LVQTools
Bachelor project: implementing LVQ in R

Sander Kelders

June 24, 2010

Abstract

We present an implementation of several LVQ algorithms in one tool for easy use and evaluation of results. LVQtools contains implementations of heuristic and generalized versions of LVQ1, RLVQ, MLVQ and entropy based LVQ. The tool provides customization of LVQ versions, initialization, output and more. First we give an introduction of LVQ followed by the algorithms implemented. Afterwards the possibilities of the package are discussed and a few results are presented.

Introduction

Learning Vector Quantization (LVQ) as proposed by Kohonen [1] is a rather simple and intuitive method of classification. It is also quite a powerful tool and easy to implement. Many variants exist to the 'plain vanilla' LVQ-variant and research into new systems and variants is still being conducted. Whoever uses LVQ can choose between all the variants to make the classification as simple or complex to suit his wishes, whether the problem is a simple choice between two strictly separated classes to mutli-class problems with closely related classes. Among the variants exist purely heuristic methods, methods with a mathematic basis, relevance learning methods and many more. Because it is such a flexible and intuitive tool LVQ has become quite popular and is being used in a variety of fields such as bioinformatics, image analysis, robotics, etc.

The implementation is developed in R [2], which is a language and environment for statistical computation and graphics. R provides many tools for statistical analysis and visualization of results. It provides an environment to work in and is extensible through scripting and up/downloading of packages. R is particularly popular in bioinformatics and is therefore chosen for this project. The LVQTools package will provide several LVQ-versions along with different methods of validation in order to advertise LVQ without the need for programming skills or a thorough knowledge of LVQ.

The basis for this project was laid by Kohonen's *Learning Vector Quantization* [1]. Ever since the LVQ-algorithm has been elaborated on by various people. Among those elaborations the possiblility of relevance learning. B. Hammer and T. Villmann propose a set of weighting factors for the input dimensions in their paper *Generalized relevance learning vector quantization* [3]. With this set of weighting factors (relevances) the level of importance of the seperate features can be specified. By adjusting the relevances and at the same time the prototypes the level of importance of each feature can be evaluated and important and unimportant features can be identified. This works well when the different classes exhibit a certain similarity but this is not always the case. The classes could be very different and the importance of features could too. For such different classes more than one set of relevances might be used. Classwise relevances use a set of relevances for each class and local relevances use a set for each prototype as opposed to the single set used in global relevances.
Recently a new version of relevance learning in LVQ has been developed using relevance matrices.[5,6] Instead of a relevance vector a matrix is used.With a matrix not only the importance of each feature can be evaluated but correlations between features might be found and evaluated.
Cost function based LVQ is another elaboration. Sato and Yamada proposed updating the prototypes by gradient descent in their work *Generalized learning vector quantization.* [4] They propose to base the updating of prototypes on a 'cost function'. This cost function is created with heuristics in mind and is an indication of the effectiveness of the current prototype configuration. By minimizing or maximizing the function (dependent on what kind of function) the best configuration is sought for the prototypes as well as the relevances.
The last version of LVQ discussed here is divergence-based LVQ. In *Divergence based Learning Vector Quantization* [7] a whole new distance measure is proposed. Instead of using a Euclidean metric, feature vectors are interpreted as probability distributions. This paves the way to use dissimilarity measures between probablilty distributions as a

distance measure. Many dissimilarity measures exist but in the next section we discuss only two divergence based dissimilarity measures: Renyi divergence and Cauchy-Schwarz divergence.

Algorithm

LVQ is a classification-method. It uses prototypes to represent large sets of data, which contain feature vectors. The prototypes are adjustable feature vectors which are trained with one (or more) trainingset(s). When presented with new data, of which the class is not yet determined, each seperate feature vector is classified by using the prototypes. In order to classify the vectors a certain distance measure is needed to compare the training samples with the prototypes. This distance measure is dependent on the LVQ-variant, however a squared euclidean distance measure is often used. The way the prototypes are used to classify new data as well as the way the prototypes are trained is highly dependent on the LVQ-version.

In general LVQ is as follows:
Let there be a training set which is comprised of samples of the form $(\boldsymbol{\xi}, y) \in \mathbb{R}^N \times \{1, ..., C\}$, where N is the dimensions of the feature vector and C the number of classes. A seperate test set might also be present and it will be of the same form. The prototypes are of a similar form: $\boldsymbol{\omega} \in \mathbb{R}^N$ and for each class $c(\boldsymbol{\omega}) \in \{1, ...C\}$ at least one prototype exists. Let there be a learning rate τ_1 for the prototypes and for the relevances τ_2 which controls the rate at which the prototypes and the relevances respectively adapt. Let $0 < \tau_1 < 1$, $0 < \tau_2 < 1$ and usually $\tau_2 \ll \tau_1$ hold.
The following actions will be performed:

1. Present a sample from the training set usually randomly selected.
2. Determine the closest prototype or a set of closest prototypes, based on the complete distance measure.
3. Update prototypes according to selected scheme and if appropriate update relevances.

This is done for every sample in the training set. Such a round is called an epoch and in one training usually many epochs are performed. The different LVQ schemes differ in determining the closest prototype and the update rules. An LVQ scheme determines the closest prototype in how the distance between prototypes and samples is calculated but also by using or not using different kinds of relevances. The update rules might be heuristic or might have a mathematical base. The learning rate of the prototypes and relevances as well as the use of relevances is also of influence.
During and/or after the training the prototype-configuration might be tested with a test set and/or training set to evaluate the performance of the training.

LVQ1

LVQ1 is as follows:
Let the distance measure be a squared Euclidean distance measure:

$$d(\boldsymbol{\omega}, \boldsymbol{\xi}) = \sum_i (\xi_i - \omega_i)^2 \tag{1}$$

The update is as follows:

$$\text{for } c(\boldsymbol{\omega}_W) = c(\boldsymbol{\xi}) \text{ and } \forall i : \omega_{W,i}^{new} = \omega_i + \tau_1(\xi_i - \omega_i) \tag{2}$$

$$\text{for } c(\boldsymbol{\omega}_W) \neq c(\boldsymbol{\xi}) \text{ and } \forall i : \omega_{W,i}^{new} = \omega_i - \tau_1(\xi_i - \omega_i) \tag{3}$$

where $\boldsymbol{\omega}_W$ is the closest prototype to the sample.

Relevances

The addition of relevances adds a couple of operations to the LVQ1-algorithm as discussed above:
Let $\boldsymbol{\lambda}$ be a vector of length N. Let $\boldsymbol{\lambda}$ satisfy the following conditions:

$$\forall i : \lambda_i \geq 0 \tag{4}$$

$$\sum_i \lambda_i = 1 \tag{5}$$

To incorporate the relevances they will become a part of the distance measure:

$$d(\boldsymbol{\omega}, \boldsymbol{\xi}) = \sum_i \lambda_i (\xi_i - \omega_i)^2 \tag{6}$$

When the prototypes are updated so will the relevances be updated. How the relevances are updated is based on the resulting distance and if the contribution of the dimension is high (feature vector and prototype have a great distance in respect to the corresponding dimension) or low (feature vector and prototype are close in respect to the corresponding dimension). When using the Euclidean distance measure as in (1) the update rules for relevances are as follows:

The nearest prototype is of the same class:

$$\lambda_i^{new} = \lambda_i - \tau_2(\xi_i - \omega_i)^2 \tag{7}$$

The nearest prototype is not of the same class:

$$\lambda_i^{new} = \lambda_i + \tau_2(\xi_i - \omega_i)^2 \tag{8}$$

After the update the relevance-vector has to be normalized so it once again satisfies (4) and (5).

The combination of the update and the normalization form a heuristic to the relevances updates. This heuristic is as follows:

1. When the nearest prototype is of the same class and

(a) the contribution of the dimension is high, the relevance will be decreased.

(b) the contribution of the dimension is low, the relevance will be increased.

2. When the nearest prototype is not of the same class and

(a) the contribution of the dimension is high, the relevance will be increased.

(b) the contribution of the dimension is low, the relevance will be decreased.

Generalized LVQ

Generalized LVQ (or GLVQ) bases its update on a cost function. The cost function used here is of the form:

$$f = \sum_i \Phi(\mu) \tag{9}$$

In general Φ is a sigmoid function to avoid extreme values and keep the algorithm stable. In this case the identity function is chosen because the choice of μ keeps the values between 1 and -1:

$$\Phi(\mu) = \mu \text{ and } \mu = \frac{d_J - d_K}{d_J + d_K} \tag{10}$$

where

$$d_J = d(\boldsymbol{\omega}_J, \boldsymbol{\xi}) \text{ with } c(\boldsymbol{\omega}_J) = c(\boldsymbol{\xi}) \tag{11}$$

$$d_K = d(\boldsymbol{\omega}_K, \boldsymbol{\xi}) \text{ with } c(\boldsymbol{\omega}_K) \neq c(\boldsymbol{\xi}) \tag{12}$$

This cost function is a ratio between the difference, of the distance between the closest correct prototype $\boldsymbol{\omega}_J$ and the closest incorrect prototype $\boldsymbol{\omega}_K$, and the sum of these distances. The updating of the prototypes is now a stochastic gradient descent based on (10). Assuming there is a learningrate τ_1 and using a Euclidean distance measure (1), updating the prototypes takes the following form:

$$\boldsymbol{\omega}_J^{new} = \boldsymbol{\omega}_J + \tau_1 \cdot \Phi'(\mu) \cdot \mu^+ \cdot 2(\boldsymbol{\xi} - \boldsymbol{\omega}_J) \tag{13}$$

$$\boldsymbol{\omega}_K^{new} = \boldsymbol{\omega}_K + \tau_1 \cdot \Phi'(\mu) \cdot \mu^- \cdot 2(\boldsymbol{\xi} - \boldsymbol{\omega}_K) \tag{14}$$

where

$$\mu^+ = \frac{\delta\mu}{\delta d_J} = \frac{2 \cdot d_K}{(d_J + d_K)^2} \tag{15}$$

$$\mu^- = \frac{\delta\mu}{\delta d_K} = \frac{-2 \cdot d_J}{(d_J + d_K)^2} \tag{16}$$

When using relevances updating is only slightly different:

$$\forall i : \omega_{J,i}^{new} = \omega_{J,i} + \tau_1 \cdot \Phi'(\mu) \cdot \mu^+ \cdot 2\lambda_i(\xi_i - \omega_{J,i}) \tag{17}$$

$$\forall i : \omega_{K,i}^{new} = \omega_{K,i} + \tau_1 \cdot \Phi'(\mu) \cdot \mu^- \cdot 2\lambda_i(\xi_i - \omega_{K,i}) \tag{18}$$

Still the relevances should also be updated and this is also based on the costfunction when using GLVQ. This is only slighty different then updating the prototypes:

$$\forall i : \delta\lambda_i = \tau_2 \cdot \Phi'(\mu) \cdot (\mu^+ \cdot (\xi_i - \omega_{J,i})^2 + \mu^- \cdot (\xi_i - \omega_{K,i})^2) \tag{19}$$

And for local and classwise relevances:

$$\forall i : \delta\lambda_i^J = \tau_2 \cdot \Phi'(\mu) \cdot (\mu^+ \cdot (\xi - \omega_{J,i})^2 \tag{20}$$

$$\forall i : \delta\lambda_i^K = \tau_2 \cdot \Phi'(\mu) \cdot \mu^- \cdot (\xi_i - \omega_{K,i})^2 \tag{21}$$

After the update the relevances have to be normalized so they once again satisfy (4) and (5).

Matrix LVQ

With a relevance-matrix not only can be established how relevant the seperate dimensions are, but also if correlations between these dimensions are important. When using a Euclidean distance measure (1) as a basis, the distance measure using relevance-matrix looks as follows:

$$d^\Lambda(\boldsymbol{\omega}, \boldsymbol{\xi}) = (\boldsymbol{\xi} - \boldsymbol{\omega})^T \Lambda (\boldsymbol{\xi} - \boldsymbol{\omega}) \tag{22}$$

with $\Lambda \in \mathbb{R}^{N \times N}$. Λ should positive definite in order to define a useable metric. In order to enforce this we substitute:

$$\Lambda = \Omega^T \Omega \tag{23}$$

The distance calculations will have the following form: $\mathbf{x}^T \Lambda \mathbf{x}$. We can substitute Λ, which gives us: $\mathbf{x}^T \Omega^T \Omega \mathbf{x} = (\Omega^T \mathbf{x})^2$. And $(\Omega^T \mathbf{x})^2 \geq 0$ for all $\mathbf{x}$.

Using Matrix-LVQ the distance measure has the following form:

$$d^\Lambda(\boldsymbol{\omega}, \boldsymbol{\xi}) = (\boldsymbol{\xi} - \boldsymbol{\omega})^T \Lambda (\boldsymbol{\xi} - \boldsymbol{\omega}) = (\boldsymbol{\xi} - \boldsymbol{\omega})^T \Omega^T \Omega (\boldsymbol{\xi} - \boldsymbol{\omega}) = \sum_{ijk} (\xi_i - \omega_i) \Omega_{ki} \Omega_{kj} (\xi_j - \omega_j) \tag{24}$$

A heuristic way of updating is still possible, however MLVQ is usually used in conjunction with the Generalized version, which gives rise to GMLVQ. For the updates we need to take the derivative of d^Λ with respect to $\boldsymbol{\omega}$ and Ω:

$$\nabla_\omega d^\Lambda(\boldsymbol{\omega}, \boldsymbol{\xi}) = -2\Lambda(\boldsymbol{\xi} - \boldsymbol{\omega}) = -2\Omega^T \Omega (\boldsymbol{\xi} - \boldsymbol{\omega}) \tag{25}$$

$$\frac{\delta d^\Lambda(\boldsymbol{\omega}, \boldsymbol{\xi})}{\delta \Omega_{lm}} = \sum_j (\xi_m - \omega_m) \Omega_{lj} (\xi_j - \omega_j) + \sum_i (\xi_i - \omega_i) \Omega_{li} (\xi_m - \omega_m) = 2 \cdot (\xi_m - \omega_m) [\Omega(\boldsymbol{\xi} - \boldsymbol{\omega})]_l \tag{26}$$

This results in the following updates:

$$\boldsymbol{\omega}_J^{new} = \boldsymbol{\omega}_J + \tau_1 \cdot \Phi'(\mu) \cdot \mu^+ \cdot 2\Lambda(\boldsymbol{\xi} - \boldsymbol{\omega}_J) \tag{27}$$

$$\boldsymbol{\omega}_K^{new} = \boldsymbol{\omega}_K + \tau_1 \cdot \Phi'(\mu) \cdot \mu^- \cdot 2\Lambda(\boldsymbol{\xi} - \boldsymbol{\omega}_K) \tag{28}$$

$$\Delta\Omega_{lm} = \tau_2 \cdot \Phi'(\mu) \cdot (\mu^+ \cdot 2 \cdot (\xi_m - \omega_m)[\Omega(\boldsymbol{\xi} - \boldsymbol{\omega})]_l + \mu^- 2 \cdot (\xi_m - \omega_m)[\Omega(\boldsymbol{\xi} - \boldsymbol{\omega})]_l) \tag{29}$$

And for local and classwise relevances:

$$\Delta\Omega_{lm}^J = \tau_2 \cdot \Phi'(\mu) \cdot \mu^+ \cdot 2 \cdot (\xi_m - \omega_m^J)[\Omega(\boldsymbol{\xi} - \boldsymbol{\omega}^J)]_l \tag{30}$$

$$\Delta\Omega_{lm}^K = \tau_2 \cdot \Phi'(\mu) \cdot \mu^- \cdot 2 \cdot (\xi_m - \omega_m^K)[\Omega(\boldsymbol{\xi} - \boldsymbol{\omega}^K)]_l \tag{31}$$

In order to keep the algorithm stable the matrix needs to be normalized after every update. So we enforce:

$$\sum_i \Lambda_{ii} = 1 \tag{32}$$

This way we keep the diagonal stable, which correspond to the values of a vector of relevances. To this end we note that:

$$\Lambda_{ii} = \sum_k \Omega_{ki}\Omega_{ki} = \sum_k (\Omega_{ki})^2 \tag{33}$$

So to normalize we divide all elements of Ω by

$$(\sum_{ki} (\Omega_{ki})^2)^{1/2} \tag{34}$$

The heuristic update of the relevance matrix is much the same as the heuristic update for the relevance vector:
The nearest prototype is of the same class:

$$\Omega_{ij}^{new} = \Omega_{ij} - \tau_2(|\xi_i - \omega_i| \cdot |\xi_j - \omega_j|) \tag{35}$$

The nearest prototype is not of the same class:

$$\Omega_{ij}^{new} = \Omega_{ij} + \tau_2(|\xi_i - \omega_i| \cdot |\xi_j - \omega_j|) \tag{36}$$

Afterwards the matrix needs to normalized the same way as in GMLVQ.

This combination of updating and normalization again forms the following heuristic:

1. When the nearest prototype is of the same class and
   (a) the contribution of the dimension is high, the relevance will be decreased.
   (b) the contribution of the dimension is low, the relevance will be increased.
2. When the nearest prototype is not of the same class and
   (a) the contribution of the dimension is high, the relevance will be increased.
   (b) the contribution of the dimension is low, the relevance will be decreased.

Entropy based LVQ

Entropy based LVQ interprets the prototypes and training samples as probability distributions and uses dissimilarity measures as distance measures. The feature vectors and the prototypes have to satisfy a few properties in order for this measure to work. They must be nonnegative and each vector must sum up to one:

$$\forall \omega_i \in \boldsymbol{\omega}, \omega_i \geq 0 \tag{37}$$

and

$$\forall \xi_i, \in \boldsymbol{\xi}, \xi_i \geq 0 \tag{38}$$

$$\forall \boldsymbol{\xi}, \sum_i \xi_i = 1 \tag{39}$$

and

$$\forall \boldsymbol{\omega}, \sum_i \omega_i = 1 \tag{40}$$

Two examples of dissimilarity measures used in *Divergence based Learning Vector Quantization* are Cauchy-Schwarz divergence and Renyi divergence. The distance measures then look like this:

$$d_{cs}(\boldsymbol{\xi}, \boldsymbol{\omega}) = \frac{1}{2} log(\boldsymbol{\xi}^2 \boldsymbol{\omega}^2) - log(\boldsymbol{\xi}^T \boldsymbol{\omega}) \tag{41}$$

$$d_{re}(\boldsymbol{\xi}, \boldsymbol{\omega}) = \frac{1}{\alpha - 1} log(\sum_i \xi_i (\frac{\xi_i}{w_i})^{\alpha - 1}) \tag{42}$$

And the corresponding derivatives read:

$$\frac{\delta d_{cs}(\boldsymbol{\xi}, \boldsymbol{\omega})}{\delta \omega_j} = \frac{-\omega_j}{\boldsymbol{\omega}^2} - \frac{\xi_j}{\boldsymbol{\xi}^T \boldsymbol{\omega}} \tag{43}$$

$$\frac{\delta d^{\alpha}_{re}(\boldsymbol{\xi}, \boldsymbol{\omega})}{\delta \omega_j} = \frac{1}{\alpha - 1} \frac{-(\alpha - 1)(\frac{\xi_j}{\omega_j})^{\alpha}}{\sum_i \xi_i \cdot (\frac{\xi_i}{\omega_i})^{\alpha - 1}} \tag{44}$$

These derivatives can be used in Generalized LVQ and the result in the following updates:
CauchySchwarz:

$$\omega^{new}_{J,j} = \omega_{J,j} + \tau_1 \cdot \Phi'(\mu) \cdot \mu^+ \cdot \frac{-\omega_{J,j}}{\boldsymbol{\omega}^2_J} - \frac{\xi_j}{\boldsymbol{\xi}^T \boldsymbol{\omega}_J}) \tag{45}$$

$$\omega^{new}_{K,j} = \omega_{K,j} + \tau_1 \cdot \Phi'(\mu) \cdot \mu^- \cdot \frac{-\omega_{K,j}}{\boldsymbol{\omega}^2_K} - \frac{\xi_j}{\boldsymbol{\xi}^T \boldsymbol{\omega}_K}) \tag{46}$$

Renyi:

$$\omega^{new}_{J,j} = \omega_{J,j} + \tau_1 \cdot \Phi'(\mu) \cdot \mu^+ \cdot \frac{1}{\alpha - 1} \frac{-(\alpha - 1)(\frac{\xi_j}{\omega_{J,j}})^{\alpha}}{\sum_i \xi_i \cdot (\frac{\xi_i}{\omega_{J,i}})^{\alpha - 1}}) \tag{47}$$

$$\omega_{K,j}^{new} = \omega_{K,j} + \tau_1 \cdot \Phi'(\mu) \cdot \mu^- \cdot \frac{1}{\alpha - 1} \frac{-(\alpha - 1)(\frac{\xi_j}{\omega_{K,j}})^\alpha}{\sum_i \xi_i \cdot (\frac{\xi_i}{\omega_{K,i}})^{\alpha - 1}}) \tag{48}$$

In the case of renyi-divergence the variable α determines what order of renyi-divergence is used. A popular version is 2nd-order renyi-divergence.

Another heuristic and simpler way of updating is by winner takes all and only taking the derivative of the distance measure instead of the whole cost function. The updates then look like this:
Cauchy-Schwarz:

$$\omega_i^{new} = \omega_i \pm \tau_1 \frac{\omega_i}{\sum_i \omega_i^2} - \frac{\xi_i}{\sum_i \xi_i \omega_i} \tag{49}$$

Renyi:

$$\omega_i^{new} = \omega_i \pm \tau_1 \frac{1}{\alpha - 1} - \frac{(\alpha - 1)(\frac{\xi_i}{\omega_i})^\alpha}{\sum_i \xi_i \cdot (\frac{\xi_i}{\omega_i})^{\alpha - 1}} \tag{50}$$

When the prototype is of the same class as the presented sample the update is substracted from the old prototype in order to find a minimum and when it is not of the same class the update is added to find a maximum.

After each update the prototypes must be normalized so they satisfy (37) and (40) again.

Implementation

In order to realize the bundling of all these versions of LVQ a clear structure and division of functionality is needed. A first attempt strived to make a framework in which the user could specify the update-, distancemeasure- and cost-functions. This would have made the LVQTools very flexible and extensible. However the resulting framework was quite sluggish, since no code-optimization was possible in the user-specified functions and since every single possible piece of information had to be calculated for every update, because it was not known in advance what information the user-functions would use. The flexibility and extensibility was lacking as well since the user could only make the user-functions on basis of a limited set of information pieces. The disappointing speed and lack of flexibility was the driving force behind the decision to make a few specific and optimized implementations of LVQ.

For a full description read the documentation: *LVQToolsdocumentation*, *LVQToolsUsage*, and *LVQToolsOutputUsage*.

LVQ versions

The different algorithms can be chosen through a layered structure. The core of the distance measure the distance calculation without relevances can be determined by choosing between three different versions:

1. LVQ1

2. divergence based Cauchy-Schwarz LVQ
3. divergence based Renyi LVQ

In the case of divergence based LVQ the distance measure is set exactly as in the *Algorithm* section. In the case of LVQ1 the distance measure can partly be determined by the user. The user can determine p in:

$$d(\boldsymbol{\omega}, \boldsymbol{\xi}) = \sum_i |\xi_i - \omega_i|^p \tag{51}$$

with $p \in \mathbb{R}$ and $1 \leq p$.

Each distance measure can be combined with either heuristic update rules or generalized update rules based on a cost function which are both discussed in the *Algorithm* section.
In the case of LVQ1 relevances can be used. Both vector relevances and matrix relevances or none at all can be used but when using matrix relevances only a Euclidean distance measure can be used.
Through this structure the following algorithms can be used:

1. LVQ1
2. Relevance LVQ
3. Matrix LVQ (only with Euclidean distance measure)
4. Generalized LVQ
5. Generalized Relevance LVQ
6. Generalized Matrix LVQ (only with Euclidean distance measure)
7. Cauchy-Schwarz LVQ
8. Renyi LVQ
9. Generalized Cauchy-Schwarz LVQ
10. Generalized Renyi LVQ

The learning rate of both the prototypes and the relevances can be specified by either one number or a sequence of numbers of as many numbers as there are epochs. This way the learning rate can also be given by a precalculated function.

Validation

LVQ can be used for a variety of goals from obtaining a usable set of prototypes to the evaluation of relevances. To accommodate the different approaches LVQTools can be used in three different ways:

1. Train once using given dataset
2. Train once using given training set and test using test set
3. Divide given dataset in a number of sets and perform n-fold-crossvalidation

Input

Input can be given in different ways and can be preprocessed to fit the users needs. A file can be specified to read a training or testing set from, but these sets can also be specified in a variable. The latter is especially usefull when using RExcel to import values from Excel files into R.
Preprocessing of input can be done in two ways:

1. Replacing missing values
2. Normalizing data sets

Missing values can be replaced by the mean of the data set or by the mean of the class the sample it belongs to.

Normalizing data sets can be done in three different ways:

1. using ztransform
2. using Inter Quantile Range
3. transforming samples so they sum up to 1 satisfying (39)

Using ztransform or the IQR a specific class can be selected to be the basis for the normalization. If no class is selected for this purpose the whole data set is used as the basis for normalization. The third option transforms samples to sum up to 1 satisfying (39) only and not (38). The user needs to use a data set with nonnegative values. It was decided not to transform a data set to all nonnegative values, since the transformation to nonnegative and forcing each sample to sum up to 1 might tranform the data set so that all the samples' ratio's would be completely different resulting in an entirely different data set.

Initializaton

Prototype-initialization is possible by several methods:

1. random window: randomly within the range of the training set
2. random sample: the same value as a random sample
3. zero: all values at 0
4. mean: all values at the mean of the whole training set
5. class mean: all values at the mean of the appropriate class

Zero cannot be used with entropy based LVQ since it would violate (40). Random window can be used with entropy based LVQ but the prototypes have to be normalized immediately after initialization.

Relevances be they matrix or vector can be initialized randomly or can be provided by the user.

Output

To make this a useful tool information about every LVQ-run should be available. To this end many variables can be given as output and many functions for output-visualization are available. The following information can be specified as output:

1. prototype end configuration
2. prototype progress: the prototype configuration after every epoch
3. relevances end configuration
4. relevance progress: the relevance configuration after every epoch
5. train error: the percentage of errors when testing the prototype end configuration with the training set
6. train error progress: the percentage of errors when testing the prototype configuration with the training set after every epoch
7. test error: the percentage of errors when testing the prototype end configuration with the test set
8. test error progress: the percentage of errors when testing the prototype configuration with the test set after every epoch
9. the progress of the cost function

With generalized LVQ versions the cost cost function as in (10) is used. With heuristic LVQ versions the following cost function is used for cost function progress:

$$\sum_i d(\boldsymbol{\omega}_{W,i}, \boldsymbol{\xi}_i) \tag{52}$$

where $\boldsymbol{\omega}_{W,i}$ is the closest prototype to $\boldsymbol{\xi}$ with $c(\boldsymbol{\omega}_{W,i}) = c(\boldsymbol{\xi}_i)$.

All of these values can be plotted or shown on screen by the `show...` functions and they can be extracted from the resulting output object by using the `get...` functions.

Efficiency and stability

Since LVQ usually uses many epochs to get stable and useful results efficiency is an important factor in this Tool. Wherever possible R's functionality and quick vector-arithmetic is used. To this end different distance-calculating-functions are used for training and testing. When training the distance from all prototypes to each presented feature-vector are calculated in one assignment as opposed to using for-loops which do not use R's vector-arithmetic. When testing the distance from a prototype to all feature-vectors are calulated in one assignment. This makes for quite an efficient program, however this is lost when training with other than global relevances or when using matrix-relevances. In R there is no quick way to multiply a set of vectors with a matrix and when using more than one set of relevances each calculation needs another set and there is no quick way to multiply a set of vectors with each of another set of vectors either.

The divergence based method using Renyi-divergence is prone to numerical instability. Some of the update rules divide by seperate components of the prototypes. These values can become very small resulting in very large update values. To combat this instability the values of the prototypes cannot become smaller than 1×10^{-3}.

Results

In order to demonstrate some of the functions of LVQTools we show and discuss in this section some results from experiments with LVQTools. For this we use the Winsconsin Breast Cancer data set from the UCI data repository [8]. All presented graphs were produces with the tool itself. Each experiment was performed with the following configuration:

1. 1 prototype per class
2. 50 epochs
3. the prototype learning rate $\tau_1 = \frac{0.005}{1+0.0001(t-1)}$, with t = current epoch
4. the relevance learning rate $\tau_2 = \frac{0.0001}{1+0.0001(t-1)}$, with t = current epoch
5. prototype initialization by random sample
6. n-fold-cross validation with $n = 3$
7. normalization for GMLVQ by ztransform
8. normalization for Cauchy-Schwarz and Renyi by transforming samples so they sum up to 1 satisfying (39)

In *figure 1 GMLVQ results* and *figure 2 GMLVQ results train and testerror* the results of Generalized Matrix LVQ are shown. In each collumn the progress of the cost function, the test error, the train error and the end configuration of the relevance matrix are shown respectively. For all three folds these result are shown ending with the last train and test error per fold. The cost function as well as the train and testerror show a steady decline towards a minimum resulting in a train and test error around or below 10%.

In *figure 3 Cauchy-Schwarz LVQ results* the results of Caucy-Schwarz LVQ are shown. In each row the progress of the cost function, the test error and the train error are shown respectively. For all three folds these result are shown ending with the last train and test error per fold. The cost function as well as the train and test error quickly find a minimum but further training slowly removes the prototypes from this position. At first glance the run of the first fold doesn' seem to perform very well since the test and train error seem to increase greatly after a few epochs. This is due to a 'lucky' initialization where the prototypes are initilized at a position already producing a low train and test error.

In *figure 4 Renyi LVQ results* the results of Renyi LVQ are shown. In each row the progress of the cost function, the test error and the train error are shown respectively. For all three folds these result are shown ending with the last train and test error per fold. In these runs the cost function and the test and train errors first increase until some threshold where suddenly a minimum is found and maintained.

The results shown here are not necessarily the best results possible. Better results might be achieved through the customization of variables, usage of more epochs and so on. The graphs shown here are merely to demonstrate some of the possibilities of LVQTools.

Figure 1: GMLVQ results

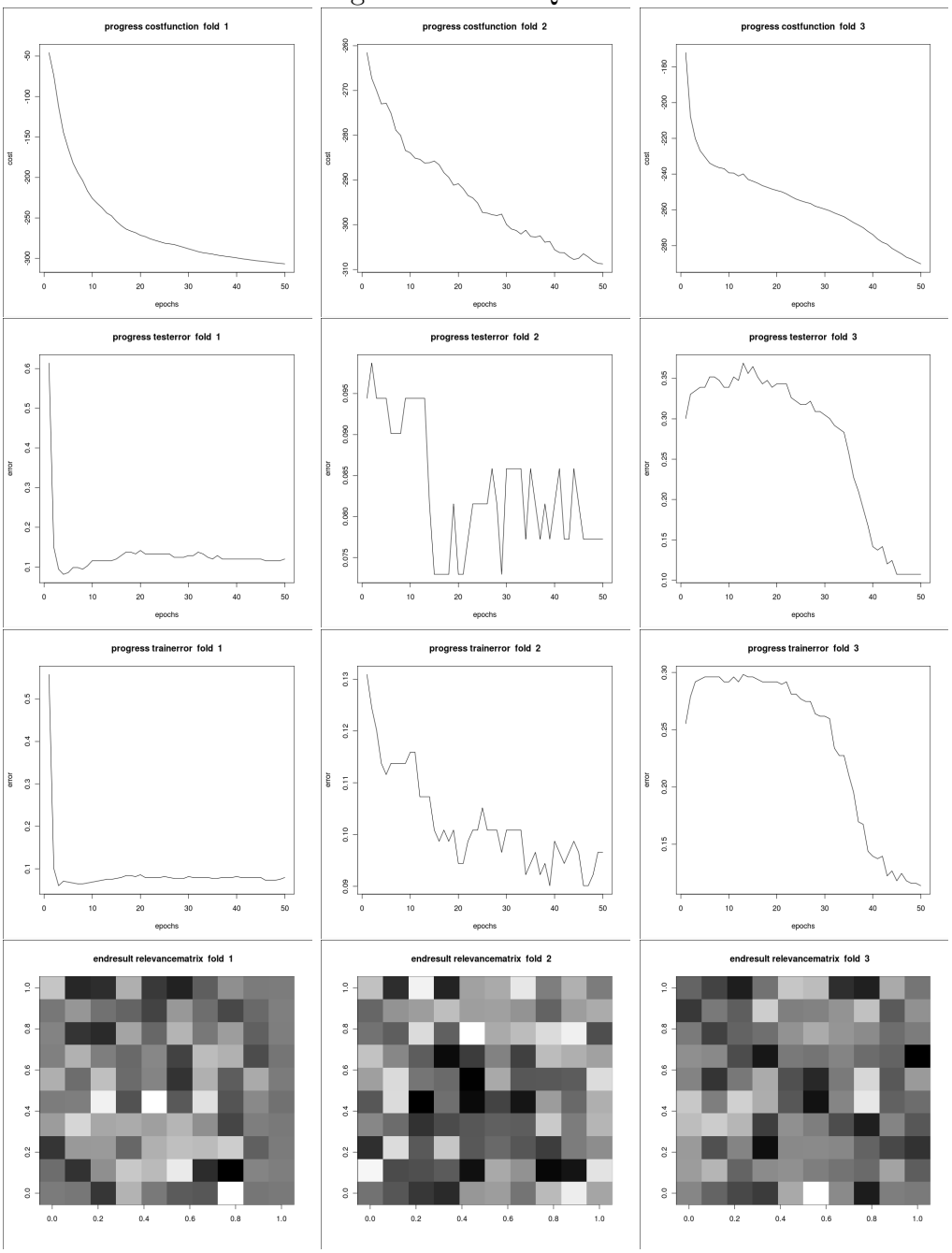


Figure 2: GMLVQ results train and testerror

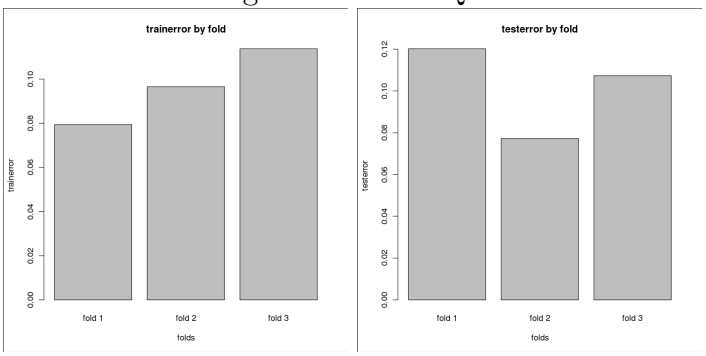


Figure 3: Cauchy-Schwarz LVQ results

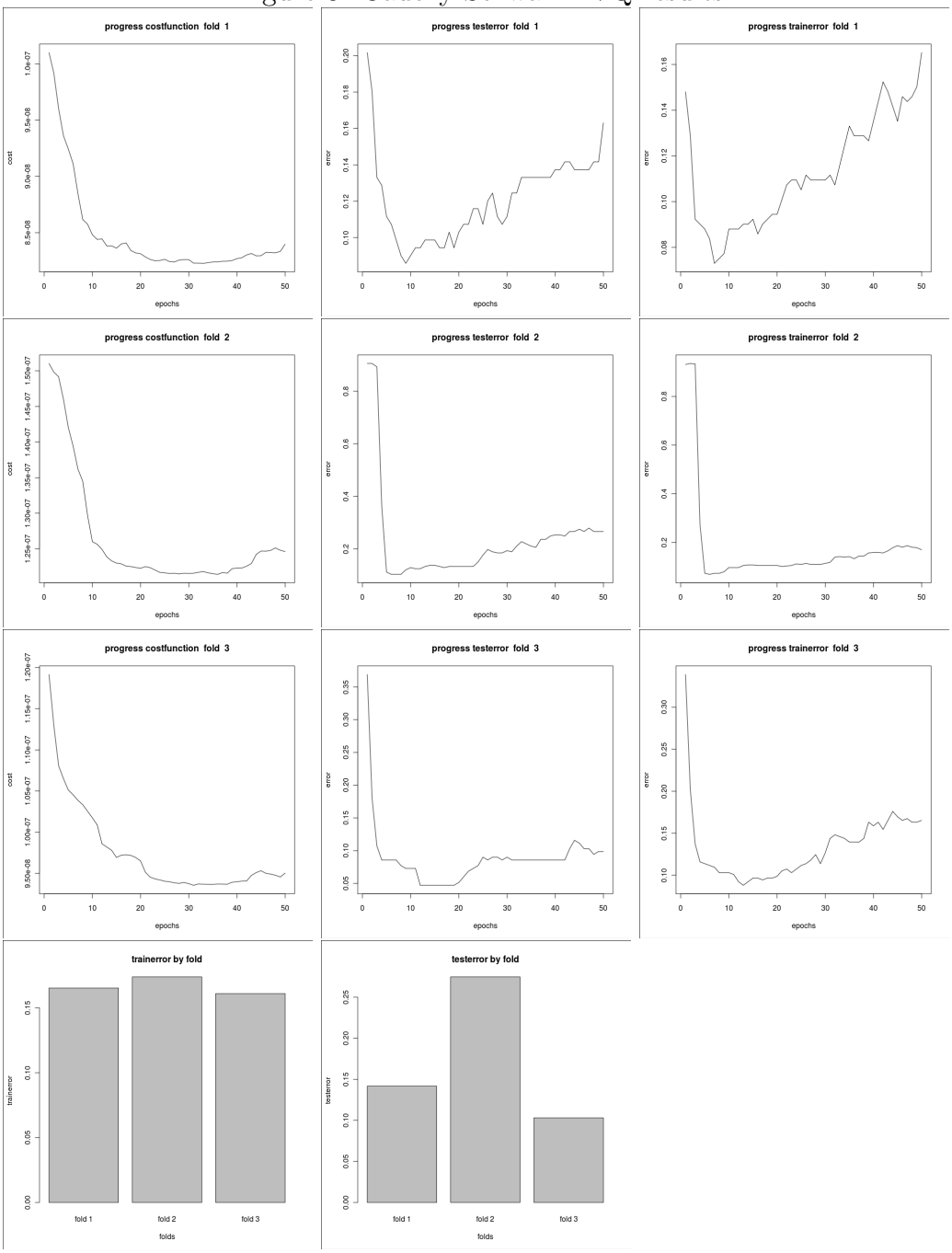
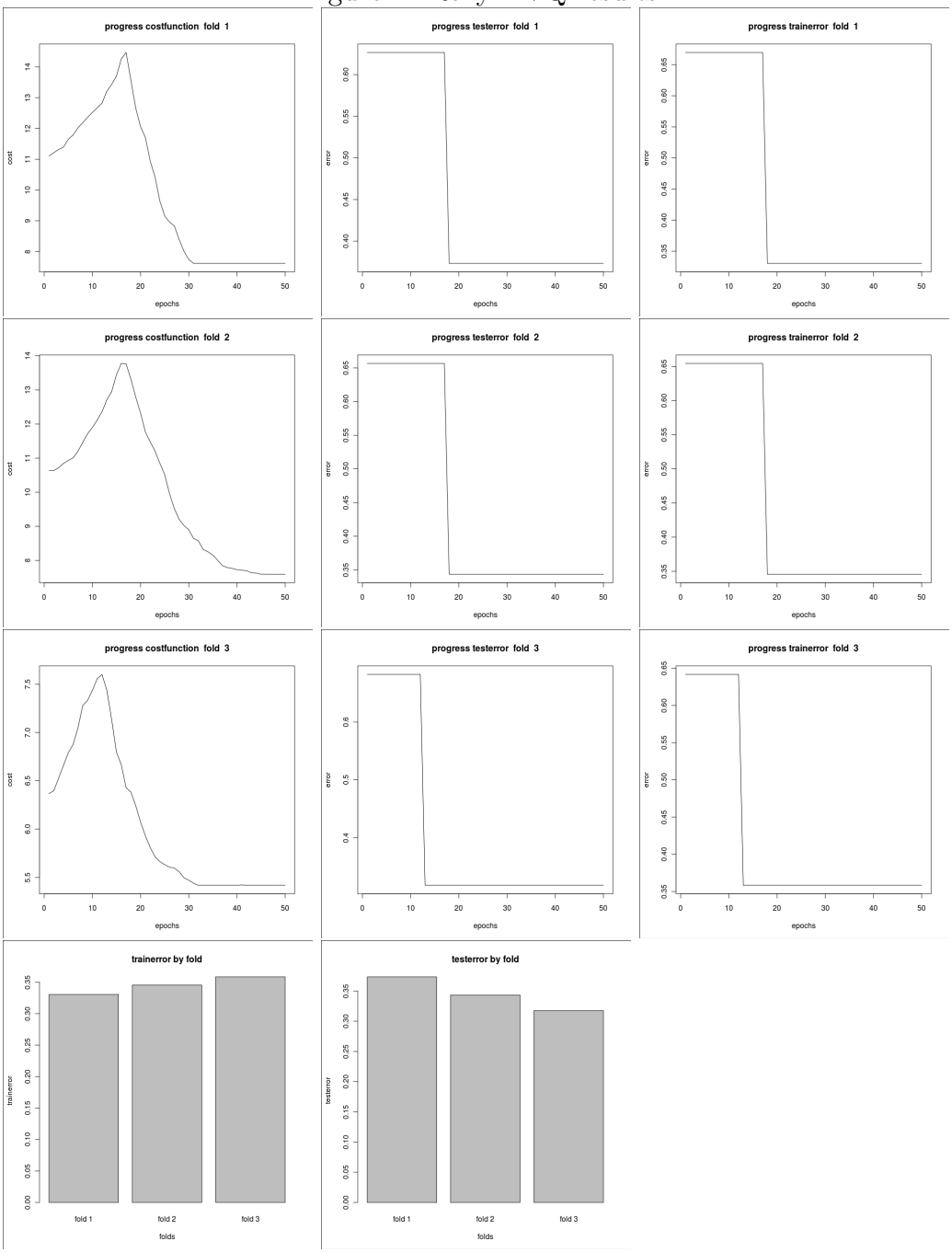


Figure 4: Renyi LVQ results



Conclusion and Future Work

We discussed the bundling of some LVQ algorithms including heuristic and generalized versions of LVQ1, RLVQ, MLVQ and entropy based LVQ in LVQTools. The user can choose the version of LVQ as well as many other settings such as prototype initialization and data set normalization by setting certain parameters. A set of output possibilities and corresponding functions to show and extract the output is provided as well. Results show a reliable decline in train and test errors however depending on data set and LVQ version better results might be obtained by adjusting parameters to fit the specific LVQ run.

In the future this tool might be expanded with more LVQ versions. Since the only real difference in LVQ versions are the distance measure and update rules only these functions have to be added for extensions along with some accounting in calling these functions and error checking. The addition of MLVQ with other than a Euclidean metric would require some more work since it is currently implemented solely for a Euclidean metric. Another improvement might be the visualization of a relevance matrix by a grayscale matrix. These and other improvements can be easily added by adding the functions to the appropriate files and some accounting in calling these functions and error checking.

References

[1] T. Kohonen, *Self-Organizing Maps*, 2nd edition, Berlin, Heidelberg: Springer, 1997.

[2] The R Project for Statistical Computing: http://www.r-project.org/

[3] B. Hammer and T. Villmann, "Generalized relevance learning vector quantization", *Neural Networks*, vol. 15, no. 8-9, pp. 1059-1068, 2002.

[4] A. S. Sato and K. Yamada, Generalized learning vector quantization. In *Advances in Neural Information Processing Systems*, volume 8, pages 423-429, 1996.

[5] P.Schneider, M.Biehl, and B.Hammer. Adaptive relevance matrices in learning vector quantization. Neural Computation December 2009, Vol. 21, No. 12, Pages 3532-3561.

[6] P.Schneider, M.Biehl, and B.Hammer. Relevance matrices in LVQ. In M. Verleysen, editor, *Proc. of European Symposium on Artificial Neural Networks (ESANN)*, pages 37-42, Bruges, Belgium, April 2007.

[7] E. Mwebaze, P. Schneider, F.-M. Schleif, S. Haase, T. Villmann, M. Biehl, Divergence Based Learning Vector Quantization, 18th Europ. Symp. on Artificial Neural Networks, ESANN 2010, M. Verleysen (ed.), d-side publishing, 247-252 (2010)

[8] D.J. Newman, S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases. Irvine, CA: University of California, Department of Information and Computer Science, available at: http://www.ics.uci.edu/ mlearn/MLRepository.html, 1998