The Many Facets of Big Data

James Kwok

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

ACPR 2013

Big Data

1. volume
   - sample size is big
   - feature dimensionality is big
2. variety
   - multiple formats: social, video, unstructured
   - problem is big and has many related tasks
3. velocity
   - batch $\longrightarrow$ real-time

Big Data

1. volume
   - sample size is big
   - feature dimensionality is big
2. variety
   - multiple formats: social, video, unstructured
   - problem is big and has many related tasks
3. velocity
   - batch $\longrightarrow$ real-time

Big Feature Dimension

Real-world data are often high-dimensional

Abstract—**Real-world data sets are highly complicated. They can contain a lot of features, and may involve multiple learning tasks with intrinsically or explicitly represented task relationships. In this paper, we briefly discuss several recent approaches that can be used in these scenarios. The algorithms presented are flexible in capturing the task relationships, computationally efficient with good scalability, and have better empirical performance than the existing approaches.**

Keywords-**sparse modeling; multitask learning; multilabel learning**

I. Summary

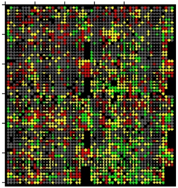
Real-world data sets are highly complicated. They can be high-dimensional and contain many spurious features, as is typical in applications involving text and images. This calls for some form of feature selection or feature extraction. Besides, they may involve multiple learning tasks, whose relationships can be represented either intrinsically or explicitly (as in hierarchical classification). When the number of tasks grows to the hundreds or even thousands, computationally inefficiency becomes another important issue. Instead of performing feature selection and learning the tasks individually, it is now well-known that better performance can be obtained by integrating feature selection with classifier training in the creation of a sparse model. Moreover, by harnessing the task relationships, various tasks can borrow strength from each other.

However, sparse modeling techniques, such as the famous lasso method, does not perform well when the features are highly correlated or exist as groups. Specifically, instead of selecting all the relevant groups of features, typically only one feature from each group will be selected. This can be problematic in many applications. For example, in bioinformatics, it is known that genes function in co-regulated groups. The discovery of the entire group is thus essential to the understanding of pathways in gene regulatory networks. In [1], [2], we proposed an efficient feature grouping method in which feature groups can be automatically discovered simultaneously with regression shrinkage, without the need to pre-specify the grouping structure as in group lasso. Computationally, we show that its key step can be solved by a simple iterative group merging algorithm which is highly efficient. Experimental results on a number of toy and real-world data sets demonstrate that it achieves comparable accuracy with state-of-the-art methods on both regression and classification problems.

As for multitask learning (MTL) methods, traditional techniques require assuming that all the tasks are related, which can lead to negative transfer when this assumption is violated. Hence, besides grouping features together during learning as discussed above, it is also desirable to group the tasks. Existing methods model these task relationships at the task level, and the features are assumed to always observe the same set of task clusters or covariance structure. This may be too restrictive in some real-world applications. In [3], we proposed a novel MTL formulation that captures task relationships at the feature-level. Depending on the interactions among tasks and features, the proposed method constructs different task clusters for different features, and is thus more fine-grained. Moreover, we do not require the number of clusters to be pre-specified. Instead, depending on the task complexity and usefulness of each feature, different numbers of clusters can be formed for different features. Computationally, the proposed formulation can be efficiently solved by accelerated proximal methods [4]. Empirically, it outperforms existing methods, and the feature-specific task clusters obtained also agree with the known/plausible task structures of the data.

A special form of multi-task learning is multilabel classification, in which the inputs of all the tasks are the same. In many multilabel classification problems, the labels are organized in the form of a tree or directed acyclic graph (DAG). However, current research efforts typically ignore the label dependencies or can only exploit the dependencies in tree-structured hierarchies. In [5], we presented a hierarchical multilabel classification (HMC) algorithm which can be used on both tree- and DAG-structured hierarchies. The key idea is to formulate the search for the optimal consistent multilabel as the finding of the best subgraph in a tree/DAG. Using a simple greedy strategy, the proposed algorithm is computationally efficient, easy to implement, does not suffer from the problem of insufficient/skewed training data in classifier training, and can be readily used on large hierarchies. This can be used for both mandatory leaf node prediction [6] and non-mandatory leaf node prediction [5], and can also be extended to hierarchy-aware loss function that is more appropriate for HMC [7]. Besides, when

text



bioinformatics

hyperspectral

image

- feature extraction / feature selection

Feature Selection

1. filter approach
   - as preprocessing step (no interaction with learning algorithm)
2. wrapper approach
   - use the learning algorithm to score feature subsets
3. embedded approach
   - perform feature selection and learning simultaneously $\rightarrow$ sparse solution

Regularized risk minimization

$$\text{minimize} \quad loss\ \ell(w) + \text{sparsity-inducing regularizer}\ \Omega(w)$$

Lasso (Tibshirani, 1996)

- $\Omega(w) = \|w\|_1$

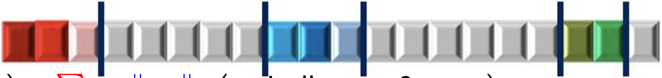
- highly correlated features $\rightarrow$ tends to arbitrarily select only one of them

Structured Sparsity

Features often have intrinsic structures → Structured sparsity

Group lasso (Yuan & Lin, 2006)

- e.g., represent a categorical feature by a group of dummy binary features



- $\Omega(w) = \sum_g \eta_g \|w_g\|_p$ (typically, $p = 2$ or ∞)
    - w_g: subvector of w for the group g

Fused lasso (Tibshirani et al., 2005)

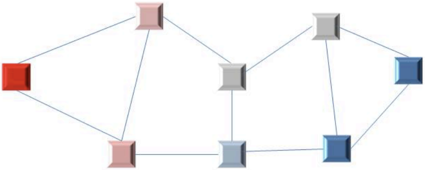
- features are ordered in some sequential way (e.g., time)

- $\Omega(w) = \|w\|_1 + \lambda \sum_{i=1}^{d-1} |w_i - w_{i+1}|$
- encourages sparsity and successive feature coefficients to be similar

Structured Sparsity...

Graph lasso

- features are ordered in a graph $G = (V, E)$



- $\Omega(w) = \|w\|_1 + \sum_{(i,j)\in E} |w_i - w_j|$
- encourages coefficients for nearby graph nodes to be similar

Requires the structure to be known in advance

- group structure / sequential ordering / graph

- may not be available

Sparse Modeling with Automatic Feature Grouping

Goal

Feature coefficients

1. sparse
2. grouped automatically
3. have similar magnitudes for features in the same group

Example

- group of dummy variables for the same categorical variable
- protein-protein interaction networks: groups of co-regulated genes
- text classification: groups of correlated words

Advantages

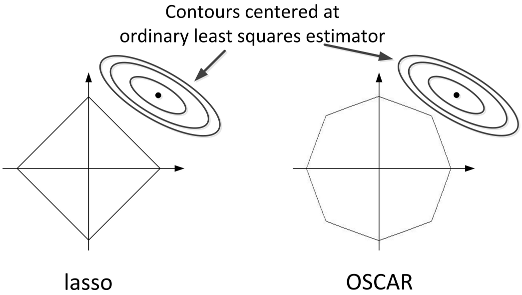
- variance reduction $\longrightarrow$ better accuracy
- simpler model $\longrightarrow$ better interpretation

OSCAR Regularizer (Bondell and Reich, 2008)

Octagonal Shrinkage and Clustering Algorithm for Regression

$$\Omega(w) = \|w\|_1 + \lambda \sum_{i<j} \max\{|w_i|, |w_j|\}$$

- ℓ_1-regularizer: encourages sparsity
- (convex) pairwise ℓ_∞-regularizer: tries to tie every coefficient pair $|w_i|, |w_j|$ together



constraint regions

Challenging Optimization Problem

$$\min_w \|y - Xw\|^2 + \lambda_1 \|w\|_1 + \lambda_2 \sum_{i<j} \max\{|w_i|, |w_j|\}$$

- convex, but difficult to solve
- $O(d^2)$ terms (d: dimensionality)

Tailor-made solvers (Bondell & Reich, 2008)

- quadratic program with $O(d^2)$ variables and $O(d^2)$ constraints

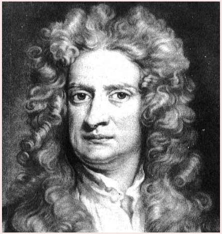
Generic solvers

- ProxFlow algorithm (Mairal et al., 2010)
- complexity $O(d^5)$

particularly challenging on high-dimensional data

How to Solve this Optimization Problem?

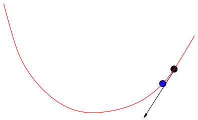
What would Isaac Newton do?



Newton's method!

Back to Basics: Gradient Descent

$\ell(w) + \lambda\Omega(w)$



LOOP

1. find descent direction
2. choose stepsize
3. descent

Problem: ℓ and/or Ω may be nonsmooth

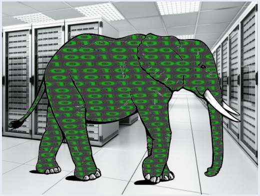
- SVM: hinge loss (nonsmooth) $+\|w\|_2^2$ (smooth)
- lasso: square loss (smooth) $+\|w\|_1$ (nonsmooth)

- extend gradient to nonsmooth functions $\rightarrow$ subgradient

(Sub)Gradient Descent

Advantages

- easy to implement
- low per-iteration complexity $\longrightarrow$ good scalability (BIG data)



Disadvantage

- uses first-order (gradient or subgradient) information
- slow convergence rate (especially for nonsmooth objectives) $\longrightarrow$ may require a large number of iterations

Accelerated Gradient Methods

First developed by Nesterov in 1983

- for smooth optimization

$$\min_w \; f(w) \quad (f \text{ is smooth in } w)$$

ℓ and/or Ω may be nonsmooth

Extension to composite optimization (Nesterov, 2007)

- objective has both smooth and nonsmooth components

$$\min_w \underbrace{f(w)}_{smooth} + \underbrace{r(w)}_{nonsmooth}$$

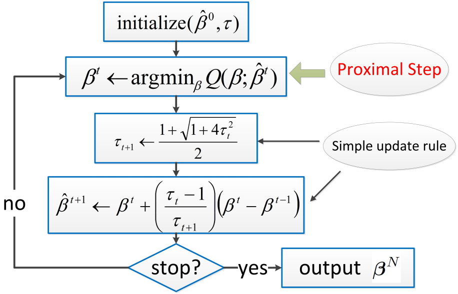
- recently popularly used in machine learning

FISTA (Beck & Teboulle, 2009)

Gradient descent on $f(w)$

$$\hat{w}^t - \frac{1}{L}\nabla f(\hat{w}^t) = \arg\min_w (w - \hat{w}^t)^T \nabla f(\hat{w}^t) + \frac{L}{2}\|w - \hat{w}^t\|^2$$

For $f(w) + \Omega(w)$



$$\arg\min_w Q(w, \hat{w}^t) \equiv (w - \hat{w}^t)^T \nabla f(\hat{w}^t) + \frac{L}{2}\|w - \hat{w}^t\|^2 + \Omega(w)$$

- L: Lipschitz constant of ∇f

Accelerated Convergence

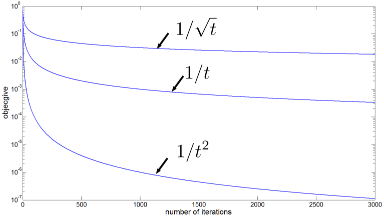
Convergence rate

gradient descent

- smooth objective: $O(1/t)$
- nonsmooth objective: $O(1/\sqrt{t})$

accelerated gradient descent

- $O(1/t^2)$



Proximal Step for OSCAR

$$\min_w \underbrace{\|y - Xw\|^2}_{\ell(w)} + \underbrace{\lambda_1 \|w\|_1 + \lambda_2 \sum_{i<j} \max\{|w_i|, |w_j|\}}_{\Omega(w) nonsmooth}$$

How to efficiently compute the proximal step
$\min_w \ (w - \hat{w}^t)^T \nabla \ell(\hat{w}^t) + \frac{L}{2}\|w - \hat{w}^t\|^2 + \Omega(w)$?

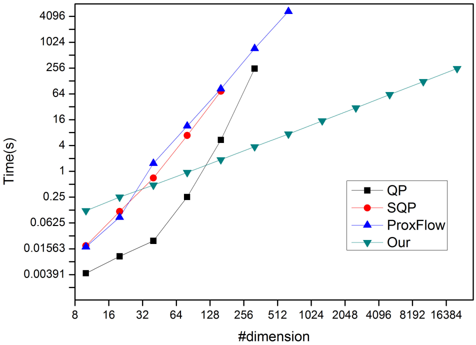
can be solved in $O(d \log d)$ time!

Total time: $O\left(\frac{1}{\sqrt{\epsilon}}(dn + d \log d)\right)$

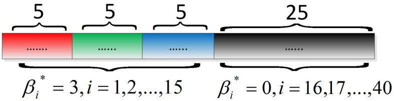
- $\frac{1}{\sqrt{\epsilon}}$: convergence rate of FISTA
- dn: time to compute gradient of $\|y - Xw\|^2$
- $d \log d$: time for proximal step
- typically, $n \gg \log d \rightarrow$ time $= O\left(\frac{1}{\sqrt{\epsilon}} dn\right)$

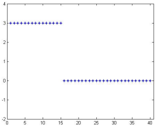
Experiment: Speedup

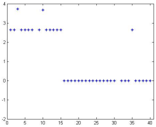
- sample size $n = 1000$, dimension $d = \{10, 20, \ldots, 20480\}$
- compare with QP, SQP (Bondell & Reich, 2008) and ProxFlow (Mairal et al., 2010)

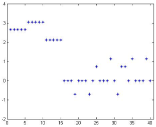


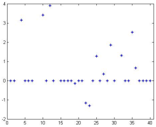
Recovers the Feature Structure




(a) ground truth


(b) OSCAR


(c) fused lasso


(d) lasso

- OSCAR can select relevant and correlated features

Breast Cancer Data

- 300 genes (features); 295 tumors (examples)

	lasso	fused lasso	elastic net	OSCAR
test accuracy	65.9	72.4	71.8	74.1
#nonzero features	40.0	217.2	147.0	103.8

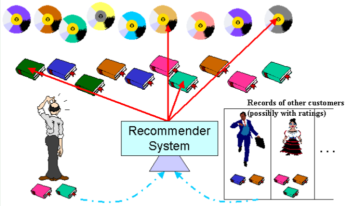
[Zhong & Kwok. Efficient sparse modeling with automatic feature grouping. **ICML**-2011, and **TNNLS**-2012]

Big Problems

- often involves multiple learning tasks
- these tasks are related (share some information)

Example

digit recognition



product recommendation

- harness the task relationships → learn all tasks together
- allow tasks to borrow strength from each other

Multitask Learning (MTL)

Regularized risk minimization

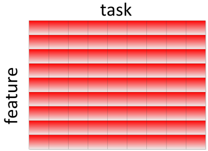
- tasks $1, 2, \ldots, T$
- weights $w_1, w_2, \ldots, w_T$
- add another regularizer

$$\min_{w_1,\ldots,w_T} \sum_{t=1}^{T} \left(\underbrace{\ell_t(w_t)}_{\text{loss}} + \underbrace{\Omega_t(w_t)}_{\text{task-specific regularizer}} \right) + \underbrace{\Omega_{\mathrm{MTL}}(w_1, w_2, \ldots, w_T)}_{\text{MTL regularizer}}$$

Different Assumptions for Ω_{MTL}

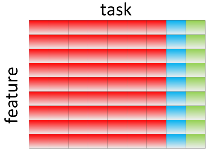
Assume: w_t's form one group of correlated tasks

- regularized MTL



Assume: w_t's form one cluster with a few outliers

- robust MTL

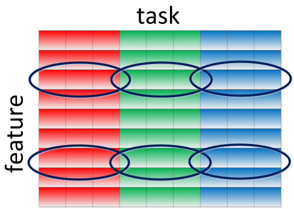


Unlikely to hold with a BIG number of tasks

Task Grouping

Assume: w_t's form clusters

- clustered MTL



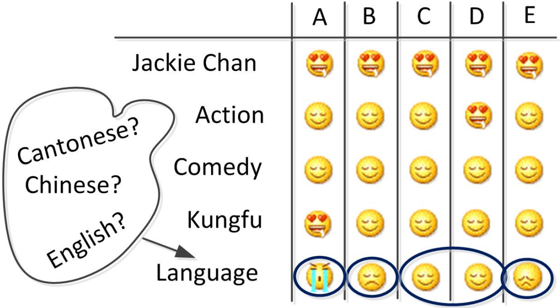
How many clusters?

- needs to be fixed beforehand in clustered MTL

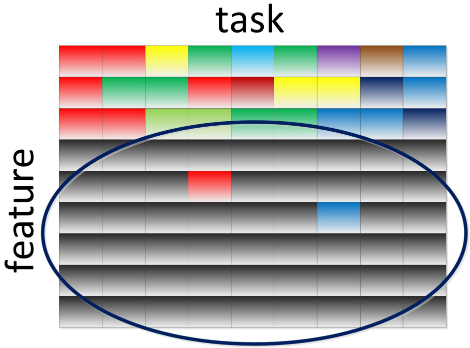
Every feature sees the same clustering structure

- flexible enough?

Motivating Example: Movie Recommendation



Example 2: Features with Different Discriminating Power



Goal

Still want the tasks to be clustered, but

- task cluster structure can vary from feature to feature
- infer the clustering structure automatically

Idea

extend the feature clustering idea to task clustering in MTL

Flexible Task-Clustered MTL

1. decompose each w_t into $u_t + v_t$
   - $u_1, u_2, \ldots, u_T$: clustered
   - v_t: task-specific variation
2. cluster the u_t's feature by feature
   - for each feature d, minimize $|u_{i,d} - u_{j,d}|$ for all tasks i and j

$$\min_{U,V} \underbrace{\sum_{t=1}^{T} \|y_t - X_t(u_t + v_t)\|^2}_{\text{loss}} + \lambda_1 \|U\|_{clus} + \underbrace{\lambda_2 \|U\|_F^2 + \lambda_3 \|V\|_F^2}_{\text{ridge regularizers}}$$

- $U = [u_1, \ldots, u_T], V = [v_1, \ldots, v_T]$
- $\|U\|_{clus} = \sum_{d=1}^{D} \sum_{i<j} |u_{i,d} - u_{j,d}|$
  - a convex relaxation of k-means clustering on each feature

Special Cases

$$\min \sum_{t=1}^{T} \|y_t - X_t(u_t + v_t)\|^2 + \lambda_1 \|U\|_{clus} + \lambda_2 \|U\|_F^2 + \lambda_3 \|V\|_F^2$$

1. $\lambda_1 = \lambda_2 = \lambda_3 = 0$: independent LS regression on each task
2. $\lambda_1 = \infty$: regularized MTL (Evgeniou et al, 2005)
3. $\lambda_1 = 0$: independent ridge regression on each task

Nice Properties

Theoretical

With high probability, for large enough sample size,

- the obtained task weights converge to the ground truth
- U captures the clustering structure
  - tasks i, j are in the same cluster for feature d $\rightarrow$ $u_{i,d} = u_{j,d}$
  - tasks i, j are in different clusters $\rightarrow$ $u_{i,d} \neq u_{j,d}$

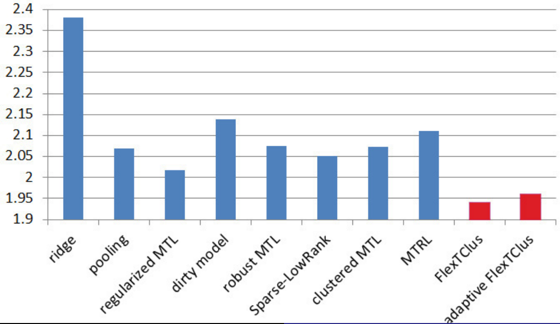
Computational

- optimization using FISTA
- similar to feature grouping, the proximal step can be efficiently computed in $O(TDn + DT\log T)$ time
  - T: number of tasks
  - D: feature dimension
  - n: sample size

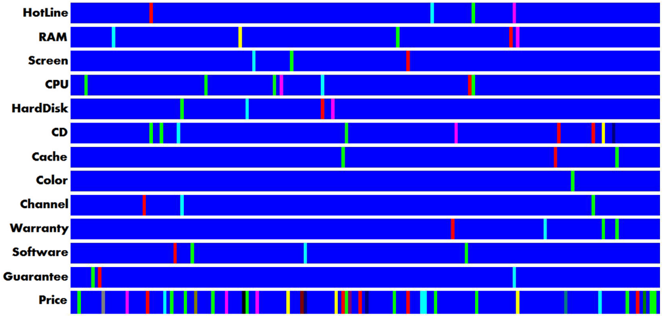
Experiment: Product Rating

Predict the ratings of students (tasks) on personal computers (each described by 13 attributes)

Root mean squared error



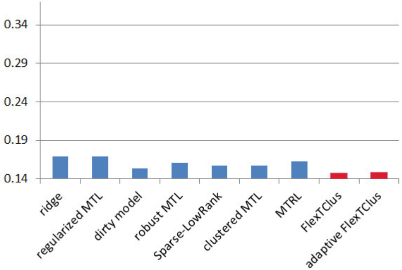
Product Rating: Task Clustering Structure

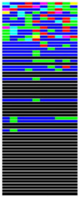


- one main cluster for the first 12 attributes (related to performance & service)
- lots of varied opinions on the last attribute (price)

Experiment: Digit Recognition

- 10-class classification problem → 10 1-vs-rest problems
- use PCA features





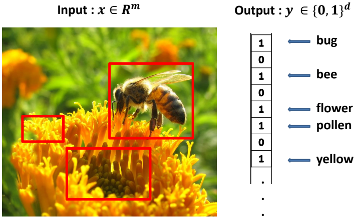
- FlexTClus has the lowest classification error
- leading PCA features are more discriminative; trailing PCA features form one cluster close to zero (black)

[Zhong & Kwok. Convex multitask learning with flexible task clusters. **ICML**-2012]

What if there are Many Many Tasks?

Multilabel classification

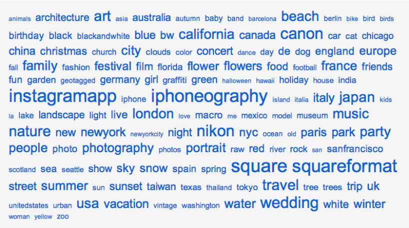
- an instance can have more than one labels



- cf. multiclass classification: an instance can have only one label
- basic approach: one label, one task (binary relevance)

Too Many Tasks

Flickr: > 20 million unique tags (labels) in 2010



Too Many Tasks...

Open Directory Project: the largest human-edited Web directory

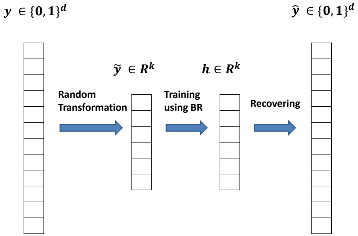
- over 4 million websites
- 787,774 categories (labels)

Top: Computers *(94,925)*

- Computer Science *(1,876)*
- Hardware *(5,205)*
- Internet *(24,521)*
- Security *(2,575)*
- Software *(27,390)*
- Systems *(2,570)*

- Algorithms *(303)*
- Artificial Intelligence *(1,188)*
- Artificial Life *(208)*
- Bulletin Board Systems *(93)*
- CAD and CAM *(731)*
- Computer and Technology Law@ *(105)*
- Data Communications *(948)*
- Data Formats *(1,679)*
- Desktop Publishing *(113)*
- E-Books *(155)*
- Emulators *(375)*
- Fonts@ *(272)*
- Games@ *(31,921)*
- Graphics *(1,243)*
- Hacking *(167)*
- Home Automation *(67)*
- Human-Computer Interaction *(261)*
- Intranet *(39)*
- MIS@ *(375)*
- Mobile Computing *(486)*
- Multimedia *(2,455)*
- Newsgroups@ *(275)*
- Open Source *(699)*
- Operating Systems@ *(7,745)*
- Parallel Computing *(321)*
- Performance and Capacity *(43)*
- Programming *(14,397)*
- Robotics *(722)*
- Speech Technology *(372)*
- Supercomputing *(35)*
- Usenet *(275)*
- Virtual Reality *(330)*

Label Transformation

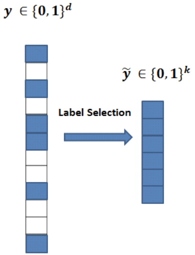


1. projects the d-dimensional label vector to a k-dimensional space, where $k \ll d$
2. learn a regression model for each dimension of the transformed label vector
3. predict in the low-dimensional space, then back-project to the d-dimensional space

The transformed labels, though fewer in quantity, may be more difficult to learn

Label Selection

- selects only a few output labels for training; reconstruct the other output labels from this label subset



- label subset comes from the original labels $\rightarrow$ learning problems will not be more difficult

How to find the label subset?

- optimization (group-sparse learning problem)
- expensive, esp. with a lot of labels

Column Subset Selection Problem

Finding the label subset is a column subset selection problem

Column Subset Selection Problem (CSSP)

- given: matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$; positive integer k
- find C (k columns of $\mathbf{A}$) that spans $\mathbf{A}$ as much as possible
  - $\mathbf{A}_C$: submatrix of $\mathbf{A}$ with columns indexed by C

$$\min_{C:|C|=k} \|\mathbf{A} - \underbrace{\mathbf{A}_C \mathbf{A}_C^{\dagger}}_{\text{project onto } \mathbf{A}_C} \mathbf{A}\|_F$$

- in our case, $\mathbf{A}$ is just the label matrix $\mathbf{Y}$!
  - $\mathbf{Y}$: each column is a label, each row is a sample

Existing CSSP Solvers

- perform random sampling of the columns
- sampling probability is based on the leverage score $p_i = \|(\mathbf{V}_k^T)_{(i)}\|_2^2/k$
  - $\mathbf{V}_k$: top k right singular vectors of $\mathbf{Y}$; $(\mathbf{V}_k^T)_{(i)}$: ith column of $\mathbf{V}_k^T$
  - leverage has been used to detect outliers in regression analysis $\rightarrow$ importance of each sample

1. (Drineas et al., 2006): sample $O(k^2)$ columns
   - a lot more than what we need (which is k)
2. (Boutsidis et al., 2009)
   - sample $\Theta(k \log k)$ columns; then post-process to get k columns
   - post-processing can be even more computationally expensive than the sampling step itself!

Multilabel Classification via CSSP (ML-CSSP)

1: compute the sampling probability p_i for each column in $\mathbf{Y}$ based on leverage;
2: $C \leftarrow \emptyset$;
3: **while** $|C| < k$ **do**
4: sample with replacement a column from $\mathbf{Y}$ using p_i's;
5: **if** $i \notin C$ **then**
6: $C \leftarrow C \cup \{i\}$;
7: **end if**
8: **end while**
9: train classifiers for the k selected labels.

Some Properties

With high probability, $\|\mathbf{Y} - \mathbf{Y}_C \mathbf{Y}_C^\dagger \mathbf{Y}\|_F \leq \text{constant} \times \|\mathbf{Y} - \mathbf{Y}_k\|_F$

- $\mathbf{Y}_k$: best rank-k approximation of $\mathbf{Y}$

With high probability, we can get k different columns in $O(k \log k)$ sampling rounds

- may still need to sample $O(k \log k)$ columns in the worst case
- empirical results show much smaller number

Time complexity

(Drineas et al., 2006)	$O(ndk)$	+	$O(k^2)$		
(Boutsidis et al., 2009)	$O(ndk)$	+	$O(k \log k)$	+	$O(k^3 \log^2 k \log(k \log k))$
ours	$O(ndk)$	+	$O(k \log k)$		

Can also be kernelized

Experiments

Compare ML-CSSP with

1. label transformation methods: PLST (Tai & Lin, 2012), CPLST (Chen & Lin, 2012) , CL (Zhou et al., 2012)
2. label selection method: MOPLMS (Balasubramanian & Lebanon, 2012)
3. Binary relevance (BR)

data set	#samples	#features	#labels
cal500	502	68	174
corel5k	5,000	499	374
delicious	16,105	500	983
EUR-Lex (dc)	19,348	5,000	412
EUR-Lex (desc)	19,348	5,000	3,993
dmoz	394,756	829,208	35,437

RMSE and AUPRC

RMSE

data set	ML-CSSP	MOPLMS	PLST	CPLST	CL	BR
cal500	4.93	5.04	4.97	5.00	5.70	5.06
corel5k	**1.89**	**1.89**	1.91	1.91	2.71	1.91
delicious	4.29	-	4.27	**4.26**	5.58	**4.26**
EUR-Lex (dc)	**1.22**	-	**1.22**	**1.23**	2.03	1.50
EUR-Lex (desc)	**2.93**	-	3.02	3.06	4.51	3.51
dmoz	**2.83**	-	2.95	-	-	4.02

AUPRC (Area Under the Precision-Recall Curve)

data set	ML-CSSP	MOPLMS	PLST	CPLST	CL	BR
cal500	**0.500**	0.459	0.488	0.412	0.169	0.442
corel5k	**0.089**	0.080	0.079	0.082	0.011	0.083
delicious	0.220	-	0.182	0.227	0.089	**0.237**
EUR-Lex (dc)	**0.180**	-	**0.180**	0.167	0.036	0.173
EUR-Lex (desc)	**0.094**	-	0.018	0.086	0.016	0.086
dmoz	**0.016**	-	0.001	-	-	0.012

- many methods cannot be run because of the large number of labels (marked “-”)
- ML-CSSP obtains better performance

Encoding Time

- time to perform label transformation/selection

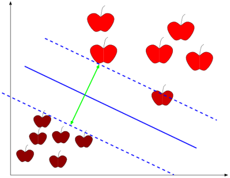
data set	ML-CSSP	MOPLMS	PLST	CPLST	CL
cal500	0.0	10.7	0.0	0.0	182.0
corel5k	0.2	36.8	0.2	0.3	292.0
delicious	11.6	-	11.6	16.0	5675.2
EUR-Lex (dc)	4.0	-	4.0	352.9	547.1
EUR-Lex (desc)	153.8	-	153.8	511.7	15585.5
dmoz	1428.9	-	1428.7	-	-

- ML-CSSP is almost as efficient as the fastest one (PLST)

[Bi & Kwok. Efficient multi-label classification with many labels. **ICML**-2013]

Big Sample Size

ML tool: Kernel methods



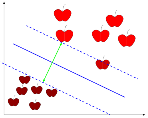
Example (big data problem)

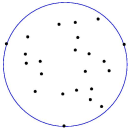
SVM on n training examples:

- $O(n^2)$ memory for the $n \times n$ kernel matrix
- inverting/eigenvalue decomposition of the kernel matrix $\longrightarrow$ $O(n^3)$ time

Core Vector Machine

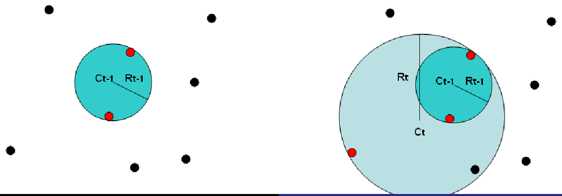
1. SVM training is a minimum enclosing ball (MEB) problem





2. near-optimal solutions are good enough in practical applications $\longrightarrow$ efficient approximation algorithms

- at each iteration, expand the current ball $B(c_t, r_t)$ by including the furthest point
- repeat until all the points are covered by $B(c_t, (1+\epsilon)r_t)$



CVM in Classification

KDDCUP-99 intrusion detection data set (4,898,431 patterns)

method		# train patns input to SVM	# test errors	SVM training time (in sec)	other proc time (in sec)
random sampling	0.001%	47	25,713	0.000991	500.02
	0.01%	515	25,030	0.120689	502.59
	0.1%	4,917	25,531	6.944	504.54
	1%	49,204	25,700	604.54	509.19
	5%	245,364	25,587	15827.3	524.31
active learning		747	21,634	94192.213	
CB-SVM		4,090	20,938	7.639	4745.483
CVM		4,898,431	19,513	1.4	

- CVM is fast and accurate

[Tsang, Kwok, Cheung. Core vector machines: Fast SVM training on very large data sets. **JMLR**, 2005]

Potential Limitations

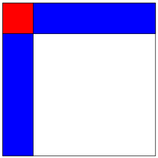
Still need to solve a QP in finding the MEB of the core set

- very large data set $\longrightarrow$ large core set $\longrightarrow$ large QP

Low-Rank Approximation: Nyström Algorithm

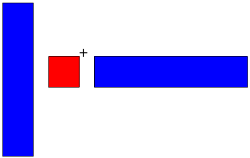
Sample m columns from matrix K

- $C = \begin{bmatrix} W \\ S \end{bmatrix} \quad K = \begin{bmatrix} W & S^T \\ S & B \end{bmatrix}$



Rank-k Nyström approximation:

$CW_k^+ C^T$



Time complexity: $O(nmk + m^3)$

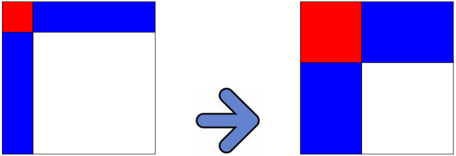
- $m \ll n \rightarrow$ much lower than the $O(n^3)$ complexity

How to Choose the Columns?

1. random sampling
2. probabilistic
   - chooses the columns based on a data-dependent probability
3. greedy approach
4. clustering-based
   - inexpensive; with interesting theoretical properties

[Zhang & Kwok. Clustered Nystrom method for large scale manifold learning and dimension reduction. **TNN**, 2010]

Tradeoff between Accuracy and Efficiency



- more columns sampled, more accurate is the approximation

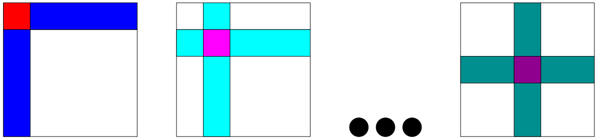
Example (data set with several millions samples)

- sampling only 1% of the columns
- W larger than $10,000 \times 10,000$
- SVD on W dominates and becomes prohibitive

Ensemble Nyström Algorithm (Kumar et al., 2009)

Replace the large SVD by a number of small SVDs

- ensemble of n_e Nyström approximators, each samples m columns



- each expert performs a standard Nyström approximation
- linearly combine the n_e rank-k approximations $\tilde{G}_{1,k}, \tilde{G}_{2,k}, \ldots, \tilde{G}_{n_e,k}$

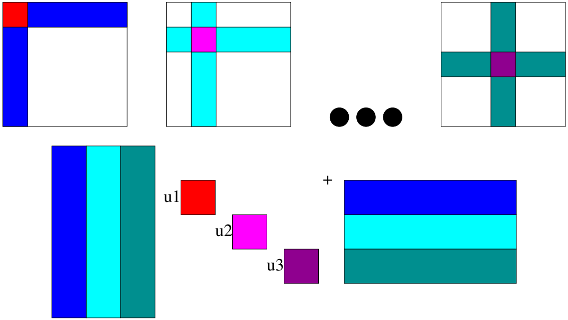
$$\tilde{G}^{ens} = \sum_{i=1}^{n_e} \mu_i \tilde{G}_{i,k} \quad (\mu_i\text{'s: mixture weights})$$

Ensemble Nyström Algorithm...

Time complexity

- $O(n_e nmk + n_e m^3 + C_\mu)$
  - C_μ: cost of computing the mixture weights
  - serial implementation: (roughly) n_e times that of Nyström
  - parallel implementation: (roughly) similar to Nyström

Implicitly, approximate $W^+ \in \mathbb{R}^{n_e m \times n_e m}$ by a block diagonal matrix

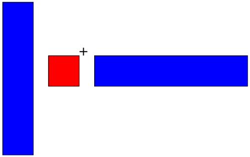


- less accurate

Nyström + Randomized SVD

Idea

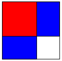
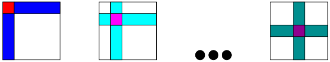
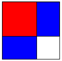
1. sample more columns (like ensemble Nyström)
2. use a SVD approximation that is more accurate but still relatively efficient



- standard Nyström: SVD ($O(m^3)$ complexity)
- ensemble Nyström: block diagonal assumption
- our proposal: randomized SVD ($O(m^2 k + k^3)$ complexity)

Is it Efficient?

Recall that $n \gg m \gg k$

Nyström 	$O(nmk + m^3)$
ensemble Nyström 	$O(nmk + n_e k^3 + C_\mu)$
randomized SVD	$O(n^2 k + k^3)$
Nyström + rand. SVD 	$O(nmk + m^2 k + k^3) = O(nmk + k^3)$

Is it Accurate?

Rank-k standard Nyström approximation $\hat{K}$ (with m randomly sampled columns)

$$\mathbb{E}\,\|K - \hat{K}\|_2 \leq \|K - K_k\|_2 + \frac{2n}{\sqrt{m}}(\max_i K_{ii})$$

- K_k: best rank-k approximation

Proposed method

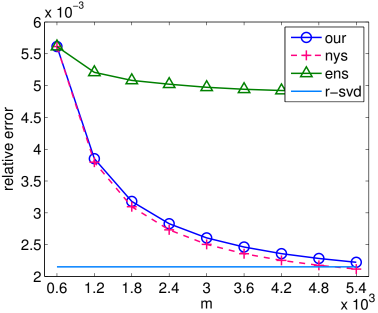
$$\mathbb{E}\,\|K - \hat{K}\|_2 \leq \zeta^{1/q}\|K - K_k\|_2 + (1 + \zeta^{1/q})\frac{n}{\sqrt{m}}(\max_i K_{ii})$$

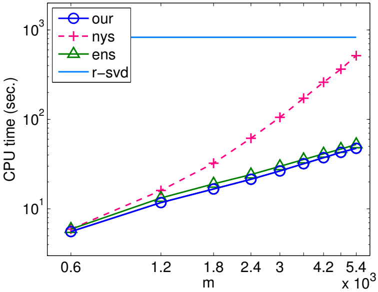
- ζ: constant and $\zeta^{1/q}$ close to 1
  $\rightarrow$ becomes $\|K - K_k\|_2 + \frac{2n}{\sqrt{m}}(\max_i K_{ii})$, same as that for standard Nyström

Proposed method is as accurate as standard Nyström

Experiment

x-axis: number of sampled columns



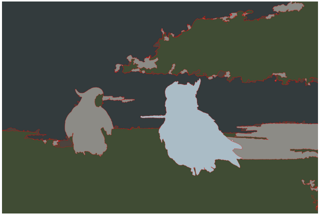


	accuracy	cost
randomized SVD	✓	✗
standard Nyström	✓	✗
ensemble Nyström	✗	✓
proposed method	✓	✓

Image Segmentation

- Intel Xeon X5540 CPU, 16GB memory (matlab)

3872 × 2592 (10M pixels)



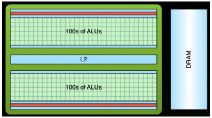
16.8 sec

4752 × 3168 (15M pixels)

22.6 sec

Graphics Processors (GPU)

Popularly used in entertainment, high-performance computing



NVIDIA Tesla C1060

- 240 streaming processor cores
- peak single-precision (SP) performance: 933 GFLOPS
- peak double-precision (DP) performance: 78 GFLOPS

Intel Core i7-980X CPU

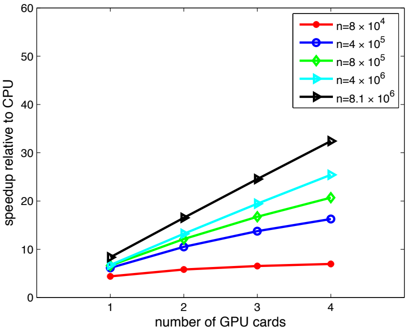
- 6 cores
- peak SP: 158.4 GFLOPS
- peak DP: 79.2 GFLOPS

Experiments on GPU

Machine

- two Intel Xeon X5560 2.8GHz CPUs, 32G RAM
- four NVIDIA Tesla C1060 GPU cards

Linear speedup with number of GPUs (MNIST-8M data set)



Different Numbers of Sampled Columns m

m	time (sec) run on CPU	run on 4 GPUs	(speedup)	relative approx. error
2,000	908.1	24.5	(37.1x)	5.9×10^{-4}
4,000	1,789.4	47.2	(37.9x)	5.5×10^{-4}
6,000	2,647.8	81.7	(32.4x)	5.3×10^{-5}
8,000	3,556.5	104.8	(33.9x)	5.4×10^{-5}
10,000	4,426.4	119.5	(37.0x)	3.0×10^{-5}
20,000	8,988.2	253.8	(35.4x)	1.1×10^{-5}

More GPU Comparisons

- GPU versions of standard Nyström and ensemble Nyström

Time

number of samples (n)	Nyström	ours	ensemble Nyström
4×10^5	244.9	12.7	25.3
8×10^5	368.8	22.4	39.3
4×10^6	2,041.7	94.2	46.5
8.1×10^6	6,884.0	81.7	81.2

Relative approximation error

n	Nyström	ours	ensemble Nyström
4×10^5	8.5×10^{-5}	9.4×10^{-5}	4.4×10^{-4}
8×10^5	8.2×10^{-5}	8.3×10^{-5}	4.5×10^{-4}
4×10^6	9.0×10^{-6}	9.1×10^{-6}	6.9×10^{-5}
8.1×10^6	7.8×10^{-6}	7.8×10^{-6}	6.9×10^{-5}

	accuracy	cost
standard Nyström	✓	✗
ensemble Nyström	✗	✓
proposed method	✓	✓

Conclusion

Big data is difficult to handle

- high-dimensional
- involves a lot of tasks (related in some complex manner)
- big sample size
- high velocity

But there is hope

- better models
  - OSCAR, flexible task clustering, transfer learning
- better tools: optimization solvers and approximations
  - accelerated gradient descent, column subset selection solvers, Nyström algorithm, stochastic algorithms, online algorithms
- better hardware
  - GPU, parallel/distributed architecture

I seem to have been only like a boy playing on the seashore, ...
whilst the great ocean of truth lay all undiscovered before me.
Isaac Newton

I seem to have been only like a boy playing on the seashore, ...
whilst the great ocean of data lay all undiscovered before me.
Modern day Isaac Newton