A Quest for Structure:
Jointly Learning Graph Structure & Semi-Supervised Classification



Xuan Wu*, Lingxiao Zhao*, Leman Akoglu

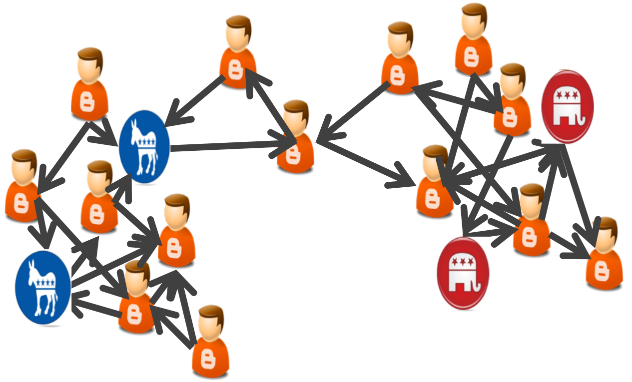
*: Equal Contribution

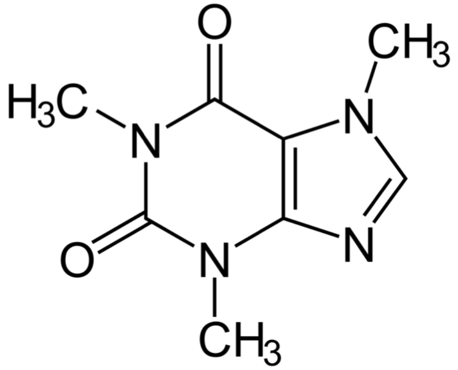
Agenda

- **Problem Introduction:**
  - Motivation for Learning Graph
  - Graph-based Semi-supervised Learning (SSL)
  - Existing Solution For Getting Graph for SSL
- **PG-Learn**: Parallel Graph Learning for SSL
  - Gradient-based Graph Learning for SSL
  - Adaptive Parallel Search
- **Empirical Evaluation**
  - Datasets & Baselines
  - Result

Motivation

- **Explicit, well defined graph**
  - Limited and have noise
  - Usually just connections (No weights)
  - “Right” graph for any tasks? No!



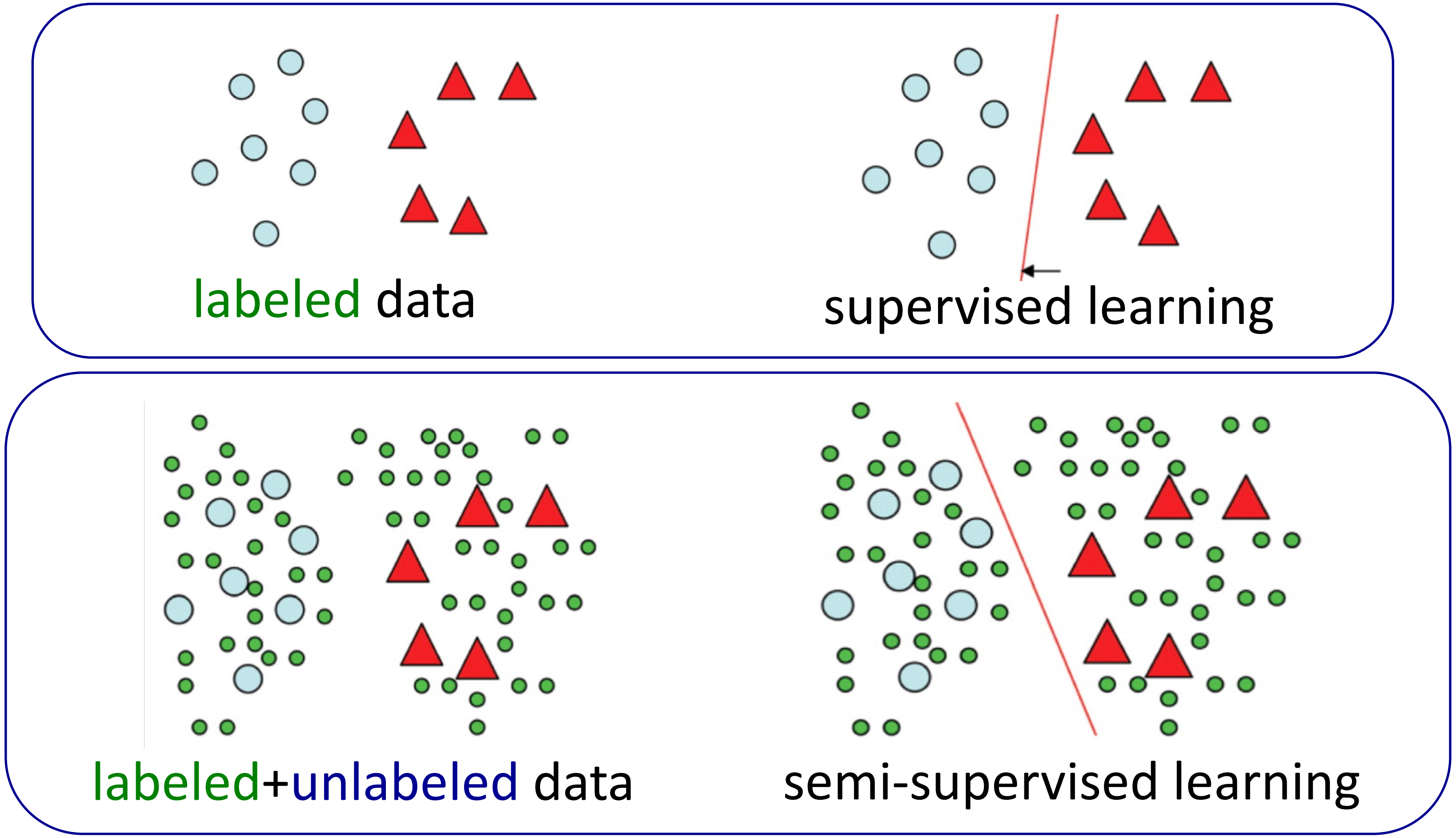


- **Implicit graph**
  - Not given the data
  - Need to be constructed based on domain knowledge
  - Needed for lots of algorithms

Question: how to learn a graph for a particular task, from raw, high-dimensional, and noisy data?

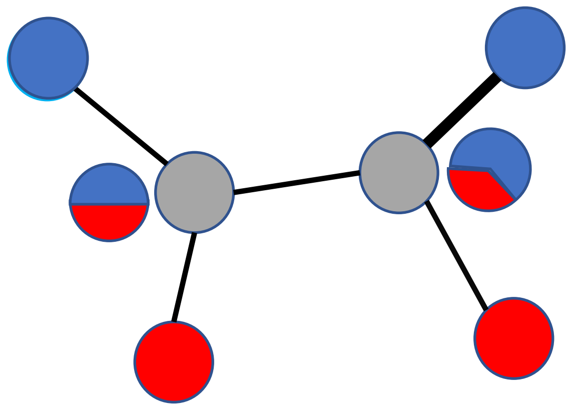
Background: SSL

- **Semi-supervised Learning**



Background: Graph-based SSL

- **Given**
  - set **L** of labeled nodes
  - set **U** of unlabeled nodes
  - a graph **W** of all nodes



- **Assign**
  - Label **Y** or Class Probability **F** to unlabeled nodes $T = L \cup U$
- **Solution**

$$\arg\min_{F \in \mathbb{R}^{n \times c}} tr((F - Y)^T (F - Y) + \alpha F^T L F)$$

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$$

$$D := diag(\mathbf{W}1_n)$$

Closed-form

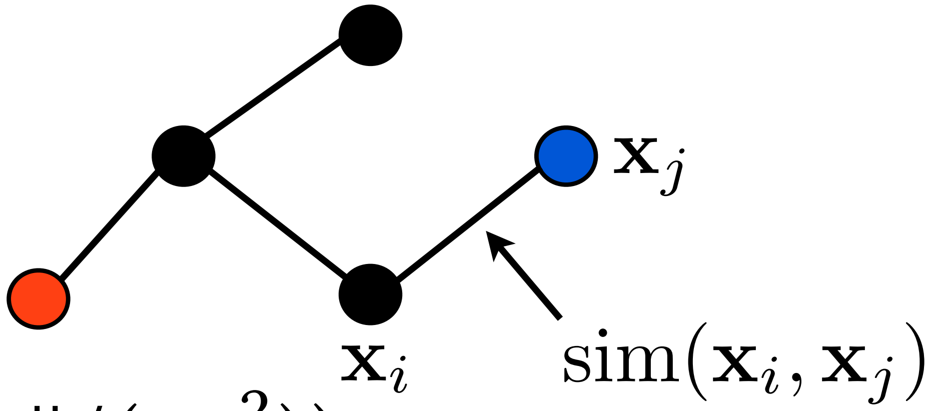
$$\mathbf{F}^* = (\mathbf{I} + \alpha\mathbf{L})^{-1}\mathbf{Y}$$

Iterative solution

$$F^{(t+1)} \leftarrow \mu P F^{(t)} + (1 - \mu) Y$$

What you would do for W?

Most typical way:



- Getting weights between pairs by their “similarity”, using RBF kernel

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|/(2\sigma^2))$$

- Sparsification
  - ε-neighborhood
  - kNN
- Hyperparameters: (σ, ε) or (σ, k)
  - Random search on cross validation
  - Grid search on cross validation

W Matters!

"SSL algorithms are strongly affected by the graph sparsification parameter value and the choice of the adjacency graph construction and weighted matrix generation methods."

Influence of Graph Construction on Semi-supervised Learning.
Celso Andre R. de Sousa, Solange O. Rezende, Gustavo E. A. P. A. Batista. ECML/PKDD 2013.

Existing Solutions

- **Unsupervised**
  - Locally Linear Embedding [Roweis&Soul *Science* 2000]
  - b-matching [Jebara+ *ICML* 2009]
  - Low-Rank Representation [Liu+ *ICML* 2010]
  - Anchor Graph Regularization [Wang+ *TKDE* 2016]

No use of labels, not graph Learning

- **Supervised**
  - Distance metric learning [Dhillon+ *ACL* 2010]
  - Multiple kernel learning [Li+ *IJCAI* 2016]
  - Constrained self-representation [Zhuang+, Image Proc. 2017]
  - …

Not task-driven and/or scalable

Agenda

- **Problem Introduction:**
  - Motivation for Learning Graph
  - Graph-based Semi-supervised Learning (SSL)
  - Existing Solution For Getting Graph for SSL
- **PG-Learn**: Parallel Graph Learning for SSL
  - Gradient-based Graph Learning for SSL
  - Adaptive Parallel Search
- **Empirical Evaluation**
  - Datasets & Baselines
  - Result

Task-driven
Effective
Scalable
No hyperparameter to tune

Parameterize W More Generally

- **Single bandwidth** is not enough
  - Recall: $\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|/(2\sigma^2))$
  - Different feature may prefer different bandwidth
- **Dimension-specific** kernel bandwidth

$$\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\sum_{m=1}^{d} \frac{(\boldsymbol{x}_{im} - \boldsymbol{x}_{jm})^2}{\sigma_m^2}\right)$$

$$W_{ij} = \exp\left(-(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \boldsymbol{A} (\boldsymbol{x}_i - \boldsymbol{x}_j)\right)$$

$$\boldsymbol{A} := diag(\boldsymbol{a}) \quad \boldsymbol{A}_{mm} = a_m = 1/\sigma_m^2$$

- **Difficulty**
  - Number of parameters: d can be more than **thousands**

random search / grid search won't work

Problem Formulation

- **Given**

$$\mathcal{D} := \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_l, y_l), \boldsymbol{x}_{l+1}, \ldots, \boldsymbol{x}_{l+u}\}$$

- **Infer**
  - $A := diag(\boldsymbol{a})$: bandwidths per dimension
  - k : sparsity of kNN graph (kNN is used to sparse W)

  } **W**

  - Labels for unlabeled points

  } **Task**

Jointly learning a graph W and solving SSL task,
so that W captures "right" structure needed by the task.

Link Quality of W with Task

Task-driven

- **Define a loss g of W over F***

$$\mathbf{F}^* = \arg \min_{F \in \mathbb{R}^{n \times c}} tr((F - Y)^T (F - Y) + \alpha F^T L F)$$

$$= \underline{(\mathbf{I} + \alpha \mathbf{L})^{-1}} \mathbf{Y}$$

A function of W

  - **F*** is "better" means W has better quality

- **Using validation set** $\mathcal{V} \subset \mathcal{L}$
  - "better" means smaller **"difference"** between F* and Y (true label) over validation set.

g(F*) over **validation set** measures the **quality** of **W** of the task

[the smaller the better]

Validation Loss g(F*)

Many ways to define the validation loss

- As long as it can measure the different between F* and Y

  e.g. $g_A(\mathcal{V}) = \sum_{v\in\mathcal{V}}(1 - F_{vc_v})$

- We choose a pairwise ranking-based loss
  - Validation set is quiet small
  - Pairwise **makes full use of** information

$$g_A(\mathcal{V}) = \sum_{c'=1}^{c} \sum_{\substack{(v,v'):\, v\in\mathcal{V}_{c'},\\ v'\in\mathcal{V}\setminus\mathcal{V}_{c'}}} -\log\sigma(F_{vc'} - F_{v'c'})$$

Node inside c

Node outside c

Prob of ranking v above v', based on output F

Minimizing g

Scalable

- **Use gradient descent**
  - F* has closed form, can get gradient w.r.t. W
  - Deriving gradient is omitted, please see our paper
  - Make full use of sparsity
- **Complexity**
  - Computational complexity
  - Memory complexity

$$O(n[kctd + dk^2 + \log n])$$

$$O(knd)$$

k: #NNs, **c**: #classes, **t**: # power method iterations

- linear in dimensionality, log-linear in sample size
- linear in both dimensionality & size

Summarize So Far

Initialize k and $\boldsymbol{a}$ (vector containing a_m's); $t := 0$

repeat

Compute $\boldsymbol{F}^{(t)}$ using kNN graph on current a_m's

Compute gradient $\frac{\partial g}{\partial a_m}$ based on $\boldsymbol{F}^{(t)}$ for each a_m

Update a_m's by $\boldsymbol{a}^{(t+1)} := \boldsymbol{a}^{(t)} - \gamma \frac{\mathrm{d}g}{\mathrm{d}\boldsymbol{a}}$; $t := t + 1$

until a_m's have converged

Adaptive Parallel Search

How about k and initial a?

- Non-convex problem: Different initial point matters
- Sparsity **k** always matters a lot

Solution

- Try many **effective** configurations as much as possible in limited time

A simple & effective idea – **Successive Halving** [Jamieson, AISTATS 2016]

1. pick **a set** of (hyperparameter) configurations
2. run for a fixed amount of time (i.e. iterations)
3. evaluate configurations (metric of interest)
4. keep the **best half** (terminate the worst half)
5. repeat 2. – 4. until **one** configuration remains

} 0^{th} - order

Adaptive Parallel Search

How about k and initial a? No hyperparameter to tune

- Non-convex problem: Different initial point matters
- Sparsity **k** always matters a lot

Solution

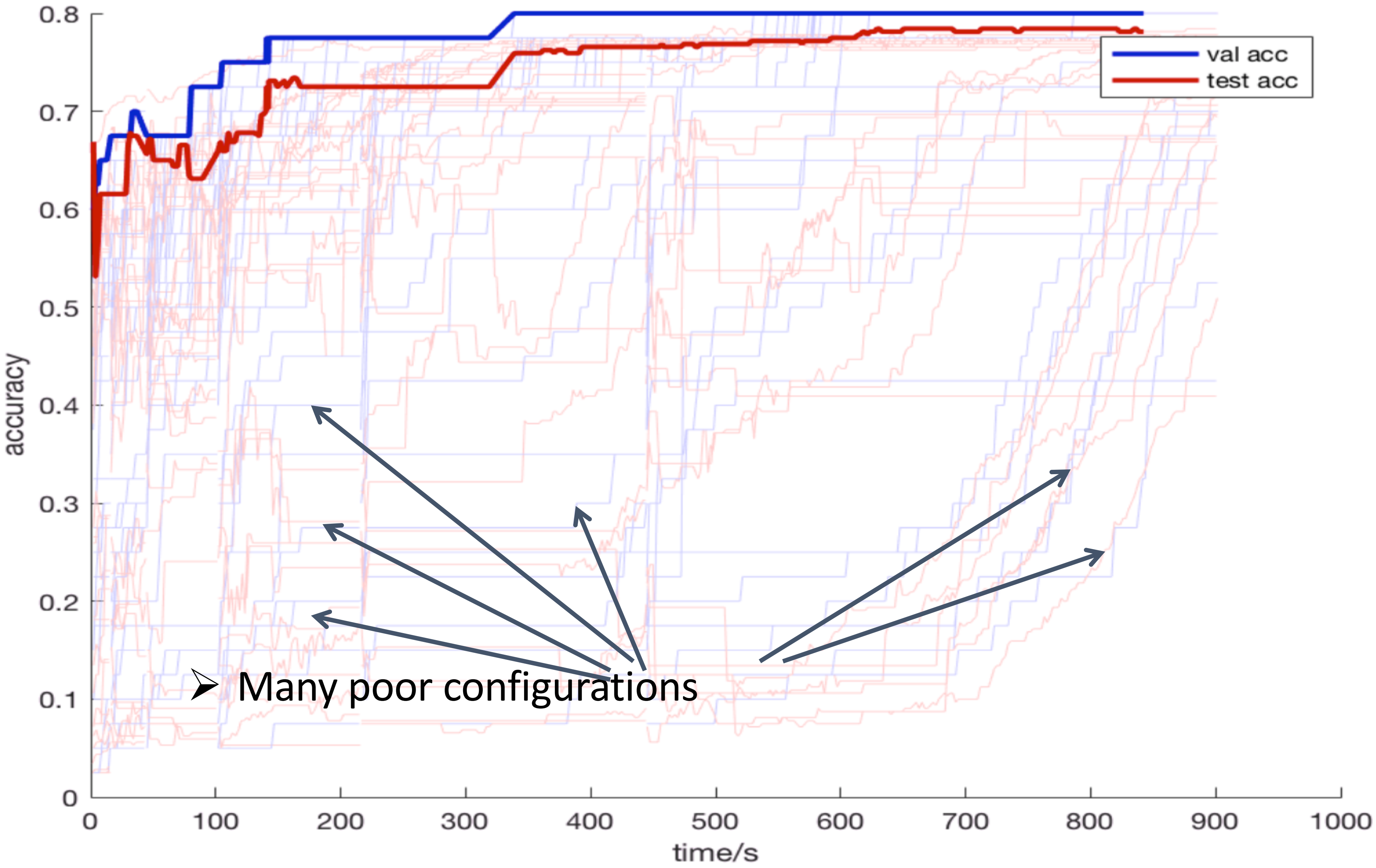
- Try many **effective** configurations as much as possible in limited time.

  A simple & effective idea – **Successive Halving** [Jamieson, AISTATS 2016]
- Improve it by fully parallel

  After **halving**, restart new configurations to **reuse** threads
- And

  Not 0^{th} – order anymore, our solution combined with 1^{st} –order optimization

➢Test accuracy improves by time



➢ Many poor configurations

Agenda

- **Problem Introduction:**
  - Motivation for Learning Graph
  - Graph-based Semi-supervised Learning (SSL)
  - Existing Solution For Getting Graph for SSL
- **PG-Learn: Parallel Graph Learning for SSL**
  - Gradient-based Graph Learning for SSL
  - Adaptive Parallel Search
- **Empirical Evaluation**
  - Datasets & Baselines
  - Result

Task-driven

Scalable

No hyperparameter to tune

Effective

Datasets

Name	#pts n	#dim d	#cls c	description
COIL	1500	241	6	objects with various shapes
USPS	1000	256	10	handwritten digits
MNIST	1000	784	10	handwritten digits
UMIST	575	644	20	faces (diff. race/gender/etc.)
YALE	320	1024	5	faces (diff. illuminations)

Baselines

strawmen

(1) *Grid* search (GS): k-NN graph with RBF kernel where k and bandwidth σ are chosen via grid search,

(2) $Rand_d$ search (RS): k-NN with RBF kernel where k and different bandwidths $\boldsymbol{a}_{1:d}$ are randomly chosen,

gradient-based

(3) *MinEnt*: Minimum Entropy based tuning of $\boldsymbol{a}_{1:d}$'s as proposed by Zhu et al. [30] (generalized to multi-class),

self-representation

(4) *AEW*: Adaptive Edge Weighting by Karasuyama et al. [14] that estimates $\boldsymbol{a}_{1:d}$'s through local linear reconstruction, and

metric learning

(5) *IDML*: Iterative self-learning scheme combined with distance metric learning by Dhillon et al. [8].

Single-thread Results

10% labeled data, avg'ed across 10 random samples

Dataset	PG-LRN	*MinEnt*	*IDML*	*AEW*	*Grid*	*$Rand_d$*
COIL	**0.9232**	0.9116▲	0.7508▲	0.9100▲	0.8929▲	0.8764▲
USPS	**0.9066**	**0.9088**	0.8565▲	0.8951▲	0.8732▲	0.8169▲
MNIST	**0.8241**	**0.8163**	0.7801△	0.7828▲	0.7550▲	0.7324▲
UMIST	**0.9321**	0.8954▲	0.8973△	0.8975▲	0.8859▲	0.8704▲
YALE	**0.8234**	0.7648△	0.7331▲	0.7386▲	0.6576▲	0.6797▲

Symbols ▲ (p<0.005) and △ (p<0.01)
w.r.t. the paired Wilcoxon signed rank test.

Single-thread Results

Increasing labeling % , results averaged across all datasets

Labeled	PG-L	*MinEnt*	*IDML*	*AEW*	*Grid*	*Rand*$_d$
10% acc.	**0.8819**	0.8594▲	0.8036▲	0.8448▲	0.8129▲	0.7952▲
rank	**1.20**	2.20	4.40	2.80	4.80	5.60
20% acc.	**0.8900**	0.8504▲	0.8118▲	0.8462▲	0.8099▲	0.8088▲
rank	**1.42**	2.83	4.17	2.92	4.83	4.83
30% acc.	**0.9085**	0.8636▲	0.8551▲	0.8613▲	0.8454▲	0.8386▲
rank	**1.33**	3.67	3.83	3.17	4.00	5.00
40% acc.	**0.9153**	0.8617▲	0.8323▲	0.8552▲	0.8381▲	0.8303▲
rank	**1.67**	3.67	3.50	3.67	4.00	4.50
50% acc.	**0.9251**	0.8700△	0.8647▲	0.8635▲	0.8556▲	0.8459▲
rank	**1.50**	3.17	3.83	3.67	4.00	4.83

Symbols ▲ (*p*<0.005) and △ (*p*<0.01)
w.r.t. the paired Wilcoxon signed rank test.

Parallel results with Noisy Features

- **Double the feature space by adding 100% new columns with Normal(0,1) noise**

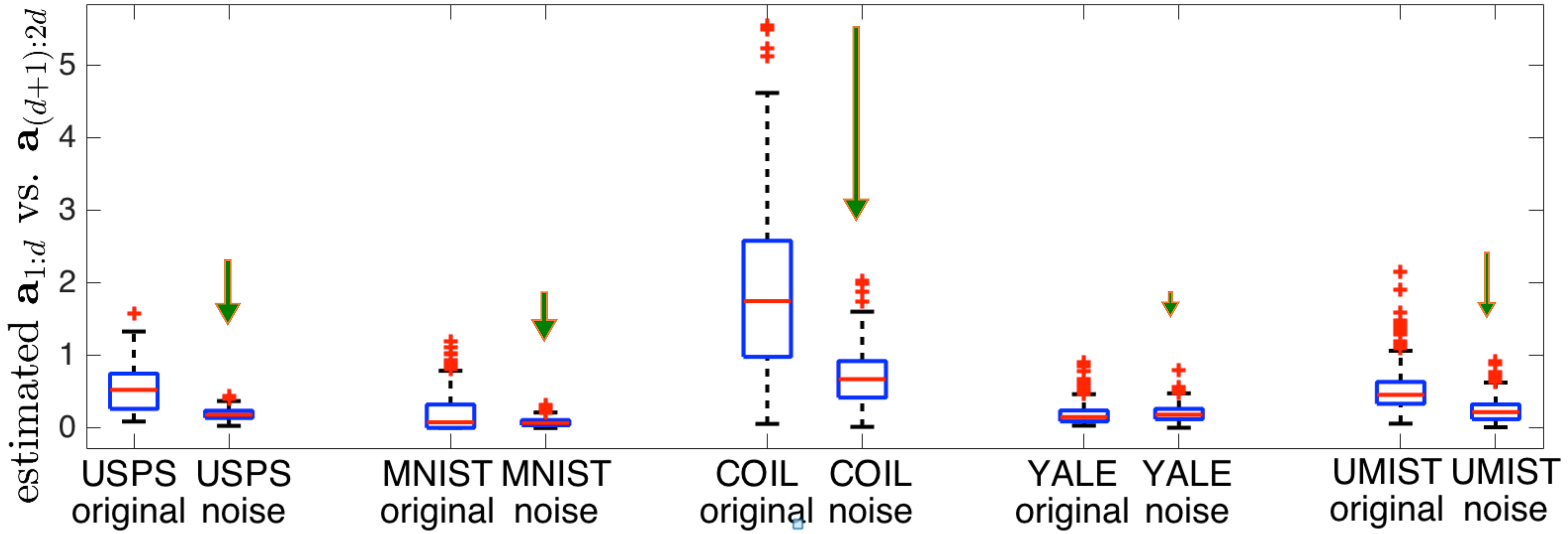
Dataset	PG-LRN	***MinEnt***	***Grid***	***Rand***$_d$
COIL	**0.9044**	0.8197▲	0.6311▲	0.6954▲
USPS	**0.9154**	0.8779△	0.8746▲	0.7619▲
MNIST	**0.8634**	0.8006▲	0.7932▲	0.6668▲
UMIST	**0.8789**	0.7756▲	0.7124▲	0.6405▲
YALE	**0.6859**	0.5671▲	0.5925▲	0.5298▲

- ➢ IDML failed to learn metric due to degeneracy
- ➢ AEW authors' implementation threw out-of-memory errors

Parallel results with Noisy Features

investigating learned feature weights

Effective



➢ PG-Learn estimates lower weights for **noisy** columns

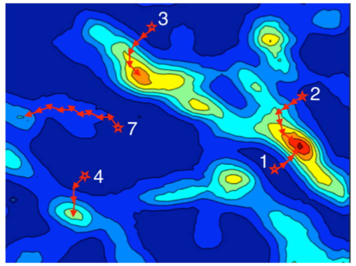
Code, Data, Slides

Task-driven

Scalable

No need to tune

Effective



PG-Learn

https://pg-learn.github.io

Thanks!

Conference attending is funded by travel grant from SIGIR