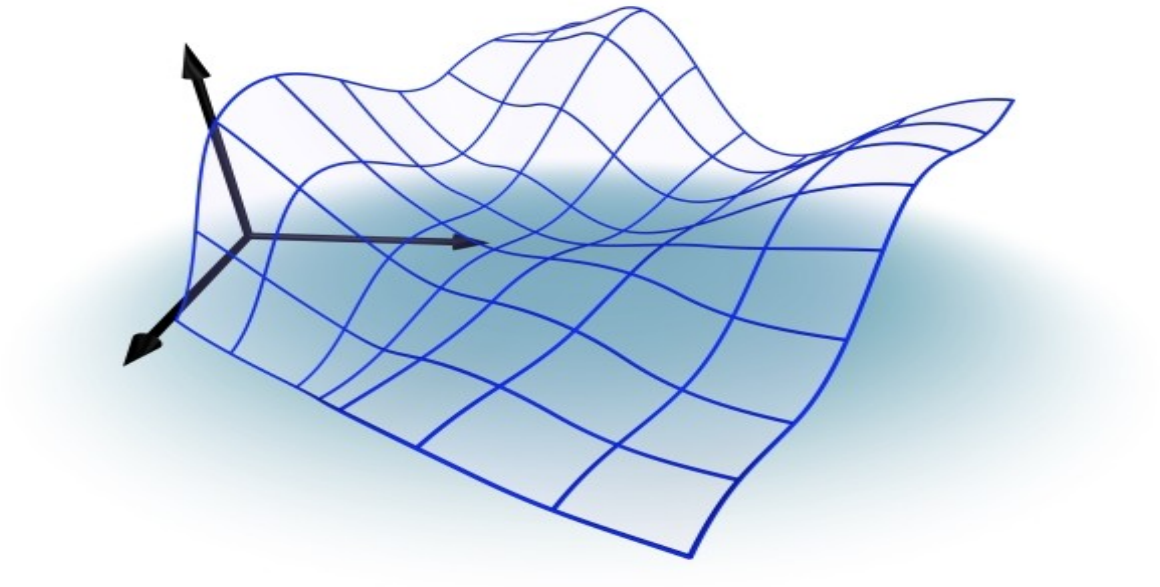
Self-Organizing Map Plug-in v1.0

for the Yale Machine Learning Environment



Bastian Tenbergen

btenberg@uos.de
http://www-lehre.inf.uos.de/~btenberg/
School of Human Sciences
University of Osnabrueck

October, 17th, 2006

Table of Contents

Section 1

Introduction

During the last few years, the channels of acquiring musical data have increased rapidly. Due to the growing importance of broad-band network connections, home computer users are no longer limited to radio or television broadcasting services or other non-computer aided methods to acquire copies of musical pieces. More or less legal peer-to-peer networks, online music stores and personalized Internet radio stations provide the user with a constant and never ending flow of musical information. The choice seems to be endless. On the one hand this large supply makes it possible to get any piece of musical information any time with a minimal effort of time and costs. On the other hand, this gives rise to a new, challenging problem: Retrieving musical information in large corpora.

This plug-in for the Machine Learning Environment Yale (http://yale.sf.net) provides one operator to classify digital song files with regard to extracted musical features as supplied before the classification. The plug-in does so by making use of a Self-Organizing Map and classifies the audio data into clusters. This allows for a variety of different tasks. As an example, meta information for unnamed song files can be retrieved. Song files that are not named or named not according to the actual song name can easily be identified in a cluster containing much fewer songs than the original corpus (which might be excessively large). In addition, applying this operator on a set of files that are most often played by a user can help automating user preference analysis to individualize the offer of online music stores and other personalized music services.

Section 2

System requirements

Software Requirements

In order to use this plug-in, you need the following software.

- Yale – Yet Another Learning Environment, Version 3.3 or higher.
  (http://yale.sf.net)
- Value Series Processing Plug-in for Yale, Version 3.3 or higher.
  (http://yale.sf.net)
- Java (TM) 2 Runtime Environment, Standard Edition, build 1.5.0_06-b05 or later
  (http://java.sun.com)
- any operating system, the Java Runtime Environment is available for
  (this software was developed on Microsoft Windows XP and Debian Linux, support for other operating systems cannot be provided yet)

Hardware Requirements

Minimal tested requirements (No support for evolutionary algorithm):

- IBM compatible processor with 900MHz
- 224MB SD-Ram
- 32MB Shared Memory Graphics Adapter
- 80MB of free hard disk space

Recommended system set up (evolutionary algorithm possible)

- IBM compatible processor with at least 2,7GHz or any 64Bit or Multi-Core Processor
- 1536MB DDR-Ram or more
- 64MB DDR Memory Graphics Adapter
- 1GB free hard disk space (for swap files)

Section 3

Installation

3.1 Installing Yale

Download the desired version from http://yale.sf.net and copy it to a local directory on your hard disk.
If you are using Microsoft Windows, simply double-click the installer.
On any non-Windows operating system or if you don't like Windows installers, change to the folder where you downloaded the package file and unzip the file “yale-3.3-bin.zip“. This will create a Yale home directory with all necessary files.

Please note that Yale as well as both its plug-ins, the ValueSeriesPlugin and the Self-Organizing Map Plug-in are written entirely in Java. You therefore need to have a Java version installed, either version 1.5.0_06 or higher. Without the correct Java version, you will neither be able to run Yale, nor the plug-in at hand.

You can obtain a Java installation package from http://java.sun.com. For installation on your platform, please refer to the Java documentation.

3.2 Installing the plug-ins

You should have received a copy of the Self-Organizing Map Plug-in, either source, full or binary, together with this file. If not, please visit http://www-lehre.inf.uos.de/~btenberg/ to obtain a copy. The ValueSeriesPlugin can be downloaded on http://yale.sf.net.
To install any plug-in, simply copy the *.jar file into the /lib/plug-in/ directory in Yale's home directory or run the installer exe-file, if provided. On start-up, Yale will load the plug-in automatically.

Section 4

The operator parameters

After having loaded and clicked on the Self-Organizing Map Operator, you will a screen like this.

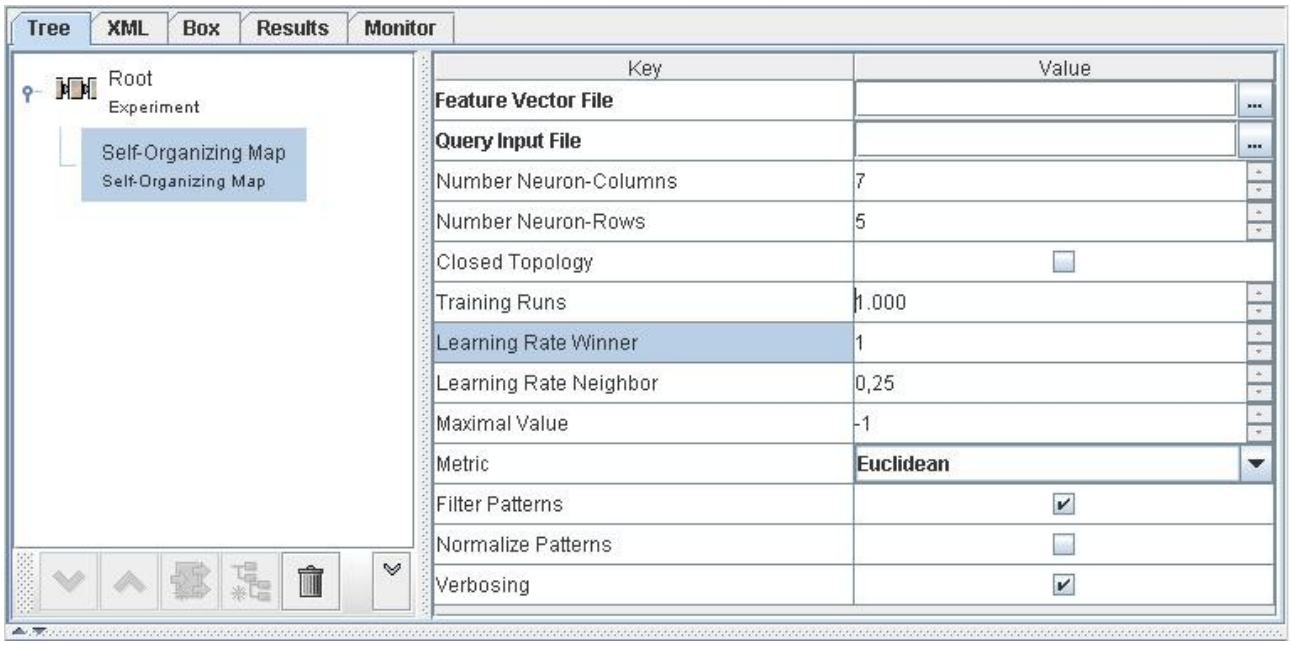


Figure 1. Overview over the operator parameters.

In this section, the parameters of the operator will be explained.

The operator takes a Feature Vector File and a Query Input File as the mandatory input. All other parameters have standard values that do not need to be changed.

For the **Feature Vector File**, choose either one of the feature vector files that came in one package together with the experiment files or choose a file containing features that you extracted from music files yourself. In any case, the feature vector file needs to be a file

containing a number of feature vectors, each of which is represented in an individual line. The features in each line need to be blank space separated double values. Not-a-Number values are silently ignored. The last element of each line should be an identifier such as the song name.

The **Query Input File** should be one of the files that have been processed, in case you have extracted features from a set of music files yourself. Simply choose the file you are eager to find similarity clusters for. Please note that this file is neither manipulated, nor are features extracted from this file by this operator. The features of this file already need to be extracted and must hence be found in the feature vector file. This variable simply stores the string representation of the file name and uses this to retrieve the similarity cluster.

This operator is implemented as a Self-Organizing Map. In neural networks like this, neurons can be aligned in different kinds of topologies. The most essential topology in this operator is a rectangle, however modifications by changing the number and distribution of neurons are possible. The parameters **Number Neuron-Columns** and **Number Neuron-Rows** allow you to change the topology according to your gusto. Depending on the number of input patterns (i.e. the number of files, features have been extracted from), different topologies and amounts of neurons might lead to a more or less satisfying results. Feel invited to try a variety of topologies and numbers of neurons to find the ideal setting for your task. The following topologies are possible:

Topology	***Parameter setting***
Row	**x** neurons in column and **1** neuron in row ***or*** **1** neuron in column and **x** neurons in row
Rectangle	**x** neurons in column and **y** neurons in row
Square	**x** neurons in column and **x** neurons in row

Table 1. Parameter settings for network topologies.

The parameter **Closed Topology** even enlarges the choice of possible topologies in this Self-Organizing Map. If checked, the last neuron of every row and column will be bidirectionally connected to the first neuron of the respective column or row. Two

bidirectionally connected neurons can be understood as two neurons that are directly adjacent to each other. This is important for neighbor relations. See below for details. Hence, a closed row topology will convert the row into a circle, and a closed rectangular topology will convert the rectangle into a torus. Please notice that in the latter case, you will no longer map a multi-dimensional input space onto a two dimensional, but on a three dimensional space.

The parameter **Training Runs** determines the number of classification runs, i.e. the number of times the training patterns are propagated through the network. The more runs there are, the more exact the classification is going to be and the more time the operator needs to finish.

The **Learning Rate** parameters determines the learning rates for the **winner** and the winner's **neighbor neurons**. The winner is the neuron that in each classification run is closest to a given pattern. The neighbor neurons are those neurons that are adjacent (or, in other words, bidirectionally connected) to the winner neuron. The learning rate determines how far the weight vector of the respective neuron is "pulled" into the direction of the input pattern. Note that a non-zero winner learning rate together with a neighbor learning rate of zero represents a WTA-network (winner-takes-all-network), in which only the winner neuron is updated. A zero learning rate for both winner and neighbors causes the network not to learn at all; the output after all training runs have been completed is the same as the initial state of the network.

The threshold value and the weights of each neuron need to be randomly chosen when the network is invoked. **Maximal Value** is the parameter that limits the maximum value, the threshold and weights of each neuron may not exceed. A value of -1 lets the network itself compute a maximum sensible value. Automatic computation might take very long depending on the input dimensionality. Modifying this parameter makes sense when the scope of the input is very narrow. If the input patterns have a relatively wide scope, choose a relatively large value or -1 to avoid unsatisfying classification results.

In the next field, you can choose a **Metric**. This is the method of distance measurement that is the underlying foundation of the algorithm's clustering process. You can choose

between "Euclidean Distance", "Manhattan Distance" or "Nominal Distance". Select that distance that provides best results with regard to your feature vectors.

Some extracted features might not be important for the clustering process, since they have a very small absolute value. To find out which features play a minor role, you can use the **Filter Patterns** option to filter out those patterns with values close to zero. The filtering is performed by setting every feature with an absolute value smaller than one – $|x|<1$ – to zero.

The next Parameter allows for a **Normalization** of the **Patterns**. Depending on the underlying feature set, feature values can be of a very broad scope in same cases . In that case, features with a very high absolute value might play a more important role during the similarity measure than those features with a small absolute value. To avoid that behavior, the user can choose to normalize the input vectors. This is done by dividing each pattern $x_{i,j}$, with i being the pattern number (i.e. row in the pattern matrix) and j being the column (i.e. a certain feature), by the root mean square deviation of the column, σ_j.

The last parameter – **Verbosing** – is lets the user toggle the Verbosity. If checked, the operator will print detailed information about what is going on on Yale's Log Console. These information will also be written into the experiment's log file. Uncheck this box if you wish to have a slim, more stream-lined output. The result output will remain the same in both cases.

Section 5

Using the operator

For every experiment, keep in mind that you can change the log verbosity level as well as the experiment's log and result files in the root operator of the respective experiment. The log verbosity only changes the amount of information that is printed on Yale's Log

Console. The Experiment Log file and the Result file store information about the experiment. This is persistent; you can view and evaluate the outcomes of the experiment after you closed Yale by referring to these two files.

5.1 Sample experiments

For this operator, there are sample experiments available. You can find these example experiments either in the full package of the plug-in or by downloading the sample experiment package.

The sample experiments come together with feature vector files. In these files features are stored that have been extracted from a number of audio files. These can be found in the “feature_vector_files“ directory. The small packs contain extracted features from a set of twenty files, the medium packs from 100 files, the large pack from 360 files, and the huge pack from 1,554 files. The feature vector files are meant to be used with the experiment “simple_classification“ if no ValueSeriesPlugin is installed and/or you do not want to extract features of music files yourself.

5.1.1 simple_classification.xml

Start the Yale Learning Environment and open the experiment “simple_classification.xml“. An experiment with the Self-Organizing Map Operator as the only operator is then loaded. "simple_classification.xml" classifies patterns into similarity clusters.

To run this experiment, choose either one of the provided feature vector files or an own feature vector file. As the Query Input File, choose the file of which you are interested in finding similarity clusters. If this file is not available (e.g. you use one of the provided feature vector files and did not extract your own features from files), simply create a new file with 0 bytes of size. Do that by typing `touch "filename"` on Linux systems or on Windows systems `mk "filename"`. On Windows system you can also right-click somewhere on your desktop and choose `"New -> New Textfile"` and type in a file name. In any case, make sure the file name occurs in the list of processed files. To find that out, simply load the feature vector file you use in your favorite text editor and look for a file name. The names of the processed files should be saved as the last element in each row.

Once all mandatory input requirements are met, feel free to change the rest of the

parameters according to your preference and/or input patterns. Once you are done, click the "Run experiment" button in Yale (looks like a standard play button) or hit `ALT+R`. When the experiment is finished, Yale will automatically bring up the Result-Tab and display the results of the computation.

If the results are not satisfying, consider changing the amount of neurons in rows and columns, changing the Closed Topology value and the learning rates for winner and neighbors.

5.1.2. simple_extraction.xml

This experiment only extracts a generic feature set of a number of audio files. The audio files must be in a generic MPEG-1 Layer III (mp3) with a sampling rate of 44.1KHz, as originally developed by Fraunhofer Institute of Germany and Thomson Multimedia SA of France, described in ISO/IEC11172-3. Although pretty much any bitrate should work, the bitrate needs to be constant in the whole file (i.e. files encoded with a variable bitrate might not work).
As mentioned above, the extracted feature set is a generic one. In multiple runs of a evolutionary algorithm, this feature set has proven to provide the best results to extract a suitable feature set from audio files for a variety of different tasks. It is important to mention that this file is equivalent to the experiment "music_feature_extraction.xml" that can be found in the example experiment package for Yale's ValueSeriesPlugin, version 3.2 or higher. In order to run this experiment, you need to have the ValueSeriesPlugin, version 3.3 or higher installed. Refer to section 3.2 to learn about installing plug-ins into Yale.

Before you run this experiment, make sure you have set the source_dir parameter in the Input operator to a directory containing music files that meet the above requirements. In addition, please set the example_set_file parameter in the experiment's Output operator. Keep this file in mind – this is the file the extracted features are stored in and that you might eventually want to load into the Self-Organizing Map Operator as the Feature Vector File.

5.1.3. extraction_and_classification.xml

After having started Yale, open the experiment "extraction_and_classification.xml". This

experiment contains a generic feature extraction algorithm to extract features from a corpus of digital music files and classifies them with the SOM-operator. Essentially, this experiment is a combination of the experiments "simple_extraction.xml" and "simple_classification.xml". You need to have the ValueSeriesPlugin v3.3 or higher installed in order to use this experiment.

In the experiment's Input operator, enter the path to a directory containing music files that meet the requirements in section 5.1.2. in the source_dir parameter. Further down, in the Output operator, please enter a file to store the extracted features by setting the example_set_file parameter accordingly. **In The SOM-Operator, make sure you set the same exact file as the Feature Vector File as you set as the example_set_file in the Output operator.** This is very important because otherwise, no classification will be possible and the operator will terminate. The Query Input File should be one of the files in the directory you entered in the Input operator's source_dir parameter. Once you have entered all mandatory parameters as described above, you are ready to run the experiment.

5.1.4. extraction_method_finder_GP.xml

This experiment finds the ideal feature extraction algorithm on the basis of a source directory containing audio data in terms of genetic programming. As the other feature extraction experiments, this experiment requires the ValueSeriesPlugin, v3.3 or higher, to be installed on your system. In addition, your computer should meet the Genetic Programming requirements as described in section 2 of this manual to be able to run this experiment.

You need to enter a source directory containing music data files in the MusicExampleSource operator. To find out if a specific file or a set of files are valid, please refer to section 5.1.2. In the ValueSeriesGP operator, make sure to set the number of individuals to a sensible amount on the basis of the amount of files that are to be processed and the performance class of your computer. The number of individuals should be larger than the amount of files in the source directory.

Please note that the automatic feature extraction demands a large amount of time and space. Several days of runtime on very large corpora are not unusual.

In case of problems with any of the above experiments, please refer to section 6 or contact the author.

5.2 Creating and Modifying experiments

You can easily integrate the Self-Organizing Map Operator into your own experiments by creating new ones or modifying existing ones. To add this operator to an experiment, right-click on the root operator and choose "`New Operator`". You will find the Self-Organizing Map Operator in the "`Kohonen Networks`"-group. The "`New Operator`"-option is replaced by a "`Replace Operator`"-option if you right-click on an existing operator instead of the root operator. In that case, the marked operator is being replaced by the Self-Organizing Map Operator.

In order to use this operator in own experiments, please ensure that a feature vector file is already present before the experiment starts or a feature vector file is created during the experiment and saved to the hard disk before the SOM-Operator is used for the first time.

Section 6

Troubleshooting

The experiment "**extraction_method_finder_GP.xml**" terminates with an "**IllegalArgumentException**". The message detail is "n must be positive".

Reason. During the Genetic Algorithm, no syntactically correct individual could be created so that during the tournament selection, that selects the next individual to be copied into the next generation, no individual was in the pool to be randomly chosen.

Solution. Increase the amounts of individuals in the ValueSeriesGP operator of your experiment. Note that increasing the number of individuals results in an exponential increase in time and space.

The experiment "**extraction_method_finder_GP.xml**" terminates with an "**OutOfMemoryError**".

Reason. The Java Heap Size is exceeded. This might have two reasons. Either your computer does not have enough random access memory (RAM) or Java only uses a minimal amount of the available memory.

Solution. In the first case, increase the amount of physical RAM in your computer by purchasing additional memory. In the second case, chose one of the following solutions.

1. If you are running Yale from command line, simply run the java command with the option -XmxNNN where NNN is a random number smaller than the amount of RAM installed in your computer in megabytes. Here is an example:

```
"java -Xmx1536 -jar /path/to/yale/lib/yale.jar"
```

This will reserve exactly 1,5 gigabytes of memory for Java. Please note that there is no blank space between the "-Xmx" and the number. Make sure you don't reserve all the available memory for Java – the operating system needs memory, too.

2. If you are running Yale from a script or executable under Microsoft Windows, follow the following steps.

- Log into your system with an account with administrative rights.
- Right-click on "`My Computer`"
- Click on "`properties`"
- Click on the tab "`Advanced`"
- Click on the button "`Environment Variables`" or "`System Variables`"
- In the lower list called "S`ystem Variables`", click on "`New`" and enter "`JAVA_HOME`" as the name of the new variable and the path to your Java installation as the value. Example: "`C:\Program Files\Java\jre1.5.0_06`"
- Repeat the last step entering "`MAX_JAVA_MEMORY`" as the name and a random number bigger than 256 and smaller than your maximum RAM as the value of the variable.
- Close all windows, restart your operating system and start Yale using the YaleGUI.bat script in the /scripts/ directory of Yale.

The experiment "**extraction_and_classification.xml**" or "**simple_classification.xml**" does not find any clusters.

Reason. The Query Input File is probably not among the files that have been extracted, i.e. not in the source directory.

Solution. Make sure to choose one of the files in the source directory as the Query Input File. If you do not have access to these files, i.e. you use the sample feature vector files that came with the sample experiments, create a file with one of the file names in the feature vector files. Please refer to section 5.1.1. for a more detailed explanation.

The experiment "**extraction_and_classification.xml**" fails.

Reason. You probably did not enter the same file as the example_set_file and Feature Vector File in the experiment's Output operator and Self-Organizing Map operator.

Solution. Simply enter the same file for both parameters. Make sure you have write access to the directory you store this file in and that the entire file name, including path and extension are spelled correctly. Pay attention to letter capitalization, too.

In case of any further problems, please contact the author of this plug-in

Section 7

Extending this plug-in

Since this plug-in is published under the GNU Public License, version 2, you may redistribute and/or modify it according to your own will. You should have received a copy of the GNU-PL among with this package. If not, please contact the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA. You can obtain the source code of this plug-in by contacting the author or downloading it form http://www-lehre.inf.uos.de/~btenberg/.

Please be so kind to mention the original author and the original authors of Yale and its plug-ins in your work. If you want, feel free to send a copy of your work based upon this

plug-in to its original author and share your insights.
Make sure that you publish your work based on this plug-in under the GNU Public License, too, as required by the license itself.

To extend this plug-in, you need to extend or replace the file "operators.xml" that can be found in the full, source or binary package of this plug-in In addition to that, you need to add your classes to the classpath. Since this is a plug-in for the Leaning Environment Yale, extending it is similar to extending Yale. Please refer to the Yale Tutorial for further information. The tutorial can be obtained from http://yale.sf.net.

Section 8

References

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., and Euler, T. **YALE: Rapid Prototyping for Complex Data Mining Tasks.** In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2006), ACM Press, 2006.

Fischer, S., Klinkenberg, R., Mierswa, I., and Ritthoff, O. **YALE: Yet Another Learning Environment - Tutorial.** No. CI-136/02, Collaborative Research Center 531, University of Dortmund, Dortmund, Germany, 2002.

Mierswa, I.. **Value Series Processing with Yale User Guide.** From http://yale.sf.net

Mierswa, I., and Morik, K. **Automatic Feature Extraction for Classifying Audio Data.** In Machine Learning Journal, Vol. 58, pages 127-149, 2005.

Hacker, S. **MP3, the definitive guide.** O'Reilly & Associates, Inc., 2000.