Shrub Ensembles for Online Classification

Sebastian Buschjäger[1], Sibylle Hess[2], Katharina Morik[1]

[1]Artificial Intelligence Group, TU Dortmund, {sebastian.buschjaeger, katharina.morik}@tu-dortmund.de, [2]Data Mining Group, Technische Universiteit Eindhoven, s.c.hess@tue.nl

Shrub Ensembles at a Glance

Edge learning is in important topic in online learning. Existing methods are not well-suited for edge devices with few resources. Shrub Ensembles trains small trees (→ shrubs) on a sliding window and aggressively prunes underperforming shrubs form the ensemble using proximal gradient descent. Shrub Ensembles offers competitive predictive performance using a fraction of computational resources and memory.

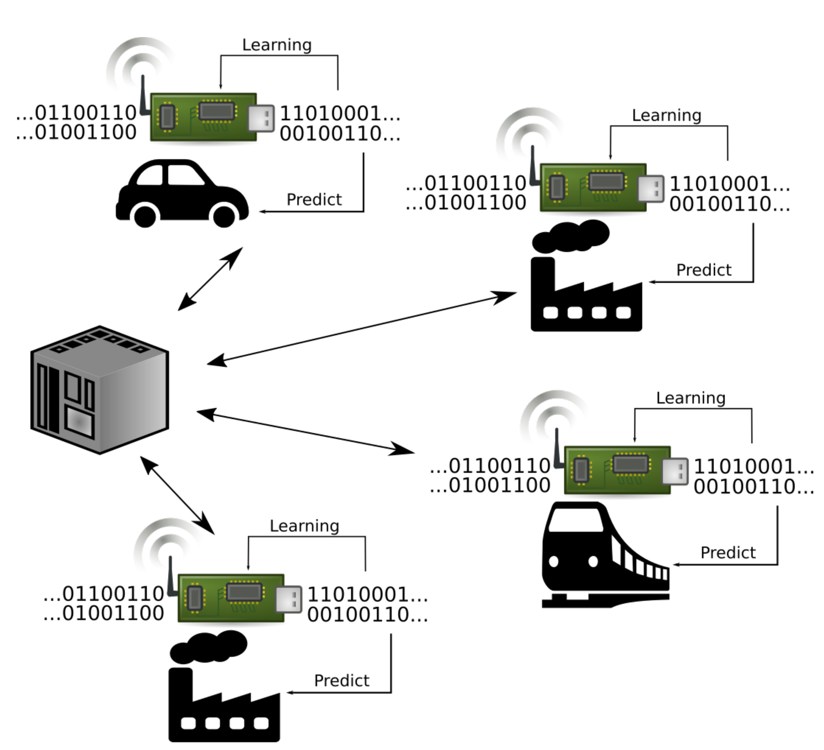
Introduction

Edge Computing has become an integral part of everyday life. It is estimated that there were roughly 22 Billion IoT devices used in 2018. Edge computing combines the computational capabilities of small devices with communication technologies to form a World Wide Web in which devices process data locally and exchange only compressed data when necessary. The benefits are:

- Data privacy is enhanced
- Communication costs are minimized
- Response time is minimized

Due to the resource-constraints of edge devices the data is processed on-the-fly. The algorithm must be

- Computationally efficient
- Memory efficient
- Adaptive to evolving streams and concept drift



Gradient-based Tree Ensembles

Gradient-based learners view the tree ensemble as a special type of neural network in which the structure of the trees is given beforehand and soft splits are used to approximate the axis-aligned splits of a regular decision tree. They offer

\+ constant memory consumption
\+ joint optimization of all trees
\- backpropagation through tree → costly gradients

Incremental Tree Ensembles

Incremental learners build axis-aligned trees by continually adding new nodes to the tree. For each leaf node they maintain a list of possible splits and if a split significantly outperforms all other splits it is added to the tree. They offer

\+ small runtime and easy implementation
\+ proven in practice
\- nodes are not removed → unbound memory

Research Question

Can we design an online ensemble with bounded memory that uses few computations for small devices?

Our Approach

Insight: Patterns repeat over time

$\dots, \boxed{(x_1,y_1),(x_2,y_2),(x_3,y_3)}, (x_4, ?) \dots, \boxed{(x_1,y_1),(x_2,y_2),(x_3,y_3)}, (x_4, ?) \dots$

train — $h_1(x)$ train — $h_1(x)$

Idea: Learn optimal ensemble of patterns (→ trees) over time from sliding windows. Formally:

$$\arg\min_{w \in \mathbb{R}^K} \sum_{t=1}^{T} \ell\left(\sum_{i=1}^{K} w_i h_i(x_t), y_t\right)$$
$$\text{s.t. } \|w\|_0 \leq M, w_i \geq 0, \sum_{i=1}^{K} w_i = 1$$

- K (theoretical) number of patterns in stream
- $\ell\colon \mathbb{R}^C \times \mathcal{Y} \to \mathbb{R}_+$ loss with C classes
- $M \geq 1$ maximum number of ensemble members
- $\|w\|_0 = \Sigma_{i=1}^{K} \mathbb{1}\{w_i \neq 0\}$

Algorithm basis: Proximal Gradient Descent

$$w \leftarrow \mathcal{P}\left(w - \alpha \nabla_w L_{\mathcal{B}}(w)\right)$$

- $\mathcal{B}$ is the current window with $|\mathcal{B}| = B$ examples
- $\nabla_w L_{\mathcal{B}}(w)$ gradient on $\mathcal{B}$, $\alpha \in \mathbb{R}_+$ step-size
- $\mathcal{P}(w)$ prox-operator for constraints $\Delta = \left\{w \in \mathbb{R}_+^K \middle| \Sigma_{i=1}^K w_i = 1, \|w\|_0 = M\right\}$

Our Algorithm

Algorithm 1: Shrub Ensembles.

```
w ← (0); B ← []; H ← []                ▷ Init.
for next item (x, y) do
    if |B| = B then                    ▷ Update batch
        B.pop_first()
    B.append((x, y))
    h_new ← train(B)                   ▷ Add new classifier
    H.append(h_new)
    w ← (w_1, ..., w_M, 0)             ▷ Initialize weight
    w ← w − α∇_w L_B(w)                ▷ Gradient step
    w, H ← sorted(w, H)                ▷ Sort decreasing order
    w ← P(w)                           ▷ Project on feasible set
    w, H ← prune(w, H)                 ▷ Remove zero weights
```

Runtime: $\mathcal{O}(dB^2 \log B + \log M)$ per example, where $\mathcal{O}(dB^2 \log B)$ is due to training the trees (e.g. using CART) and $\log M$ is due to maintaining the sorted list of weights.

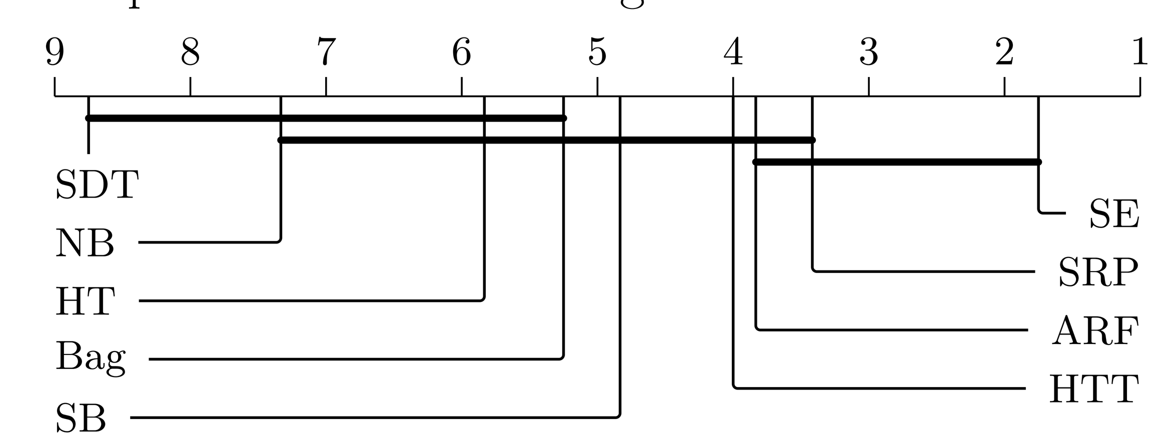
Memory: $\mathcal{O}(dB + 2B(M+1))$ per example. Here $\mathcal{O}(dB)$ is due to the sliding window and $2B(M+1)$ are the trees in the ensemble. The trees are all small because they are trained on small windows. Hence they are more akin to shrubs instead of large trees giving the name to our method.

Theoretical Results

A formal Theorem in the paper establishes that for a step-size $\alpha > \frac{BC}{4m}$ with $m = ||w||_0$ Shrub Ensembles will always replace that tree with the smallest weight if $m = M$. Hence it tries to maintain as many old trees as possible, but also quickly adapts to new situations if necessary.

Experimental Results

In our experimental analysis we are interested in the accuracy-memory trade-off of the algorithms:
1) We plot the Pareto Front of each algorithm using different hyperparameter configurations.
2) We computed the Area Under the Curve of this Plot to quantify the accuracy-memory trade-off.
3) We rank each method according to its trade-off and plot them in a CD diagram



Shrub Ensembles (SE) offers the best accuracy-memory trade-off compared to other methods and ranks first in nearly every experiment. The code is available under `https://github.com/sbuschjaeger/se-online`

Acknowledgements

Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", DFG project number 124020371, SFB project A1, `http://sfb876.tu-dortmund.de`. Part of the work on this research has been funded by the Federal Ministry of Education and Research of Germany as part of the competence center for machine learning ML2R (01|18038A), `https://www.ml2r.de/`.