ICML
International Conference On Machine Learning

Growing Adaptive Multi-Hyperplane Machines

Nemanja Djuric, Zhuang Wang, Slobodan Vucetic
Uber ATG Facebook Temple University

Overview

- Intro to Multi-hyperplane Machines (MMs)
- Adaptive MMs (AMMs) extension
- Proposed approach: **Growing AMMs**
- Regret and generalization bounds
- Experimental results

(Uni-hyperplane) Multi-class SVM

- Let us assume a multi-class data set of size T

$$\mathcal{D} = \{(\mathbf{x}_t, y_t), t = 1, \ldots, T\}$$

- Task is to learn a function mapping a data point into one of M classes
- Multi-class SVM (Crammer & Singer, 2001) is of the form

$$f(\mathbf{x}) = \arg\max_{i \in \mathcal{Y}} g(i, \mathbf{x}), \text{ where } g(i, \mathbf{x}) = \mathbf{w}_i^\top \mathbf{x}$$

where the model is parameterized by the weight matrix

$$\mathbf{W} = [\mathbf{w}_1 \; \mathbf{w}_2 \; \ldots \; \mathbf{w}_M]$$

Training of multi-class SVM

- Let us define a margin-based loss for a data point $\mathbf{x}_t$ as

$$l\big(\mathbf{W};(\mathbf{x}_t, y_t)\big) = \max\Big(0, 1 + \max_{i \in \mathcal{Y} \setminus y_t} g(i, \mathbf{x}_t) - g(y_t, \mathbf{x}_t)\Big)$$

- Then, we minimize the following overall loss using SGD

$$\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}) \equiv \frac{\lambda}{2}||\mathbf{W}||_F^2 + \frac{1}{T}\sum\nolimits_{t=1}^{T} l\big(\mathbf{W};(\mathbf{x}_t, y_t)\big)$$

Multi-hyperplane Machines (MMs)

- Aiolli & Sperduti (2005) extended the model to a fixed number of weights per class

$$\mathbf{W} = \left[\mathbf{w}_{1,1} \ldots \mathbf{w}_{1,b_1} \middle| \mathbf{w}_{2,1} \ldots \mathbf{w}_{2,b_2} \middle| \ldots \middle| \mathbf{w}_{M,1} \ldots \mathbf{w}_{M,b_M}\right]$$

- The per-class score for a data point is now defined as

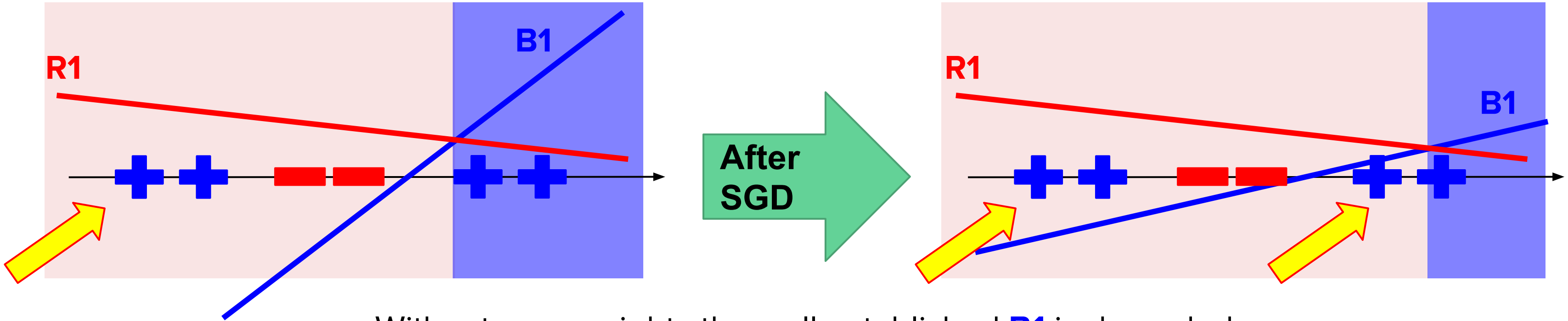
$$g(i, \mathbf{x}) = \cancel{\mathbf{w}_i^\top \mathbf{x}} \text{ (before)} \qquad g(i, \mathbf{x}) = \max_j \mathbf{w}_{i,j}^\top \mathbf{x} \text{ (after)}$$

- The authors propose an approximate method to train the model, solving a convex approximation of the original loss

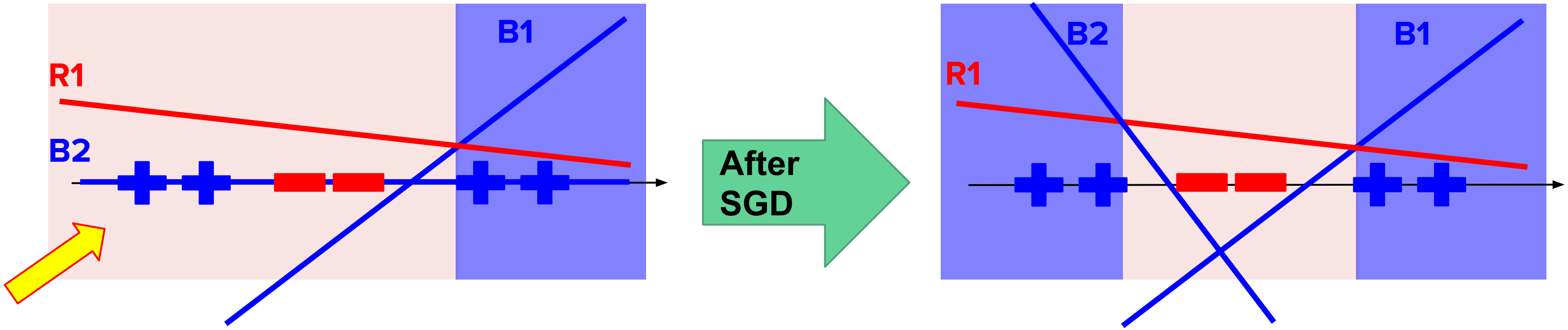
Adaptive MMs (AMMs)

- The authors of (Wang et al., 2011) introduced an infinite number of zero-weights per class, activated during training when neither of the non-zero weights provides positive prediction
- In this way, the model *adapts* to the complexity of the task, and learns an appropriate number of weights

The effect of infinite zero-weights



Without zero weights the well-established **B1** is degraded



With zero-weights a new **B2** weight is activated to cover the misclassified blue cluster

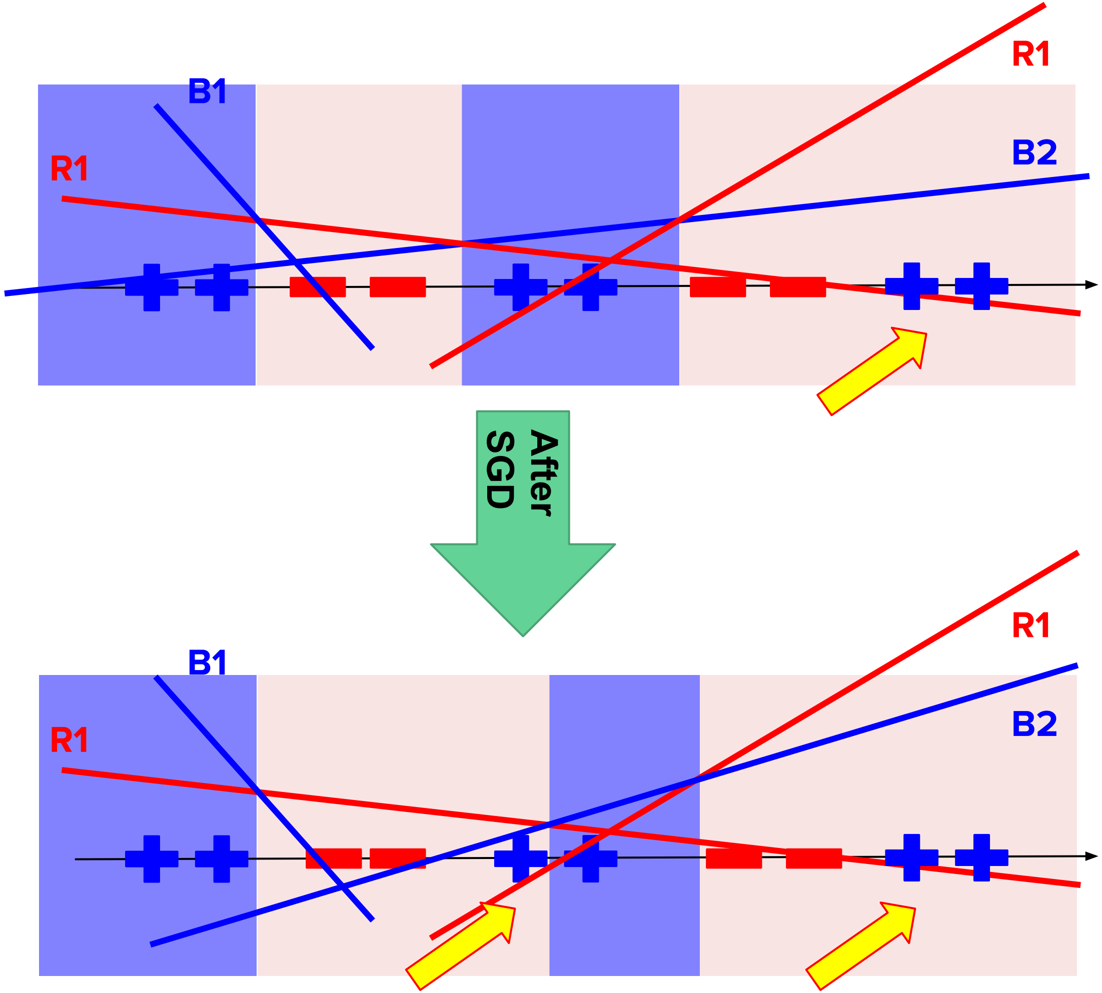
Growing AMMs

- We propose a novel training procedure, where a winning weight is duplicated at random before the SGD update is applied
- GAMM was motivated by the Growing LVQ (Fritzke, 1994; 1995), which introduced a similar strategy for prototype duplication in LVQ method
- Proposed step during gradient step when misclassification happens:

duplication probability ← fixed value (e.g., 0.2)
if (coin flip with duplication probability is successful):

- **duplicate the winning true-class weight and add to the model**
- **update the duplicate weight with gradient step**
- **decrease duplication probability**

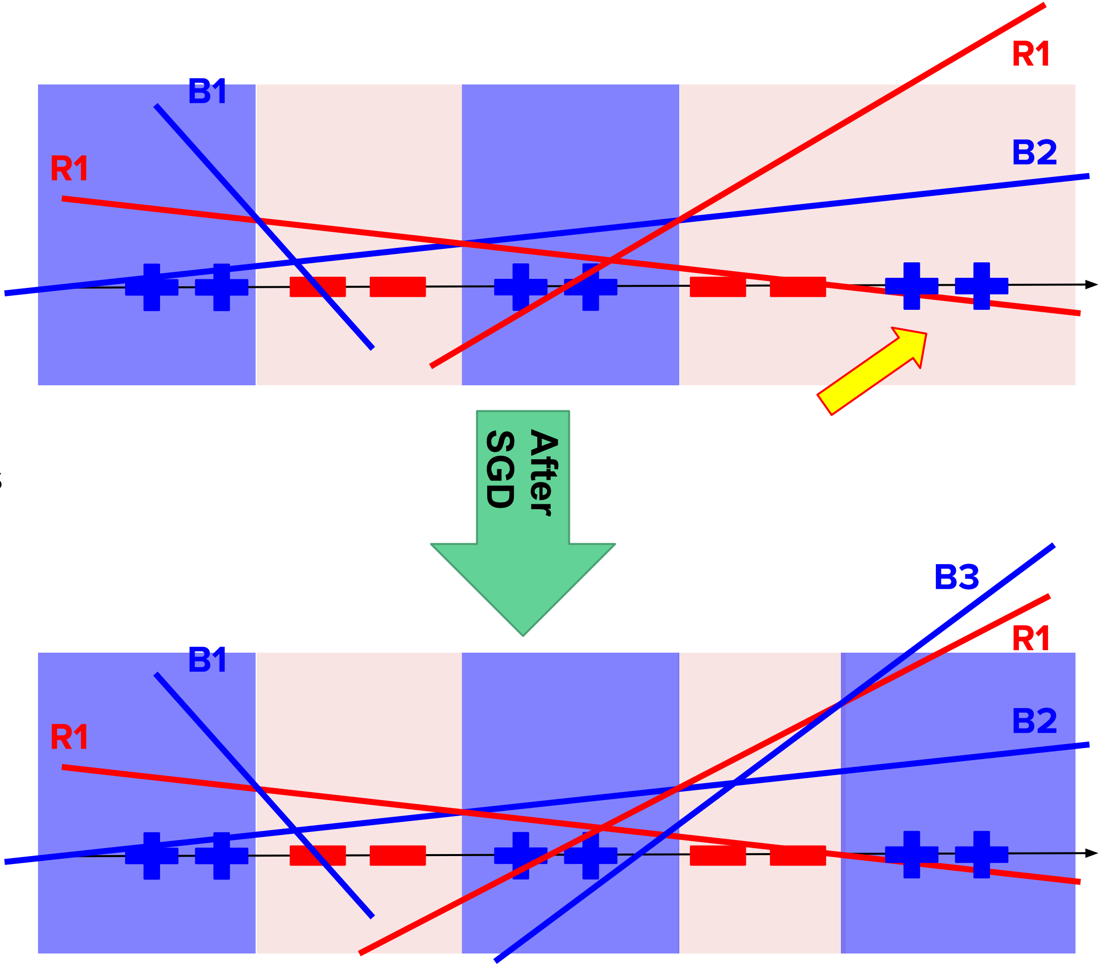
Gradient step w/o duplication



Without duplication the well-established **B2** is degraded

Gradient step with duplication

With duplication a new **B3** weight is spawned from **B2** and updated to cover the misclassified blue cluster



Regret bound

- Let $\mathbf{W}^*$ be the optimal solution of the MM problem, p be an initial duplication probability and β a multiplicative factor that reduces it
- Then, the following regret bound holds

$$\frac{1}{T}\sum_{t=1}^{T}\left(\mathcal{L}^{(t)}(\mathbf{W}^{(t)}|\mathbf{z}) - \mathcal{L}^{(t)}(\mathbf{W}^*|\mathbf{z})\right) \leq \frac{(2+c)^2\left(2+p/(1-\beta)\right)}{\lambda} + \frac{(2+c)^2\left(2+p/(1-\beta)\right)^2}{2T\lambda}\left(\frac{p(2\beta+3)}{(1-\beta)^2} + \ln(T) + 1\right)$$

- Compared to AMM duplication causes an additional regret of $(2+c)2p/((1-\beta)\lambda)$ (in the first term on r.h.s.), as well as regret that vanishes as the iteration count T grows (in the second term on r.h.s.)

Generalization bound

- Define the risk of function $f : \mathbb{R}^D \to \mathcal{Y}$ as

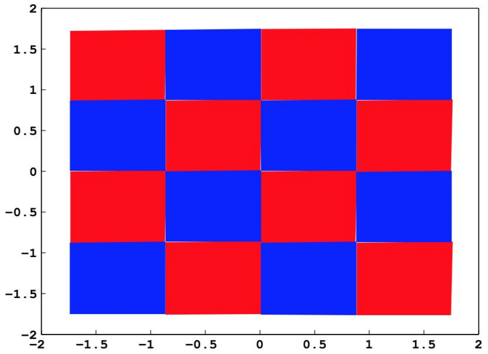
$$R(f) = \mathbb{E}[\ell(y, f(\mathbf{x}))], \text{ where } \ell(y, f(\mathbf{x})) = \mathbb{1}_{y \neq f(\mathbf{x})}$$

- Let $\widetilde{R}_N$ be an empirical risk on an N-sample, $\mathcal{F}$ be a class of functions that MM can implement, and $\|\mathbf{x}\| \leq 1$ without the loss of generality
- Then, with probability of at least $1 - \delta$, the risk of any function $f \in \mathcal{F}$ is bounded from above as (b_i is a number of weights for the i-th class)

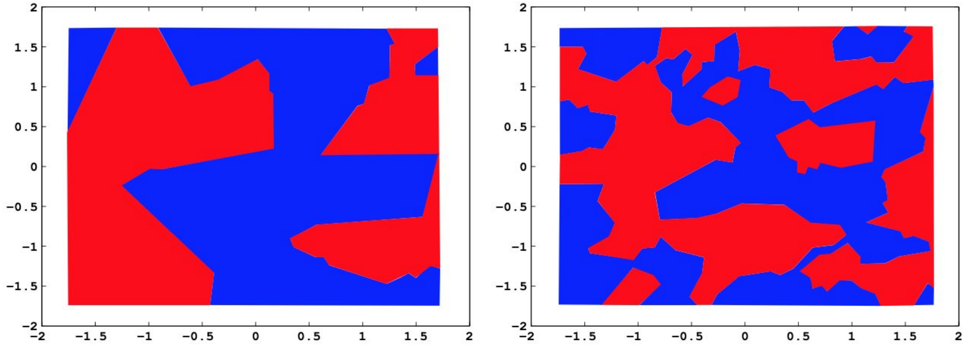
$$R(f) \leq \widetilde{R}_N(f) + \frac{4 + 4K\|\mathbf{W}\|}{\sqrt{N}} + (\|\mathbf{W}\| + 1)\sqrt{\frac{\ln \frac{1}{\delta}}{2N}}, \text{ where } K = \sum_{i=1}^{M} b_i \sum_{j \neq i}^{M} b_j.$$

Experiments on synthetic data

- We considered a number of baselines, including linear SVM, RBF-SVM, AMM, as well as LogitBoost, Random Forest, Neural Nets, and LVQ2.1
- We used *checkerboard* and *weights* data
  - *weights* data was generated by randomly sampling n_w unit weights and assigning a random class to each



4×4 *checkerboard* data

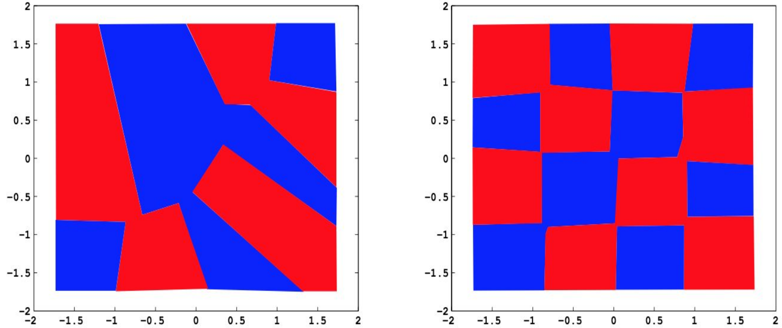


weights data with n_w=50 (left) and n_w=200 (right)

Experiments on *checkerboard* data

- We increased data complexity by varying the *n×m* pattern
- We can see that the GAMM performs significantly better than AMM

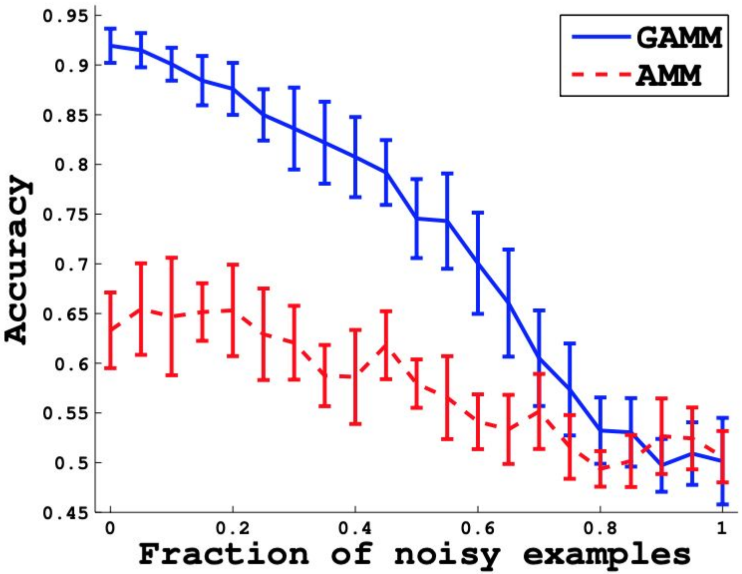
Pattern	AMM	GAMM
1×2	0.48 ± 0.33	0.42 ± 0.29
1×3	1.39 ± 0.72	1.33 ± 0.62
1×4	3.23 ± 0.70	2.45 ± 0.51
1×5	4.90 ± 0.97	3.73 ± 0.71
2×2	1.49 ± 0.57	1.08 ± 0.40
2×3	2.98 ± 0.85	2.20 ± 0.52
2×4	10.33 ± 6.77	3.77 ± 0.89
2×5	20.79 ± 7.42	5.90 ± 1.61
3×3	16.92 ± 9.72	4.17 ± 0.77
3×4	24.47 ± 7.42	5.69 ± 0.93
3×5	34.87 ± 4.84	7.97 ± 1.43
4×4	35.87 ± 3.39	7.38 ± 1.53
4×5	35.13 ± 5.26	13.11 ± 1.84
5×5	43.27 ± 3.71	22.15 ± 2.08



4×4 solution of AMM (left) and GAMM (right)

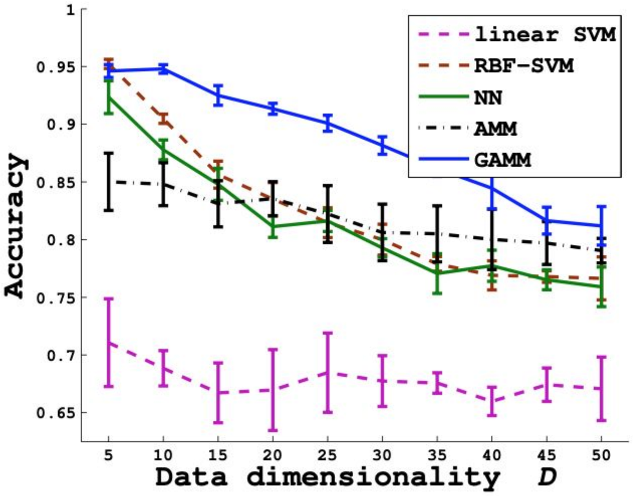
Experiments on *checkerboard* data

- Adding noise to training data does not impact GAMM as much as AMM
- The result shows that GAMM is much more robust to noise

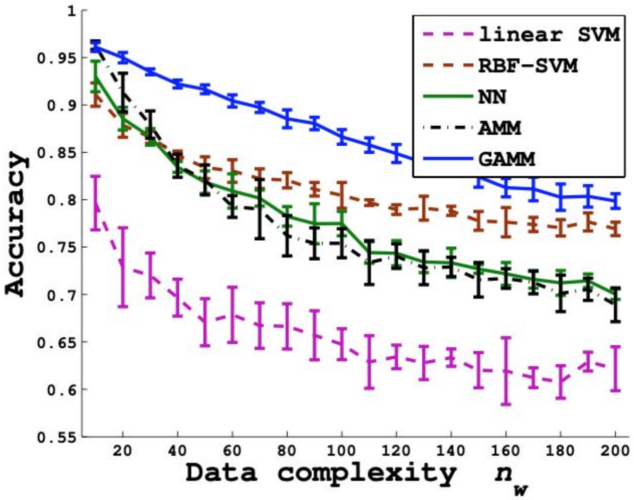


Experiments on *weights* data

- We run experiments with different data complexities n_w and dimensionalities D
- The results show very good GAMM performance in both cases, outperforming the baselines as the task is becoming more and more complex



Increasing D as $n_w = 50$



Increasing n_w as $D = 20$

Experiments on real-world data

- We evaluated GAMM on real-world data of different characteristics

	# classes	# train	# dim	linear SVM	RBF-SVM	AMM	GAMM
usps	10	7,291	256	8.38 ± 0.23	4.81	7.32 ± 0.35	6.28 ± 0.19
letter	26	15,000	16	25.84 ± 1.12	2.02	17.47 ± 0.93	11.69 ± 0.47
ijcnn1	2	49,990	22	7.76 ± 0.19	1.31	2.58 ± 0.21	2.16 ± 0.20
rcv1	2	677,399	47,236	2.31 ± 0.03[1]	2.17[1]	2.29 ± 0.09[1]	2.11 ± 0.09[1]
mnist	2	8,000,000	784	24.18 ± 0.39[2]	0.43[2]	3.40 ± 0.33[2]	2.84 ± 0.02[2]

[1] *rcv1* training times: 1.5s (linear SVM); 20.2h (RBF-SVM); 9s (AMM); 32s (GAMM)

[2] *mnist* training times: 1.1min (linear SVM); 4min (AMM); 19min (GAMM); RBF-SVM accuracy reported after 2 days of P-packSVM training on 512 processors (Zhu et al., 2009)

- GAMM consistently outperformed the AMM algorithm, in all cases by a significant margin, while inheriting very favorable scalability of AMM
- Compared to RBF-SVM, on some tasks GAMM comes close while requiring significantly less time to train

Thank you for your attention!

- ***Key takeaway:***
  GAMM occupies an important niche: computational cost comparable to linear SVMs with accuracy approaching that of kernel SVMs

Test it yourself at https://github.com/djurikom/BudgetedSVM !