Efficient SVM Optimization without an Optimizer

James Kwok

Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Hong Kong

Joint work with Ivor Tsang, Andras Kocsor

November 2007, Nanjing

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

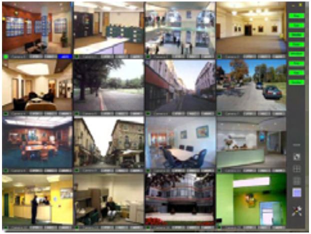
Outline

1. Introduction
2. Core Vector Machine
3. Ball Vector Machine
4. Experiments
5. Conclusion

Popularity of Kernel Methods

Supervised learning: classification / regression

- e.g., text classification Google YAHOO!
- e.g., face detection (video surveillance, digital camera)



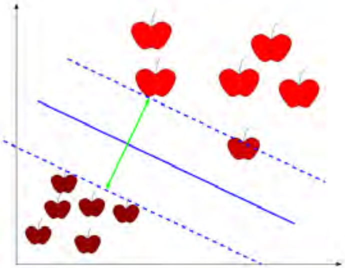
standard digital camera: 10M pixels

Kernel method: Support vector machines (SVM) / support vector regression

Support Vector Machines (SVM)

Classification problem:

- training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^m, \mathbf{x}_i \in \mathbb{R}^d, y_i \in \{\pm 1\}$ (labels)

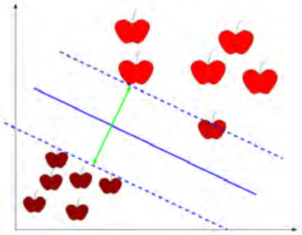


Large-margin method: Maximize the margin separating opposite classes

Maximizing the Margin

Let the (linear) classifier be $\mathbf{w}'\mathbf{x} + b$

$$
\begin{aligned}
\min \quad & \frac{1}{2}\|\mathbf{w}\|^2 (\text{primal}) \\
\text{s.t.} \quad & \mathbf{w}'\mathbf{x}_i + b \geq 1, \ \ \text{if } y_i = 1, \\
& \mathbf{w}'\mathbf{x}_i + b \leq -1, \ \ \text{if } y_i = -1 \\
\max \quad & \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \mathbf{x}_i' \mathbf{x}_j \\
\text{s.t.} \quad & \sum_{i=1}^{m} \alpha_i y_i = 0, \quad \alpha_i \geq 0 \ \ (\text{dual}) \\
& (\alpha_i : \text{Lagrange multiplier})
\end{aligned}
$$



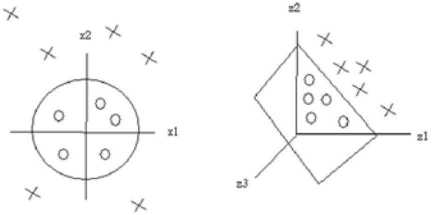
Quadratic programming (QP) problem (globally optimal solution)

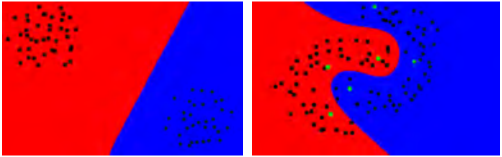
Support vectors: patterns with $\alpha_i > 0$

Kernel Trick

Classifier: linear $\longrightarrow$ nonlinear

- map the data from input space to feature space $\mathcal{F}$ using φ





Only inner products in $\mathcal{F}$ are needed: $\varphi(\mathbf{x}_i)'\varphi(\mathbf{x}_j) \longrightarrow \underbrace{k(\mathbf{x}_i, \mathbf{x}_j)}_{\text{kernel}}$

SVM Optimization

Needs a QP solver

Problem 1

Needs $O(m^2)$ memory just to write down $m \times m$ kernel matrix $= [k(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^{m}$ (m training examples)

- If $m = 20,000$ and it takes 4 bytes to represent a kernel entry, we would need 1.6Gbytes to store the kernel matrix

Problem 2

Involves inverting the kernel matrix $\rightarrow$ $O(m^3)$ time

Key observation

Near-optimal approximate solutions are often good enough in practical applications

Core Vector Machine (CVM) [Tsang, Kwok, Cheung 2005]

1. Formulate kernel methods as minimum enclosing ball problems
2. Obtain approximately optimal solutions efficiently with the use of core-sets

Classification

- one/two-class CVM [Tsang, Kwok & Cheung, (JMLR) 2005]
- one-class classification with Bregman divergence [Nock & Nielsen, (ECML) 2005]
- cluster based CVM [Asharaf, Murty & Shevade, (ICDM) 2006]
- multiclass CVM [Asharaf, Murty & Shevade, (ICML) 2007]

Regression

- core vector regression [Tsang, Kwok & Lai, (ICML) 2005]

Semi-supervised learning

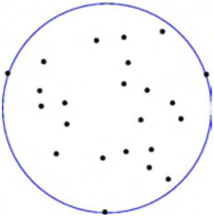
- sparsified LapCVM [Tsang & Kwok, (NIPS) 2006]

Others

- coreset learning [Har-Peled, Roth & Zimak, (IJCAI) 2007]
- feature extraction [Tsang, Kocsor & Kwok, (KDD) 2006]

Minimum Enclosing Ball (MEB) $\Leftrightarrow$ SVM

A problem in Computational Geometry



Given $\mathcal{S} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, minimum enclosing ball of $\mathcal{S}$ (MEB($\mathcal{S}$)):

- the smallest ball $B(\mathbf{c}, R)$ that contains all $\mathbf{x}$'s in $\mathcal{S}$

$$
\begin{aligned}
\text{(primal)} \quad & \min_{R,\mathbf{c}} \ R^2 \\
& \text{s.t.} \quad \|\mathbf{c} - \varphi(\mathbf{x}_i)\|^2 \leq R^2, \quad i = 1, \ldots, m \\
\text{(dual)} \quad & \max_{\boldsymbol{\alpha}} \ \boldsymbol{\alpha}' \text{diag}(\mathbf{K}) - \boldsymbol{\alpha}' \mathbf{K} \boldsymbol{\alpha} \\
& \text{s.t.} \quad \boldsymbol{\alpha}' \mathbf{1} = 1, \ \boldsymbol{\alpha} \geq \mathbf{0}
\end{aligned}
$$

- $\boldsymbol{\alpha} = [\alpha_i, \ldots, \alpha_m]'$: Lagrange multipliers
- $\mathbf{K}_{m \times m} = [k(\mathbf{x}_i, \mathbf{x}_j)]$: kernel matrix
- $\mathbf{0} = [0, \ldots, 0]'$, $\mathbf{1} = [1, \ldots, 1]'$

MEB $\Leftrightarrow$ SVM...

$$\text{Assume } k(\mathbf{x},\mathbf{x}) = \kappa, \quad \text{a constant} \tag{1}$$

Holds for

1. isotropic kernel $k(\mathbf{x},\mathbf{y}) = K(\|\mathbf{x}-\mathbf{y}\|)$ (e.g., Gaussian)
2. dot product kernel $k(\mathbf{x},\mathbf{y}) = K(\mathbf{x}'\mathbf{y})$ (e.g., polynomial) with normalized inputs
3. any normalized kernel $k(\mathbf{x},\mathbf{y}) = \frac{K(\mathbf{x},\mathbf{y})}{\sqrt{K(\mathbf{x},\mathbf{x})}\sqrt{K(\mathbf{y},\mathbf{y})}}$

Combine with $\boldsymbol{\alpha}'\mathbf{1} = 1$, we have $\boldsymbol{\alpha}'\text{diag}(\mathbf{K}) = \kappa$

$$\max_{\boldsymbol{\alpha}} -\boldsymbol{\alpha}'\mathbf{K}\boldsymbol{\alpha} \;\;:\;\; \boldsymbol{\alpha} \geq \mathbf{0}, \;\; \boldsymbol{\alpha}'\mathbf{1} = 1 \tag{2}$$

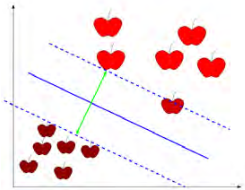
Conversely, whenever the kernel k satisfies (**??**),

Any QP of the form in (**??**) $\leftrightarrow$ a MEB problem

Example: Two-Class SVM

$$\boxed{\max_{\boldsymbol{\alpha}} \ -\boldsymbol{\alpha}'\mathbf{K}\boldsymbol{\alpha} \quad : \quad \boldsymbol{\alpha}'\mathbf{1} = 1, \ \boldsymbol{\alpha} \geq \mathbf{0}}$$

$$\{\mathbf{z}_i = (\mathbf{x}_i, y_i)\}_{i=1}^{m}$$



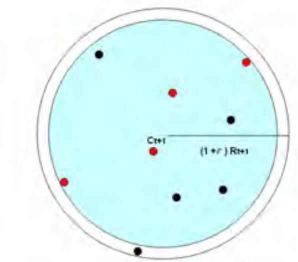
$$(\text{primal}) \quad \min_{\mathbf{w},b,\rho,\xi_i} \quad \|\mathbf{w}\|^2 + b^2 - 2\rho + C\sum_{i=1}^{m} \xi_i{}^2 \quad : \quad y_i(\mathbf{w}'\varphi(\mathbf{x}_i) + b) \geq \rho - \xi_i$$

$$(\text{dual}) \quad \max_{\boldsymbol{\alpha}} \quad -\boldsymbol{\alpha}'\left(\mathbf{K} \odot \mathbf{y}\mathbf{y}' + \mathbf{y}\mathbf{y}' + \frac{1}{C}\mathbf{I}\right)\boldsymbol{\alpha} \ : \ \boldsymbol{\alpha} \geq \mathbf{0}, \quad \boldsymbol{\alpha}'\mathbf{1} = 1$$

$$\tilde{\mathbf{K}} = \left[y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) + y_i y_j + \frac{\delta_{ij}}{C}\right], \quad \text{with} \quad \tilde{k}(\mathbf{z}, \mathbf{z}) = \kappa + 1 + \frac{1}{C} \quad (\text{constant})$$

Approximate MEB Algorithm

- Finding exact MEBs is inefficient for large d

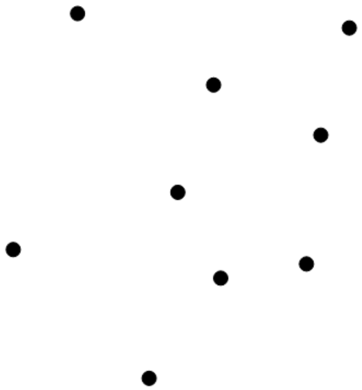


- $(1+\epsilon)$ -approximation
- the $(1+\epsilon)$-expansion of the blue ball contains all the points
- the blue ball is the MEB of the red points (coreset)

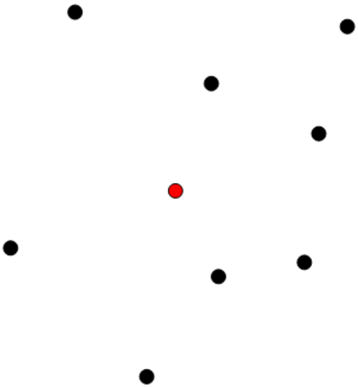
Approximate MEB algorithm [Bădoiu & Clarkson, 2002]

1. At the tth iteration, the current estimate $B(\mathbf{c}_t, r_t)$ is expanded incrementally by including the furthest point outside the $(1+\epsilon)$-ball $B(\mathbf{c}_t, (1+\epsilon)r_t)$
   - we relax it to any point outside $B(\mathbf{c}_t, (1+\epsilon)r_t)$
2. Repeat until all the points in $\mathcal{S}$ are covered by $B(\mathbf{c}_t, (1+\epsilon)r_t)$

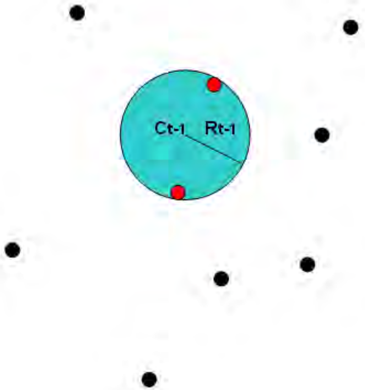
Example



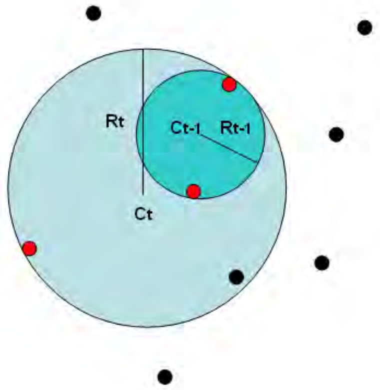
Example



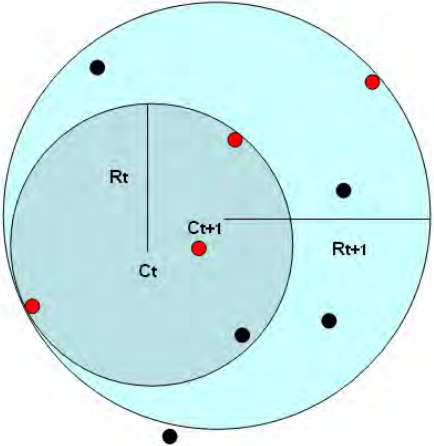
Example



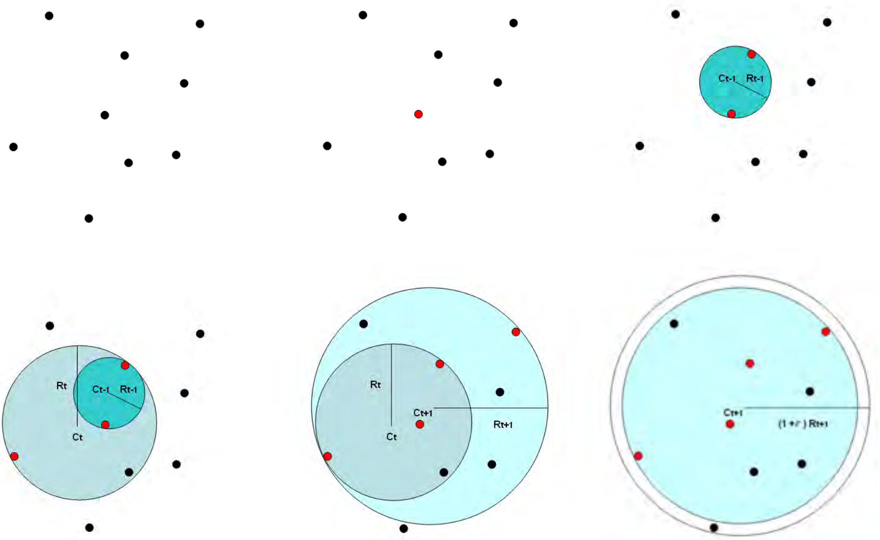
Example



Example



Core Vector Machine (CVM)



Numerical Optimization

CVM algorithm

1. Initialize $\mathbf{c}_0 = \varphi(\mathbf{z}_0)$, $R_0 = 0$ and $\mathcal{S}_0 = \{\varphi(\mathbf{z}_0)\}$.
2. Terminate if no $\varphi(\mathbf{z})$ falls outside $B(\mathbf{c}_t, (1+\epsilon)R_t)$. Otherwise, let $\varphi(\mathbf{z}_t)$ be such a point. Set $\mathcal{S}_{t+1} = \mathcal{S}_t \cup \{\varphi(\mathbf{z}_t)\}$
3. Find MEB($\mathcal{S}_{t+1}$)
4. Increment t by 1 and go back to step 2

Numerical solver is still required in finding MEB($\mathcal{S}_t$)

- QP subproblem
- requires the use of a sophisticated numerical solver for an efficient implementation
  - LIBSVM
- For complicated/very large data sets $\Rightarrow$ internal optimization can be expensive

Question

Can we have a simpler algorithm without using any numerical solver?

Enclosing Ball (EB) Problem

CVM $\leftrightarrow$ minimum enclosing ball

Minimum Enclosing Ball (MEB) Problem

Find the smallest ball $B(\mathbf{c}, r)$ that encloses all the points in $\mathcal{S}$

- some optimization appears inevitable

Enclosing Ball (EB) Problem

Given the radius $r \geq R^*$, find a ball $B(\mathbf{c}, r)$ that encloses all the points in $\mathcal{S}$

- $\|\mathbf{c} - \varphi(\mathbf{z}_i)\|^2 \leq r^2$ for all $\varphi(\mathbf{z}_i)$'s in $\mathcal{S}$

Ball Vector Machine (BVM)

$(1+\epsilon)$-approximation algorithm for EB$(\mathcal{S}, r)$

1: Initialize $\mathbf{c}_0 = \varphi(\mathbf{z}_0)$
2: Terminate if no $\varphi(\mathbf{z})$ falls outside $B(\mathbf{c}_t, (1+\epsilon)r)$. Otherwise, let $\varphi(\mathbf{z}_t)$ be such a point
3: Find the smallest update to the center such that $B(\mathbf{c}_{t+1}, r)$ touches $\varphi(\mathbf{z}_t)$
4: Increment t by 1 and go back to step 2

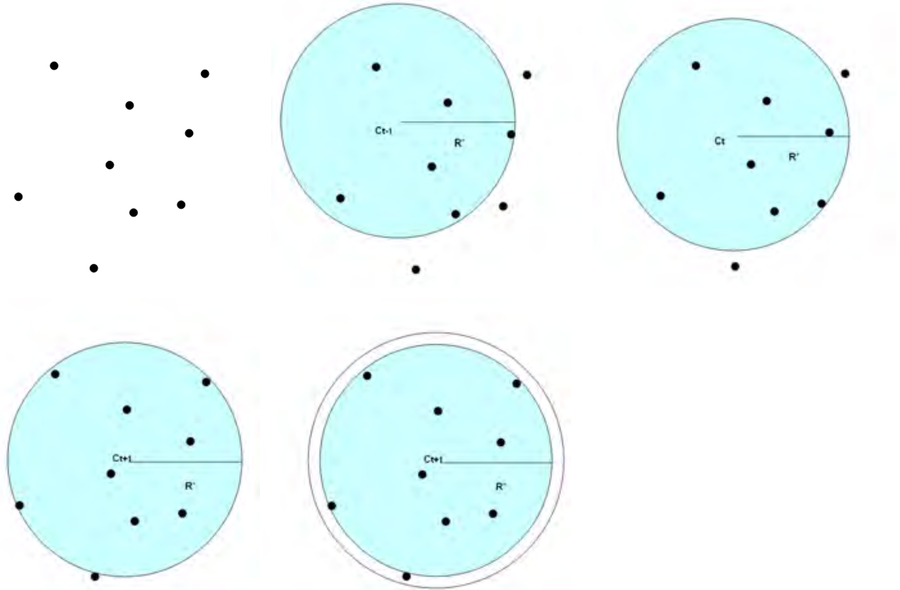
CVM algorithm

1: Initialize $\mathbf{c}_0 = \varphi(\mathbf{z}_0)$, $R_0 = 0$ and $\mathcal{S}_0 = \{\varphi(\mathbf{z}_0)\}$.
2: Terminate if no $\varphi(\mathbf{z})$ falls outside $B(\mathbf{c}_t, (1+\epsilon)R_t)$. Otherwise, let $\varphi(\mathbf{z}_t)$ be such a point. Set $\mathcal{S}_{t+1} = \mathcal{S}_t \cup \{\varphi(\mathbf{z}_t)\}$
3: Find MEB$(\mathcal{S}_{t+1})$
4: Increment t by 1 and go back to step 2

BVM is similar to CVM

- except that the update of the ball's center is different

Example



Efficient Update of the Ball's Center

At the tth iteration, the ball's center is updated such that the new ball just touches $\varphi(\mathbf{z}_t)$

$$\min_{\mathbf{c}} \|\mathbf{c} - \mathbf{c}_t\|^2 \ : \ r^2 \geq \|\mathbf{c} - \varphi(\mathbf{z}_t)\|^2$$

The new center can be obtained analytically as

$$\mathbf{c}_{t+1} = \varphi(\mathbf{z}_t) + \beta_t(\mathbf{c}_t - \varphi(\mathbf{z}_t))$$

- $\beta_t = r/\|\mathbf{c}_t - \varphi(\mathbf{z}_t)\|$
- no numerical optimization solver is needed!

$\mathbf{c}_{t+1}$ is a convex combination of $\mathbf{c}_t$ and $\varphi(\mathbf{z}_t)$

- for any $t > 0$, $\mathbf{c}_t$ is always a linear combination of $\mathbf{c}_0$ and $\mathcal{S}_t = \{\varphi(\mathbf{z}_i)\}_{i=1}^t$
- distance between $\mathbf{c}_{t+1}$ and any pattern $\varphi(\mathbf{z})$ can be computed efficiently

Quality of Prediction

Let $B(\hat{\mathbf{c}}, (1+\epsilon)r)$ be any $(1+\epsilon)$-approximation of $\text{EB}(\mathcal{S}, r)$, then

$$\frac{\|\hat{\mathbf{c}}-\mathbf{c}^*\|}{R^*} \leq \sqrt{\frac{(1+\epsilon)^2 r^2}{{R^*}^2} - 1}$$

Recall that for the L2-SVM:

$$\min_{\mathbf{w},b,\xi_i,\rho} \quad \|\mathbf{w}\|^2 + b^2 - 2\rho + C\sum_{i=1}^{n} \xi_i^2$$

$$\text{s.t.} \qquad y_i(\mathbf{w}'\varphi(\mathbf{x}_i) + b) \geq \rho - \xi_i, \quad i = 1, \ldots, n,$$

- $\mathbf{c} = [\mathbf{w}', b, \sqrt{C}\xi_1, \ldots, \sqrt{C}\xi_n]'$

For any input $\mathbf{x}$,

- optimal prediction function $f^*(\mathbf{x}) = {\mathbf{w}^*}'\varphi(\mathbf{x}) + b^*$
- approximated prediction function $\hat{f}(\mathbf{x}) = \hat{\mathbf{w}}'\varphi(\mathbf{x}) + \hat{b}$

$$|\hat{f}(\mathbf{x}) - f^*(\mathbf{x})| = \left|(\hat{\mathbf{w}} - \mathbf{w}^*)'\varphi(\mathbf{x}) + (\hat{b} - b^*)\right| \leq \sqrt{\|\hat{\mathbf{c}} - \mathbf{c}^*\|^2}\sqrt{k_{ii} + 1}$$

ϵ small $\Rightarrow \|\hat{\mathbf{c}} - \mathbf{c}^*\|^2$ small $\Rightarrow \hat{f}(\mathbf{x})$ close to $f^*(\mathbf{x})$

Time Complexity

Theorem 1 in [Panigraphy, 2004]

When a point falling outside $B(\mathbf{c}_t, (1+\epsilon)r)$ is picked

- BVM algorithm obtains an $(1+\epsilon)$-approximation of $\text{EB}(\mathcal{S}, r)$ in $O(1/\epsilon^2)$ iterations
- overall time complexity: $O(1/\epsilon^4)$

When the furthest point is picked

- BVM algorithm obtains an $(1+\epsilon)$-approximation of $\text{EB}(\mathcal{S}, r)$ in $O(1/\epsilon)$ iterations
- computing such a point takes $O(m|\mathcal{S}_t|)$
- overall time complexity: $O(m/\epsilon^2)$
  $\Rightarrow$ computationally expensive for large m

Multi-Scale $(1+\epsilon)$-Approximation Algorithm for $\text{EB}(\mathcal{S}, r)$

Idea

- Instead of using a small ϵ from the very beginning, start with a much larger $\epsilon' = 2^{-1}$
- After an $(1+\epsilon')$-approximation of $\text{EB}(\mathcal{S}, r)$ has been obtained, reduce ϵ' by half and repeated until $\epsilon' = \epsilon$

Assume that $2^{-1} \geq \epsilon = 2^{-M}$ for some positive integer M

1: Initialize $\mathbf{c}_{\text{EB}_0} = \varphi(\mathbf{z}_0)$.
2: For $m = 1$ to M do
3: Set $\epsilon_m = 2^{-m}$. Find $(1+\epsilon_m)$-approximation of $\text{EB}(\mathcal{S}, r)$ using BVM Algorithm, with $\mathbf{c}_{\text{EB}_{m-1}}$ as warm start

In finding the EB, only requires $\varphi(\mathbf{z}_t)$ to be outside $B(\mathbf{c}_t, (1+\epsilon)r)$

$\Rightarrow$ avoid expensive distance computations

Multi-Scale Approximation Algorithm...

Converges in at most $3\log_2\frac{1}{\epsilon} + \left(1 - \frac{{R^*}^2}{r^2}\right)\frac{1}{\epsilon^2} + \frac{6}{\epsilon}$ iterations

$r = R^*$

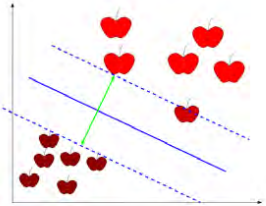
- number of iteration is $O(1/\epsilon)$
- time complexity $O(1/\epsilon^2)$
- space complexity $O(1/\epsilon)$

$r \longrightarrow R^*$

- the $1/\epsilon^2$ term becomes negligible
- number of iteration is close to $O(1/\epsilon)$

cf. CVM takes $O(1/\epsilon^8)$ time and $O(1/\epsilon^2)$ space

How to Set r in the SVM Setting?



primal

$$\min \quad \|\mathbf{w}\|^2 + b^2 - 2\rho + C\sum_{i=1}^m \xi_i{}^2$$
$$\text{s.t.} \quad y_i(\mathbf{w}'\varphi(\mathbf{x}_i) + b) \geq \rho - \xi_i$$

dual

$$\max \quad -\boldsymbol{\alpha}' \left(\mathbf{K} \odot \mathbf{yy}' + \mathbf{yy}' + \frac{1}{C}\mathbf{I}\right)\boldsymbol{\alpha}$$
$$\text{s.t.} \quad \boldsymbol{\alpha} \geq \mathbf{0}, \quad \boldsymbol{\alpha}'\mathbf{1} = 1$$

$$\tilde{k}_{ij} = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) + y_i y_j + \frac{\delta_{ij}}{C}$$

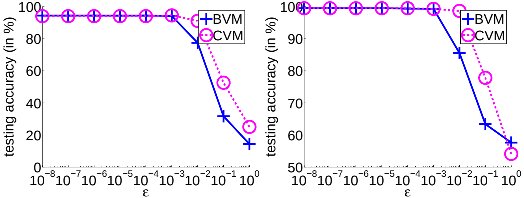
- $\tilde{k}_{ii} = k_{ii} + \frac{1}{C} \equiv \tilde{\kappa}$. Set $r = \sqrt{\tilde{\kappa}}$

$\sqrt{\tilde{\kappa}} \geq R^*$, where R^* is the radius of the MEB

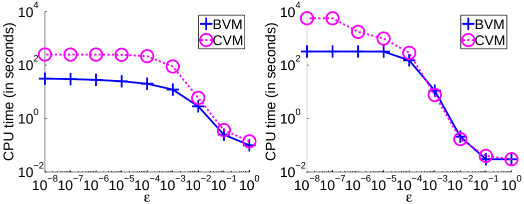
Often, $r = \sqrt{\tilde{\kappa}} \simeq R^*$

- ϵ is small and $r \simeq R^*$, center of this EB problem is close to the center of MEB
- ball's center $\leftrightarrow$ SVM's weight and bias
- the obtained BVM is also close to the desired SVM solution

Varying ϵ ("letter" , "usps")

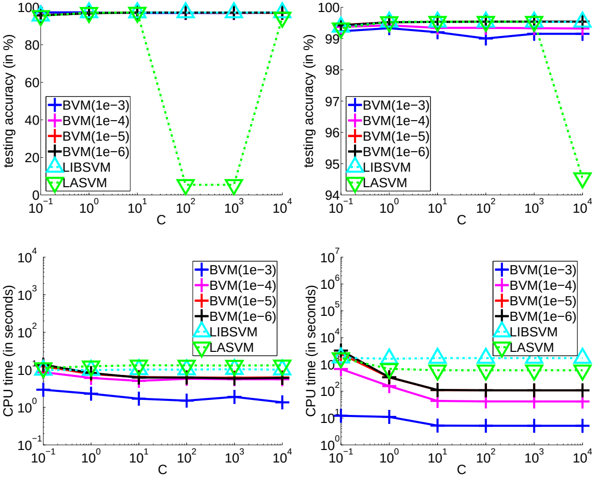


- BVM and CVM have high accuracies for $\epsilon \in [10^{-8} : 10^{-3}]$
- performance deteriorates when ϵ is further increased



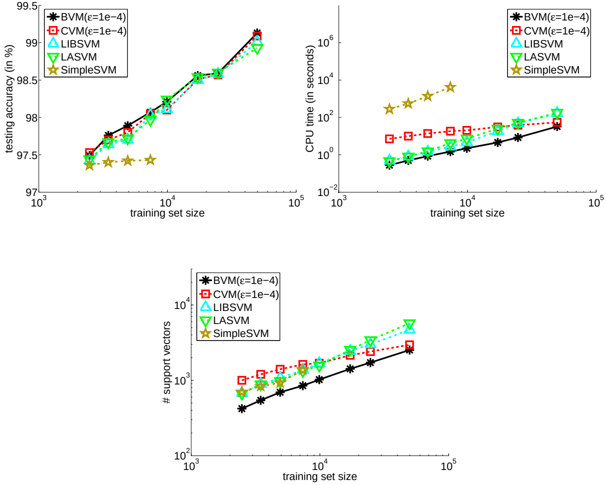
- training time and number of support vectors are stable for $\epsilon \leq 10^{-4}$
- BVM training is faster than CVM

Varying C ("reuters", "usps")



- BVM has almost the same accuracies as LIBSVM
- training time and number of support vectors obtained by BVM (with $\epsilon \geq 10^{-4}$) are comparable with those of LIBSVM

Varying the Training Set Size (“web”)



- Both BVM and CVM have comparable accuracies as the other implementations

Different Kernels ("usps")

		normalized polynomial kernel $d = 2$	normalized polynomial kernel $d = 3$	normalized polynomial kernel $d = 4$	Gaussian kernel	Laplacian kernel
accur. (%)	BVM	99.39	99.54	99.63	99.51	99.36
	CVM	**99.48**	99.54	**99.64**	99.46	99.46
	LIBSVM	99.42	99.60	**99.64**	99.53	99.52
	LASVM	99.43	**99.62**	**99.64**	**99.54**	**99.53**
CPU time	BVM	51.26	**52.67**	**66.16**	124.57	224.74
	CVM	**47.12**	94.42	214.40	**26.47**	**143.61**
	LIBSVM	1,808.28	2,506.49	3,642.42	2,404.75	4,964.87
	LASVM	1,424.28	1,156.23	1,770.63	1,167.86	2,473.22
#SV	BVM	665	**691**	**793**	1,105	1,538
	CVM	**453**	783	1,353	**560**	**1,522**
	LIBSVM	1,428	2,326	3,544	2,050	4,462
	LASVM	933	1,899	3,187	1,624	4,059

Both BVM and CVM are fast and have good performance

Data Sets

data sets	#class	dim	#train patns.	#test patns
optdigits	10	64	3,823	1,797
satimage	6	36	4,435	2,000
w3a	2	300	4,912	44,837
pendigits	10	16	7,494	3,498
reuters	2	8,315	7,770	3,299
letter	26	16	15,000	5,000
web	2	300	49,749	14,951
ijcnn1	2	22	49,990	91,701
extended usps	2	676	266,079	75,383
intrusion	2	127	4,898,431	311,029

Testing Accuracies (in %)

data	BVM	CVM	LIBSVM	LASVM	SimpleSVM
optdigits	96.38	96.38	96.77	N/A	**96.88**
satimage	89.35	89.55	89.55	N/A	**89.65**
w3a	**97.89**	97.80	97.70	97.72	97.42
pendigits	**97.97**	97.85	97.91	N/A	**97.97**
reuters	96.75	96.96	**97.15**	97.09	–
letter	**94.47**	94.12	94.25	N/A	94.23
web	**99.13**	99.09	99.01	98.93	–
ijcnn1	97.58	**98.67**	98.19	98.42	94.10
usps	99.42	99.52	**99.53**	**99.53**	–
intrusion	91.97	**92.44**	–	–	–

- BVM and CVM have accuracies comparable with the other SVM implementations
- Only BVM and CVM (but neither LIBSVM nor LASVM) can work on “intrusion” (with around 5 million training examples)

CPU Time (in sec) used in SVM Training

data	BVM	CVM	LIBSVM	LASVM	SimpleSVM
optdigits	**1.65**	24.86	1.79	N/A	81.15
satimage	1.82	14.81	**1.06**	N/A	221.01
w3a	**0.85**	13.82	1.46	1.54	1384.34
pendigits	1.31	12.10	**0.82**	N/A	41.22
reuters	**6.32**	63.51	9.76	13.81	–
letter	19.87	215.73	**10.85**	N/A	1290.55
web	**32.59**	54.46	168.84	178.73	–
ijcnn1	99.95	62.78	**57.96**	140.25	2201.35
usps	**150.46**	288.96	1578.27	753.09	–
intrusion	0.73	**0.70**	–	–	–

- BVM is usually faster than CVM, and is faster/comparable with the other implementations

Number of Support Vectors

data	BVM	CVM	LIBSVM	LASVM
optdigits	1583	2154	**1306**	N/A
satimage	1956	2333	**1433**	N/A
w3a	**694**	1402	1072	979
pendigits	1990	2827	**1206**	N/A
reuters	**925**	1496	1356	1359
letter	10536	12440	**8436**	N/A
web	**2522**	2960	4674	5718
ijcnn1	4006	**3637**	5700	5525
usps	**1524**	2576	2178	1803
intrusion	99	**51**	–	–

- All obtain comparable numbers of support vectors
- On the large data sets ("reuters", "web", "ijcnn1", "usps" and "intrusion"), CVM and, even better, BVM can have fewer support vectors

MNIST Digits Database

An extended MNIST digit database

- the original training set contains 60,000 patterns with 784 features
- Loosli, Canu & Bottou [2007] extended the training set to 8.1 million patterns by incorporating 135 transformations

Using all 8.1M patterns

	BVM	LASVM
accuracy (%)	98.66	99.33
CPU time (s)	8 hours	(8 days)

Using 1/3 of the training patterns

	BVM	LIBSVM	LASVM
#misclassified patterns	2	3	2
accuracy (in %)	99.91	99.86	99.91
CPU time (s)	238.33	7981.48	1797.43
#support vectors	1,605	2,183	1,618

Conclusion

Enclosing Ball (EB) problem is simpler than Minimum Enclosing Ball (MEB) problem

- update of $\mathbf{c}_t$ does not require any numerical solver
- multi-scale $(1+\epsilon)$-approximation algorithm for faster convergence

$\Rightarrow$ easy to implement

$\Rightarrow$ BVM is faster than CVM

Experimentally,

- BVM's accuracy is comparable with the other SVM implementations
- usually faster than CVM, and is faster/comparable with others
- can handle very large data sets
- can have fewer support vectors