Introduction to the ENTOOL Toolbox

C. Merkwirth[1]

J.Wichard[2]

M. Ogorzałek[1]

ChristianMerkwirth@web.de

[1]Department for Information Technologies

Jagiellonian University, Kraków, Poland

[2]FMP, Berlin, Germany

The ENTOOL Toolbox for Statistical Learning

- The ENTOOL toolbox for statistical learning is designed to make state-of-the-art machine learning algorithms available under a common interface
- Allows construction of single models or ensembles of (heterogenous) models
- Supports decorrelation of models by offering resampling techniques
- Though primarily designed for regression, it is possible to construct ensembles of classifiers with ENTOOL

The ENTOOL Toolbox for Statistical Learning

- The ENTOOL toolbox for statistical learning is designed to make state-of-the-art machine learning algorithms available under a common interface
- Allows construction of single models or ensembles of (heterogenous) models
- Supports decorrelation of models by offering resampling techniques
- Though primarily designed for regression, it is possible to construct ensembles of classifiers with ENTOOL

- Requirements:
  - Matlab (TM)
- Operating systems:
  - Windows
  - Linux
  - Solaris (outdated)

Ensemble Methods

Ensemble: Averaging the output of several separately trained models

- Simple average
  $\bar{f}(\vec{x}) = \frac{1}{K} \sum_{k=1}^{K} f_k(\vec{x})$
- Weighted average
  $\bar{f}(\vec{x}) = \sum_k w_k f_k(\vec{x})$ with $\sum_k w_k = 1$

Ensemble Methods

Ensemble: Averaging the output of several separately trained models

- Simple average
  $\bar{f}(\vec{x}) = \frac{1}{K} \sum_{k=1}^{K} f_k(\vec{x})$
- Weighted average
  $\bar{f}(\vec{x}) = \sum_k w_k f_k(\vec{x})$ with $\sum_k w_k = 1$

Error decomposition:

$$
\begin{aligned}
e(\vec{x}) &= (y(\vec{x}) - \bar{f}(\vec{x}))^2 \\
\bar{\epsilon}(\vec{x}) &= \frac{1}{K} \sum_{k=1}^{K} (y(\vec{x}) - f_k(\vec{x}))^2 \\
\bar{a}(\vec{x}) &= \frac{1}{K} \sum_{k=1}^{K} (f_k(\vec{x}) - \bar{f}(\vec{x}))^2 \\
e(\vec{x}) &= \bar{\epsilon}(\vec{x}) - \bar{a}(\vec{x})
\end{aligned}
$$

Ensemble Methods

Ensemble: Averaging the output of several separately trained models

- Simple average
  $\bar{f}(\vec{x}) = \frac{1}{K}\sum_{k=1}^{K} f_k(\vec{x})$
- Weighted average
  $\bar{f}(\vec{x}) = \sum_k w_k f_k(\vec{x})$ with $\sum_k w_k = 1$

Error decomposition:

$$
\begin{aligned}
e(\vec{x}) &= (y(\vec{x}) - \bar{f}(\vec{x}))^2 \\
\bar{\epsilon}(\vec{x}) &= \frac{1}{K}\sum_{k=1}^{K} (y(\vec{x}) - f_k(\vec{x}))^2 \\
\bar{a}(\vec{x}) &= \frac{1}{K}\sum_{k=1}^{K} (f_k(\vec{x}) - \bar{f}(\vec{x}))^2 \\
e(\vec{x}) &= \bar{\epsilon}(\vec{x}) - \bar{a}(\vec{x})
\end{aligned}
$$

Integrating over input space:

$$\mathbf{E} = \bar{\mathbf{E}} - \bar{\mathbf{A}}$$

Ensemble Methods

Ensemble: Averaging the output of several separately trained models

- Simple average
  $\bar{f}(\vec{x}) = \frac{1}{K} \sum_{k=1}^{K} f_k(\vec{x})$
- Weighted average
  $\bar{f}(\vec{x}) = \sum_k w_k f_k(\vec{x})$ with $\sum_k w_k = 1$

Error decomposition:

$$
\begin{aligned}
e(\vec{x}) &= (y(\vec{x}) - \bar{f}(\vec{x}))^2 \\
\bar{\epsilon}(\vec{x}) &= \frac{1}{K} \sum_{k=1}^{K} (y(\vec{x}) - f_k(\vec{x}))^2 \\
\bar{a}(\vec{x}) &= \frac{1}{K} \sum_{k=1}^{K} (f_k(\vec{x}) - \bar{f}(\vec{x}))^2 \\
e(\vec{x}) &= \bar{\epsilon}(\vec{x}) - \bar{a}(\vec{x})
\end{aligned}
$$

Integrating over input space:

$$\mathbf{E} = \bar{\mathbf{E}} - \bar{\mathbf{A}}$$

Ensemble generalization error is always smaller than that of the individual models

Ensemble Methods

Ensemble: Averaging the output of several separately trained models

- Simple average
  $\bar{f}(\vec{x}) = \frac{1}{K} \sum_{k=1}^{K} f_k(\vec{x})$
- Weighted average
  $\bar{f}(\vec{x}) = \sum_k w_k f_k(\vec{x})$ with $\sum_k w_k = 1$

Interpretation:

- Ensemble should consist of well trained but diverse models
- Often ensemble outperforms best constituting model
- Classification also benefits from ensembling

Error decomposition:

$$
\begin{aligned}
e(\vec{x}) &= (y(\vec{x}) - \bar{f}(\vec{x}))^2 \\
\bar{\epsilon}(\vec{x}) &= \frac{1}{K} \sum_{k=1}^{K} (y(\vec{x}) - f_k(\vec{x}))^2 \\
\bar{a}(\vec{x}) &= \frac{1}{K} \sum_{k=1}^{K} (f_k(\vec{x}) - \bar{f}(\vec{x}))^2
\end{aligned}
$$

$$
e(\vec{x}) = \bar{\epsilon}(\vec{x}) - \bar{a}(\vec{x})
$$

Integrating over input space:

$$
\mathbf{E} = \bar{\mathbf{E}} - \bar{\mathbf{A}}
$$

Ensemble generalization error is always smaller than that of the individual models

Crosstraining – Constructing Ensembles

- Finesse: Efficiently reuse samples by combining training, validation and selection of models
- Additional benefit of reduced correlation between models
- Repeatedly partition data set randomly into two sample classes
  - Training set, used for training and stopping criteria
  - Test set, used only for accessing generalization error after model has been trained

Crosstraining – Constructing Ensembles

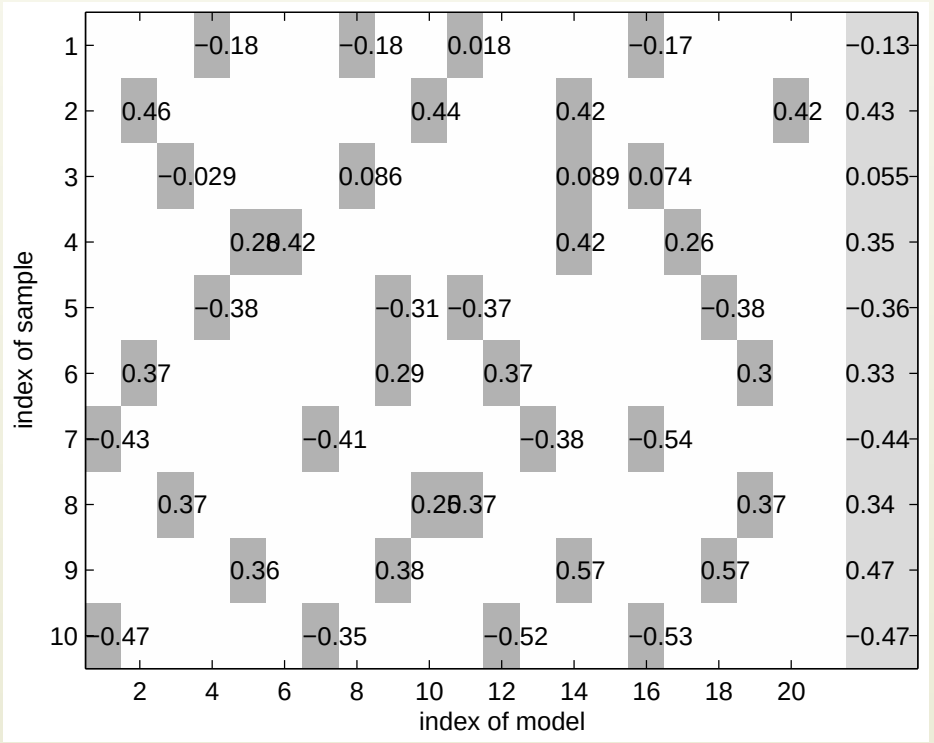
- Finesse: Efficiently reuse samples by combining training, validation and selection of models
- Additional benefit of reduced correlation between models
- Repeatedly partition data set randomly into two sample classes
  - Training set, used for training and stopping criteria
  - Test set, used only for accessing generalization error after model has been trained
- Train population of (heterogenous) models, select best ones according to error on test set
- Repartition data set, taking care that test sets are mutually disjunct
- Combine best models of all partitionings to ensemble
- Optionally weight models according to the estimated generalization error on the total data set

Out-of-Train – CV for Ensembles

- Out-of-Train Calculation is combination of Cross Validation and Ensemble Averaging
- Inspired by Breiman's Out-Of-Bag technique
- Construct ensemble of models on numerous train/test partitionings
- Don't use test samples of each partition for model selection etc.
- OOT output for one sample of data set is average of all models where this sample was not in training set (out-of-train)

Out-of-Train – CV for Ensembles

- Out-of-Train Calculation is combination of Cross Validation and Ensemble Averaging
- Inspired by Breiman's Out-Of-Bag technique
- Construct ensemble of models on numerous train/test partitionings
- Don't use test samples of each partition for model selection etc.
- OOT output for one sample of data set is average of all models where this sample was not in training set (out-of-train)



- Similar to traditional CV
- Accounts for ensemble gain

Importance of Input Variables

- Motivation: Determine variable importance with respect to prediction accuracy
- Might help uncovering causal relationships of underlying process
- Problem: Ensemble of heterogenous (nonlinear) models is even more difficult to analyze than single models
- Idea: Combine surrogate data method with OOT calculation

Importance of Input Variables

- Motivation: Determine variable importance with respect to prediction accuracy
- Might help uncovering causal relationships of underlying process
- Problem: Ensemble of heterogenous (nonlinear) models is even more difficult to analyze than single models
- Idea: Combine surrogate data method with OOT calculation

- To determine importance of x_i:
  - Create surrogate/replicate of the original input data set where values of x_i are randomly permuted to destroy information content
  - Calculate OOT output for surrogate data set
  - Compare errors of OOT output of surrogate and original data set
  - If OOT error increases significantly, x_i is important!
  - Average importance over several surrogate data sets for same variable to smooth out noise

Importance of Input Variables

- Motivation: Determine variable importance with respect to prediction accuracy
- Might help uncovering causal relationships of underlying process
- Problem: Ensemble of heterogenous (nonlinear) models is even more difficult to analyze than single models
- Idea: Combine surrogate data method with OOT calculation
- No retraining of the ensemble necessary
- Uncovers linear and nonlinear relationships

- To determine importance of x_i:
  - Create surrogate/replicate of the original input data set where values of x_i are randomly permuted to destroy information content
  - Calculate OOT output for surrogate data set
  - Compare errors of OOT output of surrogate and original data set
  - If OOT error increases significantly, x_i is important!
  - Average importance over several surrogate data sets for same variable to smooth out noise

ENTOOL Software Architecture

- Each model type is implemented as separate class
- All model classes share common interface
- Exchange model types by exchanging constructor call
- Automatic generation of ensembles of models
- Models are divided into two brands:
  1. Primary models like linear models, neural networks, SVMs etc.
  2. Secondary models that rely on primary models to calculate output. All ensemble models are secondary models.

ENTOOL Software Architecture

- Each model type is implemented as separate class
- All model classes share common interface
- Exchange model types by exchanging constructor call
- Automatic generation of ensembles of models
- Models are divided into two brands:
  1. Primary models like linear models, neural networks, SVMs etc.
  2. Secondary models that rely on primary models to calculate output. All ensemble models are secondary models.

- Lifecycle of a model can be divided into three phases:
  1. During construction, topology of the model is specified. The model can't be used yet.
  2. Model has now to be trained on some training data set $(\vec{x}_i, y_i)$
  3. After training, the model can be evaluated on new/unseen inputs $(\vec{x}_n)$
- Constructors should assign random default topologies in order to create uncorrelated models
- It is possible to construct ensembles of ensembles

Syntax

- Generating an example data set:
  `x = randn(500, 4);` generate a data set with 500 samples of dim. 4
  `x_new = randn(200, 4);` generate a test set with 200 samples
  `y = x(:,1) - 4.5 * x(:,3) .* x(:,4);` generate scalar output values

Syntax

- Generating an example data set:
  `x = randn(500, 4);` generate a data set with 500 samples of dim. 4
  `x_new = randn(200, 4);` generate a test set with 200 samples
  `y = x(:,1) - 4.5 * x(:,3) .* x(:,4);` generate scalar output values
- Turning progress visualization on/off:
  `global entooldrawit` this variable controls the visualization
  `entooldrawit = 1;` 1/0 turns visualization on/off

Syntax

- Generating an example data set:
  `x = randn(500, 4);` generate a data set with 500 samples of dim. 4
  `x_new = randn(200, 4);` generate a test set with 200 samples
  `y = x(:,1) - 4.5 * x(:,3) .* x(:,4);` generate scalar output values
- Turning progress visualization on/off:
  `global entooldrawit` this variable controls the visualization
  `entooldrawit = 1;` 1/0 turns visualization on/off
- Constructor syntax:
  `model = perceptron3;` will create a MLP model with default topology
  `model = perceptron3(12);` MLP model with 12 hidden layer neurons
  `model = ridge;` will create a ridge regression model with penalty $\lambda = 0$
  `model = ridge(0:10);` will create a ridge regression model with automatic selection of optimal penalty λ out of $0, 1, \ldots, 10$

Syntax

- Generating an example data set:
  `x = randn(500, 4);` generate a data set with 500 samples of dim. 4
  `x_new = randn(200, 4);` generate a test set with 200 samples
  `y = x(:,1) - 4.5 * x(:,3) .* x(:,4);` generate scalar output values
- Turning progress visualization on/off:
  `global entooldrawit` this variable controls the visualization
  `entooldrawit = 1;` 1/0 turns visualization on/off
- Constructor syntax:
  `model = perceptron3;` will create a MLP model with default topology
  `model = perceptron3(12);` MLP model with 12 hidden layer neurons
  `model = ridge;` will create a ridge regression model with penalty $\lambda = 0$
  `model = ridge(0:10);` will create a ridge regression model with automatic selection of optimal penalty λ out of $0, 1, \ldots, 10$
- Training syntax:
  `model = train(model, x, y, [], [], 0.05);`
  trains model with ϵ-insensitive loss of 0.05 on data set $(\vec{x}_i, y_i)$

Syntax cont.

- Evaluation syntax:
  `y_new = calc(model, x_new)` evaluates the model on new inputs

Syntax cont.

- Evaluation syntax:
  `y_new = calc(model, x_new)` evaluates the model on new inputs
- How to build a crosstraining ensemble of models:
  `ens = crosstrainensemble;` will create an empty ensemble object
  `ens = train(ens, x, y, [], [], 0.05);` calls training routines for several primary models and joins them into ensemble object

Syntax cont.

- Evaluation syntax:
  `y_new = calc(model, x_new)` evaluates the model on new inputs
- How to build a crosstraining ensemble of models:
  `ens = crosstrainensemble;` will create an empty ensemble object
  `ens = train(ens, x, y, [], [], 0.05);` calls training routines for several primary models and joins them into ensemble object
- Ensemble evaluation:
  `y_new = calc(ens, x_new)` evaluates the ensemble on new inputs

Syntax cont.

- Evaluation syntax:
  `y_new = calc(model, x_new)` evaluates the model on new inputs
- How to build a crosstraining ensemble of models:
  `ens = crosstrainensemble;` will create an empty ensemble object
  `ens = train(ens, x, y, [], [], 0.05);` calls training routines for several primary models and joins them into ensemble object
- Ensemble evaluation:
  `y_new = calc(ens, x_new)` evaluates the ensemble on new inputs
- How to build a crossvalidation ensemble of models:
  `ens = cvensemble;` will create an empty ensemble object
  `ens = train(ens, x, y, [], [], 0.05);` calls training routines for several primary models and joins them into ensemble object

Syntax cont.

- Evaluation syntax:
  `y_new = calc(model, x_new)` evaluates the model on new inputs
- How to build a crosstraining ensemble of models:
  `ens = crosstrainensemble;` will create an empty ensemble object
  `ens = train(ens, x, y, [], [], 0.05);` calls training routines for several primary models and joins them into ensemble object
- Ensemble evaluation:
  `y_new = calc(ens, x_new)` evaluates the ensemble on new inputs
- How to build a crossvalidation ensemble of models:
  `ens = cvensemble;` will create an empty ensemble object
  `ens = train(ens, x, y, [], [], 0.05);` calls training routines for several primary models and joins them into ensemble object
- How to determine the OOT error on the training data set:
  `yh = outoftraincalc(ens)` evaluates the ensemble on training set
  `regression_error(y, yh)` display the OOT error

Adjusting class specific training parameters

- 5th argument when calling `train` specifies training parameters

Adjusting class specific training parameters

- 5th argument when calling `train` specifies training parameters
- Except topology, often training parameters have to be specified:

```
tp = get(perceptron3, 'trainparams')
maxiter:  250
errormode:  0
weightdecay:  1.0000e-006
dont_scale_outputs:  0
initweights:  0.3000
initialstepsize:  1.0000e-003
minstepsize:  1.0000e-006
maxstepsize:  0.0500
...
```

Adjusting class specific training parameters

- 5th argument when calling `train` specifies training parameters
- Except topology, often training parameters have to be specified:

```
tp = get(perceptron3, 'trainparams')
maxiter:  250
errormode:  0
weightdecay:  1.0000e-006
dont_scale_outputs:  0
initweights:  0.3000
initialstepsize:  1.0000e-003
minstepsize:  1.0000e-006
maxstepsize:  0.0500
...
```

- Assign new value: `tp.maxiter = 400`

Adjusting class specific training parameters

- 5th argument when calling `train` specifies training parameters
- Except topology, often training parameters have to be specified:

```
tp = get(perceptron3, 'trainparams')
maxiter:  250
errormode:  0
weightdecay:  1.0000e-006
dont_scale_outputs:  0
initweights:  0.3000
initialstepsize:  1.0000e-003
minstepsize:  1.0000e-006
maxstepsize:  0.0500
...
```

- Assign new value: `tp.maxiter = 400`
- And give training parameters while training:

```
model = train(perceptron3, x, y, [], tp, 0.05);
```

Specifying which Model Types to Ensemble

- Ensemble constructor will train several models on dataset:

```
tp = get(crosstrainensemble, 'trainparams')
nr_cv_partitions:  8
frac_test:  0.2000
minimum_testsamples:  5
remove_worst:  0.3300
use_models:  0.8000
weight_models:  0
modelclasses:  6x3 cell
scaledata:  1
```

Specifying which Model Types to Ensemble

- Ensemble constructor will train several models on dataset:

```
tp = get(crosstrainensemble, 'trainparams')
nr_cv_partitions:  8
frac_test:  0.2000
minimum_testsamples:  5
remove_worst:  0.3300
use_models:  0.8000
weight_models:  0
modelclasses:  6x3 cell
scaledata:  1
```

- Remove ridge models from the list of model classes:

```
tp.modelclasses = {'perceptron3', [], {}; ...
{'vicinal2', [], {}}
```

Specifying which Model Types to Ensemble

- Ensemble constructor will train several models on dataset:

```
tp = get(crosstrainensemble, 'trainparams')
nr_cv_partitions:  8
frac_test:  0.2000
minimum_testsamples:  5
remove_worst:  0.3300
use_models:  0.8000
weight_models:  0
modelclasses:  6x3 cell
scaledata:  1
```

- Remove ridge models from the list of model classes:

```
tp.modelclasses = {'perceptron3', [], {}; ...
{'vicinal2', [], {}}
```

- And give training parameters while training:

```
ens = train(crosstrainensemble, x, y, [], tp, 0.05);
```

Primary Models Types

ridge Linear model trained by ridge regression

perceptron3 Multilayer perceptron with iRPROP+ training

vicinal2 k-nearest-neighbor regression with adaptive metric

Ensemble Classes

ensemble Virtual parent class for all ensemble classes

crosstrainensemble Ensemble class that trains models according to crosstraining scheme. Creates ensembles of decorrelated models.

cvensemble Ensemble class that trains models according to crossvalidation/out-of-training scheme. Can be used to access OOT error.

subspaceensemble Creates an ensemble of models where each single model is trained on a random subset of the input variables

Literature

- Krogh, Vedelsby Neural Network Ensembles, Cross Validation and Active Learning Advances in Neural Information Processing Systems 7, MIT Press 1995
- Peronne, Cooper When networks disagree: Ensemble methods for neural networks Neural Networks for Speech and Image Processing, Chapman Hall 1993
- Hastie, Tibshirani, Friedman The Elements of Statistical Learning Springer 2001
- Vapnik The Nature of Statistical Learning Theory Springer 1999
- Chih-Chung Chang, Chih-Jen Lin LIBSVM : a library for support vector machines Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`, 2001
- Freund Short introduction to boosting 1999
- Domingos A unified bias-variance decomposition for zero-one and squared loss 2000
- Merkwirth et al Ensemble Methods for Classification in Cheminformatics 2004
- Nørgaard Neural Network Based System Identification Toolbox, Tech. Report. 00-E-891, Department of Automation, Technical University of Denmark 2000
- Orr Matlab Functions for Radial Basis Function Networks 1999
- Cohen, Intrator A Hybrid Projection Based and Radial Basis Function Architecture, Lecture Notes in Computer Science, 2000