Greedy Column Subset Selection: New Bounds and Distributed Algorithms

Jason Altschuler, Aditya Bhaskara, Gang (Thomas) Fu, Vahab Mirrokni, Afshin Rostamizadeh, and Morteza Zadimoghaddam

Google

ICML 2016

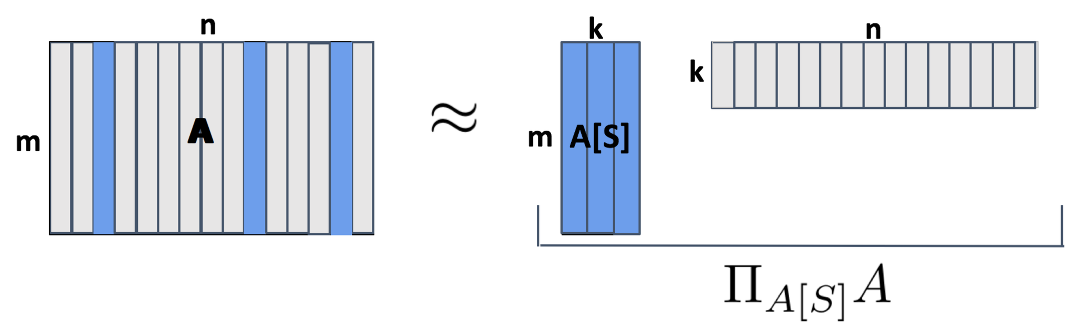
Motivation for Column Subset Selection (CSS)

- Low-rank approximation is useful in many applications, e.g. dimensionality reduction, signal denoising, compression, etc.

$$\underset{X,\ \mathrm{rank}(X)=k}{\arg\min} \ \|A - X\|_F^2$$

- But matrix columns often have inherent meaning (e.g. instances by features matrix), and unconstrained low-rank approximation does not respect this.
- This motivates CSS, which is low-rank approximation in the column space of *A*:

$$\underset{S\subset[n],\ |S|=k}{\arg\min} \ \|A - \Pi_{A[S]}A\|_F^2$$



CSS for Dimensionality Reduction

- Common application of CSS: feature selection on instances by features data matrix
  - Unsupervised: doesn't need labeled data
  - Classifier independent: can reuse output for different classifiers
  - Interpretable: generate features by subselecting instead of arbitrary function
  - Efficient during inference: feature subselection instead of matrix multiplication (SVD). CSS good if latency sensitive, projection matrix prohibitively large, or sparse data
- Has been considered in many previous works: Drineas et al 2004, Frieze et al 2004, Deshpande et al 2006, Drineas et al 2008, Boutsidis et al 2009, Farahat et al 2011, Civril et al 2012, Guruswami et al 2012, Cohen et al 2015, Farahat et al 2015, Boutsidis et al 2016, to name just a *few*....

Generalized CSS (GCSS)

- GCSS($A \in \mathbb{R}^{m\times n_A}$, $B \in \mathbb{R}^{m\times n_B}$, k) seeks *k* columns of *B* to explain *A*:

$$\underset{S\subset[n_B],\ |S|=k}{\arg\min} \ \|A - \Pi_{B[S]}A\|_F^2$$

- Note this is equivalent to:

$$\underbrace{\underset{S\subset[n_B],\ |S|=k}{\arg\max} \ \|\Pi_{B[S]}A\|_F^2}_{\text{denote by } f(S)}$$

- GCSS ⟷ maximizing *f* subject to cardinality constraint
- Intuition: *f(S)* measures how much the selected columns *S* "explain/cover" *A*

References

- Boutsidis et al., *An improved approximation algorithm for the column subset selection problem*. SODA 2009.
- Boutsidis et al., *Communication-optimal distributed principal component analysis in the column-partition model.* STOC 2016.
- Civril et al., *Column Subset Selection via sparse approximation of SVD*. Theor. Comput. Sci. 2012.
- Cohen et al., *Dimensionality reduction for k-means clustering and low-rank approximation*. STOC 2015.
- Deshpande et al., *Efficient volume sampling for row/column subset selection*. FOCS 2010.
- Drineas et al., *Clustering large graphs via the singular value decomposition*. Mach. Learn. 2004.
- Drineas et al., *Relative-error CUR matrix decompositions*. SIAM J. Matrix Analysis Applications 2008.
- Farahat et al., *A fast greedy algorithm for generalized column subset selection*. NIPS 2013.
- Farahat et al., *Greedy column subset selection for large-scale data sets*. Know. Inf. Syst. 2015.
- Guruswami et al., *Optimal column-based low-rank matrix reconstruction*. SODA 2012.
- Johnson et al., *Extensions of Lipschitz mappings into a Hilbert space*. Modern Anal. & Prob.1982.
- Mirzasoleiman et al., *Lazier than lazy greedy*. AAAI 2015.
- Sarlos, *Improved approximation algorithms for large matrices via random projections*. FOCS 2006.

(Single-machine) Greedy Algorithm

$S \leftarrow \varnothing$
for $i = 1 : k$
  Pick column B_j that maximizes $f(S \cup \{B_j\})$
  $S \leftarrow S \cup \{B_j\}$
Return S

Our Contributions

- We prove a tight approximation guarantee for the greedy algorithm.
- We give the first distributed implementation with provable approximation factors.
- We present further optimizations for the greedy algorithm.
- We find encouraging preliminary empirical results showing these algorithms have accuracy comparable with the state-of-the-art and are extremely scalable.

Approximation Guarantee for GREEDY

Theorem: Consider GCSS(A, B, k) with accuracy parameter $\varepsilon > 0$. Let OPT_k be an optimal set of columns from B. If $r = O\left(\frac{k}{\varepsilon\,\sigma_{\min}(\mathrm{OPT}_k)}\right)$

$$f(\mathrm{GREEDY}_r) \geq (1-\varepsilon)\, f(\mathrm{OPT}_k)$$

And this is tight up to a constant factor.

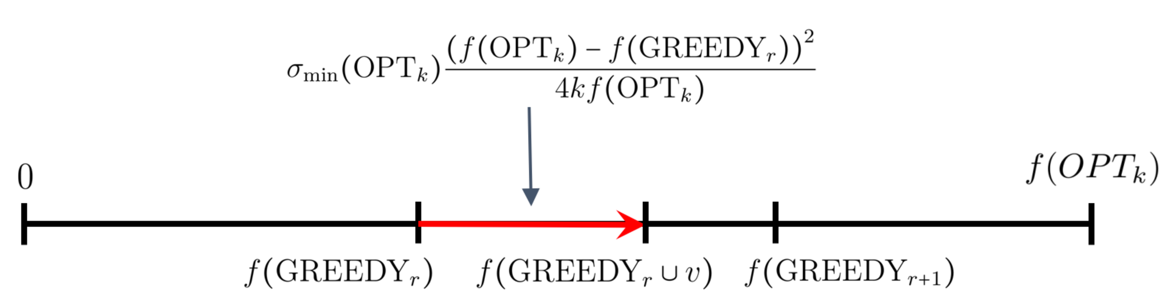
- We expect OPT_k to be well-conditioned ⟶ $\frac{1}{\sigma_{\min}(\mathrm{OPT}_k)}$ small
- Significant improvement upon current bounds, which depend on *worst* singular value of *any* k columns

Proof Sketch

- **Key lemma:** If $f(\mathrm{GREEDY}_r) \leq f(OPT_k)$, there exists $v \in \mathrm{OPT}_k$ s.t.

$$f(\mathrm{GREEDY}_r \cup v) - f(\mathrm{GREEDY}_r) \geq \sigma_{\min}(\mathrm{OPT}_k)\frac{(f(\mathrm{OPT}_k) - f(\mathrm{GREEDY}_r))^2}{4k f(\mathrm{OPT}_k)}$$

- In words: exists column in OPT_k giving large marginal gain to GREEDY_r
- Observe, by definition, $f(\mathrm{GREEDY}_{r+1}) \geq f(\mathrm{GREEDY}_r \cup v)$
- Thus each iteration of GREEDY substantially closes gap to $f(\mathrm{OPT}_k)$



- After $r = O\left(\frac{k}{\varepsilon\,\sigma_{\min}(\mathrm{OPT}_k)}\right)$ such iterations, gets within $1-\varepsilon$ of $f(\mathrm{OPT}_k)$

Further Optimizations for GREEDY

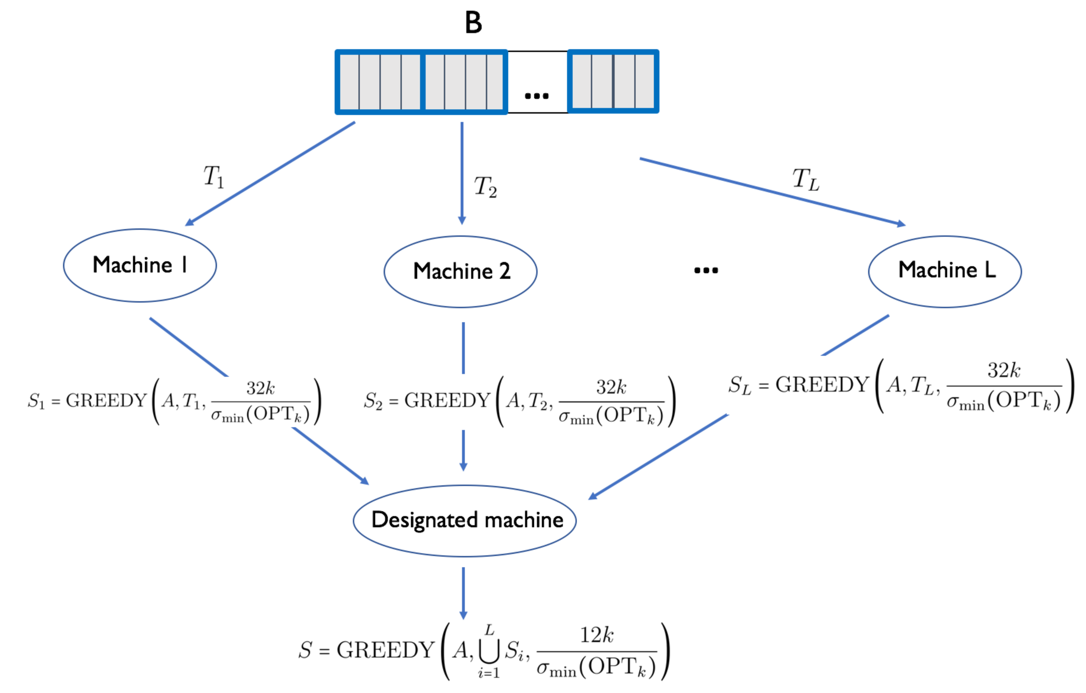
- Bottleneck is computing (marginal) gain of candidate column
- Naive implementation of GREEDY has complexity:

$$O(\underbrace{k}_{\text{iterations}} \cdot (\underbrace{n_B}_{\text{marginal utility calls}} \cdot \underbrace{(kmn_A)}_{\text{complexity of marginal utility}})) = O(k^2 m n_A n_B)$$

- GREEDY++ has 4 optimizations that preserve our $1-\varepsilon$ approximation:
  1. JL Lemma [Johnson & Lindenstrauss 1982, Carlos 2006]. Randomly project to $m' \approx \frac{k\log(\max(n_A, n_B))}{\varepsilon^2}$ rows.
  2. Projection-Cost Preserving Sketches [Cohen et al 2015]. Sketch A with $n'_A \approx \frac{k}{\varepsilon^2}$ columns.
  3. Stochastic Greedy [Mirzasoleiman et al 2015]. Each iteration only makes $\frac{n_B}{k}\log\frac{1}{\varepsilon}$ calls to marginal utility.
  4. Updating *A* and *B* every iteration [Farahat et al 2013]. After each iteration, remove projections of *A* and *B* on selected column.
- GREEDY++ has complexity: $O\left(kn_B\tilde{n}\log\tilde{n}\frac{\log\frac{1}{\varepsilon}}{\varepsilon^2}\right)$ where $\tilde{n} = \max\left(\frac{k}{\varepsilon^2}, n_B\right)$

(Distributed) Greedy Coreset Algorithm

- GCSS(A, B, k) with L machines



- Easy to implement in MapReduce
- Nuance: need to randomly partition B in first step (can construct arbitrarily bad instances if don't randomize)
  - Gives 2-pass streaming algorithm in random arrival model for columns
- Can do multiple rounds for massive datasets and better approximations

Approximation Guarantee

Theorem [1-round result]: GREEDY-CORE gives objective value of $\Omega\left(\frac{\sigma_{\min}(\mathrm{OPT}_k)}{\sigma_{\max}(\mathrm{OPT}_k)} \cdot f(\mathrm{OPT}_k)\right)$ in expectation.

Theorem [multiple-round result]: $O\left(\frac{\sigma_{\max}(\mathrm{OPT}_k)}{\sigma_{\min}(\mathrm{OPT}_k)} \cdot \frac{1}{\varepsilon}\right)$ rounds gives objective value of $\Omega\left((1-\varepsilon)f(\mathrm{OPT}_k)\right)$ in expectation.

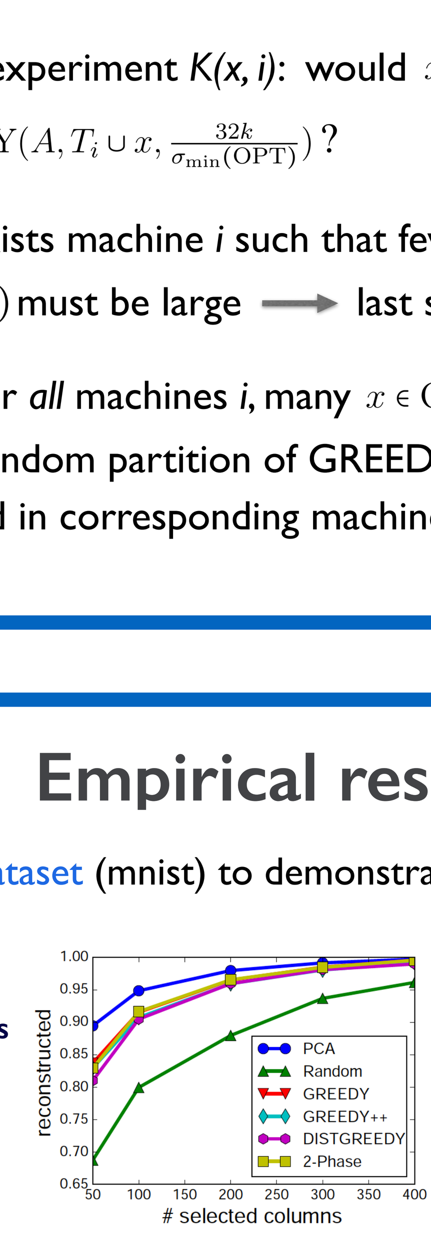
Proof Sketch of 1-round result

- Key thought experiment *K(x, i)*: would $x \in \mathrm{OPT}_k$ be selected from $\mathrm{GREEDY}\left(A, T_i \cup x, \frac{32k}{\sigma_{\min}(\mathrm{OPT})}\right)$?
  - Case 1: Exists machine *i* such that few $x \in \mathrm{OPT}_k$ pass *K(x,i)* ⟶ $f(S_i)$ must be large ⟶ last stage has good choices.
  - Case 2: For *all* machines *i*, many $x \in \mathrm{OPT}_k$ pass *K(x, i)* ⟶ in random partition of GREEDY-CORE, each $x \in \mathrm{OPT}_k$ is selected in corresponding machine with high probability.

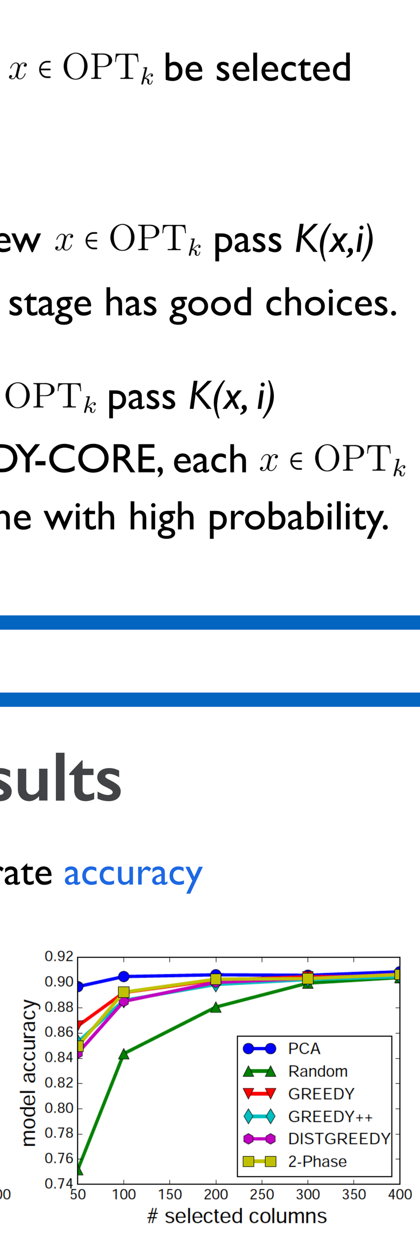
Empirical results

- Small-scale dataset (mnist) to demonstrate accuracy

m = **60K instances**
n = **784 features**
10 classes



fraction of matrix covered by selected features



accuracy of LIBLINEAR SVM using selected features

- Large-scale dataset (news20.binary) to demonstrate scalability

m = **15K instances**
n = **100K features**
0.033% nonzero entries
2 classes

n	Rand	2-Phase	DISTGREEDY	PCA
500	54.9	81.8 (1.0)	80.2 (72.3)	85.8 (1.3)
1000	59.2	84.4 (1.0)	82.9 (16.4)	88.6 (1.4)
2500	67.6	87.9 (1.0)	85.5 (2.4)	90.6 (1.7)

% of matrix covered by selected features.
(Speedup over 2-phase algorithm in parentheses)