Meta-Learning

Anett Seeland, Mario Michael Krell

DFKI Bremen & Universität Bremen
Robotics Innovation Center
Director: Prof. Dr. Frank Kirchner
www.dfki.de/robotics

Motivation

When *wise people* make **critical** decisions,
they usually take into account
the opinions of *several* experts
rather than relying
on their own judgment or
that of a solitary trusted adviser.

Outline

Goals of Meta-Learning

Goals:

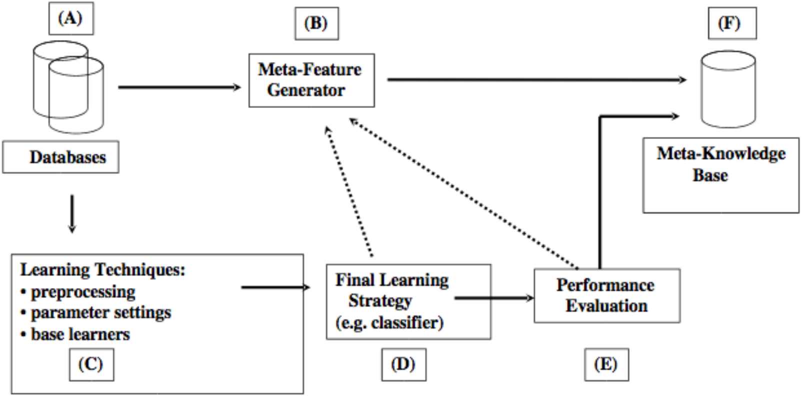
- understand the interaction between mechanism of learning and the concrete contexts in which that mechanism is applicable
- build model-selection assistants
- build task-adaptive learners

Differences to Base-Learning:

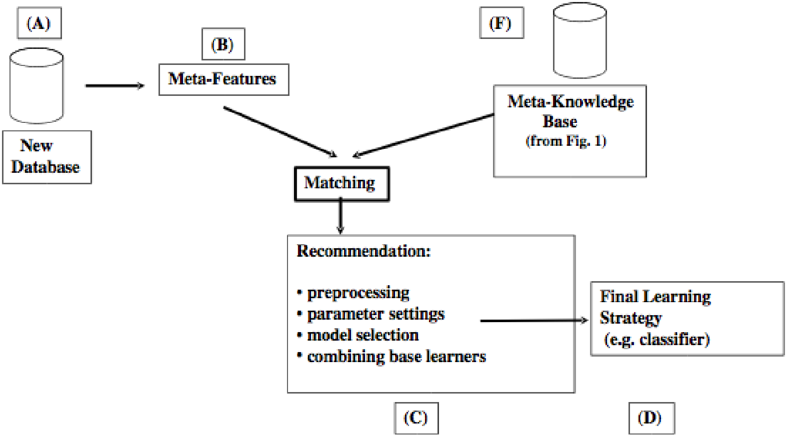
- accumulating experience on performance of multiple applications of a learning system

$\Rightarrow$ Don't choose the same learner when it has already failed in the past!

A Meta-Learning Architecture – The Knowledge-Acquisition Mode



A Meta-Learning Architecture – The Advisory Mode



Techniques in Meta-Learning I

Dataset Characterization:

- Statistical and Information-Theoretic Characterization
  - # classes, # features, # examples, $\frac{\#\text{ examples}}{\#\text{ features}}$
  - degree of correlation between features and target concept
  - average class entropy, class-conditional entropy
  - skewness, kurtosis, signal-to-noise ratio
- Model-Based Characterization
  - e.g. decision tree: nodes per feature, max depth, imbalance
- Landmarking
  - performance of different learning mechanisms
  - sampling landmarks

Techniques in Meta-Learning II

Mapping Datasets to Predictive Models:

- Hand-Crafting Meta Rules
- Learning at the Meta-Level
- Mapping Query Examples to Models
- Ranking

Learning from Base-Learners:

- Ensembles
- ...

Outline

Ensemble methods

Ensemble: using many classifiers together

- could be different types of classifiers
- each classifier gives a different view
- usually each is trained differently
- need a sensible method of combining classifiers

Popular methods:

- bagging – learning over different resamples
- boosting – learning over different distributions

Combining classifiers

Suppose that we have:

- m labeled examples $\langle \mathbf{x}, f(\mathbf{x}) \rangle$, where $f : R \rightarrow \{-1, +1\}$
- a number of T hypotheses $h_1, \ldots, h_T$, each trained on the data
- each $h_j : R \rightarrow \{-1, +1\}$

How should we combine the T classifiers?

- choose h_t with the least training error
- choose h_t with the least validation error
- use all T classifiers in a sum
- use all T classifiers in a weighted sum

Outline

Unstable learners

Imaging:

- several randomly chosen training datasets of the same size
- build a decision tree for each dataset

Will the trees make the same prediction for each new test instance?

$\rightarrow$ No! (particularly if datasets are fairly small)

$\rightarrow$ reason: standard decision tree induction is an unstable process

$\Rightarrow$ predictions made by voting become more reliable as more votes are taken into account

$\Rightarrow$ combined classifiers will seldom be less accurate

$\Rightarrow$ however improvement is not guaranteed

Bias-Variance Decomposition

Suppose the ideal situation of an infinite amount of data:

$\rightarrow$ error will still occur because no learning scheme is perfect

$\rightarrow$ **bias** for the learning problem

However in practical situation there is a 2nd source of error:

- stems from the particular training set used (unavoidably finite)

$\rightarrow$ not fully representative of the actual population of instances

$\rightarrow$ **variance** for the learning method

total expected error is a sum of bias and variance

$\Rightarrow$ combining multiple classifiers decreases the expected error by reducing the variance component

Bagging – bootstrap aggregation

- attempts to neutralize the instability of learning methods
- uses random sampling with replacement
- applies the learning scheme to each artificially derived dataset
- votes with equal weights

combined model ...

+ ... often performs significantly better than single model
+ ... is never substantially worse

Bagging – bootstrap aggregation

```
Bagging(𝒟, T):
    model generation(𝒟, T):
        m ← |𝒟|
        for t ∈ {1, ..., T} do
            𝒟_t ← bootstrap(𝒟, m)
            h_t ← build_model(𝒟_t)
    classification(x):
        for t ∈ {1, ..., T} do
            store(h_t(x))
        return class that has been predicted most often
```

Bagging enhancements

- bagging: if the underlying learning method is unstable

→ make the learning method even more unstable

- bagging uses voting

→ for probability outputs, average probabilities instead

- cost-sensitive bagging?

→ *MetaCost* combines predictive benefits of bagging with a comprehensible model for cost-sensitive prediction

  - uses bagging ensemble to relabel the training data by giving every training instance the prediction that minimizes the expected cost
  - learns a single new classifier from the relabeled data

Randomization

Other ways of introducing randomness:

- use built-in random component of a learning algorithm
- consider algorithms that greedily pick the best option at every step
  - pick randomly one of the N best options instead of a single winner
  - choose a random subset of options and pick the best from that

$\rightarrow$ tradeoff: more variety $\leftrightarrow$ less use of data

Advantages of Randomization:

- can be applied even to stable learners

$\rightarrow$ make classifiers diverse without losing too much performance

Outline

Ideas behind Boosting

- design significant different models
- each model treats a reasonable percentage of the data correctly

→ aim: models complement one another, each being a specialist in a part of the domain

- weighting is used to give more influence to the more successful models

Bagging vs. Boosting

Similarities:

- both use voting
- both combine models of the same type, e.g. decision trees

Differences:

- Boosting is iterative

$\rightarrow$ encourage new models to become experts for instances handled incorrectly by earlier ones

- Boosting weights a model's contribution by its performance

The AdaBoost Algorithm

```
AdaBoost(𝒟, T):
    model generation(𝒟, T):
        for i ∈ {1, ..., m} do  W_1(i) ← 1/m
        for t ∈ {1, ..., T} do
            h_t ← build_model(𝒟, W_t)
            ε_t ← error(𝒟, W_t, h_t)
            if ε_t = 0 or ε_t ≥ 1/2 then continue
            α_t ← 1/2 ln((1 − ε_t)/ε_t)
            for i ∈ {1, ..., m} do
                W_{t+1}(i) ← W_t(i)/Z_t × { e^{−α_t}  if h_t(x_i) = f(x_i)
                                          { e^{α_t}   if h_t(x_i) ≠ f(x_i)
                // Z_t to normalize W_{t+1}
```

$$W_{t+1}(i) \leftarrow \frac{W_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = f(x_i) \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq f(x_i) \end{cases}$$

The AdaBoost Algorithm

```
AdaBoost(D, T):
    ⋮
    classification(x):
        H(x) = sign(Σ_{t=1}^{T} α_t h_t(x))
```

$\texttt{AdaBoost}(\mathcal{D}, T)$**:**

$\vdots$

classification(x**):**

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

If model class can handle weighted instances:

$$\epsilon = \frac{\sum w_i}{\sum w_j} \text{ for } i \in \{i|h(x_i) \neq f(x_i)\} \text{ and } j \in \{1, \ldots, m\}$$

Otherwise, resample the data according to a distribution W_t

How much can the weights change?

$$W_{t+1}(i) \leftarrow \frac{W_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = f(x_i) \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq f(x_i) \end{cases}$$

recall:

- $0 \leq \epsilon_t \leq 1$
- $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$
- $e^{\alpha_t} = \sqrt{(1-\epsilon_t)/\epsilon_t}$
- $e^{-\alpha_t} = \sqrt{\epsilon_t/(1-\epsilon_t)}$

Thus, if $\epsilon_t \approx 0$ then $W_t(i) \rightarrow 0$

Role of α_t

Recall the final classifiers's form:

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Also,

$$\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$$

Thus,

- $\epsilon_t = 0.5 \rightarrow \alpha_t = 0$
- $\epsilon_t < 0.5 \rightarrow \alpha_t > 0$
- $\epsilon_t > 0.5 \rightarrow \alpha_t < 0$

Greedy search

Interestingly, AdaBoost is a greedy strategy.

It always minimizes the error with respect to the current W_t, and does not backtrack.

How does boosting help?

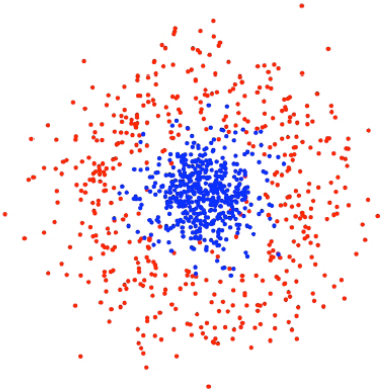
Assumption:

- easy to find *weak learners* that are "often" correct (slightly more than random)
- hard to find a *strong learner*, that is highly accurate

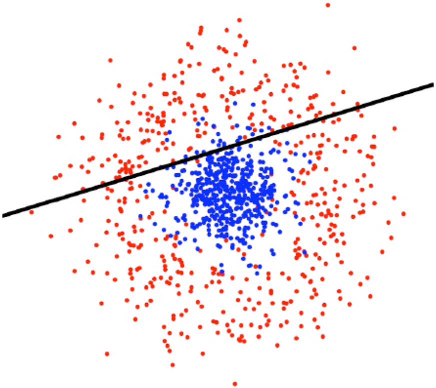
Combination of weak learners to generate a strong learner:

- focuses effort on hard-to-classify examples
- takes hard work off the learner
- learners only have to specialize in small areas
- can identify outliers

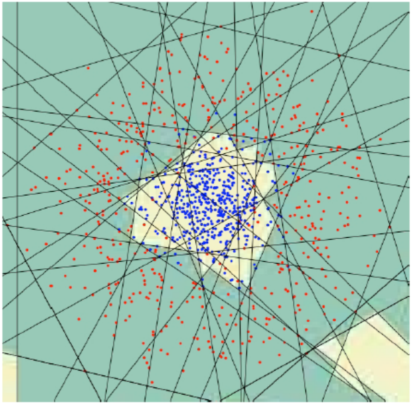
How does boosting help?



How does boosting help?



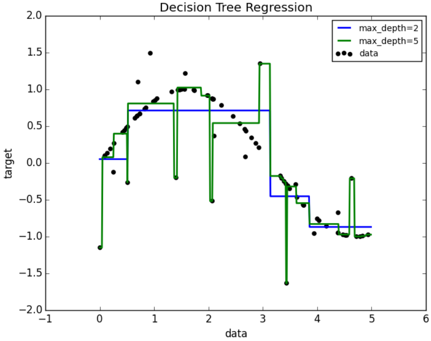
How does boosting help?



Gradient boosting

- generalizes AdaBoost to optimize an arbitrary differentiable loss function

→ inspired from regression



Gradient boosting

- boosting can be seen as an iterative functional gradient descent algorithm
- example: MSE for regression

→ the residuals $y - F(x)$ are the negative gradients of the squared error loss function $\frac{1}{2}(y - F(x))^2$

Gradient Boosting – Idea of the algorithm

- input: $\mathcal{D}$, T, and some specified loss function $L(y, F(x))$
- in each step, the previous model F_{t-1} is improved by adding an estimator h (weak learner)

$\rightarrow$ $F_t = F_{t-1} + h(x)$

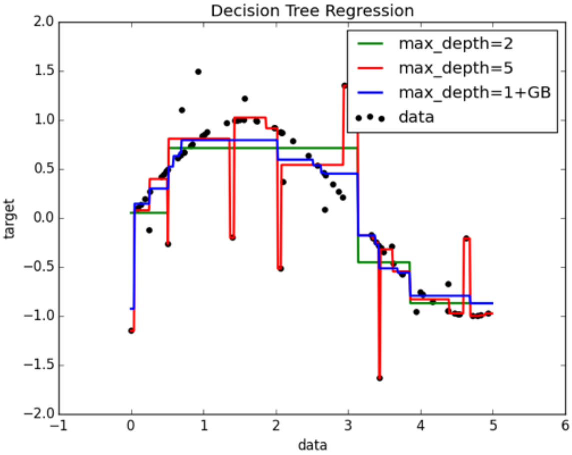
- How to find h?

$\rightarrow$ optimal h would imply $F_t = F_{t-1} + h(x) = y$, $h(x) = y - F_{t-1}$

$\rightarrow$ train h that approximates the negative gradient of L

- $F(x) = \sum_{t=1}^{T} \gamma_t h_t(x) + const$
- $\gamma_t = \arg\min_{\gamma} \sum_{i=1}^{m} L(y_i, F_{t-1}(x_i) + \gamma h_t(x_i))$

Gradient boosting for decision tree regression



Literature

- “Meta-Learning – Concepts and Techniques”, O. Maimon, L. Rokach (eds.), In: *Data Mining and Knowledge Discovery Handbook* (2nd Ed.), 2010
- “Data Mining: Practical Machine Learning Tools and Techniques” (2nd Ed.), Ian H. Witten, Eibe Frank, Morgan Kaufmann, 2005
- “A short introduction to boosting”, Y. Freund, R. Schapire, N. Abe,In: JOURNAL-JAPANESE SOCIETY FOR ARTIFICIAL INTELLIGENCE, Vol. 14, p.771–780,1999