Two Distributed Optimization Algorithms for Machine Learning

Yingyu Liang
Princeton University
Distributed Data

Sensor networks

Distributed databases

Communication: crucial resource
Distributed Data

- $s$ servers connected to a central coordinator
- Local datasets $P_1, P_2, \ldots, P_s$
- Global dataset $P = \bigcup_i P_i$
• Distributed PCA [NIPS’14]
  • Simple algo: build global solution on local solutions
  • Bound the error introduced into downstream applications
  • Joint work with Nina Balcan, Vandana Kanchanapally, David Woodruff

• Distributed Frank-Wolfe [SDM’15]
  • Adapt FW to the distributed setting
  • Turn FW sparsity guarantees into communication efficiency
  • Joint work with Aurelien Bellet, Alireza Garakani, Nina Balcan, Fei Sha
1. Distributed PCA
Principal Component Analysis

• Classic definition (after normalizing data)

Given $P \in \mathbb{R}^{n \times d}$ where each row is a data point in dim $d$, find the best rank $t$ subspace $Y$ by

$$
\min_Y \| P - PYY^\top \|_F^2.
$$

Let $P^{(t)}$ denote $PYY^\top$.

• Usage: summarize/visualize the data
Principal Component Analysis

• More important: pre-process data for downstream applications

• Our goal: compute $P^{(t)}$ s.t. for downstream applications,
  any $\alpha$-approx. on $P^{(t)}$ is $\alpha(1 + \epsilon)$-approx. on $P$
  • Typical downstream application: $k$-means clustering (find $k$ centers to
    minimize the sum of square distances from the points to their nearest
    centers)
  • Our algo with slight modification can solve the classic PCA optimization
Distributed PCA

- $s$ servers connected to a central coordinator
- Global data $P$ partitioned to local datasets $P_1, P_2, \ldots, P_s$
Our results

• Simple algorithm disPCA
  • Guarantee holds for arbitrary data and partition
  • Holds for a wide family of problems: $k$-means, subspace clustering,…
  • Low communication cost

• Speedup version Fast disPCA
  • Easy plug-in of randomized speedup techniques
  • Merely comprise the quality and communication
Review

• Classic PCA solved by SVD on data

\[ P = U \Sigma V^\top \]

\[ V^{(t)}: \text{first } t \text{ column of } V \]

\[ P^{(t)} = U \Sigma^{(t)} (V^{(t)})^\top \]
• Classic PCA solved by SVD (equivalently, factorize covariance)

compute $S = P^\top P$, factorize $S = VDV^\top$

$V^{(t)}$: first $t$ column of $V$

$P^{(t)}$
Distributed PCA

Algorithm [Balcan–Kanchanapally–L.-Woodruff’14]

- Each server: SVD $P_i = U_i \Sigma_i V_i^T$
  send $\Sigma_i^{(t)} V_i^{(t)}$ to Coordinator

- Coordinator: estimate $S$ from $\Sigma_i^{(t)} V_i^{(t)}$
  factorize $S = V \Lambda V^T$ to get $V^{(t)}$
Distributed PCA

Algorithm [Balcan–Kanchanapally–L.–Woodruff ’14]

- Each server: SVD $P_i = U_i \Sigma_i V_i^\top$
  
  $$S = \sum_i (P_i^{(t)})^\top P_i^{(t)} = \sum_i (\Sigma_i V_i^{(t)})^\top \Sigma_i V_i^{(t)}$$

- Coordinator: estimate $S$ from $\sum_i \Sigma_i V_i^{(t)}$
  factorize $S = \Lambda \Lambda^\top$ to get $V^{(t)}$
Factorizing $S$ is equivalent to SVD on $\left[ \Sigma_1^{(t)} V_1^{(t)} ; \ldots ; \Sigma_s^{(t)} V_s^{(t)} \right]$
Distributed PCA

Theorem 1 (Distributed PCA for $k$-means)

If $t = O(k/\epsilon^2)$, then any $\alpha$-approximation solution for the data after distributed PCA is also $(1 + \epsilon)\alpha$-approximation solution for the original data. The communication is $O(skd/\epsilon^2)$.

- Similar guarantee also holds for subspace clustering, and their constraint versions like NNMF, LDA, …
Key: Close Projection

- Project the original and PCA’ed data on any $k$-dim subspace
- If the projections are close, then any approx. for $P(t)$ is also for $P$
Set \( t = O(k/\epsilon^2) \)

Projections are close: proved by plugged in the SVD truncation
Randomized speedup techniques do not harm Close Projection:

- Subspace embedding [Clarkson-Woodruff’13]…
  - Property: $\|HPy\|_2 \approx \|Py\|_2$, $\forall y$; take advantage of sparsity

- Randomized SVD [Halko et al.’10]
  - Probe the row space by random projection
Experiments

• Little lost of clustering quality when dim: 60K → 20

NewsGroups (18K in dim 60K). s=25, k=20.
Experiments

- Speedup: 20x
2. Distributed Frank-Wolfe
Example: Lasso for selecting features

- Training data $A \in \mathbb{R}^{d \times m}, y \in \mathbb{R}^{d}$
- Optimization problem:

$$\min_{\alpha \in \mathbb{R}^{m}} \| y - A\alpha \|_2^2 \quad \text{s.t.} \quad \| \alpha \|_1 \leq \beta$$

- Non-zero entries in $\alpha$: select the corresponding features
- $\ell_1$ regularization: reduce \# features selected
Introduction

- $s$ servers connected to a central coordinator
- Features distributed over the servers
Introduction

Problem of interest

Learn sparse combinations of \( m \) distributed “atoms”:

\[
\min_{\alpha \in \mathbb{R}^m} f(\alpha) = g(A\alpha) \quad \text{s.t.} \quad \|\alpha\|_1 \leq \beta \quad (A \in \mathbb{R}^{d \times m})
\]

• Atoms are columns of \( A \), distributed over the servers
• More applications: Kernel SVM, distributed boosting,…
\[
\min_{\alpha \in \mathcal{D}} f(\alpha): \text{convex } \mathcal{D}, f; \text{ continuously differentiable } f
\]

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

for \( k = 0, 1, \ldots \) do

\[
\begin{align*}
\mathbf{s}^{(k)} &= \arg \min_{\mathbf{s} \in \mathcal{D}} \langle \mathbf{s}, \nabla f(\alpha^{(k)}) \rangle \\
\alpha^{(k+1)} &= (1 - \gamma)\alpha^{(k)} + \gamma \mathbf{s}^{(k)}
\end{align*}
\]

end for

Figure adapted from [Jaggi, 2013]

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

After \( O(1/\varepsilon) \) iterations, FW returns \( \alpha \) s.t. \( f(\alpha) - f(\alpha^*) \leq \varepsilon \).
FW Algorithm

- On $l_1$ ball, solution to the subproblem lies at vertex (basis vector)
  - FW is greedy: $k$ non-zero entries at step $k$
  - FW is efficient: simply find max absolute entry of gradient

- Similar derivation for simplex constraint
Distributed FW

\[
\min_{\alpha \in \mathbb{R}^m} \quad f(\alpha) = g(A\alpha) \quad \text{s.t.} \quad \|\alpha\|_1 \leq \beta \quad (A \in \mathbb{R}^{d \times m})
\]

Distributed Frank-Wolfe (dFW)

1. Each server computes its local gradient
2. Each server broadcasts its largest absolute value
3. Server with global best broadcasts corresponding atom \(a_j\)
4. All servers perform a FW update and start over
Distributed FW

Theorem 1 (Convergence of exact dFW)

After $O(1/\epsilon)$ rounds and $O(sd/\epsilon)$ total communication, each server holds an $\epsilon$-approximate solution.

- Tradeoff between communication and error
- No dependence on the total number of atoms
Approximate Variant

- Exact dFW is scalable but requires synchronization
  - Unbalanced local computation leads to significant wait time

- Strategy to balance local costs:
  - Server $i$ clusters its atoms into $k_i$ groups
  - We use the greedy $k$-center algorithm [Gonzalez, 1985]
  - Run dFW on resulting centers

**Greedy $k$-center($A, C, k$)**

Repeat $k$ times:

$$j' = \arg \max_{j \in A} d(a_j, C) \text{ with } d(a_j, C) = \min_{l \in C} \|a_j - a_l\|_1;$$

$$C = C \cup \{a_{j'}\};$$
Approximate Variant

- Use-case examples:
  - Balance number of atoms across nodes
  - Set $k_i$ proportional to computational power of server $i$
Approximate Variant

Theorem 2 (Convergence of approximate dFW)

- \( r^{\text{opt}}(A, k) \) to be the optimal \( \ell_1 \)-radius of partitioning atoms \( A \) into \( k \) clusters, and \( r^{\text{opt}}(k) := \max_i r^{\text{opt}}(A_i, k_i) \)
- \( L := \max_\alpha \| \nabla g(A\alpha) \|_\infty \)

After \( O(1/\epsilon) \) iterations, the algorithm returns a solution with optimality gap at most \( \epsilon + O(Lr^{\text{opt}}(k)) \).

- Can gradually add more centers to make extra error vanish
**Lower Bound on Communication**

**Theorem 3 (Communication lower bound)**

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

- Identify a problem for which any $\epsilon$-approx. has $\Omega(1/\epsilon)$ atoms
- Distribute data on 2 servers s.t. these atoms are almost evenly split
- For any fixed dataset on one server, there are $T$ different instances on the other s.t. in any 2 such instances, the sets of selected atoms are different
- Any server needs $\log(T) = \Omega(d/\epsilon)$ bits to find the selected atoms
Experiments: baseline

• Two baselines
  • Random: each server picks a fixed set of atoms at random
  • Local FW [Lodi et al., 2010]: each server runs FW locally to select a fixed set of atoms

• Selected atoms are sent to the coordinator which solves the problem using only these atoms

• Experiment setup
  • SVM with RBF kernel on Adult dataset (32K in dim 123)
  • LASSO on Dorothea dataset (100K in dim 1.15K)
  • Atoms distributed across $N$ servers uniformly at random
Experiments: baseline

(a) Kernel SVM results

(b) LASSO results
Experiments: ADMM

• Compare to ADMM [Boyd et al., 2011], a popular algorithm to tackle many distributed optimization

• Experiment setup
  • Synthetic data (100K in dim 10K) with varying sparsity
  • Atoms distributed across 100 servers uniformly at random
Experiments: ADMM

LASSO results (MSE vs communication)
Thanks! Q&A