Machine Learning 2

B. Ensembles / B.1. Stacking & Bagging

Lars Schmidt-Thieme

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

Syllabus

			A. Advanced Supervised Learning
Fri.	24.4.	(1)	A.1 Generalized Linear Models
Fri.	1.5.	—	— *Labour Day* —
Fri.	8.5.	(2)	A.2 Gaussian Processes
Fri.	15.5.	(3)	A.3 Advanced Support Vector Machines
			B. Ensembles
Fri.	22.5.	(4)	B.1 Stacking
			& B.2 Boosting
Fri.	29.5.	(5)	B.3 Mixtures of Experts
Fri.	5.6.	—	— *Pentecoste Break* —
			C. Sparse Models
Fri.	12.6.	(6)	C.1 Homotopy and Least Angle Regression
Fri.	19.6.	(7)	C.2 Proximal Gradients
Fri.	26.6.	(8)	C.3 Laplace Priors
Fri.	3.7.	(9)	C.4 Automatic Relevance Determination
			D. Complex Predictors
Fri.	10.7.	(10)	D.1 Latent Dirichlet Allocation (LDA)
Fri.	17.7.	(11)	Q & A

Outline

1. Model Averaging & Voting

2. Stacking

3. Bagging & Random Forests

Outline

Model Selection

- Assume you have learned C many models $\hat{y}_c$ for a given problem:

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

  e.g.,
  - $\hat{y}_1$: a linear model
  - $\hat{y}_2$: a decision tree
  - $\hat{y}_3$: a support vector machine
  - $\hat{y}_4$: a neural network
- Which one would you deploy and use now for your application?

Model Selection

If we have **several models**

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

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

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

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):

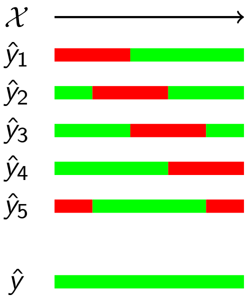
$$\hat{y}(x) := \frac{1}{C} \sum_{c=1}^{C} \hat{y}_c(x)$$

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:

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

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:

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

Model Averaging / Algorithm

```
learn-modelaveraging($\mathcal{D}^{\text{train}}, (a_c)_{c=1:C}$) :
    for $c := 1, \ldots, C$:
        $\hat{y}_c := a_c(\mathcal{D}^{\text{train}})$
    $\hat{y} := \frac{1}{C} \sum_{c=1}^{C} \hat{y}_c$
    return $\hat{y}$
```

where

- a_c is a learning algorithm for component model c

Outline

Can we Learn the Combination Weights?

- we have learned C many models $\hat{y}_c$ $(c = 1, \dots, C)$ for a task.
- and aim to combine their outputs like:

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

- can we learn the combination weights α_c ?

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

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

  - 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

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

  - 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.

Optimizing Hyperparameters of Component Models

1. Optimize each component model's hyperparameters on its own.
   - strong component models
2. Optimize hyperparameters of all component models jointly.
   - usually too many hyperparameters
   - not done
3. Do not optimize hyperparameters.
   - just choose some
   - see next section

Stacking / Algorithm

```
learn-stacking($\mathcal{D}^{\text{train}}, (a_c)_{c=1:C}, a_{\text{2nd}}$) :
  $\mathcal{D}^{\text{train}\prime}, \mathcal{D}^{\text{val}} = \text{split}(\mathcal{D}^{\text{train}})$
  for $c := 1, \ldots, C$:
    $\hat{y}_c := a_c(\mathcal{D}^{\text{train}\prime})$
  $\mathcal{D}^{\text{train}}_{\text{2nd}} := \{(x'_1, x'_2, \ldots, x'_C, y) \mid x, y \in \mathcal{D}^{\text{val}}, x'_c := \hat{y}_c(x)\}$
  $\hat{y}_{\text{2nd}} := a_{\text{2nd}}(\mathcal{D}^{\text{train}}_{\text{2nd}})$
  $\hat{y} := y_{\text{2nd}} \circ (\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_C)$
  return $\hat{y}$
```

where

- a_c is a learning algorithm for component model c and
- a_{2nd} is a learning algorithm for the combination model

Note: Model averaging is a special case where split assigns all data to the train' partition and a_{2nd} always returns $\frac{1}{C}\sum_{c=1}^{C} x'_c$.

Outline

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 input 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 $\mathcal{D}_c^{\text{train}}$.
- **aggregate**/average these models:

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

- Q: can we represent boostrap aggregation by case weights?

Bootstrap

- resample explicitly:

```
bootstrap($X = \{X_1, X_2, \ldots, X_N\}$):
  $X' := \emptyset$
  do $N$ times:
    $n \sim \text{unif}(\{1, \ldots, N\})$
    $X' := X' \cup \{X_n\}$
  return $X'$
```

- use case weights:

```
bootstrap-cw($X = \{X_1, X_2, \ldots, X_N\}$):
  for $n = 1, \ldots, N$:
    $w_n \sim \text{binomial}(N, \frac{1}{N})$
  return $\{(X_n, w_n) \mid n = 1, \ldots, N\}$
```

Excursion: Importance Sampling

- vanilla SGD samples instances uniformly:

```
$n \sim \text{unif}\{1, \ldots, N\}$
$\theta^{(t+1)} := \theta^{(t)} - \eta(\nabla\ell(x_n, y_n) + \nabla\text{reg}(\theta))$
```

- **importance sampling**: twist the sampling for faster convergence

```
for $n := 1, \ldots, N$:
    $p_n := ||\nabla\ell(x_n, y_n)||$
$n \sim \text{cat}((p_n)_{n=1:N})$
$\theta^{(t+1)} := \theta^{(t)} - \eta\frac{1}{p_n}(\nabla\ell(x_n, y_n) + \nabla\text{reg}(\theta))$
```

  - also must correct the stepsize by $\frac{1}{p_n}$
  - $\nabla\ell(x_n, y_n)$ too expensive, use cheaper proxy [**?**, **?**]
  - and/or compute only for a minibatch, not for all N samples

Bagging / Algorithm

```
learn-bagging($\mathcal{D}^{\text{train}}, a, C$) :
    for $c := 1, \ldots, C$:
        $\mathcal{D}_c^{\text{train}} \sim \text{bootstrap}(\mathcal{D}^{\text{train}})$
        $\hat{y}_c := a_c(\mathcal{D}_c^{\text{train}})$
    $\hat{y} := \frac{1}{C} \sum_{c=1}^{C} \hat{y}_c$
    return $\hat{y}$
```

or more compact:

```
learn-bagging($\mathcal{D}^{\text{train}}, a, C$) :
    return learn-modelaveraging($\mathcal{D}^{\text{train}}, (a \circ \text{bootstrap})_{c=1:C}$)
```

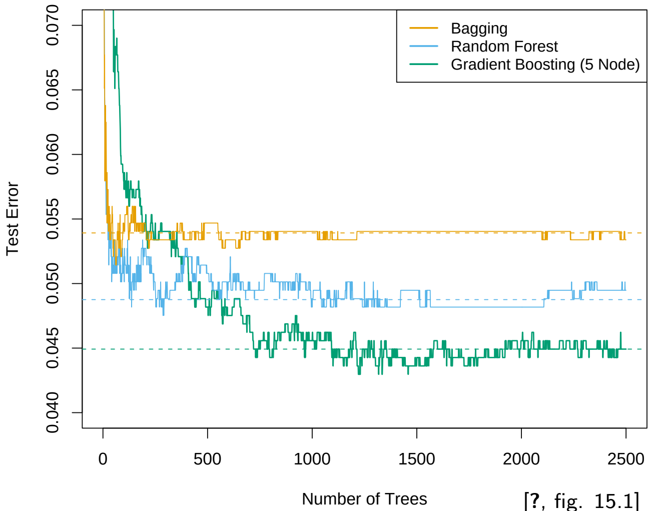
where

- a is a learning algorithm for a component model

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)



[?, fig. 15.1]

Summary (1/2)

- Combining models (**Ensembling**) can be simply accomplished by averaging the models.
- Weighted averages often provide better ensembles.
  - Estimating weights antiproportional to the error of the component models (**Bayesian Model Averaging**).
  - Learning combination weights by linear regression/classification (**linear stacking**).
- **Stacking** can use any model to learn how to combine the predictions of a set of component models (**2nd stage model**)
  - component models and combination model are learned sequentially
    - simple, easy to parallelize
    - not optimal
- Component models must have **uncorrelated errors** to yield a good ensemble.
  - different models, with different hyperparameters, using different variables, using different instances

Summary (2/2)

- **Bagging** is a ensemble strategy based on different instances / subsamples.
- **Random Forests** combine
  - Bagging and
  - random variable subsets
  - esp. for trees as component models.

Further Readings

- Averaging, Voting, Stacking: [**?**, chapter 16.6], [**?**, chapter 8.8], [**?**, chapter 14.2].
- Bayesian model averaging: [**?**, chapter 14.1], [**?**, chapter 16.6.3], [**?**, chapter 8.8].
- Bagging: [**?**, chapter 16.2.5], [**?**, chapter 8.7], [**?**, chapter 14.2].
- Random Forests: [**?**, chapter 15], [**?**, chapter 16.2.5], [**?**, chapter 14.3].

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.

Tyler B Johnson and Carlos Guestrin.
Training Deep Models Faster with Robust, Approximate Importance Sampling.
In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7265–7275. Curran Associates, Inc., 2018.

Angelos Katharopoulos and Francois Fleuret.
Not All Samples Are Created Equal: Deep Learning with Importance Sampling.
In *International Conference on Machine Learning*, pages 2525–2534, July 2018.

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