A Divide-and-Conquer Solver for Kernel Support Vector Machines

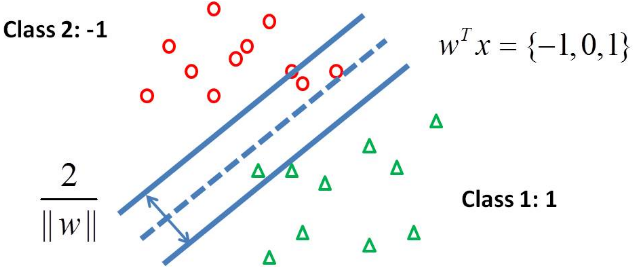
Cho-Jui Hsieh
Dept of Computer Science
UT Austin

ICML
Beijing, China
June 23, 2014

Joint work with S. Si and I. S. Dhillon

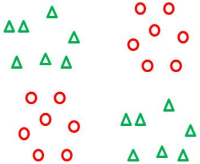
Support Vector Machines (SVM)

- SVM is a widely used classifier.
- Given:
  - Training data points $\mathbf{x}_1, \cdots, \mathbf{x}_n$.
  - Each $\mathbf{x}_i \in \mathbb{R}^d$ is a feature vector:
  - Consider a simple case with two classes: $y_i \in \{+1, -1\}$.
- Goal: Find a hyperplane to separate these two classes of data: if $y_i = 1$, $\mathbf{w}^T\mathbf{x}_i \geq 1 - \xi_i$; $y_i = -1$, $\mathbf{w}^T\mathbf{x}_i \leq -1 + \xi_i$.



Support Vector Machines (SVM)

- What if the data is not linearly separable?



$$x \rightarrow \varphi(x) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}$$

Solution: map data $\mathbf{x}_i$ to higher dimensional(maybe infinite) feature space $\varphi(x_i)$, where they are linearly separable.

- **Kernel trick:** $K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_j)$**.**
- Various types of kernels:
  - Gaussian kernel: $K(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x}-\mathbf{y}\|_2^2}$;
  - Polynomial kernel: $K(\mathbf{x}, \mathbf{y}) = (\gamma \mathbf{x}^T \mathbf{y} + c)^d$.

Support Vector Machines (SVM)

- The dual problem for SVM:

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha},$$
$$\text{s.t. } 0 \leq \alpha_i \leq C, \quad \text{for} \quad i = 1, \ldots, n,$$

  where $Q_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ and $e = [1, \ldots, 1]^T$.
- At optimum: $\mathbf{w} = \sum_i \alpha_i^* y_i \varphi(\mathbf{x}_i)$,

  Prediction: $\mathbf{w}^T \varphi(\hat{\mathbf{x}}) = \sum_i \alpha_i^* y_i K(\mathbf{x}_i, \hat{\mathbf{x}})$.

Support Vector Machines (SVM)

- The dual problem for SVM:

$$\min_{\boldsymbol{\alpha}} \quad \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha},$$
$$\text{s.t. } 0 \le \alpha_i \le C, \quad \text{for} \quad i = 1, \dots, n,$$

  where $Q_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$ and $e = [1, \dots, 1]^T$.

- Challenge for solving kernel SVMs:
  - Space: $O(n^2)$;
  - Time: $O(n^3)$ (assume $O(n)$ support vectors).
- n=Number of variables = number of samples.

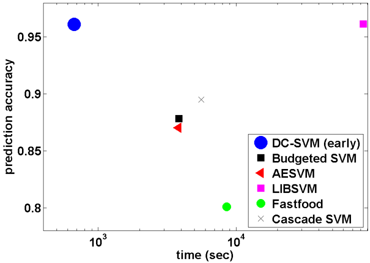
Scalability

- LIBSVM takes more than 8 hours to train on a CoverType dataset with 0.5 million samples (with prediction accuracy 96%).
- Many **inexact** solvers have been developed:

  AESVM (Nadan et al., 2014), Budgeted SVM (Wang et al., 2012), Fastfood (Le et al., 2013), Cascade SVM (Graf et al., 2005), . . .

  1-3 hours, with prediction accuracy $85 - 90\%$.
- Divide the problem into smaller subproblems – **DC-SVM**

  11 minutes, with prediction accuracy 96%.



DC-SVM with a single level – data division

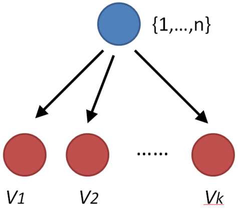
- Partition $\boldsymbol{\alpha}$ into k subsets $\{\mathcal{V}_1, \ldots, \mathcal{V}_k\}$.
- Solve each subproblem independently:

$$\min_{\boldsymbol{\alpha}_{(i)}} \frac{1}{2}(\boldsymbol{\alpha}_{(i)})^T Q_{(i,i)} \boldsymbol{\alpha}_{(i)} - \mathbf{e}^T \boldsymbol{\alpha}_{(i)},$$
$$\text{s.t. } 0 \leq \boldsymbol{\alpha}_{(i)} \leq C,$$

- Approximate solution for the whole problem:

$$\bar{\boldsymbol{\alpha}} = [\bar{\boldsymbol{\alpha}}_{(1)}, \ldots, \bar{\boldsymbol{\alpha}}_{(k)}].$$

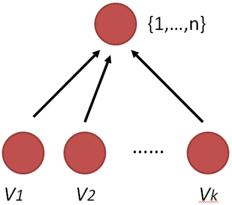
- Space complexity: $O(n^2) \rightarrow O(n^2/k^2)$.
- Time complexity: $O(n^3) \rightarrow O(n^3/k^2)$.



DC-SVM with a single level – conquer step

- Use $\bar{\boldsymbol{\alpha}}$ to initialize a global coordinate descent solver.
- Converges quickly if

  $\|\bar{\boldsymbol{\alpha}} - \boldsymbol{\alpha}^*\|$ is small.
- What clustering algorithm should we use to minimize $\|\bar{\boldsymbol{\alpha}} - \boldsymbol{\alpha}^*\|$?



Quality of $\bar{\alpha}$ (solution from subproblems)

- α^*: solution of SVM with kernel K.
- $\bar{\alpha}$: solution of SVM with

$$\tilde{K}(\mathbf{x}, \mathbf{y}) = I(\pi(\mathbf{x}) = \pi(\mathbf{y}))K(\mathbf{x}, \mathbf{y}),$$

  where $\pi(\cdot)$ is the cluster indicator.
- The error comes from the between-cluster kernels:

$$D(\pi) = \sum_{i,j:\pi(\mathbf{x}_i)\neq\pi(\mathbf{x}_j)} |K(\mathbf{x}_i, \mathbf{x}_j)|.$$

Kernel kmeans clustering

- **Theorem 1**: For a given partition π, the corresponding $\bar{\boldsymbol{\alpha}}$ satisfies

$$0 \leq f(\bar{\boldsymbol{\alpha}}) - f(\boldsymbol{\alpha}^*) \leq (1/2)C^2 D(\pi),$$

  and furthermore,

$$\|\boldsymbol{\alpha}^* - \bar{\boldsymbol{\alpha}}\|_2^2 \leq C^2 D(\pi)/\sigma_n,$$

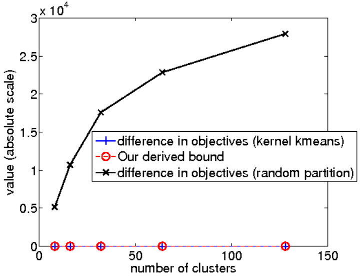
  where σ_n is the smallest eigenvalue of the kernel matrix.
- Want a partition which
  (1) Minimizes $D(\pi) = \sum_{i,j:\pi(\mathbf{x}_i)\neq\pi(\mathbf{x}_j)} |K(\mathbf{x}_i, \mathbf{x}_j)|$.
  (2) Have balanced cluster sizes (for efficient training).
- Use kernel kmeans (but slow).
- Two step kernel kmeans:
  - Run kernel kmeans on a subset of samples with size $m \ll n$ to find cluster centers.
  - Identify the clusters for the rest of data.

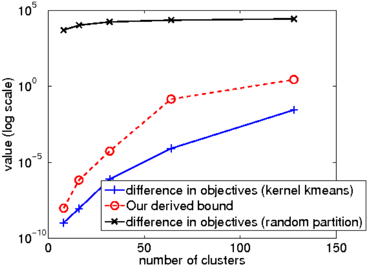
Demonstration of the bound

- **Theorem 1**: For a given partition π, the corresponding $\bar{\alpha}$ satisfies

$$0 \leq f(\bar{\alpha}) - f(\alpha^*) \leq (1/2)C^2 D(\pi).$$

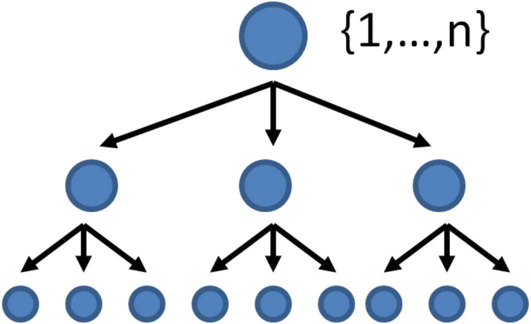
- Covertype dataset with 10000 samples and $\gamma = 32$ (best in cross validation).
- Our data partition scheme leads to a good approximation to the global solution α^*.





DC-SVM with multiple levels

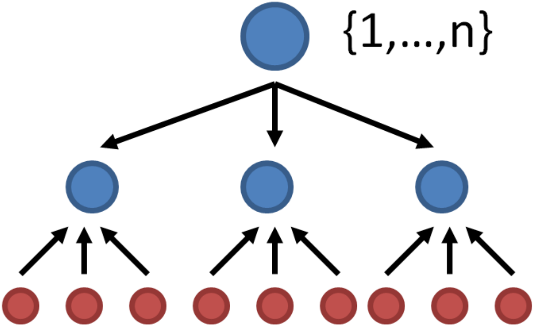
- Run DC-SVM with multiple levels.



Data Division

DC-SVM with multiple levels

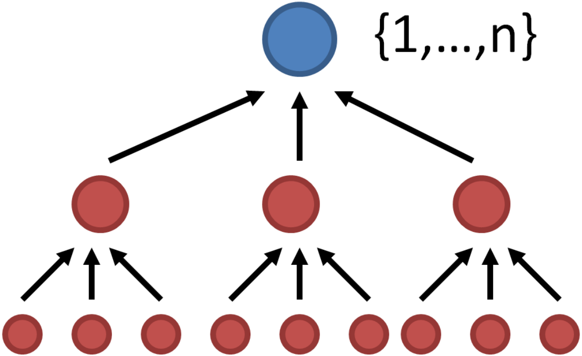
- Run DC-SVM with multiple levels.



Solve the leaf level problems.

DC-SVM with multiple levels

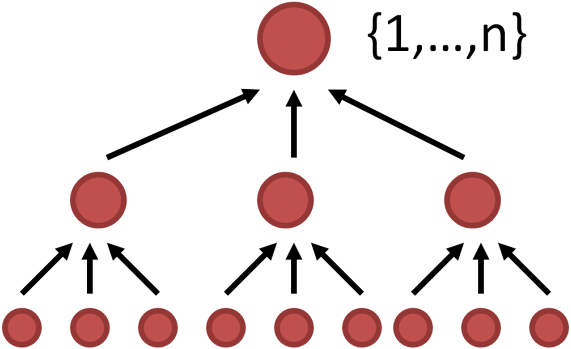
- Run DC-SVM with multiple levels.



Solve the intermediate level problems.

DC-SVM with multiple levels

- Run DC-SVM with multiple levels.



Solve the original problem.

Early Prediction

- An **anytime algorithm** – stop at any level and give the prediction.
- Prediction using the l-th level solution

  faster training time; the prediction accuracy is close to or even better than the global SVM solution.
- Naive way to predict $\hat{\mathbf{x}}$: $\text{sign}(\sum_{i=1}^{n} y_i \bar{\alpha}_i K(\mathbf{x}_i, \hat{\mathbf{x}}))$.
- Prediction by $\tilde{K}$:

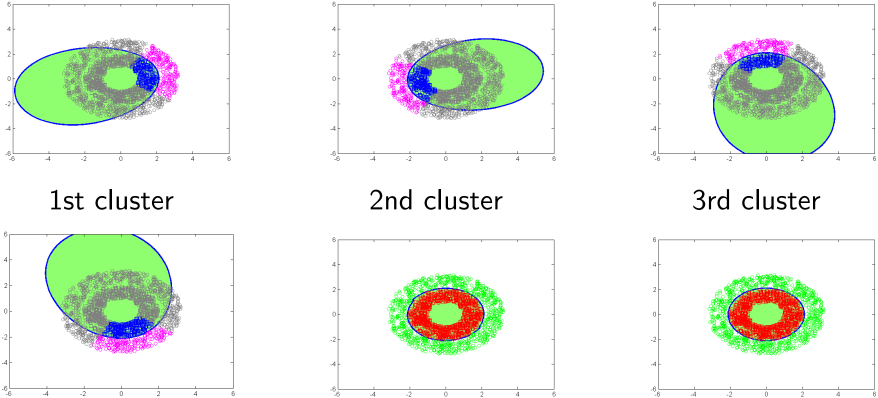
  $\text{sign}(\sum_{i=1}^{n} y_i \bar{\alpha}_i \tilde{K}(\mathbf{x}_i, \hat{\mathbf{x}})) = \text{sign}(\sum_{i \in \mathcal{V}_{\pi(\hat{\mathbf{x}})}} y_i \bar{\alpha}_i K(\mathbf{x}_i, \hat{\mathbf{x}}))$

  Use nearest model to predict; better performance.
- Prediction time reduced from $O(d(\#SV))$ to $O(d(\#SV)/k)$

	webspam $k = 50$	webspam $k = 100$	covtype $k = 50$	covtype $k = 100$
Prediction by K	92.6% / 1.3ms	89.5% / 1.3ms	94.6% / 2.6ms	92.7% / 2.6ms
Prediction by $\bar{K}$	**99.1% / .17ms**	**99.0% / .16ms**	**96.1% / .4ms**	**96.0% / .2ms**

Toy Example

Two Circle Data: each circle is a class; not separable by kernel kmeans.



1st cluster

2nd cluster

3rd cluster

4th cluster

DC-SVM (early)

RBF SVM

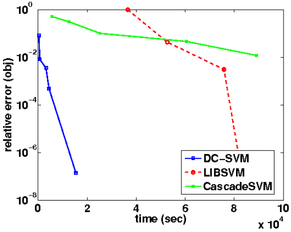
Methods included in comparisons

- DC-SVM: proposed method for solving exact global SVM problem.
- DC-SVM (EARLY): proposed method with early stopping (at 64 clusters).
- LIBSVM (Chang and Lin, 2011)
- CASCADE SVM (Graf et al., 2005)
- FASTFOOD (Le et al., 2013)
- LASVM (Bordes et al., 2005)
- LLSVM (Zhang et al., 2012)
- SPSVM (Keerthi et al., 2006)
- LTPU (Moody and Darken., 1989)
- BUDGETED SVM (Wang et al., 2012; Djuric et al., 2013)
- AESVM (Nandan et al., 2014)

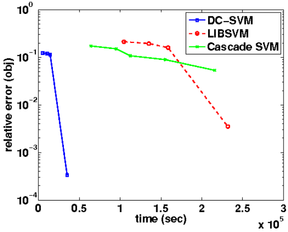
Results with Gaussian kernel.

	webspam		covtype		mnist8m	
	$n = 2.8 \times 10^5, d = 254$		$n = 4.65 \times 10^5, d = 54$		$n = 8 \times 10^6, d = 784$	
	$C = 8, \gamma = 32$		$C = 32, \gamma = 32$		$C = 1, \gamma = 2^{-21}$	
	time(s)	acc(%)	time(s)	acc(%)	time(s)	acc(%)
DC-SVM (early)	**670**	99.13	**672**	96.12	**10287**	99.85
DC-SVM	10485	**99.28**	11414	**96.15**	71823	**99.93**
LIBSVM	29472	**99.28**	83631	96.15	298900	99.91
LIBSVM (subsample)	1267	98.52	5330	92.46	31526	98.95
LaSVM	20342	99.25	102603	94.39	171400	98.95
CascadeSVM	3515	98.1	5600	89.51	64151	98.3
LLSVM	2853	97.74	4451	84.21	65121	97.64
FastFood	5563	96.47	8550	80.1	14917	96.5
SpSVM	6235	95.3	15113	83.37	121563	96.3
LTPU	4005	96.12	11532	83.25	105210	97.82
Budgeted SVM	2194	98.94	3839	87.83	29266	98.8
AESVM	3027	98.90	3821	87.03	16239	96.6

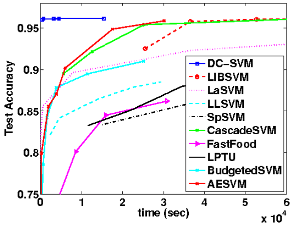
Results with Gaussian kernel



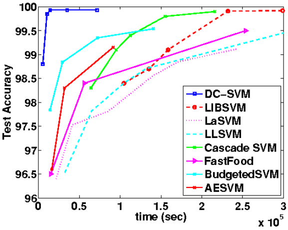
covtype objective function



MNIST8m objective function

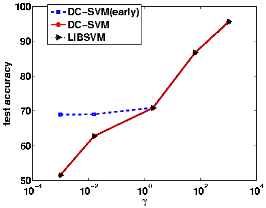


covtype prediction accuracy

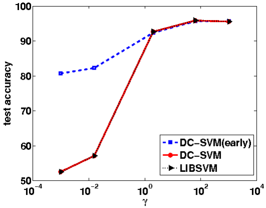


MNIST8m prediction accuracy

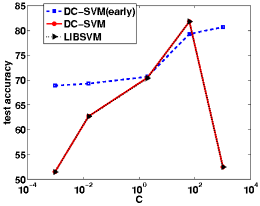
Results with grid of C, γ



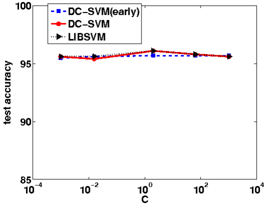
webspam $C = 2^{-10}$



webspam $C = 2^{10}$



webspam $\gamma = 2^{-10}$



webspam $\gamma = 2^{10}$

The results for DC-SVM and LIBSVM coincide with each other because they solve the exact SVM problem.

Conclusions

- We have proposed a novel divide-and-conquer algorithm for solving kernel SVM.
  - Divide the problem into smaller subproblems.
  - Solutions from subproblems are close to the original problem (when using kernel kmeans).
  - Run DC-SVM with multiple levels to solve the original problem.
  - Run DC-SVM with early prediction: yields competitive prediction accuracy 100 times faster than exact SVM solvers.
- Software can be downloaded at
  `http://www.cs.utexas.edu/~cjhsieh/dcsvm`

References

[1] C.-J. Hsieh, S. Si and I. S. Dhillon *A Divide-and-Conquer Solver for Kernel Support Vector Machines*, ICML, 2014.

[2] C.-C. Chang and C.-J. Lin, *LIBSVM: A Library for Support Vector Machines*, ACM TIST, 2011.

[3] H. P. Graf, E. Cosatto, L. Bottou, I. Dundanovic and V. Vapnik, *Parallel Support Vector Machines: The Cascade SVM* , NIPS, 2005.

[4] K. Zhang, L. Lan, Z. Wang and F. Moerchen, *Scaling up Kernel SVM on Limited Resources: A Low-rank Linearization Approach*, AISTATS, 2012.

[5] Q. V. Le, T. Sarlos and A. J. Smola, *Fastfood – Approximating Kernel Expansions in Loglinear Time.* , ICML, 2013.

[6] A. Bordes, S. Ertekin, J. Weston and L. Bottou, *Fast Kernel Classifiers with Online and Active Learning* , JMLR, 2005.

[7] S. S. Keerthi, O. Chapelle and D. DeCoste, *Building Support Vector Machines with Reduced Classifier Complexity* , JMLR, 2006.

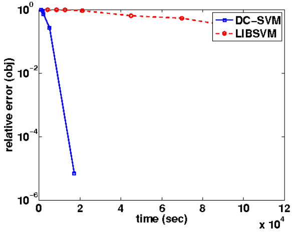
[8] M. Nandan, P. R. Khargonekar and S. S. Talathi, *Fast SVM Training using Approximate Extreme Points.* , JMLR, 2014.

[9] Z. Wang, K. Crammer and S. Vucetic, *Breaking the Curse of Kernelization: Budgeted Stochastic Gradient Descent for Large-scale SVM Training* , JMLR, 2012.

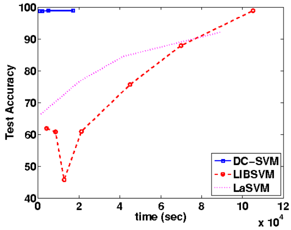
Results with grid of C, γ

dataset	C	γ	DC-SVM (early)		DC-SVM		LIBSVM	
			acc(%)	time(s)	acc(%)	time(s)	acc(%)	time(s)
webspam	2^{-10}	2^{-10}	**86**	**806**	61	26324	61	45984
webspam	2^{-10}	2^{-6}	**83**	**935**	61	22569	61	53569
webspam	2^{-10}	2^{1}	87.1	**886**	**91.1**	10835	**91.1**	34226
webspam	2^{-10}	2^{6}	**93.7**	**1060**	92.6	6496	92.6	34558
webspam	2^{-10}	2^{10}	98.3	**1898**	**98.5**	7410	**98.5**	55574
webspam	2^{-6}	2^{-10}	**83**	**793**	68	24542	68	44153
webspam	2^{-6}	2^{-6}	**84**	**762**	69	33498	69	63891
webspam	2^{-6}	2^{1}	93.3	**599**	**93.5**	15098	93.1	34226
webspam	2^{-6}	2^{6}	**96.4**	**704**	**96.4**	7048	**96.4**	48571
webspam	2^{-6}	2^{10}	98.3	**1277**	**98.6**	6140	**98.6**	45122
webspam	2^{1}	2^{-10}	**87**	**688**	78	18741	78	48512
webspam	2^{1}	2^{-6}	**93**	**645**	81	10481	81	30106
webspam	2^{1}	2^{1}	98.4	**420**	**99.0**	9157	**99.0**	35151
webspam	2^{1}	2^{6}	**98.9**	**466**	**98.9**	5104	**98.9**	28415
webspam	2^{1}	2^{10}	98.3	**853**	**98.7**	4490	98.7	28891
webspam	2^{6}	2^{-10}	**93**	**759**	80	24849	80	64121
webspam	2^{6}	2^{-6}	**97**	**602**	83	21898	83	55414
webspam	2^{6}	2^{1}	98.8	**406**	**99.1**	8051	**99.1**	40510
webspam	2^{6}	2^{6}	**99.0**	**465**	98.9	6140	98.9	35510
webspam	2^{6}	2^{10}	98.3	**917**	**98.7**	4510	**98.7**	34121

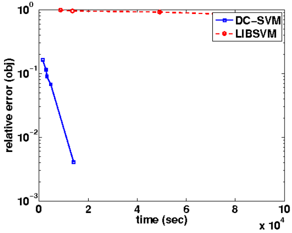
Results for polynomial kernel $K(\mathbf{x}_i, \mathbf{x}_j) = (\eta + \gamma \mathbf{x}_i^T \mathbf{x}_j)^3$



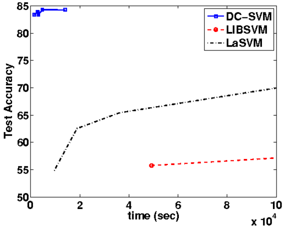
webspam objective function



webspam prediction accuracy



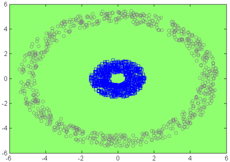
covtype objective function



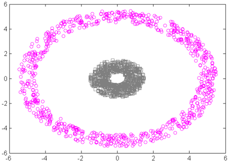
covtype prediction accuracy

Toy Example

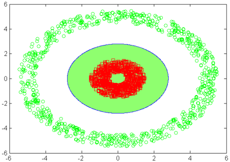
Two Circle Data: each circle is a class; separable by kernel kmeans.



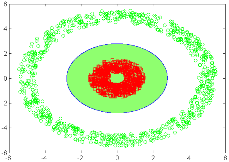
1st cluster



2nd cluster



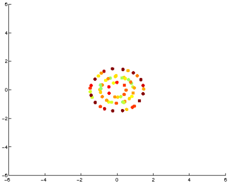
DC-SVM (early)



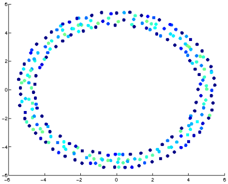
RBF SVM

Toy Example

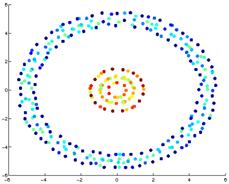
Two Circle Data: each circle is a class; separable by kernel kmeans.



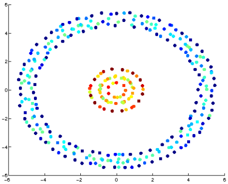
1st cluster



2nd cluster



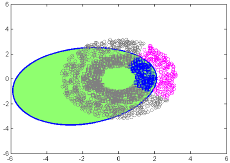
DC-SVM (early)



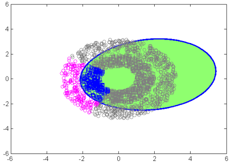
RBF SVM

Toy Example

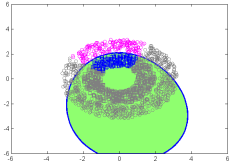
Two Circle Data: not separable by kernel kmeans



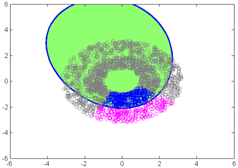
1st cluster



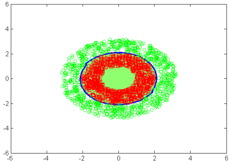
2nd cluster



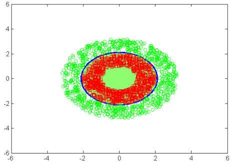
3rd cluster



4th cluster



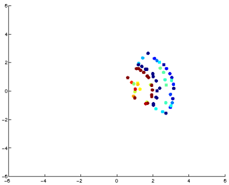
DC-SVM (early)



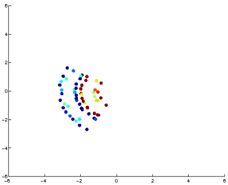
RBF SVM

Toy Example

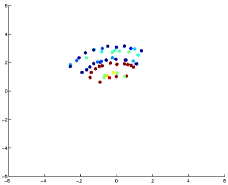
Two Circle Data: not separable by kernel kmeans



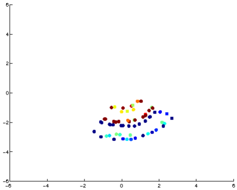
1st cluster



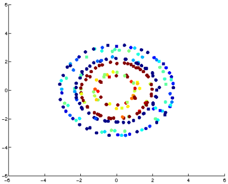
2nd cluster



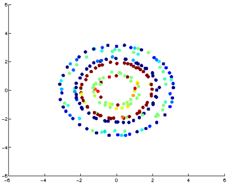
3rd cluster



4th cluster



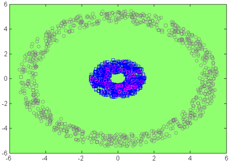
DC-SVM (early)



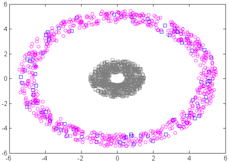
RBF SVM

Toy Example

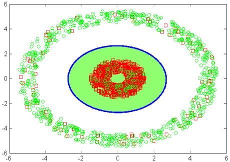
Two Circle Data: separable by kernel kmeans; 10% noise.



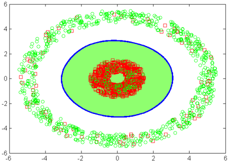
1st cluster



2nd cluster



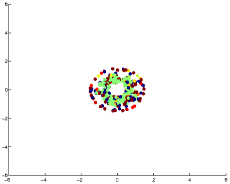
DC-SVM (early)



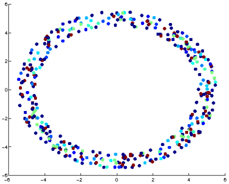
RBF SVM

Toy Example

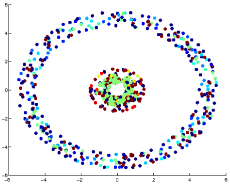
Two Circle Data: separable by kernel kmeans; 10% noise.



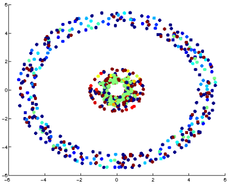
1st cluster



2nd cluster



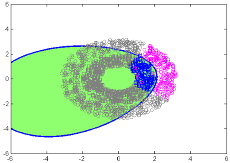
DC-SVM (early)



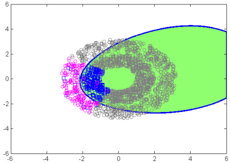
RBF SVM

Toy Example

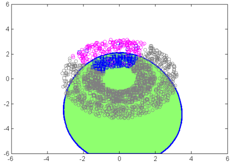
Two Circle Data: not separable by kernel kmeans; 10% noise.



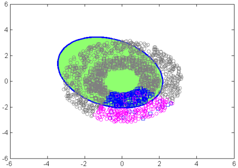
1st cluster



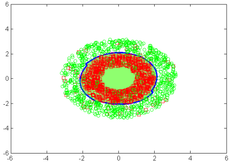
2nd cluster



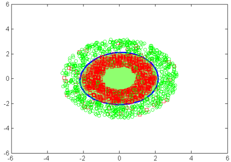
3rd cluster



4th cluster



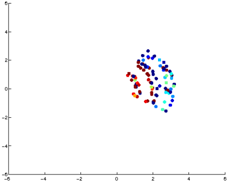
DC-SVM (early)



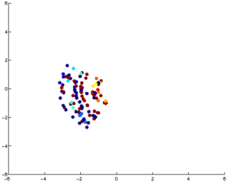
RBF SVM

Toy Example

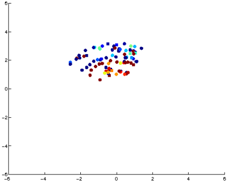
Two Circle Data: not separable by kernel kmeans; 10% noise.



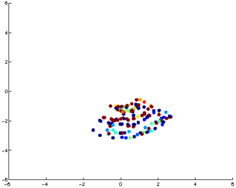
1st cluster



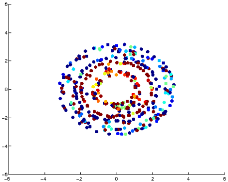
2nd cluster



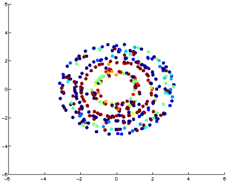
3rd cluster



4th cluster



DC-SVM (early)



RBF SVM