Metric Learning
(and incidentally some distributed optimization)

Aurélien Bellet

Joint work with A. Habrard and M. Sebban (LaHC St-Etienne), A. Bagheri Garakani, K. Liu, F. Sha and Y. Shi (USC), Y. Liang (Princeton), M.-F. Balcan (CMU)

November 12, 2014

A bit about me

- PhD in Computer Science (Dec 2012)
  - Université Jean Monnet, Saint-Etienne
  - Advisors: Marc Sebban, Amaury Habrard
- Postdoc in 2013–2014 (18 months)
  - University of Southern California, Los Angeles
  - Working with Fei Sha
- Joined Télécom ParisTech on October 15
  - Chaire "Machine Learning for Big Data"
  - Working with Stéphan

Outline of the talk

1. Metric learning
   - Problem statement
   - Some contributions

2. Interlude: the Frank-Wolfe algorithm

3. Leveraging Frank-Wolfe in large-scale learning
   - Similarity learning for sparse high-dimensional data
   - Distributed and communication-efficient sparse learning

Metric learning

Motivation

- Distance and similarity functions are everywhere in machine learning and data mining
  - Nearest neighbors, clustering, kernel methods, ranking, dimensionality reduction, visualization...

- How to define an appropriate similarity score?
  - Crucial to performance of above algorithms
  - Obviously, problem-dependent
  - Let's learn it from data!

Metric learning

Basic recipe

1. Pick a parametric form of distance or similarity function
   - (Generalized) Mahalanobis distance

$$d_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}') = \sqrt{(\boldsymbol{x} - \boldsymbol{x}')^{\mathrm{T}} \boldsymbol{M} (\boldsymbol{x} - \boldsymbol{x}')} \text{ with } \boldsymbol{M} \text{ symmetric PSD}$$

   - Bilinear similarity

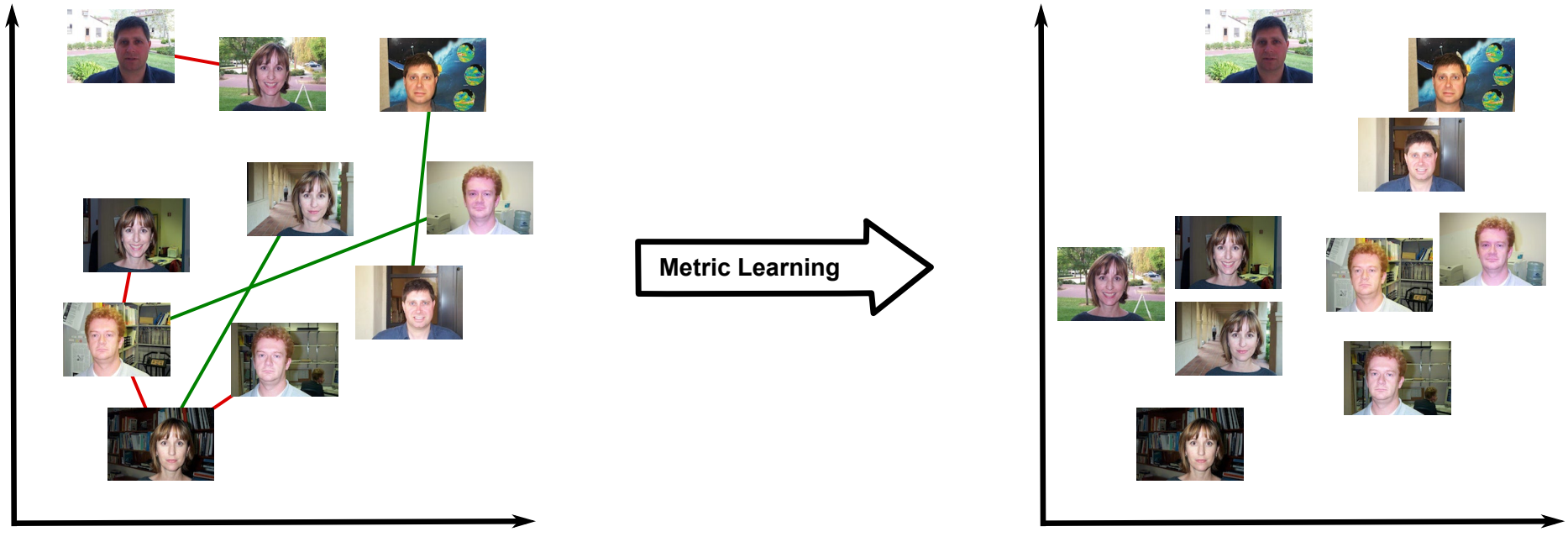
$$S_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^{\mathrm{T}} \boldsymbol{M} \boldsymbol{x}' \text{ with } \boldsymbol{M} \in \mathbb{R}^{d \times d}$$

2. Collect similarity judgements on data pairs/triplets
   - $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are similar (or dissimilar)
   - $\boldsymbol{x}_i$ is more similar to $\boldsymbol{x}_j$ than to $\boldsymbol{x}_k$

3. Estimate parameters such that metric best satisfies them
   - Convex optimization and the like

Metric learning

Illustration



Metric learning

A statistical view (pairwise case)

- Training data $\{z_i = (x_i, y_i)\}_{i=1}^n$ drawn from an unknown distribution μ over $\mathcal{X} \times \mathcal{Y}$ ($\mathcal{Y}$ is the label set)
- Minimize the empirical risk

$$\hat{R}(\boldsymbol{M}) = \frac{2}{n(n-1)} \sum_{1 \le i < j \le n} \mathbb{I}\left[r(y_i, y_j)(d_{\boldsymbol{M}}(x_i, x_j) - 1) > 0\right]$$

  where $r(y, y') = 1$ if $y = y'$ and -1 otherwise

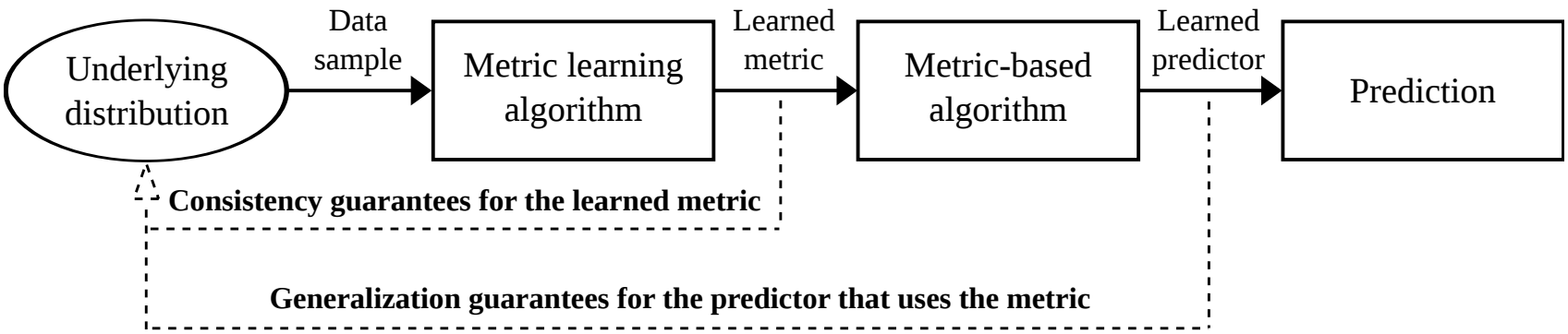
- Hope to achieve small expected risk

$$R(\boldsymbol{M}) = \mathop{\mathbb{E}}_{z, z' \sim \mu} \left[\mathbb{I}\left[r(y, y')(d_{\boldsymbol{M}}(x, x') - 1) > 0\right]\right]$$

- (In practice: convex loss function and regularization on $\boldsymbol{M}$)

Metric learning

Some contributions – Similarity learning for linear classification



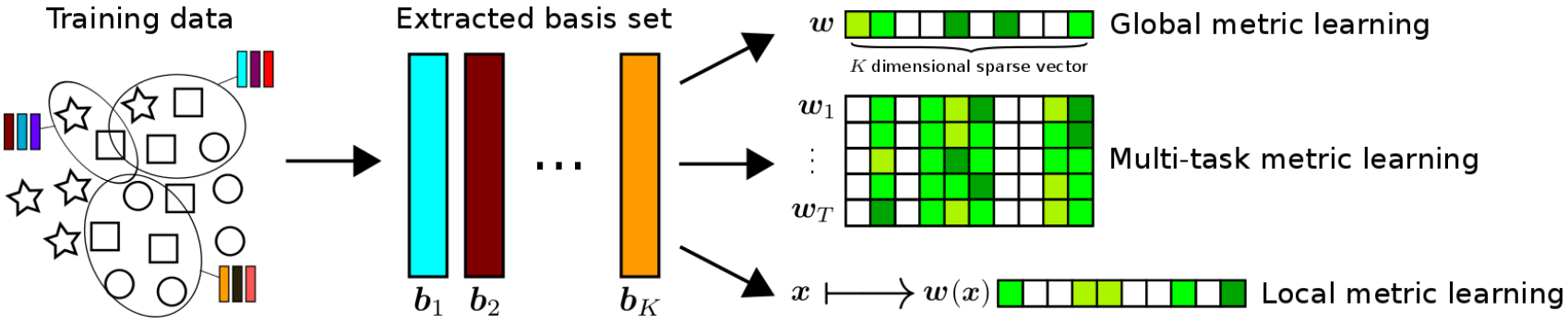
- Learn a similarity function S for a linear classifier h

$$h(x) = \text{sign}\left[\mathop{\mathbb{E}}_{z' \sim \mu} y' w(x') S(x, x')\right]$$

- Build upon theory of learning with good similarity functions [Balcan and Blum, 2006, Balcan et al., 2008]
- Bounds on the expected risk of both the metric and the classifier (using algorithmic stability)
- Efficient algorithms for data represented as numerical vectors but also structured objects (strings, trees)

Metric learning

Some contributions – Learning multiple metrics



- Fix a dictionary of bases $\{\boldsymbol{b_i}\}_{i=1}^K$ and assume a decomposition $\boldsymbol{M} = \sum_{i=1}^K w_i \boldsymbol{b_i} \boldsymbol{b_i}^\mathrm{T}$ with $w_i \geq 0$
- Learn K parameters instead of d^2 – use sparsity for selection
- Convenient to learn multiple metrics:
  - Multi-task setting: learn a metric per task with group sparsity to share information
  - Instance-specific metrics: learn a transformation $\boldsymbol{x} \mapsto w(\boldsymbol{x})$
- Efficient stochastic optimization algorithms and generalization bounds based on algorithmic robustness

Interlude: the Frank Wolfe algorithm (a.k.a. conditional gradient)

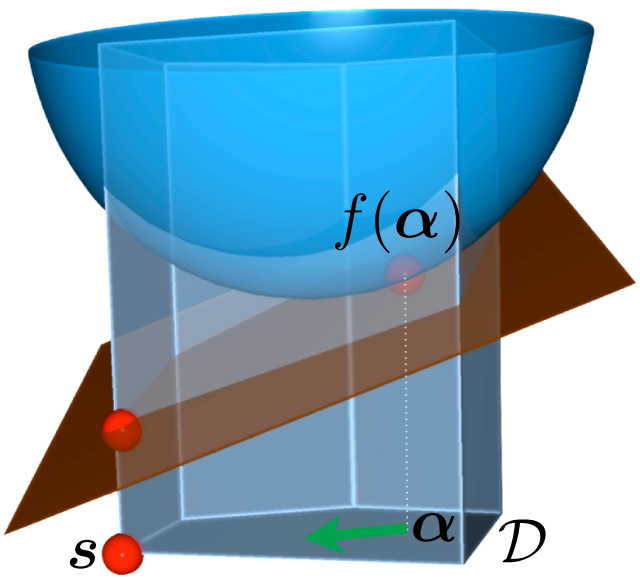
The Frank-Wolfe algorithm

Setup and algorithm

Convex minimization over feasible domain $\mathcal{D}$

$$\min_{\boldsymbol{\alpha} \in \mathcal{D}} \quad f(\boldsymbol{\alpha})$$

- f convex and smooth (with L-Lipschitz gradient w.r.t. $\|\cdot\|$)
- $\mathcal{D}$ convex and compact



Let $\boldsymbol{\alpha}^{(0)} \in \mathcal{D}$

for $k = 0, 1, \ldots$ **do**

$\boldsymbol{s}^{(k)} = \arg\min_{\boldsymbol{s} \in \mathcal{D}} \left\langle \boldsymbol{s}, \nabla f(\boldsymbol{\alpha}^{(k)}) \right\rangle$

$\boldsymbol{\alpha}^{(k+1)} = (1 - \gamma)\boldsymbol{\alpha}^{(k)} + \gamma \boldsymbol{s}^{(k)}$

end for

Figure adapted from [Jaggi, 2013]

The Frank-Wolfe algorithm

Convergence and interesting properties

Convergence [Frank and Wolfe, 1956, Clarkson, 2010, Jaggi, 2013]

At any $k \geq 1$, $\boldsymbol{\alpha}^{(k)}$ is feasible and satisfies

$$f(\boldsymbol{\alpha}^{(k)}) - f(\boldsymbol{\alpha}^*) \leq \frac{2L \operatorname{diam}_{\|\cdot\|}(\mathcal{D})^2}{k+2}$$

- Projection-free algorithm
- Solve a linear problem at each iteration
  - Solution is at a vertex of $\mathcal{D}$
- If $\mathcal{D}$ has special structure, each iteration can be very cheap

The Frank-Wolfe algorithm

Use-case: $\mathcal{D}$ is a convex hull

- When $\mathcal{D} = \text{conv}(\mathcal{A})$, FW is greedy
  - At each iteration, add an element $a \in \mathcal{A}$ to the current iterate
- Example 1: $\mathcal{D}$ is the ℓ_1-norm ball
  - $\mathcal{A} = \{\pm \boldsymbol{e}_i\}_{i=1}^n$
  - Linear problem: find maximum absolute entry of gradient
  - Iterates are sparse: $\boldsymbol{\alpha}^{(0)} = \mathbf{0} \implies \|\boldsymbol{\alpha}^{(k)}\|_0 \leq k$
  - FW finds an ϵ-approximation with $O(1/\epsilon)$ nonzero entries, which is worst-case optimal [Jaggi, 2013]
- Example 2: $\mathcal{D}$ is the trace-norm ball
  - $\mathcal{A} = \{\boldsymbol{u}\boldsymbol{v}^{\mathrm{T}} : \boldsymbol{u} \in \mathbb{R}^n, \|\boldsymbol{u}\|_2 = 1, \boldsymbol{v} \in \mathbb{R}^m, \|\boldsymbol{v}\|_2 = 1\}$
  - Linear problem: find largest singular vector of gradient
  - Iterates are low-rank: $\boldsymbol{M}^{(0)} = \mathbf{0} \implies \text{rank}(\boldsymbol{M}^{(k)}) \leq k$
  - FW finds an ϵ-approximation of rank $O(1/\epsilon)$, which is worst-case optimal [Jaggi, 2011]

Similarity learning
for sparse high-dimensional data

Similarity learning for sparse high-dimensional data

Motivation

- Assume data points are high-dimensional ($d > 10^4$) but D-sparse (on average) with $D \ll d$
  - Bags-of-words (text, image), bioinformatics, etc
- Existing metric learning algorithms fail
  - Intractable: training cost $O(d^2)$ to $O(d^3)$, memory $O(d^2)$
  - Severe overfitting
- Practitioners use dimensionality reduction (PCA, RP)
  - Poor performance in presence of noisy features
  - Resulting metric difficult to interpret for domain experts
- Contributions of this work
  - Learn similarity in original high-dimensional space
  - Time/memory costs independent of d
  - Explicit control of similarity complexity

Similarity learning for sparse high-dimensional data

Basis set

- We want to learn a similarity function $S_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^{\mathrm{T}} \boldsymbol{M} \boldsymbol{x}'$
- Given $\lambda > 0$, for any $i, j \in \{1, \ldots, d\}$, $i \neq j$ we define

$$\boldsymbol{P}_{\lambda}^{(ij)} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \lambda & \cdot & \lambda & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \lambda & \cdot & \lambda & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix} \quad \boldsymbol{N}_{\lambda}^{(ij)} = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \lambda & \cdot & -\lambda & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & -\lambda & \cdot & \lambda & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}$$

$$\mathcal{B}_{\lambda} = \bigcup_{ij} \left\{ \boldsymbol{P}_{\lambda}^{(ij)}, \boldsymbol{N}_{\lambda}^{(ij)} \right\}$$

$$\boldsymbol{M} \in \mathcal{D}_{\lambda} = \mathrm{conv}(\mathcal{B}_{\lambda})$$

- One basis involves only 2 features:

$$S_{\boldsymbol{P}_{\lambda}^{(ij)}}(\boldsymbol{x}, \boldsymbol{x}') = \lambda(x_i x_i' + x_j x_j' + x_i x_j' + x_j x_i')$$

$$S_{\boldsymbol{N}_{\lambda}^{(ij)}}(\boldsymbol{x}, \boldsymbol{x}') = \lambda(x_i x_i' + x_j x_j' - x_i x_j' - x_j x_i')$$

Similarity learning for sparse high-dimensional data

Problem formulation and convergence

- Optimization problem

$$\min_{\boldsymbol{M} \in \mathbb{R}^{d \times d}} \quad f(\boldsymbol{M}) = \frac{1}{T} \sum_{t=1}^{T} \ell\left(\left\langle \boldsymbol{A}^t, \boldsymbol{M} \right\rangle\right) \quad \text{s.t.} \quad \boldsymbol{M} \in \mathcal{D}_\lambda$$

  where $\boldsymbol{A}^t = \boldsymbol{x}_1^t (\boldsymbol{x}_2^t - \boldsymbol{x}_3^t)^{\mathrm{T}}$ and ℓ is the smoothed hinge loss

- Use a FW algorithm to solve it

Convergence

Let $L = \frac{1}{T} \sum_{t=1}^{T} \|\boldsymbol{A}^t\|_F^2$. At any iteration $k \geq 1$, the iterate $\boldsymbol{M}^{(k)} \in \mathcal{D}_\lambda$ of the FW algorithm:

- has at most rank $k+1$ with $4(k+1)$ nonzero entries
- uses at most $2(k+1)$ distinct features
- satisfies $f(\boldsymbol{M}^{(k)}) - f(\boldsymbol{M}^*) \leq 16 L \lambda^2 / (k+2)$

Similarity learning for sparse high-dimensional data

Complexity analysis

- FW can be implemented efficiently on this problem
- An optimal basis can be found in $O(TD^2)$ time and memory even though there are $O(d^2)$ different bases
- An approximately optimal basis can be found in $O(mD^2)$ with $m \ll T$ using a Monte Carlo approximation of the gradient
  - Or even $O(mD)$ using a heuristic (good results in practice)
- Storing $\boldsymbol{M}^{(k)}$ requires only $O(k)$ memory
  - Or even the entire sequence $\boldsymbol{M}^{(0)}, \ldots, \boldsymbol{M}^{(k)}$ at the same cost

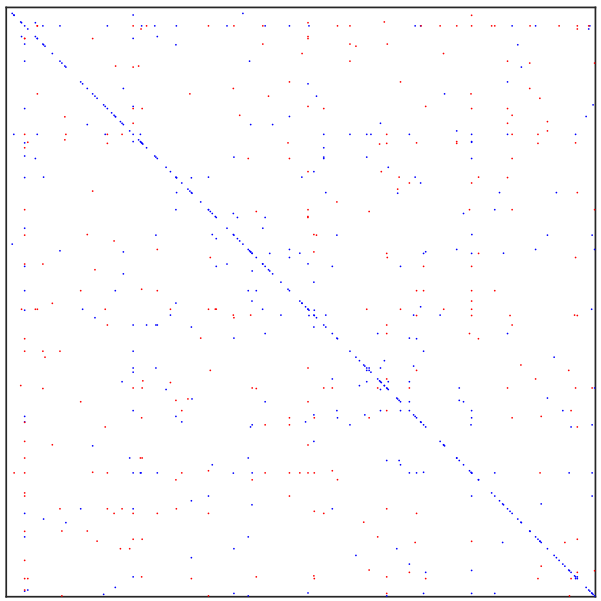
Similarity learning for sparse high-dimensional data

Preliminary experiments

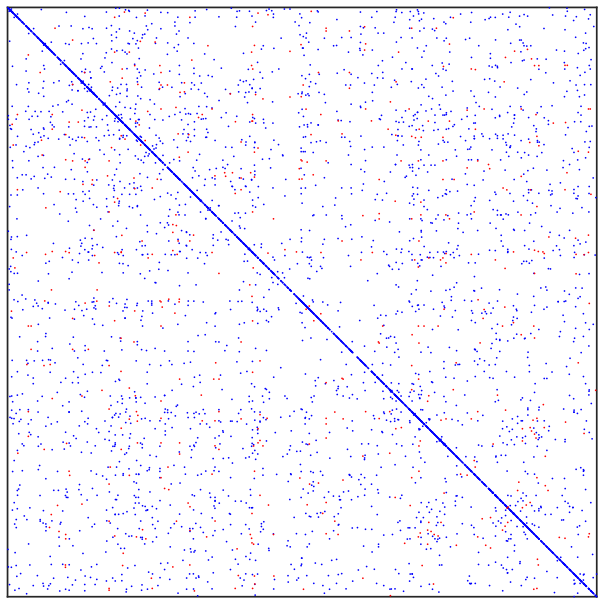
- K-NN test error on datasets with d up to 10^5

Datasets	IDENTITY	RP+OASIS	PCA+OASIS	DIAG-ℓ_2	DIAG-ℓ_1	HDSL
dexter	20.1	24.0	9.3	8.4	8.4	**6.5**
dorothea	9.3	11.4	9.9	6.8	6.6	**6.5**
rcv1_2	6.9	7.0	4.5	3.5	3.7	**3.4**
rcv1_4	11.2	10.6	6.1	6.2	7.2	**5.7**

- Sparsity structure of the matrices



(a) dexter ($20,000 \times 20,000$ matrix, 712 nonzeros)



(b) rcv1_4 ($29,992 \times 29,992$ matrix, 5263 nonzeros)

Similarity learning for sparse high-dimensional data

Ongoing work

- Generalization bounds specific to each iterate
  - Show trade-off between hypothesis complexity (optimization error) and overfitting
  - Validate early stopping strategy

- If we assume the existence of a ground truth sparse metric
  - Can we recover it based on similarity judgements?
  - Preliminary results on synthetic data encouraging
  - Theory?

Distributed and communication-efficient sparse learning

Distributed and communication-efficient sparse learning

Distributed setting

- General setting
  - Data arbitrarily distributed across different sites (*nodes*)
  - Examples: large-scale data, sensor networks, mobile devices
  - Communication between nodes can be a serious bottleneck
- Research questions
  - Theory: study tradeoff between communication complexity and learning/optimization error
  - Practice: derive scalable algorithms, with small communication and synchronization overhead

Distributed and communication-efficient sparse learning

Problem of interest

Problem of interest

Learn sparse combinations of n distributed "atoms":

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \quad f(\boldsymbol{\alpha}) = g(\boldsymbol{A}\boldsymbol{\alpha}) \quad \text{s.t.} \quad \|\boldsymbol{\alpha}\|_1 \leq \beta \qquad (\boldsymbol{A} \in \mathbb{R}^{d \times n})$$

- Atoms are distributed across a set of N nodes $V = \{v_i\}_{i=1}^N$
- Nodes communicate across a network (connected graph)
- Note: domain can be unit simplex Δ_n instead of ℓ_1 ball

$$\Delta_n = \{\alpha \in \mathbb{R}^n : \boldsymbol{\alpha} \geq 0, \sum_i \alpha_i = 1\}$$

Distributed and communication-efficient sparse learning

Applications

- Many applications
  - LASSO with distributed features
  - Kernel SVM with distributed training points
  - Boosting with distributed learners
  - ...

Example: Kernel SVM

- Training set $\{\boldsymbol{z}_i = (\boldsymbol{x}_i, y_i)\}_{i=1}^n$
- Kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \varphi(\boldsymbol{x}), \varphi(\boldsymbol{x}') \rangle$
- Dual problem of L2-SVM:

$$\min_{\boldsymbol{\alpha} \in \Delta_n} \quad \boldsymbol{\alpha}^{\mathrm{T}} \tilde{\boldsymbol{K}} \boldsymbol{\alpha}$$

- $\tilde{\boldsymbol{K}} = [\tilde{k}(\boldsymbol{z}_i, \boldsymbol{z}_j)]_{i,j=1}^n$ with $\tilde{k}(\boldsymbol{z}_i, \boldsymbol{z}_j) = y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) + y_i y_j + \frac{\delta_{ij}}{C}$
- Atoms are $\tilde{\varphi}(\boldsymbol{z}_i) = [y_i \varphi(\boldsymbol{x}_i), y_i, \frac{1}{\sqrt{C}} \boldsymbol{e}_i]$

Distributed and communication-efficient sparse learning

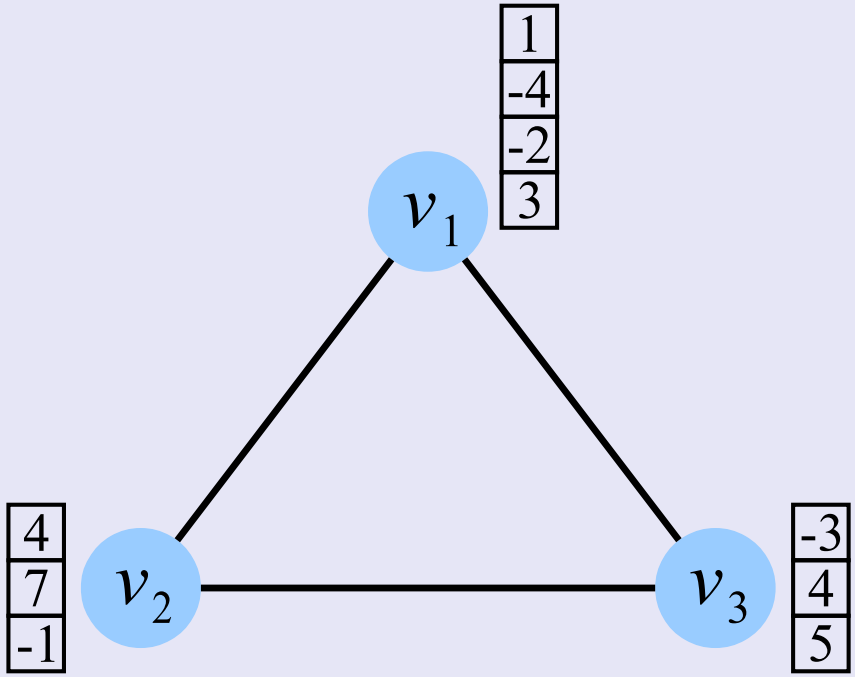
Distributed FW algorithm (dFW)

Recall our problem

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \quad f(\boldsymbol{\alpha}) = g(\boldsymbol{A}\boldsymbol{\alpha}) \quad \text{s.t.} \quad \|\boldsymbol{\alpha}\|_1 \leq \beta \qquad (\boldsymbol{A} \in \mathbb{R}^{d\times n})$$

Algorithm steps

1. Each node computes its local gradient $\boldsymbol{a}_j \in \mathbb{R}^d$



Distributed and communication-efficient sparse learning

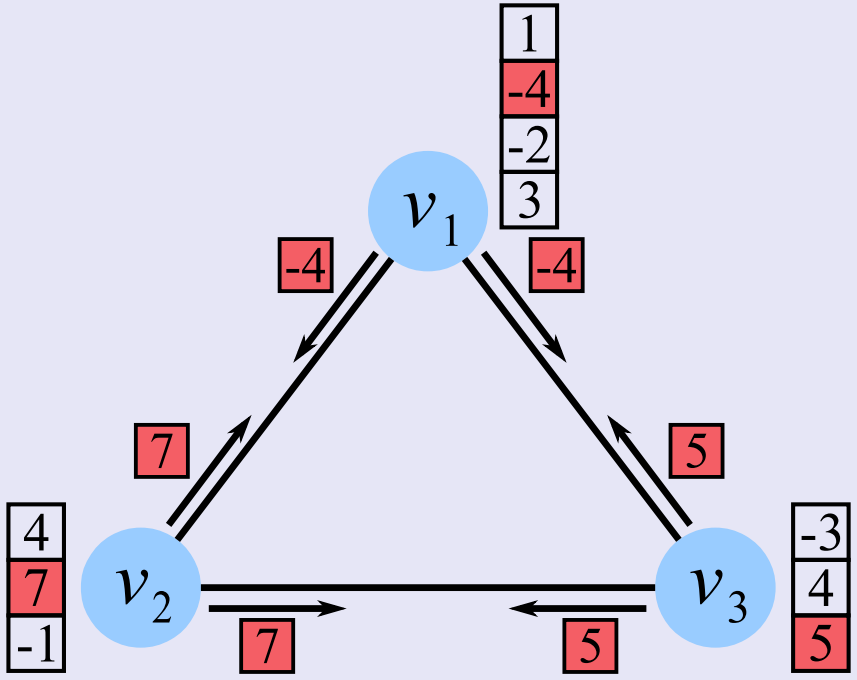
Distributed FW algorithm (dFW)

Recall our problem

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \quad f(\boldsymbol{\alpha}) = g(\boldsymbol{A}\boldsymbol{\alpha}) \quad \text{s.t.} \quad \|\boldsymbol{\alpha}\|_1 \leq \beta \qquad (\boldsymbol{A} \in \mathbb{R}^{d\times n})$$

Algorithm steps

2. Each node broadcast its largest absolute value $\boldsymbol{a}_j \in \mathbb{R}^d$



Distributed and communication-efficient sparse learning

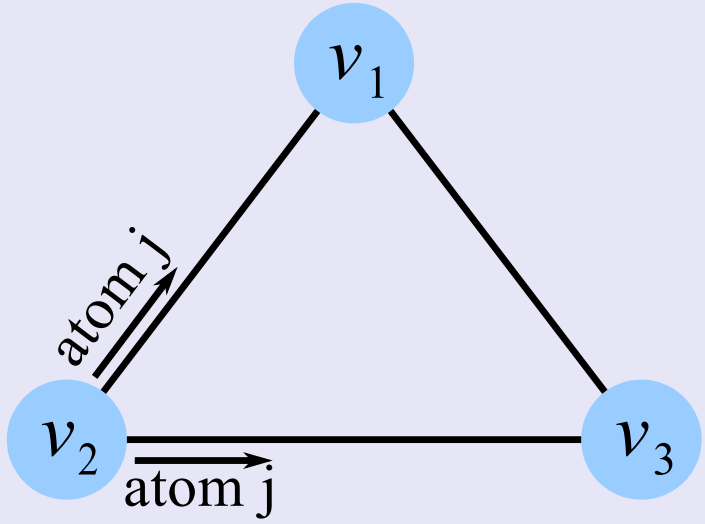
Distributed FW algorithm (dFW)

Recall our problem

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \quad f(\boldsymbol{\alpha}) = g(\boldsymbol{A}\boldsymbol{\alpha}) \quad \text{s.t.} \quad \|\boldsymbol{\alpha}\|_1 \leq \beta \qquad (\boldsymbol{A} \in \mathbb{R}^{d\times n})$$

Algorithm steps

3. Node with global best broadcasts corresponding atom $\boldsymbol{a}_j \in \mathbb{R}^d$



Distributed and communication-efficient sparse learning

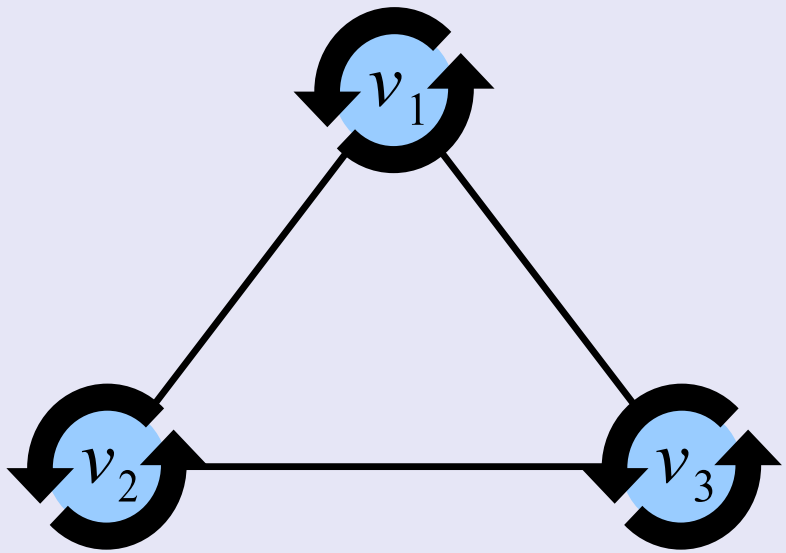
Distributed FW algorithm (dFW)

Recall our problem

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^n} \quad f(\boldsymbol{\alpha}) = g(\boldsymbol{A}\boldsymbol{\alpha}) \quad \text{s.t.} \quad \|\boldsymbol{\alpha}\|_1 \leq \beta \qquad (\boldsymbol{A} \in \mathbb{R}^{d\times n})$$

Algorithm steps

4. All nodes perform a FW update and start over $\boldsymbol{a}_j \in \mathbb{R}^d$



Distributed and communication-efficient sparse learning

Convergence of dFW

- Let B be the cost of broadcasting a real number

Theorem (Convergence of exact dFW)

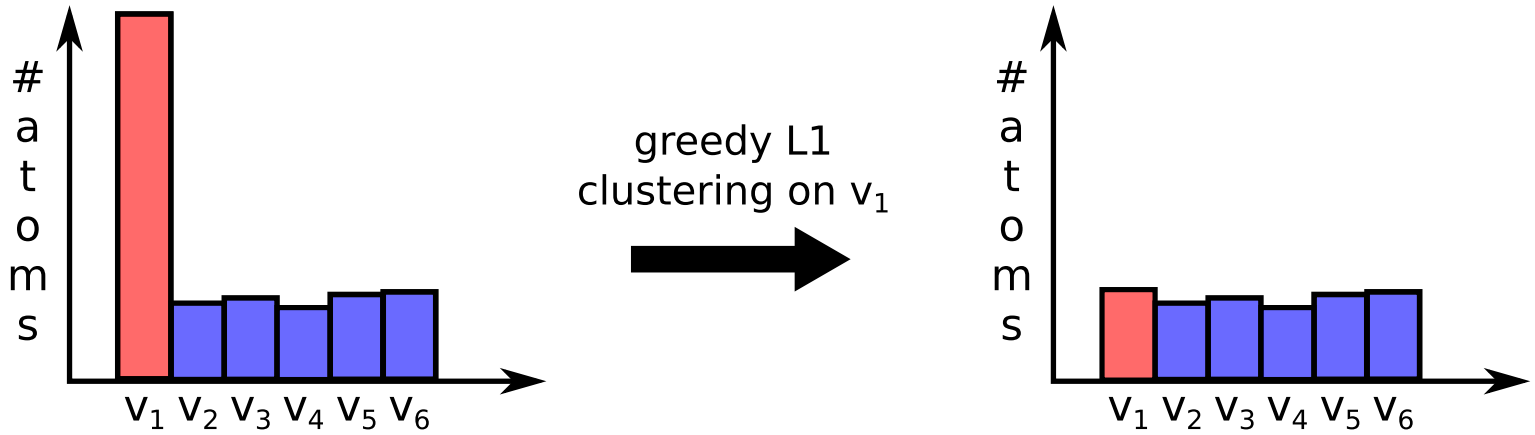
After $O(1/\epsilon)$ rounds and $O\left((Bd + NB)/\epsilon\right)$ total communication, each node holds an ϵ-approximate solution.

- Tradeoff between communication and optimization error
- No dependence on total number of combining elements

Distributed and communication-efficient sparse learning

Approximate variant

- Exact dFW is scalable but requires synchronization
  - Unbalanced local computation $\rightarrow$ significant wait time
- Strategy to balance local costs:
  - Node v_i clusters its n_i atoms into m_i groups
  - We use the greedy m-center algorithm [Gonzalez, 1985]
  - Run dFW on resulting centers
- Use-case examples:
  - Balance number of atoms across nodes
  - Set m_i proportional to computational power of v_i



Distributed and communication-efficient sparse learning

Approximate variant – Analysis

- Define
  - $r^{opt}(\mathcal{A}, m)$ to be the optimal ℓ_1-radius of partitioning atoms in $\mathcal{A}$ into m clusters, and $r^{opt}(\boldsymbol{m}) := \max_i r^{opt}(\mathcal{A}_i, m_i)$
  - $G := \max_{\boldsymbol{\alpha}} \|\nabla g(\boldsymbol{A}\boldsymbol{\alpha})\|_\infty$

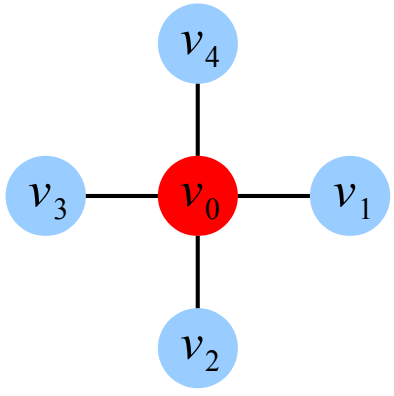
Theorem (Convergence of approximate dFW)

After $O(1/\epsilon)$ iterations, the algorithm returns a solution with optimality gap at most $\epsilon + O(Gr^{opt}(\boldsymbol{m}^0))$. Furthermore, if $r^{opt}(\boldsymbol{m}^{(k)}) = O(1/Gk)$, then the gap is at most ϵ.

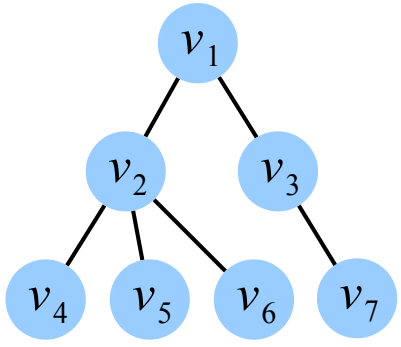
- Additive error depends on cluster tightness
- Can gradually add more centers to make error vanish

Distributed and communication-efficient sparse learning

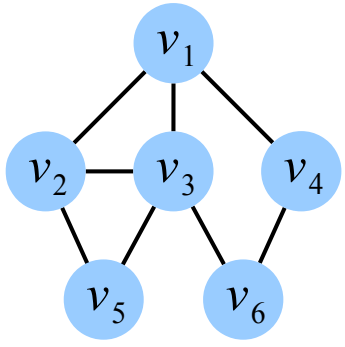
Communication complexity – Dependence on network topology



Star graph



Rooted tree



General connected graph

- Star graph and rooted tree: $O(Nd/\epsilon)$ communication (use network structure to reduce cost)
- General connected graph: $O(M(N+d)/\epsilon)$, where M is the number of edges (use a message-passing strategy)

Distributed and communication-efficient sparse learning

Communication complexity – Lower bound

Theorem (Communication lower bound)

Under mild assumptions, the worst-case communication cost of any deterministic algorithm is $\Omega(d/\epsilon)$.

- Shows that dFW is worst-case optimal in ϵ and d
- Proof outline:
  1. Identify a problem instance for which any ϵ-approximate solution has $O(1/\epsilon)$ atoms
  2. Distribute data across 2 nodes s.t. these atoms are almost evenly split across nodes
  3. Show that for any fixed dataset on one node, there are T different instances on the other node s.t. in any 2 such instances, the sets of selected atoms are different
  4. Any node then needs $O(\log T)$ bits to figure out the selected atoms, and we show that $\log T = \Omega(d/\epsilon)$

Distributed and communication-efficient sparse learning

Experiments

- Objective value achieved for given communication budget
  - Comparison to baselines (not shown)
  - Comparison to distributed ADMM

- Runtime of dFW in realistic distributed setting
  - Exact dFW
  - Benefits of approximate variant
  - Asynchronous updates

Distributed and communication-efficient sparse learning

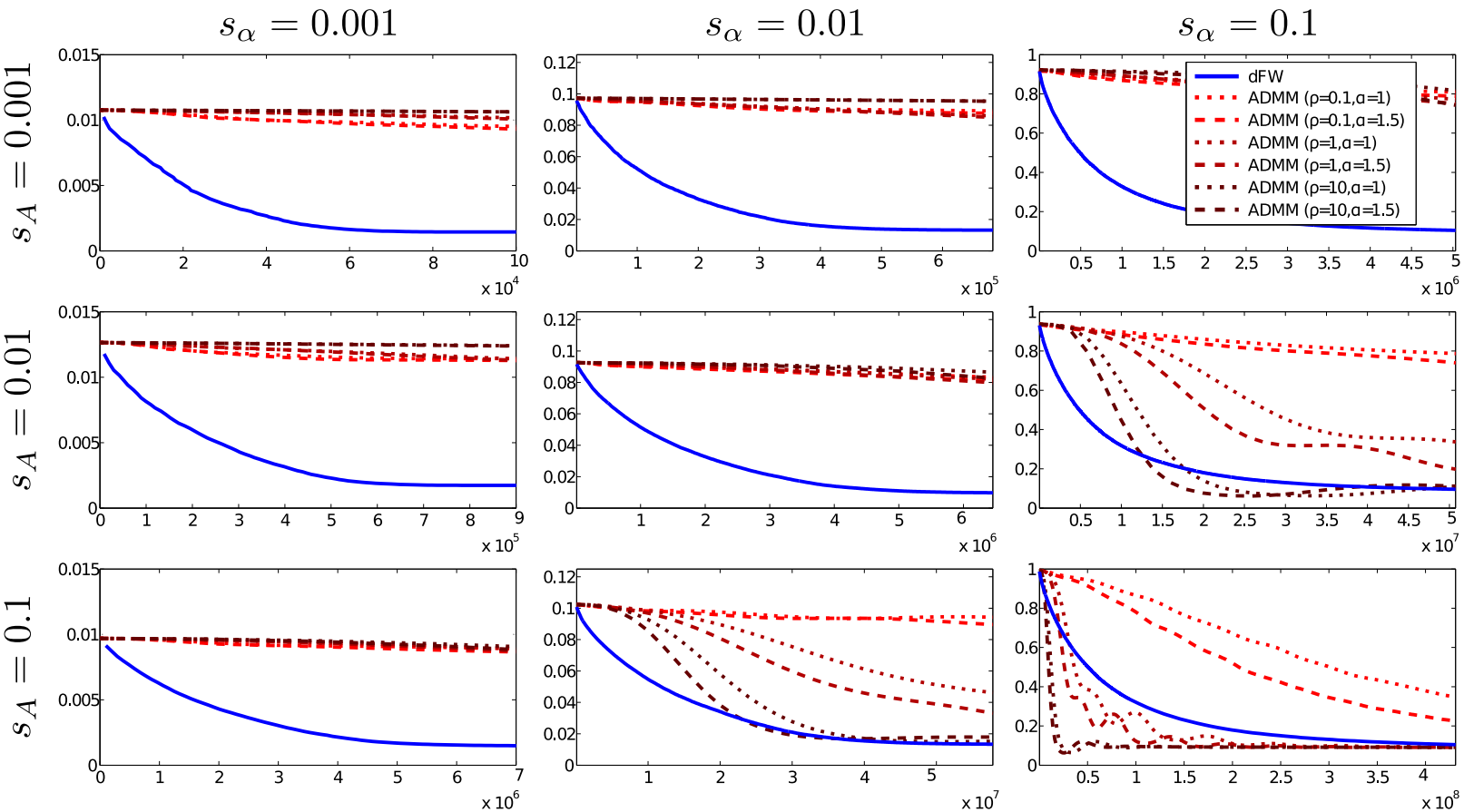
Experiments – Comparison to distributed ADMM

- ADMM [Boyd et al., 2011] is popular to tackle many distributed optimization problems
  - Like dFW, can deal with LASSO with distributed features
  - Parameter vector $\boldsymbol{\alpha}$ partitioned as $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_N]$
  - Communicates partial/global predictions: $\boldsymbol{A}_i \boldsymbol{\alpha}_i$ and $\sum_{i=1}^{N} \boldsymbol{A}_i \boldsymbol{\alpha}_i$
- Experimental setup
  - Synthetic data ($n = 100K$, $d = 10K$) with varying sparsity
  - Atoms distributed across 100 nodes uniformly at random

Distributed and communication-efficient sparse learning

Experiments – Comparison to distributed ADMM

- dFW advantageous for sparse data and/or solution, while ADMM is preferable in the dense setting
- Note: no parameter to tune for dFW



LASSO results (MSE vs communication)

Distributed and communication-efficient sparse learning

Experiments – Realistic distributed environment

- Network specs
  - Fully connected with $N \in \{1, 5, 10, 25, 50\}$ nodes
  - A node is a single 2.4GHz CPU core of a separate host
  - Communication over 56.6-gigabit infrastructure

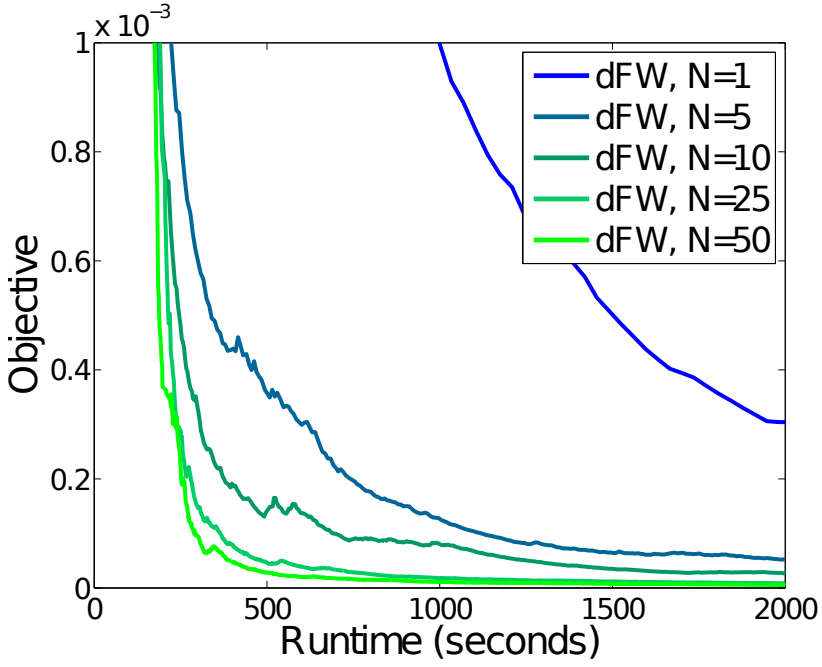
- The task
  - SVM with Gaussian RBF kernel
  - Speech data with 8.7M training examples, 41 classes
  - Implementation of dFW in C++ with openMPI[1]

[1] http://www.open-mpi.org

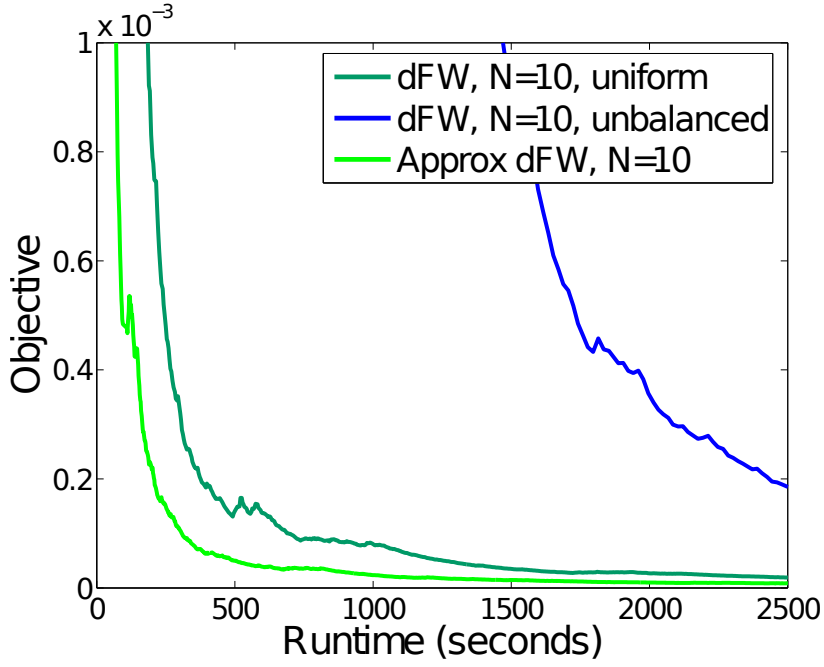
Distributed and communication-efficient sparse learning

Experiments – Realistic distributed environment

- When distribution of atoms is roughly balanced, exact dFW achieves near-linear speedup
- When distribution is unbalanced (e.g., 1 node has 50% of the data), great benefits from approximate variant



(a) Exact dFW on uniform distribution

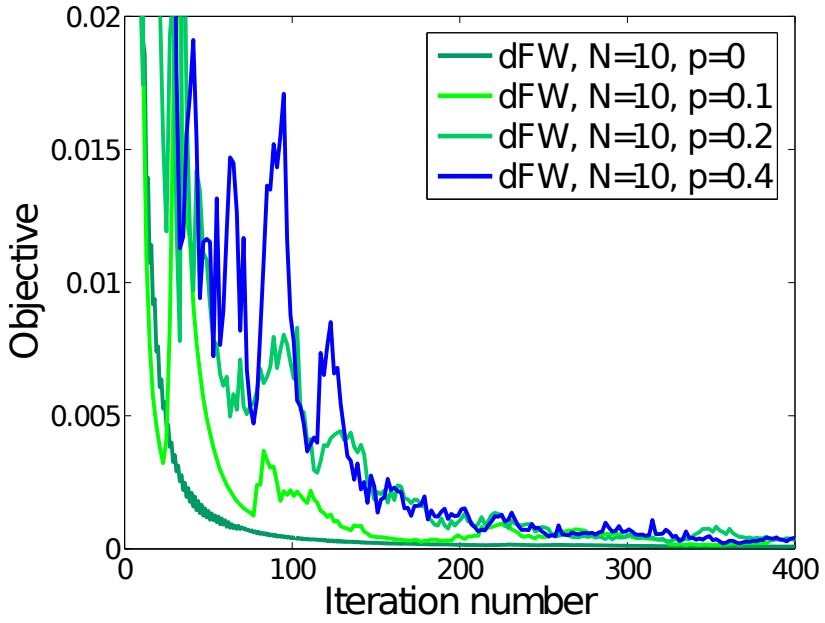


(b) Approximate dFW to balance costs

Distributed and communication-efficient sparse learning

Experiments – Realistic distributed environment

- Another way to reduce synchronization costs is to perform asynchronous updates
- To simulate this, we randomly drop communication messages with probability p
- dFW is fairly robust, even with 40% random drops



dFW under communication errors and asynchrony

Summary and perspectives

- Take-home messages
  - Metric learning is an important topic
  - Frank-Wolfe can be useful to tackle large-scale problems
- Refer to papers for details, proofs and additional experiments
- Future directions
  - Propose and analyze an asynchronous version of dFW
  - Distributed metric learning

References I

[Balcan and Blum, 2006] Balcan, M.-F. and Blum, A. (2006).
On a Theory of Learning with Similarity Functions.
In *ICML*, pages 73–80.

[Balcan et al., 2008] Balcan, M.-F., Blum, A., and Srebro, N. (2008).
Improved Guarantees for Learning via Similarity Functions.
In *COLT*, pages 287–298.

[Boyd et al., 2011] Boyd, S. P., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011).
Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers.
Foundations and Trends in Machine Learning, 3(1):1–122.

[Clarkson, 2010] Clarkson, K. L. (2010).
Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm.
ACM Transactions on Algorithms, 6(4):1–30.

[Frank and Wolfe, 1956] Frank, M. and Wolfe, P. (1956).
An algorithm for quadratic programming.
Naval Research Logistics Quarterly, 3(1-2):95–110.

[Gonzalez, 1985] Gonzalez, T. F. (1985).
Clustering to minimize the maximum intercluster distance.
Theoretical Computer Science, 38:293–306.

References II

[Jaggi, 2011] Jaggi, M. (2011).
Sparse Convex Optimization Methods for Machine Learning.
PhD thesis, ETH Zurich.

[Jaggi, 2013] Jaggi, M. (2013).
Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization.
In *ICML*.