Otto-von-Guericke University of Magdeburg

Faculty of Computer Science

Laboratory Internship

Implementation of a software to create interpretable Fuzzy Rules from Support Vector Machines

Author:

Steven Schwenke

February 20, 2009

Supervisor:

Christian Moewes

Department of Knowledge and Language Processing

Otto-von-Guericke University of Magdeburg

Universitätsplatz 2

D-39106 Magdeburg

Background of the project

This program is an implementation of the papers [1] and [2]. The considered papers contributed to the task of rule extraction from Support Vector Machines (SVM). Therefore, [2] creates SVFI (Support Vector Fuzzy Inference) rules based on support vectors of a given SVM by creating one rule per support vector. A rule consists of n clauses, where n is the number of dimensions. Each clause in a rule establishes a relationship between the dimension specific value of the vector and an input x. The class of the support vector is added after listing the clauses by the word "THEN" followed by the class and a coefficient which expresses the importance of that rule.

The following rule base has been learned from a simple two-dimensional data set with ten support vectors.

Rule 0: IF x0 isCloseTo 0.056 AND x1 isCloseTo 0.652 THEN 1.0 with 69.82
Rule 1: IF x0 isCloseTo 0.186 AND x1 isCloseTo 0.642 THEN 1.0 with 120.0
Rule 2: IF x0 isCloseTo 0.58 AND x1 isCloseTo 0.53 THEN 1.0 with 58.72
Rule 3: IF x0 isCloseTo 0.88 AND x1 isCloseTo 0.314 THEN 1.0 with 66.23
Rule 4: IF x0 isCloseTo 0.944 AND x1 isCloseTo 0.22 THEN 1.0 with 120.0
Rule 5: IF x0 isCloseTo 0.136 AND x1 isCloseTo 0.578 THEN -1.0 with 120.0
Rule 6: IF x0 isCloseTo 0.352 AND x1 isCloseTo 0.51 THEN -1.0 with 48.57
Rule 7: IF x0 isCloseTo 0.968 AND x1 isCloseTo 0.06 THEN -1.0 with 26.21
Rule 8: IF x0 isCloseTo 0.874 AND x1 isCloseTo 0.196 THEN -1.0 with 120.0
Rule 9: IF x0 isCloseTo 0.49 AND x1 isCloseTo 0.478 THEN -1.0 with 120.0

Obviously the interpretation of these rules is difficult because there is one rule for every support vector and each rule has n fuzzy clauses where n is the number of dimensions. Even for relatively small problems this approach leads to huge number of rules.

[1] enhances this SVFI approach by pruning the constructed rules both in number and in length. Additionally, the interpretation of the rules is incremented by using fuzzy partitions to describe the input space. The example above can be converted to the following four rules:

IF x0 is middle THEN CLASS=1.0 WITH 0.79
IF x0 is big AND x1 is small THEN CLASS=1.0 WITH 0.63
IF x1 is middle THEN CLASS=-1.0 WITH 0.79
IF x0 is big THEN CLASS=-1.0 WITH 0.79

The used example is named example_01 and is part of the downloadable archive.

The scope of operation of the program is

- Creating a SVM from raw data by using [3],
- Create Support Vector Fuzzy Inference Rules [2],
- Create interpretable Fuzzy Rules by using predefined fuzzy partitions [1]

[1] Stergios Papadimitriou and Konstantinos Terzidis. Efficient and interpretable fuzzy classifiers from data with support vector learning. Intelligent Data Analysis 9 (2005), page 527 to 550.
*[2] Yixin Chen and James Z. Wang, Support Vector Learning for Fuzzy Rule-Based Classification Systems, IEEE Transactions on Fuzzy Systems, Vol II, No. 6, December 2003 , page 716 to 728.*
[3] Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for support vector machines, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

Example workflow

The following example should give you an introduction on how to use the program. You can find the used files in /data.

After starting the program you see the main GUI. The upper half contains buttons to control the program, the lower half is a textbased output.
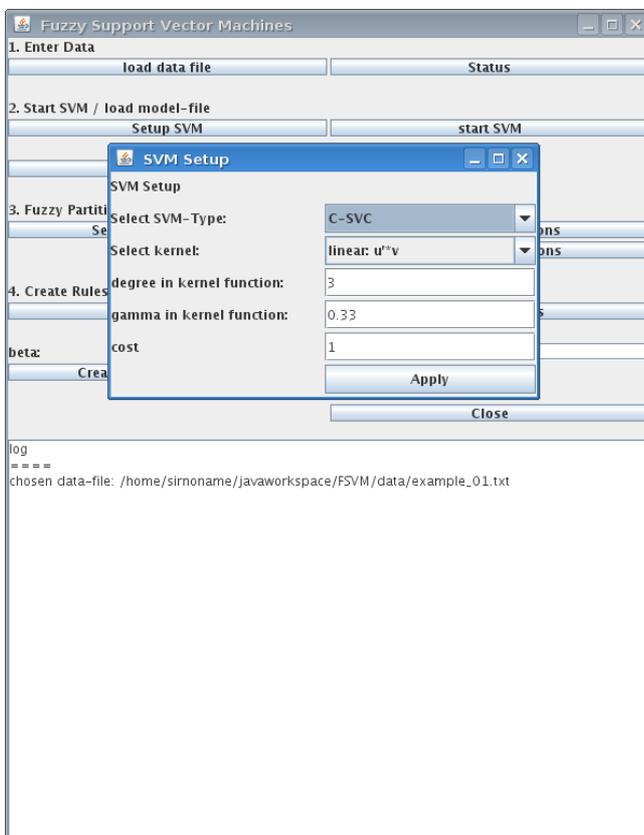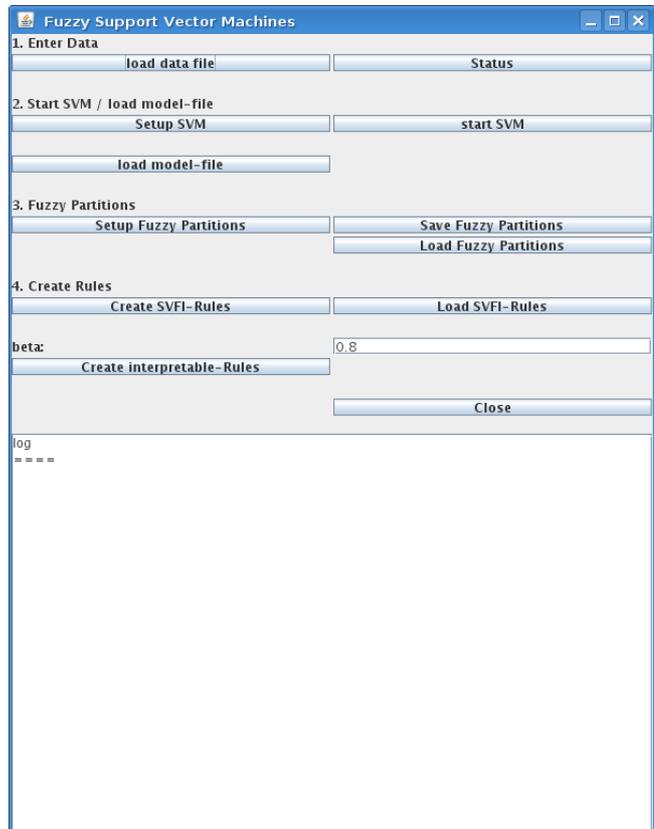
Let`s assume you only have raw data (no SVM) to start with. Load the example_01.txt in the data folder by clicking "load data file". The console should tell you that loading the file was successful. To set up the SVM click the according button. The dialog on the left appears.
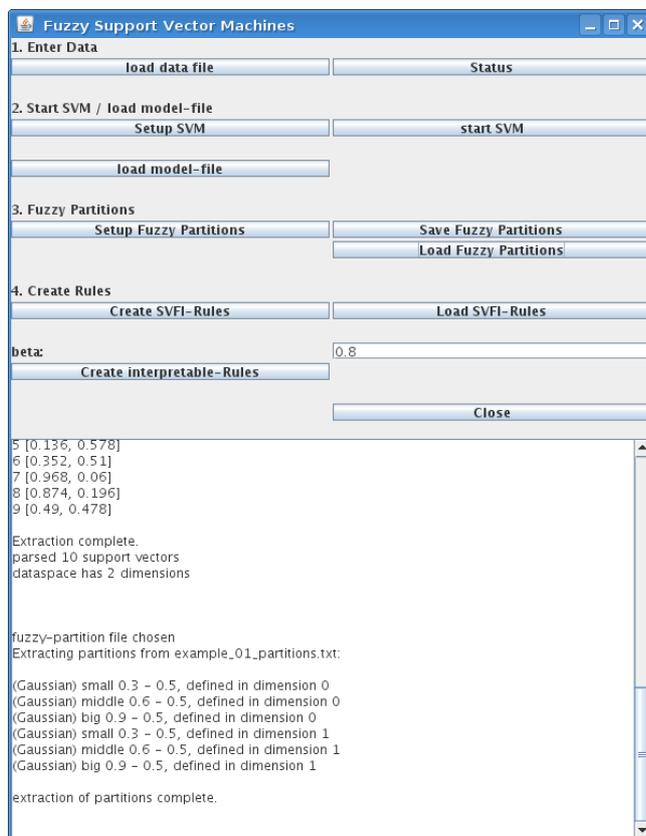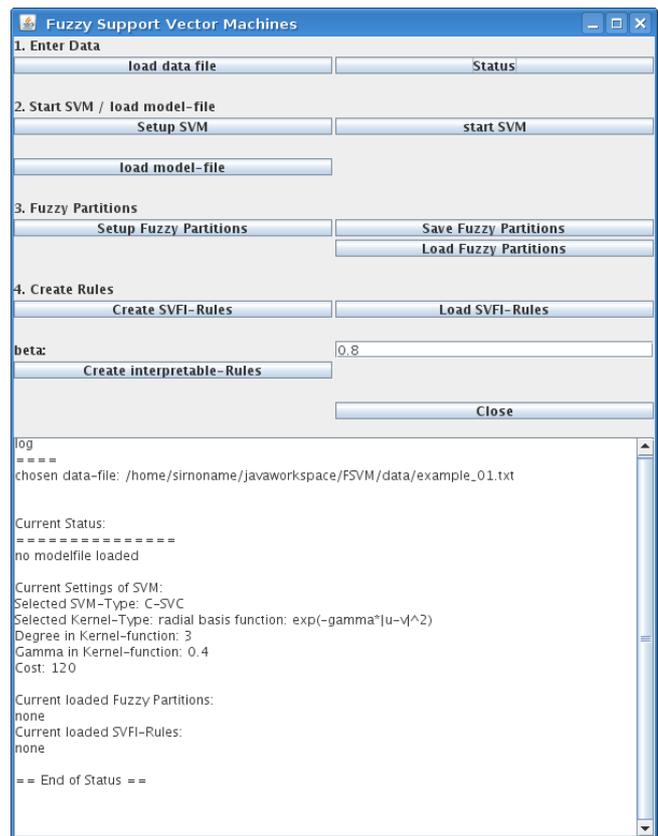
Here you can enter the parameters for the SVM. For this example we use the following setting:

SVM Type: C-SVC
SVM Kernel: Radial Basis Function
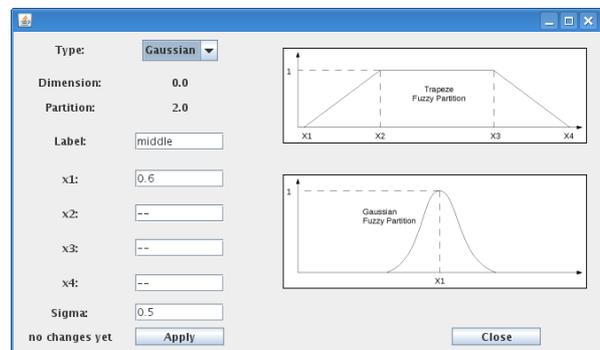*gamma in kernel function: 0.4*
cost: 120

Note that changes in these values are not mentioned by the console. If you want to get sure that you have entered the right values, click the button "status" in the upper right corner of the main interface. You can see every actual value of the program.

After setting up the SVM it can be started by clicking "start SVM". As you can read in the console, the modelfile has been saved to /data and directly loaded to extract the needed values.

To create human readable rules it is necessary to define fuzzy partitions which describe the input space adequately. You can define partitions from scratch up or load them by clicking "Load Fuzzy Partitions". In this example we load some gaussian partitions by loading the file example_01_partitions.txt via the button "Load Fuzzy Partitions". The log shows the loaded partitions as in the picture below.

Fuzzy Support Vector Machines

1. Enter Data
load data file   Status

2. Start SVM / load model-file
Setup SVM   start SVM
load model-file

3. Fuzzy Partitions
Setup Fuzzy Partitions   Save Fuzzy Partitions
Load Fuzzy Partitions

4. Create Rules
Create SVFI-Rules   Load SVFI-Rules
beta: 0.8
Create interpretable-Rules

Close

```
log
====
chosen data-file: /home/sirnoname/javaworkspace/FSVM/data/example_01.txt

Current Status:
===============
no modelfile loaded

Current Settings of SVM:
Selected SVM-Type: C-SVC
Selected Kernel-Type: radial basis function: exp(-gamma*|u-v|^2)
Degree in Kernel-function: 3
Gamma in Kernel-function: 0.4
Cost: 120

Current loaded Fuzzy Partitions:
none
Current loaded SVFI-Rules:
none

== End of Status ==
```

Fuzzy Support Vector Machines

1. Enter Data
load data file   Status

2. Start SVM / load model-file
Setup SVM   start SVM
load model-file

3. Fuzzy Partitions
Setup Fuzzy Partitions   Save Fuzzy Partitions
Load Fuzzy Partitions

4. Create Rules
Create SVFI-Rules   Load SVFI-Rules
beta: 0.8
Create interpretable-Rules

Close

```
5 [0.136, 0.578]
6 [0.352, 0.51]
7 [0.968, 0.06]
8 [0.874, 0.196]
9 [0.49, 0.478]

Extraction complete.
parsed 10 support vectors
dataspace has 2 dimensions


fuzzy-partition file chosen
Extracting partitions from example_01_partitions.txt:

(Gaussian) small 0.3 - 0.5, defined in dimension 0
(Gaussian) middle 0.6 - 0.5, defined in dimension 0
(Gaussian) big 0.9 - 0.5, defined in dimension 0
(Gaussian) small 0.3 - 0.5, defined in dimension 1
(Gaussian) middle 0.6 - 0.5, defined in dimension 1
(Gaussian) big 0.9 - 0.5, defined in dimension 1

extraction of partitions complete.
```

You can view and change every single partition by clicking "Setup Fuzzy Partitions", even after loading them from a file:

Fuzzy Partitions Setup

Current dimension: 0
refresh values
default fuzzy partition: - none -
amount of fuzzy-partitions: 3

Edit / Add a Fuzzy Partation   add a new partition
Edit / Add
Delete a Fuzzy Partation   small
Delete
Set default Fuzzy Partition: none
Apply Default Partition

Close

Type: Gaussian
Dimension: 0.0
Partition: 2.0
Label: middle

Trapeze Fuzzy Partition
X1   X2   X3   X4

x1:   0.6
x2:   --
x3:   --
x4:   --
Sigma: 0.5

Gaussian Fuzzy Partition
X1

no changes yet   Apply   Close

3

You can define, change and delete fuzzy partitions. Per dimension it is possible to define one default partition which will not appear in the generated rules. Please note that after adding or editing a partition you have to click the button "refresh values" to actualize the values in that dialog.

The defined partitions can be safed by clicking "Safe Fuzzy Partitions".

The next step is to create SVFI rules. The created rules should appear in the console, see the picture on the right.
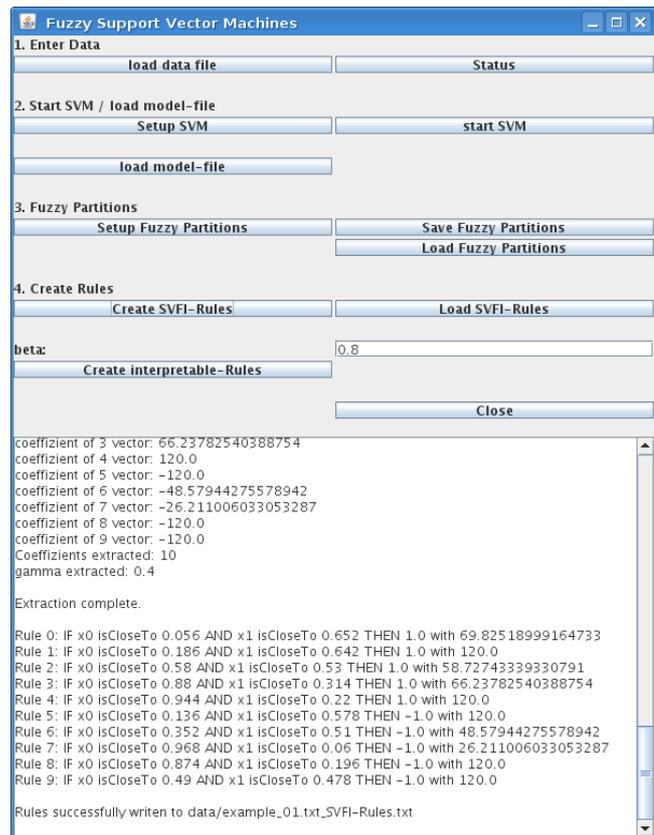
The beta-value which can be changed directly in the main interface is the membership treshold for a fuzzy clause to be added to a rule. This value has to be between 0 (every fuzzy clause is added to the rules) and 1 (most likely no clause is added) and has been added to the main interface to make experimenting easier. With a beta value of 0.4 we geht the following rules:

IF x0 is small AND x1 is middle THEN CLASS=1.0 WITH 0.5621071969972175.
IF x0 is small AND x1 is middle THEN CLASS=1.0 WITH 0.6181014321563288.
IF x0 is middle AND x1 is middle THEN CLASS=1.0 WITH 0.629907242042748.
IF x0 is big AND x1 is small THEN CLASS=1.0 WITH 0.635861373694327.
IF x0 is big AND x1 is small THEN CLASS=1.0 WITH 0.6260940338848473.
IF x0 is small AND x1 is middle THEN CLASS=-1.0 WITH 0.6026957890725707.
IF x0 is small AND x1 is middle THEN CLASS=-1.0 WITH 0.623011248307501.
IF x0 is big AND x1 is small THEN CLASS=-1.0 WITH 0.5621251847153232.
IF x0 is big AND x1 is small THEN CLASS=-1.0 WITH 0.6221545743566753.
IF x0 is middle AND x1 is middle THEN CLASS=-1.0 WITH 0.6031733132121143.

To lower the number of rules we simply have to increase beta. Wih beta = 0.78 we get:

IF x1 is middle THEN CLASS=1.0 WITH 0.7935812477446055.
IF x1 is middle THEN CLASS=1.0 WITH 0.7950745837864074.
IF x0 is middle AND x1 is middle THEN CLASS=1.0 WITH 0.629907242042748.
IF x0 is big AND x1 is small THEN CLASS=1.0 WITH 0.635861373694327.
IF x0 is big AND x1 is small THEN CLASS=1.0 WITH 0.6260940338848473.
IF x1 is middle THEN CLASS=-1.0 WITH 0.79711258224591.
IF x0 is small AND x1 is middle THEN CLASS=-1.0 WITH 0.623011248307501.
IF x0 is big THEN CLASS=-1.0 WITH 0.7905397391871014.
IF x0 is big AND x1 is small THEN CLASS=-1.0 WITH 0.6221545743566753.

4

Finally, with beta = 0.796 we get only 4 short rules:

IF x0 is middle THEN CLASS=1.0 WITH 0.7972465084092101.
IF x0 is big AND x1 is small THEN CLASS=1.0 WITH 0.635861373694327.
IF x1 is middle THEN CLASS=-1.0 WITH 0.79711258224591.
IF x0 is big THEN CLASS=-1.0 WITH 0.7968065497763223.

A visualisation of a SVM with the used setting is provided by the program "svm_toy" [3]. The x0-axis is horizontal with 0.0 at the upper left of the picture and 1.0 at the upper right. The x1-axis is vertical with 0.0 at the upper left and 1.0 at the lower left.
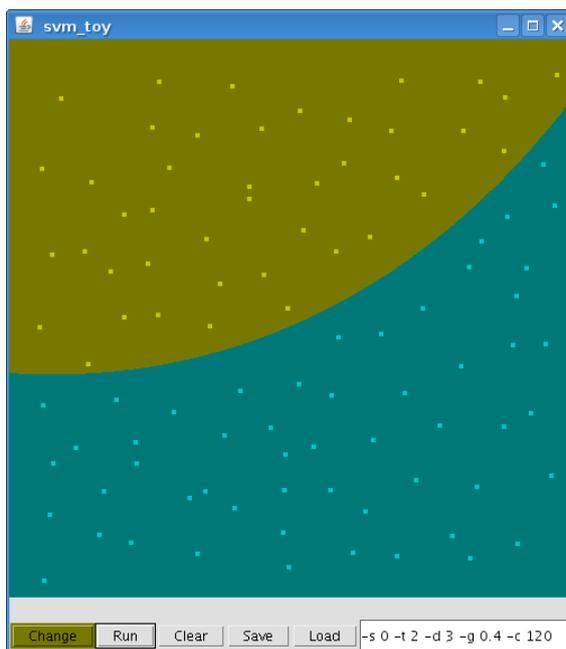
Filetypes

You will notice that during the work with the program several files has been saved to the data-folder:

Partition files

These files contain the fuzzy partitions. The following file is part of the archive:

(Trapeze) high 0.5 - 0.5 - 0.75 - 0.75, defined in dimension 1 as default
(Trapeze) very_high 0.75 - 0.75 - 1.0 - 1.0, defined in dimension 1
(Gaussian) medium 0.5 - 1.0, defined in dimension 0
(Gaussian) medium 0.5 - 1.0, defined in dimension 1

The first word of each line indicates the type of the partition, followed by the label, the parameters and, if given, the "default" flag. Trapezoidal partitions have four parameters which are the points of the trapeze in ascending order. Gaussian partitions only have two parameters: The center of the gaussian curve and it`s sigma. If a line ends with "as default" this partition is a default partition and will not appear in the rules.

SVFI Rule Files

Here the constructed SVFI Rules are stored. The rules look like

Rule 0: IF x0 isCloseTo 0.056 AND x1 isCloseTo 0.652 THEN 1.0 with 69.8
Rule 1: IF x0 isCloseTo 0.186 AND x1 isCloseTo 0.642 THEN 1.0 with 120.0

Papadimitriou Files

The final interpretable rules are stored in files with the ending "Papadimitrou-Rules.txt" and look like

IF x0 is big AND x1 is small THEN CLASS=1.0 WITH 0.635861373694327.
IF x1 is middle THEN CLASS=-1.0 WITH 0.79711258224591.

Appendix (Downloads)

compiled version + Java-Code

The program consists of the following Java-files:

Dimension.java

Dimensions.java

FSVM.java

FSVMLib.java

FuzzyClause.java

FuzzyPartition.java

GUI_EdditAFuzzyPartition.java

Gui.java

SupportVector.java

SupportVectors.java

SVFIRule.java

The graphics were created using the following files:

picture_gaussian.gif

picture_gaussian.odp

picture_gaussian.xcf

picture_trapeze.gif

picture_trapeze.odp

picture_trapeze.xcf

All these files are included in the downloadable version of the software. Additionally, there is a folder named "documentation" which contains details about all classes.

Example 01

This dataset demonstrates the basic function of the program and the implemented algorithms.

The derived rule base does not give a sufficient interpretation of the dataset:

IF x0 is middle THEN CLASS=1.0 WITH 0.79
IF x0 is big AND x1 is small THEN CLASS=1.0 WITH 0.64
IF x1 is middle THEN CLASS=-1.0 WITH 0.79
IF x0 is big THEN CLASS=-1.0 WITH 0.79

In fact, the interpretation of the rules can easily lead to fundamental misunderstanding of the dataset: If x1 is middle, the class should be -1. In fact this is wrong because the seperating hyperplane goes right through that area.
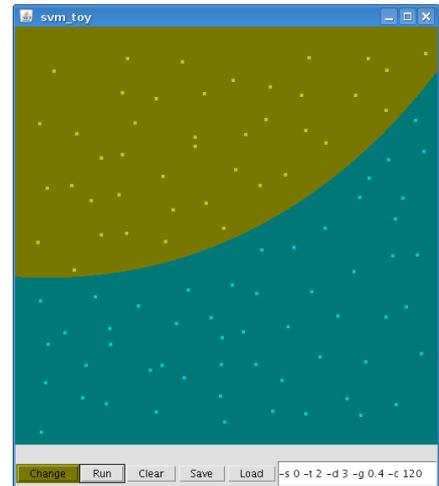
Example 02

This dataset is generated and describes the XOR function. [1] states to extract the four XOR rules out of (another!) generated XOR dataset, which could not be confirmed from us. The expected result is:

If x0 is small and x1 is small then class = 1
If x0 is big and x1 is big then class = 1
If x0 is big and x0 is small then class = -1
If x0 is small and x1 is big then class = -1

Because in [1] every clause has to have a bigger membership to a specific fuzzy partition as beta, more and more clauses are denied with increasing beta. That leads to rules with only one clause:

IF x0 is medium THEN CLASS=1.0 WITH 0.99
IF x1 is medium THEN CLASS=1.0 WITH 0.99
IF x0 is medium THEN CLASS=-1.0 WITH 0.99
IF x1 is medium THEN CLASS=-1.0 WITH 0.99

The used partitions has been extracted from the pictures in [1].

From these rules the original XOR function, as stated in [1], can not be derived.
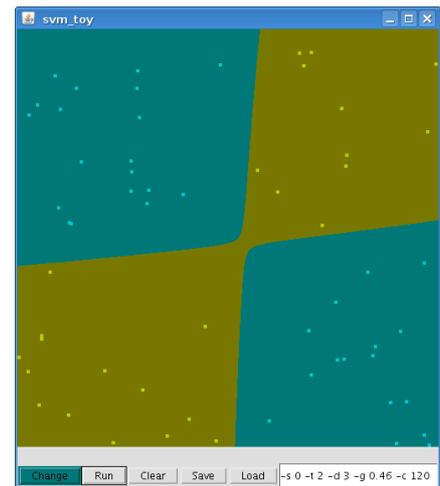
Example 03

This dataset demonstrates a seperatet class surrounded by another class. The expected result are statements like

If x0 and x1 are middle, the class is 1, else the class is 0.

The derived rules are unable to provide such a statement because each rule specifies only one dimension:

IF x1 is small THEN CLASS=-1.0 WITH 0.79
IF x1 is small THEN CLASS=-1.0 WITH 0.79
IF x0 is middle THEN CLASS=-1.0 WITH 0.79
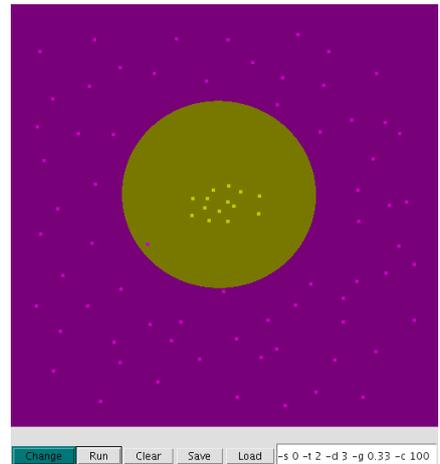
FAQs

- Q: The program crashes while loading a data- or a model-file. Why?
- A: The correct structure of data- and model-files, which both has to be libSVM-conform, is not verified. If the program crashes while loading these files, check their structure.

- Q: The program reports "number of dimensions doesnt`t match" while reading a modelfile. What does that mean?
- A: If the value of a support vector is zero, the libsvm-modelfiles doesn`t store that value (this is called "sparse format"). This format is not supported by the program so you have to add the zero-values.

- Q: The program mixes the values of my modelfile, why?
- A: Make sure that the values in your modelfile are ascending (lowest dimension first, highest last) and separated by colons with the dimension. This is the right format: "1: 0.34 2: 0.12 3: 0.20".

- Q: Where are the generated files like fuzzy-partitions or modelfiles stored?
- A: All generated files are stored in /data in the program-folder.

- Q: The files are named curious: There is "model" and "rules" in the name of one file. How can I make the program name the files in the right way?
- The program uses the name of the data- or the modelfile for naming the rule-files. The file with raw data should have no ending (no .txt) and the modelfile should end with ".model".

Credits

This software and all it`s files, except all files of the libsvm [3], has been created by Steven Schwenke and can be downloaded at http://stevenschwenke.de/index.php?section=studies&subsection=projectFuzzySupportVectorMachine .

The project, which has taken place from november 2008 to february 2009, was supervised by Christian Moewes.

Copyright