Sparse Regression as a Sparse Eigenvalue Problem

Baback Moghaddam
Jet Propulsion Laboratory
California Institute of Technology
baback@jpl.nasa.gov

Amit Gruber, Yair Weiss
The Hebrew University
Jerusalem, Israel
{amitg,yweiss}@cs.huji.ac.il

Shai Avidan
Adobe Systems Inc
Newton, MA USA
avidan@adobe.com

***Abstract*— We extend the l_0-norm "subspectral" algorithms developed for sparse-LDA [5] and sparse-PCA [6] to more general quadratic costs such as MSE in linear (or kernel) regression. The resulting "Sparse Least Squares" (SLS) problem is also NP-hard, by way of its equivalence to a rank-1 sparse eigenvalue problem. Specifically, for minimizing general quadratic cost functions we use a highly-efficient method for direct eigenvalue computation based on partitioned matrix inverse techniques that leads to $\times 10^3$ speed-ups over standard eigenvalue decomposition. This increased efficiency mitigates the $O(n^4)$ complexity that limited the previous algorithms' utility for high-dimensional problems. Moreover, the new computation prioritizes the role of the less-myopic backward elimination stage which becomes even more efficient than forward selection. Similarly, branch-and-bound search for Exact Sparse Least Squares (ESLS) also benefits from partitioned matrix techniques. Our Greedy Sparse Least Squares (GSLS) algorithm generalizes Natarajan's algorithm [9] also known as Order-Recursive Matching Pursuit (ORMP). Specifically, the forward pass of GSLS is exactly equivalent to ORMP but is more efficient, and by including the backward pass, which only doubles the computation, we can achieve a lower MSE than ORMP. In experimental comparisons with LARS [3], forward-GSLS is shown to be not only more efficient and accurate but more flexible in terms of choice of regularization.**

I. Introduction

Consider the general case of a linear system $Ax = y$ with the solution $\hat{x} = \text{argmin}\ ||Ax - y||^2$. We embrace all special cases here (*e.g.* noisy, noiseless, over/under-determined, *etc.*). Typical examples in machine learning and statistics would be in Least Squares (LS) regression. The m-by-n design matrix A can be "short and fat" ($m < n$) for over-complete dictionaries, or "tall and skinny" ($m > n$) in over-determined systems, or square-symmetric in non-parametric local (kernel) regression. We then *impose* sparsity with a direct cardinality constraint: $\text{card}(\hat{x}) = k$ where $k < n$. This leads to the more general sparse quadratic minimization problem:

$$\underline{\text{Sparse MinQuad}}: \qquad \textbf{min} \quad \frac{1}{2} x^T Q\ x - b^T x + c \qquad (1)$$
$$\text{subject to} \quad \text{card}(x) = k$$

where in this case $Q = A^T A$ and $b = A^T y$ (with an irrelevant offset $c = y^T y/2$). Note that additional regularization terms can be easily absorbed into this general quadratic form via $Q \leftarrow Q + \tau R$, where the (scalar) parameter τ defines the amount of regularization and the matrix R defines the type of norm penalty induced. For example, setting $R = I_n$ for an l_2 ridge penalty or using a kernel matrix $R = K$ for RKHS norms ($R = K^{-1}$ would be more common in spatial statistics).

Sparse-MinQuad is non-convex, combinatorial and NP-hard. In fact, at first glance it may appear to require specialized algorithmic machinery to solve. However, Eq.(1) is also a (sparse) generalized eigenvalue problem, with rather simple eigenstructure (only one finite eigenvalue). This equivalence is easily seen by Lagrangian reformulation (ignoring sparsity for the moment) which yields a Generalized Rayleigh Quotient (GRQ) problem: $\max\ (x^T P\ x)/(x^T Q\ x)$, where P is the rank-1 outer-product bb^T. Specifically, the GRQ's principal eigenvector is directly proportional to the LS solution $\hat{x}$ (modulo a sign ambiguity) and the corresponding eigenvalue is related to the LS error: $||A\hat{x} - y||^2 = ||y||^2 - \lambda_{\max}(P, Q)$.

It is important to note that ordinarily there is no real advantage to reformulating the solution of a linear system (or general quadratic minimization) with a GRQ. In fact, the eigenvector solution is typically more costly. However, the addition of a *sparsity* constraint changes things, making the (sparse) eigenvalue approach not only more informative but also more efficient as we will see. Indeed, sparse generalized EVDs — *i.e.,* a GRQ plus a cardinality constraint — were recently addressed in [6], [5], where relatively efficient algorithms for approximate (greedy) and *exact* (branch-and-bound) solutions were derived using a "subspectral" framework, by analyzing the *subspectrum* of (P, Q) — *i.e.,* the eigenvalues of their *submatrices*. These discrete search algorithms were applied to sparse-PCA/LDA for feature (variable) selection, yielding competitive results (*e.g.* the greedy algorithms out-performed various continuous and convex relaxation techniques).

We should note that we are not concerned here with the special case problem of "sparse recovery" where $\hat{x}$ is already sparse to begin with and we then ask under what conditions (on algorithms, the matrix A and/or the noise) can we guarantee exact recovery of the "true" sparsity pattern. Indeed, in all our experiments, and the practical learning scenarios they typify, we fully anticipate the LS solution to be *dense* (and not just because of noise). Hence, we are more interested in the resulting economy of predictions and parsimonious models than in the pursuit (recovery) of any *intrinsic* sparsity.

We begin by reviewing basic properties of sparse EVDs, the variational eigenvalue bounds that inform and define their solutions, and the resulting discrete search algorithms used in [6], [5]. We then show how to exploit the special rank-1 property of a quadratic cost function to derive highly-efficient and streamlined algorithms for general-case sparse quadratic optimization as in Sparse-MinQuad.

II. Background

Several new techniques have been recently developed for sparse spectral decomposition. Zou *et al.* [13] proposed a sparse PCA algorithm (called SPCA) using l_1-penalized regression on regular PCs. Subsequently, d'Aspremont *et al.* [2] relaxed the hard cardinality constraint with a simpler *convex* approximation using semi-definite programming (SDP) for a more "direct" formulation (called DSPCA). In contrast, an alternative *discrete* spectral framework was recently proposed by Moghaddam *et al.* [6], using variational eigenvalue bounds on the covariance "sub-spectrum" derived by the eigenvalue *Inclusion Principle*. This "subspectral" view not only leads to an exact formulation of NP-hard sparse eigenvector problems but also suggests a simple post-processing step ("variational renormalization") which can be used to improve all continuously-derived solutions. Substantial performance gains were obtained using a simple *greedy* search algorithm (GSPCA) that was also faster than most continuous methods, albeit for smaller-scale problems ($n < 1000$). The subspectral framework was also extended to *supervised* discriminant learning problems via *generalized* EVDs [5] which effectively subsumes sparse-PCA as a special case of sparse-LDA. In addition to greedy techniques, exact and *optimal* subspectral algorithms based on branch-and-bound search (with spectral bounds) were also proposed [6], [5]. The computational speedups proposed in Section IV, though ideal for greedy (sequential) search, also apply to the index "pivoting" operations in branch-and-bound.

III. Sparse Generalized EVD

Given a symmetric matrix pair (P, Q) with $Q \succ 0$, we wish to maximize the Generalized Rayleigh Quotient $R(x) = (x^T P\ x)/(x^T Q\ x)$. The optimal solution is of course the eigenvector corresponding to the maximal eigenvalue of the matrix $Q^{-\frac{1}{2}} P Q^{-\frac{1}{2}}$. Without the sparsity constraint the GRQ obeys the global bounds $\lambda_1(P,Q) \leq R(x) \leq \lambda_n(P,Q)$ where λ's are ranked in *increasing* order, thus $\lambda_{\min} = \lambda_1$ and $\lambda_{\max} = \lambda_n$. The *sparse* version of GRQ is obtained by adding a cardinality-constraint $\mathrm{card}(x) = k$ which leads to a non-convex objective function and the NP-hardness. Note that the special case of $Q = I$ defaults to sparse-PCA, therefore any algorithm for sparse-LDA will also solve sparse-PCA.

The key subspectral optimality condition for sparse EVDs is based on the following key equality

$$\frac{x^T P\ x}{x^T Q\ x} = \frac{z^T P_k\ z}{z^T Q_k\ z} \tag{2}$$

where $z \in \mathcal{R}^k$ is the nonzero subvector of x and (P_k, Q_k) are the $k \times k$ principal submatrices of (P, Q) obtained by deleting the rows/columns corresponding to the zero indices of x. Hence the reduced quadratic form in z is equivalent to a standard *unconstrained* GRQ and since this subproblem's maximum is $\lambda_k(P_k, Q_k)$, this must also be the optimal R. This reveals the true combinatorial nature of sparse EVDs wherein solving for the optimal solution is inherently a discrete search for the k indices which maximize $\lambda_{\max}$ of the indexed *subproblem* (P_k, Q_k). In fact, continuous optimization techniques are only useful in yielding a sparsity pattern with which to solve an *unconstrained* subproblem in (P_k, Q_k). Otherwise, they are needlessly sub-optimal and must be "variationally renormalized" using the above equality. It is shown in [6] that the *ad-hoc* method of "simple thresholding" (ST) — setting the smallest loadings to zero and renormalizing to unit-norm — is greatly enhanced by this "fix."

A. Generalized Spectral Bounds

The subspectral $\lambda_{\max}(P_k, Q_k)$ are key in defining sparse GRQ solutions. Due to their combinatorial numbers however, we prefer a more concise characterization in terms of the global $\lambda_i(P, Q)$ which are more readily available. The global spectrum and all its subspectra are indeed related.

Theorem 1 *Generalized Inclusion Principle* [6]. Consider the symmetric pair $P, Q \in \mathcal{S}^n$ with generalized spectrum $\lambda_i(P, Q)$. Let (P_k, Q_k) be a corresponding pair of $k \times k$ principal submatrices with $1 \leq k \leq n$, and generalized subspectrum $\lambda_i(P_k, Q_k)$. Then, for all $1 \leq i \leq n$

$$\lambda_i(P,Q) \quad \leq \quad \lambda_i(P_k,Q_k) \quad \leq \quad \lambda_{i+n-k}(P,Q) \tag{3}$$

In other words, the generalized eigenvalues of (P, Q) form upper and lower bounds for the generalized eigenvalues of all the principal submatrices (P_k, Q_k). Indeed, the subspectrum of (P_m, Q_m) and (P_{m+1}, Q_{m+1}) interleave or *interlace* each other, with the eigenvalues of the larger matrix pair "bracketing" those of the smaller one. For *positive-definite* symmetric matrices (covariances), augmenting P_m to P_{m+1} (adding a new variable) will always *expand* the spectral range: reducing $\lambda_{\min}$ and increasing $\lambda_{\max}$. This *monotonicity* has important theoretical and practical consequences for combinatorial optimization.

Since we wish to maximize the GRQ objective in order to minimize the quadratic in Eq.(1), the relevant inequality in Eq.(3) is the one with $i = k$, thus yielding

$$\lambda_k(P,Q) \quad \leq \quad \lambda_{\max}(P_k,Q_k) \quad \leq \quad \lambda_n(P,Q) \tag{4}$$

This shows that the k-th *smallest* eigenvalue of (P, Q) is a lower bound for a GRQ objective with cardinality k. We should note that this lower bound is of no use in the rank-1 case being considered here, since the $\lambda_k(P, Q)$ are all zero except for $k = n$.

With the discrete approach, branch-and-bound techniques [10] are ideally suited for sparse-LDA. In [5], the generalized inclusion bounds (mainly the *upper* bound in Eq.(4) for subproblems of varying sizes) are used for exact search (ESLDA) to find globally optimal solutions, albeit for smaller problems ($n < 60$) since branch-and-bound can exhibit exponential worst-case complexity.

Greedy techniques like *backward elimination* can also exploit the monotonic nature of nested submatrices and their "bracketing" eigenvalues: start with the full index set $I = \{1, 2, \ldots, n\}$ and sequentially delete the variable j which yields the maximum $\lambda_{\max}(P_{\backslash j}, Q_{\backslash j})$ until only k elements remain. For *small* cardinalities $k \ll n$, the polynomial cost of backward search makes its forward counterpart *forward selection* more attractive (despite it being potentially “myopic”): start with the null index set $I = \{\}$ and sequentially add the variable j which yields the maximum $\lambda_{\max}(P_{+j}, Q_{+j})$ until k elements are selected.

Various theoretical performance guarantees exist for greedy search which recommend its use. For example, in [6] we show that the GRQ obeys certain ”nesting” bounds on backward search, where among all the n possible $(n-1)$-by-$(n-1)$ principal submatrices of the pair (P, Q), obtained by deleting a single (say j-th) row and column, there is *at least* one whose objective value is no less than $\frac{n-1}{n}$ of $\lambda_{\max}(P, Q)$

$$\max_{j} \quad \lambda_{\max}(P_{\backslash j}, Q_{\backslash j}) \quad \geq \quad \frac{n-1}{n}\, \lambda_{\max}(P, Q) \qquad (5)$$

This “best-case” nesting bound can be applied recursively in backward search to show, for example, that our greedy solutions are guaranteed to achieve no less than the fraction k/n of the initial (global) $\lambda_{\max}(P, Q)$. In practice one captures far more variance than this bound, due to the overly pessimistic assumptions implicit in the recursion of Eq.(5). Note that a lower bound on the sparse GRQ objective directly translates to an upper bound on the MSE in the regression setting.

The greater efficiency of forward search can be combined with the greater performance of backward search. The resulting *bi-directional* or ”dual-pass” search was proposed in [6] for sparse-EVDs: simply pick the better of the 2 solutions found by forward and backward passes. This strategy has led to very good results (*e.g.,* out-performing various leading continuous algorithms for sparse-PCA). A full dual-pass search has the added benefit of giving near-optimal solutions for *all* cardinalities (at once), with a complexity that is far less demanding than finding single k solutions (one at a time).

IV. Efficient Eigenvalue Computation

In the general setting of full-rank matrices the subspectral algorithms for sparse-PCA [6] and sparse-LDA [5] will require $O(k^3)$ EVDs for each subproblem (P_k, Q_k) examined in the discrete search. This computational burden is essentially unavoidable and leads to the usual difficulties with high-dimensional problems. Mainly, a full forward pass (for all k) has $O(n^4)$ complexity while a full backward pass has $O(n^5)$ complexity. Unfortunately, this limits the use of the often more accurate *backward* search for very large n. Nevertheless, it is possible to significantly speed-up both the forward and backward passes (independently) in the special case of a rank-1 GRQ and do so in such a way that the new backward search runs even *faster* than the improved forward search.

As was recently shown in [7], for general quadratic cost functions such as the LS problem in Eq.(1), the equivalent GRQ maximization can be made exceedingly efficient, as the only finite eigenvalue $\lambda_{\max}(P_k, Q_k)$ can be computed in closed-form as $b_k^T Q_k^{-1} b_k$. This is due to the rank-1 P matrix in the GRQ numerator being a simple outer-product $P = bb^T$. Hence the computational complexity of GSLS hinges on our ability to invert Q_k submatrices ”on-the-fly.” A naive implementation, even with a Cholesky decomposition, is still grossly inefficient as Q_k and $Q_{k\pm1}$ differ by a single row/column. Therefore, partitioned matrix inverse techniques [4] using simple rank-1 updates for the required Q_k^{-1} are highly recommended. The implementation details are given in the APPENDIX.

Moreover, by computing the *increments* of change in the GRQ (instead of final values), intermediate terms (matrix-vector byproducts) will cancel, leading to an essentially ”loop-free” array computation over the available indices being considered for inclusion/deletion. Consequently, these optimized algorithms offer a significant speed-up (*e.g.,* by several orders of magnitude). This allows subspectral algorithms to be used in a much wider range of optimization problems.

For example, a full (dual-pass) run of the algorithm in [5] for a matrix of size $n = 1024$, using ”on-the-fly” Cholesky computation of $\lambda_{\max}(P_k, Q_k)$, takes approximately 12 hours (in Matlab 7.2 on a 3.2GHz P4) where 80% of the cputime is taken up by the *backward* pass. In stark contrast, using our rank-1 updates on partitioned inverses requires only 2 minutes, where the backward pass now takes up only 40% of the total cputime (due to its simpler rank-1 updates, see APPENDIX). This is a speed-up factor of 340. For even larger matrices ($n > 2000$) the cputimes were $\times 10^3$ faster than the default eigenvalue decomposition. For example, with $n = 2048$ the original algorithm required nearly 2 weeks of cputime whereas our optimized version required just 20 minutes, and with its backward search using only $1/3$ of the total cputime.

V. Experiments

We now demonstrate the efficacy of the GSLS algorithm for solving different types of sparse regression problems (both primal and dual) using several well-known ML benchmark datasets. We will demonstrate not only the speed advantages (compared to ORMP) but also the quality of solutions (compared to LARS) obtained in both training (optimization) and testing (prediction on held-out data).

A. Comparisons to ORMP

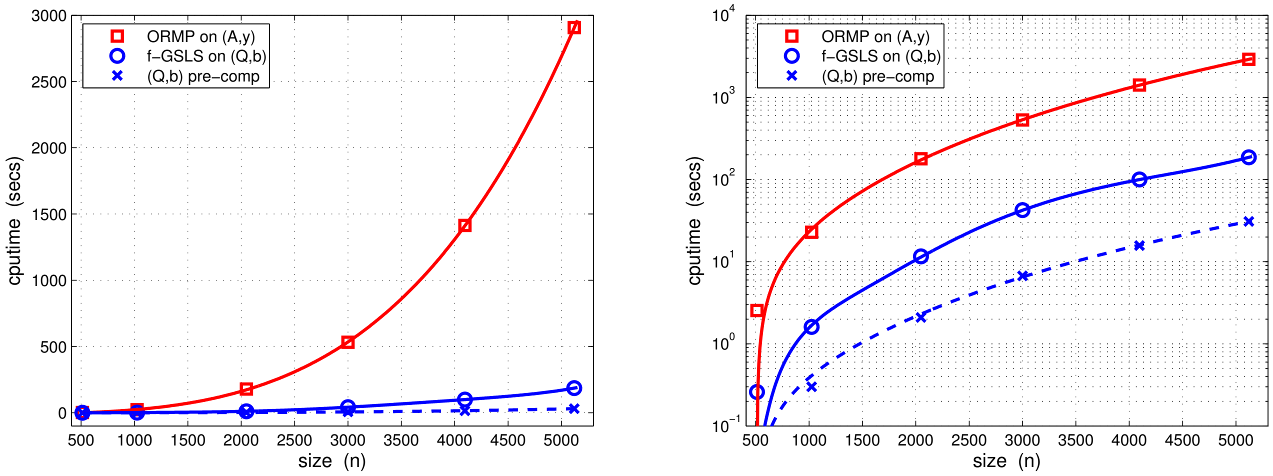
We first compare the forward stage of our GSLS to Natarajan’s algorithm (ORMP). Since they are exactly equivalent we focus on the running times for various (n, k). For ORMP we use the standard Natarjan algorithm as detailed in Nair *et al.* [8], but optimized for Matlab (*e.g.* minimizing for-loops and maximizing array computations as much as possible).

In kernel regression experiments (with a square-symmetric $A = K$) we included the cputime for GSLS’s pre-computation of $Q = K^T K$ and $b = K^T y$. The total running times for Natarjan/ORMP and GSLS were averaged for different (n, k) and were used to compute speed-up factors (in favour of GSLS) shown in Table IV. Forward GSLS is between $\times 3$ to $\times 15$ faster than ORMP. Although not all (n, k) values for this table are typical of a *sparse* model (with $k \ll n$), it is instructive to see how the computational complexity of the two algorithms scales for ”not-so-sparse” cases.

	$k\ =\ 4$	8	16	32	64	128	256	512
$n\ =\ 512$	4.6	8.2	9.1	9.9	9.5	6.9	4.0	1.9
1024	2.6	5.1	8.9	12.9	15.4	13.3	8.5	4.0
2048	1.5	3.2	6.1	10.9	16.8	19.1	14.5	7.3
4096	0.9	1.8	3.5	6.8	12.5	19.2	20.4	12.8
5120	0.7	1.5	3.0	5.9	11.2	18.5	22.5	15.6

TABLE I

Speed advantage of forward GSLS compared to Natarajan's (ORMP) for different n and k. The cputimes for GSLS include the pre-computation of $Q = A^T A$ and $b = A^T y$.



(a) (b)

Fig. 1. Total running times for Natarajan (ORMP) and forward-GSLS at fixed sparsity $(k = n/10)$ *vs.* the matrix size n on (a) linear and (b) log scales. Times for GSLS's pre-computation of (Q, b) are shown in dashed.

A more realistic scenario would be to fix the sparsity (*e.g.* to 10% "diversity") and then increase n. This is illustrated in Figure 1 which plots the running times for the two algorithms for increasing n with fixed $k = n/10$. Forward GSLS is now more than $\times 10$ faster than ORMP. Furthermore, the cost of pre-computing (Q, b) is quite negligible, taking up less than 10% of the total cputime.

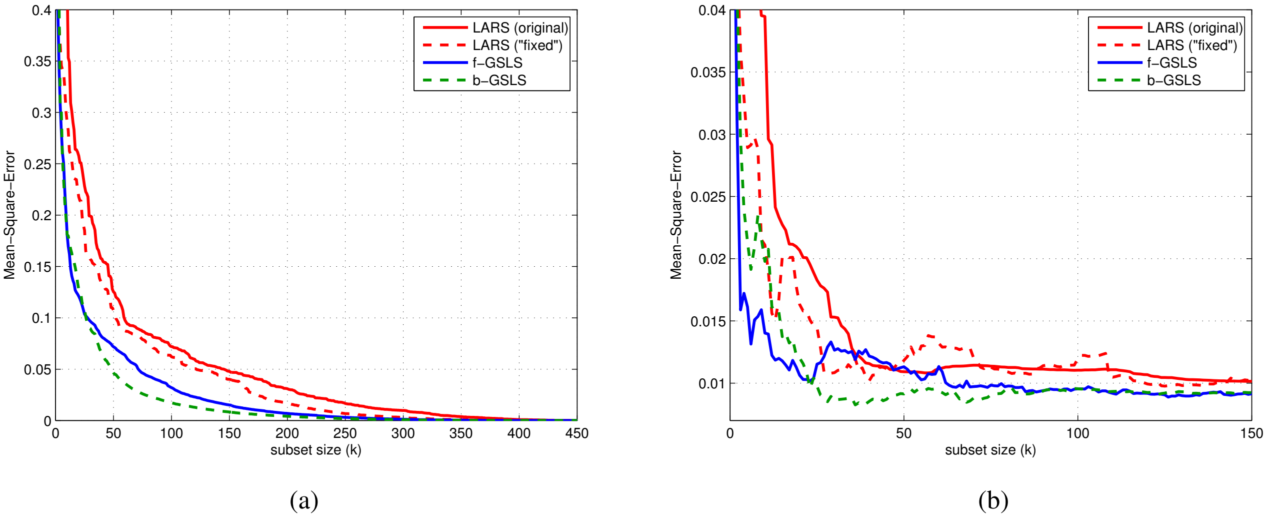
B. Comparisons to LARS

We next compare our GSLS algorithm to the current state-of-the-art algorithm LARS for *Least Angle Regression* [3] in various sparse regression settings (subset selection). For this, we use the LARS-EN Matlab Toolbox of [12] which is closely based on the original R/S-plus LARS toolbox. All computations were carried out in Matlab 7.2 on a 3.2GHz Pentium 4.

1) Sparse Kernel Regression on Boston Housing Data: We start with sparse kernel regression (basis selection) with the Boston Housing dataset, using the standard train/test split of 455/51 examples. We first fit a squared-exponential kernel to the training set using the marginal likelihood optimization of a full (non-sparse) model using the GPML toolbox of Rasmussen & Williams [11]. The 3 hyperparameters (signal variance, kernel width and noise variance) were then used to define the kernel function for both training and prediction. The LS solution of $y = Kx$ was then "sparsified" using a full (dual) pass of GSLS, with $Q = K^T K$ and $b = K^T y$, to find subsets of all cardinalities, and then compared to a full forward pass of LARS.

Note that as this is a dual or *kernel* regression problem, the sparsity pattern of $\hat{x}$ will identify the reduced "active set" of *training* data and not the original variables. The non-zero elements of $\hat{x}$ are equivalent to $\hat{x}_s = K_s \setminus y$ where K_s is the subset of k columns from the training set kernel matrix. The standard method for making predictions on a new input is to simply compute $y_* = k_*^T \hat{x}_s$ were k_* is the k-dimensional vector of active set kernel functions and $\hat{x}_s$ is the non-zero subvector of $\hat{x}$.

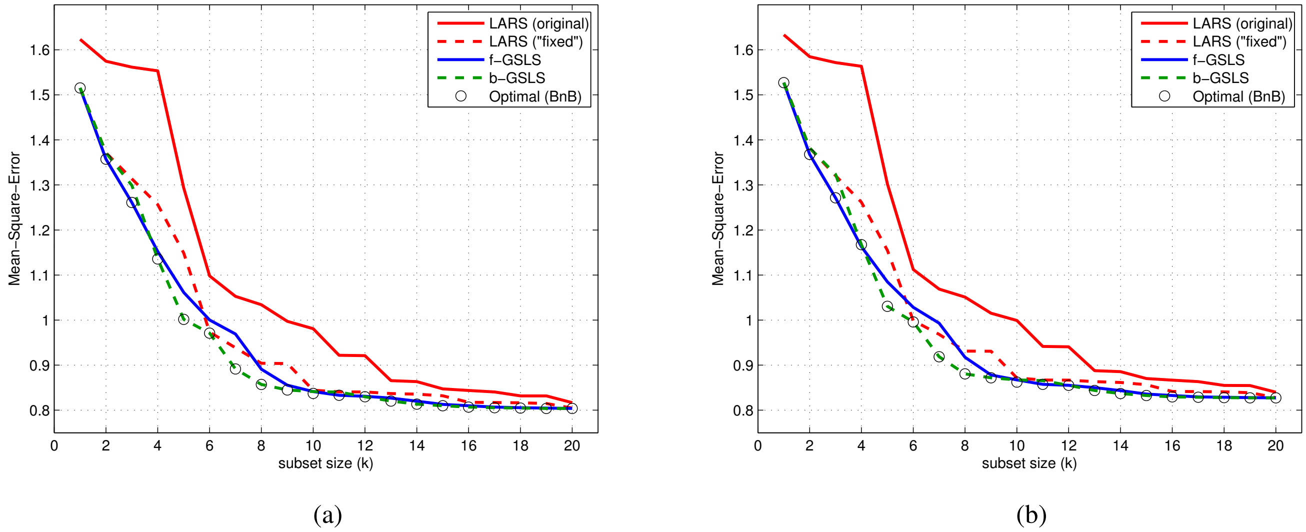
For this moderately-sized problem $(n = 455)$ the total running times of GSLS, including the (Q, b) pre-computation, were quite negligible using the inverse partitioned matrix updates: 1.5 secs for forward search and 1.3 secs for backward search. In sharp contrast, a Cholesky decomposition for the eigenvalue computation gave forward and backward running times of 258 secs and 920 secs, respectively. This represents 2-3 orders of magnitude speedups using the improved GSLS algorithm. Furthermore, the partitioned matrix inverse techniques lead to a backward elimination stage which is comparable in speed (often *faster*) than forward selection. This



(a)

(b)

Fig. 2. Sparse kernel regression on Boston Housing data: MSE *vs.* sparsity parameter k on (a) the training set and (b) the test set. Subsets and coefficients were found by LARS, forward-GSLS (same as ORMP) and backward-GSLS.



(a)

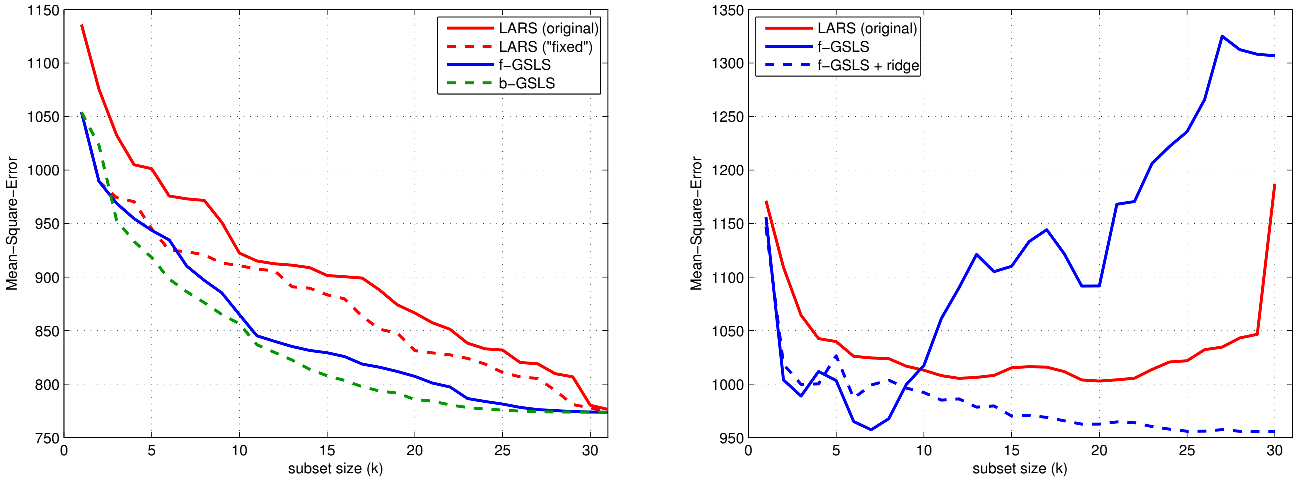
(b)

Fig. 3. Sparse regression on SARCOS data: MSE *vs.* sparsity parameter k on (a) the training set and (b) the test set. Subsets and coefficients were found by LARS, forward-GSLS (same as ORMP), backward-GSLS, and ESLS (Branch-and-Bound).

is significant, since the quality of sparse solutions found by backward elimination is generally superior to those found by forward selection. The running time for LARS was 4.2 secs (or roughly $\times 3$ slower than GSLS).

Figure 2(a) shows the training set MSE for the best subsets found by GSLS and LARS. For the latter, we use both the original coefficients and their “re-solved” versions, following the “variational renormalization” suggested in [6]. This “fix” corresponds to using the optimal LS coefficients *after* the sparsity pattern has been found and will naturally improve the fit (optimize the objective). However, this is not always recommended for making *predictions* due to the risk of overfitting. Figure 2(a) also shows that for $k > 20$ backward-GSLS has a lower MSE than its forward counterpart, confirming its ”less myopic” tendency for subset selection. Since forward-GSLS is equivalent to Natarajan’s (ORMP), this example demonstrates the potential advantage of ”dual-pass” GSLS over Orthogonal Matching Pursuit, both in terms of speed and quality.

The LARS training set error in Figure 2(a) reflects the algorithm’s conservative selection strategy: picking different indices than ORMP and not fully committing to their optimal LS coefficients. Nevertheless, optimizing LARS’s coefficients *after* the fact (with the “fix”) can often be a useful compromise (keeping the same variables but modifying their weights). This can be seen in the predictions on the held-out test set in Figure 2(b), where except for a small range of cardinalities the “fix” does in fact improve LARS’s predictions. But overall,



(a)

(b)

Fig. 4. Sparse regression on Wisconsin BC data: MSE *vs.* sparsity parameter k on (a) the training set and (b) the test set. Subsets and coefficients were found by LARS and GSLS. In the test set (b) forward-GSLS is shown with and without a ridge penalty term ($R = I$).

forward-GSLS is clearly making better predictions than LARS on this dataset. Moreover, GSLS is faster than LARS even after including the backward pass, and the intersection of the two passes yields better (non-nested) subsets.

2) Sparse Regression for SARCOS Inverse-Kinematics Data: We next compare GSLS and LARS for predicting torques in the robot inverse-kinematics model SARCOS dataset from [11]. Here we use the 21-dimensional inputs (positions, velocities and accelerations for 7 joints) to predict input torques for the 6th joint (as this seemed to be the more difficult of the 7 torques). The 40k training set measurements densely sample the joint state-space, hence we would expect the prediction performance (on 4k measurements) to correspond fairly well with the training set performance. This is confirmed by the results shown in Figure 3 in which we also show the globally optimal subsets found by ESLS using branch-and-bound. For this small-scale problem ($n = 21$) GSLS finds the optimal subsets for most k. In fact, judging by the "fixed" LARS curve in Figure 3(a), LARS does in fact find the optimal subsets in few cases ($k = 2, 6, 10$).

3) Sparse Regression for Wisconsin Breast Cancer Data: Finally, we use the Wisconsin Breast Cancer dataset (from the UCI ML Repository) to predict the "time-to-recurrence" variable by regressing on the other 32 covariates. The 194 cases were randomly split into equal train/test partitions of size 97. Due to the small size of the training set and the relatively non-trivial task of predicting this particular "outcome" variable, we should expect some over-fitting, leading to poor predictions. This is confirmed by the results shown in Figure 4 where for clarity we have omitted the curves for backward-GSLS and "fixed" LARS in Figure 4(b), neither of which does substantially better than their alternatives. We see that despite its low MSE at first ($k < 10$) forward-GSLS eventually leads to poor predictions for larger k, especially compared to LARS which shows better generalization. Of course, this is not unexpected, as doing better in the optimization task in Figure 4(a) is no guarantee of doing better in the prediction task in Figure 4(b). But as mentioned in the introduction we can guard against over-fitting by adding a ridge penalty term to the GSLS objective with $Q \leftarrow Q + \tau R$ (with $R = I$). Figure 4(b) shows the resulting *penalized* forward-GSLS doing quite well, even better than LARS. This demonstrates the greater flexibility (if not superiority) of GSLS compared to LARS, since we have the freedom to control not only the *amount* of shrinkage (with τ) but also the *type* of regularization (with R), hence being able to implement a variety (or even *mixture*) of ridge/RKHS penalties and arbitrary Gaussian priors on x.

VI. Discussion

We have shown that the sparse LS solution of $y = Ax$ is equivalent to a rank-1 subspectral optimization problem (*i.e.* a sparse generalized EVD), which can also be used to solve a general-case sparse quadratic problem such as Sparse-MinQuad in Eq.(1). We argued that the alternative subspectral approach is more illuminating (given the variational bounds in Theorem 1) and leads to more exact (well-defined) solutions by way of eigenvalue *subproblems* (in contrast to l_1 regularized continuous techniques like Basis Pursuit [1] which solve a convex approximation of the problem).

ORMP spends a large portion of its cputime projecting residuals on the columns of A and then renormalizing (orthogonalizing) them. Despite this, ORMP runs faster than the original subspectral algorithm in [5] which used standard eigenvalue computation for $\lambda_{\max}(P_k, Q_k)$. Our inverse partitioned matrix version leads to dramatic $\times 10^3$ speedups which subsequently make GSLS $\times 10$ more efficient than ORMP.

Using a *dual-pass* search, GSLS can also out-perform ORMP in terms of quality as shown in Figure 2(a) for example.

For finding solutions at all cardinalities, the dual-pass GSLS is *still* more efficient than a forward-only ORMP, especially for large n. Indeed, in such cases GSLS's backward pass is more efficient than the forward by virtue of the partitioned matrix inverse implementation. The ability to surpass ORMP in terms of both quality *and* speed is significant as this algorithm is often viewed as the "gold standard" in the field, and since any improvements or enhancements to it are typically made by sacrificing quality for speed (see discussions in [8]).

Sparse quadratic optimization problems are common to many application areas in diverse fields. Indeed, a wide range of sparse estimation problems can be reduced to Eq.(1) with the appropriate (Q, b). In the sparse *regression* setting (starting from $y = Ax$) we showed that the pre-requisite computation of (Q, b) — something that is *not* required by ORMP — was in fact quite negligible (requiring less than 10% of the total cputime). Of course, if A is highly over-complete $(m \ll n)$ and n very large, then the memory storage of Q may become costly. In such cases ORMP may be preferred, as it works with (overwrites) the same A matrix, thus requiring $O(nm)$ storage as opposed to $O(n^2)$ with GSLS. Nevertheless, many optimization problems come with (Q, b) already defined, in which case ORMP (or LARS) is of no direct use in solving them. Therefore, for sparse quadratic optimization problems (without inequalities) our subspectral algorithm can be put to very good practical use.

Compared to the LARS algorithm, ORMP on (A, y) can sometimes be too aggressive or "too greedy" [3]. The same criticism applies to our equivalent forward-GSLS. However, in addition to being ×10 faster than ORMP (and ×3 faster than LARS) we can in fact achieve better predictions by simply adding different types of regularization to forward-GSLS via the matrix Q, for example the ridge penalty in Figure 4(b). In contrast, it is less clear how LARS could implement arbitrary forms of regularization, other than its own "built-in" form with conservative ("half-angle") steps.

Furthermore, backward-GSLS is often superior to the forward selection of ORMP and LARS and can be readily incorporated into a *dual*-GSLS (with less than twice the cputime). In fact, due to the partitioned matrix updates, backward-GSLS is significantly faster than LARS for $\sim 10^3$ variables, *including* the pre-computation of (Q, b). For example, in tests with $n = 4000$, LARS took 11.6 hours, forward-GSLS 3.2 hours (×4 faster) and backward-GSLS only 0.65 hours (×18 faster). Also, GSLS tends to have lower MSE in both the optimization (training) and prediction (testing). Moreover, GSLS finds variable subsets that are *not* necessarily nested.

Finally, the efficient eigenvalue computations in Section IV (detailed in the APPENDIX below) can also be applied to *binary* (2-class) sparse-LDA and can thus be used for the GSLDA algorithm in [5], with Q as the within-class covariance matrix and b as the difference of class means. In fact, the partitioned matrix method used here for a sparse regression defined by (A, y) was first derived for speeding-up sparse binary classification given (Q, b) in sparse-LDA [7].

Acknowledgments

This research was conducted at the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration.

References

[1] S. Chen, D. L. Donoho, and M. A. Saunders. Atomic Decomposition by Basis Pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1999.

[2] A. d'Aspremont, L. El Ghaoui, M. I. Jordan, and G. R. G. Lanckriet. A Direct Formulation for Sparse PCA using Semidefinite Programming. In *Neural Information Processing Systems 17*. 2004.

[3] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least Angle Regression. *Annals of Statistics*, 32(2):407–499, 2004.

[4] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins Press, 1989.

[5] B. Moghaddam, Y. Weiss, and S. Avidan. Generalized Spectral Bounds for Sparse LDA. In *International Conference on Machine Learning*. ICML'06, June 2006.

[6] B. Moghaddam, Y. Weiss, and S. Avidan. Spectral Bounds for Sparse PCA: Exact & Greedy Algorithms. In *Neural Information Processing Systems 18*. 2006.

[7] B. Moghaddam, Y. Weiss, and S. Avidan. Fast Pixel/Part Selection with Sparse Eigenvectors. In *International Conference on Computer Vision*. ICCV'07, Rio de Janeiro, Brazil, October 2007.

[8] P. B. Nair, A. Choudhury, and A. J. Keane. Some Greedy Learning Algorithms for Sparse Regression and Classification with Mercer Kernels. *Journal of Machine Learning Research*, 3:781–801, 2002.

[9] B. K. Natarajan. Sparse Approximate Solutions to Linear Systems. *SIAM Journal of Computing*, 25(2):227–234, 1995.

[10] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley, New York, 1988.

[11] C. E. Rasmussen and C. K. I. Williams. http://guassianprocesses.org/gpml/code.

[12] K. Sjöstrand. Matlab implementation of LASSO, LARS, the Elastic Net and SPCA. Informatics and Mathematical Modelling, Technical University of Denmark (DTU), 2005.

[13] H. Zou, T. Hastie, and R. Tibshirani. Sparse Principal Component Analysis. *Journal of Computational and Graphical Statistics*, 15(2), 2003.

Appendix

For forward greedy search, let s be the current subset of k indices and $t = s \cup i$ for a candidate $i \notin s$. Given the current inverse Q_{ss}^{-1}, the new augmented inverse is

$$Q_{tt}^{-1} = \left[\begin{array}{cc} Q_{ss}^{-1} + r_i v_i v_i^T & -r_i v_i \\ -r_i v_i^T & r_i \end{array} \right]$$

where $v_i = Q_{ss}^{-1} Q_{si}$ with (si) indexing the rows in s and the i-th column of Q and the scalar $r_i = 1/(Q_{ii} - Q_{si}^T v_i)$. The new candidate GRQ objective is $\lambda_{\max}(b_t b_t^T, Q_t) = b_t^T Q_{tt}^{-1} b_t$. If we expand the expression for the *incremental* change $\Delta_i = \lambda_{\max}(b_t b_t^T, Q_t) - \lambda_{\max}(b_s b_s^T, Q_s)$ intermediate terms will cancel, leading to a (loop-free) Matlab *array computation* for Δ.

For backward greedy search (going from the index set t down to s), by partitioning the current inverse as follows

$$Q_{tt}^{-1} = \left[\begin{array}{cc} U_{ss} & u_i \\ u_i^T & z_i \end{array} \right]$$

a simpler rank-1 update results: $Q_{ss}^{-1} = U_{ss} - u_i u_i^T / z_i$. Once again, by solving for the increments Δ_i, many unnecessary calculations can be avoided. This backward computation is now even more efficient than the forward one, since "growing" an inverse is harder than "shrinking" it.