Quick-Start-Guide: Using the software tool 'uSort' (aka 'solver')

v1.0
rev. 2 05.08.2023

URL: https://github.com/dsandersGit/GIT_Solver

Content

What it is:

uSort is a tool that uses machine learning approaches for learning how to predict various classes of samples. The necessary data consist of a set of samples, each exhibiting a class-label (a classes descriptive word) and multiple determinable numeric variables (aka features). The following table sketches such a data set using an extract of the well known IRIS dataset (https://en.wikipedia.org/wiki/Iris_flower_data_set).

Sepal_Length	Sepal_Width	Petal_Length	Petal_Width	Species / Class
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3	1.4	0.2	Iris-setosa
4.7	3.2	1.3	0.2	Iris-setosa
7	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor
7	3.2	4.7	1.4	Iris-versicolor
6.4	3.2	4.5	1.5	Iris-versicolor
6.9	3.1	4.9	1.5	Iris-versicolor

Fig. 1: Extract of the Iris data set

'Species/Class' is the class-label, the other columns are determinable characteristics (here: morphologic variations of the iris flower).
Aim of the machine learning process is the discovery of characteristic relations of variables that is linked to a specific class. Such relations will be called 'models'.
When models are found, they can be applied on new, unclassified data to predict the respective class. Such models can be saved and applied later for classification.

'uSort' enables both model discovery and classification.

What you need to run the app:

i) *'uSort.jar'* is a JAVA runnable JAR file. For execution the JAR file must be opened with a JAVA runtime environment (JRE). This can be installed on a computer or the JAR file can be opened with JRE's 'javaw.exe' (windows, *'uSort' should be OS independent)*. Free JRE's can be found online.

ii) Data. For a quick-start you should find the Iris data set together with the app. Use this to get a feeling.

If you want to try your own data, concern the following:

- arrange your samples with class-label and variables in a line using your preferred spreadsheet app. A header is optional.
- save as 'comma separated file' *.csv
- class-label can be at any column position, only one class label is allowed
- number of variables should be smaller than number of samples. Otherwise any discriminant analysis will tend to 'overfit' and find models that does not generalize new unclassified samples. If possible, remove highly correlated variables (>redundant information). Spearman rank correlation could be beneficial.
- minimum number of classes is 2

How to run:

Start the app

Start the app (double click on uSort.jar if JRE is installed, otherwise set the paths in cmd.exe, type <JRE PATH>\javaw.exe" -jar "<APP PATH>\uSort.jar" %)
in case of problems: net search for 'run a java jar file'

Load data

A user interface pops up. Select the data set file you want to process for learning (menubar / iconbar right-hand-side). The |data|-tab will show selected features of the selected data set:

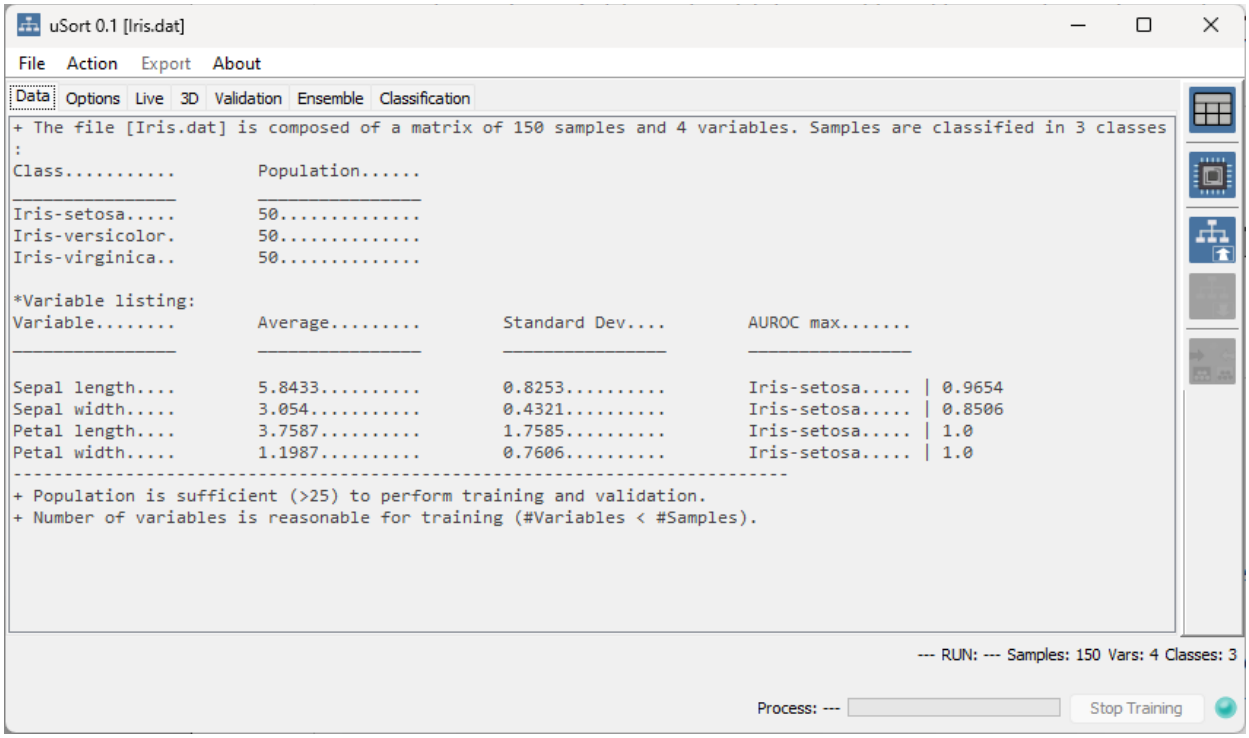


Fig. 2: |data|-tab when iris data set is loaded

Note: The 'AUROC max' (highest value for single class, AUROC - area under receiver operator characteristics) is only calculated for smaller sample / variable number due to calculation times

Set options

Select the |Options|-tab:

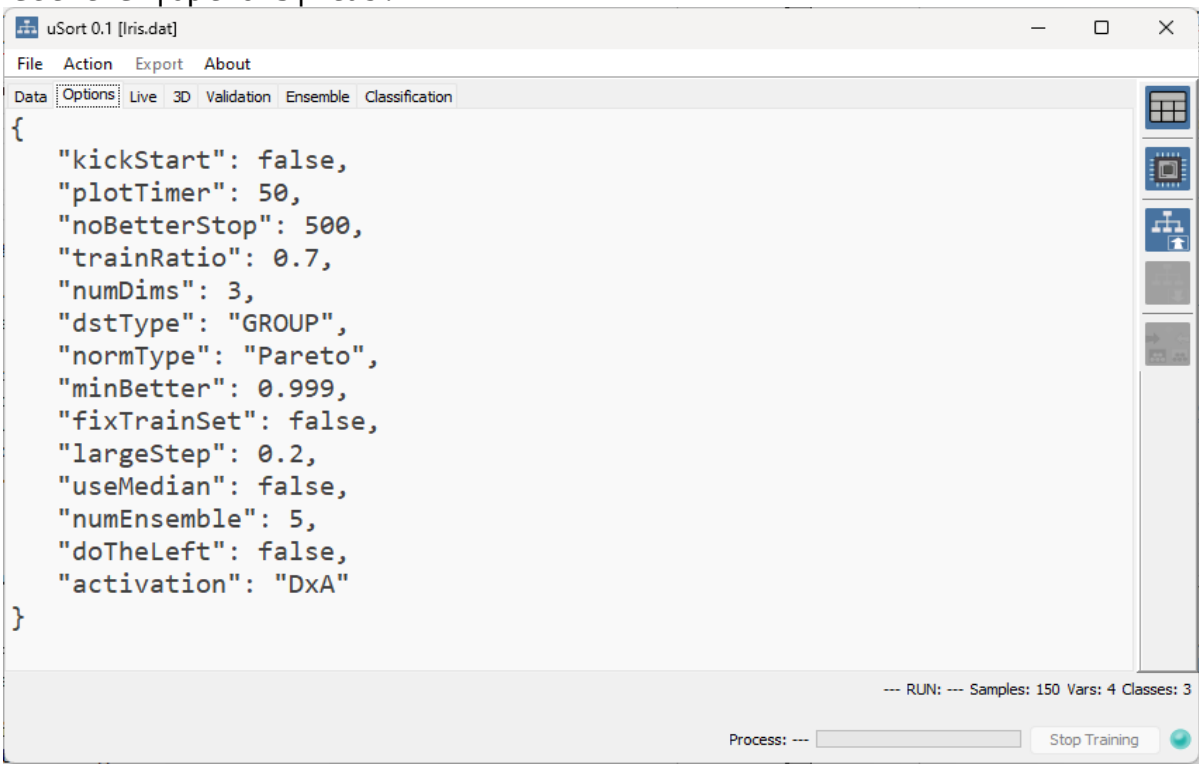


Fig. 3: |options|-tab [default]

Options affects the learning/training process. Important options are:

- **numEnsemble**: number of cycles that are trained. Together they form an 'ensemble' where multiple models are used for classification. Higher number might raise accuracy.
- **noBetterStop**: learning/training will stop when no improvement of model accuracy/bonus is achieved. Number determines how many unsuccessful steps are tolerated. Higher number > longer training time, but might improve accuracy
- **numDims**: Algorithm is multivariate, numDims sets the number of dimensions. Complex data / multiple classes might benefit from higher dimensionality
- **fixTrainSet**: samples are split in training and testing subsets. Learning algorithm only uses the train set while the test set result indicates the generalization potential of models. A fixes train/test set can be selected 'true' for the whole ensemble, or for each model an new train/testing subset is created
- **trainRation**: determines the ratio of the number of training to testing samples. 0.7 > 70% of sample will be used for training, 30% for testing. Ration is applied for each class. For small sample numbers splitting train / testing is not recommended.

Change options in the text. For boolean options use true/false. Options are processed as JSON format and will be stored together with your

ensemble.

Run a training

Switch to the |Live|-tab, then start 'train' (menubar / iconbar right-hand-side):

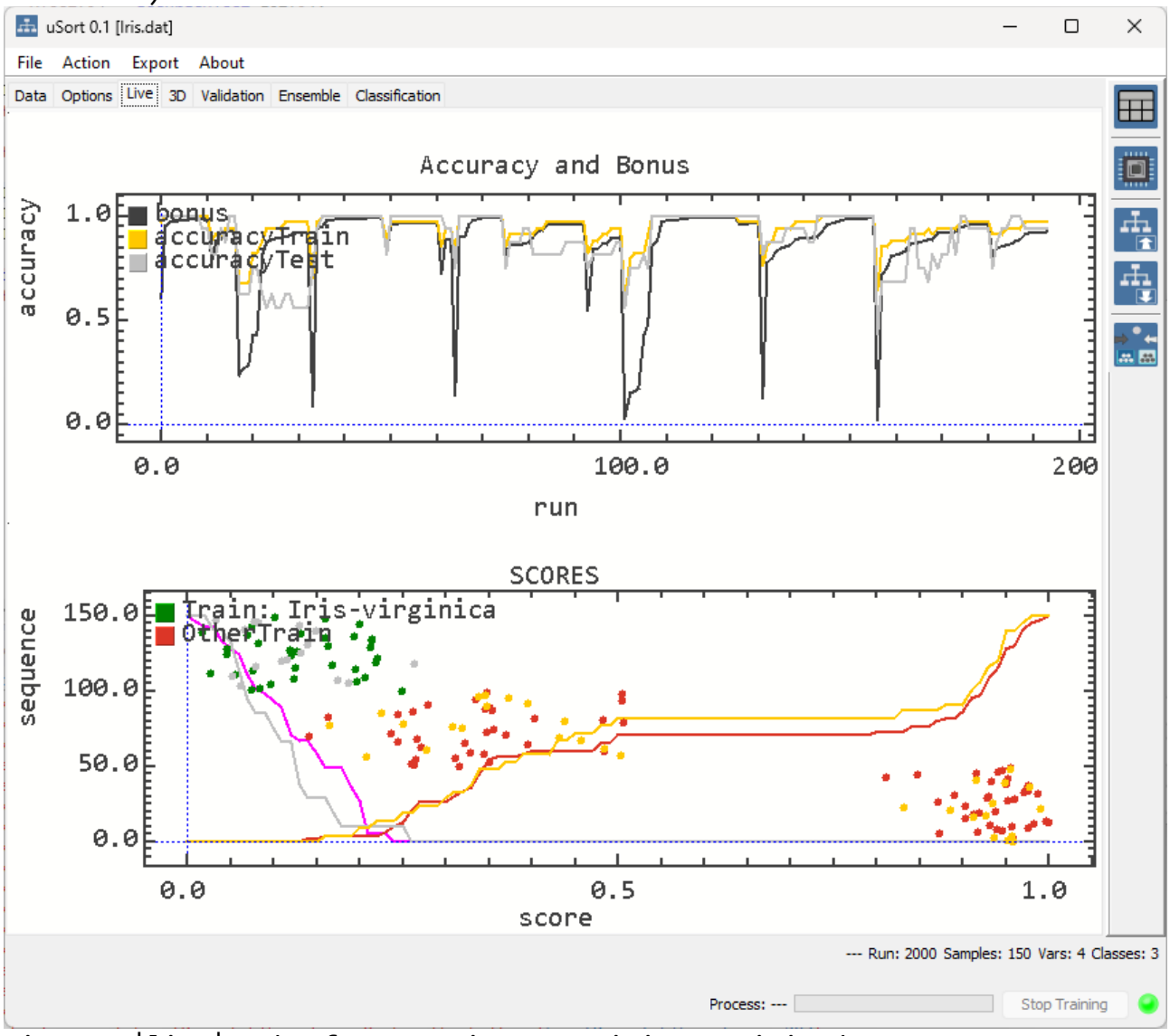
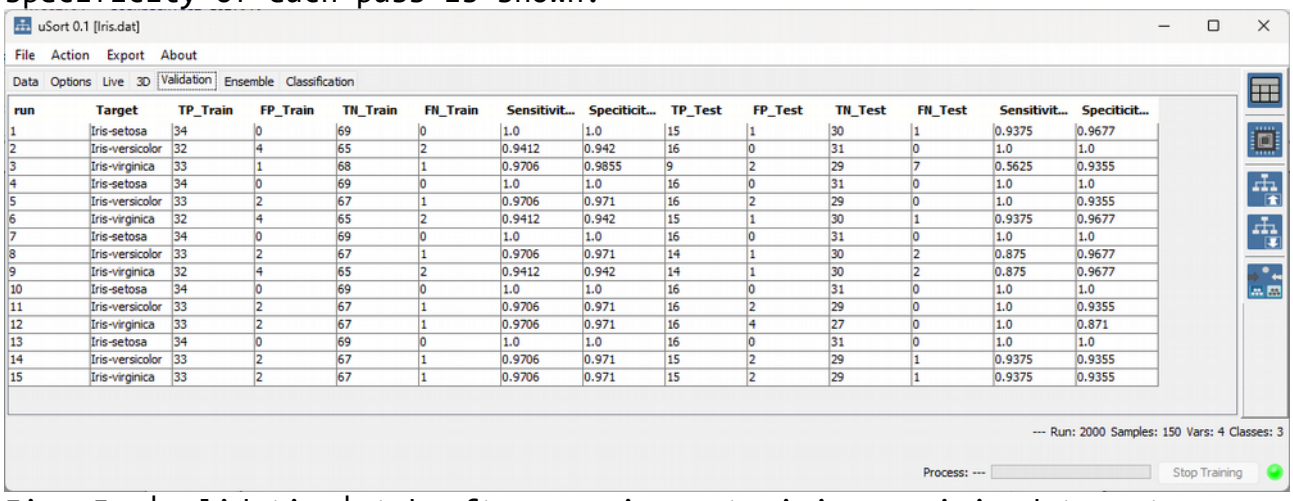


Fig. 4: |live|-tab after running a training on iris data set

Learning / Training is visualized live. Top graph plots accuracy of the classification for both training and testing data on the run, furthermore the underlying algorithm's bonus is shown. Only improved bonus is plotted.
Well aligned 'accuracyTrain' and 'accuracyTest' indicates good generalization potential.

Bottom graph plots the samples sequence (number/sorting) on the 'score'. The score is the result of the discrimination algorithm. In each pass only one sample class is set as target, and the algorithm tries shifting the target to score 0 while pushing all other samples towards 1. When no more improvement is found the model is stored and the target will be set to the next class.

Next to |live|-tab progress can be followed looking at the |validation|-tab. Here, the detailed accuracy of each pass is listed live. Both training and testing samples are analysed on the model's performance by true positive (TP) false positive (FP) true negative (TN) and false negative (FN) classification. Furthermore, the sensitivity and specificity of each pass is shown.

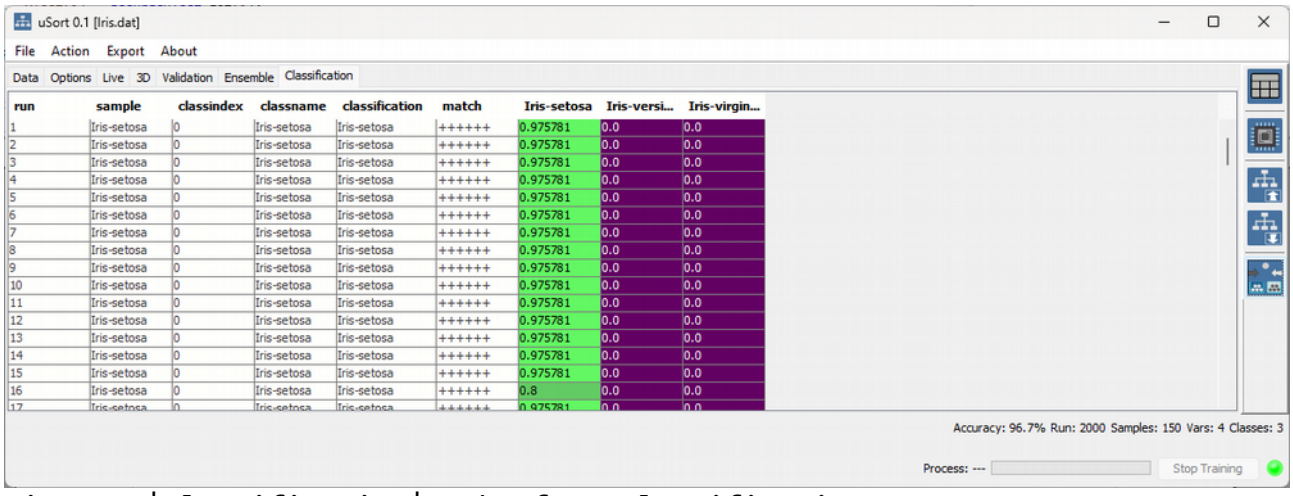


run	Target	TP_Train	FP_Train	TN_Train	FN_Train	Sensitivit...	Speciticit...	TP_Test	FP_Test	TN_Test	FN_Test	Sensitivit...	Speciticit...
1	Iris-setosa	34	0	69	0	1.0	1.0	15	1	30	1	0.9375	0.9677
2	Iris-versicolor	32	4	65	2	0.9412	0.942	16	0	31	0	1.0	1.0
3	Iris-virginica	33	1	68	1	0.9706	0.9855	9	2	29	7	0.5625	0.9355
4	Iris-setosa	34	0	69	0	1.0	1.0	16	0	31	0	1.0	1.0
5	Iris-versicolor	33	2	67	1	0.9706	0.971	16	2	29	0	1.0	0.9355
6	Iris-virginica	32	4	65	2	0.9412	0.942	15	1	30	1	0.9375	0.9677
7	Iris-setosa	34	0	69	0	1.0	1.0	16	0	31	0	1.0	1.0
8	Iris-versicolor	33	2	67	1	0.9706	0.971	14	1	30	2	0.875	0.9677
9	Iris-virginica	32	4	65	2	0.9412	0.942	14	1	30	2	0.875	0.9677
10	Iris-setosa	34	0	69	0	1.0	1.0	16	0	31	0	1.0	1.0
11	Iris-versicolor	33	2	67	1	0.9706	0.971	16	2	29	0	1.0	0.9355
12	Iris-virginica	33	2	67	1	0.9706	0.971	16	4	27	0	1.0	0.871
13	Iris-setosa	34	0	69	0	1.0	1.0	16	0	31	0	1.0	1.0
14	Iris-versicolor	33	2	67	1	0.9706	0.971	15	2	29	1	0.9375	0.9355
15	Iris-virginica	33	2	67	1	0.9706	0.971	15	2	29	1	0.9375	0.9355

Fig. 5: |validation|-tab after running a training on iris data set

The training result

After finishing, the developed model-ensemble is accessible. For testing it can be applied on the current data set (menubar / iconbar right-hand-side). Here, the whole ensemble, not individual models, are used to classify samples. Respective classes numbers indicate the 'frequency' of positive classification. The highest frequency determines the overall classification:



run	sample	classindex	classname	classification	match	Iris-setosa	Iris-versi...	Iris-virgin...
1	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
2	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
3	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
4	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
5	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
6	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
7	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
8	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
9	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
10	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
11	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
12	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
13	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
14	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
15	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.975781	0.0	0.0
16	Iris-setosa	0	Iris-setosa	Iris-setosa	++++++	0.8	0.0	0.0

Fig. 6: |classification|-tab after classification

The developed model ensemble can be viewed in the |ensemble|-tab. This is the instruction on how to process new, unclassified sample's variables, towards estimating the class belonging.

Save the ensemble for further classification jobs.

Run a classification

Load a data set. Load a classification ensemble. Variables in both must be equal. Run a classification.

The algorithm

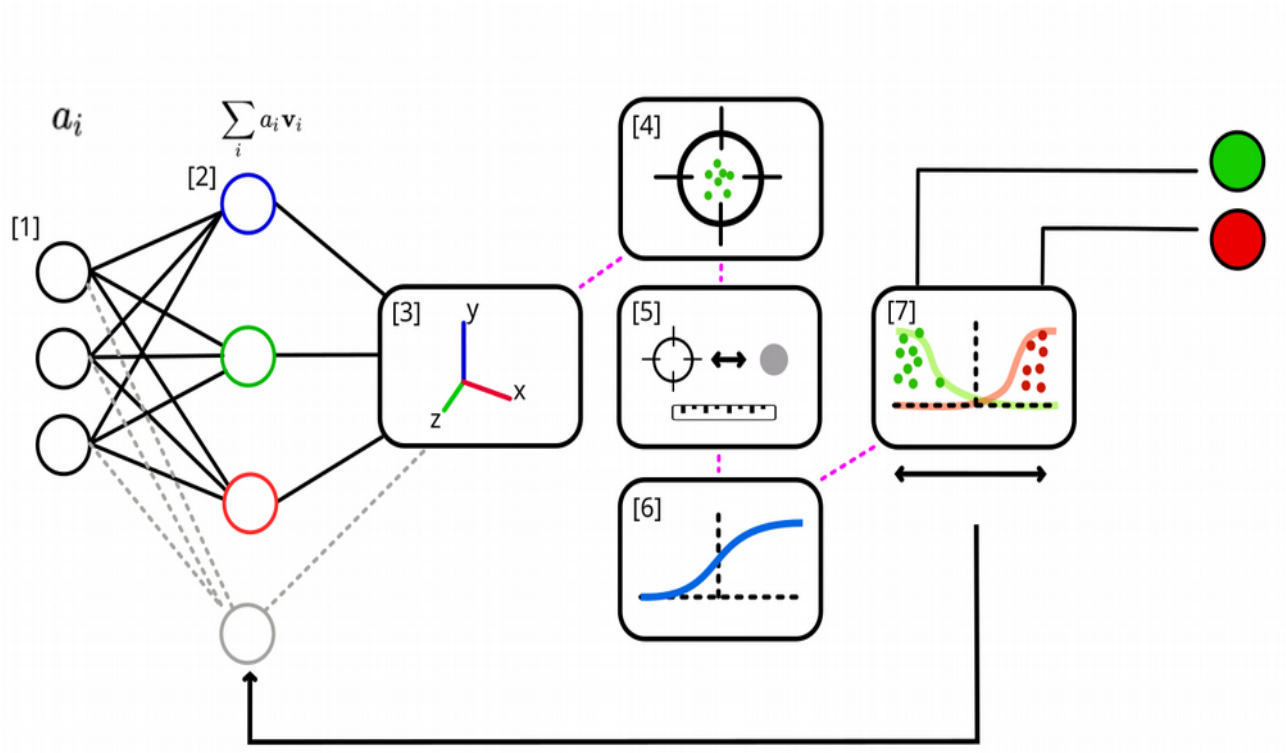


Fig. 7: Sketch of the algorithm used for classification training

In each pass one class is set as target. All other classes are equally non-targets. Aim of a pass is to separate targets from non targets based on characteristic relations of variables.

Variables are the fundamental data [1]. They are multiplied initially by random vectors in a linear combination [2]. Independently, multiple linear combinations are calculated (see 'numDims') [2]. The results are considered as multi-dimensional points for each sample [3]. The target class's average/median (in each pass one class is set as target) is set to 'center' [4]. The distance to the 'center' is determined for all samples [5]. All distances are weighted by a sigmoidal function [6] to produce scores between 0 (close to target) and 1 (far away from target). A split between target/non-target is determined for classification [7]. Back-propagation via adjustment of vectors [7]>[2] is used to increase separation (>bonus).
If no further improvement of separation is achieved for steps (see 'noBetterStop') the current pass is finished. The respective results make

up one model. Next class is set as target and a new pass is initiates until all classes are passed through (cycle). Depending on the 'numEnsemble' value cycles is are repeated to finally form the model ensemble.