the end of its values. 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.

CHAPTER 3

PROGRAM USAGE

Multimedia Data Mining Tool (MDT) is designed to fulfill the need of a simple tool for learning common image processing and data mining operations. In the evaluation framework designed by Collier et al., the user interface criteria is one of the most significant measures which asks if it is easy to navigate and uncomplicated (Collier et, al.,1999). MDT aims this ease of use criteria for educational purposes. MDT also enables users to apply JAI filters on image files and run Weka's data mining algorithms on image file datasets.

This tool will provide the functionalities of manipulation and filtering of image files, conversion of XML dataset files to Weka's arff format files suitable for data mining, and application of data mining tasks on these groups of files. We need to provide a set of data sets.

MDT includes a powerful user interface which is developed in Eclipse environment using Java Programming Language. The interface provides the user the ability to open, create, edit, or save tables which hold image datasets. Functionalities of MDT classes include extracting image files' properties and pixel data, applying imaging operations on image files, writing to and reading from XML tables, writing to Weka's ARFF data format, calling data mining algorithms of Weka with the help of Weka's Simple CLI classes.

Table format of MDT will be explained in section 3.1. Section 3.2 presents the properties of specially designed XML file format, and section 3.3 builds up a usage scenario.

3.1 TABLE FORMAT OF MDT

Table format in MDT is specially designed to hold the image files with their properties. The clear and informative format provides rich information about each image with their preview so enables user to easily select the images that will be added to the dataset.

Each image file forms one row of the table while properties of the image shown in separate columns. Columns are listed in Table 3.1.

Table 3.1 Columns of MDT table

Column Name	Description
preview	Thumbnail preview of the image
filename	Name of the image file
originalURL	Full path of the image file
fileType	File type of the image
fileSize	Size of the image file
width	Image width in pixels
height	Image height in pixels
tileInfo	Tile dimensions, if there is more than one tile
tileBound	Tile bounds, if image is tiled
tileGridOffset	Tile offset, if image is tiled
numberOfBands	Number of bands of the image, extracted from samplemodel
dataType	Data type of the image, extracted from sample model
colorMapType	Colormap type extracted from the color model
numOfComponents	Number of color components
bitsPerPixels	Bits per pixel components in the color model
transparency	Transparency information
class	Class value used for data mining

3.2 XML FILE FORMAT OF MDT

A sample file is shown in Figure 3.1. It can be seen that the <table> tag is the root element. It represents the dataset table. The table is considered as two parts. The only two child elements are <schema> which contains table structure information and <data> which contains actual data.

Schema consists of the properties of the table columns. Each table column corresponds to an <attribute> element. Name, data type, size and visual order of the columns are listed as attribute values of the <attribute> element. Attributes other than “name” are given default values; they are not actively used in implementation. Column is given this name because of the data mining terminology; each column is called as attributes. A column may either be one of default columns or added by user. The <builtinattributes> and <userattributes> elements indicate this.

Data consists of the table rows each of which represent an instance (image file). Each <row> element represents one instance of the table. Actual data is listed between column name tags this corresponds to a cell value.

```

<?xml version="1.0" encoding="UTF-8"?>
<table name="myTable" creationdate="06 07 2007" modificationdate="06 07 2007">
  <schema>
    <builtinattributes>
      <attribute name="preview" datatype="binary" size="3" visualorder="1" />
      <attribute name="fileName" datatype="string" size="50" visualorder="2" comment="..."/>
      <attribute name="originalURL" datatype="string" size="255" visualorder="3" />
      <attribute name="fileType" datatype="string" size="50" visualorder="4" />
      <attribute name="fileSize" datatype="string" size="50" visualorder="5" />
      <attribute name="width" datatype="integer" size="5" visualorder="6" />
      <attribute name="height" datatype="integer" size="5" visualorder="7" />
      <attribute name="tileInfo" datatype="integer" size="3" visualorder="8" />
      <attribute name="tileBounds" datatype="boolean" size="1" visualorder="9" comment="..."/>
      <attribute name="tileGridOffset" datatype="string" size="3" visualorder="10" />
      <attribute name="numberOfBands" datatype="string" size="3" visualorder="11" />
      <attribute name="dataType" datatype="integer" size="3" visualorder="12" />
      <attribute name="colorMapType" datatype="boolean" size="1" visualorder="13" comment="..."/>
      <attribute name="numOfComponents" datatype="string" size="3" visualorder="14" />
      <attribute name="bitsPerPixel" datatype="string" size="50" visualorder="15" />
      <attribute name="transparency" datatype="string" size="50" visualorder="16" />
      <attribute name="CLASS" datatype="string" size="50" visualorder="17" />
    </builtinattributes>
    <userattributes>
      <attribute name="grayscale" datatype="string" size="3" visualorder="18"/>
    </userattributes>
  </schema>
  <data>
    <row>
      <builtinattributes>
        <preview>subject1_1.bmp</preview>
        <filename>subject1_1.bmp</filename>
        <originalURL>D:\DataSETS\dataset1\subject1_1.bmp</originalURL>
        <fileType>bmp</fileType>
        <fileSize>2102</fileSize>
        <width>32</width>
        <height>32</height>
        <tileInfo>no tile</tileInfo>
        <tileBounds>no tile</tileBounds>
        <tileGridOffset>no tile</tileGridOffset>
        <numberOfBands>1</numberOfBands>
        <dataType>byte</dataType>
        <colorMapType>ComponentColorModel</colorMapType>
        <numOfComponents>1</numOfComponents>
        <bitsPerPixel>8</bitsPerPixel>
        <transparency>opaque</transparency>
        <class>Subject1</class>
      </builtinattributes>
      <userattributes>
        <grayscale>yes</grayscale>
      </userattributes>
    </row>
    <row>
      ...
    </row>
    ...
  </data>
</table>

```

Figure 3.1 XML file format of MDT

3.3 USAGE SCENARIOS

This part attempts to put together a scenario involving a set of operations to serve as a useful example for MDT usage. In this scenario of using MDT, we will create our own image dataset tables, modify our images using built-in filters, run a classification and a clustering algorithm, and finally interpret the results.

For this usage scenario, we will assume that MDT is installed in our computer, and we already have an image database which consists of gray-scale photos of different subjects with different pose angles and facial expressions. Datasets will be explained in more detail in the following section.

Steps will be given under separate titles. Section 3.3.1 presents the issues to consider when selecting our dataset instances. Section 3.3.2 demonstrates how to create a table and build up a dataset. Section 3.3.3 explains an image processing operation. Finally, section 3.3.4 explains how to carry on a data mining operation.

3.3.1 Image Dataset Preparation

MDT's ARFF file contains only two types of values; the pixel information and class for any image. Each pixel consists of values that built up an array, of which size is specified by the band value. Band value forms the number of columns (attributes) for each pixel value in ARFF files, so all image files should have the same band value. This column appropriateness is only one of the issues that enforce many image properties to be identical for the dataset. Others that can be clearly seen are type, size, etc.

Another issue is that since the attribute number grows rapidly, performance issues may occur during data mining process, so it is better to keep the file size (widthxheight value) as small as possible. Also, for the same performance issues, it is better to use grayscale images.

In the experiments, we used three different databases. All of these provide grayscale images of various subjects. The images are frontal poses of individuals with

small variations in face position, illumination, facial details and expressions, etc. The databases are included in the thesis material and listed as:

- (a) AT&T Olivetti Research Laboratory, Cambridge Database
- (b) CMU AMP Face Expression Database
- (c) Equinox Corporation Human Identification at a Distance Database

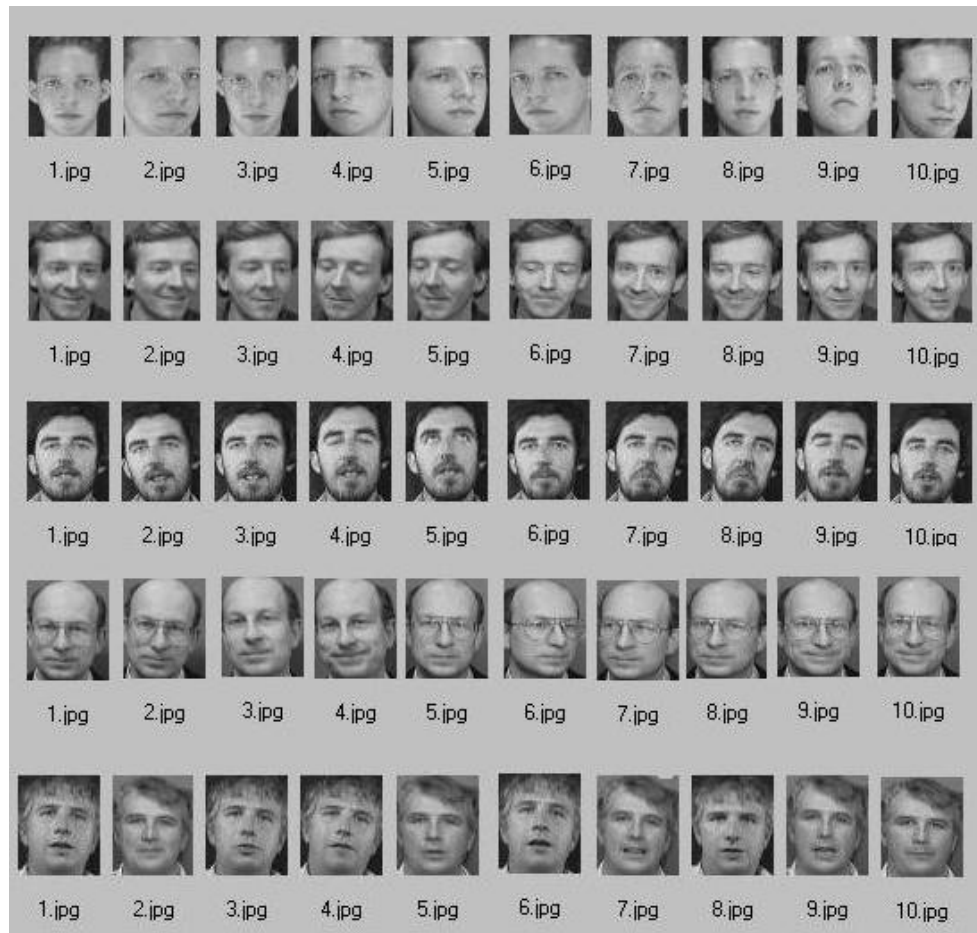


Figure 3.2 Grayscale images of subjects in different poses

An example part from the AT&T Olivetti image dataset is given in Figure 3.2. The images are 8 bit gray level images with homogenous background and different by varying lighting conditions, facial expressions (open/closed eyes, smiling/non-smiling) and facial details (glasses/no-glasses). We will try to carry out a face recognition operation, and pick up 5 different pose images of 5 distinct subjects. Subjects will be

named as s1, s2, s3, s4 and s5 respectively. The images are resized to 23x28 pixel sizes and converted to JPEG format.

3.3.2 Table Creation

Tables may either be created or loaded from an existing MDT XML document. At first, we will create a new table and carry out a classification. For clustering operation we will use the same table so we will need to load it again.

To create a table, select New from File menu. A dialog box appears asking the user to enter a name for the new table. Figure 3.3 illustrates creating a new table.



Figure 3.3 Creating a table

When we input a name and click on the create button, a new empty table will be shown on desktop, at the time an MDT XML file created with empty data tags in user directory.

Now we can add image files to our table. We may either use new row option in the Edit menu or import from Explorer. If Edit menu is used, a file chooser appears that let any file to be chosen. Adding an image using Edit menu is illustrated in Figure 3.4.

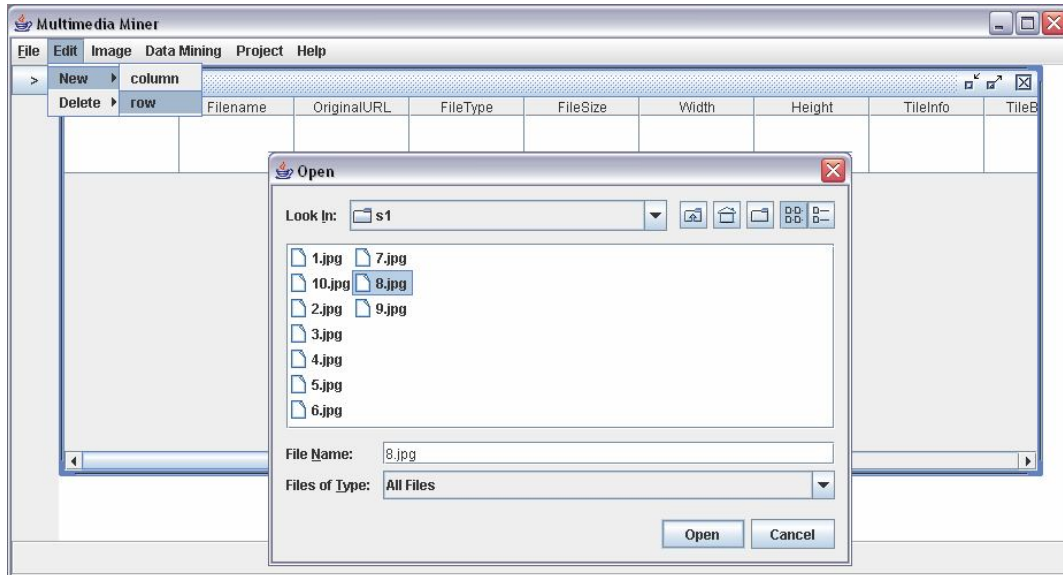


Figure 3.4 Adding row using menu

If we need to preview the images before importing, we should cancel the file chooser in order to use the Explorer. Left on the desktop pane, there is the explorer, clicking on the button makes it visible.

We can add images to the table either by selecting the image and clicking import button or only by double clicking on the image. The image is automatically added to the table. Figure 3.5 shows the explorer.

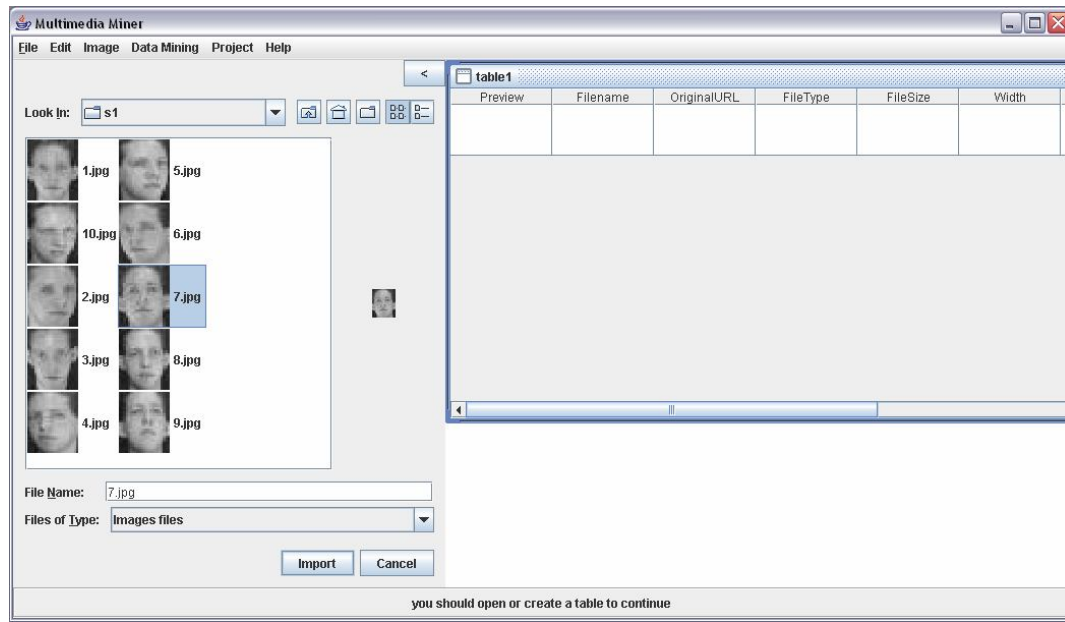


Figure 3.5 Importing image from explorer

Add or remove operations for rows and column are provided in edit menu. Column or row is selected by a single click, table cell values can be modified by double clicking.

It can be clearly seen that each image file forms a row with its properties as its columns. One or multiple rows (images) can be selected and selection from one of the image filter operations from the image menu starts the process.

3.3.3 Image Processing Example

We can do some image processing on our files. We select an image by single clicking on a row. Then we go to Image Menu, select one group of operations, and then select one of the filters. Figure 3.6 shows an example.

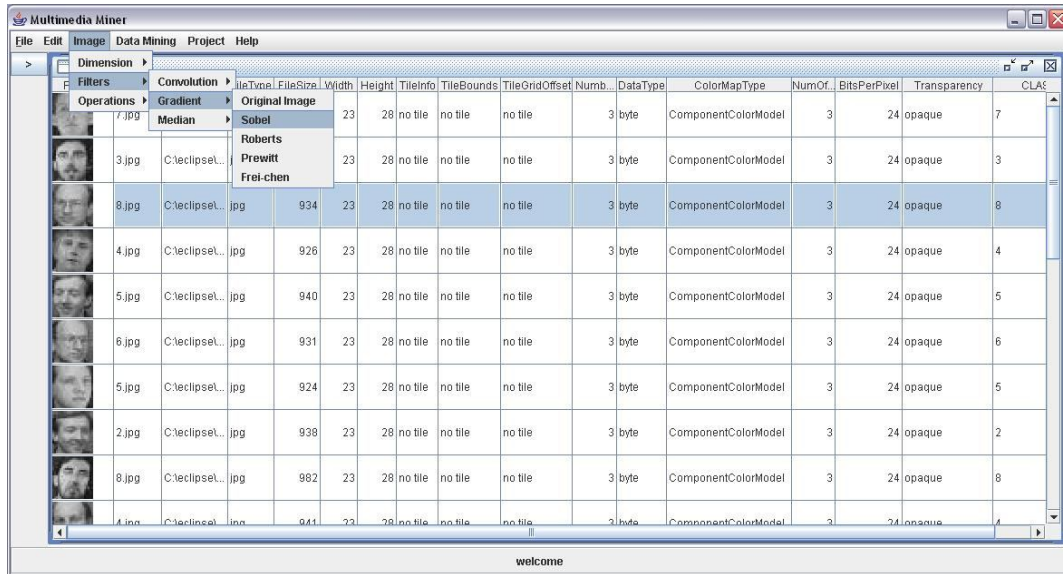


Figure 3.6 Applying filter to an instance

Result of the operation can be seen in the preview column immediately, in Figure 3.7. Image files are replaced with the filtered ones. This process is not revisable, he should be careful to back up the images. The images and tables can be collected under a directory for convenience.

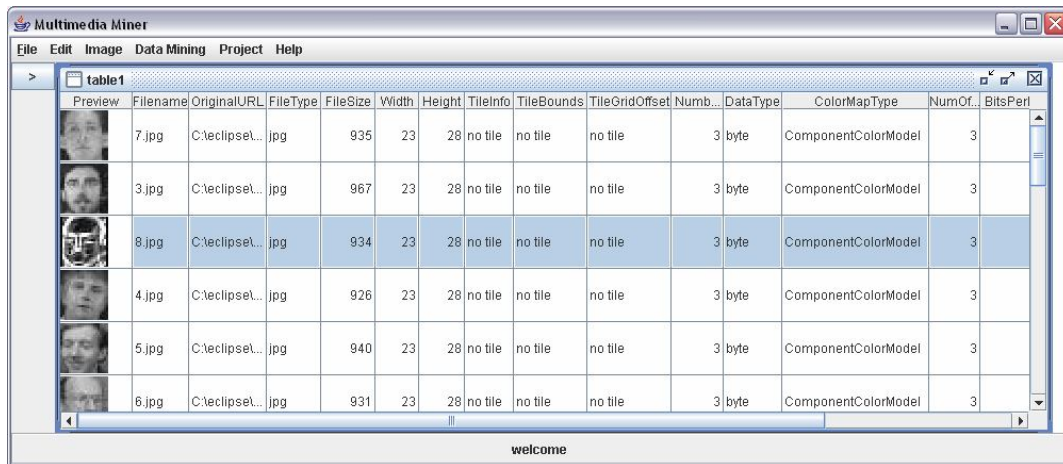


Figure 3.7 Result of filtering operation

3.3.4 Data Mining Example

As mentioned before, in order for the data mining algorithms run correctly, properties of image files in our table must be identical. That's why we should restore the image we modified before. This can be done by just changing the modified file with its original copy, i.e. copying the original and pasting it into the same directory. Then we will have our original table as our dataset.

In the table dataset, we have 5 different pose images of 5 subjects. We should check the class values under the CLASS column. By default, class column value of an instance is given the image file's name. The user should provide the correct class values. For prediction, we should place a question mark (?) into the cell under the CLASS column indicating the unknown class value. As shown in Figure 3.8 we put question marks to images of subjects called s4, s3, s5 and s2, respectively.

preview	fileName	originalURL	fileType	fileSize	width	height	tileInfo	tileBo...	tileOri...	numb...	dataType	colorMap...	numOf...	bitsPer...	transpar...	CLASS
	7.jpg	C:\eclipse...	jpg	940	23	28	no tile	no tile	no tile	3	byte	Compon...	3	24	opaque	s2
	9.jpg	C:\eclipse...	jpg	973	23	28	no tile	no tile	no tile	3	byte	Compon...	3	24	opaque	s3
	8.jpg	C:\eclipse...	jpg	942	23	28	no tile	no tile	no tile	3	byte	Compon...	3	24	opaque	s5
	3.jpg	C:\eclipse...	jpg	932	23	28	no tile	no tile	no tile	3	byte	Compon...	3	24	opaque	?
	9.jpg	C:\eclipse...	jpg	934	23	28	no tile	no tile	no tile	3	byte	Compon...	3	24	opaque	s2
	6.jpg	C:\eclipse...	jpg	962	23	28	no tile	no tile	no tile	3	byte	Compon...	3	24	opaque	?
	7.jpg	C:\eclipse...	jpg	940	23	28	no tile	no tile	no tile	3	byte	Compon...	3	24	opaque	s4
	10.jpg	C:\eclipse...	jpg	919	23	28	no tile	no tile	no tile	3	byte	Compon...	3	24	opaque	?
	1.jpg	C:\eclipse...	jpg	934	23	28	no tile	no tile	no tile	3	byte	Compon...	3	24	opaque	?

Figure 3.8 Preparing the dataset for data mining

The table should be saved before going for data mining process. We save the table by clicking on Save from the File Menu. At the time we save the table, four files are created. The first one is MDT's XML file. It contains table data as seen on the screen. The other three are ARFF files, a training file, a test file and a complete ARFF version

of the XML file. Training file contains instances of which class values are known. Test file is built up of instances of which class values are unknown, i.e given as question mark. The complete ARFF file is combination of test and training files which we need for clustering operation.

We click on to the Data Mine menu after saving the table, two data mining options are shown, one is classification and the other is clustering. We choose classification, then, Weka's Generic Object Editor (GOE) pops up to let us choose one of the classification algorithms. By clicking on the upper left Choose button, as shown in Figure 3.9, a list is expanded from which we can select our algorithm.

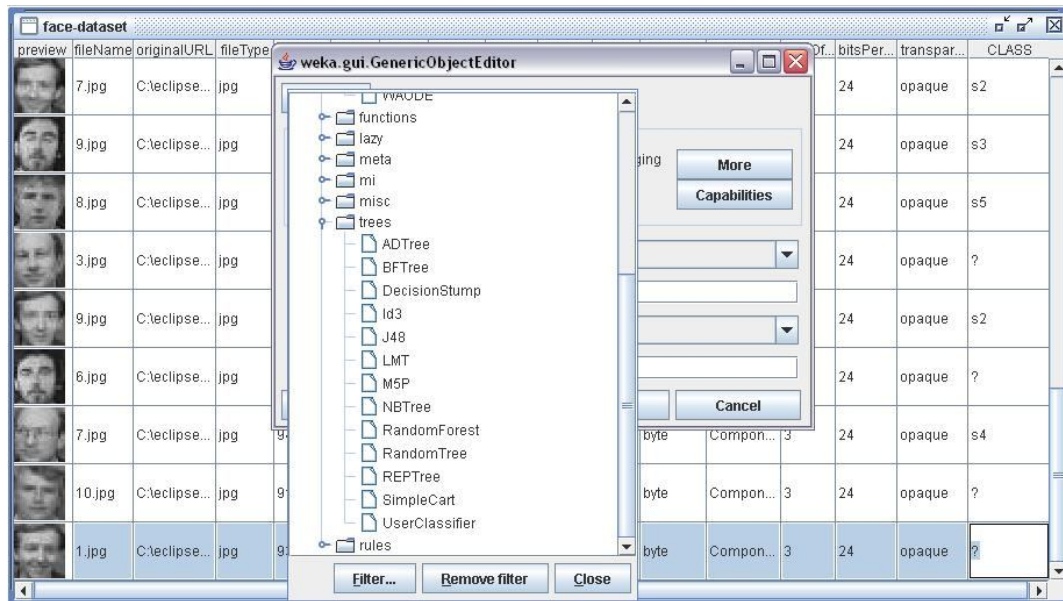


Figure 3.9 Choosing classification algorithm

We choose the J48 under the tree directory in the list and GEO is customized for the algorithm in order to let us make changes in its options. Figure 3.10 shows the customized GOE, by means of which we can set execution options for the algorithm.

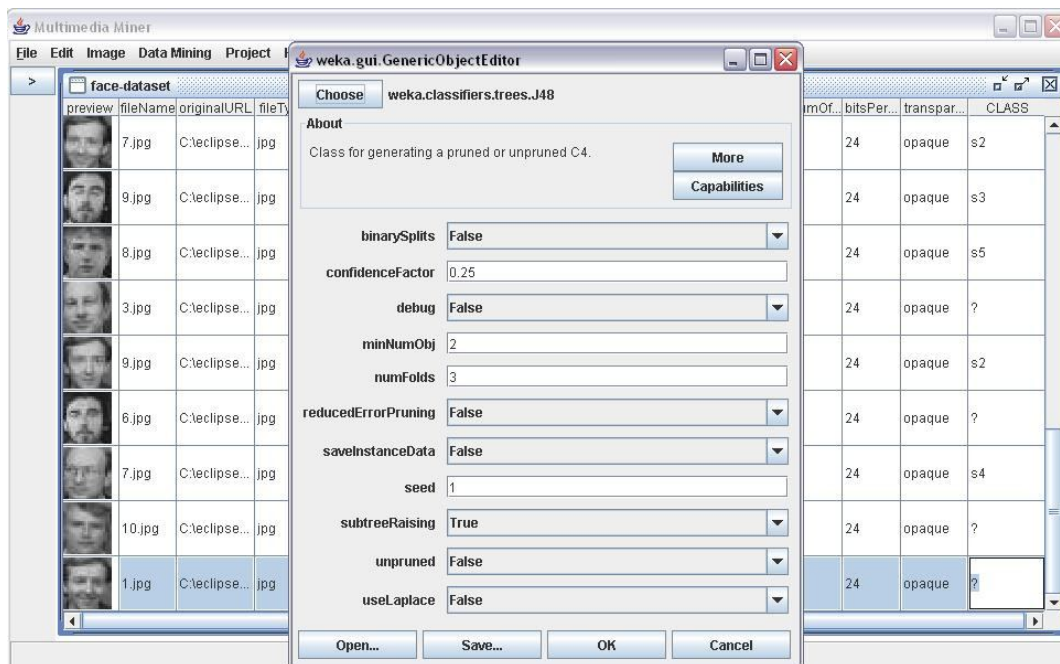


Figure 3.10 Generic Object Editor of J48 tree

We just click on OK to use the default options. We will see a window is showed which will be used for starting the data mining process and viewing the results.

Figure 3.11 shows the data mining window. The start button starts the process by sending the command shown in the text line between the start and predictions buttons. The empty text area under the buttons will output the results.

Clicking on the start button will run the algorithm with our table given as its parameters. In Figure 3.12 output of the algorithm is shown. Under the prediction column, the predicted classes are listed. We had put question marks for subjects 4, 3, 5 and 2. In the figure it can be seen that the first, the third and the fourth instances are classified correctly, i.e., as s4, s5 and s2 respectively. However the second one is classified incorrectly, s3 is predicted as it was s5. Clicking on the prediction button pops up the table and puts the predicted values to CLASS column of the corresponding instances.

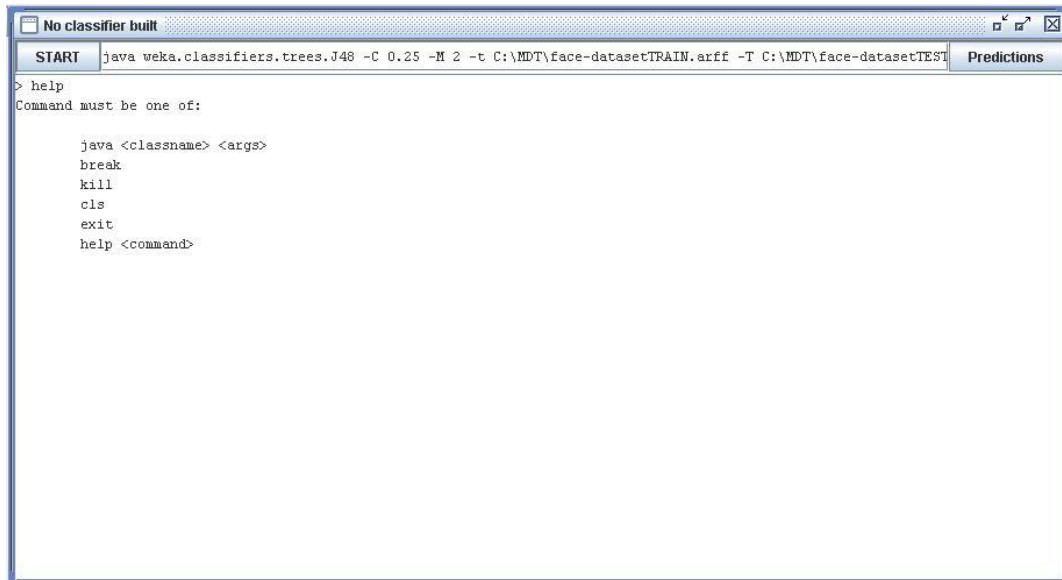


Figure 3.11 Data mining window for classification

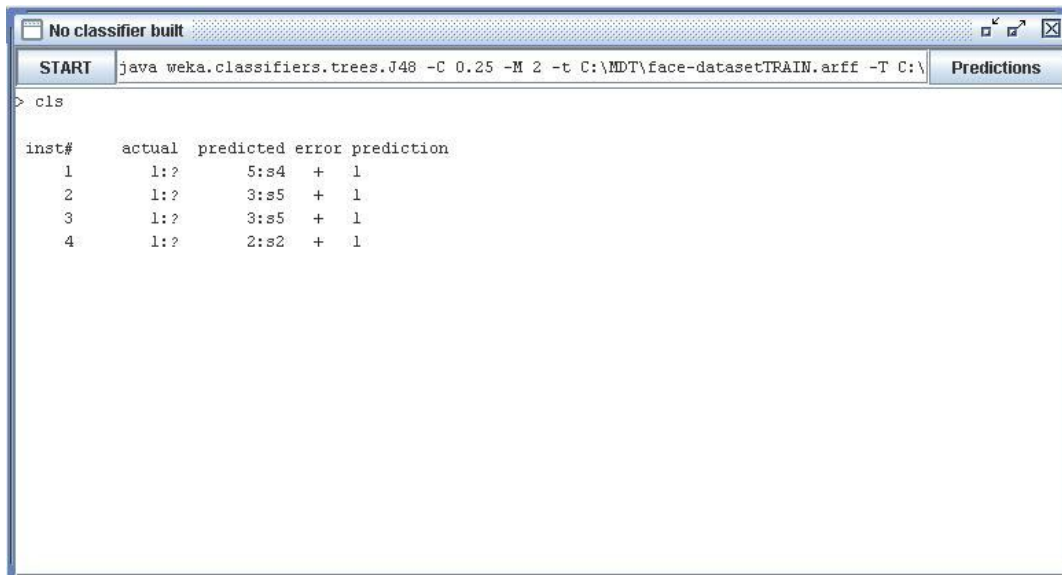


Figure 3.12 Results of the classification

For clustering operation, we choose the Clustering option from the Data Mining menu. Again, Weka's Generic Object Editor (GOE) shows up to let us choose one of the clustering algorithms. By clicking on the upper left Choose button, as shown in Figure 3.13, a list is expanded from which we can select our algorithm.

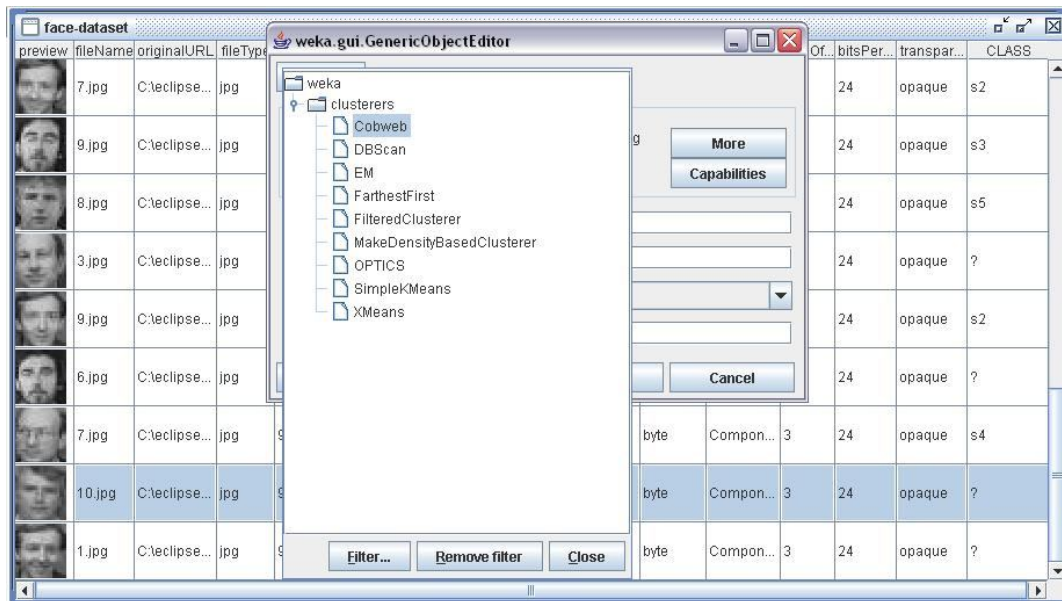


Figure 3.13 Selecting clustering algorithm

We choose the EM algorithm under the clusterers directory in the list and GEO is customized for the algorithm in order to let us make changes in its options. Figure 3.14 shows the customized GOE, by means of which we can adjust the data mining options for the algorithm.

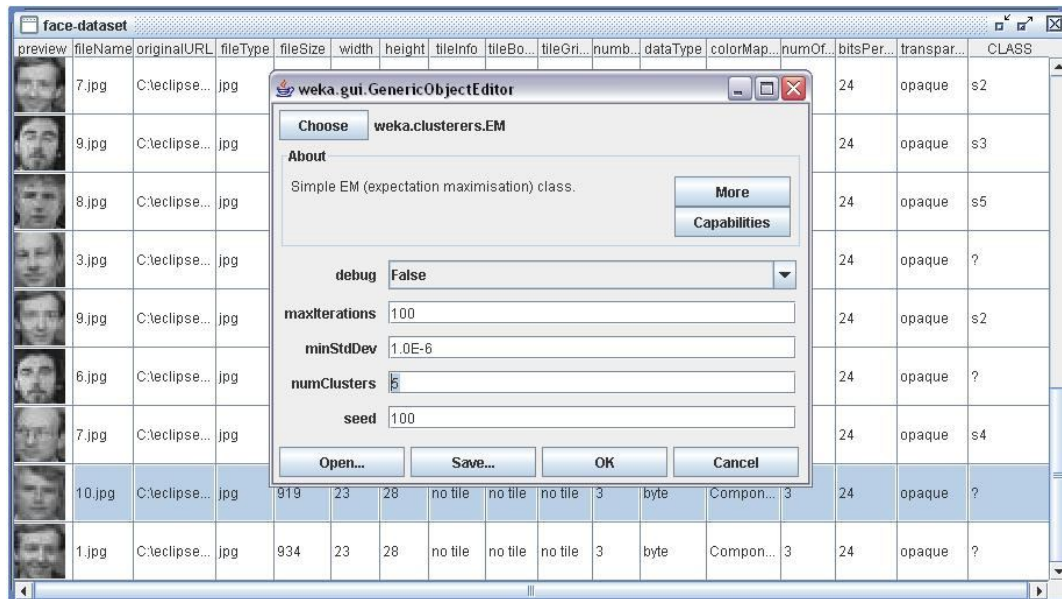



Figure 3.14 Generic Object Editor of EM algorithm

We enter the value 5 into the numClusters field. This means we need to have 5 clusters since we have 5 subjects. After adjusting the cluster number, we click on OK and again, the clustering window is showed which will be used for starting the data mining process and viewing the results. Figure 3.15 shows the data mining window.



```

No clusterer built yet!
START java weka.clusterers.EM -I 100 -N 5 -M 1.0E-6 -S 100 -t C:\MDT\face-dataset.arff -p 0
> help
Command must be one of:
  java <classname> <args>
  break
  kill
  cls
  exit
  help <command>

```

Figure 3.15 Data mining window for clustering

Clicking on the start button will run the algorithm with our table given as its parameters. In Figure 3.16 the output of the clustering operation is shown. Two columns are listed. The first column is the instance numbers which correspond to the row index. The second column lists the cluster numbers into which the corresponding instance is assigned. The 5 clusters are numbered from 0 to 4. In the figure, the results show that the first and second instances are assigned to the cluster 2, the third one is assigned to cluster 3, etc. By examining all of the results, assignments are made as follows; subject 1 is assigned to cluster 2, subject 2 to cluster 3, subject 3 to cluster 1, subject 4 to cluster 0 and finally, subject 5 to cluster 4. We can see that the two of the instances are clustered incorrectly, i.e., the third and fifth instances which are images of subject 5 are assigned to cluster 1.



The screenshot shows a Java command window titled "No clusterer built yet!". The command executed is `java weka.clusterers.EM -I 100 -N 5 -M 1.0E-6 -S 100 -t C:\MDT\face-dataset.arff -p 0`. The output displays 25 rows of data, each consisting of an index number followed by a cluster assignment number.

```
START java weka.clusterers.EM -I 100 -N 5 -M 1.0E-6 -S 100 -t C:\MDT\face-dataset.arff -p 0
0 2
1 2
2 3
3 1
4 1
5 1
6 0
7 2
8 1
9 4
10 1
11 0
12 2
13 3
14 0
15 2
16 3
17 1
18 4
19 0
20 3
21 1
22 0
23 4
24 3
```

Figure 3.16 Results of the clustering

CHAPTER 4

IMPLEMENTATION

4.1 INTRODUCTION TO MDT CLASS STRUCTURE

MDT is developed by using Java Programming Language. MDT's classes with their descriptions are given in Table 4.1.

The main class that starts up the user interface is the MainWindow class. Basic methods of this class will be explained in the following section.

MyTableModel class is designed to manage the actual table data. The model of MDT holds its data in a form of vector of vectors. Data table vector consists of elements of instances each of which is also a vector. Elements of the instance vectors correspond to the columns in the table. ReadXMLTable class is a SAX Handler used to read a file and populate the data table from MDT XML file. It reads the text data inside the XML tags which are column names and place the values as table cells under the corresponding columns. WriteXMLTable class creates MDT XML document from user table. It gathers the data from the table model. WriteARFFTable class does the same function to create an ARFF file which will be used for data mining. Writer classes use an instance of the ImageAdapter class for each image and extract pixel information. The ImageAdapter class extracts the image information listed in table columns. MyFilters class provides the filtering operations used for image processing.

The JAI related ImageAdapter and MyFilters classes are developed by making use of the source code which is part of the Java Advanced Imaging Stuff site, by Rafael Santos(Santos, 2007).

Table 4.1 Basic classes of MDT

Class Name	Description
AlgorithmArgs	Builds up the option string for data mining algorithms
ExplorerFilter	Extends FileFilter
	Filters file view of MyExplorer to show only image files of specific types
FileImage	Extends FileView
	Creates thumbnail previews of the files inside MyExplorer
ImageAdapter	Gathers properties of the image file
MainWindow	Extends JFrame
	Base class for user interface operations
MiningOutputs	Extends JInternalFrame, implements ActionListener
	User interface for data mining operations
MyExplorer	Extends JPanel
	Allows users to preview and import image files
MyFilters	Extends JFrame
	Define and apply image filters
MyTable	Defines MDT's tables
MyTableModel	Extends AbstractTableModel, implements TableModelListener
	Table Model of MDT table
PredictionsExtractor	Shows the predicted results of unknown instances on the table
Previewer	Extends JComponent
	Creates preview of the selected image
ReadXMLTable	Extends DefaultHandler
	Reads tables from specific XML format documents of MDT
Renderer	Extends JLabel, implements TableCellRenderer
	Provide image rendering inside the table cell
WriteARFF	Creates ARFF file for data mining
WriteXMLTable	Creates MDT's XML document of the table

4.2 MAINWINDOW CLASS

In the following figure, some of the basic methods of MainWindow are listed in Table 4.2. MainWindow initiates the graphical user interface and provides methods menu options.

Table 4.2 Methods of MainWindow class

Method Name	Description
MainWindow()	Constructor, initializes user interface
main(String[])	main, initialize objects
createAndShowGUI()	creates and shows user interface components
createStatusBar()	creates status bar
createMenuBar()	creates main menu and items
saveTable(String)	save current table in XML and ARFF format
loadATable()	loads an XML format table
createAtable(String)	creates an empty table
addArow()	insert an instance (an image file) to the table
addColumnn()	insert a new column(a user-defined property attribute) to table
deleteColumn(int)	removes the selected column from the table
deleteRow(int)	removes the selected row from the table
quit()	stops the execution and exits the program
createMenuItem(String, int,int,String, boolean)	create an item of the main menu
createDialog(String)	creates dialogs
RefreshTable()	refreshes table data
getTableNane()	returns table name of the active table
setTableName(String)	sets table name of the active table
getFileName()	returns filename of the active table
setFileName(String)	sets filename of the active table
getDbType()	returns type of the active table
setDbType(String)	sets type of the active table
getFilePath()	returns current file path
setFilePath(String)	sets current file path
getDesktop()	returns desktop component
setDesktop(JDesktopPane)	sets desktop component

If the user creates a new table `MainWindow.createATable(String doctype)` calls `WriteXMLTable.createEmptyTable()` in order to create a new table with the given name and extension(doctype). The `createEmptyTable()` will be explained later. If the user opens an existing table, `MainWindow.loadATable()` is called. This method reads a file and populates table data from the XML file using SAX. It calls an instance of `ReadXMLTable` as the SAX event handler. Figure shows part of `loadATable()` :

```
protected void loadAtable() throws IOException {
    ...
    //Reads a file and populates the table data from an XML file using SAX
    //SAX event handler is the instance of ReadXMLTable
    DefaultHandler handler = new ReadXMLTable (welcome);
    SAXParserFactory factory = SAXParserFactory.newInstance();
    factory.setValidating(true);

    //Parse the input
    SAXParser saxParser = factory.newSAXParser();
    saxParser.parse (new File (getFilepath()), handler );
    ...
}
```

Figure 4.1 Function `loadATable()`

Methods for table column modification are `addColumn()` and `deleteColumn(int index)`. The `addColumn()` method ask user for column name input and adds a new element of empty string at the end of each row in the current table. Some code from `addColumn()` is listed in Figure 4.2.


```

protected void addColumn() {
    ...
    String newColname = (String)JOptionPane.showInputDialog
        (this, "enter a name for new column :", "Add New Column",
            JOptionPane.QUESTION_MESSAGE);

    if ((newColname != null) && (newColname.length() > 0)) {
        myTable.mtm.columnNames.addElement(newColname);

        for (int i=0; i< myTable.mtm.getData().size(); i++){
            ((Vector) myTable.mtm.getData().elementAt(i)).addElement(" ");
        }
        myTable.mtm.fireTableDataChanged();
        myTable.mtm.fireTableStructureChanged();
        RefreshTable();
    }
}

```

Figure 4.2 Function addColumn()

The deleteColumn() removes an element which is at the given index –selected column index- from each row of the data table. Most important part of this function is listed in Figure 4.3.

```

protected void deleteColumn(int index) {
    ...
    int ind = myTable.table.getSelectedColumn();
    if (ind > 7 && !myTable.table.getColumnName(ind).equals("CLASS")) {
        // cannot delete constant 8 columns
        for (int i = 0; i < myTable.mtm.getData().size(); i++) {
            Vector v =(Vector) myTable.mtm.getData().get(i);
            v.removeElement(ind);
        }
        //remove the column header
        myTable.mtm.columnNames.remove(ind);
        MainWindow.myTable.mtm.fireTableStructureChanged();
        MainWindow.myTable.mtm.fireTableDataChanged();
    } else {
        System.out.println( " You cannot remove constant columns " );
    }
    ...
}

```

Figure 4.3 Function deleteColumn()

Methods for row modification are addRow() and deleteRow(). The addRow() function calls the importButtonAction() of MyExplorer.This action creates an ImageAdapter object to gather the image properties as table columns. The deleteRow()

function simply removes the element at given index from data vector and notifies the listener of MyTableModel class. The function is listed in Figure 4.4.

```
protected static void deleteRow(int index) {
    MainWindow.myTable.mtm.getData().removeElementAt(index);
    MainWindow.myTable.mtm.fireTableRowsDeleted(
        MainWindow.myTable.mtm.getData().size(),
        MainWindow.myTable.mtm.getData().size());
}
```

Figure 4.4 Function deleteRow()

The saveTable() function calls createDocument() of WriteXMLTable and WriteARFF classes according to the specified document type.

```
public static void saveTable(String doctype) {
    ...
    if (doctype.equals("xml")) {
        XmlFileFilter filter = new XmlFileFilter();
        ...
        if (result == JFileChooser.APPROVE_OPTION) {
            welcome.setTablename(saveDialog.getSelectedFile().getName());
            XMLwriter=new WriteXMLTable(welcome);
            XMLwriter.createDocument(filename);
        }
    } else if (doctype.equals("arff")){
        ArffFileFilter filter = new ArffFileFilter();
        ...
        if (result == JFileChooser.APPROVE_OPTION) {
            ...
            ARFFwriter = new WriteARFF(welcome);
            ARFFwriter.createDocument(filename, "simple");
        }
    } else if (doctype.equals("arff_train_and_test")) {
        ...
        welcome.setTablename(originalTableName+"TRAIN");
        ARFFwriter = new WriteARFF(welcome);
        ARFFwriter.createDocument(filename, "train");
        welcome.setTablename(originalTableName+"TEST");
        ARFFwriter = new WriteARFF(welcome);
        ARFFwriter.createDocument(filename, "test");
        welcome.setTablename(originalTableName);
    } else {
        System.out.println("No format selected");
    }
}
```

Figure 4.5 Function saveTable()

CHAPTER 5

CONCLUSION

A lightweight tool with a simple user interface is very important for the education of students that take data mining and image processing courses. Therefore an application that can integrate image processing and data mining of image files is quite valuable for this purpose.

In this thesis, a data mining tool for image files is developed. The application is developed with Java programming language because it provides the advantages of powerful JAI API and Swing classes. For the data mining implementation, WEKA is preferred because of its open source structure, extendibility with user defined classes and large scale of algorithm options.

The three fundamental facilities of MDT are storing and conversion of table datasets, applying image processing filters to the instances which are image files, and running classification and clustering algorithms on image pixel information.

First feature is supported by means of two types of converters that implement the conversion of table data to XML format and ARFF format of WEKA. The table data provided a functional way for managing datasets easily. Converters for XML and ARFF handled all of the document processing operations which can be considered as an advantage to the user by hiding details of file formats. ARFF converter has an extended ability to use the pixel extraction class.

The second capability is provided through a set of pixel level filters and operators that are developed by using functions from the Java Advanced Imaging Application Programming Interface. The image filtering operations are provided in pixel level and

worked well enough to illustrate the results. Using small sized gray scale images provided improved visibility and performance.

Finally, the third goal is accomplished by sending Java calls to Weka's data mining API in order to apply data mining tasks to image datasets. The two enabled data mining tasks are classification and clustering. The classification and clustering tasks were based on numeric pixel data that is extracted during the process of ARFF file conversion. Data mining process was successful because of the powerful weka algorithms and small sized grayscale image instances with specific face positions.

The tool only works for image files now, but it may be adapted for data mining of many more multimedia file types for example, audio, video, and text files in the future. Current system is able to work with grayscale and small sized images because of the performance issues, better computer systems will provide better results especially in data mining process and enable using color images. Also, using small set of advanced image processing techniques may speed up the process that will yield better results.

REFERENCES

- Chang, S.F., Chen, W., Sundaram, H., “VideoQ: a fully automated video retrieval system using motion sketches”, *Proceedings of the Fourth IEEE Workshop on Applications of Computer Vision*, Princeton, New Jersey, October 1998, pp. 270
- Collier, K., Carey, B., Sautter, D., Marjaniemi, C., “A Methodology for Evaluating and Selecting Data Mining Software”, *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences, 1999. HICSS-32*, Maui-HI USA, 1999 Vol. 6, pp. 11, 1999.
- Fayyad, U.M., Djorgovski, S.G., and Weir, N., “Automating the Analysis and Cataloging of Sky Surveys”, *Advances in Knowledge Discovery and Data Mining*, 471–493, 1996
- Grosky, W. and Tao, Y., “Multimedia Data Mining and Its Implications for Query Processing”, *The Ninth International Workshop on Database and Expert Systems Applications (DEXA'98)*, Vienna, Austria, August 1998, pp. 95-100.
- Gudivada, V., Raghavan, V.V. and Vanapipat, K. “A Unified Approach to Data Modeling and Retrieval for a Class of Image Database Applications,” *Multimedia Database Systems*, Springer-Verlag, New York, Inc., Secaucus, NJ, pp73-78, 1996.
- Gudivada V. and Raghavan V., “Content-based image retrieval systems”, *IEEE Computer*, Vol.28, No.9, pp. 18–22, September 1995.
- Han, J., Chiang, J., Chee, S., Chen J., Chen, Q., Cheng, S., Gong, W., Kamber, M., Liu, G., Koperski, K., Lu, Y., Stefanovic, N., Winstone, L., Xia, B., Zaiane, O. R., Zhang, S. and Zhu, H. “DBMiner: A system for data mining in relational databases and data warehouses”, *Proceedings of. CASCON'97: Meeting of Minds*, Toronto, Canada, November 1997, pp. 249-260.
- Hand D., Mannila H., and Smyth P., *Principles of Data Mining*, MIT Press, Cambridge, MA, 2001.

- Harold, E. R., *Processing XML with Java*, Pearson Prentice Hall, Boston, 2003.
- Holmes, G., Donkin, A., Witten, I.H., “WEKA: A Machine Learning Workbench”, *Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems*, pp. 357-361, 1994.
- Hsu, W., Lee, M.L., and Goh, K.G, “Image Mining in IRIS: Integrated Retinal Information System”, *ACM SIGMOD*, 2000.
- Hsu, W., Lee, M. L.,and Zhang, J., “Image Mining: Trends and Developments”, *Journal of Intelligent Information Systems*, Vol19:1, pp. 7–23, January 2002
- Kitamoto, A, “Data Mining forTyphoon Image Collection.”, *Second InternationalWorkshop on Multimedia Data Mining (MDM/KDD’2001)*, 2001.
- Missaoui, R. and Palenichka, R. M., “Effective Image and Video Mining: an Overview of Model-Based Approaches”, *Proceedings of the 6th international workshop on Multimedia data mining: mining integrated media and complex data*, pp. 43-52, 2005
- Ordonez, C. and Omiecinski, E.,”Discovering Association Rules Based on Image Content., *IEEE Advances in Digital Libraries Conference*, 1999.
- Prasad, C. K., *Developing Imaging Applications Using the Java2D^[tm]*, *JAI and New ImageIO APIs*, 2002.
<http://access1.sun.com/techarticles/ImagingApps/JImage.html>
- Roden, J., Burl, M., Fowlkes, C., “The Diamond Eye image mining system”, *Eleventh International Conference on Scientific and Statistical Database Management*, 1999. , 28-30 July 1999, pp. 283
- Rushing, J., Ramachandran, R., Nair U., Graves S., Welch R., Hong Lin, “ADaM: A data mining toolkit for scientists and engineers”, *Computers & geosciences (Comput. geosci.)*, Vol. 31, No. 5, pp. 607-618, 2005
- Santos, R., *Java Advanced Imaging Stuff*, 2007, <http://jaistuff.dev.java.net>
- The official Java Sun web site, <http://java.sun.com>

The official XML web site, <http://www.w3.org/xml>

Wactlar, H.D., Kanade, T., Smith, M.A., Stevens, S.M., Intelligent access to digital video: Informedia project”, *Computer*, Vol. 29, No. 5, pp. 46-52, May 1996.

Weka homepage, <http://www.cs.waikato.ac.nz/~ml/weka/>

Witten, I. H. and Frank E., *Data Mining: Practical machine learning tools and techniques*, Morgan Kaufmann, San Francisco, 2005.

Zaiane, O. R., Han, J., Li, Z. N., Chee, S. H., Chiang, J. Y., “MultiMediaMiner: A System Prototype for MultiMedia Data Mining”, *SIGMOD Conference*, p581-583, 1998.