A Short Introduction to ENTOOL

C. Merkwirth, J. Wichard

January 14, 2003

Contents

1 Introduction

ENTOOL is a Matlab-toolbox for regression modeling. Aim of this toolbox is to generate and train regression models on data sets consisting of pre-image/image pairs (observations).
This documentation shows the user the usage of the toolbox and gives a guideline, how to add own model classes in the ENTOOL environment. It also gives a more than short introduction to the principles of ensemble learning and references to the literature.
Since regression is a field of statistics going back at least to Gauss, it is not possible to carefully implement any regression technique that has been invented so far, and even less possible to write an in-depth documentation for the used models. To get an overview over the field of machine learning, the reader should have a look into:
Hastie, Tibshirani, Friedman - The Elements of Statistical Learning, Springer 2001.

ENTOOL combines several well know model-classes to build ensembles, that perform much better than the well-trained single models would do by there own.
The implementation of this toolbox is based on the following principles:

- The user should be able to use the toolbox 'out-of-the-box', i.e. with a set of default-parameters to initialize the models and some demos, that show the general usage.
- The user should have direct access to the model-parameters and the initial pool of model-classes that are used for ensembling
- The user should be able to add his own model-classes to the toolbox without any problems

We choose an object oriented design to realize this aims. Every model-type is implemented as a separate class with a clearly defined data interface. A detailed description is given below.

2 Ensemble Models

2.1 Bias and Variance

First of all we have to point out why an ensembles of models performs better than a single model would do. Consider a dataset D with pairs of input-output-data $(\vec{x}_\mu, y_\mu), \mu = 1, \ldots, N$, where $\vec{x}_\mu \in \mathbf{R}^d$ and the y_μ are scalar values. We suppose an underlying functional relationship of the form

$$y = f(\vec{x})_{real} + \epsilon,$$

where the additional error has zero mean and does not depend on $\vec{x}$. The goal of regression is to find a close approximation $f(\vec{x})$ of the assumed deterministic function $f(\vec{x})_{real}$ with additional constrains:

- The modeling error (training error) $E = \sum_{\mu=1}^{N}(y_\mu - f(\vec{x}_\mu))^2$ should be as small as possible
- The generalization ability of the model $f(\vec{x})$ should be as high as possible,i.e. the error on unseen data sets, that were not used for model-training, should also be small

These two constrains mark a crucial problem in regression estimation, know as the bias-variance dilemma (for a detailed description of that problem see for example [3]).
If we want to reduce the training error, we can increase the complexity of the model, in general the number off free model parameters. This can be done, until the training error is zero, but than we expect a high out-of-sample error, because we lost generalization. Figure 2.1 gives an idea of that bias-variance dilemma.

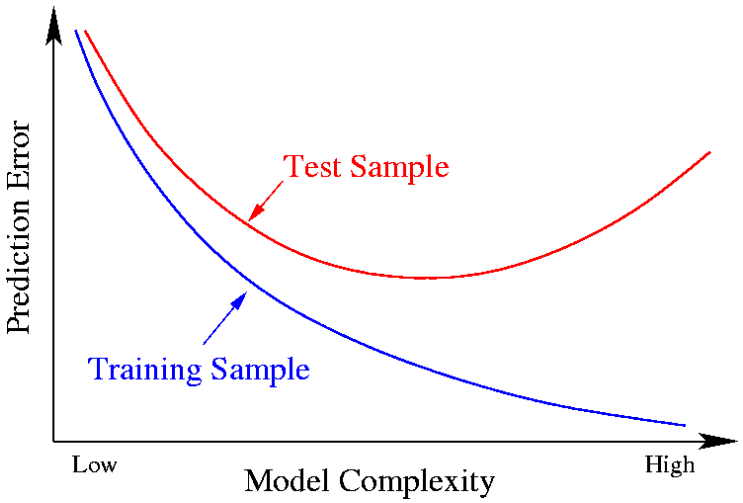

Figure 1: The bias-variance-tradeoff in regression. The training error can be reduced by increasing the complexity of the model. The out-of-sample error is calculated on a test data set that is not used during the training. The test error could be seen as an estimation of the generalization ability of the trained model. It increases at a certain level of complexity, that means the model fits the peculiarities of the training data (i.e. the noise), when the complexity is increased further.

2.2 Ensembles

If we average the output of several different models, we call it an ensemble of models, or simply an ensemble. The idea of averaging different models was developed in the neural network community in the beginning of the 90's [2, 9]. It was pointed out by Krogh et al. [4], that the generalization error of the ensemble is lower than the mean of the generalization error of the single ensemble members.
To see this, we define the weighted ensemble average as

$$\bar{f}(\mathbf{x}) = \sum_{k=1}^{K} w_k f_k(\mathbf{x}), \tag{1}$$

where $f_k(\mathbf{x})$ denotes the k-individual model and the weights $\sum_k w_k = 1$ sum to one.

The generalization error of the ensemble

$$e(\mathbf{x}) \quad = \quad (y(\mathbf{x}) - \bar{f}(\mathbf{x}))^2$$

can now be decomposed in the following manner:

$$e(\mathbf{x}) \quad = \quad \bar{\epsilon}(\mathbf{x}) - \bar{a}(\mathbf{x}), \tag{2}$$

with the two quantities

$$\bar{\epsilon}(\mathbf{x}) \quad = \quad \sum_{k=1}^{K} w_k (y(\mathbf{x}) - f_k(\mathbf{x}))^2 \quad \text{Average error of the individual models}$$

$$\bar{a}(\mathbf{x}) \quad = \quad \sum_{k=1}^{K} w_k (f_k(\mathbf{x}) - \bar{f}(\mathbf{x}))^2 \quad \text{Average ambiguity of the ensemble}$$

If we look at this error decomposition, we can conclude:

- The ensemble generalization error $e(\mathbf{x})$ is always smaller than the expected error of the individual models $\bar{\epsilon}(\mathbf{x})$
- An ensemble should consist of well trained but diverse models in order to increase the ensemble ambiguity

Note, that up to this point, no assumptions about the used models were made.

2.3 Cross Validation

In order to estimate the generalization error and to select models for the final ensemble we use a cross-validation scheme for model training ([3]).
The cross-validation is done in several training rounds on different training sets,

because this increases the ambiguity of the ensemble and leads to better generalization [4]. Another advantage of this method is, that we get an unbiased estimator of the ensemble generalization error and at the same time training the ensemble on the whole data set. This is useful in situations, where only a few data points are available.

The whole procedure consists of the following steps:

- The data is divided in two sets
- Several models are trained on the first set
- The models are compared by evaluating the prediction errors on the unseen data of the test set
- The best performers are taken out and become ensemble members
- The data is divided again in a way that the new test set has minimal overlap with the former ones
- The procedure stops if the ensemble has the desired size

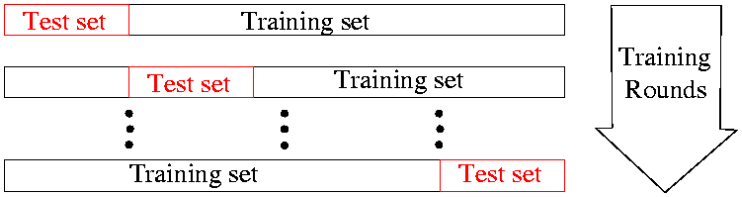

Figure 2: Cross training Scheme

Figure 2.3 illustrates the data partition during the training of the cross-validation ensemble.

2.4 Diversity of Models

The cross-validation mentioned above is one way of introducing model-diversity, because the training on slightly different data sets leads to different models. Krogh et al. have pointed out in their work [5] that the benefit of ensemble learning is high if the ensemble members disagree, which leads to a larger ambiguity term in the error decomposition in equation 2.
In the ENTOOL, we obtain a high ensemble-ambiguity by using different modeling approaches. It seems to be clear, that a (Multi-Layer-Perceptron) MLP has other features than a k-nearest-neighbor model oder than a simple linear regression model.
The current implementation of the ENTOOL has the following model types as Matlab classes:

- Multivariate adaptive regression splines
- Radial basis functions Networks (RBF)
- Linear regression
- Polynomial regression
- K-nearest-neighbor models with adaptive metric
- Multilayer perceptron (MLP) trained with first order gradient decent
- Perceptron trained with a second order gradient decent

3 Implementation and design principles

3.1 Models and model classes

- Every model is an object of some class. The constructor is used to create an empty model of the specified topology (if there's any), and the training methods takes these model and trains it on the given training observations. After that, the model can be used to predict the output of prior unseen training data.
- Every model class is implemented as separate Matlab class, e.g. the nearest-neighbor based regressor can be found as class vicinal, polynomial models as class polynomial etc.
- The command interface to all these classes is basically the same, so it's quite easy to change the type of model used just be changing the line that contains the constructor of that model.
- Example:

  model = vicinal(12);
  model = train(model, X, Y, sampleclass, ...);

  can be changed to:

  model = polynomial(3,4);
  model = train(model, X, Y, sampleclass, ...);

The matrices X and Y are constructed be rearranging the input-output-data $(\vec{x}_\mu, y_\mu), \mu = 1, \ldots, N$ from equation 2.1. The lines of the $(N \times d)$-matrix X are the input-vectors $\vec{x}_\mu$ and the $(N \times 1)$-matrix Y consists of the scalar outputs y_μ

$$X = \begin{pmatrix} \vec{x}_1 \\ \vdots \\ \vec{x}_N \end{pmatrix} \qquad Y = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}.$$

The vector 'sampleclass' has the same size as Y and consists of zeros and ones. The entries describe whether a sample belongs to the trailings set or not.

$$sampleclass = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \quad \text{that means} \quad \begin{pmatrix} (\vec{x}_1, y_1) \text{ belongs to the training set} \\ (\vec{x}_2, y_2) \text{ belongs to the test set} \\ \vdots \\ (\vec{x}_N, y_N) \text{ belongs to the training set} \end{pmatrix}$$

All classes offer at least the following methods:

- get
  - [value] = get(model, param)
  - get the properties 'param' from the ensemble object 'model'
- set
  - [model] = set(model, param, value)
  - set the model properties 'param' to the given 'value'
- train
  - [model, trainerr] = train(model, X, Y, sampleclass, trainparams, eps, varargin)
  - train the ensemble object 'model' with the parameters given in 'trainparams' and the epsilon insensitive error loss 'eps'
- calc
  - [xout] = calc(model, X)
  - calculate the output of the ensemble object 'model' to the given input data in X
- display
  - display(model)
  - display a detailed description of the relevant model parameters, for example the topology and the weights of the MLP, etc.

3.2 The ensemble class

A special class is the class ensemble, which is the parent class for all ensemble generating techniques. Since Matlab does not support real virtual classes, you may construct instances of the ensemble class, but there's no method to train such an object on some data set. An ensemble is a collection of models. To output of these models will be averaged, according to the weights (confidence) that is assigned to each model.

Class crosstrainensemble trains several models of same or different type on varying train/test partitions of the training data, then selects an appropriate subset of these models to increase generalization ability. By default, the models of the ensemble are not weighted.

It possible to train an already trained crosstrainensemle again on the same data set or a data set from the same underlying dependency, in this case it will keep the constituting models and weights, the new models will be added to the old ones.

Objects of child classes of the ensemble class them self are models, so it should be possible to create ensembles of other ensembles etc.
Therefore, one could divide the models in two classes:

1. Primary models, that implement basic regression algorithms
2. Secondary models, that use other models to build ensembles.

3.3 Example

The input data is given in X and the output in Y as described in 3.

Call the constructor for the class of your choice, in this case the cross-training:

```
ens = crosstrainensemble;
```

Get the default training parameter and modify the number of training partitions (enstrainparams.nr_cv_partitions):

```
enstrainparams = get(ens, 'trainparams');
enstrainparams.nr_cv_partitions = 6;
```

Train the ensemble:

```
ens = train(ens, X, Y, [], enstrainparams, 0.05);
```

3.4 Directory and file overview

Ensemble Classes

@cvensemble	-	Class cvregensemble
@crosstrainensemble	-	Class crosstrainensemble
@subspaceensemble	-	Class subspaceensemble
@ensemble	-	Class ensemble (parent of the ensemble classes)

Model Classes

@ares	-	Class ares (Multivariate adaptive regression splines)
@baseline	-	Class baseline (prediction with the mean of the data)
@linear	-	Class linear (linear regression)
@perceptron	-	Class perceptron (multi-layer-perceptron, trained with 1. order gradient decent)
@perceptron2	-	Class perceptron2 (single-layer-perceptron, trained with 2. order gradient decent)
@polynomial	-	Class polynomial (polynomial regression)
@prbfn	-	Class prbfn (radial basis function network)
@vicinal	-	Class vicinal (primary model: k nearest neighbors)
@rbf	-	Class rbf (radial basis functions)

Directories

demos	-	Several demos for this toolbox
documentation	-	Documentation for this toolbox
mextools	-	Mextools, needed for compiling the mex-code in tools
tools	-	Lots of helper m-files, also contains all mex source-code and a makefile (make.m)

Files

startup.m	-	Startup file, does necessary path settings for this toolbox
install.m	-	Installation routine for compiling the mex functions

4 Contributions from other toolboxes

4.1 Neural Network Based System Identification TOOLBOX

The @perceptron2 class is simply an interface to some functions from the "Neural Network Based System Identification TOOLBOX" [7]

by Magnus Nørgaard
Department of Automation
Department of Mathematical Modelling
Technical Report 00-E-891, Department of Automation
Technical University of Denmark

Please see the technical report [7] for further description.

References

[1] Freund: "Short introduction to boosting", J. Japan. Soc. for Artif. Intel. 14(5), pp.771-780, (1999)

[2] Hansen, Salamon: "Neural network ensembles," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 12, no. 10, pp.993 - 1001, (1990).

[3] Hastie, Tibshirani, Friedman: "The Elements of Statistical Learning", Springer (2001)

[4] Krogh, Vedelsby: "Neural Network Ensembles, Cross Validation and Active Learning", Advances in Neural Information Processing Systems 7, MIT Press (1995)

[5] Krogh, Sollich: "Statistical mechanics of ensemble learning", Physical Review E, 55(1), pp.811–825, (1997)

[6] McNames: "Innovations in Local Modeling for Time Series Prediction", Ph.D. Thesis, Stanford University (1999)

[7] Norgaard: "Neural Network Based System Identification Toolbox", Tech. Report. 00-E-891, Department of Automation, Technical University of Denmark (2000)

[8] Orr: "Matlab Functions for Radial Basis Function Networks", (1999)

[9] Perrone, Cooper: "When networks disagree: Ensemble methods for neural networks", Neural Networks for Speech and Image Processing, Chapman Hall (1993)

[10] Suykens, Vandewalle: "Nonlinear Modeling - Advanced Black-Box Techniques", Kluwer Academic Publishers (1998)