Divide & Conquer Methods for Large-Scale Data Analysis

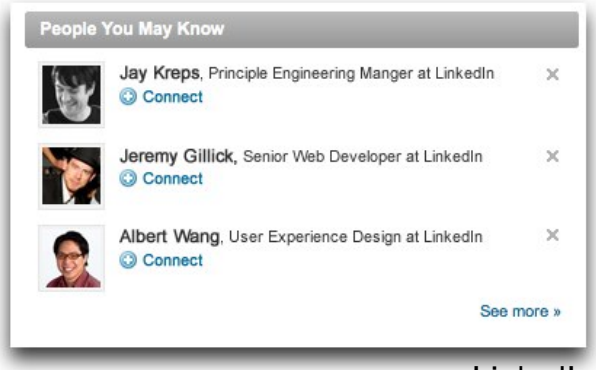
Inderjit S. Dhillon
Dept of Computer Science
UT Austin

International Conference on Machine Learning and Applications
Detroit, MI
Dec 4, 2014

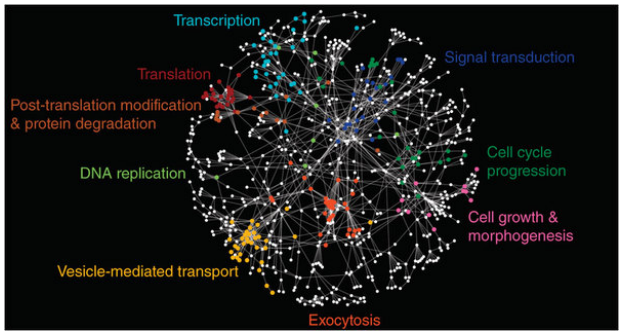
Joint work with C.-J. Hsieh and S. Si

Machine Learning Applications

Link prediction



LinkedIn.



gene-gene network

fMRI

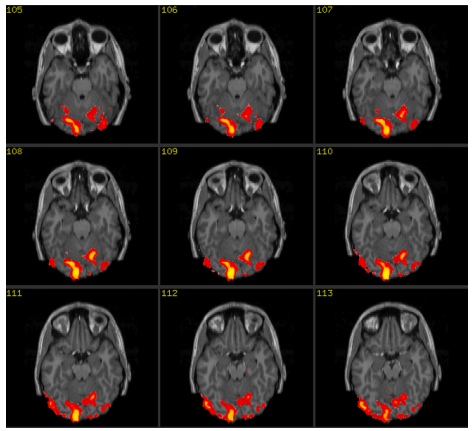
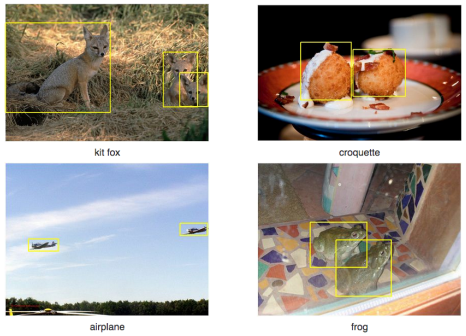
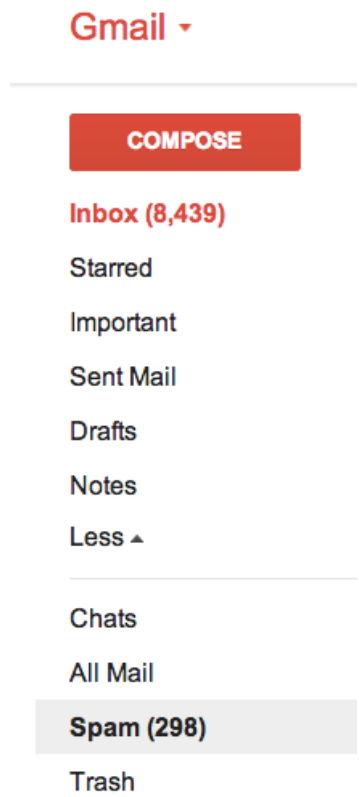


Image classification



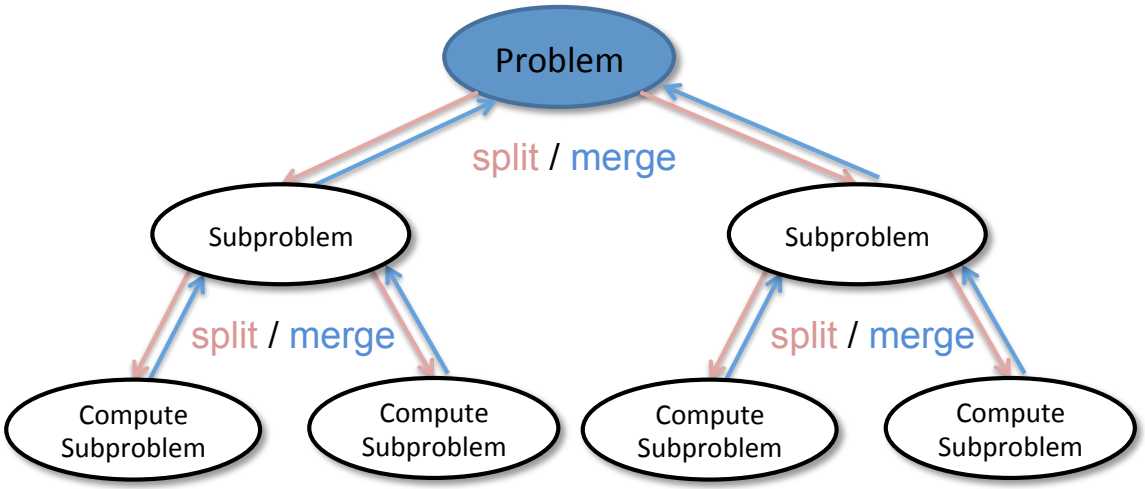
Spam classification

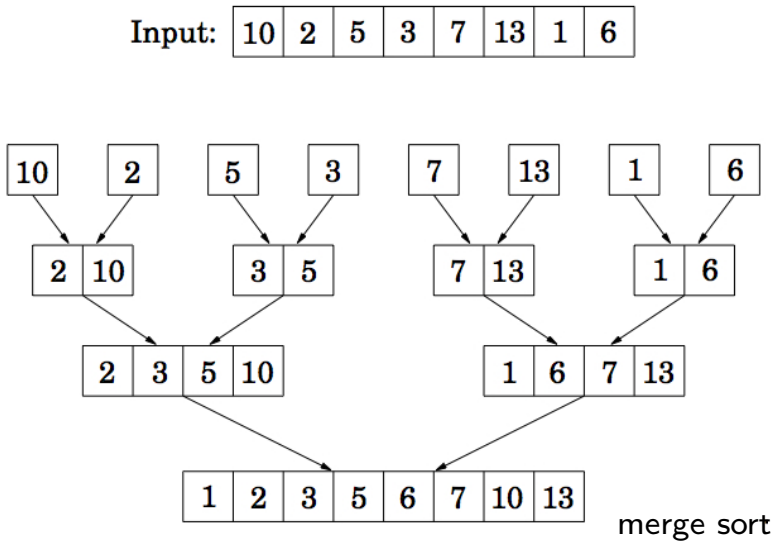


How to develop machine learning algorithms that scale to massive datasets?

Divide & Conquer Method

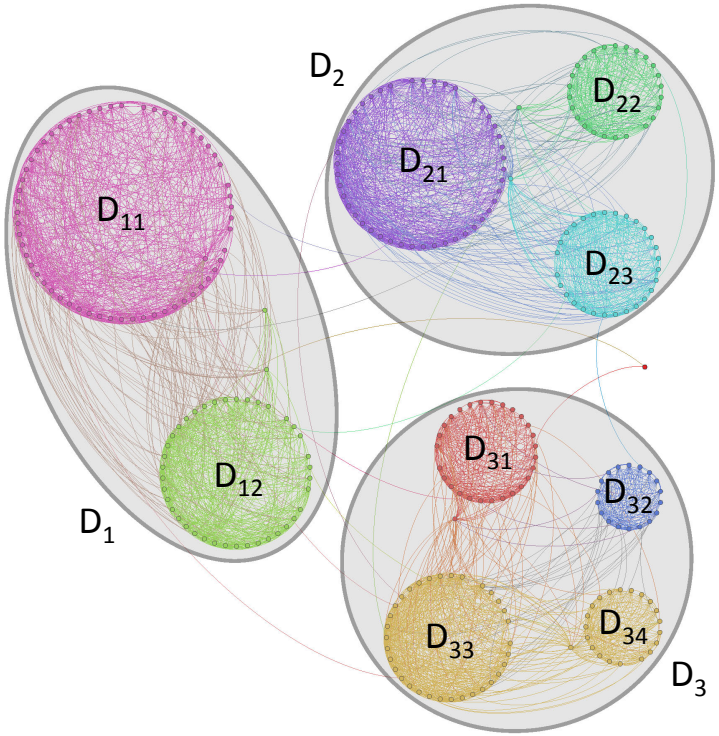
- Divide original (large) problem into smaller instances
- Solve the smaller instances
- Obtain solution to original problem by combining these solutions

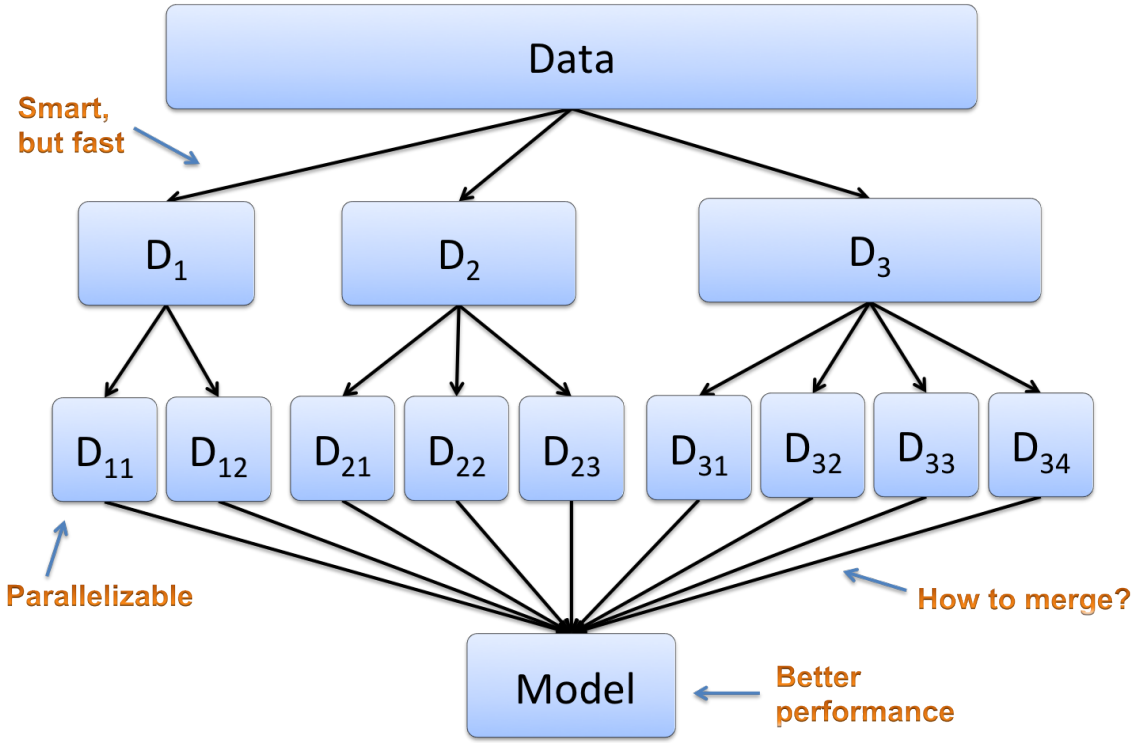




Divide & Conquer Method

Strategy:





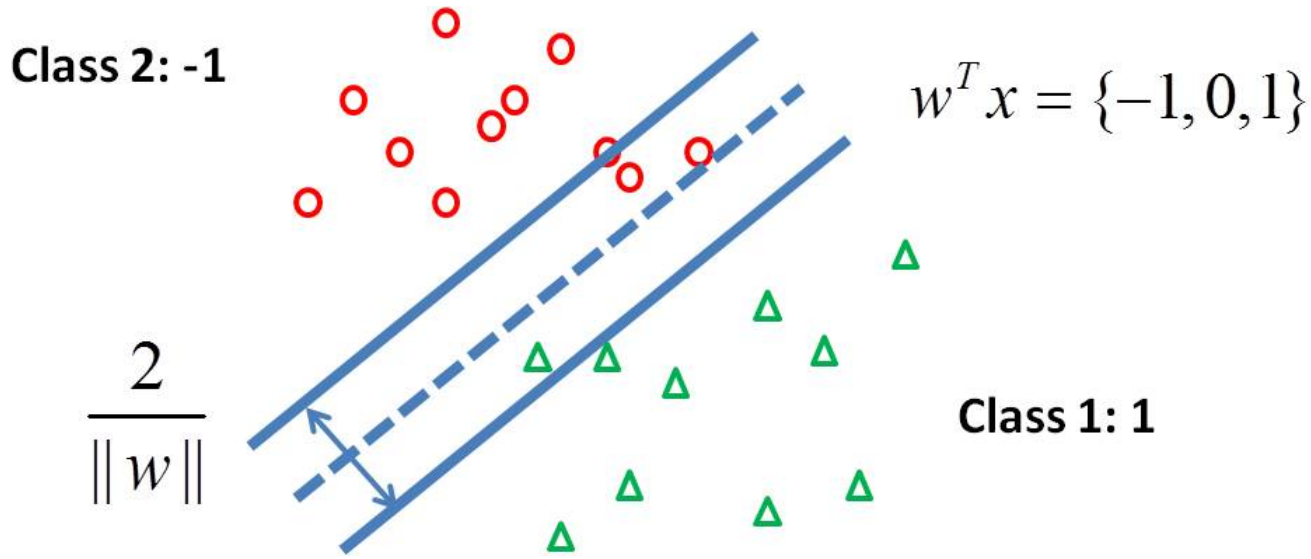
Three Examples in Machine Learning

classification using kernel SVM
dimensionality reduction for social network analysis
learning graphical model structure

Divide & Conquer Kernel SVM

Linear Support Vector Machines (SVM)

- SVM is a widely used classifier.
- Given:
  - Training data points $x_1, \cdots, x_n$.
  - Each $x_i \in \mathbb{R}^d$ 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: if $y_i = 1$, $w^T x_i \geq 1 - \xi_i$; $y_i = -1$, $w^T x_i \leq -1 + \xi_i$.



Support Vector Machines (SVM)

- Given training data $x_1, \cdots, x_n \in \mathbb{R}^d$ with labels $y_i \in \{+1, -1\}$.
- SVM primal problem:

$$\min_{w,\xi} \ \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i$$
$$\text{s.t. } \ y_i(w^T x_i) \geq 1 - \xi_i, \ \ \xi_i \geq 0, \ \ i = 1, \ldots, n,$$

Support Vector Machines (SVM)

- Given training data $x_1, \cdots, x_n \in \mathbb{R}^d$ with labels $y_i \in \{+1, -1\}$.
- SVM primal problem:

$$\begin{aligned} \min_{w,\xi} \ & \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i \\ \text{s.t.} \ & y_i (w^T x_i) \geq 1 - \xi_i, \ \xi_i \geq 0, \ i = 1, \ldots, n, \end{aligned}$$

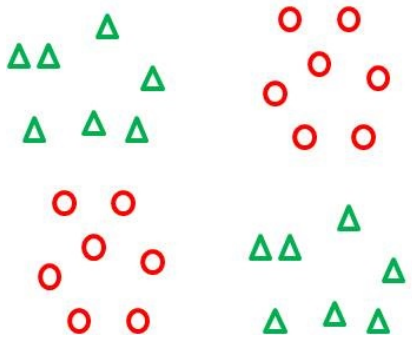
- SVM dual problem:

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

  where $Q_{ij} = y_i y_j x_i^T x_j$ and $e = [1, \ldots, 1]^T$.
- At optimum: $w = \sum_{i=1}^{n} \alpha_i y_i x_i$,

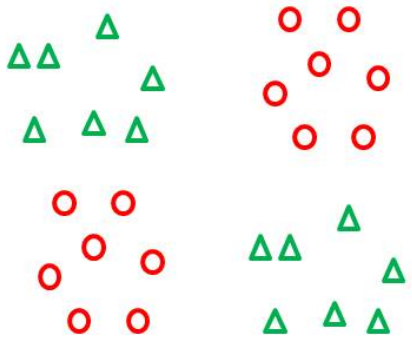
Support Vector Machines (SVM)

- What if the data is not linearly separable?



Support Vector Machines (SVM)

- What if the data is not linearly separable?



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

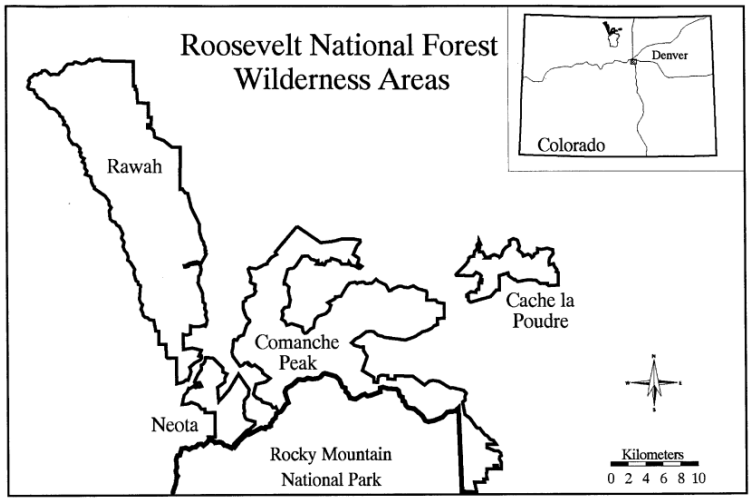
Solution: map data x_i to higher dimensional(maybe infinite) feature space $\varphi(x_i)$, where the classes are linearly separable.

- Kernel trick: $K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)$.
- Various types of kernels:
  - Gaussian kernel: $K(x, y) = e^{-\gamma \|x-y\|_2^2}$;
  - Linear kernel: $K(x, y) = x^T y$;
  - Polynomial kernel: $K(x, y) = (\gamma x^T y + c)^d$.
- Solve the dual problem for SVM.

Linear vs. Nonlinear SVM

- UCI Forest Cover dataset (transform to binary version):
  Predicting forest cover type from cartographic features:
  (Elevation, slope, soil type, . . .)
  464,810 training samples, 54 features
  Transform to binary problem: Lodgepole Pine vs. the rest 6 types

	Training Time	Prediction Time	Prediction Accuracy
Liblinear	3.6 secs	1x	76.35%
LibSVM (Gaussian kernel)	1 day	78862x	96.15%



Kernel SVM

- Recently, (Fernández-Delgado et al., 2014) evaluated 179 classifiers from 17 families using 121 UCI data sets.
- Kernel SVM is the second best classifier (slightly outperformed by random forest)

Rank	Acc.	κ	Classifier
32.9	82.0	63.5	parRF_t (RF)
33.1	**82.3**	**63.6**	rf_t (RF)
36.8	81.8	62.2	svm_C (SVM)
38.0	81.2	60.1	svmPoly_t (SVM)
39.4	81.9	62.5	rforest_R (RF)
39.6	82.0	62.0	elm_kernel_m (NNET)
40.3	81.4	61.1	svmRadialCost_t (SVM)
42.5	81.0	60.0	svmRadial_t (SVM)
42.9	80.6	61.0	C5.0_t (BST)
44.1	79.4	60.5	avNNet_t (NNET)

Table from (Fernández-Delgado et al., 2014)

Scalability for kernel SVM

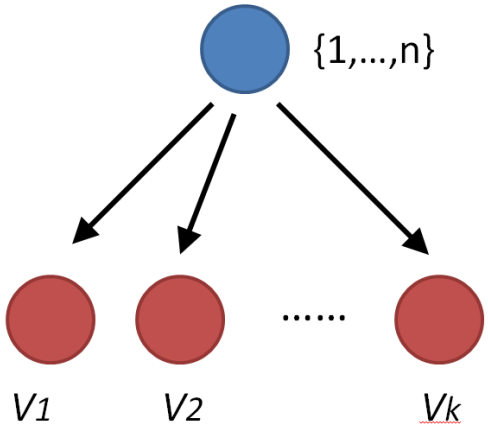
- Challenge for solving kernel SVMs:
  - Space: $O(n^2)$;
  - Training Time: $O(n^3)$.
  - Prediction Time: $O(\bar{n}d)$
    ($\bar{n}$: number of support vectors)
- Our solution: Divide-and-Conquer SVM (DC-SVM)

	Training Time	Prediction Time	Prediction Accuracy
Liblinear	3.6 secs	1x	76.35%
LibSVM	1 day	78862x	96.15%
DC-SVM (early)	10 mins	308x	96.12%
DC-Pred++	6 mins	18.8x	95.19%

DC-SVM with a single level – divide step

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

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



- Approximate solution for the whole problem:

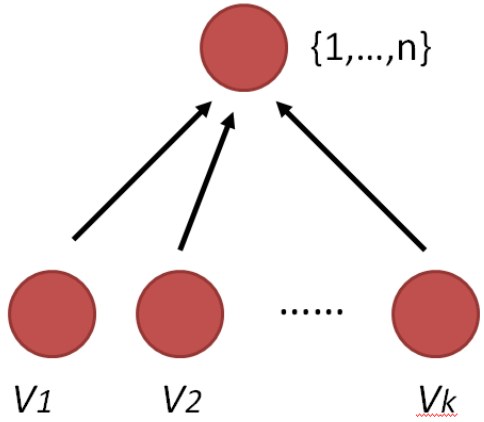
$$\bar{\alpha} = [\bar{\alpha}_{(1)}, \ldots, \bar{\alpha}_{(k)}].$$

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

DC-SVM with a single level – conquer step

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

  (1) $\|\bar{\alpha} - \alpha^*\|$ is small.
  (2) Support vectors in α^* are correctly identified in $\bar{\alpha}$.
- Question: how to find a partitioning that satisfies both?



Data Partitioning

Theorem 1

For a given partition π, the corresponding $\bar{\boldsymbol{\alpha}}$ satisfies

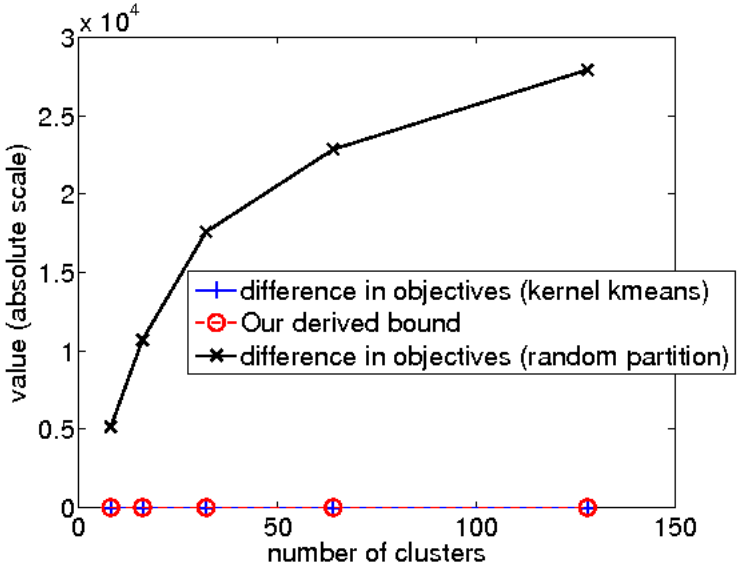
$$0 \leq f(\bar{\boldsymbol{\alpha}}) - f(\boldsymbol{\alpha}^*) \leq (1/2)C^2 D(\pi),$$

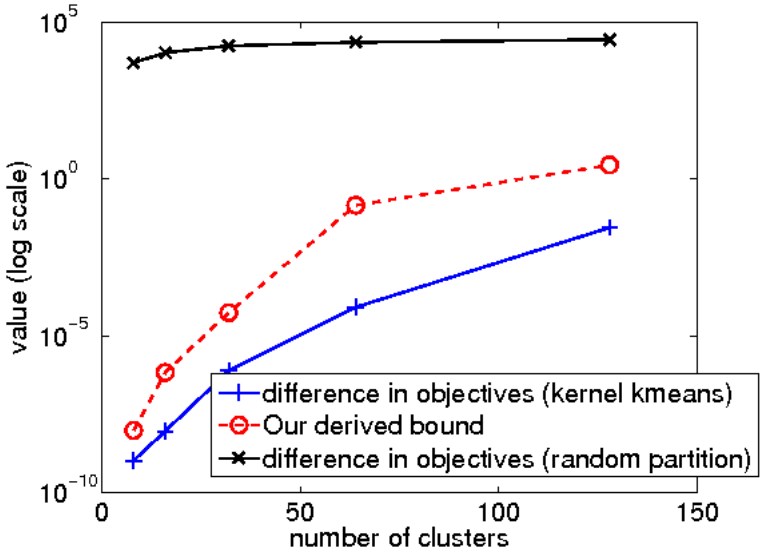
where $D(\pi) = \sum_{i,j:\pi(x_i)\neq\pi(x_j)} |K(x_i, x_j)|$.

- Want a partition which
  - (1) Minimizes $D(\pi)$.
  - (2) Has 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 $m \ll n$.
  - Identify the clusters for the rest of data.

Demonstration of the bound

- 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 $\boldsymbol{\alpha}^*$.





Quickly identifying support vectors

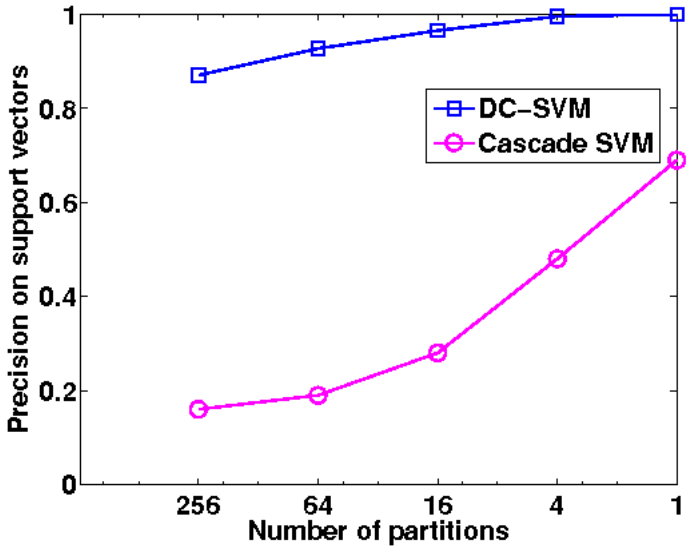
- Divide-and-conquer scheme helps to quickly identify support vectors.

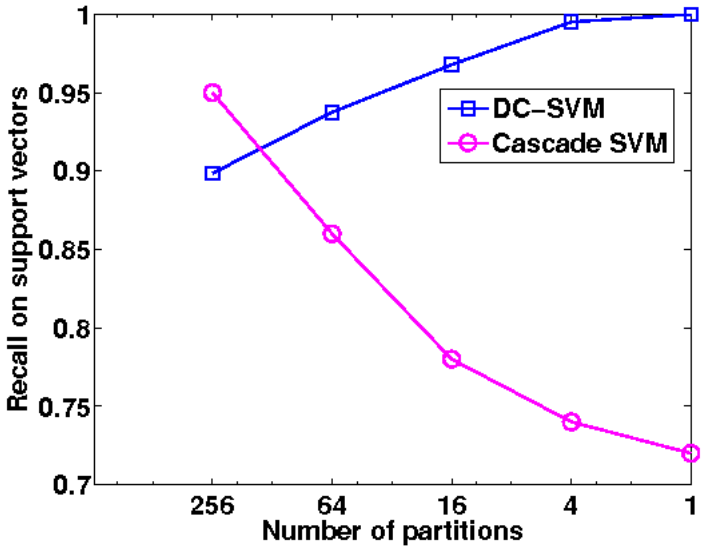
Theorem 2

If $\bar{\alpha}_i = 0$ and

$$\nabla_i \bar{f}(\bar{\boldsymbol{\alpha}}) > CD(\pi)(1 + \sqrt{n}K_{max}/\sqrt{\sigma_n D(\pi)}),$$

where $K_{max} = \max_i K(x_i, x_i)$, then x_i will not be a support vector of the whole problem (i.e., $\alpha_i^* = 0$).



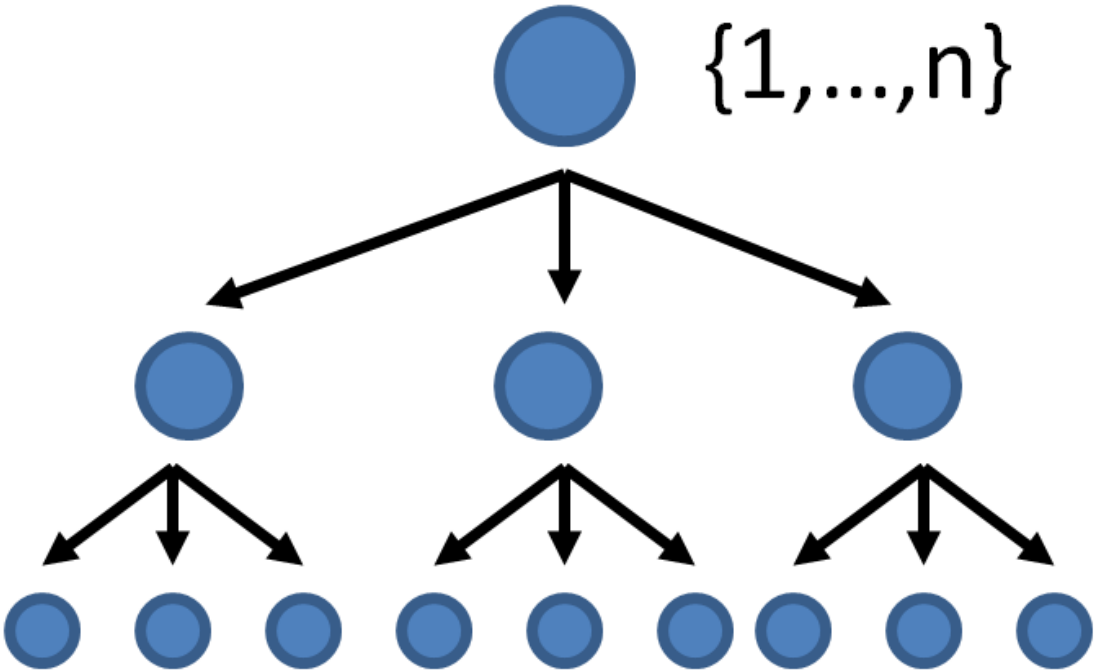


DC-SVM with multiple levels

- A tradeoff in choosing k (number of partitions):
  - Small $k \Rightarrow$ smaller $\|\bar{\alpha} - \alpha^*\|$ but more time to solve subproblems.
  - Large $k \Rightarrow$ larger $\|\bar{\alpha} - \alpha^*\|$ but less time to solve subproblems.

DC-SVM with multiple levels

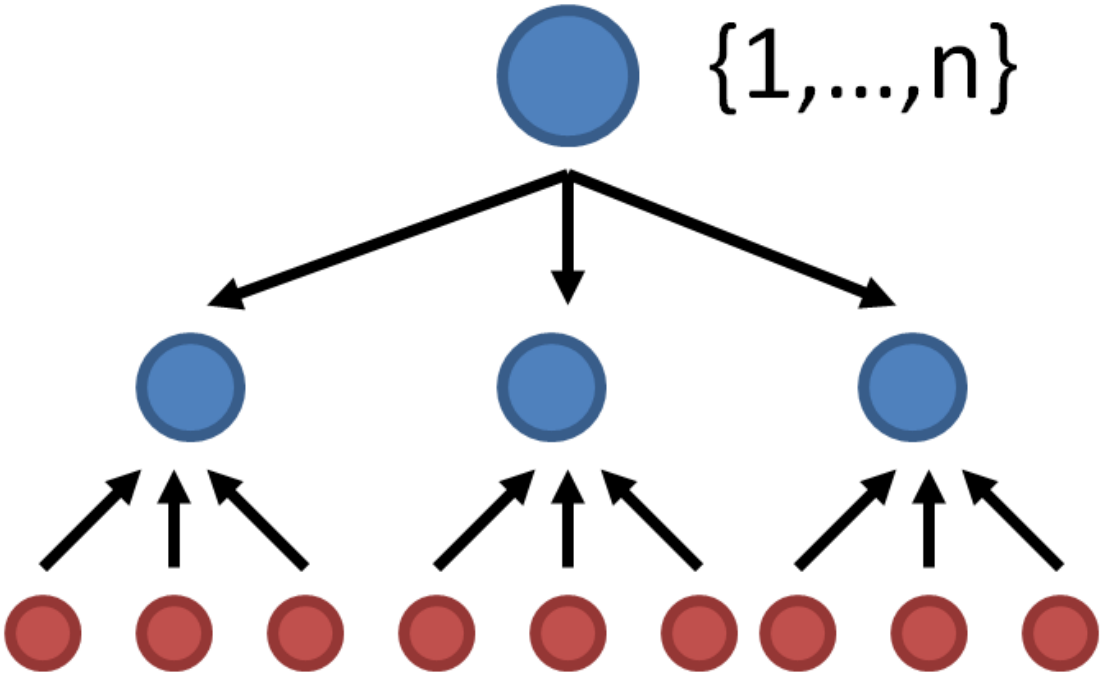
- A tradeoff in choosing k (number of partitions):
  - Small $k \Rightarrow$ smaller $\|\bar{\alpha} - \alpha^*\|$ but more time to solve subproblems.
  - Large $k \Rightarrow$ larger $\|\bar{\alpha} - \alpha^*\|$ but less time to solve subproblems.
- Therefore we run DC-SVM with multiple levels.



Data Division

DC-SVM with multiple levels

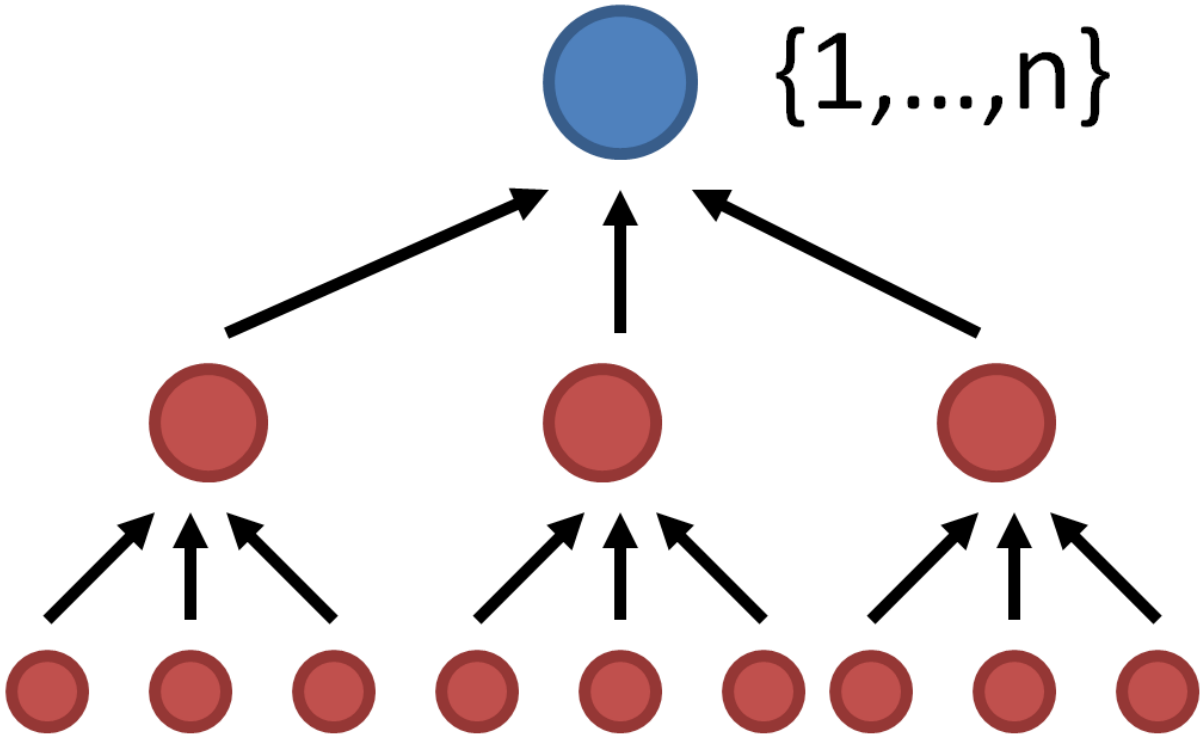
- A tradeoff in choosing k (number of partitions):
  - Small $k \Rightarrow$ smaller $\|\bar{\alpha} - \alpha^*\|$ but need more time to solve subproblems.
  - Large $k \Rightarrow$ larger $\|\bar{\alpha} - \alpha^*\|$ but less time to solve subproblems.
- Therefore propose to run DC-SVM with multiple levels.



Solve the leaf level problems.

DC-SVM with multiple levels

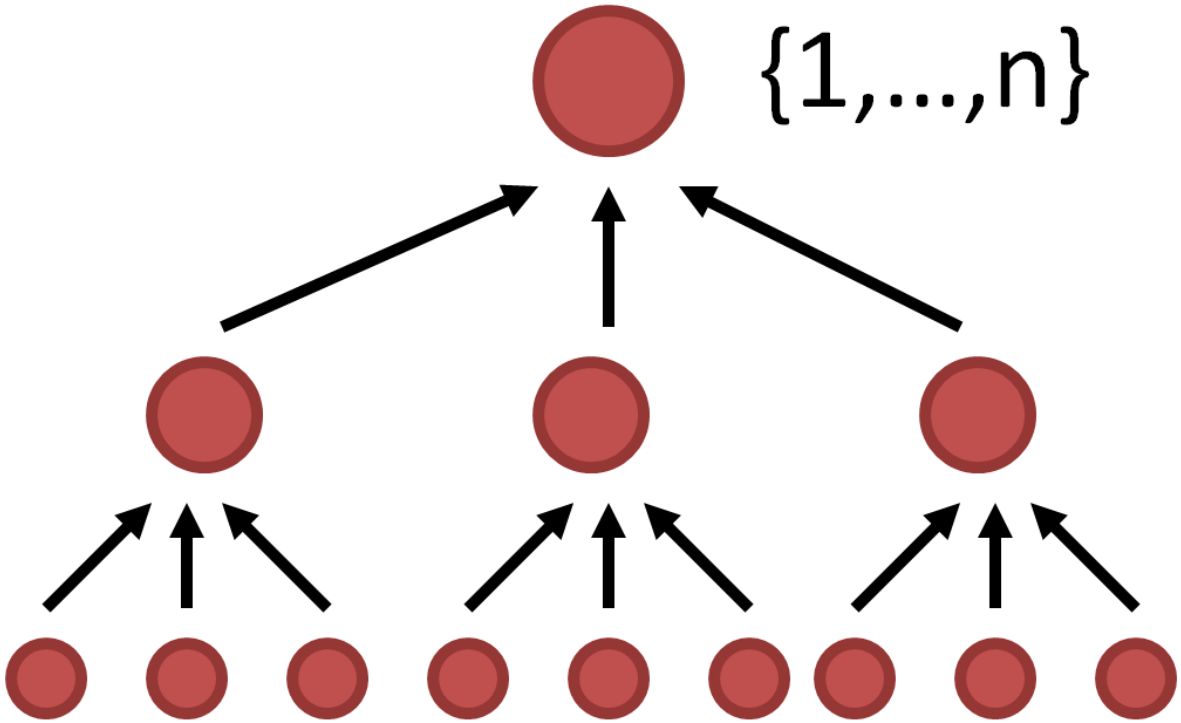
- A tradeoff in choosing k (number of partitions):
  - Small $k \Rightarrow$ smaller $\|\bar{\alpha} - \alpha^*\|$ but need more time to solve subproblems.
  - Large $k \Rightarrow$ larger $\|\bar{\alpha} - \alpha^*\|$ but less time to solve subproblems.
- Therefore propose to run DC-SVM with multiple levels.



Solve the intermediate level problems.

DC-SVM with multiple levels

- A tradeoff in choosing k (number of partitions):
  - Small $k \Rightarrow$ smaller $\|\bar{\alpha} - \alpha^*\|$ but need more time to solve subproblems.
  - Large $k \Rightarrow$ larger $\|\bar{\alpha} - \alpha^*\|$ but less time to solve subproblems.
- Therefore propose to run DC-SVM with multiple levels.



Solve the original problem.

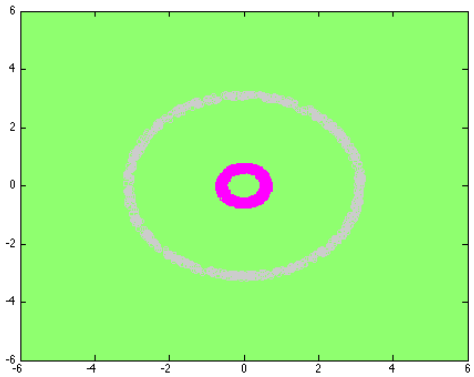
Early Prediction

- Prediction using the l-th level solution

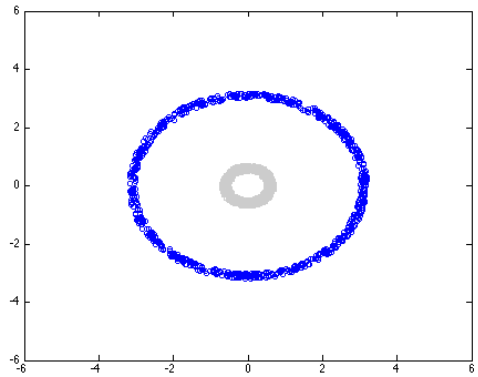
  faster training time; the prediction accuracy is close to or even better than the global SVM solution.
- Prediction: $\text{sign}(\sum_{i \in \mathcal{V}_{\pi(x)}} y_i \alpha_i K(x_i, x))$.
- Prediction time reduced from $O(d(\#SV))$ to $O(d(\#SV)/k)$
- Covtype ($k = 256$): 78862x $\rightarrow$ 308x
- **Still not fast enough for real-time applications!**

Illustrative Toy Example

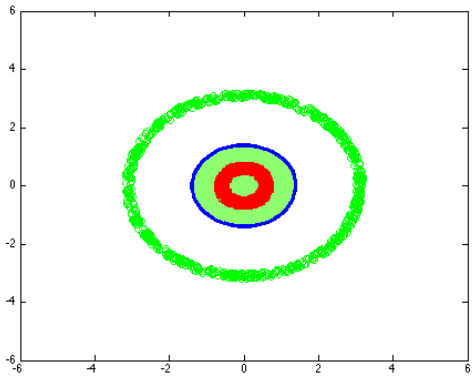
Two Circle Data: each circle is a class; separable by kernel kmeans.



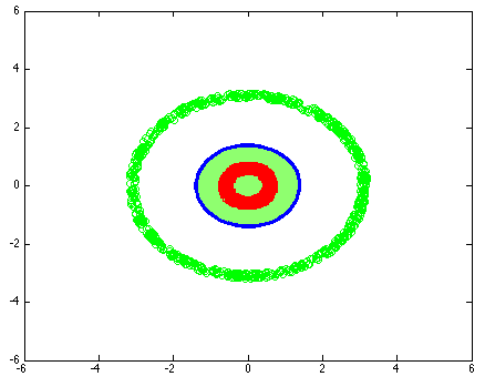
1st cluster



2nd cluster



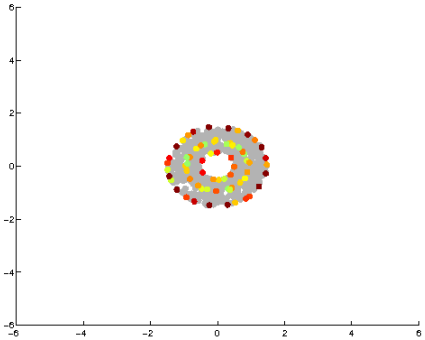
DC-SVM (early)



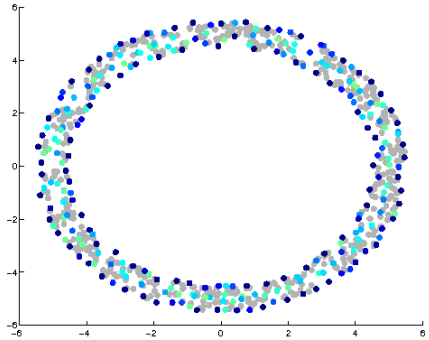
RBF SVM

Illustrative Toy Examples

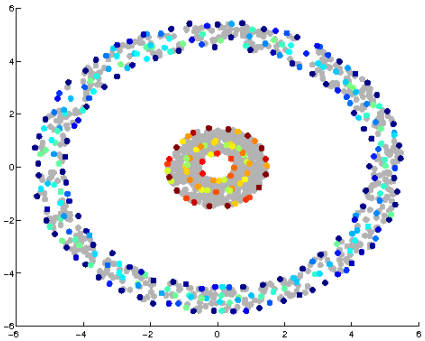
Two Circle Data: each circle is a class; separable by kernel kmeans.



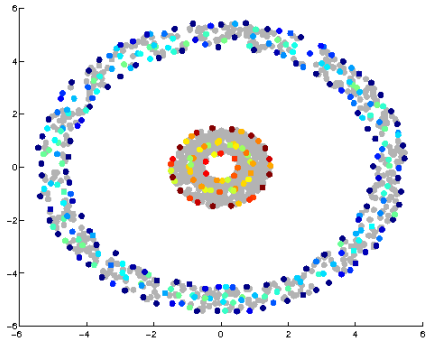
1st cluster



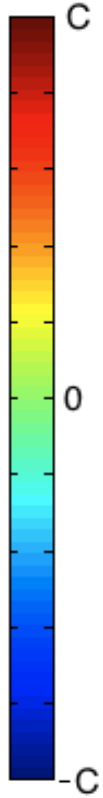
2nd cluster



DC-SVM (early)

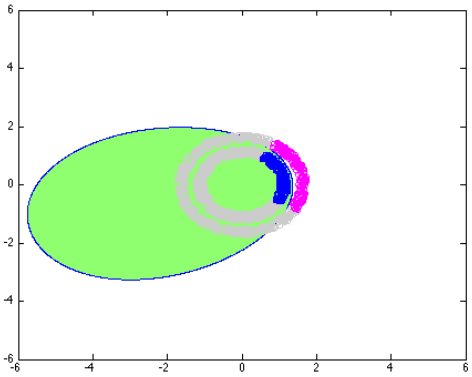


RBF SVM

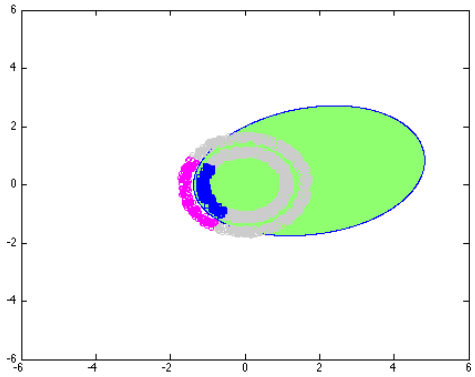


Illustrative Toy Examples

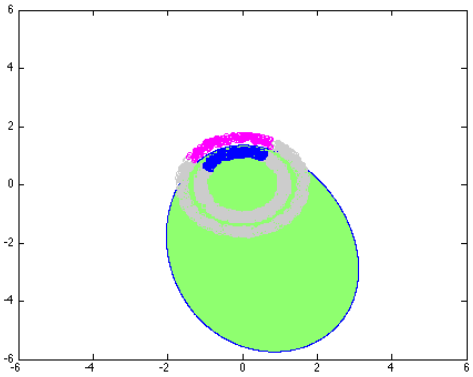
Two Circle Data: not separable by kernel kmeans



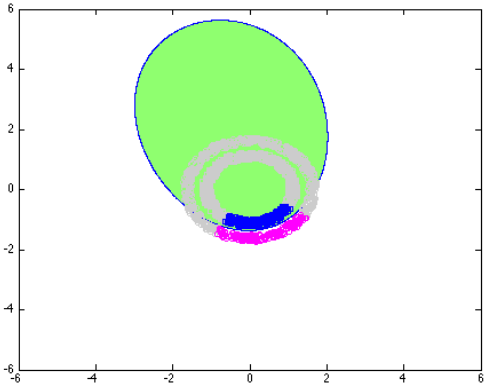
1st cluster



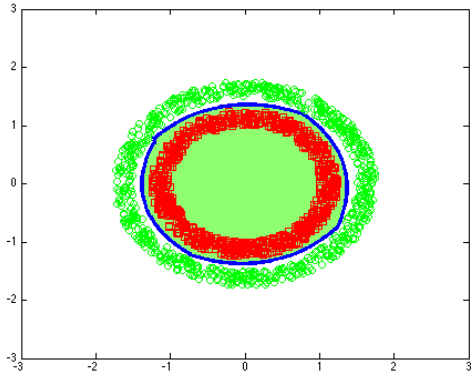
2nd cluster



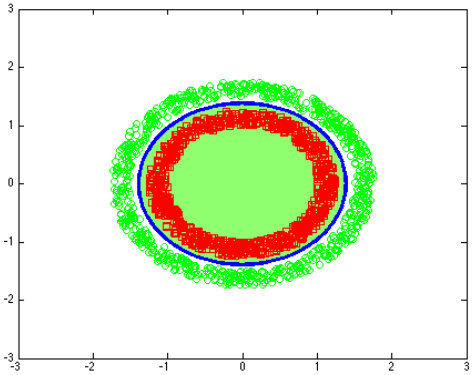
3rd cluster



4th cluster



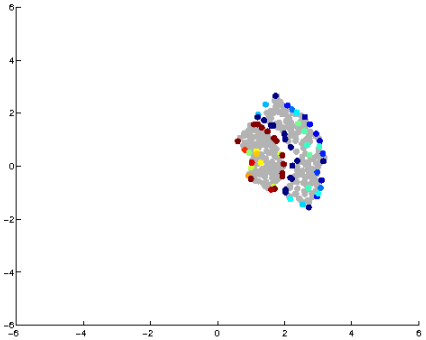
DC-SVM (early)



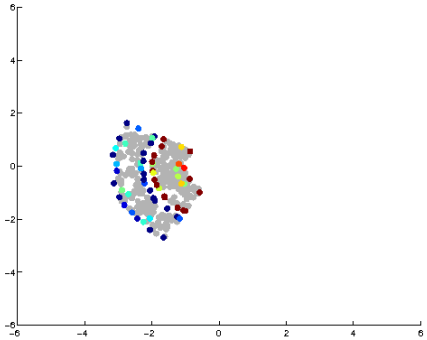
RBF SVM

Illustrative Toy Examples

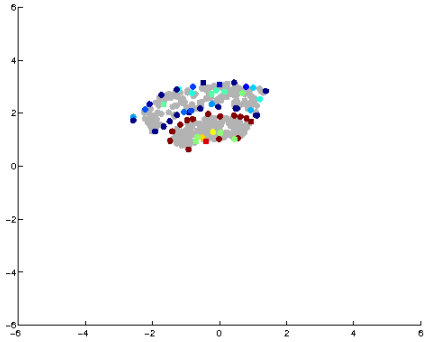
Two Circle Data: not separable by kernel kmeans



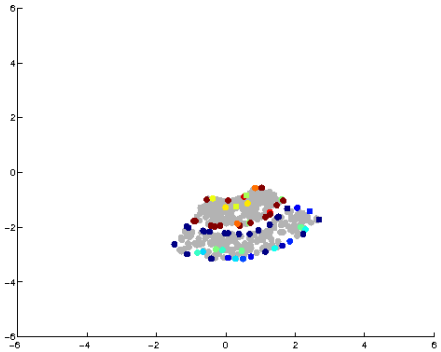
1st cluster



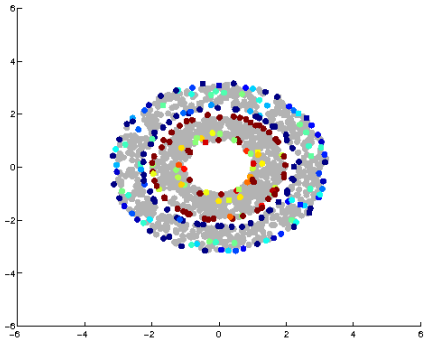
2nd cluster



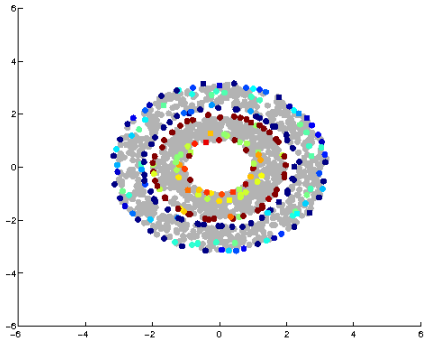
3rd cluster



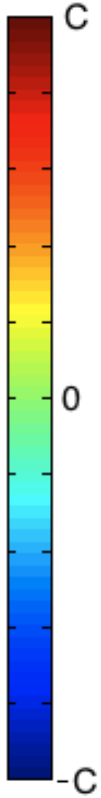
4th cluster



DC-SVM (early)



RBF SVM



Methods included in comparisons

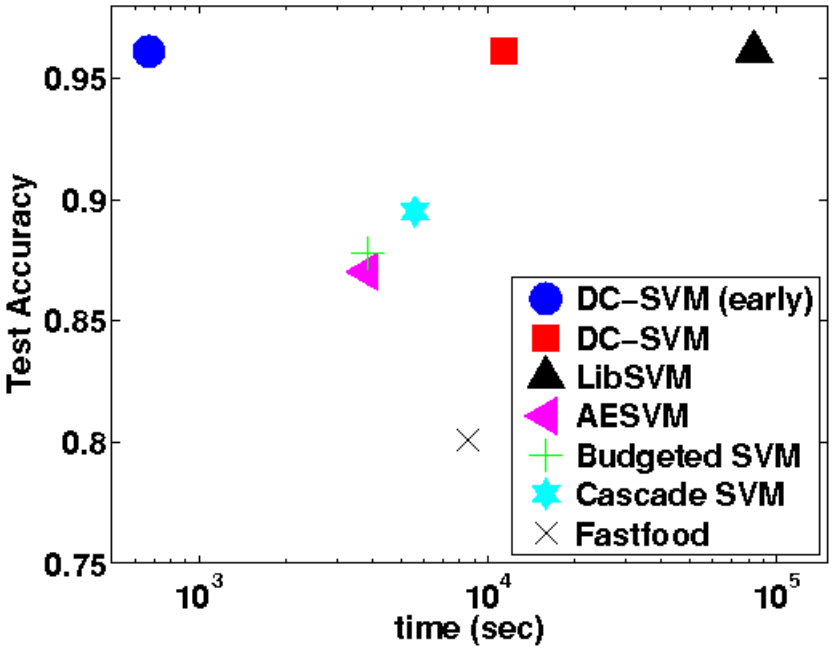
- DC-SVM: our proposed method for solving the global SVM problem.
- DC-SVM (EARLY): our proposed method with early stopping (at 64 clusters).
- LIBSVM (Chang and Lin, 2011)
- CASCADE SVM (Graf et al., 2005)
- FASTFOOD (Le et al., 2013)
- LASVM (Bordes et al., 2005)
- LLSVM (Zhang et al., 2012)
- SPSVM (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.

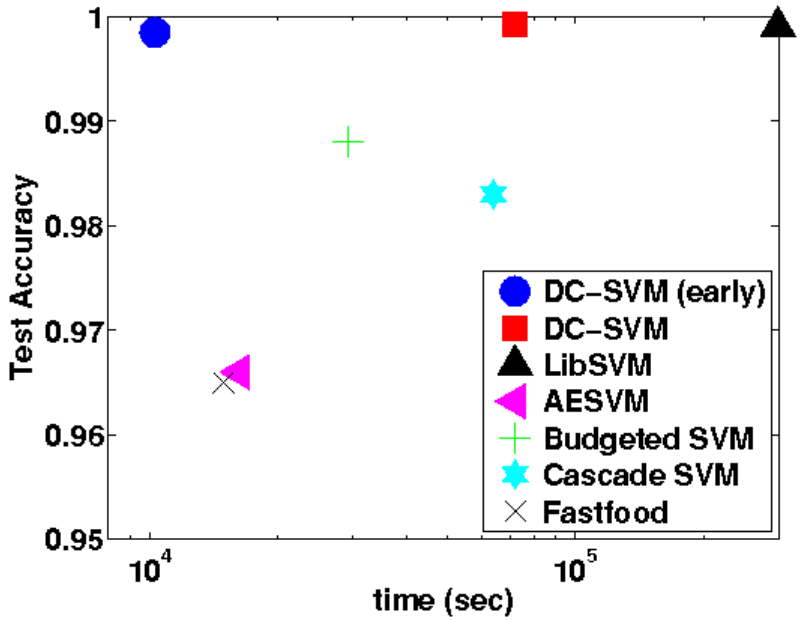
	webspam		covtype		mnist8m	
	$n = 2.8 \times 10^5, d = 254$		$n = 4.65 \times 10^5, d = 54$		$n = 8 \times 10^6, d = 784$	
	$C = 8, \gamma = 32$		$C = 32, \gamma = 32$		$C = 1, \gamma = 2^{-21}$	
	time(s)	acc(%)	time(s)	acc(%)	time(s)	acc(%)
DC-SVM (early)	**670**	99.13	**672**	96.12	**10287**	99.85
DC-SVM	10485	**99.28**	11414	**96.15**	71823	**99.93**
LIBSVM	29472	**99.28**	83631	96.15	298900	99.91
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

Time vs Prediction Accuracy

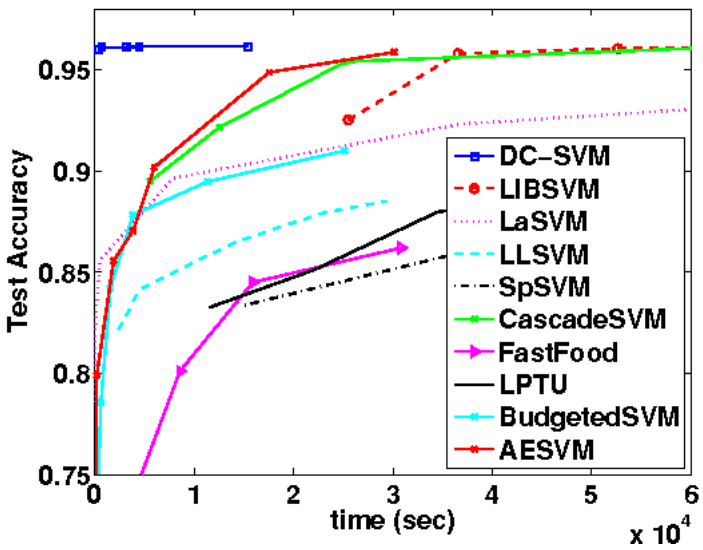


Covtype dataset

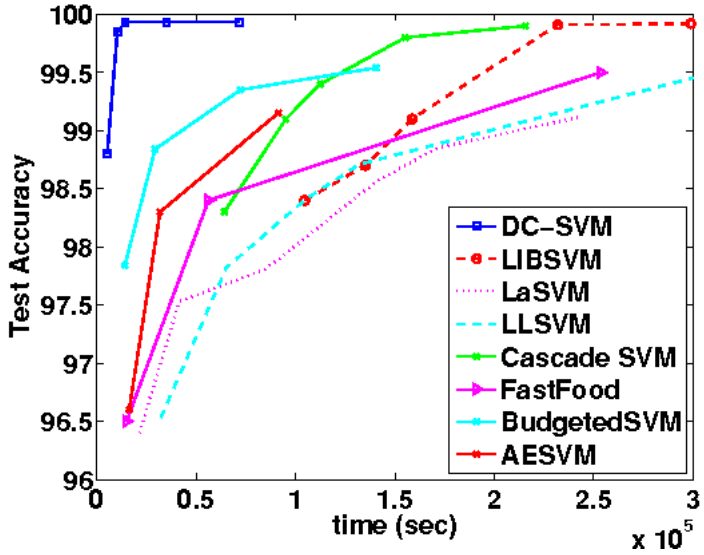


MNIST dataset

Results with Gaussian kernel



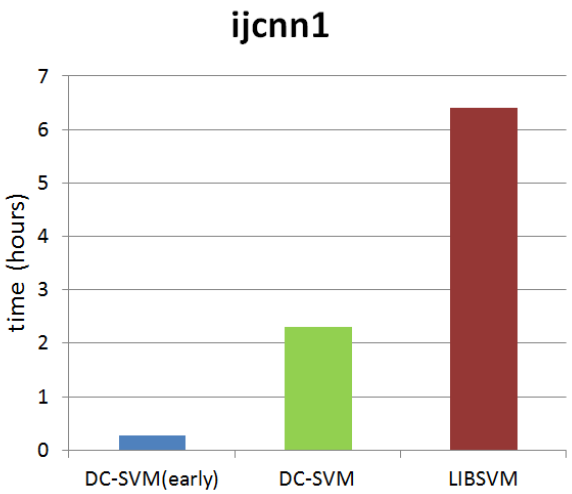
covtype prediction accuracy

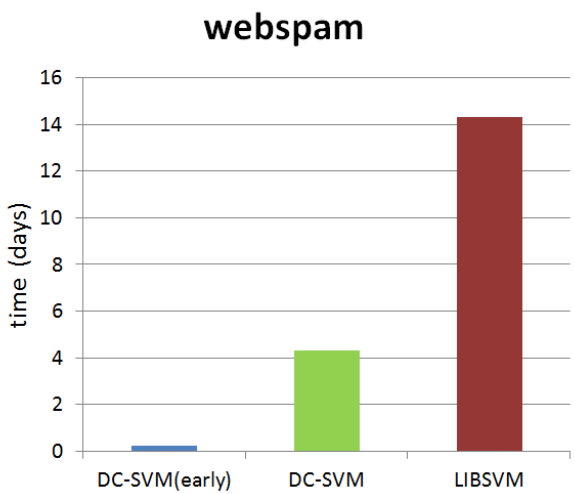


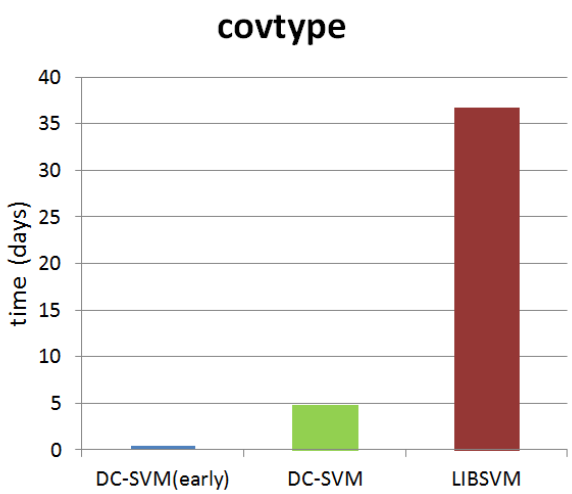
MNIST8m prediction accuracy

Results with grid of C, γ

- Total time on the grid of parameters C, γ:
  - $C = 2^{-10}, 2^{-6}, 2^{1}, 2^{6}, 2^{10}$
  - $\gamma = 2^{-10}, 2^{-6}, 2^{1}, 2^{6}, 2^{10}$







Fast Prediction for Kernel SVM

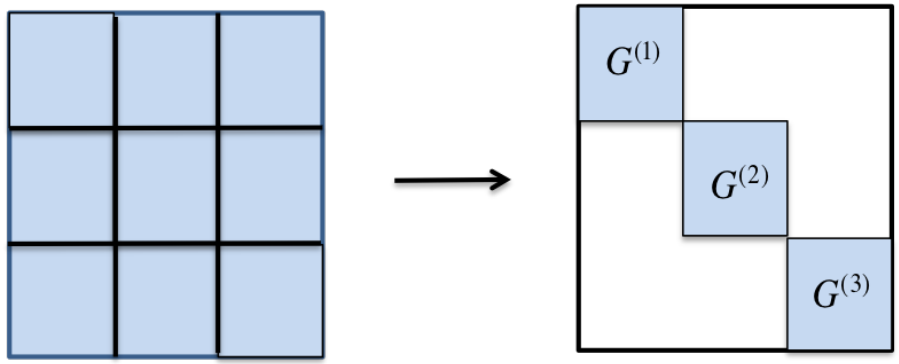
Prediction for SVM

- Linear SVM: $y = sign(w^T x + b)$; $O(d)$.
- Kernel SVM: $y = sign(\sum_{i=1}^{n} \alpha_i K(x, x_i))$; $O(d(\#SV))$.
- DC-SVM with early prediction: $y = sign(\sum_{i \in \mathcal{V}_{\pi(x)}} \alpha_i K(x, x_i))$; $O(d(\#SV/k))$.
- DC-Pred++: $O(dt)$; $t \ll \#SV/k$

Goal: achieve kernel SVM's accuracy using linear SVM prediction time.

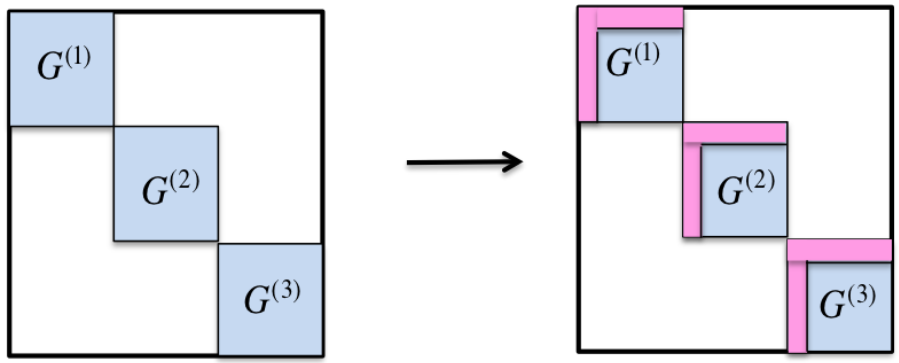
DC-SVM with Nyström Approximation

- DC-SVM with early prediction



Drawback: still too many support vectors in each block

- Use Nyström approximation within each block



Nyström Kernel Approximation

- Goal: rank-m approximation $\tilde{G}$ to kernel matrix G based on landmark points $u_1, \ldots, u_m$.

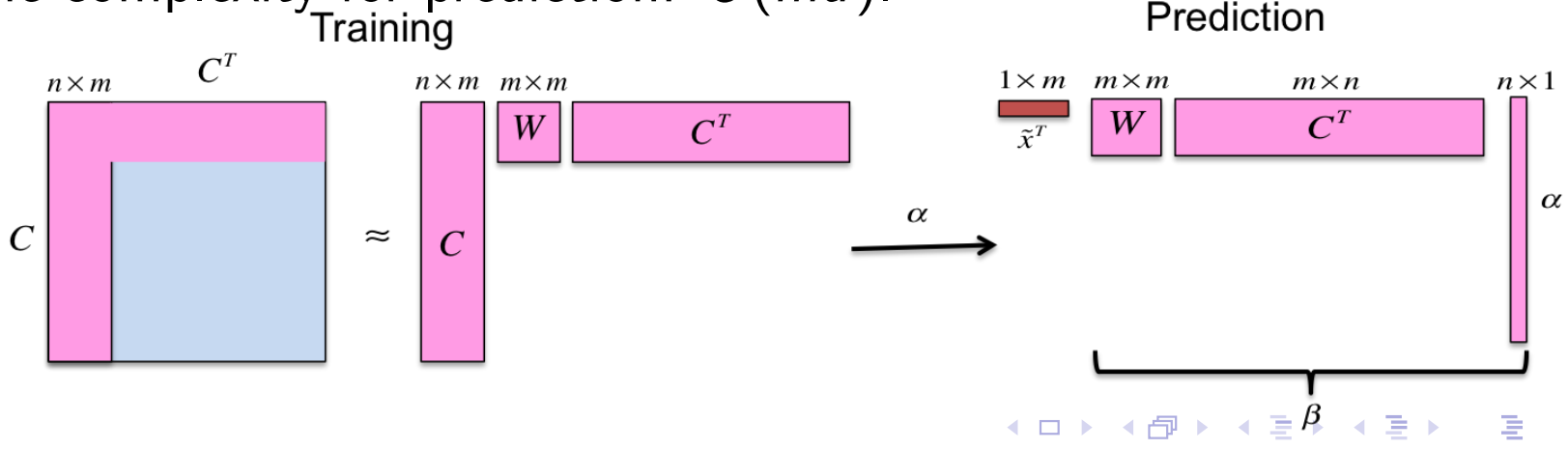
$$G \approx \tilde{G} = CWC^T.$$

- Perform prediction on a new point x given the model $\boldsymbol{\alpha}$:

$$\sum_{i=1}^{n} \alpha_i \tilde{K}(x, x_i) = \tilde{x}^T WC^T \boldsymbol{\alpha} = \tilde{x}^T \boldsymbol{\beta}$$

$$\text{where} \quad \tilde{x} = [K(x, u_1), \ldots, K(x, u_m)]^T.$$

- The main computation is to form $\tilde{x}$.
- Time complexity for prediction: $O(md)$.

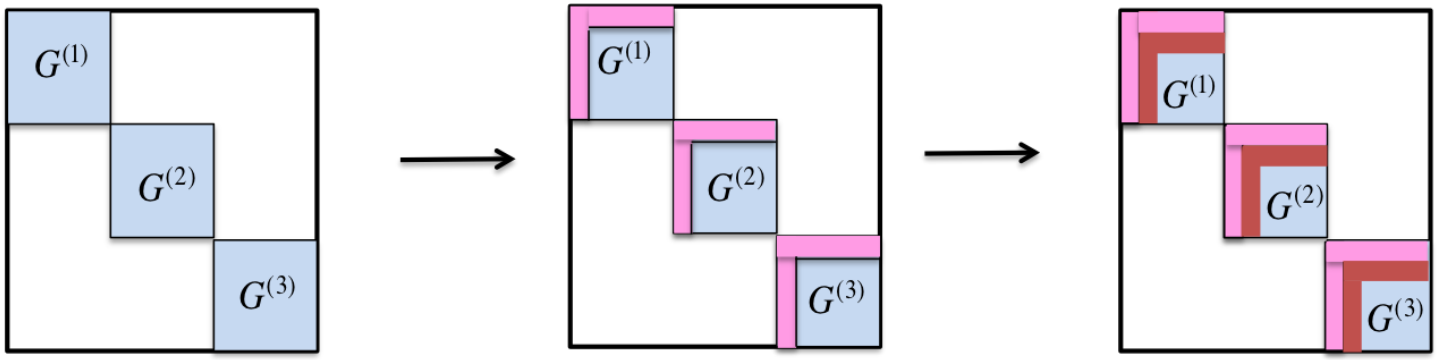


Improvements over Traditional Nyström Approximation

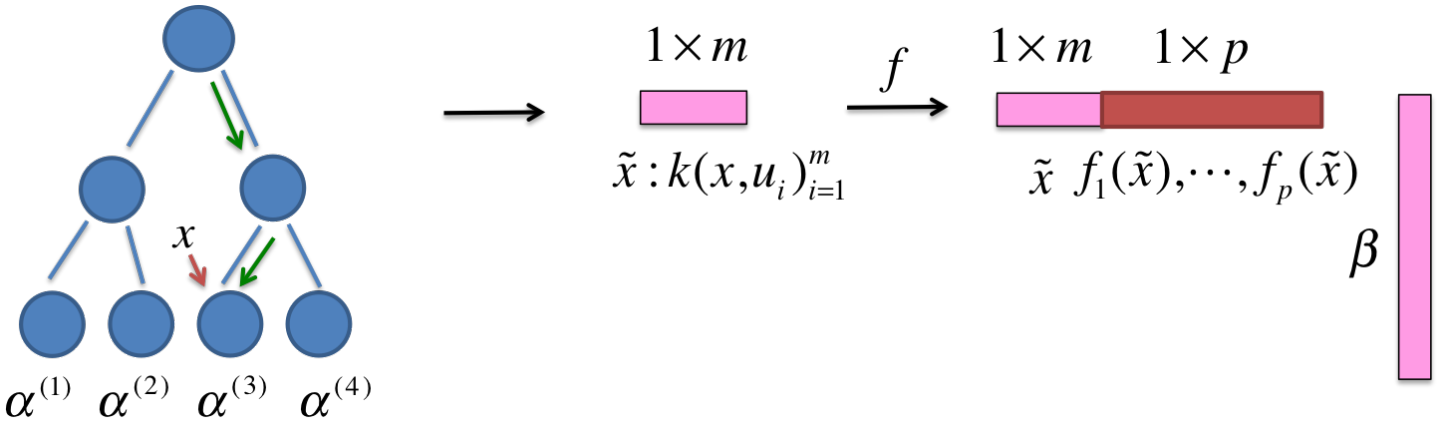
- Traditional Nyström Approximation emphasizes kernel approximation error $\|G - \tilde{G}\|_F$.
- We are interested in model approximation error $\|\boldsymbol{\alpha}^* - \bar{\boldsymbol{\alpha}}\|$.
- Use $\boldsymbol{\alpha}$-weighted kmeans centroids as the landmark points.
- Add pseudo landmark points for faster prediction.

DC-Pred++

- Training:



- Prediction:



Kernel SVM Results

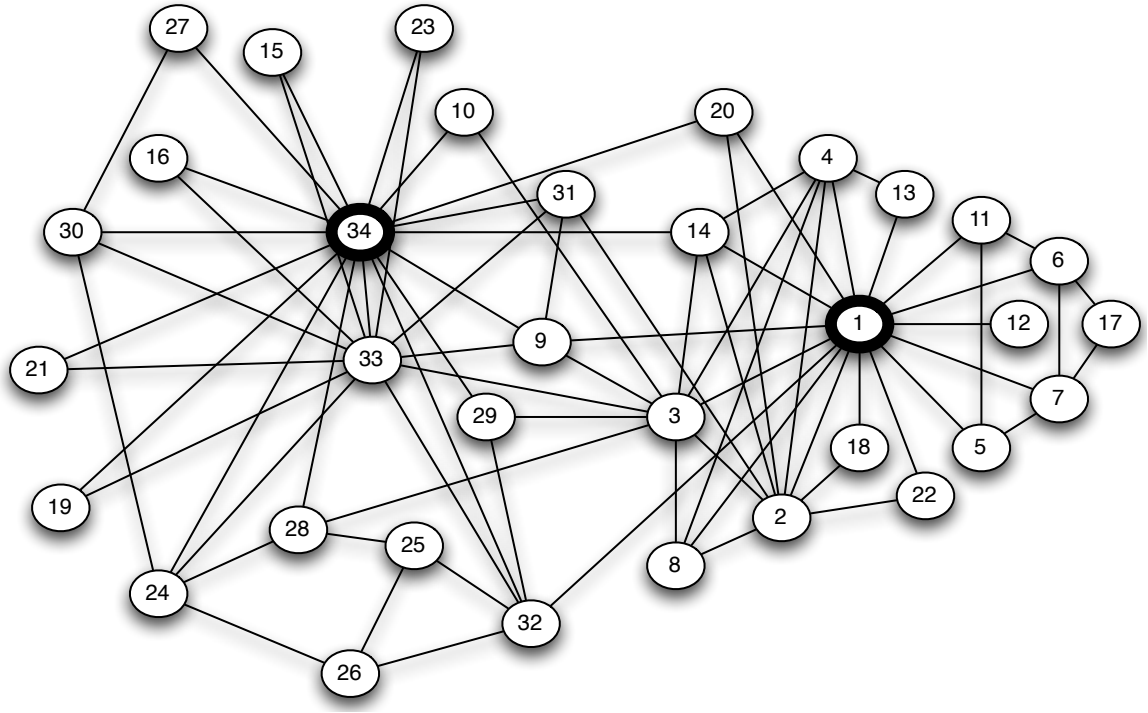
Dataset	Metric	DC-Pred++	LDKL	kmeans Nyström	AESVM	Liblinear
Letter	Prediction Time	**12.8x**	29x	140x	1542x	1x
$n = 12,000$	Accuracy	**95.90%**	95.78%	87.58%	80.97%	73.08%
CovType	Prediction Time	**18.8x**	35x	200x	3157x	1x
$n = 522,910$	Accuracy	**95.19%**	89.53%	73.63%	75.81%	76.35%
USPS	Prediction Time	14.4x	**12.01x**	200x	5787x	1x
$n = 7291$	Accuracy	95.56%	**95.96%**	92.53%	85.97%	83.65%
Webspam	Prediction Time	**20.5x**	23x	200x	4375x	1x
$n = 280,000$	Accuracy	**98.4%**	95.15%	95.01%	**98.4%**	93.10%

Social Network Analysis

Social Networks

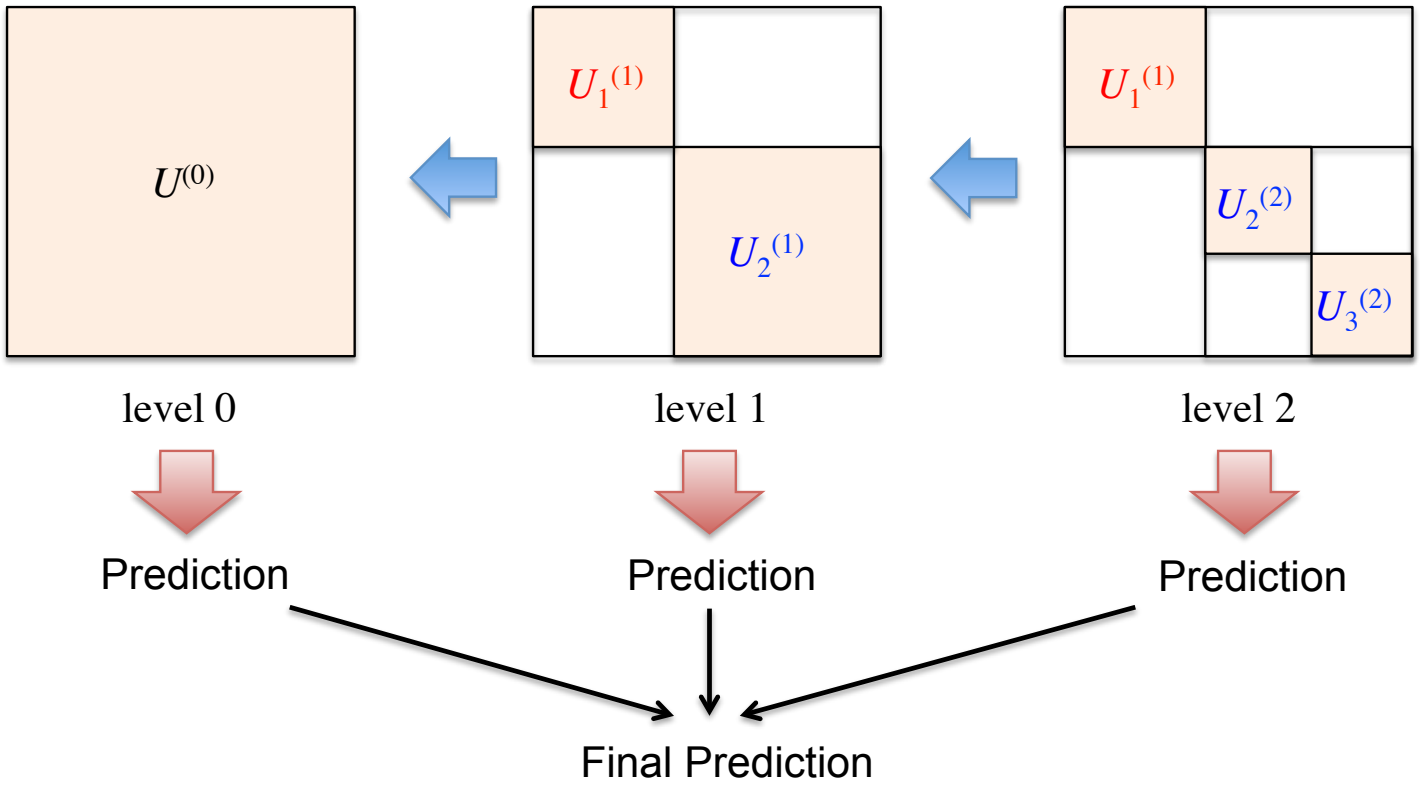
- Nodes: individual actors (people or groups of people)
- Edges: relationships (social interactions) between nodes

Example: Karate Club Network:



Multi-Scale Link Prediction

- Combine predictions at each level to make final predictions



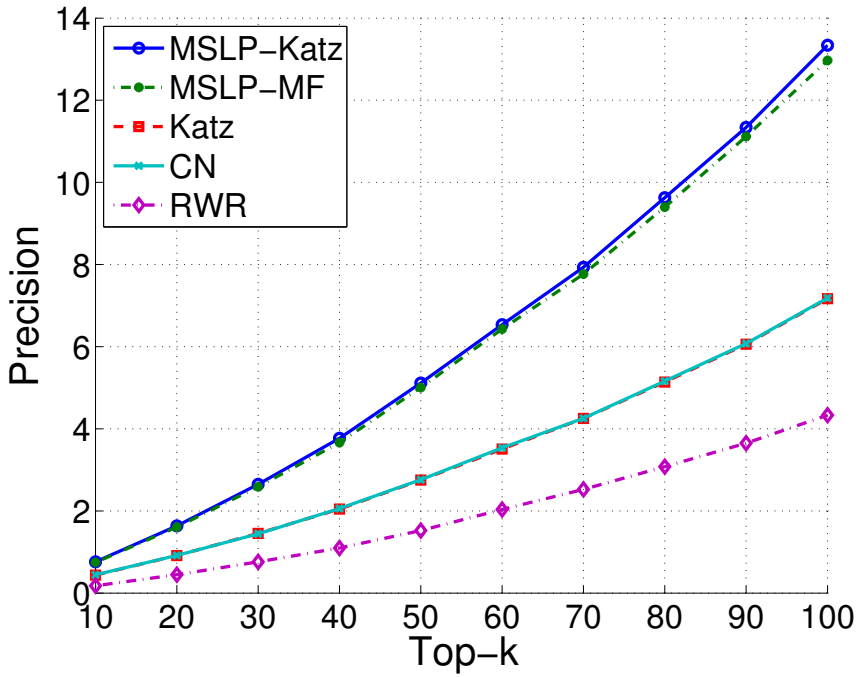
Experimental Results

Flickr dataset: 1.9M users & 42M links.

- Test set: sampled 5K users.
- Precision at top-100 & AUC results:

Method	Prec	AUC
Preferential Attachment	1.02	0.6981
Common Neighbor	7.08	0.8649
Adamic-Adar	7.29	0.8758
Random Walk w/ Restarts	5.49	0.7872
Logistic Regression*	2.54	0.7115
Katz	7.17	0.8429
MSLP-Katz	**13.34**	**0.8924**
MF	12.05	0.9078
MSLP-MF	**13.07**	**0.9145**

*using network-based features.



Learning Graphical Model Structure

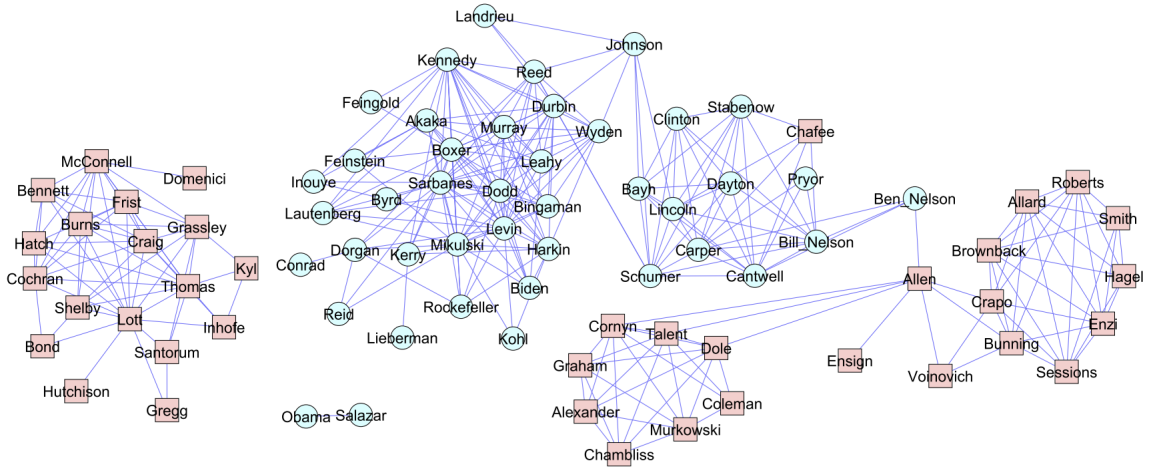
Sparse Inverse Covariance Estimation

- Given: n i.i.d. samples $\{y_1, \ldots, y_n\}$, $y_i \sim \mathcal{N}(\mu, \Sigma)$,
- Goal: Estimate the inverse covariance $\Theta = \Sigma^{-1}$.
- The resulting optimization problem:

$$\Theta = \arg\min_{X \succ 0} \left\{ -\log\det X + \mathrm{tr}(SX) + \lambda \|X\|_1 \right\} = \arg\min_{X \succ 0} f(X),$$

where $\|X\|_1 = \sum_{i,j=1}^{n} |X_{ij}|$ and $S = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\mu})(y_i - \hat{\mu})^T$.

- Regularization parameter $\lambda > 0$ controls the sparsity.
- Real world example: graphical model which reveals the relationships between Senators: (Figure from Banerjee et al, 2008)

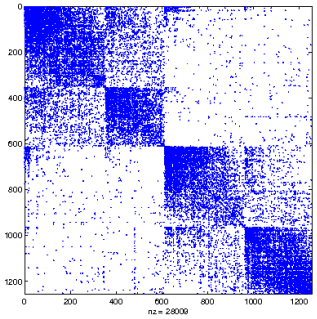


Divide-and-Conquer QUIC

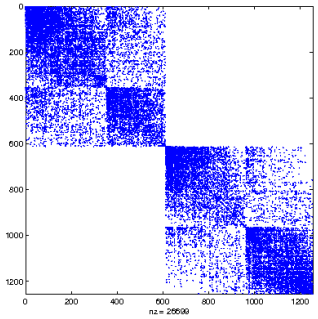
- How can we solve high dimensional problems?

  Time complexity is $O(p^3)$, so computationally expensive to solve it directly.

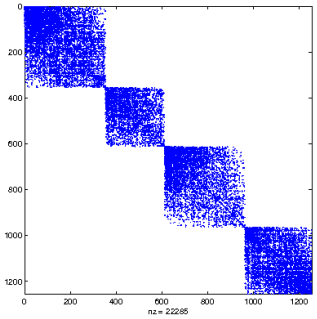
- Solution: Divide & Conquer!



Θ^*

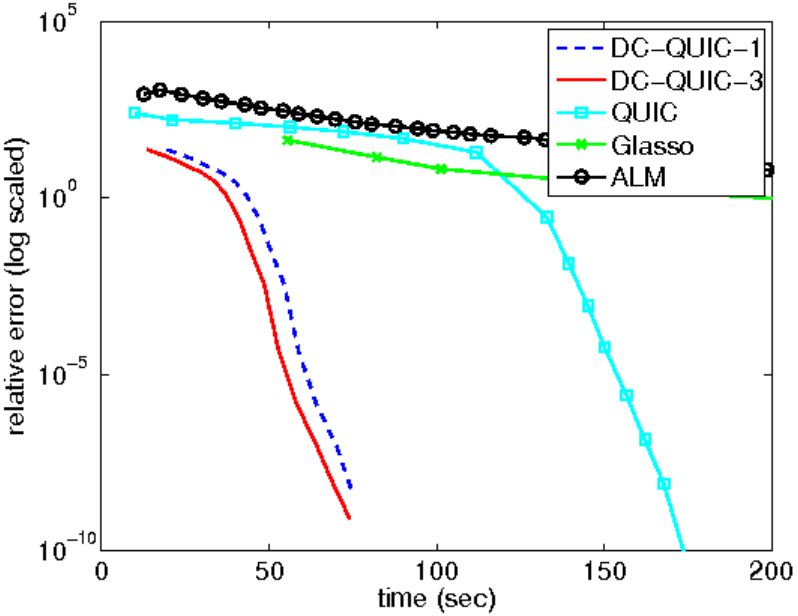


Θ from level-1 clusters

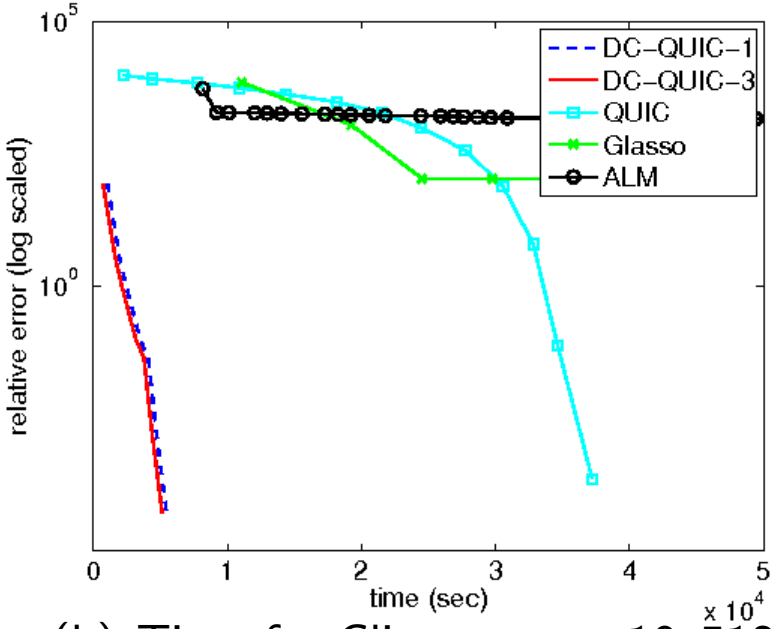


Θ from level-2 clusters.

Performance of Divide & Conquer QUIC



(a) Time for Leukemia, $p = 1,255$



(b) Time for Climate, $p = 10,512$

Conclusions

- Divide & Conquer approach for big data problems
  - Divide & Conquer Kernel SVM — fast training and fast prediction
  - Social Network Analysis: Multi-Scale Link Prediction
  - Sparse Inverse Covariance Estimation: Divide & Conquer QUIC

- Interesting questions
  - What other data analysis problems can benefit from a divide & conquer approach? Regression? Matrix completion? Multi-label learning?
  - What are the corresponding algorithms?
  - Theoretical/statistical guarantees?
  - Can one exploit the multiplicity of local and global models for better prediction?
  - Parallelization? Needs dynamic load balancing, asynchronous parallelism.

References

Divide and Conquer Kernel SVM

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

[2] S. Si, C.-J. Hsieh and I. S. Dhillon *Memory Efficient Kernel Approximation*, ICML, 2014.

[3] C.-J. Hsieh, S. Si and I. S. Dhillon *Fast Prediction for Large-Scale Kernel Machines*, NIPS, 2014.

Divide and Conquer Link-Prediction/Eigen-decomposition

[4] D. Shin, S. Si and I. S. Dhillon *Multi-Scale Link Prediction*, CIKM, 2012.

[5] S. Si, D. Shin, I. S. Dhillon and B. Parlett *Multi-Scale Spectral Decomposition of Massive Graphs*, NIPS, 2014.

Divide and Conquer Sparse Inverse Covariance Estimation

[6] C.-J. Hsieh, M. Sustik I. S. Dhillon and P. Ravikumar *Sparse Inverse Covariance Matrix Estimation using Quadratic Approximation*. NIPS, 2011.

[7] C.-J. Hsieh, I. S. Dhillon, P. Ravikumar and A. Banerjee *A Divide-and-Conquer Method for Sparse Inverse Covariance Estimation*, NIPS, 2012.

[8] C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, P. Ravikumar, and R. A. Poldrack *BIG & QUIC: Sparse Inverse Covariance Estimation for a Million Variables*, NIPS, 2013.

[9] C.-J. Hsieh, I. S. Dhillon, P. Ravikumar and S. Becker, and P. A. Olsen *QUIC & DIRTY: A Quadratic Approximation Approach for Dirty Statistical Models*, NIPS, 2014.