Machine Learning 2

5. Ensembles

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Computer Science
University of Hildesheim, Germany

Syllabus

		A. Advanced Supervised Learning
Tue. 5.4.	(1)	A.1 Generalized Linear Models
Tue. 12.4.	(2)	A.2 Gaussian Processes
Tue. 19.4.	(3)	A.2b Gaussian Processes (ctd.)
Tue. 26.4.	(4)	A.3 Advanced Support Vector Machines
Tue. 3.5.	(5)	A.4 Neural Networks
Tue. 10.5.	(6)	A.5 Ensembles
Tue. 17.5.	—	— Pentecoste Break —
Tue. 24.5.	(7)	A.5b Ensembles (ctd.)
Tue. 31.5.	(8)	A.6 Sparse Linear Models — L1 regularization
Tue. 7.6.	(9)	A.6b Sparse Linear Models — L1 regularization (ctd.)
Tue. 14.6.	(10)	A.7. Sparse Linear Models — Further Methods
		B. Complex Predictors
Tue. 21.6.	(11)	B.1 Latent Dirichlet Allocation (LDA)
Tue. 28.6.	(12)	B.2 Deep Learning
Tue. 5.7.	(13)	Questions and Answers

Outline

Outline

Model Selection

If we have **several models**

$$\hat{y}_c : \mathbb{R}^M \rightarrow \mathcal{Y}, \quad c = 1, \dots, C$$

for the same task, so far we tried to **select the best one**

$$\begin{aligned} \hat{y} :=& \hat{y}_{c^*} \quad \text{with} \\ c^* :=& \operatorname*{arg\,min}_{c \in \{1,\dots,C\}} \ell(\hat{y}_c, \mathcal{D}^{\text{val}}) \end{aligned}$$

using validation data $\mathcal{D}^{\text{val}}$ and deploy it (**model selection**).

Model Averaging & Voting

Alternatively, having **several models**

$$\hat{y}_c : \mathbb{R}^M \to \mathcal{Y}, \quad c = 1, \ldots, C$$

one also can **combine them** (**model combination**, **ensemble**), e.g.,

model averaging, for continuous outputs
(regression, classification with uncertainty):

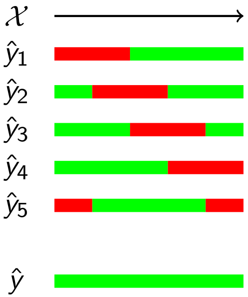
$$\hat{y}(x) := \frac{1}{C} \sum_{c=1}^{C} \hat{y}_c(x)$$

voting, for nominal outputs
(classification without uncertainty):

$$\hat{y}(x) := y^* \text{ with } n_{y^*}(x) \text{ maximal among all } n_y(x)$$
$$n_y(x) := |\{c \in \{1, \ldots, C\} \mid \hat{y}_c(x) = y\}|$$

Why Ensembles ?

- an ensemble usually improves accuracy
  - if component models make different types of errors



Weighted Model Averaging I: Bayesian Model Averaging

$$\hat{y}(x) := \sum_{c=1}^{C} \alpha_c \hat{y}_c(x)$$

with **component model weights** $\alpha \in \mathbb{R}^C$.

Bayesian Model Averaging:

$$p(y \mid x) := \int_{\mathcal{M}} p(y \mid x, m, \mathcal{D})\, p(m \mid \mathcal{D}) dm$$
$$\overset{\text{MC}}{\approx} \sum_{c=1}^{C} p(y \mid x, m_c, \mathcal{D})\; p(m_c \mid \mathcal{D})$$

Weighted Model Averaging I: Bayesian Model Averaging

$$\hat{y}(x) := \sum_{c=1}^{C} \alpha_c \hat{y}_c(x)$$

with **component model weights** $\alpha \in \mathbb{R}^C$.

Bayesian Model Averaging:

$$\begin{aligned} p(y \mid x) &:= \int_{\mathcal{M}} p(y \mid x, m, \mathcal{D})\, p(m \mid \mathcal{D}) dm \\ &\overset{\text{MC}}{\approx} \sum_{c=1}^{C} \underbrace{p(y \mid x, m_c, \mathcal{D})}_{=\hat{y}_c(x)} \underbrace{p(m_c \mid \mathcal{D})}_{=\alpha_c} \end{aligned}$$

Weighted Model Averaging II: Linear Stacking

$$\hat{y}(x) := \sum_{c=1}^{C} \alpha_c \hat{y}_c(x)$$

with **component model weights** $\alpha \in \mathbb{R}^C$.

Linear Stacking:

- learn α's minimizing the loss on validation data:

$$\alpha := \arg\min_{\alpha} \ell(\sum_{c=1}^{C} \alpha_c \hat{y}_c(x), \mathcal{D}^{\text{val}})$$

- actually a Generalized Linear Model with C features

$$x'_c(x) := \hat{y}_c(x), \quad c = 1, \ldots, C$$

  and parameters α.

(General) Stacking

- Build the **second stage dataset**:

  $$\mathcal{D}^{\text{val}}_{\text{2nd stage}} := \{(x', y) \mid x'_c := \hat{y}_c(x), c = 1, \ldots, C, (x, y) \in \mathcal{D}^{\text{val}}\} \subseteq \mathcal{Y}^C \times$$

- Learn a **second stage prediction model** for the 2nd stage data set

  $$\hat{y}_{\text{2nd stage}} : \mathcal{Y}^C \to \mathcal{Y}$$

  - e.g., a linear model/GLM, a SVM/SVR, a neural network etc.

(General) Stacking

- Build the **second stage dataset**:

  $\mathcal{D}^{\text{val}}_{\text{2nd stage}} := \{(x', y) \mid x'_c := \hat{y}_c(x), c = 1, \ldots, C, (x, y) \in \mathcal{D}^{\text{val}}\} \subseteq \mathcal{Y}^C \times$

- Learn a **second stage prediction model** for the 2nd stage data set

$$\hat{y}_{\text{2nd stage}} : \mathcal{Y}^C \to \mathcal{Y}$$

  - e.g., a linear model/GLM, a SVM/SVR, a neural network etc.

- to predict a new instance x,
  - first, compute the predictions of the (1st stage) component models

$$x'_c := \hat{y}_c(x), \quad c = 1, \ldots, C$$

  - then compute the final prediction of the 2nd stage model:

$$\hat{y}(x) := \hat{y}_{\text{2nd stage}}(x'_1, \ldots, x'_C)$$

- non-linear second stage models can capture interactions between the different component models.

Origins of Model Heterogeneity

Model heterogeneity can stem from different roots:

- different model families
  - e.g., GLMs, SVMs, NNs etc.
  - used to win most challenges, e.g., Netflix challenge
- different hyperparameters (for the same model family)
  - e.g., regularization weights, kernels, number of nodes/layers etc.
- different variables used
  - e.g., **Random Forests**
- trained on different subsets of the dataset
  - **Bagging**

Bootstrap Aggregation (Bagging)

- **bootstrap** is a resampling method
  - sample with replacement uniformly from the original sample $\mathcal{D}^{\text{train}}$
  - as many instances as the original sample contains
  - in effect, some instances may be missing in the resample, others may occur twice or even more frequently
- draw C bootstrap samples from $\mathcal{D}^{\text{train}}$:

$$\mathcal{D}_c^{\text{train}} \sim \text{bootstrap}(\mathcal{D}^{\text{train}}), \quad c = 1, \ldots, C$$

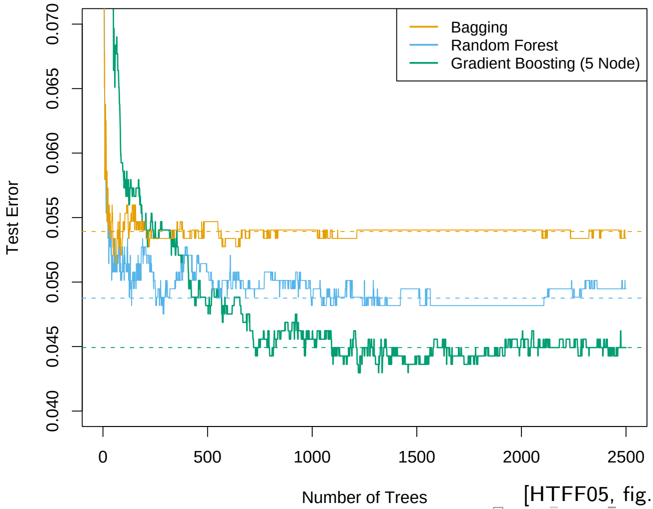
- train a model $\hat{y}_c$ for each of these datasets $\mathcal{D}_c^{\text{train}}$.
- average these models:

$$\hat{y}(x) := \frac{1}{C} \sum_{c=1}^{C} \hat{y}_c(x)$$

Random Forests

- bagging often creates datasets that are too similar to each other
  - consequently, models correlate heavily and ensembling does not work well
- to decorrelate the component models, one can train them on different subsets of variables
- **Random Forests**
  - use decision trees as component models
    - binary splits
    - regularized by minimum node size (e.g., 1, 5 etc.)
    - no pruning
    - sometimes using just **decision tree stumps** (= a single split)
  - trained on bootstrap samples
  - using only a random subset of variables
    - actually, using a random subset of variables for each single split.
    - e.g., $\lfloor\sqrt{m}\rfloor$, $\lfloor m/3 \rfloor$.
  - finally model averaging/voting the decision trees

Bagging & Random Forests / Example (spam data)



[HTFF05, fig. 15.1]

Outline

Consecutive vs Joint Ensemble Learning

So far, ensembles have been constructed in two **consecutive steps**:

- 1st step: create heterogeneous models
  - learn model parameters for each model separately
- 2nd step: combine them
  - learn combination weights (stacking)

Consecutive vs Joint Ensemble Learning

So far, ensembles have been constructed in two **consecutive steps**:

- 1st step: create heterogeneous models
  - learn model parameters for each model separately
- 2nd step: combine them
  - learn combination weights (stacking)

Advantages:

- simple
- trivial to parallelize

Disadvantages:

- models are learnt in isolation

Consecutive vs Joint Ensemble Learning

So far, ensembles have been constructed in two **consecutive steps**:

- 1st step: create heterogeneous models
  - learn model parameters for each model separately
- 2nd step: combine them
  - learn combination weights (stacking)

Advantages:

- simple
- trivial to parallelize

Disadvantages:

- models are learnt in isolation

New idea: **Learn model parameters and combination weights jointly**

$$\ell(\mathcal{D}^{\text{train}}; \Theta) := \sum_{n=1}^{N} \ell(y_n, \sum_{c=1}^{C} \alpha_c \hat{y}(x_n; \theta_c)), \quad \Theta := (\alpha, \theta_1, \ldots, \theta_C)$$

Boosting

Idea: fit models (and their combination weights)

- sequentially, one at a time,
- relative to the ones already fitted,
- but do not consider to change the earlier ones again.

Boosting

Idea: fit models (and their combination weights)

- sequentially, one at a time,
- relative to the ones already fitted,
- but do not consider to change the earlier ones again.

$$
\begin{aligned}
y^{(C')}(x) :=& \sum_{c=1}^{C'} \alpha_c \hat{y}(x;\theta_c), \quad C' \in \{1,\ldots,C'\}\\
=& \hat{y}^{(C'-1)}(x) + \alpha_{C'} \hat{y}(x;\theta_{C'})
\end{aligned}
$$

$$
\ell(\mathcal{D}^{\text{train}}, \hat{y}^{(C')}) = \sum_{n=1}^{N} \ell(y_n, \hat{y}^{(C')}(x_n))
$$

$$
(\alpha_{C'}, \theta_{C'}) := \underset{\alpha_{C'},\theta_{C'}}{\arg\min} \sum_{n=1}^{N} \ell(y_n,\, \hat{y}^{(C'-1)}(x_n) + \alpha_{C'}\, \hat{y}(x_n;\theta_{C'}))
$$

Boosting

Idea: fit models (and their combination weights)

- sequentially, one at a time,
- relative to the ones already fitted,
- but do not consider to change the earlier ones again.

$$y^{(C')}(x) := \sum_{c=1}^{C'} \alpha_c \hat{y}(x;\theta_c), \quad C' \in \{1,\ldots,C'\}$$
$$= \hat{y}^{(C'-1)}(x) + \alpha_{C'}\hat{y}(x;\theta_{C'})$$
$$\ell(\mathcal{D}^{\text{train}}, \hat{y}^{(C')}) = \sum_{n=1}^{N} \ell(y_n, \hat{y}^{(C')}(x_n))$$
$$(\alpha_{C'},\theta_{C'}) := \arg\min_{\alpha_{C'},\theta_{C'}} \sum_{n=1}^{N} \ell(y_n, \underbrace{\hat{y}^{(C'-1)}(x_n)}_{=:\hat{y}_n^0} + \alpha_{C'}\underbrace{\hat{y}(x_n;\theta_{C'})}_{=:\alpha\hat{y}_n})$$

Convergence & Shrinking

Models are fitted iteratively

$$C' := 1, 2, 3, \ldots$$

- convergence is assessed via **early stopping**: once the error on a validation sample

$$\ell(\mathcal{D}^{\text{val}}, \hat{y}^{(C')})$$

  does not decrease anymore over a couple of iterations, the algorithm stops and returns the best iteration so far.

- To deaccelerate convergence to the training data, usually **shrinking the combination weights** is applied:

$$\alpha_{C'} := \nu\,\alpha_{C'}, \quad \text{e.g., with } \nu = 0.02$$

L2 Loss Boosting (Least Squares Boosting)

For L2 loss

$$\ell(y, \hat{y}) := (y - \hat{y})^2$$

we get

$$\ell(y_n, \hat{y}_n^0 + \alpha \hat{y}_n) = \ell(y_n - \hat{y}_n^0, \alpha \hat{y}_n)$$

and thus **fit the residuals**

$$\theta_{C'} := \arg\min_{\theta_{C'}} \sum_{n=1}^{N} \ell(y_n - \hat{y}_n^0, \hat{y}(x_n; \theta_{C'}))$$

$$\alpha_{C'} := 1$$

Exponential Loss Boosting (AdaBoost)

For (weighted) exponential loss

$$\ell(y, \hat{y}, w) := w\, e^{-y\hat{y}}, \quad y \in \{-1, +1\}, \hat{y} \in \mathbb{R}$$

we get

$$\ell(y_n, \hat{y}_n^0 + \alpha\hat{y}_n, w_n^0) = \ell(y_n, \hat{y}_n^0, w_n^0)\, \ell(y_n, \alpha\hat{y}_n, 1)$$

Exponential Loss Boosting (AdaBoost)

For (weighted) exponential loss

$$\ell(y, \hat{y}, w) := w\, e^{-y\hat{y}}, \quad y \in \{-1, +1\}, \hat{y} \in \mathbb{R}$$

we get

$$\begin{aligned}\ell(y_n, \hat{y}_n^0 + \alpha\hat{y}_n, w_n^0) &= \underbrace{\ell(y_n, \hat{y}_n^0, w_n^0)}_{=: w_n} \ell(y_n, \alpha\hat{y}_n, 1)\\ &= \ell(y_n, \alpha\hat{y}_n, w_n)\end{aligned}$$

Exponential Loss Boosting (AdaBoost)

The loss in iteration C'

$$\arg\min_{\alpha,\hat{y}_n} \sum_{n=1}^{N} \ell(y_n, \alpha\hat{y}_n, w_n) = \arg\min_{\alpha_{C'},\theta_{C'}} \sum_{n=1}^{N} \ell(y_n, \alpha_{C'}\hat{y}(x_n,\theta_{C'}), w_n^{(C')})$$

is minimized sequentially:

1. Learn $\theta_{C'}$:

$$w_n^{(C')} := \ell(y_n, \hat{y}^{(C'-1)}(x_n), w_n^{(C'-1)})$$

$$\hat{\theta}_{C'} := \arg\min_{\theta_{C'}} \sum_{n=1}^{N} \ell(y_n, \hat{y}(x_n,\theta_{C'}), w_n^{(C')})$$

2. Learn $\alpha_{C'}$:

$$\mathrm{err}_{C'} := \frac{\sum_{n=1}^{N} w_n^{(C')} \delta(y_n \neq \hat{y}(x_n,\theta_{C'}))}{\sum_{n=1}^{N} w_n^{(C')}}$$

$$\alpha_{C'} := \frac{1}{2}\log\frac{1-\mathrm{err}_{C'}}{\mathrm{err}_{C'}}$$

AdaBoost

```
1: procedure ADABOOST($\mathcal{D}^{\text{train}} = \{(x_1, y_1), \ldots, (x_N, y_N)\}, C$)
2:     $w_n := \frac{1}{N}, \quad n := 1, \ldots, N$
3:     for $c := 1, \ldots, C$ do
4:         fit a classifier to data with case weights $w$:
5:             $\theta_c := \arg\min_\theta \ell(\mathcal{D}^{\text{train}}, \hat{y}(\theta), w)$
6:         $\text{err}_C := \frac{\sum_{n=1}^N w_n \delta(y_n \neq \hat{y}(x_n, \theta_c))}{\sum_{n=1}^N w_n}$
7:         $\alpha_c := \log \frac{1-\text{err}_c}{\text{err}_c}$
8:         $w_n := w_n e^{\alpha_c \delta(y_n \neq \hat{y}(x_n, \theta_c))}, \quad n = 1, \ldots, N$
9:     return $(\alpha, \theta)$
```

C number of component models

Functional Gradient Descent Boosting

So far, we have to derive the boosting equations **for each loss individually**.

Idea:

- compute the **gradient of the loss function** for an additional additive term and
- fit the next model that **mimicks best a gradient update step**

Advantage:

- works **for all differentiable losses**.

Functional Gradient Descent Boosting

Functional gradient:

$$\begin{aligned}\nabla_{\hat{y}} \ell(\mathcal{D}^{\text{train}}, \hat{y})|_{\hat{y}^{(C'-1)}} &= \nabla_{\hat{y}} \left(\sum_{n=1}^{N} \ell(y_n, \hat{y}_n) \right) |_{\hat{y}^{(C'-1)}} \\ &= \left(\frac{\partial \ell}{\partial \hat{y}} (y_n, \hat{y}^{(C'-1)}(x_n)) \right)_{n=1,\ldots,N}\end{aligned}$$

Functional Gradient Descent Boosting

Functional gradient:

$$\begin{aligned}\nabla_{\hat{y}} \ell(\mathcal{D}^{\text{train}}, \hat{y})|_{\hat{y}^{(C'-1)}} =& \nabla_{\hat{y}} \left(\sum_{n=1}^{N} \ell(y_n, \hat{y}_n) \right) |_{\hat{y}^{(C'-1)}} \\ =& \left(\frac{\partial \ell}{\partial \hat{y}} (y_n, \hat{y}^{(C'-1)}(x_n)) \right)_{n=1,\ldots,N}\end{aligned}$$

A functional gradient update step would do:

$$\hat{y}^{(C')} = \hat{y}^{(C'-1)} - \eta \nabla_{\hat{y}} \ell(\mathcal{D}^{\text{train}}, \hat{y})$$

Functional Gradient Descent Boosting

Functional gradient:

$$\nabla_{\hat{y}} \ell(\mathcal{D}^{\text{train}}, \hat{y})|_{\hat{y}^{(C'-1)}} = \nabla_{\hat{y}} \left(\sum_{n=1}^{N} \ell(y_n, \hat{y}_n) \right) |_{\hat{y}^{(C'-1)}}$$

$$= \left(\frac{\partial \ell}{\partial \hat{y}} (y_n, \hat{y}^{(C'-1)}(x_n)) \right)_{n=1,\ldots,N}$$

A functional gradient update step would do:

$$\hat{y}^{(C')} = \hat{y}^{(C'-1)} - \eta \nabla_{\hat{y}} \ell(\mathcal{D}^{\text{train}}, \hat{y})$$

Boosting adds the next model:

$$\hat{y}^{(C')} = \hat{y}^{(C'-1)} + \alpha_{C'} \hat{y}(\theta_{C'})$$

Functional Gradient Descent Boosting

Functional gradient:

$$\nabla_{\hat{y}} \ell(\mathcal{D}^{\text{train}}, \hat{y})|_{\hat{y}^{(C'-1)}} = \nabla_{\hat{y}} \left(\sum_{n=1}^{N} \ell(y_n, \hat{y}_n) \right) |_{\hat{y}^{(C'-1)}}$$
$$= \left(\frac{\partial \ell}{\partial \hat{y}}(y_n, \hat{y}^{(C'-1)}(x_n)) \right)_{n=1,\ldots,N}$$

A functional gradient update step would do:

$$\hat{y}^{(C')} = \hat{y}^{(C'-1)} - \eta \nabla_{\hat{y}} \ell(\mathcal{D}^{\text{train}}, \hat{y})$$

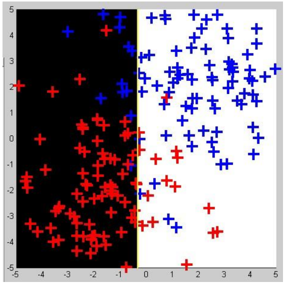
Boosting adds the next model:

$$\hat{y}^{(C')} = \hat{y}^{(C'-1)} + \alpha_{C'} \hat{y}(\theta_{C'})$$

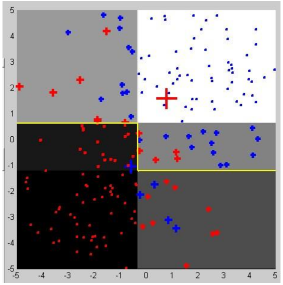
To mimick the gradient update step with steplength $\eta := 1$:

$$\theta_{C'} := \arg\min_{\theta_{C'}} \sum_{n=1}^{N} (- \left(\nabla_{\hat{y}} \ell(\mathcal{D}^{\text{train}}, \hat{y})|_{\hat{y}^{(C'-1)}} \right)_n - \hat{y}(x_n, \theta_{C'}))^2$$

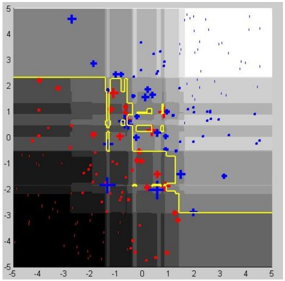
AdaBoost / Example (Decision Tree Stumps)



$C' = 1$



$C' = 3$



$C' = 120$

[Mur12, fig. 16.10]

Performance Comparison / Low Dimensional Data

MODEL	1ST	2ND	3RD	4TH	5TH	6TH	7TH	8TH	9TH	10TH
BST-DT	0.580	0.228	0.160	0.023	0.009	0.000	0.000	0.000	0.000	0.000
RF	0.390	0.525	0.084	0.001	0.000	0.000	0.000	0.000	0.000	0.000
BAG-DT	0.030	0.232	0.571	0.150	0.017	0.000	0.000	0.000	0.000	0.000
SVM	0.000	0.008	0.148	0.574	0.240	0.029	0.001	0.000	0.000	0.000
ANN	0.000	0.007	0.035	0.230	0.606	0.122	0.000	0.000	0.000	0.000
KNN	0.000	0.000	0.000	0.009	0.114	0.592	0.245	0.038	0.002	0.000
BST-STMP	0.000	0.000	0.002	0.013	0.014	0.257	0.710	0.004	0.000	0.000
DT	0.000	0.000	0.000	0.000	0.000	0.000	0.004	0.616	0.291	0.089
LOGREG	0.000	0.000	0.000	0.000	0.000	0.000	0.040	0.312	0.423	0.225
NB	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.030	0.284	0.686

Table 16.3 Fraction of time each method achieved a specified rank, when sorting by mean performance across 11 datasets and 8 metrics. Based on Table 4 of (Caruana and Niculescu-Mizil 2006). Used with kind permission of Alexandru Niculescu-Mizil.

11 datasets, $\sim$ 10.000 instances, 9-200 variables [Mur12, p. 582]

Performance Comparison / High Dimensional Data

TABLE 11.3. *Performance of different methods. Values are average rank of test error across the five problems (low is good), and mean computation time and standard error of the mean, in minutes.*

Method	Screened Features		ARD Reduced Features	
	Average Rank	Average Time	Average Rank	Average Time
Bayesian neural networks	1.5	384(138)	1.6	600(186)
Boosted trees	3.4	3.03(2.5)	4.0	34.1(32.4)
Boosted neural networks	3.8	9.4(8.6)	2.2	35.6(33.5)
Random forests	2.7	1.9(1.7)	3.2	11.2(9.3)
Bagged neural networks	3.6	3.5(1.1)	4.0	6.4(4.4)

5 datasets, 100–6.000 instances, 500-100.000 variables

[HTFF05, p. 414]

Outline

Underlying Idea

So far, we build ensemble models where the combination weights do not depend on the predictors:

$$\hat{y}(x) := \sum_{c=1}^{C} \alpha_c \, \hat{y}_c(x)$$

i.e., all instances x are reconstructed from their predictions $\hat{y}_c(x)$ by the component models in the same way α.

Underlying Idea

So far, we build ensemble models where the combination weights do not depend on the predictors:

$$\hat{y}(x) := \sum_{c=1}^{C} \alpha_c \, \hat{y}_c(x)$$

i.e., all instances x are reconstructed from their predictions $\hat{y}_c(x)$ by the component models in the same way α.

New idea: allow each instance to be reconstructed in an instance-specific way.

$$\hat{y}(x) := \sum_{c=1}^{C} \alpha_c(x) \, \hat{y}_c(x)$$

Mixtures of Experts

$$x_n \in \mathbb{R}^M, y_n \in \mathbb{R}, c_n \in \{1, \ldots, C\}, \theta := (\beta, \sigma^2, \gamma):$$

$$\begin{aligned} p(y_n \mid x_n, c_n; \theta) :=& \mathcal{N}(y \mid \beta_{c_n}^T x_n, \sigma_{c_n}^2) \\ p(c_n \mid x_n; \theta) :=& \text{Cat}(c \mid \mathcal{S}(\gamma^T x)) \end{aligned}$$

with **softmax function**

$$\mathcal{S}(x)_m := \frac{e^{x_m}}{\sum_{m'=1}^M e^{x_{m'}}}, \quad x \in \mathbb{R}^M$$

- C component models (**experts**) $\mathcal{N}(y \mid \beta_c^T x, \sigma_c^2)$
- each model c is expert in some region of predictor space, defined by its component weight (**gating function**) $\mathcal{S}(\gamma^T x)_c$
- a mixture model with latent nominal variable $z_n := c_n$.

Mixtures of Experts

$$\begin{tikzcd} x_n \arrow[d] \arrow[dd, bend left] \\ c_n \arrow[d] \\ y_n \end{tikzcd}$$

$$x_n \in \mathbb{R}^M, y_n \in \mathbb{R}, c_n \in \{1, \ldots, C\}, \theta := (\beta, \sigma^2, \gamma):$$

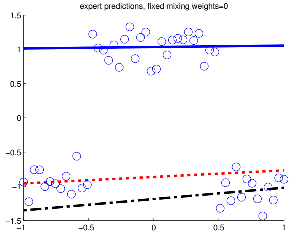
$$\begin{aligned} p(y_n \mid x_n, c_n; \theta) :=& \mathcal{N}(y \mid \beta_{c_n}^T x_n, \sigma_{c_n}^2) \\ p(c_n \mid x_n; \theta) :=& \text{Cat}(c \mid \mathcal{S}(\gamma^T x)) \end{aligned}$$

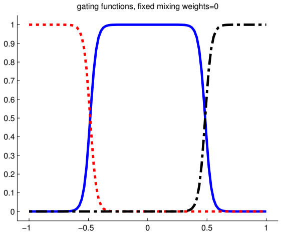
with **softmax function**

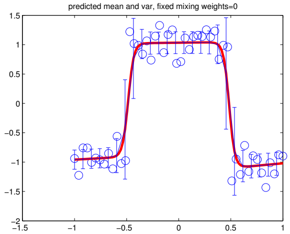
$$\mathcal{S}(x)_m := \frac{e^{x_m}}{\sum_{m'=1}^M e^{x_{m'}}}, \quad x \in \mathbb{R}^M$$

- C component models (**experts**) $\mathcal{N}(y \mid \beta_c^T x, \sigma_c^2)$
- each model c is expert in some region of predictor space, defined by its component weight (**gating function**) $\mathcal{S}(\gamma^T x)_c$
- a mixture model with latent nominal variable $z_n := c_n$.

Mixtures of Experts/ Example







component models component weight mixture of experts

[Mur12, fig. 11.6]

Mixtures of Experts

Generic Mixtures of Experts model:

- variables: $x_n \in \mathcal{X}, y_n \in \mathcal{Y}$
- latent variables: $c_n \in \{1, \ldots, C\}$
- component models: $p(y_n \mid x_n, c_n; \theta^y)$
  - a separate model for each c: $p(y_n \mid x_n, c; \theta^y) = p(y_n \mid x_n; \theta^y_c)$, with θ^y_c and $\theta^y_{c'}$ being disjoint for $c \neq c'$.
- combination model: $p(c_n \mid x_n; \theta^c)$

Example Mixture of Experts model:

- variables: $\mathcal{X} := \mathbb{R}^M, \mathcal{Y} := \mathbb{R}$
- component models: linear regression models $\mathcal{N}(y \mid \beta_c^T x, \sigma_c^2)$
- combination model: logistic regression model $\text{Cat}(c \mid \mathcal{S}(\gamma^T x))$

For prediction:

$$p(y \mid x) = \sum_{c=1}^{C} p(y \mid x, c)\, p(c \mid x)$$

Mixtures of Experts

Generic Mixtures of Experts model:

- variables: $x_n \in \mathcal{X}, y_n \in \mathcal{Y}$
- latent variables: $c_n \in \{1, \ldots, C\}$
- component models: $p(y_n \mid x_n, c_n; \theta^y)$
  - a separate model for each c: $p(y_n \mid x_n, c; \theta^y) = p(y_n \mid x_n; \theta^y_c)$, with θ^y_c and $\theta^y_{c'}$ being disjoint for $c \neq c'$.
- combination model: $p(c_n \mid x_n; \theta^c)$

Example Mixture of Experts model:

- variables: $\mathcal{X} := \mathbb{R}^M, \mathcal{Y} := \mathbb{R}$
- component models: linear regression models $\mathcal{N}(y \mid \beta_c^T x, \sigma_c^2)$
- combination model: logistic regression model $\text{Cat}(c \mid \mathcal{S}(\gamma^T x))$

For prediction:

$$p(y \mid x) = \sum_{c=1}^{C} \underbrace{p(y \mid x, c)}_{=\hat{y}_c(x)} \underbrace{p(c \mid x)}_{=\alpha_c(x)}$$

Learning Mixtures of Experts

complete data likelihood:

$$\ell(\theta^y, \theta^c, c; \mathcal{D}^{\text{train}}) := \prod_{n=1}^{N} p(y_n | x_n, c_n; \theta^y) p(c_n | x_n; \theta^c), \quad c_n \in \{1, \ldots, C\}$$

Cannot be computed, as c_n is unknown.

Learning Mixtures of Experts

complete data likelihood:

$$\ell(\theta^y, \theta^c, c; \mathcal{D}^{\text{train}}) := \prod_{n=1}^{N} p(y_n | x_n, c_n; \theta^y) p(c_n | x_n; \theta^c), \quad c_n \in \{1, \ldots, C\}$$

Cannot be computed, as c_n is unknown.

weighted complete data likelihood:

$$\ell(\theta^y, \theta^c, w; \mathcal{D}^{\text{train}}) := \prod_{n=1}^{N} \prod_{c=1}^{C} (p(y_n | x_n, c; \theta^y) p(c | x_n; \theta^c))^{w_{n,c}}, \quad w_n \in \Delta_C$$

$$-\log \ell(\theta^y, \theta^c, w; \mathcal{D}^{\text{train}}) = -\sum_{n=1}^{N} \sum_{c=1}^{C} w_{n,c} (\log p(y_n | x_n, c; \theta^y) + \log p(c | x_n; \theta^c)$$

Note: $\Delta_C := \{w \in [0,1]^C \mid \sum_{c=1}^{C} w_c = 1\}$.

Learning Mixtures of Experts

complete data likelihood:

$$\ell(\theta^y, \theta^c, c; \mathcal{D}^{\text{train}}) := \prod_{n=1}^{N} p(y_n|x_n, c_n; \theta^y)p(c_n|x_n; \theta^c), \quad c_n \in \{1, \ldots, C\}$$

Cannot be computed, as c_n is unknown.

weighted complete data likelihood:

$$\ell(\theta^y, \theta^c, w; \mathcal{D}^{\text{train}}) := \prod_{n=1}^{N} \prod_{c=1}^{C} (p(y_n|x_n, c; \theta^y)p(c|x_n; \theta^c))^{w_{n,c}}, \quad w_n \in \Delta_C$$

$$-\log \ell(\theta^y, \theta^c, w; \mathcal{D}^{\text{train}}) = -\sum_{n=1}^{N} \sum_{c=1}^{C} w_{n,c} (\log p(y_n|x_n, c; \theta^y) + \log p(c|x_n; \theta^c)$$

Cannot be computed either, as w_n is unknown;
but w_n can be treated as parameter.

Note: $\Delta_C := \{w \in [0,1]^C \mid \sum_{c=1}^{C} w_c = 1\}$.

Learning Mixtures of Experts

$$\text{minimize} \quad -\log \ell(\theta^y, \theta^c, w; \mathcal{D}^{\text{train}})$$
$$= -\sum_{n=1}^{N} \sum_{c=1}^{C} w_{n,c} \left(\log p(y_n | x_n, c; \theta^y) + \log p(c | x_n; \theta^c)\right), \quad w_n \in \Delta_C$$

Block coordinate descent (EM):

Learning Mixtures of Experts

$$\text{minimize} \quad -\log \ell(\theta^y, \theta^c, w; \mathcal{D}^{\text{train}})$$

$$= -\sum_{n=1}^{N} \sum_{c=1}^{C} w_{n,c} \left(\log p(y_n | x_n, c; \theta^y) + \log p(c | x_n; \theta^c) \right), \quad w_n \in \Delta_C$$

Block coordinate descent (EM):

1. Minimize w.r.t. θ^y:
   - decomposes into C problems $$\arg\min_{\theta_c^y} -\sum_{n=1}^{N} w_{n,c} \log p(y_n | x_n; \theta_c^y)$$
   - learn C component models for $\mathcal{D}^{\text{train}}$ with case weights $w_{n,c}$.

Learning Mixtures of Experts

$$\text{minimize} \quad -\log \ell(\theta^y, \theta^c, w; \mathcal{D}^{\text{train}})$$
$$= -\sum_{n=1}^{N}\sum_{c=1}^{C} w_{n,c} \left(\log p(y_n | x_n, c; \theta^y) + \log p(c | x_n; \theta^c)\right), \quad w_n \in \Delta_C$$

Block coordinate descent (EM):

1. Minimize w.r.t. θ^y:
   - decomposes into C problems $$\arg\min_{\theta^y_c} -\sum_{n=1}^{N} w_{n,c} \log p(y_n | x_n; \theta^y_c)$$
   - learn C component models for $\mathcal{D}^{\text{train}}$ with case weights $w_{n,c}$.
2. Minimize w.r.t. θ^c:
   - solve $$\arg\min_{\theta^c} -\sum_{n=1}^{N}\sum_{c=1}^{C} w_{n,c} \log p(c | x_n; \theta^c)$$
   - learn a combination model for target c on

$$\mathcal{D}^{\text{train,wcompl}} := \{(x_n, c, w_{n,c}) \mid n = 1, \ldots, N, c = 1, \ldots, C\}$$

Learning Mixtures of Experts

$$\text{minimize} \quad -\log \ell(\theta^y, \theta^c, w; \mathcal{D}^{\text{train}})$$

$$= -\sum_{n=1}^{N} \sum_{c=1}^{C} w_{n,c} \left(\log p(y_n | x_n, c; \theta^y) + \log p(c | x_n; \theta^c)\right), \quad w_n \in \Delta_C$$

Block coordinate descent (EM):

3. Minimize w.r.t. $w_{n,c}$:
   - decomposes into N problems

$$\arg\min_{w_{n,c}} -\sum_{c=1}^{C} w_{n,c} \left(\log p(y_n \mid x_n; \theta^y_c) + \log p(c \mid x_n; \theta^c)\right), \quad w_n \in \Delta_C$$

Learning Mixtures of Experts

$$\text{minimize} \quad -\log \ell(\theta^y, \theta^c, w; \mathcal{D}^{\text{train}})$$

$$= -\sum_{n=1}^{N} \sum_{c=1}^{C} w_{n,c} \left(\log p(y_n | x_n, c; \theta^y) + \log p(c | x_n; \theta^c)\right), \quad w_n \in \Delta_C$$

Block coordinate descent (EM):

3. Minimize w.r.t. $w_{n,c}$:
   - decomposes into N problems

$$\arg\min_{w_{n,c}} -\sum_{c=1}^{C} w_{n,c} \underbrace{\left(\log p(y_n \mid x_n; \theta^y_c) + \log p(c \mid x_n; \theta^c)\right)}_{=:a_c}, \quad w_n \in \Delta_C$$

Learning Mixtures of Experts

$$\text{minimize} \quad -\log \ell(\theta^y, \theta^c, w; \mathcal{D}^{\text{train}})$$
$$= -\sum_{n=1}^{N}\sum_{c=1}^{C} w_{n,c}\left(\log p(y_n|x_n, c; \theta^y) + \log p(c|x_n; \theta^c)\right), \quad w_n \in \Delta_C$$

Block coordinate descent (EM):

3. Minimize w.r.t. $w_{n,c}$:
   - decomposes into N problems
     $$\arg\min_{w_{n,c}} -\sum_{c=1}^{C} w_{n,c} \underbrace{\left(\log p(y_n \mid x_n; \theta^y_c) + \log p(c \mid x_n; \theta^c)\right)}_{=:a_c}, \quad w_n \in \Delta_C$$
   - analytical solution
     $$w_{n,c} = \frac{a_c}{\sum_{c'=1}^{C} a_{c'}} = \frac{\log p(y_n \mid x_n; \theta^y_c) + \log p(c \mid x_n; \theta^c)}{\sum_{c'=1}^{C} \log p(y_n \mid x_n; \theta^y_{c'}) + \log p(c' \mid x_n; \theta^c)}$$

Remarks

- Mixtures of experts can use **any model as component model**.
- Mixtures of experts can use **any classification model as combination model**.
  - both models need to be able to **deal with case weights**
  - both models need to be able to **output probabilities**

Remarks

- Mixtures of experts can use **any model as component model**.
- Mixtures of experts can use **any classification model as combination model**.
  - both models need to be able to **deal with case weights**
  - both models need to be able to **output probabilities**

- if data is **sparse**, sparsity can be naturally used in both, component and combination models.

Remarks

- Mixtures of experts can use **any model as component model**.
- Mixtures of experts can use **any classification model as combination model**.
  - both models need to be able to **deal with case weights**
  - both models need to be able to **output probabilities**

- if data is **sparse**, sparsity can be naturally used in both, component and combination models.

- Updating the three types of parameters can be **interleaved**.
  - this way, $w_{n,c}$ never has to be materialized (but for a mini batch, possibly a single n)

Outlook: Hierarchical Mixture of Experts

$$\begin{tikzcd}
x_n \arrow[d] \arrow[dd, bend left] \\
c_n \arrow[d] \\
y_n
\end{tikzcd}$$

mixture of experts

$$\begin{tikzcd}
x_n \arrow[d] \arrow[ddd, bend left] \\
c_n^1 \arrow[d] \arrow[dd, bend left] \\
c_n^2 \arrow[d] \\
y_n
\end{tikzcd}$$

hierarchical mixture of experts

Outline

Variable Importance

Some models allow to assess the importance of single variables (or more generally subsets of variables; **variable importance**), e.g.,

- linear models: the z-score
- decision trees: the number of times a variable occurs in its splits

Variable Importance

Some models allow to assess the importance of single variables (or more generally subsets of variables; **variable importance**), e.g.,

- linear models: the z-score
- decision trees: the number of times a variable occurs in its splits

Variable importance of ensembles of such models can be measured as **average variable importance in the component models**:

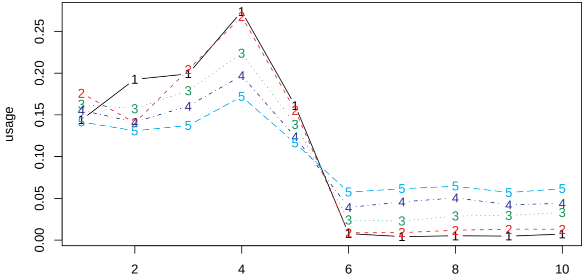
$$\text{importance}(X_m, \hat{y}) := \frac{1}{C} \sum_{c=1}^{C} \text{importance}(X_m, \hat{y}_c), \quad m \in \{1, \ldots, M\}$$

Variable Importance / Example

Synthetic data:

$$x \sim \text{uniform}([0,1]^{10})$$
$$y \sim \mathcal{N}(y \mid 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5, 1)$$

Model: Bayesian adaptive regression tree (variant of a random forest; see [Mur12, p. 551]).



[Mur12, fig. 16.21]

Color denotes the number C of component models.

Variable Dependence: Partial Dependence Plot

For any model $\hat{y}$ (and thus any ensemble), the dependency of the model on a variable X_m can be visualized by a **partial dependence plot**:

plot $z \in \text{range}(X_m)$ vs.

$$\hat{y}_{\text{partial}}(z; X_m, \mathcal{D}^{\text{train}}) := \frac{1}{N} \sum_{n=1}^{N} \hat{y}((x_{n,1}, \ldots, x_{n,m-1}, z, x_{n,m+1}, \ldots, x_{n,M})),$$

or for a subset of variables

$$\hat{y}_{\text{partial}}(z; X_V, \mathcal{D}^{\text{train}}) := \frac{1}{N} \sum_{n=1}^{N} \hat{y}(\rho(x, V, z)), \quad V \subseteq \{1, \ldots, M\}$$

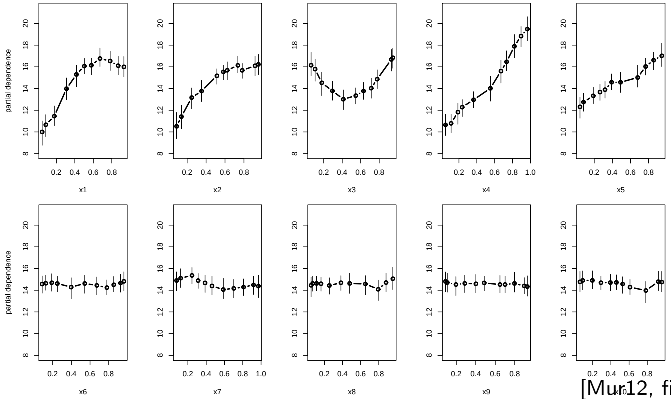
$$\text{with } \rho(x, V, z)_m := \begin{cases} z_m, & \text{if } m \in V \\ x_m, & \text{else} \end{cases}, \quad m \in \{1, \ldots, M\}$$

Variable Dependence / Example

Synthetic data:

$$x \sim \text{uniform}([0,1]^{10})$$

$$y \sim \mathcal{N}(y \mid 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5, 1)$$



[Mur12, fig. 16.20]

Further Readings

- Averaging, Voting, Stacking: [Mur12, chapter 16.6], [HTFF05, chapter 8.8], [Bis06, chapter 14.2].
- Bayesian model averaging: [Bis06, chapter 14.1], [Mur12, chapter 16.6.3], [HTFF05, chapter 8.8].
- Bagging: [Mur12, chapter 16.2.5], [HTFF05, chapter 8.7], [Bis06, chapter 14.2].
- Random Forests: [HTFF05, chapter 15], [Mur12, chapter 16.2.5], [Bis06, chapter 14.3].
- Boosting: [Mur12, chapter 16.4], [HTFF05, chapter 10], [Bis06, chapter 14.3].
- Mixtures of Experts: [Bis06, chapter 14.5]. [Mur12, chapter 11.2.4, 11.4.3], [HTFF05, chapter 9.5].

References

Christopher M. Bishop.
Pattern recognition and machine learning, volume 1.
springer New York, 2006.

Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin.
The elements of statistical learning: data mining, inference and prediction, volume 27.
Springer, 2005.

Kevin P. Murphy.
Machine learning: a probabilistic perspective.
The MIT Press, 2012.