Trading Representability for Scalability:

Adaptive Multi-Hyperplane Machine for Nonlinear Classification

Zhuang Wang @ Siemens Corporate Research

Nemanja Djuric @ Temple University
Koby Crammer @ The Technion
Slobodan Vucetic @ Temple University

SIEMENS

The Need for Large-scale Learning Algorithms

SIEMENS

- Cheap, pervasive and networked computing devices are now enhancing our ability to collect data to an even greater extent.
  - The size of datasets grows.

considered large 10 years ago is no longer large by current standard.

Datasets	#train	#test	#dim	#class	non-zero %	file size	domain
a9a	32,561	16,281	123	2	11.3%	2M	social survey
ijcnn	49,990	91,701	22	2	66.7%	7.5M	time series
webspam	280,000	70,000	254	2	41.9%	327M	web text
rcv1_bin	677,399	20,242	47,236	2	0.2%	35M	news text
url	1,976,130	420,000	3,231,961	2	0.004%	1.7G	Internet data
mnist8m_bin	8,000,000	10,000	784	2	19.3%	18G	OCR images
mnist8m_mc	8,000,000	10,000	784	10	19.3%	18G	OCR images

- How to learn/mine on such large data? We are seeking for something with
  - fast training time; fast prediction time; low memory consuming; the state-of-art accuracy; consistent results; simple implementation; theoretically sound…
  - Linear SVM is fast but sub-optimal on the data is non-linear
  - Kernel SVM has more powerful model but computationally expensive

SIEMENS

Filling the Representability and Scalability Gap

- How we achieve this?

Adaptive Multi-hyperplane Machine (AMM)

Multi-hyperplane Machine
(Aiolli & Sperduti, JMLR 05')

Multi-Class SVM
(Crammer & Singer, JMLR 01')

Recent advance on linear classification

Careful implementation

SIEMENS

Multi-Class Linear SVM

- Notation: $S = \{(\mathbf{x}_n, y_n), n = 1, ..., N\}$, $\mathbf{x}_n \in \mathbb{R}^D$, $y_n \in \mathcal{Y} = \{1, ..., M\}$
- Multi-Class classifier $f : \mathbb{R}^D \rightarrow \mathcal{Y}$

$$f(\mathbf{x}) = \text{argmax}_{i \in \mathcal{Y}} g(i, \mathbf{x})$$

where $g(i, \mathbf{x}) = \mathbf{w}_i^T \mathbf{x}$ is the hyperplane parameterized by the weight vector $\mathbf{w}_i \in \mathbb{R}^D$

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

- Optimization

$$\min_{\mathbf{W}} P(\mathbf{W}) \equiv \frac{\lambda}{2} ||\mathbf{W}||^2 + \frac{1}{N} \sum_{n=1}^{N} l(\mathbf{W}; (\mathbf{x}_n, y_n))$$

where $$l(\mathbf{W}; (\mathbf{x}_n, y_n)) = \max \left(0, 1 + \max_{i \in \mathcal{Y} \setminus y_n} g(i, \mathbf{x}_n) - g(y_n, \mathbf{x}_n)\right)$$

Maximal prediction from the incorrect classes

The prediction from the correct class

Classes	weights	prediction
1	$\mathbf{w}_1$	$\mathbf{w}_1^T\mathbf{x}$
2	$\mathbf{w}_2$	$\mathbf{w}_2^T\mathbf{x}$
3	$\mathbf{w}_3$	$\mathbf{w}_3^T\mathbf{x}$

SIEMENS

Multi-Hyperplane Machine

- MM Classifier (multi-hyperplane per class):

$$f(\mathbf{x}) = \text{argmax}_{i \in \mathcal{Y}} g(i, \mathbf{x}), \text{ where } g(i, \mathbf{x}) = \max_j \mathbf{w}_{i,j}^T \mathbf{x},$$

- Optimization

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

$$\min_{\mathbf{W}} P(\mathbf{W}) \equiv \frac{\lambda}{2} ||\mathbf{W}||^2 + \frac{1}{N} \sum_{n=1}^{N} l(\mathbf{W}; (\mathbf{x}_n, y_n))$$

where $$l(\mathbf{W}; (\mathbf{x}_n, y_n)) = \max \left(0, 1 + \max_{i \in \mathcal{Y} \setminus y_n} g(i, \mathbf{x}_n) - g(y_n, \mathbf{x}_n) \right)$$

Maximal prediction from the incorrect classes

The **maximal** prediction from the correct class

3 $\mathbf{w}_{3,1}$ $\mathbf{w}_{3,1}^T\mathbf{x}$

3 $\mathbf{w}_{3,2}$ $\mathbf{w}_{3,2}^T\mathbf{x}$

Non-convex Optimization!

SIEMENS

MM Training Algorithm

- Solve a series of convex approximation (by replacing the non-convex loss function by its convex upper bound)

$$\min_{\mathbf{W}} P(\mathbf{W}|\mathbf{z}) \equiv \frac{\lambda}{2}||\mathbf{W}||^2 + \frac{1}{N}\sum_{n=1}^{N} l_{cvx}(\mathbf{W}; (\mathbf{x}_n, y_n); z_n)$$

where

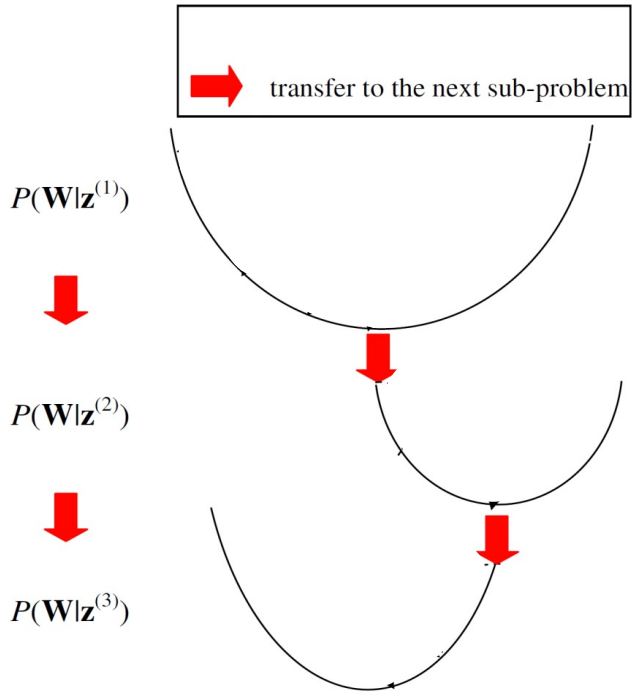
Example specific; fixed during optimization.

$$l_{cvx}(\mathbf{W}; (\mathbf{x}_n, y_n); z_n) = \max\left(0, 1 + \max_{i \in \mathcal{Y} \setminus y_n} g(i, \mathbf{x}_n) - \mathbf{w}_{y_n, z_n}^T \mathbf{x}_n\right)$$

- z is being recalculated after solving each approximated problem as

$$\mathbf{z}^{(r+1)} = \arg\min_{\mathbf{z}} P(\mathbf{W}^*|\mathbf{z})$$

where $\mathbf{W}^* = \arg\min_{\mathbf{W}} P(\mathbf{W}|\mathbf{z}^{(r)})$



SIEMENS

Adaptive Multi-hyperplane Machine (AMM)

- Solve each convex sub-problem by Stochastic Gradient Descent (SGD)
  - An iterative algorithm. At (t+1)-th iteration, updates the previous weight as

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \eta^{(t)} \nabla^{(t)} \text{ where } \nabla^{(t)} = \nabla_{\mathbf{w}^{(t)}} P^{(t)}(\mathbf{W}^{(t)}|\mathbf{z})$$

Instantaneous objective:

$$P^{(t)}(\mathbf{W}|\mathbf{z}) \equiv \frac{\lambda}{2}||\mathbf{W}||^2 + l_{cvx}(\mathbf{W}; (\mathbf{x}_t, y_t); z_t)$$

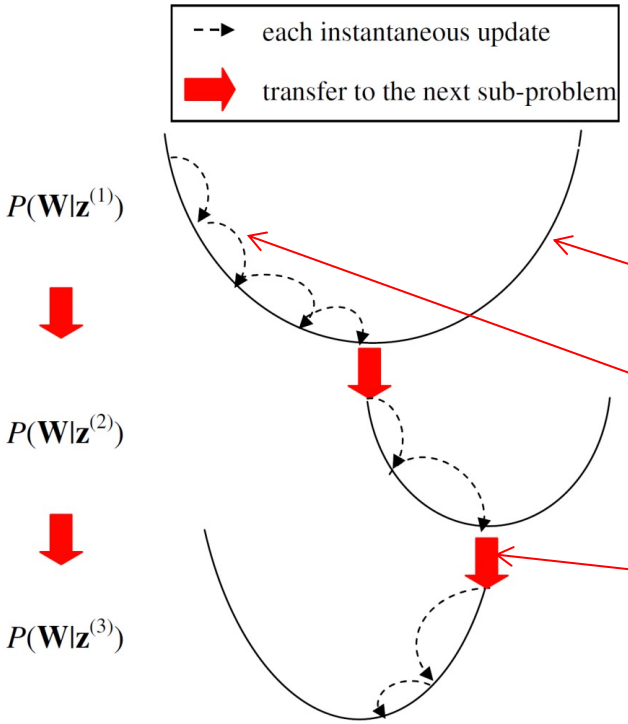
- convergence (with high probability) of each sub-problem

$$P(\mathbf{W}^{(T)}|\mathbf{z}) \leq P(\mathbf{W}^*|\mathbf{z}) + \frac{8(\ln(T)+1)}{\delta\lambda T}$$

- Adaptive property
  - initialize weight matrix as a zero matrix (i.e. setting the number of weights per class to infinity) and leave the update rule to decide which becomes non-zero during the training

SIEMENS

Batch AMMs



Algorithm 1 Training Algorithm for AMM

Input: Training set S, the regularization parameter λ
Output: $\mathbf{W}^{(t)}$
Initialize: $\mathbf{W}^{(1)} = \mathbf{0}$, $t = 1, r = 1$;

1: initialize $\mathbf{z}^{(1)}$; /* *Build 1-st sub-problem* $P(\boldsymbol{W}|\boldsymbol{z}^{(1)})$ */
2: **repeat**
3: /* *Solve each sub-problem* $P(\boldsymbol{W}|\boldsymbol{z}^{(r)})$*: lines* $4 \sim 7$*/
4: **repeat**
5: $(\mathbf{x}_t, y_t) \leftarrow t$-th example from S;
6: compute $\mathbf{W}^{(++t)}$ using (11);
7: **until** (enough epochs)
8: compute $\mathbf{z}^{(++r)}$ using (9); /* *Reassign* $\boldsymbol{z}$ */
9: **until** ($\mathbf{z}^{(r+1)} == \mathbf{z}^{(r)}$ or enough epochs)

- Advantages of AMM over MM
  - fast and scalable
  - no need to pre-specify the number of weights per class
  - simple-implementation

SIEMENS

Online AMMs

- The main difference between the batch and online versions of AMM is that the online version does not wait for convergence of each sub-problem but changes the assignment *z* after every update of SGD

Algorithm 2 Online Algorithm

Input: Training set S, the regularization parameter λ
Output: $\mathbf{W}^{(t)}$
Initialize: $\mathbf{W}^{(1)} = \mathbf{0}$, $t = 1$

1: **repeat**
2: $(\mathbf{x}_t, y_t) \leftarrow t$-th example from S;
3: calculate $\mathbf{z}_t$ by (10);
4: update $\mathbf{W}^{(++t)}$ by (11);
5: **until** (data set exhausted)

- Lose convergence but computationally more appealing and accurate enough

SIEMENS

Weight Pruning

- Motivation
  - Theoretical side: with high probability, we can upper bound the generalization error of AMM (when data is separable by the model) as

$$\frac{130}{N}\left(||\mathbf{W}||^2 B \log(4eN)\log(4N) + \log(\frac{2(2N)^K}{\delta})\right)$$

related to #weights

lower generalization bound can be achieved by reducing # weights.

  - Practical side: less weights can speed up both training and prediction and reduce model size.

- This suggestion AMM should attempt to use as few nonzero weights as possible.

SIEMENS

Update Rule with Pruning

- SGD update rule with a pruning step

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \eta^{(t)} \nabla^{(t)} \text{ followed by } \mathbf{W}^{(t+1)} \leftarrow \mathbf{W}^{(t+1)} - \Delta\mathbf{W}^{(t)}$$

where pruning is performed periodically after *k* rounds and only on the weights so that $\|\Delta\mathbf{W}^{(t)}\| \leq c/((t-1)\lambda)$

- Consequence on SGD optimization

$$\frac{1}{T}\sum_{t=1}^{T} P^{(t)}(\mathbf{W}^{(t)}|\mathbf{z}) - \frac{1}{T}\sum_{t=1}^{T} P^{(t)}(\mathbf{W}^{*}|\mathbf{z}) \leq \frac{(8+c)(\ln(T)+1)}{\lambda T} + \frac{2(4+c)c}{k\lambda}$$

average instantaneous loss between the W obtained by SGD with pruning and the optimal solution W*

Vanish when *T* is large

Upper bound on the gap

SIEMENS

Experiments

☐ Datasets

Datasets	#train	#test	#dim	#class	non-zero %	file size	domain
a9a	32,561	16,281	123	2	11.3%	2M	social survey
ijcnn	49,990	91,701	22	2	66.7%	7.5M	time series
webspam	280,000	70,000	254	2	41.9%	327M	web text
rcv1_bin	677,399	20,242	47,236	2	0.2%	35M	news text
url	1,976,130	420,000	3,231,961	2	0.004%	1.7G	Internet data
mnist8m_bin	8,000,000	10,000	784	2	19.3%	18G	OCR images
mnist8m_mc	8,000,000	10,000	784	10	19.3%	18G	OCR images

☐Competing algorithms

- Linear SVM by Pegasos (Shalev-Shwartz et al., ICML 07')
- Polynomial 2 SVM by Liblinear (Chang et al., JMLR 10')
- RBF SVM by LibSVM (Chang & Lin, 01'), LaSVM (Bortes et al., JMLR 05') & P-packSVM (Zhu et al., ICDM 09'), whichever is applicable.
- didn't include MM as it is not scalable to any of these datasets

☐ Parameters

- One hyper-parameter for Linear SVM, Poly2 SVM & AMM; two for RBF SVM.
- AMM uses 5 epochs to train and uses the pruning step.

SIEMENS

Error Rate, Training & Prediction Time

- AMMs find a better tradeoff between error rate and training and prediction time

Table 3: Error rate and training time comparison with large-scale algorithms (RBF SVM is solved by LibSVM unless specified otherwise. Poly2 and LibSVM results are from [5]).

Datasets	Error rate (%)					Training time (seconds)[1]				
	AMM *batch*	AMM *online*	Linear (Pegasos)	Poly2 SVM	RBF SVM	AMM *batch*	AMM *online*	Linear (Pegasos)	Poly2 SVM	RBF SVM
a9a	15.03±0.11	16.44±0.23	15.04±0.07	14.94	14.97	2	0.2	1	2	99
ijcnn	2.40±0.11	3.02±0.14	7.76±0.19	2.16	1.31	2	0.1	1	11	27
webspam	4.50±0.24	6.14±1.08	7.28±0.09	1.56	0.80	80	4	12	3,228	15,571
mnist_bin	0.53±0.05	0.54±0.03	2.03±0.04	NA	0.43[2]	3084	300	277	NA	2 days[2]
mnist_mc	3.20±0.16	3.36±0.20	8.41±0.11	NA	0.67[3]	13864	1200	1180	NA	8 days[3]
rcv1_bin	2.20±0.01	2.21±0.02	2.29±0.01	NA	NA	1100	80	25	NA	NA
url	1.34±0.21	2.87±1.49	1.50±0.39	NA	NA	400	24	100	NA	NA

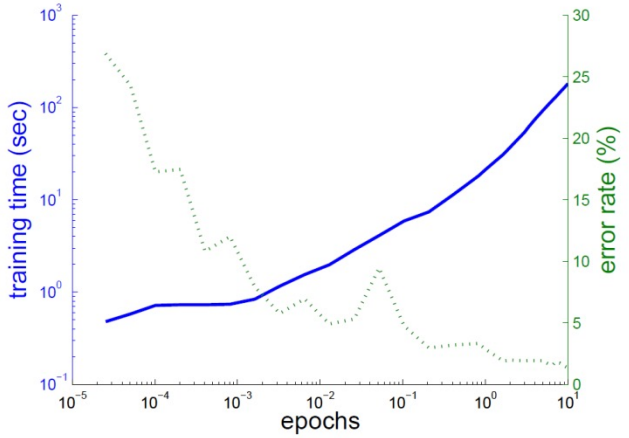
[1] excludes data loading time.
[2] achieved by parallel training P-packSVMs on 512 processors; results from [28].
[3] achieved by LaSVM; results from [12].

Table 4: The number of weights in the classifiers

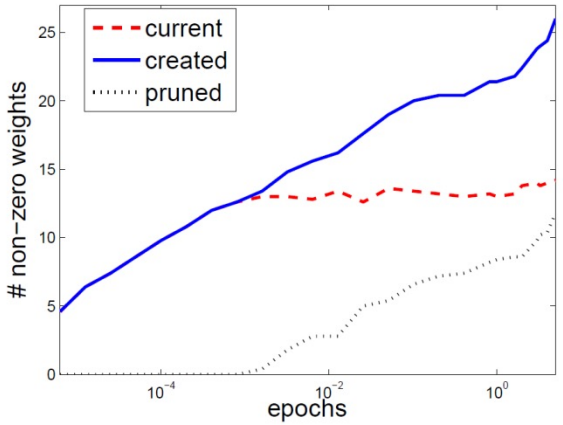
Datasets	a9a	ijcnn	webspam	mnist8m_bin	mnist8m_mc	rcv1	url
batch AMM	11±1	11±1	13±2	20±1	65±2	22±2	4±0
online AMM	16±2	15±1	10±1	13±1	61±3	44±5	5±1

SIEMENS

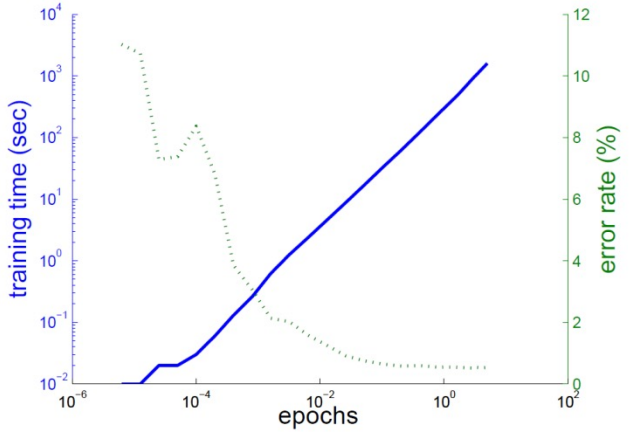
Training Online AMM in Batch Mode



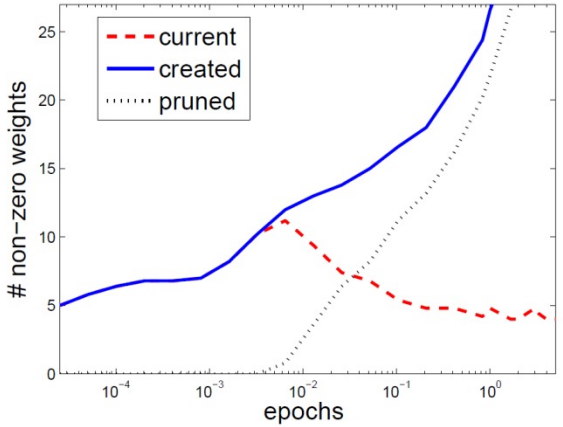
(a)



(b)



(c)



(d)

SIEMENS

Conclusions

☐ We proposed a multi-class classification algorithm AMM that has several favorable features:

- fast training and prediction, small model size, simple implementation, ability to represent nonlinear concepts, and theoretical justification of their properties.

☐ Experimental results on truly large data show AMM filling the scalability and representability gap between linear and kernel SVMs

☐ AMM could be an attractive option for large-scale classification. Our software is available at www.dabi.temple.edu/~vucetic/AMM.html

☐ Future work

- AMM represents non-linear concepts using multiple linear weights. Can we study these linear weights for non-linear feature selection?

Thank you for your time!