LVQTools documentation
Bachelor project: implementing LVQ in R

Sander Kelders

June 22, 2010

Introduction

This document contains a descrpition of the LVQTools-package. This package is part of a bachelor project implementing several LVQ-algorithms for the Intelligent Systems Group of the University of Groningen. To stimulate the usage of LVQ in biomedical applications the statistical language R has been chosen for its wide usage in biomedics.
The LVQTools-package implements the LVQ1-algorithm using local or classwise relevances-vectors or matrices or no relevances at all. Different distance-measures, including euclidean and manhattan, are also possible. In addition some entropy-based distance measures are among the possibilities. All distance measures are available in a normal or generalized context. Several initialization- and normalization-schemes are available as well as multiple tools for validation and in/output for optimal versatility.

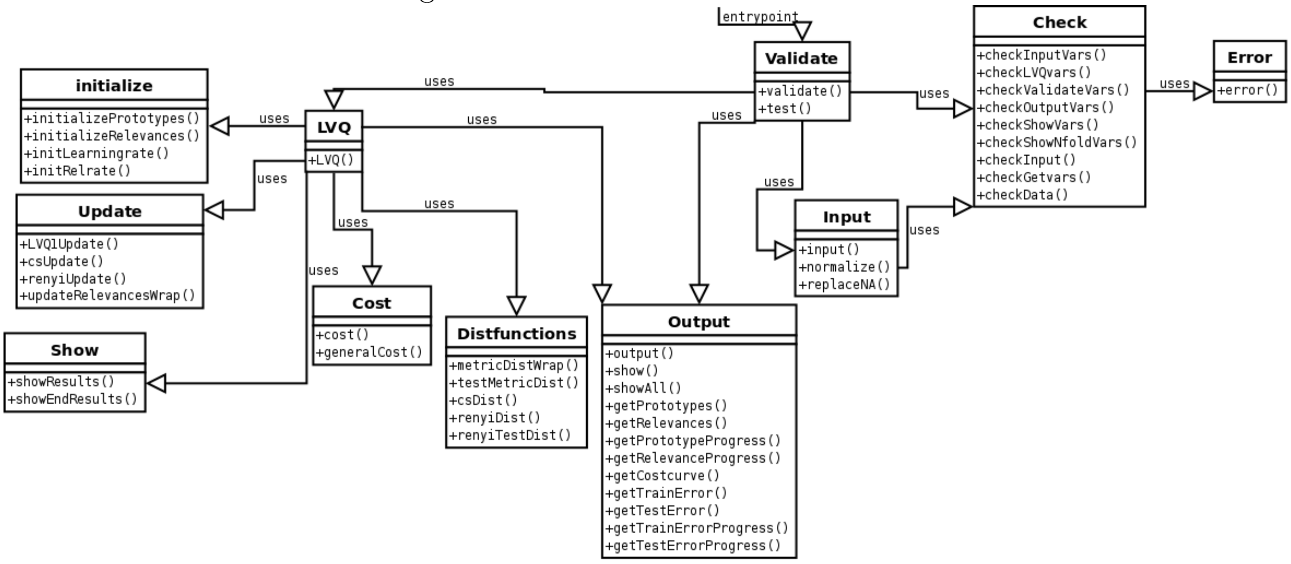
We start by giving an overview of the modules of the LVQTools-package. This is followed by a flow-diagram and an explanation. In the last and largest part each module is briefly described along with all the functions, with its parameters, it contains.

Overview

The package consists of functions which are grouped in modules by functionality Each module contains nonhelper-functions that are called by functions in other files and helper-functions. The exception to this are the `Validate` and `Output` -packages which also contain user-functions that can be called by the user. In *Figure 1: Overview* the modules are displayed along with their nonhelper- and user-functions.

Figure 2: Flow depicts the execution of a call to `validate`. Firstly, it is possible to specify input from a variable instead of letting it be read from file. These variables are checked for errors. Next, if there was no direct input, input will be read from file and, either way, the input is transformed according to the users specifications. Then, all other variables are checked for errors before starting the LVQ-algorithm. When using the `nfoldcross` scheme for nfoldcross-validation the LVQ-algorithm called more than once. At the start of the LVQ-run several variables are initialized and depending on the users specification the initial configuration is shown. Then the actual LVQ-algorithm is started and for each datapoint and epoch the following

Figure 1: Overview



actions are performed:

1. the distance, according to the specified distance-measure, from all prototypes to a certain datapoint is calculated.

2. prototypes are updated and if present so are the relevances.

3. for every epoch, if specified, the value of the costfunction, according to the LVQ-version, is calculated and stored.

4. for every epoch, if specified, the current configuration is shown.

5. for every epoch, if specified, the current configuration is shown.

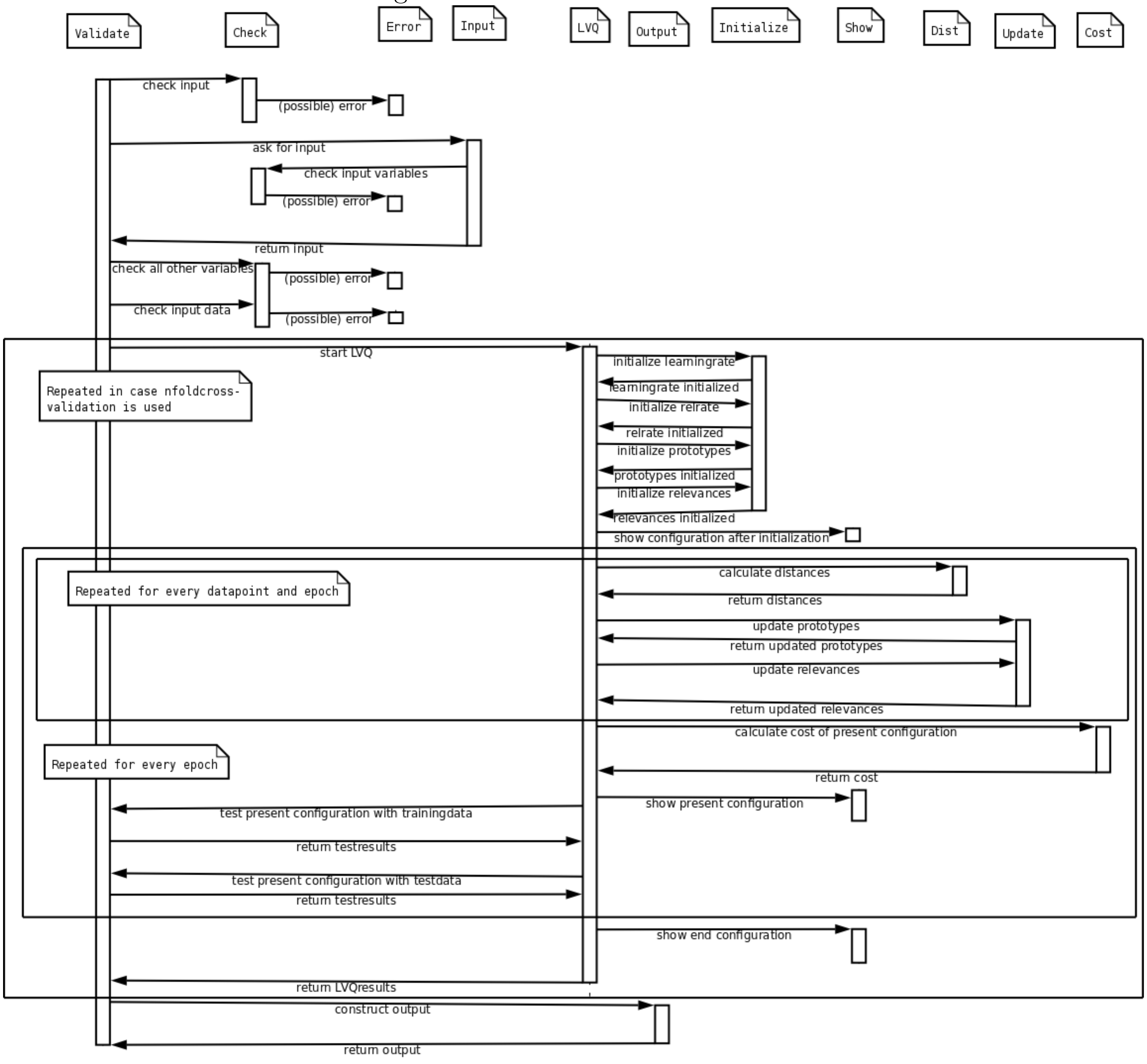
6. for every epoch, if specified, the current configuration is tested with the training and/or testset.

Eventually the results are returned and an output-object in generated.

Modules

This section contains a short description of each module along with all the functions it contains with their respective description.

Figure 2: Flow



Check

This module contains functions that check the input-variables for errors. If one or more errors are encountered the numbers of the errors are forwarded to the `error` module where an errormessage is generated.

Function	Description	Parameters	Description
`checkData`	This function checks the relevances and,possibly, normalized data for errors. This is the last check before LVQ is started.		
		`data`	The training- or test-data.
		`LVQscheme`	The version of LVQ to be used.
		`relevances`	The relevances provided by the user. `NA` when none are provided.
		`relevancescheme`	Determines the number of relevances used.
`checkGetVars`	This function checks the parameters used extraction from the output-object.		
		`LVQout`	The output-object.
		`fold`	The number indicating from which fold data needs to be extracted.
`checkInputVars`	This function checks the parameters used for input from file and normalization.		
		`normalizescheme`	Determines how the data should be normalized.
		`normalclasswise`	Determines which class will be used as a basis for normalization.
		`replaceNA`	Determines if `NA`-values should be replaced
		`replaceclasswise`	Determines if the replacement of `NA` -values should consider classes.
		`orglabellvls`	A help-variable. Contains the classlabels of the different classes.
`checkInput`	This function checks the train- and test -input directly provided by the user.		
		`traininp`	The data used for training, directly provided by the user.
		`testinp`	The data used for testing, directly provided by the user.
`checkLVQvars`	This function checks the parameters used for the LVQ-algorithm.		
		`prototypes`	A vector indexed by strings representing the classlabels. Each entry contains a number representing the number of prototypes to be used for the appropriate class.
		`learningrate`	The rate at which the prototypes will adapt. It contains either a number, between 0 and 1 or a vector of such numbers of length `epochs`.
		`epochs`	The number of times the training-data will be used to update the prototypes,
		`initscheme`	Determines the way the prototypes are initialized.

Function	Description	Parameters	Description
		`distscheme`	Determines what kind of measure is used to determine the distance from prototype to datapoint, when using the `LVQ1`-scheme.
		`relevancemode`	Determines the sort of relevances used.
		`relevancescheme`	Determines the number of relevances used.
		`LVQscheme`	Determines which LVQ-version will be used.
		`optimisationscheme`	Determines how the optimal situation will be reached and thus how the prototype and relevances will be updated.
		`relrate`	The rate at which the relevances will adapt. It contains either a number, between 0 and 1 or a vector of such numbers of length `epochs`.
		`customdist`	When using `distschemecustom` determines how the distance is calculated.
		`alfa`	When using `LVQscheme renyi` determines the version of Renyi-divergence to be used for calculating the distance.
		`show`	Determines whether or not progress should be shown during the training.
		`graphics`	Determines whether or not the trainingset and prototypes should be plotted during training. This is only available if the trainingset is 2-dimensional.
		`plotcurve`	Determines whether or not the progress of the constfunction should be plotted after each LVQ-run.
		`labellvls`	Helper-variable. Contains the classlabels of all classes.
		`dimensions`	Helper-variable. The length of a datapoint (minus the classlabel).
`checkShowNfoldVars`	This function checks for errors on the variables used to show output of a `nfoldoutput`-object.		
		`LVQoutput`	The output object of which data is to be shown.
		`protofold`	The fold of which the prototypes are to be shown.
		`relfold`	The fold of which the relevances are to be shown.
		`costfold`	The fold of which the progress of the costfunction is to be shown.
		`protoprogfold`	The fold of which the progress of the prototypes are to be shown.
		`relprogfold`	The fold of which the progress of the relevances are to be shown.
		`trainerrorfold`	The fold of which the progress of the trainerror is to be shown
		`testerrorfold`	The fold of which the progress of the testerror is to be shown

Function	Description	Parameters	Description
`checkShowVars`	This function checks for errors on the variables used to show output.		
		`LVQoutput`	The output object of which data is to be shown.
		`prototypes`	Determines whether or not the prototype endconfiguration should be shown.
		`relevances`	Determines whether or not the relevances endconfiguration should be shown.
		`costcurve`	Determines whether or not the progress of the costfunction should be plotted.
		`prototypeprogress`	Determines whether or not the progress of the prototypes should be shown.
		`relevanceprogress`	Determines whether or not the progressof the relevances should be shown.
		`trainerror`	Determines whether or not the trainerror should be shown.
		`testerror`	Determines whether or not the testerror should be shown.
		`trainerrorprogress`	Determines whether or not the progress of the trainerror should be shown.
		`testerrorprogress`	Determines whether or not the progress of the testerror should be shown.
		`relevancenumber`	When using `local` or `classwise` relevances determines which relevances should be shown.
		`relevanceprognumber`	When using `local` or `classwise` relevances determines of which relevances the progress should be shown.
`checkOutputVars`	This function checks for errors on the variables that determine the output.		
		`prototypeoutput`	Determines whether or not the prototype endconfiguration should be among the output.
		`relevanceoutput`	Determines whether or not the relevance endconfiguration should be among the output.
		`costcurve`	Determines whether or not progress of the costfunction should be calculated and returned among the output.
		`progress`	Determines whether or not the progress of the prototypes should be stored and returned among the output.
		`relevanceprogress`	Determines whether or not the progress of the relevances should be stored and returned among the output.
		`trainerror`	Determines whether or not the trained prototypes should be tested with the trainingset and if the result should be among the output.
		`testerror`	Determines whether or not the trained prototypes should be tested with the testset and if the result should be among the output.

Function	Description	Parameters	Description
		`trainerrorprogress`	Determines whether or not the trained prototypes should be tested with the trainingset after each epoch and if the result should be among the output.
		`testerrorprogress`	Determines whether or not the trained prototypes should be tested with the testset after each epoch and if the result should be among the output.
		`validatescheme`	Helper-variable. Determines if a testset should be present.
		`relevancemode`	Helper variable. Determines what kind of relevances will be used.
`checkValidateVars`	This function checks for errors on the variables used to determine what validation-methods will be used.		
		`validatescheme`	Determines what method of validation will be used.
		`nfold`	When using the `nfold`-validationscheme (nfoldcross-validation) determines in how many sets the data should be divided.
		`nrdatapoints`	Helper-variable. The size of the trainingset.

Cost

This module contains functions to calculate the cost of the current configuration of the prototypes. A version for normal LVQ1, the winner takes all principle, and a version for generalized LVQ are present. This module makes use of several functions that are defined in in the `distfunctions`-module.

Function	Description	Parameters	Description
`cost`	This function calculates the cost of the current configuration of the prototypes according to the winner takes all principle. The distance (acoording to the LVQ-scheme, relevance-mode and scheme) to the closest correct prototype is calculated and summed over all datapoints.		
		`LVQscheme`	The version of LVQ on which the distance measure depends.
		`data`	The trainingset (minus the labels) for which the cost will be calculated.
		`labels`	The labels of `data`.
		`prototypes`	The prototypes-configuration (minus the labels) with which the cost will be calculated.
		`protolabels`	The labels of `prototypes`.
		`distscheme`	When using `LVQ1-LVQscheme` determines how the difference between a prototype and a datapoint is calculated.
		`customdist`	When using `LVQ1-LVQscheme` determines how the difference between a prototype and a datapoint is calculated.
		`relevancemode`	Determines what kind of relevances are used.
		`relevancescheme`	Determines how many relevances are used.
		`relevances`	The relevances used to calculate the distance of a prototype to a datapoint.
		`alfa`	When using the `LVQscheme renyi` determines the version of renyi-divergenceto be used for the distance measure.

Function	Description	Parameters	Description
`generalCost`	This function calculates the cost of the current configuration of the prototypes according to generalized-LVQ. The closest correct and incorrect prototype to each the datapoint are calculated and the cost according to generalized LVQ is calculated.		
		`LVQscheme`	The version of LVQ on which the distance measure depends.
		`data`	The trainingset (minus the labels) for which the cost will be calculated.
		`labels`	The labels of `data`.
		`prototypes`	The prototypes-configuration (minus the labels) with which the cost will be calculated.
		`protolabels`	The labels of `prototypes`.
		`distscheme`	When using `LVQ1-LVQscheme` determines how the difference between a prototype and a datapoint is calculated.
		`customdist`	When using `LVQ1-LVQscheme` determines how the difference between a prototype and a datapoint is calculated.
		`relevancemode`	Determines what kind of relevances are used.
		`relevancescheme`	Determines how many relevances are used.
		`relevances`	The relevances used to calculate the distance of a prototype to a datapoint.
		`alfa`	When using the `LVQscheme renyi` determines the version of renyi-divergenceto be used for the distance measure.

Distfunctions

This module contains functions to calculate the distance of a prototype to a datapoint. Different functions are included for the different LVQ-versions, but also for during testing and for the different relevance-modes and -schemes. The different distance-functions are provided for efficiency-reasons. The aim is to calculate as many distances at the same time.

Function	Description	Parameters	Description
`allClasswiseDist`	This function calculates all distances based on the provided `difference` matrix using classwise relevances. It does this by selecting the appropriate function for the given `relevancemode`.		
		`difference`	This is a matrix containing the manhattan-distance of each prototype and dimension. Every line contains the difference of a prototype with at each collumn a different dimension.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme.
		`relevancemode`	Determines what kind of relevances are used.
		`relevances`	The relevances used for distance-calculation.
		`protolabels`	The labels of the prototypes. Entry `i` is the label of the difference-vector of `difference[i,]`.
`allLocalDist`	This function calculates all distances based on the provided `difference` matrix using local relevances. It does this by selecting the appropriate function for the given `relevancemode`.		
		`difference`	This is a matrix containing the manhattan-distance of each prototype and dimension. Every line contains the difference of a prototype with at each collumn a different dimension.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme.
		`relevancemode`	Determines what kind of relevances are used.
		`relevances`	The relevances used for distance-calculation.
`calcAllDist`	When using the `LVQ1`-LVQscheme calculates all distances based on the difference-matrix, `customdist`and relevances.		
		`difference`	This is a matrix containing the manhattan-distance of each prototype and dimension. Every line contains the difference of a prototype with at each collumn a different dimension.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme.
		`relevancemode`	Determines what kind of relevances are used
		`relevances`	The relevances used for distance-calculation.

Function	Description	Parameters	Description
`classwiseMatrixDist`	This function calculates all distances based on the provided `difference` matrix using classwise matrix-relevances.		
		`relevances`	The relevances used for distance-calculation.
		`difference`	This is a matrix containing the manhattan-distance of each prototype and dimension. Every line contains the difference of a prototype with at each collumn a different dimension.
		`protolabels`	The labels of the prototypes. Entry `i` is the label of the difference-vector of `difference[i,]`.
`classwiseRelevanceDist`	This function calculates all distances based on the provided `difference` matrix using classwise relevances.		
		`relevances`	The relevances used for distance-calculation.
		`difference`	This is a matrix containing the manhattan-distance of each prototype and dimension. Every line contains the difference of a prototype with at each collumn a different dimension.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme.
		`protolabels`	The labels of the prototypes. Entry `i` is the label of the difference-vector of `difference[i,]`.
`csDist`	This function calculates the distance between a datapoint and several prototypes according to Cauchy-Schwarz divergence as distance measure.		
		`protomatrix`	The prototypes which will be used for distance-calculation. (minus the labels)
		`datapoint`	The datapoint which will be used for distance-calculation. (minus the label)
`localMatrixDist`	This function calculates all distances based on the provided `difference` matrix using local matrix-relevances.		
		`relevances`	The relevances used for distance-calculation.
		`difference`	This is a matrix containing the manhattan-distance of each prototype and dimension. Every line contains the difference of a prototype with at each collumn a different dimension.
`localRelevanceDist`	This function calculates all distances based on the provided `difference` matrix using classwise relevances.		
		`relevances`	The relevances used for distance-calculation.
		`difference`	This is a matrix containing the manhattan-distance of each prototype and dimension. Every line contains the difference of a prototype with at each collumn a different dimension.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme.

Function	Description	Parameters	Description
`matrixDist`	Given the matrix Ω, the square root of the relevance matrix Λ, and a difference matrix the distance of a datapoint to several prototypes is calculated using a relevance-matrix.		
		`relmatrix`	This is Ω, the square root of the relevance-matrix Λ.
		`difference`	This is a matrix containing the manhattan-distance of several prototypes to the datapoint. Every line contains the manhattan-distance of a different prototype.
`metricDistWrap`	This function calculates the distance of a datapoint to several prototypes, when using the `LVQ1`-LVQscheme. It is a wrapper to distinguish between the different relevanceschemes.		
		`difference`	This is a matrix containing the manhattan-distance of each prototype and dimension. Every line contains the difference of a prototype with at each collumn a different dimension.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme.
		`relevancemode`	Determines what kind of relevances are used.
		`relevancescheme`	Determines how many relevances are used.
		`relevances`	The relevances used for distance-calculation.
		`protoclasses`	The labels of the prototypes. Entry `i` is the label of the difference-vector of `difference[i,]`.
`renyiDist`	This function calculates the distance of a datapoint to several prototypes according to the `renyi`-LVQscheme.		
		`protomatrix`	The prototypes which will be used for distance-calculation. (minus the labels)
		`datapoint`	The datapoint which will be used for distance-calculation. (minus the label)
		`alfa`	Determines the version of Renyi-divergence the distance-calculation.
`renyiTestDist`	This function calculates the distance of a prototype to several datapoints according to the `renyi`-LVQscheme.		
		`prototype`	The prototype which will be used for distance-calculation. (minus the labels)
		`data`	The datapoints which will be used for distance-calculation. (minus the label)
		`alfa`	Determines the version of Renyi-divergence the distance-calculation.

Function	Description	Parameters	Description
`testMetricDist`	This function calculates the distance of several datapoints to a prototype, when using the `LVQ1`-LVQscheme. It is a wrapper to distinguish between the different relevanceschemes.		
		`difference`	This is a matrix containing the manhattan-distance of each datapoint and dimension. Every line contains the difference of a datapoint to the prototype with at each collumn a different dimension.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme.
		`prototypenumber`	Determines which prototype is to be used in this calculation.
		`prototypeclass`	The class of the prototype.
		`relevancemode`	Determines what kind of relevances are used.
		`relevancescheme`	Determines how many relevances are used.
		`relevances`	The relevances used for distance-calculation.

Error

This module handles the errors. When an error is encountered by the `check`-module the associated errornumber is passed on to the `error`-module. This module than prints the associated errormessage and ends the computation.

Function	Description	Parameters	Description
`error`	This function takes a vector containing errornumbers, prints out the errormessages and the number of errors and stops the computation.		
		`errorvec`	The vector containing all the errornumbers.
`errormessage`	This function takes an errornumber and prints out the corresponding errormessage. It also contains all the errormessages hardcoded.		
		`errornum`	The errornumber.

Initialize

This module handles initialization of the prototypes, relevances and learningrates. It contains different methods of prototype-initialization and random initialization for relevances.

Function	Description	Parameters	Description
`classwiseInit`	This function initializes relevances when using classwise relevances. For each relevance-vector/matrix it calls `initializeRelVector` or `initializeRelMatrix` respectively.		
		`relevances`	The relevances which might or might not have already been initialized by the user.
		`dimensions`	The number of dimensions of the dataset.
		`relevancemode`	Determines the sort of relevances to be initialized.
		`classes`	The different classlabels present in the dataset.
`constructlabels`	This is a helper function. It constructs a vector of labels, which is to be put at the end of the matrix of prototypes.		
		`prototypes`	A vector containing the number of prototypes for each class.
`globalInit`	This function initializes relevances when using global relevances. It does this by selecting the appropriate function from `initializeRelVector` or `initializeRelMatrix`.		
		`relevances`	The relevances which might or might not have already been initialized by the user.
		`dimensions`	The number of dimensions of the dataset.
		`relevancemode`	Determines the sort of relevances to be initialized.
`initializePrototypes`	This function initializes prototypes.It does so by selecting the appropriate function according to the `initscheme`.		
		`initscheme`	Determines the way the prototypes are to be initialized.
		`prototypes`	A vector containing the number of prototypes for each class.
		`data`	The dataset to be used in training.
		`labels`	The labels of all the datapoints.
		`LVQscheme`	The version of LVQ to be used in training.

Function	Description	Parameters	Description
`initializeRelevances`	This function initializes relevances. It does so by selecting the appropriate function according to the `relevancescheme`.		
		`relevances`	The relevances which might or might not have already been initialized by the user.
		`dimensions`	The number of dimensions of the dataset.
		`relevancemode`	Determines the sort of relevances to be initialized.
		`relevancescheme`	Determines the number of relevances to be initialized.
		`classes`	The different classlabels present in the dataset.
		`nrofprototypes`	The total number of prototypes.
`initializeRelMatrix`	This function constructs and randomly initializes one relevance matrix.		
		`relevances`	The relevances which might or might not have already been initialized by the user.
		`dimensions`	The number of dimensions of the dataset.
`initializeRelVector`	This function constructs and randomly initializes one relevance vector.		
		`relevances`	The relevances which might or might not have already been initialized by the user.
		`dimensions`	The number of dimensions of the dataset.
`initLearningrate`	This function initializes the learningrate-vector if neccesary. If the learningrate is a single value, it is made into a vector of length `epochs`, with at each entry the given value.		
		`learningrate`	The rate at which the prototypes will adapt. this might or might not already have been initialized.
		`epochs`	The number of epochs for the training and the eventual length of the learningrate-vector.
`initRelrate`	This function initializes the relevance learningrate- vector if neccesary. If the learningrate is a single value, it is made into a vector of length `epochs`, with at each entry the given value.		
		`relrate`	The rate at which the relevances will adapt. this might or might not already have been initialized.
		`epochs`	The number of epochs for the training and the eventual length of the relrate-vector.

Function	Description	Parameters	Description
`localInit`	This function initializes relevances when using local relevances. For each relevance-vector/matrix it calls `initializeRelVector` or `initializeRelMatrix` respectively.		
		`relevances`	The relevances which might or might not have already been initialized
		`dimensions`	by the user. The number of dimensions of the dataset.
		`relevancemode`	Determines the sort of relevances to be initialized.
		`nrofprototypes`	The total number of prototypes.
`meanClasswiseInit`	This function is used for prototype-initialization. It initializes each prototype at the mean of its corresponding class.		
		`prototypes`	A vector containing the number of prototypes for each class.
		`data`	The dataset to be used in training.
		`labels`	The labels of all the datapoints.
`meanInit`	This function is used for prototype-initialization. It initializes all prototype at the mean of the dataset.		
		`prototypes`	A vector containing the number of prototypes for each class.
		`data`	The dataset to be used in training.
`minmax`	This is a helper function. It produces a matrix of two lines. The first line contains the minimum of each dimension of the dataset, the second the line all the maxima.		
		`data`	The dataset to be used in training.
`randomSampleInit`	This function is used for prototype-initialization. It initializes each prototype at the same location as a randomly determined sample of the dataset.		
		`prototypes`	A vector containing the number of prototypes for each class.
		`data`	The dataset to be used in training.
`randomWindowInit`	This function is used for prototype-initialization. It initializes each prototype at a random location within the range of the dataset.		
		`prototypes`	A vector containing the number of prototypes for each class.
		`data`	The dataset to be used in training.
		`LVQscheme`	The version of LVQ to be used in training.
`zeroInit`	This function is used for prototype-initialization. It initializes all prototypes by setting all values to zero.		
		`prototypes`	A vector containing the number of prototypes for each class.
		`data`	The dataset to be used in training.
		`labels`	The labels of all the datapoints.

Input

This module contains function to read data from file, to normalize data and to replace missing values. Data can be normalized according to z-transform or IQR. The data can also be normalized so that each datavector sums up to one. Missing values can be replaced by the mean of the dataset or by the mean of the corresponding class. Input files should exactly one datapoint per line. Each datapoint should list some values seperated by a whitespace and end with a classlabel. Missing values should be indicated by `NA`.

Function	Description	Parameters	Description
`input`	This is the main function of the input-module. It reads datafrom file if applicable and applies normalization and missing value replacement if applicable.		
		`datapath`	This is the location of the input-file.
		`normalizescheme`	This determines if and how the data should be normalized.
		`normalclasswise`	This determines if normalization should be conducted with respect to a certain class and which.
		`replaceNA`	This determines whether or not missing values should be replaced.
		`replaceclasswise`	This determines whether or not replacement of missing values should be conducted per class.
		`input`	Input data provided by the user if the user has provided data.
`iqrnorm`	This function performs normalization with respect to the Inter Quantile Range.		
		`data`	The data to be normalized.
		`labels`	The labels of the data.
		`normalclass`	The class on which the normalization should be based. If classwise normalization should not be performed this parameter should be `none`.
`normalize`	This is a wrapper-funtion that decides between the different normalization -schemes and -functions.		
		`data`	The data to be normalized.
		`labels`	The labels of the data.
		`normalizescheme`	This determines if and how the data should be normalized.
		`classwise`	This determines if normalization should be conducted with respect to a certain class and which.
		`ordorlabs`	Helper-parameter. The available classlabels, sorted lexicographically.
`replaceNA`	This function facilitates the replacement of missing values.		
		`data`	The data in which missing values might need replacing.

Function	Description	Parameters	Description
`replaceNAwrap`	This is a wrapper-function that decides between classwise replacement of missing values and regeular replacement.		
		`data`	The data in which missing values might need replacing.
		`classwise`	This determines whether or not replacement of missing values should be conducted per class.
		`classlabels`	The labels of the data.
`sumonenorm`	This function performs normalization by enforcing every datapoint to sum up to 1.		
		`data`	The data to be normalized.
`ztransform`	This function performs normalization according to z-transform.		
		`data`	The data to be normalized.
		`labels`	The labels of the data.
		`normalclass`	The class on which the normalization should be based. If classwise normalization should not be performed this parameter should be `none`.
`transformlabels`	This function transforms the classlabels to numbers starting at 1.		
		`classlabels`	The labels to be trandformed.

LVQ

This module contains one function: `LVQ`. It performs Learning Vector Quantization according to the given parameters.

Function	Description	Parameters	Description
`LVQ`	This function performs Learning Vector Quantization according to the given parameters.		
		`data`	The data on which training will be performed, with the classlabels, in numerical form, attached.
		`originallabels`	The labels of the dataset in their original character form.
		`testdata`	The dataset on which intermediate tests, with the prototypes, can ben performed
		`prototypes`	A vector, which is indexed by the classlabels, containing the number of prototypes per class.
		`learningrate`	The rate at which the prototypes will adapt. It contains either a number, between 0 and 1 or a vector of such numbers of length `epochs`.
		`epochs`	The number of passes to be made through the trainingset.
		`initscheme`	Determines the way the prototypes are initialized.
		`distscheme`	Determines what kind of measure is used to determine the distance from prototype to datapoint, when using the `LVQ1`-scheme.
		`relevancemode`	Determines the sort of relevances used.
		`relevancescheme`	Determines the number of relevances used.
		`LVQscheme`	The version of LVQ to be used.
		`optimisationscheme`	Determines how the optimal situation will be reached and thus how the prototype and relevances will be updated.
		`relevances`	The relevances provided by the user. `NA` when none are provided.
		`relrate`	The rate at which the relevances will adapt. It contains either a number, between 0 and 1 or a vector of such numbers of length `epochs`.
		`customdist`	When using `distschemecustom` determines how the distance is calculated.
		`alfa`	When using `LVQscheme renyi` determines the version of Renyi-divergence to be used for calculating the distance.

Function	Description	Parameters	Description
		`show`	Determines whether or not progress should be shown during the training.
		`graphics`	Determines whether or not the trainingset and prototypes should be plotted during training. This is only available if the trainingset is 2-dimensional.
		`costfunction`	Determines if the progress of the costfunction should be calculated and stored after each epoch.
		`plotcurve`	Determines whether or not the progress of the constfunction should be plotted after each LVQ-run.
		`progress`	Determines whether or not the progress of the prototypes should be stored and returned among the output.
		`relevanceprogress`	Determines whether or not the progress of the relevances should be stored and returned among the output.
		`trainerrorprogress`	Determines whether or not the trained prototypes should be tested with the trainingset after each epoch and if the result should be among the output.
		`testerrorprogress`	Determines whether or not the trained prototypes should be tested with the testset after each epoch and if the result should be among the output.

Output

This module contains functions to show the results of LVQ-trainings and -tests. In addition functions that return a certain output-value are provided. It makes use of a couple of R-classes which will also be discussed here.

Class	Description	Attribute	Description
`input`	This class is used in reading input. It makes it possible to return a workable matrix, with datapoints with their classlabels in numerical form, and a vector with the original labels.		
		`data`	The dataset that had been read or provided by the user, with classlabels in numerical form.
		`labels`	The labels of the dataset in their original form.
`LVQoutput`	This class is used in the LVQ-training and validation-schemes. It makes it possible to return all the specified output along with a few helper-variables.		
		`prototypes`	The end-configuration of the prototypes.
		`protolabels`	The labels of the prototypes in numerical form.
		`relevances`	The end-configuration of the relevances.
		`costcurve`	The progress of the costfunction.
		`protoprogress`	All the configurations of the prototypes from initialization to the end-configuration.
		`relevanceprogress`	All the configurations of the relevances from initialization to the end-configuration.
		`trainerror`	The number of errors made when classifying the trainingset using the end-configuration of the prototypes.
		`testerror`	The number of errors made when classifying the testset using the end-configuration of the prototypes.
		`trainerrorprogress`	A vector containg the numer of errors when classifying the trainingset using the prototype-configuration after every epoch.
		`testerrorprogress`	A vector containg the numer of errors when classifying the testset using the prototype-configuration after every epoch.
		`originallabels`	The labels of the trainingset in their original form.
		`nrofrelevances`	The number of relevances used in training.

Class	Description	Attribute	Description
`nfoldoutput`	This class contains all the output of a `validate`-run according to nfoldcrossvalidation.		
		`prototypes`	The end-configuration of the prototypes.
		`costcurve`	The progress of the costfunction.
		`protoprogress`	All the configurations of the prototypes from initialization to the end-configuration.
		`relevanceprogress`	All the configurations of the relevances from initialization to the end-configuration.
		`trainerror`	The number of errors made when classifying the trainingset using the end-configuration of the prototypes.
		`testerror`	The number of errors made when classifying the testset using the end-configuration of the prototypes.
		`trainerrorprogress`	A vector containg the number of errors when classifying the trainingset using the prototype-configuration after every epoch.
		`testerrorprogress`	A vector containg the number of errors when classifying the testset using the prototype-configuration after every epoch.
		`nfold`	The number of subsets created for this `validate`-run.
		`nrofrelevances`	The number of relevances used in training.
`trainoutput`	This class contains all the output of a `validate`-run according to the `train`-scheme.		
		`prototypes`	The end-configuration of the prototypes.
		`relevances`	The end-configuration of the relevances.
		`costcurve`	The progress of the costfunction.
		`protoprogress`	All the configurations of the prototypes from initialization to the end-configuration.
		`relevanceprogress`	All the configurations of the relevances from initialization to the end-configuration.
		`trainerror`	The number of errors made when classifying the trainingset using the end-configuration of the prototypes.
		`trainerrorprogress`	A vector containg the number of errors when classifying the trainingset using the prototype-configuration after every epoch.
		`nrofrelevances`	The number of relevances used in training.

Class	Description	Attribute	Description
`traintestoutput`	This class contains all the output of a `validate`-run according to the `traintest`-scheme.		
		`prototypes`	The end-configuration of the prototypes.
		`relevances`	The end-configuration of the relevances.
		`costcurve`	The progress of the costfunction.
		`protoprogress`	All the configurations of the prototypes from initialization to the end-configuration.
		`relevanceprogress`	All the configurations of the relevances from initialization to the end-configuration.
		`trainerror`	The number of errors made when classifying the trainingset using the end-configuration of the prototypes.
		`testerror`	The number of errors made when classifying the testset using the end-configuration of the prototypes.
		`trainerrorprogress`	A vector containg the number of errors when classifying the trainingset using the prototype-configuration after every epoch.
		`testerrorprogress`	A vector containg the number of errors when classifying the testset using the prototype-configuration after every epoch.
		`nrofrelevances`	The number of relevances used in training.

Function	Description	Parameters	Description
`getCostcurve`	This function returns the progress of the costfunction.		
		`LVQout`	The output-class containing the output to be returned.
		`fold`	Determines from which subset the output is to be returned.
`getPrototypeProgress`	This function returns the progress of the prototypes.		
		`LVQout`	The output-class containing the output to be returned.
		`fold`	Determines from which subset the output is to be returned.
`getPrototypes`	This function returns the end-configuration of the prototypes.		
		`LVQout`	The output-class containing the output to be returned.
		`fold`	Determines from which subset the output is to be returned.
`getRelevanceProgress`	This function returns the progress of the relevances.		
		`LVQout`	The output-class containing the output to be returned.
		`fold`	Determines from which subset the output is to be returned.
`getRelevances`	This function returns the end-configuration of the relevances.		
		`LVQout`	The output-class containing the output to be returned.
		`fold`	Determines from which subset the output is to be returned.
`getTestError`	This function returns the number of missclassifications that were encountered when classifying the testset with the prototype end-configuration.		
		`LVQout`	The output-class containing the output to be returned.
		`fold`	Determines from which subset the output is to be returned.
`getTestErrorProgress`	This function returns the number of missclassifications that were encountered when classifying the testset with all the configurations of the prototypes.		
		`LVQout`	The output-class containing the output to be returned.
		`fold`	Determines from which subset the output is to be returned.
`getTrainError`	This function returns the number of missclassifications that were encountered when classifying the trainingset with the prototype end-configuration.		
		`LVQout`	The output-class containing the output to be returned.
		`fold`	Determines from which subset the output is to be returned.
`getTrainErrorProgress`	This function returns the number of missclassifications that were encountered when classifying the trainingset with all the configurations of the prototypes.		
		`LVQout`	The output-class containing the output to be returned.
		`fold`	Determines from which subset the output is to be returned.

Function	Description	Parameters	Description
`nfoldcrossoutput`	This function constructs an `nfoldoutput`-object with the available output.		
		`LVQlist`	A list of `LVQoutput`-classes, which are to be transformed into a `nfoldcrosspoutput`-class.
		`prototypeoutput`	This determines if the end-configuration of the prototypes should be among the output.
		`relevanceoutput`	This determines if the end-configuration of the relevances should be among the output.
		`costfunction`	This determines if the progress of the costfunction should be among the output.
		`progress`	This determines if all the configurations of the prototypes should be among the output.
		`relevanceprogress`	This determines if all the configurations of the relevances should be among the output.
		`trainerror`	This determines if the number of missclassifications when testing the end-configuration of the prototypes with the trainingset should be among the output.
		`testerror`	This determines if the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
		`trainerrorprogress`	This determines if the progress of the number of missclassifications when testing the end-configuration of the prototypes with the trainingset should be among the output.
		`testerrorprogress`	This determines if the progress of the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
`orderRelevances`	This is a helper-function. It transforms the `relevanceprogress` from a progress-list of different relevances to a list of different relevanceprogresses.		
		`relevancelist`	A list of lists containing all the relevance-configurations. The first entry contains the initialized relevances, in a list if there are more than one set of relevances, the second entry the configuration after the first epoch, and so on.

Function	Description	Parameters	Description
`output`	This function constructs the appropriate output-class according to the `validatescheme` by selecting the right function.		
		`LVQresult`	The class containing all the output of this `validate`-run. It can be a `LVQoutput` -class or a list of such classes.
		`validatescheme`	The scheme used for this `validate`-run.
		`prototypeoutput`	This determines if the end-configuration of the prototypes should be among the output.
		`relevanceoutput`	This determines if the end-configuration of the relevances should be among the output.
		`costfunction`	This determines if the progress of the costfunction should be among the output.
		`progress`	This determines if all the configurations of the prototypes should be among the output.
		`relevanceprogress`	This determines if all the configurations of the relevances should be among the output.
		`trainerror`	This determines if the number of missclassifications when testing the end-configuration of the prototypes with the trainingset should be among the output.
		`testerror`	This determines if the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
		`trainerrorprogress`	This determines if the progress of the number of missclassifications when testing the end-configuration of the prototypes with the trainingset should be among the output.
		`testerrorprogress`	This determines if the progress of the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
`show`	This function shows all the selected output. The selection is made by providing the appropriate parameters with `TRUE`.		
		`LVQoutput`	The output-class containing all the output.
		`prototypes`	This determines if the end-configuration of the prototypes should be among the output.
		`relevances`	This determines if the end-configuration of the relevances should be among the output.
		`costcurve`	This determines if the progress of the costfunction should be among the output.
		`prototypeprogress`	This determines if all the configurations of the prototypes should be among the output.

Function	Description	Parameters	Description
		`relevanceprogress`	This determines if all the configurations of the relevances should be among the output.
		`trainerror`	This determines if the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
		`testerror`	This determines if the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
		`trainerrorprogress`	This determines if the progress of the number of missclassifications when testing the end-configuration of the prototypes with the trainingset should be among the output.
		`testerrorprogress`	This determines if the progress of the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
		`protofold`	Selects which prototype-end-configuration should be shown. `-1` selects all.
		`relfold`	Selects which relevance-end-configuration should be shown. `-1` selects all.
		`costfold`	Selects which costfunction-progress should be shown. `-1` selects all.
		`protoprogfold`	Selects which prototype-progress should be shown. `-1` selects all.
		`relprogfold`	Selects which prototype-progress should be shown. `-1` selects all.
		`trainerrorprogfold`	Selects which trainerror-progress should be shown. `-1` selects all.
		`testerrorprogfold`	Selects which testerror-progress should be shown. `-1` selects all.
		`relevancenumber`	Selects which relevances should be shown in the case of classwise or local relevances. `-1` selects all.
		`relevanceprognumber`	Selects of which relevances the progress should be shown in the case of classwise or local relevances. `-1` selects all.
`showAll`	This function shows all the available ouptut, by calling `show` with all the appropriate parameters to `TRUE`.		
		`LVQoutput`	The output-class containing all the output.
`showCostcurve`	This function plots the costfunction-progress.		
		`costcurve`	The progress of the costfunction to be shown.
		`fold`	Shows which progress is shown.

Function	Description	Parameters	Description
`showCostcurveNfold`	This function plots the selected costfunction-progresses by calling `showCostcurve` once or more.		
		`costlist`	A list of vectors with costfunction-progress.
		`costfold`	Shows which progress is shown.
`showEndPrototypes`	This function prints the end-configuration of the prototypes to the screen.		
		`prototypes`	The prototype-configuration to be shown.
		`fold`	Shows which prototypes are shown.
`showEndPrototypesNfold`	This function prints the selected end-configuration of the prototypes to the screen, by calling `showEndPrototypes` once or more.		
		`prototypelist`	A list of prototype-configurations.
		`fold`	Shows which prototypes are shown.
`showPrototypeProgress`	This function prints all the prototype-configuration after every epoch to the screen.		
		`protolist`	A list of prototype-configurations.
		`fold`	Shows which prototypes-progresses are shown.
`showPrototypeProgressNfold`	This function prints all the prototype-configuration after every epoch of the selected data-subset to the screen, by calling `showPrototypeProgress` once or more.		
		`protoproglist`	A list of prototype-progresses.
		`fold`	Shows which prototypes-progresses are shown.
`showRelevanceProgress`	This function plots all the relevance-configurations after every epoch (in order) in a barplot (for vectors) or a greyscale picture (for matrices).		
		`rellist`	A list containing relevance-configurations.
		`fold`	Shows what relevance-progress are shown.
		`relevancenumber`	Shows what relevance-progress are shown.

Function	Description	Parameters	Description
`showRelevanceProgressNfold`	This function plots all the relevance-configurations after every epoch (in order), of the selected datasubset, in a barplot, by calling `showRelevanceProgressWrap` once or more.		
		`relproglist`	A list containing relevance-progresses.
		`relprogfold`	Determines which relevance-progress is to be shown.
		`relevancenumber`	Determines which relevance-progress is to be shown.
`showRelevanceProgressWrap`	This function plots all the relevance-configurations after every epoch (in order), of the selected relevances, in the case of more than one set of relevances, by calling `showRelevanceProgress` once or more.		
		`relevances`	The set of relevances, or set of relevance-sets to be plotted.
		`fold`	Shows which relevance-progress is shown.
		`relevanceprognumber`	Determines which relevance-progress is to be shown.
`showRelevances`	This function plots the end-configuration of the relevances in a barplot (for vectors) or a greyscale image (for matrices).		
		`relevances`	The set of relevances which will be plotted.
		`fold`	Shows which relevance-progress is shown.
		`relevancenumber`	Shows which relevance-progress is shown.
`showRelevancesNfold`	This function plots the selected relevance end-configuration.		
		`rellist`	The list of relevance-sets.
		`relfold`	Determines which relevances is to be shown.
		`relevancenumber`	Determines which relevances is to be shown.
`showRelevancesWrap`	This function plots the selected relevances in the case of more than one set of relevances, by calling `showRelevances` once or more.		
		`relevances`	The set of relevances, or set of relevance-sets to be plotted.
		`fold`	Shows which relevance-progress is shown.
		`relevancenumber`	Determines which relevances is to be shown.

Function	Description	Parameters	Description
`showTestError`	This function prints the number of missclassifications encountered when the testset was classified with the end-configuration of the prototypes.		
		`testerror`	The number of missclasifications.
		`fold`	Shows of which subset the missclassifications originate.
`showTestErrorNfold`	This function plots all the numbers of missclassifications encountered when the testset was classified with the end-configuration of the prototypes.		
		`testerror`	A vector containing the missclassifications of all subsets.
`showTesterrorProgress`	This function plots the progress of missclassifications encountered when the testset was classified with the all the configuration of the prototypes.		
		`testerrors`	A vector containing the missclassifications when testing the testset with every configuration of the prototypes.
		`fold`	Shows of which subset the missclassifications originate.
`showTesterrorProgressNfold`	This function plots the selected progress of missclassifications encountered when the testset was classified with the end-configuration of the prototypes, by calling `showTestErrorProgress` once or more.		
		`errorlist`	A list of vectors containing the missclassifications when testing the testset with every configuration of the prototypes.
		`testfold`	Determines which subset of missclassifications will be shown.
`showTrainError`	This function plots all the numbers of missclassifications encountered when the trainingset was classified with the end-configuration of the prototypes.		
		`trainerror`	The number of missclasifications.
		`fold`	Shows of which subset the missclassifications originate.
`showTrainErrorNfold`	This function prints the selected number of missclassifications encountered when the trainingingset was classified with the end-configuration of the prototypes, by calling `showTrainError` once or more.		
		`trainerror`	A vector containing the missclassifications of all subsets.

Function	Description	Parameters	Description
`showTrainerrorProgress`	This function plots the progress of missclassifications encountered when the trainingset was classified with the all the configuration of the prototypes.		
		`trainerrors`	A vector containing the missclassifications when testing the trainingset with every configuration of the prototypes.
		`fold`	Shows of which subset the missclassifications originate.
`showTrainerrorProgressNfold`	This function plots the selected progress of missclassifications encountered when the trainingset was classified with the end-configuration of the prototypes, by calling `showTrainErrorProgress` once or more.		
		`errorlist`	A list of vectors containing the missclassifications when testing the trainingset with every configuration of the prototypes.
		`trainfold`	Determines which subset of missclassifications will be shown.

Function	Description	Parameters	Description
`trainoutput`	This function constructs an `trainoutput` -object with the available output.		
		`LVQresult`	The `LVQoutput`-class, which is to be transformed into a `trainoutput`-class.
		`prototypes`	The end-configuration of the prototypes.
		`costcurve`	The progress of the costfunction.
		`protoprogress`	All the configurations of the prototypes from initialization to the end-configuration.
		`relevanceprogress`	All the configurations of the relevances from initialization to the end-configuration.
		`trainerror`	The number of errors made when classifying the trainingset using the end-configuration of the prototypes.
		`testerror`	The number of errors made when classifying the testset using the end-configuration of the prototypes.
		`trainerrorprogress`	A vector containg the number of errors when classifying the trainingset using the prototype-configuration after every epoch.
		`testerrorprogress`	A vector containg the number of errors when classifying the testset using the prototype-configuration after every epoch.
`traintestoutput`	This function constructs an `traintestoutput`-object with the available output.		
		`LVQresult`	The `LVQoutput`-class, which is to be transformed into a `traintestoutput`-class.
		`prototypes`	The end-configuration of the prototypes.
		`costcurve`	The progress of the costfunction.
		`protoprogress`	All the configurations of the prototypes from initialization to the end-configuration.
		`relevanceprogress`	All the configurations of the relevances from initialization to the end-configuration.
		`trainerror`	The number of errors made when classifying the trainingset using the end-configuration of the prototypes.
		`testerror`	The number of errors made when classifying the testset using the end-configuration of the prototypes.
		`trainerrorprogress`	A vector containg the number of errors when classifying the trainingset using the prototype-configuration after every epoch.
		`testerrorprogress`	A vector containg the number of errors when classifying the testset using the prototype-configuration after every epoch.

Show

This module contains functions to track progress of the training while it is still running.

Function	Description	Parameters	Description
`attachLabels`	Helper function. Attaches the labels in `character` form to the prototypes so they can be printed to the screen.		
		`protomatrix`	The prototypes without their classlabels.
		`protolabels`	The labels of the prototypes in `numeric` form.
		`originallabels`	The labels of dataset in `character` form.
`plotData`	This function plots the dataset and prototypes in a graphics window.		
		`sorteddata`	The dataset sorted by class. It is a list with at every entry a matrix with datapoints of one class without classlabel.
		`prototypes`	The prototypes without their classlabels.
		`protolabels`	The labels of the prototypes in `numeric` form.
		`data`	The trainingset unsorted without classlabels.
		`labels`	The labels of the dataset.
`showEndResults`	This function, after a training, prints the endresult of the prototypes and relevances to the screen and plots the dataset and prototypes in a graphics window, if any are appropriate.		
		`protomatrix`	The end-configuration of the prototypes.
		`protolabels`	The labels of the prototypes.
		`originallabels`	The labels of dataset in `character` form.
		`relevances`	The end-configuration of the relevances.
		`costcurve`	The progress of the costfunction.
		`epochs`	The number of epochs this training contained.
		`plotcurve`	Determines if the progress of the costfunction should be shown.
		`show`	Determines if the results will be printed to the screen.
		`relevancemode`	Shows what kind of relevances are used.
		`relevancescheme`	Shows how many sets of relevances are used.

Function	Description	Parameters	Description
`showResults`	This function prints the current configuration of the prototypes and relevances to the screen and plots the dataset and prototypes in a graphics window, if any are appropriate.		
		`protomatrix`	The current configuration of the prototypes.
		`protolabels`	The labels of the prototypes.
		`epoch`	The current epoch.
		`sorteddata`	The dataset sorted by class. It is a list with at every entry a matrix with datapoints of one class without classlabel.
		`data`	The trainingset unsorted without classlabels.
		`labels`	The labels of the dataset.
		`originallabels`	The labels of dataset in `character` form.
		`costcurve`	The current progress of the costfunction.
		`relevances`	The current configuration of the relevances.
		`dimensions`	The number of values of one datapoint.
		`graphics`	Determines if the dataset and prototypes should be plotted in a graphics window.
		`show`	Determines if the current prototype- and relevance-configuration will be printed to the screen.
		`relevancemode`	Shows what kind of relevances are used.
		`relevancescheme`	Shows how many sets of relevances are used.
		`costfunction`	Determines if the progress of the costfunction should be printed to the screen.
`sortData`	This function sorts the datapoints by class, so they can be easily plotted.		
		`data`	The trainingset without classlabel attached.
		`labels`	The labels of the dataset.

Update

This module facilitates updating of prototypes and relevances. For each `LVQscheme`, `optimisationscheme`, `relevancemode`, and `relevancescheme` functions are present to divide the work-flow and update with the appropriate method.

Function	Description	Parameters	Description
`csGeneralUpdate`	This function facilitates prototype-updating when using `cauchyschwarz` LVQscheme along with `general` `optimisationscheme`.		
		`protomatrix`	The prototypes which will be updated minus their classlabel.
		`winclass`	The index of the prototype closest to the datapoint of the same classl as the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`datapoint`	The datapoint presented for this update, without its classlabel.
		`dist`	A vector containing the distances of all prototype to the datapoint.
		`learningrate`	The rate at which the prototype will adapt.
`csNormalUpdate`	This function facilitates prototype-updating when using `cauchyschwarz` LVQscheme along with `normal` `optimisationscheme`.		
		`protomatrix`	The prototypes which will be updated minus their classlabel.
		`protolabels`	The labels of the prototypes.
		`winner`	The index of the prototype closest to the datapoint.
		`datapoint`	The datapoint presented for this update, without its classlabel.
		`dataclass`	The classlabel of the datapoint.
		`learningrate`	The rate at which the prototype will adapt.
`csUpdate`	Wrapper function to distinguish between `cauchyschwarz` LVQscheme with `normal` or `general` `optimisationscheme`.		
		`optimisationscheme`	Determines how the optimal situation will be reached and thus how the prototype and relevances will be updated.
		`protomatrix`	The prototypes which will be updated minus their classlabel.
		`protolabels`	The labels of the prototypes.
		`winner`	The index of the prototype closest to the datapoint.
		`winclass`	The index of the prototype closest to the datapoint of the same classl as the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.

Function	Description	Parameters	Description
		`datapoint`	The datapoint presented for this update, without its classlabel.
		`dataclass`	The classlabel of the datapoint.
		`learningrate`	The rate at which the prototype will adapt.
		`dist`	A vector containing the distances of all prototype to the datapoint.
`entropyNormalize`	This function normalizes the prototypes so that they sum up to 1.		
		`prototype`	A prototype, without classlabel, that needs to be normalized
`generalUpdate`	This function facilitates prototype-updating when using `LVQ1` LVQscheme along with `general optimisationscheme`.		
		`protomatrix`	The prototypes which will be updated minus their classlabel.
		`protolabels`	The labels of the prototypes.
		`winclass`	The index of the prototype closest to the datapoint of the same classl as the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`diffclass`	The manhattan distance of the closest prototype of the same class to the datapoint.
		`diffnotclass`	The manhattan distance of the closest prototype not of the same class to the datapoint.
		`distclass`	The distance of the closest prototype of the same class to the datapoint.
		`distnotclass`	The distance of the closest prototype not of the same class to the datapoint.
		`learningrate`	The rate at which the prototype will adapt.
		`customdist`	Determines the distance-measure when using `LVQ1` -LVQscheme and thus also prototype-updates.
		`classrelevances`	The set of relevances belonging to the closest prototype of the same class to the datapoint
		`notclassrelevances`	The set of relevances belonging to the closest prototype not of the same class to the datapoint
		`relevancemode`	Shows what kind of relevances are used.

Function	Description	Parameters	Description
`LVQ1Update`	Wrapper function to distinguish between `normal` and `general` `optimisationscheme` when using `LVQ1` LVQscheme.		
		`optimisationscheme`	Determines how the optimal situation will be reached and thus how the prototype and relevances will be updated.
		`protomatrix`	The prototypes which will be updated minus their classlabel.
		`protolabels`	The labels of the prototypes.
		`winner`	The index of the prototype closest to the datapoint.
		`winclass`	The index of the prototype closest to the datapoint of the same class as the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`dataclass`	The classlabel of the datapoint.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`learningrate`	The rate at which the prototype will adapt.
		`dist`	A vector containing the distances of all prototype to the datapoint.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype-updates.
		`relevances`	The relevances which are used in the updating process. This might be list of relevance-sets or a single vector or matrix.
		`relevancemode`	Shows what kind of relevances are used.
		`relevancescheme`	Shows how many relevance-sets are used.
`normalizeMatrix`	This function normalizes matrix-relevances so that $\Sigma_i \Lambda_{ii} = \Sigma_{mn} \Omega_{mn}^2 = 1$ holds. Where Λ is the relevance-matrix and Ω the square root of Λ.		
		`matrix`	A relevance-matrix that needs to be normalized.
`renyiGeneralUpdate`	This function facilitates prototype-updating when using `renyi` LVQscheme along with `general` `optimisationscheme`.		
		`protomatrix`	The prototypes which will be updated minus their classlabel.
		`winclass`	The index of the prototype closest to the datapoint of the same classes the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`datapoint`	The datapoint presented for this update, without its classlabel.
		`dist`	A vector containing the distances of all prototype to the datapoint.
		`learningrate`	The rate at which the prototype will adapt.
		`alfa`	Determines what version of Renyi-divergence will be used.

Function	Description	Parameters	Description
`renyiNormalUpdate`	This function facilitates prototype-updating when using `renyi` LVQscheme along with `normal` `optimisationscheme`.		
		`protomatrix`	The prototypes which will be updated minus their classlabel.
		`protolabels`	The labels of the prototypes.
		`winner`	The index of the prototype closest to the datapoint.
		`datapoint`	The datapoint presented for this update, without its classlabel.
		`dataclass`	The classlabel of the datapoint.
		`learningrate`	The rate at which the prototype will adapt.
		`alfa`	Determines what version of Renyi-divergence will be used.
`renyiUpdate`	Wrapper function to distinguish between `renyi` LVQscheme with `normal` or `general` `optimisationscheme`.		
		`optimisationscheme`	Determines how the optimal situation will be reached and thus how the prototype and relevances will be updated.
		`protomatrix`	The prototypes which will be updated minus their classlabel.
		`protolabels`	The labels of the prototypes.
		`winner`	The index of the prototype closest to the datapoint.
		`winclass`	The index of the prototype closest to the datapoint of the same class as the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`datapoint`	The datapoint presented for this update, without its classlabel.
		`dataclass`	The classlabel of the datapoint.
		`learningrate`	The rate at which the prototype will adapt.
		`dist`	A vector containing the distances of all prototype to the datapoint.
		`alfa`	Determines what version of Renyi-divergence will be used.
`update`	This function facilitates prototype-updating when using `LVQ1` LVQscheme along with `normal` `optimisationscheme`.		
		`protomatrix`	The prototypes which will be updated minus their classlabel.
		`protolabels`	The labels of the prototypes.
		`winnerindex`	The index of the prototype closest to the datapoint.
		`dataclass`	The classlabel of the datapoint.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`learningrate`	The rate at which the prototype will adapt.

Function	Description	Parameters	Description
`updateGeneralClassMatrix`	This function updates classwise matrix-relevances according to `LVQ1` and `general` optimisationscheme.		
		`relevances`	A list with relevances-matrices of which some will be updated.
		`protolabels`	The labels of the prototypes.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`winclass`	The index of the prototype closest to the datapoint of the same classes the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`relrate`	The rate at which the relevances will adapt.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.
`updateGeneralClassRelevances`	This function updates classwise vector-relevances according to `LVQ1` and `normal` optimisationscheme.		
		`relevances`	A list with sets of relevances of which some will be updated.
		`protolabels`	The labels of the prototypes.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`winclass`	The index of the prototype closest to the datapoint of the same classes the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`relrate`	The rate at which the relevances will adapt.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.

Function	Description	Parameters	Description
`updateGeneralClassWrap`	Wrapper function to update classwise relevances and distinguish between the `normal` and `general optimisationscheme`.		
		`protolabels`	The labels of the prototypes.
		`winclass`	The index of the prototype closest to the datapoint of the same classes the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`relevances`	A list with sets of relevances of which some will be updated.
		`dataclass`	The classlabel of the datapoint.
		`relrate`	The rate at which the relevances will adapt.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`relevancemode`	Shows what kind of relevances are used.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.
`updateGeneralGlobalWrap`	Wrapper function to update global relevances and distinguish between the `normal` and `general optimisationscheme`.		
		`protolabels`	The labels of the prototypes.
		`winclass`	The index of the prototype closest to the datapoint of the same classes the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`relevances`	The set of relevances that will be updated.
		`dataclass`	The classlabel of the datapoint.
		`relrate`	The rate at which the relevances will adapt.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`relevancemode`	Shows what kind of relevances are used.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.

Function	Description	Parameters	Description
`updateGeneralLocalMatrix`	This function updates local matrix-relevances according to `LVQ1` and `general` optimisationscheme.		
		`relevances`	A list with relevance-matrices of which some will be updated.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`winclass`	The index of the prototype closest to the datapoint of the same classes the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`relrate`	The rate at which the relevances will adapt.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.
`updateGeneralLocalRelevances`	This function updates local vector-relevances according to `LVQ1` and `general` optimisationscheme.		
		`relevances`	A list with sets of relevances of which some will be updated.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`winclass`	The index of the prototype closest to the datapoint of the same classes the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`relrate`	The rate at which the relevances will adapt.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.

Function	Description	Parameters	Description
`updateGeneralLocalWrap`	Wrapper function to update local relevances and distinguish between the `normal` and `general` `optimisationscheme`.		
		`protolabels`	The labels of the prototypes.
		`winclass`	The index of the prototype closest to the datapoint of the same classes the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`relevances`	A list with sets of relevances of which some will be updated.
		`dataclass`	The classlabel of the datapoint.
		`relrate`	The rate at which the relevances will adapt.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`relevancemode`	Shows what kind of relevances are used.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.
`updateMatrix`	This function updates global matrix-relevances according to `LVQ1` and `normal` optimisationscheme.		
		`prototypeclass`	The classlabel of the prototype nearest to the datapoint.
		`relmat`	The relevance-matrix which will be updated.
		`dataclass`	The classlabel of the datapoint.
		`relrate`	The rate at which the relevances will adapt.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
`updateMatrixGeneral`	This function updates global matrix-relevances according to `LVQ1` and `general` optimisationscheme.		
		`relmat`	The relevance-matrix which will be updated.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`winclass`	The index of the prototype closest to the datapoint of the same classes the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`relrate`	The rate at which the relevances will adapt.

Function	Description	Parameters	Description
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.
`updateNormalClassWrap`	Wrapper function to update classwise relevances and distinguish between the `relevance` and `matrix` `relevancemode`.		
		`protolabels`	The labels of the prototypes.
		`winner`	The index of the prototype closest to the datapoint.
		`relevances`	A list with sets of relevances of which some will be updated.
		`dataclass`	The classlabel of the datapoint.
		`relrate`	The rate at which the relevances will adapt.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`relevancemode`	Shows what kind of relevances are used.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.
`updateNormalGlobalWrap`	Wrapper function to update global relevances and distinguish between the `relevance` and `matrix` `relevancemode`.		
		`protolabels`	The labels of the prototypes.
		`winner`	The index of the prototype closest to the datapoint.
		`relevances`	A list with sets of relevances of which some will be updated.
		`dataclass`	The classlabel of the datapoint.
		`relrate`	The rate at which the relevances will adapt.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`relevancemode`	Shows what kind of relevances are used.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.
`updateNormalLocalWrap`	Wrapper function to update local relevances and distinguish between the `relevance` and `matrix` `relevancemode`.		
		`protolabels`	The labels of the prototypes.
		`winner`	The index of the prototype closest to the datapoint.
		`relevances`	A list with sets of relevances of which some will be updated.
		`dataclass`	The classlabel of the datapoint.
		`relrate`	The rate at which the relevances. will adapt.

Function	Description	Parameters	Description
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`relevancemode`	Shows what kind of relevances are used.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.
`updateRelevances`	This function updates global vector-relevances according to `LVQ1` and `normal` optimisationscheme.		
		`prototypeclass`	The classlabel of the prototype nearest to the datapoint.
		`relvec`	The set of relevances.
		`dataclass`	The classlabel of the datapoint.
		`relrate` `difference`	The rate at which the relevances. A matrix containing the manhattan-distance of all the prototypes to the datapoint.
`updateRelevancesGeneral`	This function updates global vector-relevances according to `LVQ1` and `general` optimisationscheme.		
		`relevances`	The set of relevances which will be updated.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`winclass`	The index of the prototype closest to the datapoint of the same classes the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`relrate`	The rate at which the relevances will adapt.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.
`updateRelevancesGeneralWrap`	Wrapper function to update relevances according to `general optimisationscheme` and distinguish between global, local and classwise relevances.		
		`protolabels`	The labels of the prototypes.
		`winclass`	The index of the prototype closest to the datapoint of the same classes the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`relevances`	A list with sets of relevances of which some will be updated.
		`dataclass`	The classlabel of the datapoint.

Function	Description	Parameters	Description
		`relrate`	The rate at which the relevances will adapt.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`relevancescheme`	Shows how many relevance-sets are used.
		`relevancemode`	Shows what kind of relevances are used.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.
`updateRelevancesNormalWrap`	Wrapper function to update relevances according to `normal optimisationscheme` and distinguish between global, local and classwise relevances.		
		`protolabels`	The labels of the prototypes.
		`winner`	The index of the prototype closest to the datapoint.
		`relevances`	A list with sets of relevances of which some will be updated.
		`dataclass`	The classlabel of the datapoint.
		`relrate`	The rate at which the relevances will adapt.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`relevancescheme`	Shows how many relevance-sets are used.
		`relevancemode`	Shows what kind of relevances are used.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.

Function	Description	Parameters	Description
`updateRelevancesWrap`	Wrapper function to update relevances and distinguish between normal LVQ and generalized LVQ.		
		`protolabels`	The labels of the prototypes.
		`winclass`	The index of the prototype closest to the datapoint of the same classes the datapoint.
		`winnotclass`	The index of the prototype closest to the datapoint not of the same class as the datapoint.
		`relevances`	A list with sets of relevances of which some will be updated.
		`dataclass`	The classlabel of the datapoint.
		`relrate`	The rate at which the relevances will adapt.
		`difference`	A matrix containing the manhattan-distance of all the prototypes to the datapoint.
		`distance`	A vector containing the distances of all prototype to the datapoint.
		`optimisationscheme`	Determines how the optimal situation will be reached and thus how the prototype and relevances will be updated.
		`relevancescheme`	Shows how many relevance-sets are used.
		`relevancemode`	Shows what kind of relevances are used.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.

Validate

This is the highest-level module. This contains the entry-point-function `validate`, functions to facilitate training and testing of the prototypes and relevances and functions to facilitate nfold-cross-validation.

Function	Description	Parameters	Description
`constructFoldIndices`	Helper function. This function divides the data-indices in `nfold` subsets so the dataset can be used for nfold-cross-validation.		
		`data`	The dataset from which the indices will be extracted.
		`nfold`	The number of subsets to be made.
`constructTrainData`	Helper function. This function constructs a matrix containing a subset of the datapoints for training and testing purposes.		
		`folds`	A list of sets of indices. Each set of indices is a subset for nfold-cross-validation.
		`iteratie`	The number of the subset which is to be used in the next test.
`constructTrainLabels`	Helper function. This function constructs a vector containing a subset of the labels of the datapoints for training and testing purposes.		
		`folds`	A list of sets of labels of the dataset. Each set of labels is a subset for nfold-cross-validation.
		`iteratie`	The number of the subset which is to be used in the next test.

Function	Description	Parameters	Description
`nfoldcross`	This function performs nfold-cross-validation. It divides the dataset in `nfold` subsets which test the end-configuration of the prototypes. The end- configuration of each training is then tested on the other subsets.		
		`data`	The dataset, with labels in numerical form, which is to be used in nfold-cross-validation.
		`labels`	The labels of the dataset in original character form.
		`nfold`	The number of subsets to be made.
		`LVQscheme`	The version of LVQ to be used.
		`optimisationscheme`	Determines how the optimal situation will be reached and thus how the prototype and relevances will be updated.
		`prototypes`	A vector indexed by strings representing the classlabels. Each entry contains a number representing the number of prototypes to be used for the appropriate class.
		`learningrate`	The rate at which the prototype will adapt.
		`epochs`	The number of epochs to be used in training.
		`initscheme`	Determines the way the prototypes are initialized.
		`distscheme`	Determines what kind of measure is used to determine the distance from prototype to datapoint, when using the `LVQ1`-scheme.
		`relevancemode`	Determines what kind of relevances are used.
		`relevancescheme`	Determines how many relevance-sets are used.
		`relevances`	One or more relevance-sets if so provided by the user. Otherwise `NA`.
		`relrate`	The rate at which the relevances will adapt.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.
		`alfa`	When using `LVQscheme renyi` determines the version of Renyi-divergence to be used for calculating the distance.
		`show`	Determines if the current prototype- and relevance-configuration will be printed to the screen.
		`graphics`	Determines whether or not the trainingset and prototypes should be plotted during training. This is only available if the trainingset is 2-dimensional.
		`costcurve`	This determines if the progress of the costfunction should be among the output.

Function	Description	Parameters	Description
		`plotcurve`	Determines whether or not the progress of the constfunction should be plotted after each LVQ-run.
		`progress`	This determines if all the configurations of the prototypes should be among the output.
		`relevanceprogress`	This determines if all the configurations of the relevances should be among the output.
		`trainerror`	This determines if the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
		`testerror`	This determines if the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
		`trainerrorprogress`	This determines if the progress of the number of missclassifications when testing the end-configuration of the prototypes with the trainingset should be among the output.
		`testerrorprogress`	This determines if the progress of the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
		`costcurve`	This determines if the progress of the costfunction should be among the output.
`test`	This function tests a given prototype-configuration with the given dataset and records the number of missclassifications.		
		`data`	The dataset, with labels in numerical form, which is to be used in this test.
		`prototypes`	The prototype-configuration which will be tested.
		`protolabels`	The labels of the prototypes, in numerical form.
		`distscheme`	Determines the way the distance is calculated when using `LVQ1` LVQscheme.
		`relevancemode`	Shows what kind of relevances are used.
		`relevancescheme`	Shows how many relevance-sets are used.
		`LVQscheme`	The version of LVQ to be used in this test.
		`relevances`	The relevances-set(s), if any, which will be used in this test.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme.
		`alfa`	When using `LVQscheme renyi` determines the version of Renyi-divergence to be used for calculating the distance.

Function	Description	Parameters	Description
`train`	This function trains a set of prototypes with the given dataset.		
		`data`	The dataset, with labels in numerical form, which is to be used in this training.
		`labels`	The labels of the dataset in original character form.
		`testdata`	The dataset used for progess-testing, if applicable.
		`LVQscheme`	The version of LVQ to be used in this training.
		`optimisationscheme`	Determines how the optimal situation will be reached and thus how the prototype and relevances will be updated.
		`prototypes`	A vector indexed by strings representing the classlabels. Each entry contains a number representing the number of prototypes to be used for the appropriate class.
		`learningrate`	The rate at which the prototype will adapt.
		`epochs`	The number of epochs to be used in training.
		`initscheme`	Determines the way the prototypes are initialized.
		`distscheme`	Determines what kind of measure is used to determine the distance from prototype to datapoint, when using the `LVQ1`-scheme.
		`relevancemode`	Determines what kind of relevances are used.
		`relevancescheme`	Determines how many relevance-sets are used.
		`relevances`	One or more relevance-sets if so provided by the user. Otherwise `NA`.
		`relrate`	The rate at which the relevances will adapt.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.
		`alfa`	When using `LVQscheme renyi` determines the version of Renyi-divergence to be used for calculating the distance.
		`show`	Determines if the current prototype- and relevance-configuration will be printed to the screen.
		`graphics`	Determines whether or not the trainingset and prototypes should be plotted during training. This is only available if the trainingset is 2-dimensional.
		`costcurve`	This determines if the progress of the costfunction should be among the output.
		`plotcurve`	Determines whether or not the progress of the constfunction should be plotted after each LVQ-run.

Function	Description	Parameters	Description
		`progress`	This determines if all the configurations of the prototypes should be among the output.
		`relevanceprogress`	This determines if all the configurations of the relevances should be among the output.
		`trainerror`	This determines if the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
		`trainerrorprogress`	This determines if the progress of the number of missclassifications when testing the end-configuration of the prototypes with the trainingset should be among the output.
		`testerrorprogress`	This determines if the progress of the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
		`costcurve`	This determines if the progress of the costfunction should be among the output.
`traintest`	This function trains a set of prototypes with the given trainingset. The end-configuration is then tested with the testset.		
		`data`	The dataset, with labels in numerical form, which is to be used in this training.
		`labels`	The labels of the trainingset in original character form.
		`testdata`	The dataset, with labels in numerical form, which is to be used in this test.
		`LVQscheme`	The version of LVQ to be used in this training.
		`optimisationscheme`	Determines how the optimal situation will be reached and thus how the prototype and relevances will be updated.
		`prototypes`	A vector indexed by strings representing the classlabels. Each entry contains a number representing the number of prototypes to be used for the appropriate class.
		`learningrate`	The rate at which the prototype will adapt.
		`epochs`	The number of epochs to be used in training.
		`initscheme`	Determines the way the prototypes are initialized.
		`distscheme`	Determines what kind of measure is used to determine the distance from prototype to datapoint, when using the `LVQ1`-scheme.
		`relevancemode`	Determines what kind of relevances are used.
		`relevancescheme`	Determines how many relevance-sets are used.

Function	Description	Parameters	Description
		`relevances`	One or more relevance-sets if so provided by the user. Otherwise `NA`.
		`relrate`	The rate at which the relevances will adapt.
		`customdist`	Determines the distance-measure when using `LVQ1`-LVQscheme and thus also prototype- and relevance-updates.
		`alfa`	When using `LVQscheme renyi` determines the version of Renyi-divergence to be used for calculating the distance.
		`show`	Determines if the current prototype- and relevance-configuration will be printed to the screen.
		`graphics`	Determines whether or not the trainingset and prototypes should be plotted during training. This is only available if the trainingset is 2-dimensional.
		`costcurve`	This determines if the progress of the costfunction should be among the output.
		`plotcurve`	Determines whether or not the progress of the constfunction should be plotted after each LVQ-run.
		`progress`	This determines if all the configurations of the prototypes should be among the output.
		`relevanceprogress`	This determines if all the configurations of the relevances should be among the output.
		`trainerror`	This determines if the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
		`testerror`	This determines if the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
		`trainerrorprogress`	This determines if the progress of the number of missclassifications when testing the end-configuration of the prototypes with the trainingset should be among the output.
		`testerrorprogress`	This determines if the progress of the number of missclassifications when testing the end-configuration of the prototypes with the testset should be among the output.
		`costcurve`	This determines if the progress of the costfunction should be among the output.

Function	Description	Parameters	Description
`validate`	This is the entrypoint of the LVQTools. This function performs LVQ and validation according to the given parameters.		
		`validatescheme`	Determines how LVQ will be used. Determines how many training-runs and tests will be performed.
		`testdatapath`	The location of the testset, if one is read from memory.
		`nfold`	The number of testsets when using nfold-cross-validation.
		`LVQscheme`	The version of LVQ to be used in this training.
		`optimisationscheme`	Determines how the optimal situation will be reached and thus how the prototype and relevances will be updated.
		`inp`	The trainingset provided by the user, if applicable, otherwise `NA`.
		`testinp`	The testset provided by the user, if applicable, otherwise `NA`.
		`prototypeoutput`	Determines whether or not the prototype endconfiguration should be among the output.
		`relevanceoutput`	Determines whether or not the relevance endconfiguration should be among the output.
		`costcurve`	Determines whether or not the progress of the costfunction should be plotted.
		`progress`	Determines whether or not the progress of the prototypes should be shown.
		`relevanceprogress`	Determines whether or not the progressof the relevances should be shown.
		`trainerror`	Determines whether or not the trainerror should be shown.
		`testerror`	Determines whether or not the testerror should be shown.
		`trainerrorprogress`	Determines whether or not the progress of the trainerror should be shown.
		`testerrorprogress`	Determines whether or not the progress of the testerror should be shown.
		`datapath`	The location of the trainingset, if one is read from memory.
		`normalizescheme`	Determines how the data should be normalized.
		`normalclasswise`	Determines which class will be used as a basis for normalization.
		`replaceNA`	Determines if `NA`-values should be replaced
		`replaceclasswise`	Determines if the replacement of `NA` -values should consider classes.
		`prototypes`	A vector indexed by strings representing the classlabels. Each entry contains a number representing the number of prototypes to be used for the appropriate class.
		`learningrate`	The rate at which the prototypes will adapt. It contains either a number, between 0 and 1 or a vector of such numbers of length `epochs`.

Function	Description	Parameters	Description
		`epochs`	The number of times the training-data will be used to update the prototypes,
		`initscheme`	Determines the way the prototypes are initialized.
		`distscheme`	Determines what kind of measure is used to determine the distance from prototype to datapoint, when using the `LVQ1`-scheme.
		`relevancemode`	Determines the sort of relevances used.
		`relevancescheme`	Determines the number of relevances used.
		`relevances`	The relevances to be used in training if they are provided by the user, if not `vector()`.
		`relrate`	The rate at which the relevances will adapt. It contains either a number, between 0 and 1 or a vector of such numbers of length `epochs`.
		`customdist`	When using `distschemecustom` determines how the distance is calculated.
		`alfa`	When using `LVQscheme renyi` determines the version of Renyi-divergence to be used for calculating the distance.
		`show`	Determines whether or not progress should be shown during the training.
		`graphics`	Determines whether or not the trainingset and prototypes should be plotted during training. This is only available if the trainingset is 2-dimensional.
		`plotcurve`	Determines whether or not the progress of the constfunction should be plotted after each LVQ-run.