ADaM 4.0.1 Documentation

1 General Information

ADaM is a data mining toolkit designed for use with scientific and image data. It includes pattern recognition, image processing, optimization, and association rule mining capabilities. ADaM does not contain grid projection, advanced subsetting, advanced statistical analysis, format conversion, visualization or other types of tools that may be useful in the analysis of scientific data sets. The system consists of a set of individual components that can be used together to perform complex tasks. Components are packaged as:

- Executables
- Python Modules

Packaging the components in these forms will facilitate the rapid development of scientific data mining applications, while allowing for efficient implementation of performance critical components. Care has been taken to ensure that the components of the system are as independent as possible, in order to make it possible to use subsets of the components that are appropriate for given applications. The approach also makes it possible to use third-party components with ADaM. All components are available on MS Windows and Linux platforms.

1.1 Installation and Configuration

The ADaM toolkit is packaged as a compressed archive. In order to install the product you must unpack the archive using Zip on windows or tar and gzip on Linux. Once the executables are unpacked, you must add the directory where they reside to your PATH variable. If you wish to use Python modules, you must also update your PYTHONPATH variable.

1.2 Using Command Line Executables

All of the programs included in the ADaM product are self-documenting. Typing the program name followed by -h will cause the program to print a brief description of what it does, along with descriptions for each of its command line parameters. Most of the components read one or more files, perform some processing, and then produce one or more output files. A simple example is shown below, where an input gray level gif image is converted to ADaM image format, median filtered, and converted into an output gray level gif image:

```
X:\>ITSC_MedianFilter -h
Program: ITSC_MedianFilter

Options:
-h Print this message
-i <filename> Name of the input image
-o <filename> Name of the output image
-w <winSize> Size of window to take median in

Description:
ITSC_MedianFilter is a function that computes the median filter response for a given image. The response for a given pixel is the median of the pixel values in the neighborhood of that pixel in the source image. The winSize parameter determines the size of the neighborhood.

X:\>ITSC_CvtGifToImage -i Traffic.gif -o Traffic.bin
X:\>ITSC_MedianFilter -i Traffic.bin -o TrafficFilt.bin -w 5
X:\>ITSC_CvtImageToGif -i TrafficFilt.bin -o TrafficFilt.gif
```
1.3 Using Python Wrappers

The Python modules included with ADaM are also self-documenting, and are similar in functionality to the executables. Importing the module and then typing help(moduleName) will cause the module to print a brief description of what it does, along with the syntax for using the command. The following example demonstrates how to read in an image, median filter it, and write out the result.

```
X:\>python
ActivePython 2.2.2 Build 224 (ActiveState Corp.) based on
Python 2.2.2 (#37, Nov 26 2002, 10:24:37) [MSC 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import ITSC_ImagePython
>>> import ITSC_MedianFilterPython
>>> help(ITSC_MedianFilterPython)
Help on module ITSC_MedianFilterPython:
NAME
    ITSC_MedianFilterPython
FILE
c:\john\build\release\itsc_medianfilterpython.dll
FUNCTIONS
    MedianFilter(...)
        MedianFilter(img, out, winSize)
            -img       Input image object of type of ITSC_ImagePython
            -out      Output image object of type of ITSC_ImagePython
            -winSize  Size of window to take median in
Description:
    MedianFilter is a function that computes the median filter
    response for a given image. The response for a given pixel is
    the median of the pixel values in the neighborhood of that pixel
    in the source image. The winSize parameter determines the size
    of the neighborhood.

DATA
    __file__ = r'C:\John\Build\Release\ITSC_MedianFilterPython.dll'
    __name__ = 'ITSC_MedianFilterPython'

>>> img = ITSC_ImagePython.New()
>>> img.Read("Traffic.bin", "b")
>>> out = img.Copy()
>>> ITSC_MedianFilterPython.MedianFilter(img, out, 5)
>>> out.Write("TrafficFilt.bin","b")

2 Pattern Recognition Techniques

ADaM includes classification, clustering, and feature selection / reduction techniques as well as some simple utilities that are useful in pattern recognition applications. The pattern recognition programs in ADaM read and write ARFF (Attribute Relationship File Format) files, which is the same format used by the WEKA (Waikato Environment for Knowledge Acquisition) data mining toolkit. This format is a simple text format with each input vector specified as one line in the file. Both numeric and non-numeric attributes are supported. See Appendix A - ARFF Format for details.
2.1 Classification Techniques

Classifiers generally consist of two components: a training module and an application module. The training module uses sample patterns to learn the characteristics of the classes of interest. The application module reads the description produced by the training module and classifies patterns. The following programs are available in the current release:

- ITSC_BayesClassifierTrain: Train a Bayes classifier
- ITSC_BayesClassifierApply: Apply a Bayes classifier
- ITSC_BayesNetworkTrain: Train a Bayes belief network classifier
- ITSC_BayesNetworkApply: Apply a Bayes belief network classifier
- ITSC_BpnnClassifierTrain: Train a Backpropagation Neural Network
- ITSC_BpnnClassifierApply: Apply a Backpropagation Neural Network
- ITSC_KNNClassifierApply: Apply a K Nearest Neighbor classifier
- ITSC_KNNClassifierTrain: Train a Multi-Prototype Minimum Distance classifier
- ITSC_MpmdClassifierApply: Apply a Multi-Prototype Minimum Distance classifier
- ITSC_MpmdClassifierTrain: Train a Multi-Prototype Minimum Distance classifier
- ITSC_NaiveBayesTrain: Train a Naïve Bayes classifier
- ITSC_NaiveBayesApply: Apply a Naïve Bayes classifier
- ITSC_RsnnClassifierTrain: Train a Recursively Splitting Neural Network
- ITSC_RsnnClassifierApply: Apply a Recursively Splitting Neural Network

Note: No training module is required for K Nearest Neighbor. The training samples effectively describe the classifier since it operates by matching unknown vectors to labeled samples.

2.2 Clustering Techniques

Clustering tools take a set of patterns as input and group them into classes based on similarity. The clustering tools will output a classified pattern set and a description of the clusters. The following programs are available in the current release:

- ITSC_DBSCAN: Cluster data using DBSCAN algorithm
- ITSC_Isodata: Cluster data using Isodata algorithm
- ITSC_Kmeans: Cluster data using K-Means algorithm
- ITSC_Maximin: Cluster data using Maximin algorithm

2.3 Feature Selection / Reduction Techniques

Feature selection and reduction techniques reduce the size of the input data set. Feature selection techniques do this by choosing a subset of the available attributes. Other feature reduction techniques do this by creating a mapping of the original feature space onto a feature space of smaller dimension. Feature selection / reduction programs include:

- ITSC_BackwardElimination: Select features using backward elimination
- ITSC_ForwardSelection: Select features using forward selection
- ITSC_PrincipalComponents: Reduce features using principal components
- ITSC_Relief: RELIEF filter based feature selection algorithm
- ITSC_RemoveAttributes: Remove explicitly specified attributes

2.4 Pattern Recognition Utilities

ADaM also includes some utilities that aid in the pattern recognition process. Normalization is an important step that can improve the results produced during clustering and classification. The k-fold cross validation and accuracy utilities are useful in assessing classifier performance. Pattern recognition utilities include:
3 Association Rules

Association rule mining is used to identify relationships among attributes in large data sets. Given a set of items and transactions, an association rule miner will determine which items frequently occur together in the same transactions. The association rule module in ADaM reads an ARFF file (the same type of file used by the pattern recognition utilities), and produces a set of association rules. Each pattern vector is assumed to be a transaction, and the attribute values are the items.

ITSC_AssociationRules Mine the pattern set for association rules

4 Image Processing Features

ADaM includes a set of image processing modules that are useful for extracting features from images as a precursor to mining. The image processing programs read data in a simple binary image format described in Appendix B - Binary Image Format. The image format supports single plane, three-dimensional, real valued images. (Two-dimensional images are treated as three-dimensional images with a z size of one). The image processing toolkit include a rich set of texture features, since this is a research interest at ITSC.

A sample conversion utility, which converts to and from gif format is provided with the ADaM product. This can be used as an example for creating other translation utilities. The ESML tool provided by ITSC is the recommended method for file conversion.

4.1 Basic Image Operations

ADaM includes basic image operations for changing the size, orientation, scale and other properties of images. Typically, the range of the pixel values in the image will be mapped to the range [0..1]. Some utilities such as quantization provide an option to keep the original gray level range of [0 .. numLevels-1]. Basic operations include:

ITSC_Arithmetic Add or subtract images from one another
ITSC_Collage Create an image by overlaying parts of other images
ITSC_Crop Choose a rectangular region within an image
ITSC_ImageDiff Produce a difference image from two source images
ITSC_Equalize Performs histogram equalization on an image
ITSC_Inverse Reverses the pixel intensities (negative image)
ITSC_Quantize Reduces the number of levels in the image
ITSC_RelLevel Quantizes to three levels based on local image statistics
ITSC_Resample Applies spatial scaling to the image, changes image size
4.2 Segmentation / Edge and Shape Detection

ADaM includes utilities to find boundaries, contiguous regions, and polygons in images. The make region and mark region utilities can be used as precursors for the boundary utility. Boundary and shape extraction programs include:

- **ITSC_Boundary**: Detects boundary pixels (those not surrounded by like pixels)
- **ITSC_Polygon**: Circumscribe image regions with polygons (incl. convex hull)
- **ITSC_MakeRegion**: Tests pixels to see if they fall in a specified range
- **ITSC_MarkRegion**: Assigns unique id to each contiguous uniform pixel region

4.3 Filtering

Filtering plays an important role in many image analysis applications. ADaM has spatial domain, median, mode and morphological filters. It also has the pulse coupled neural network, which can be for image smoothing and segmentation. Filtering programs include:

- **ITSC_Dilate**: Morphological filter, performs image dilation
- **ITSC_Energy**: Computes energy (absolute value or square)
- **ITSC_Erode**: Morphological filter, performs image erosion
- **ITSC_FFT**: Fast Fourier Transform
- **ITSC_MedianFilter**: Median filter (used for smoothing)
- **ITSC_ModeFilter**: Mode filter (used for smoothing)
- **ITSC_Pcnan**: Pulse coupled neural network (used for smoothing)
- **ITSC_SpatialFilter**: Spatial domain filter, user specified mask

4.4 Texture Features

Texture features are used to classify and segment images based on local image structure. There are many different ways to extract texture features from images, and the ADaM system has a rich set of texture capabilities. These include:

- **ITSC_EvaluateRulesets**: Computes association rule statistics for multiple images
- **ITSC_FindAssociations**: Find association rules that characterize an image region
- **ITSC_Fractal**: Computes fractal dimension based texture features
- **ITSC_Gabor**: Computes Gabor filter based texture features
- **ITSC_Glcm**: Computes gray level cooccurrence based texture features
- **ITSC_Grlr**: Computes gray level run length based texture features
- **ITSC_Markov**: Computes Markov random field based texture features
- **ITSC_RuleFeatures**: Computes association rule based texture features
- **ITSC_RuleSelect**: Selects a set of association rules to discriminate textures
- **ITSC_RuleUnion**: Combines multiple sets of association rule features

5 Optimization Techniques

Optimization methods are used to identify good solutions to difficult search problems involving very large search spaces. The optimization methods in ADaM call external objective functions. They do this to decouple the optimization methods from the functions being optimized, and to allow the use of arbitrarily complex objective functions.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ITSC_GeneticAlgorithm</td>
<td>Genetic algorithm optimization</td>
</tr>
<tr>
<td>ITSC_HillClimbing</td>
<td>Stochastic hill climbing optimization</td>
</tr>
<tr>
<td>ITSC_SimulatedAnnealing</td>
<td>Simulated annealing optimization</td>
</tr>
</tbody>
</table>
Appendix A – ARFF Format

The arff format is used by ADaM Pattern Recognition operations, and comes originally from the Waikato Environment for Knowledge Acquisition (WEKA) toolkit, which is available at http://www.cs.waikato.ac.nz/ml/weka/. ADaM supports nominal and numeric attributes, but not string attributes. Datasets must begin with a name declaration of the form:

@relation data_name

The name declaration must be followed by a series of attribute specifications of the form:

@attribute attribute_name {value_one, value_two, ... value_N}

for nominal attributes, or

@attribute attribute_name numeric

for numeric attributes. (A string type is supported by WEKA, but not by ADaM).

The attribute declarations are followed by the start of data tag:

@data

Following this tag is a list of pattern vectors, one vector per line. The elements of the pattern vectors should be separated by commas.

Appendix B – Binary Image Format

Binary images will consist of a 4-word header followed by binary IEEE 32 bit floating-point numbers. The header will consist of the x, y, and z size and a marker word used to determine if byte swapping is necessary. Here is some sample code that indicates how the header can be written:

```c
int header[4];
header[0] = 0xabcd;
header[1] = mSize.x;
header[2] = mSize.y;
header[3] = mSize.z;
if (fwrite (header, sizeof(int), 4, outfile) != 4)
{
    // Error: do something about it
}
```