Improved Greedy Algorithms for Sparse Approximation of a Matrix in terms of Another Matrix

Crystal Maung
Department of Computer Science
The University of Texas at Dallas

Haim Schweitzer
Department of Computer Science
The University of Texas at Dallas

Abstract

We consider simultaneously approximating all the columns of a data matrix in terms of few selected columns of another matrix that is sometimes called "the dictionary". The challenge is to determine a small subset of the dictionary columns that can be used to obtain an accurate prediction of the entire data matrix. Previously proposed greedy algorithms for this task compare each data column with all dictionary columns, resulting in algorithms that may be too slow when both the data matrix and the dictionary matrix are large. A previously proposed approach for accelerating the run time requires large amounts of memory to keep temporary values during the run of the algorithm. We propose two new algorithms that can be used even when both the data matrix and the dictionary matrix are large. The first algorithm is exact, with output identical to some previously proposed greedy algorithms. It takes significantly less memory when compared to the current state-of-the-art, and runs much faster when the dictionary matrix is sparse. The second algorithm uses a low rank approximation to the data matrix to further improve the run time. The algorithms are based on new recursive formulas for computing the greedy selection criterion. The formulas enable decoupling most of the computations related to the data matrix from the computations related to the dictionary matrix.

1 Introduction

Representing large amounts of data in terms of a small number of atoms taken from a given dictionary is computationally challenging. Greedy algorithms that solve this problem are routinely applied in areas such as signal processing and machine learning (e.g., [1, 2, 3, 4, 5]), where the amount of data and the size of the dictionary are manageable. But applying previously proposed algorithms to large databases may be impractical.

Consider data consisting of N items and a dictionary consisting of n atoms. Previously proposed greedy algorithms (e.g., [6, 7]) compare each data item with all dictionary atoms, leading to run time that is at least proportional to nN. In fact, taking into account the comparison time the run time of a naive implementation is $O(kmnN)$, where k is the selection size and m is the size of a single data item. While there are many studies that analyze the accuracy of the greedy scheme (e.g., [6, 7, 8, 9, 10, 11]) there is a relatively small amount of published work aimed at reducing the run time of the greedy algorithms. We are aware of the studies in [7,12]. The study in [12] addresses the number of passes over the data, but it is applied only to the case where the data matrix has a single column. As discussed in Section 2, the algorithm established in [7] by Civril and Magdon-Ismail, that we call "The CM Algorithm", reduces the run time to $O(mnN)$ by keeping in memory nN intermediate values. Additional savings can be obtained by taking advantage of the dictionary sparsity. The CM can take advantage of sparse dictionaries when the average number of non-zeros

in each atom is reduced from m to no less than k. As explained in Section 2.7 sparser dictionaries do not yield an improved asymptotic run time for the CM.

We propose two new greedy algorithms. They require significantly less memory than previously proposed algorithms, retaining in memory only two values for each column of the dictionary. This compares favorably with previous algorithms that typically require partial orthogonalization of the dictionary matrix. When the dictionary matrix is sparse, the partial orthogonalization is non-sparse, so that m values may have to be retained for each column of the dictionary. Furthermore, unlike the CM our algorithms can take advantage of higher levels of dictionary sparsity to reduce their asymptotic complexity. Their asymptotic run time is reduced with higher levels of sparsity as long as the number of non-zeros is above km. This should be compared with the CM that cannot benefit from sparsity levels where the number of non-zeros falls below kn, and with the naive method that cannot take advantage of sparse dictionaries.

The first proposed algorithm is exact, producing the same output as other greedy algorithms. The second algorithm uses a low rank approximation of the data matrix to further improve the run time. The result is no longer identical to exact greedy algorithms, but it is very similar and allows for much faster run time.

The proposed algorithms make use of the (surprising) observation that the computations related to the data matrix can "almost" be decoupled from the computations related to the dictionary matrix. Specifically, we derive recursive formulas that show how to determine the selection of the j+1$^{\text{th}}$ atom using mostly computations that were already performed when the j^{th} atom was selected. Since what remains to be calculated is independent of N, the computations involving N are decoupled from the computations involving n. Using these recursive formulas, which to the best of our knowledge are new, the only computations that involve both n and N are those performed while selecting the first atom.

The first algorithm that we propose, the SOLS-exact, computes the values necessary to select the first atom, and then applies the recursive formulas to select the additional atoms. When applied to dense dictionaries it has the same asymptotic complexity as the CM, but it uses significantly less memory. As discussed earlier it performs better than the CM with sparse dictionaries. For example, if the average number of nonzero entries in each dictionary column is a constant independent of the column size, the run time of the SOLS-exact (under the assumption that km is not too big) would be $O(nN)$. This is significantly better than the $O(knN)$ of the CM and the $O(kmnN)$ of the naive approach under the same assumptions.

The second algorithm that we propose, the SOLS-lowrank, improves over the first algorithm by replacing the data matrix with a low rank approximation. Recent studies (e.g., [13, 14, 15, 16]) have shown that low rank approximations can be computed very rapidly with randomized algorithms, so that the run time of the proposed algorithm is typically not affected by the complexity of computing the low rank approximation. When the data matrix is approximated by a rank d matrix, the complexity of running the second algorithm is $O(dmn)$ for dense dictionaries, and $O(dn)$ for very sparse dictionaries (assuming again that km is not too big compared to n).

1.1 Problem statement and related work

Let Y be a data matrix of m rows and N columns. Its columns are $y_1, \ldots, y_N$, where a column y_i is an m dimensional vector. Similarly, let X be a "dictionary" matrix of m rows and n columns. Its columns, also called "atoms", are $x_1, \ldots, x_n$, where a column x_i is an m dimensional vector. We consider the problem of selecting a subset of k columns from X that can be used to approximate all the columns of Y. Specifically, suppose the columns $x_{s_1}, \ldots, x_{s_k}$ are selected from X, then each column y_i of Y can be approximated by the linear combination:

$$y_i \approx a_{i,1} x_{s_1} + \ldots + a_{i,k} x_{s_k}$$

where $a_{i,1} \ldots a_{i,k}$ are coefficients associated with y_i and chosen to minimize the approximation error. In matrix form this can be written as:

$$Y \approx SA \tag{1}$$

where $S = (x_{s_1}, \ldots, x_{s_k})$ is the $m \times k$ selection matrix, and the coefficients matrix A is $k \times N$. The quality of the selection S is determined by the following error:

$$E = \min_A \|Y - SA\|_F^2 \tag{2}$$

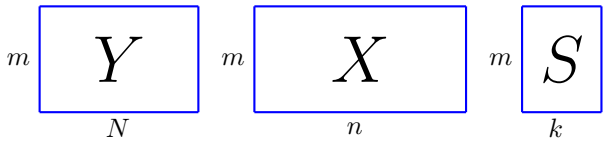


Figure 1: The relevant complexity parameters. The data matrix Y is $m \times N$, the dictionary matrix X is $m \times n$, and the selection matrix S is $m \times k$. The matrix X has z_x nonzero elements. The matrix Y may by approximated by a d-rank matrix.

Input: X, Y, k. Output: S.
0. Global pre-calculations (done once).
1. Run iteration $j = 1, \ldots, k$:
  1.0 Pre-calculations for Iteration-j.
  1.1 for $i = 1, \ldots, n:$ evaluate $f(x_i)$.
  1.2 $x_{s_j} = \arg\max f(x_i)$. Add x_{s_j} to S.
  1.3 Post-calculations for Iteration-j.

Figure 2: Algorithm GREEDY. The greedy framework of algorithms discussed in the paper.

where the matrix norm is the Frobenius norm. We are interested in the case where n (the dictionary size) is big and k (the selection size) is small, as illustrated in Fig.1. This approximation and closely related variants have attracted a lot of attention. Optimal solutions, as well as approximations within a constant are known to be NP-hard even for the N=1 case [17, 18]. There are two common approaches for computing an approximate solution. The first approach recasts the approximation as an l_1 convex optimization (e.g. [19, 20]). The second approach is greedy, selecting the k columns from X one after the other (e.g. [6, 7, 8, 10, 11, 17, 21, 22]). The l_1 algorithms typically use linear programming, and there is ample evidence (e.g. [12]) that they are much slower than the greedy approach. In this paper we consider only the greedy approach.

Referring to the matrix X as a dictionary is common in the signal processing literature (e.g. [1]), where in the heavily analyzed case of N=1 the matrix Y is called "the signal". The case N=1 is also common in machine learning and related studies, where it is known as feature selection in linear regression (e.g. [2, 3, 5]).

Another heavily researched special case has X=Y, taking the dictionary to be the same as the data matrix. Here the challenge is to approximate the data matrix Y by a small subset of its own columns. This is sometimes called "unsupervised feature selection" in machine learning (e.g. [2]) and "column subset selection" in computational linear algebra (e.g. [23]). Computing the optimal solution for this special case is believed to be NP-hard [24], although there are efficient algorithms with a guaranteed good approximation. See, e.g. [25, 26].

1.2 Paper organization

The paper is organized as follows. Section 2 reviews previously proposed greedy algorithms for the sparse matrix approximation. Their run time and memory requirements are summarized in a table shown in Fig. 3. Our main result is based on recursive formulas, derived in Section 3, for solving the SOLS variant. The improved algorithms are presented in sections 4 and 5. Experimental results are discussed in Section 6.

2 Greedy Algorithms

In this section we describe the main variants of previously proposed greedy algorithms for computing a sparse approximation in terms of dictionary atoms. Algorithm GREEDY in Fig. 2 is the basic greedy approach for selecting a subset of k columns from the matrix X to estimate the matrix Y. It includes as special cases all the algorithms discussed in this paper. For clarity, our presentation of

Algorithm	Reference	N	Memory	Run Time	Sparse X Run Time
MP	[21]	1	$O(m)$	$O(kmn)$	$O(kz_x)$
OMP	[1]	1	$O(km)$	$O(kmn)$	$O(kz_x)$
OLS	[17]	1	$O(mn)$	$O(kmn)$	$O(kmn)$
SOMP	[8]	N	$O(mN))$	$O(kmnN)$	$O(kz_xN)$
SOLS	[27]	N	$O(mn)$	$O(kmnN)$	$O(kmnN)$
CM	[7]	N	$O(mn)+nN$	$O(mnN)$	$O(N(kn+z_x))$
ISOLS-exact	this paper	N	$O(km)+2n$	$O(mnN)$	$O(N(km+z_x))$
ISOLS-lowrank	this paper	N	$O((k+d)m)+2n$	$O(dmn+T_{\text{LR}}(Y))$	$O(d(km+z_x)+T_{\text{LR}}(Y))$

Figure 3: Comparison of various algorithms. The SOLS, CM, and ISOLS-exact produce identical output. $T_{\text{LR}}(Y)$ is the cost of computing a low rank approximation for Y.

Algorithm GREEDY ignores some steps that do not have a significant impact on the run time, such as restricting the selection in 1.2 only to previously unselected vectors.

The algorithm run time is typically dominated by Step 1.1, which computes $f(x)$ (the selection criterion) kn times. In estimating the required amount of memory we assume that the storage of the input matrices X and Y is "free", but that this storage cannot be reused. This assumption is justified in situations where the matrices are sparse, or when they are stored in a secondary memory that cannot be (cheaply) overwritten.

2.1 Orthogonal projections

The following definitions and notation are useful in describing the various algorithms. We write $\mathsf{Orthogonal}(x, G)$ for the error (vector) of estimating the vector x in the column space of the matrix G. It is defined as follows:

$$\begin{aligned}&\mathsf{Orthogonal}(x, G) = x - Ga\\ &\text{where } a \text{ minimizes } \ |x - Ga|^2\end{aligned}$$

The vector Ga is the best l_2 approximation of the vector x in the column space of G. A common technique for computing orthogonal projections (see [23]) is to compute a matrix Q with columns that form an orthonormal basis to the column space of G. This can be done, for example, by the QR factorization of G. Given such Q we have:

$$\begin{aligned}\mathsf{Orthogonal}(x, G) = \mathsf{Orthogonal}(x, Q) &= x - QQ^T x\\ &= x - \textstyle\sum q_j q_j^T x\end{aligned}$$

where the summation is over all the columns of Q. In particular we have the following recursion:

$$\mathsf{Orthogonal}(x, Q_j) = \mathsf{Orthogonal}(x, Q_{j-1}) - q_j q_j^T x$$

where $Q_j = (Q_{j-1}, q_j)$. We write $\mathsf{Orthonormal}(x, G)$ for the direction of $\mathsf{Orthogonal}(x, G)$. It is defined as follows:

$$\mathsf{Orthonormal}(x, G) = \frac{r}{|r|}, \ \ r = \mathsf{Orthogonal}(x, G)$$

2.2 The MP algorithm

The Matching Pursuit algorithm (MP) was proposed in [21]. It is applied with N=1, where the data matrix Y consists of a single vector y. The changes in Algorithm GREEDY are:

0. Normalize X: $x_i \leftarrow x_i/|x_i|$. Set $r_0 = y$.
1.1 for $i = 1, \ldots, n:$ $\ f(x_i) = |r_{j-1}^T x_i|$.
1.3 $a = x_{s_j}^T r_{j-1}, \ \ r_j = r_{j-1} - a x_{s_j}$.

Observe that the value of $f(x_i)$ is inversely related to the angle between x_i and the residual r_{j-1}. Thus, the selection of x_{s_j} in Step 1.2 is the dictionary column "most similar" to (having the smallest angle with) the residual vector r_{j-1}. Step 1.3 updates the residual by removing the projection on the selected vector x_{s_j} from the current residual vector.

Using all coordinates of x_i for computing the dot product in 1.1 gives $O(kmn)$ run time. When X is sparsely represented by z_x nonzero values the dot products can be computed more efficiently and the run time reduces to $O(kz_x)$. Since only the vectors y, r_j, x_j need to be kept in memory the amount of memory needed by the MP algorithm is $O(m)$.

2.3 The OMP algorithm

The Orthogonal Matching pursuit (OMP) is only slightly more expensive than the MP but it is expected to select a better set of dictionary columns. For theoretical analysis see, e.g., [6, 10, 11]. For applications in signal processing see, e.g., [1].

The OMP is applied with N=1, so that the data matrix Y consists of a single vector y. It uses the same selection criterion as the MP, but calculates the residual vector as the orthogonal projection error. As discussed in Section 2.1 this can be done efficiently by maintaining an orthogonal matrix Q whose columns form an orthonormal basis to the selected vectors. With this implementation the changes to Algorithm GREEDY are:

0. Normalize X: $x_i \leftarrow x_i/|x_i|$. Set $r_0 = y$.
1.1 for $i = 1, \ldots, n:$ $f(x_i) = |r_{j-1}^T x_i|$.
1.3 $q_j = \mathsf{Orthonormal}(x_{s_j}, Q_{j-1})$
$Q_j = (Q_{j-1}, q_j)$
$a = q_j^T r_{j-1}, r_j = r_{j-1} - a q_j$

The calculations in Step 1.1 dominate the run time of the algorithm so that it is essentially the same as the run time of the MP, namely $O(kmn)$. The OMP requires more memory to keep and maintain the $m \times k$ matrix Q, leading to $O(km)$ memory requirement.

2.4 The OLS algorithm

The Orthogonal Least Squares (OLS) algorithm is only slightly more expensive than the OMP but it is expected to select a better set of dictionary columns. As the MP and the OMP it is implemented for N=1 so that the data matrix Y consists of a single vector y. We use the name OLS following [10,28]. The algorithm has sometimes been called OMP (e.g. [29]), OOMP (e.g. [22]), and Forward Selection (e.g. [3]) among others. See [28] for additional details about the naming of the algorithm. For a theoretical analysis of the OLS see, e.g. [10, 17]. For applications to feature selection see, e.g. [2, 3]. For applications related to signal processing see, e.g. [4]. For applications related to computer vision see, e.g. [30]. For applications related to numerical linear algebra see, e.g. [17].

The main idea behind the OLS is to select at each iteration the vector that reduces the prediction error the most. Let e be the squared error of predicting y from the selection S, and let Q be an orthonormal basis for the column space of S. We have:

$$\begin{aligned} e &= \min_a |y - Sa|^2 = |\mathsf{Orthogonal}(y, S)|^2 \\ &= |y - QQ^T y|^2 = |y|^2 - |Q^T y|^2 \end{aligned}$$

This shows that minimizing e can be achieved by finding Q that maximizes $|Q^T y|$. The greedy search can be simplified by noticing the following recursion. Let Q_j denote the submatrix with the first j columns of Q. Then:

$$|Q_j^T y|^2 = |Q_{j-1}^T y|^2 + |q_j^T y|^2$$

where q_j is the last column of Q_j. This shows that the best greedy selection of x_{s_j} is the vector x_{s_j} satisfying the following criterion:

$$x_{s_j} = \arg\max_{x_i} |q_i^T y|, \quad q_i = \mathsf{Orthonormal}(x_i, Q_{j-1}) \tag{3}$$

When q_i is orthogonal to Q_{j-1} we have: $q_i^T y = q_i^T r_{j-1}$, where $r_{j-1} = \mathsf{Orthogonal}(y, Q_{j-1})$. Therefore the local optimality criterion (3) can also be expressed as:

$$\begin{aligned} x_{s_j} &= \arg\max_{x_i} |q_i^T r_{j-1}| \\ q_i &= \mathsf{Orthonormal}(x_i, Q_{j-1}) \\ r_{j-1} &= \mathsf{Orthogonal}(y, Q_{j-1}) \end{aligned} \tag{4}$$

Using either (3) or (4) requires partial orthogonalization of the entire matrix X. It is common to use the recursion specified in Section 2.1 and replace the vector x_i with its partial orthogonalization. A typical implementation of the OLS using the local optimality criterion (4) (e.g. [17]) can be achieved with the following changes to Algorithm GREEDY:

0. Set $r_0 = y$.
1.1 for $i = 1, \ldots, n$:
$x_i \leftarrow x_i - q_j q_j^T x_i$, $f(x_i) = |r_{j-1}^T x_i|/|x_i|$.
1.3 $q_j = \mathsf{Orthonormal}(x_{s_j}, Q_{j-1})$
$Q_j = (Q_{j-1}, q_j)$
$a = q_j^T r_{j-1}$, $r_j = r_{j-1} - a q_j$

The algorithm run time is dominated by Step 1.1, that takes about twice the time of the corresponding Step 1.1 of the OMP. What makes the OLS significantly more costly is the fact that the vectors x_i in Step 1.1 are not the original dictionary columns, since they have been partially orthogonalized with respect to the previously selected vectors. The partial orthogonalization results in loss of sparsity, and the need for additional memory to store the partially orthogonalized dictionary. This requires additional mn memory, with no run time savings for sparse dictionaries.

2.5 The SOMP algorithm

The Simultaneous Orthogonal Matching Pursuit algorithm (SOMP) is a direct generalization of the OMP for handling a data matrix of N columns [8]. The changes to Algorithm GREEDY are:

0. Normalize X: $x_i \leftarrow x_i/|x_i|$. Set $R_0 = Y$.
1.1 Let $r_{j,1}, \ldots, r_{j,N}$ be R_j columns.
for $i = 1, \ldots, n$: $f(x_i) = \sum_{t=1}^{N} |r_{j-1,t}^T x_i|$.
1.3 $q_j = \mathsf{Orthonormal}(x_{s_j}, Q_{j-1})$
$Q_j = (Q_{j-1}, q_j)$
for $t = 1, \ldots, N$:
$R_j = R_{j-1} - q_j q_j^T R_{j-1}$

As in the previous cases the calculations in Step 1.1 dominate the run time. They require knN dot products of m-vectors that take $O(kmnN)$. This can be reduced to $O(kz_x N)$ when the dictionary is sparse. The amount of memory used by the algorithm to keep the matrices Q and R is $km + mN$.

2.6 The SOLS algorithm

The Simultaneous Orthogonal Least Squares algorithm (SOLS) is a direct generalization of the OLS for handling a data matrix of N columns. The algorithm is analyzed in [7, 9, 27].

As in the case of the OLS the column selected from the dictionary is the one that reduces the prediction error the most. The following discussion mirrors the discussion in 2.4. Let e_i be the squared error of predicting y_i from the selection S, and let Q be an orthonormal basis for the column space of S. The global error E is defined by: $E = \sum_{i=1}^{N} e_i$. From the derivation in Section 2.4 we have:

$$e_i = |y_i|^2 - |Q^T y_i|^2$$

so that $E = \sum_i |y_i|^2 - \sum_i |Q^T y_i|^2$. Therefore the minimum of E can be achieved by maximizing $\sum_{i=1}^{N} |Q^T y_i|^2$. let Q_j be the submatrix of the first j columns in Q, and let q_j be the last column of

Q_j, so that $Q_j = (Q_{j-1}, q_j)$. We have:

$$\begin{aligned}\sum_{i=1}^{N} |Q_j^T y_i|^2 &= \sum_{t=1}^{j} q_t^T (\sum_i y_i y_i^T) q_t \\ &= \sum_{t=1}^{j} q_t^T Y Y^T q_t = \sum_{t=1}^{j} |Y^T q_j|^2 \\ &= \|Y^T Q_j\|_F^2 = \|Y^T Q_{j-1}\|_F^2 + |Y^T q_j|^2\end{aligned}$$

where the matrix norms are the Frobenius norms. This shows that the best greedy selection of x_{s_j} is the vector x_{s_j} satisfying the following local optimality criterion:

$$x_{s_j} = \arg\max_{x_i} |Y^T q_i|, \;\; q_i = \mathsf{Orthonormal}(x_i, Q_{j-1}) \tag{5}$$

Since q_i is orthogonal to Q_{j-1} we have: $Y^T q_i = R_{j-1}^T q_i$ where $R_{j-1} = Y - Q_{j-1} Q_{j-1}^T Y$. Therefore the local optimality criterion (5) can also be expressed as:

$$\begin{aligned}x_{s_j} &= \arg\max_{x_i} |R_{j-1}^T q_i| \\ q_i &= \mathsf{Orthonormal}(x_i, Q_{j-1}) \\ R_{j-1} &= Y - Q_{j-1} Q_{j-1}^T Y\end{aligned} \tag{6}$$

The implementation of the SOLS using the local optimality criterion (6) can be achieved with the following changes to Algorithm GREEDY:

0. Set $R_0 = Y$.
1.1 for $i = 1, \ldots, n$:
$x_i \leftarrow x_i - q_j q_j^T x_i$, $f(x_i) = |R_{j-1}^T x_i| / |x_i|$.
1.3 $q_j = \mathsf{Orthonormal}(x_{s_j}, Q_{j-1})$
$Q_j = (Q_{j-1}, q_j)$
$R_j = R_{j-1} - q_j q_j^T R_{j-1}$

As in previous cases the calculations in Step 1.1 dominate the run time. This step is visited kn times, and the matrix-vector product costs $O(mN)$. Therefore the run time is $O(kmnN)$. It cannot be reduced when the dictionary X is sparse since the partial orthogonalization of X typically destroys its sparsity.

2.7 The CM algorithm

Civril and Magdon-Ismail describe in [7] a fast implementation of the SOLS algorithm that we call *The CM Algorithm*. Their motivation was column subset selection, which they achieved as follows. Suppose the goal is to select k columns from a matrix A. Set A to be the dictionary matrix X, and construct the data matrix Y consisting of the k dominant singular vectors of A, scaled by the corresponding singular values. It is shown in [7] that with this setting running the SOLS gives a good solution to column subset selection, as discussed in Section 1.1. To obtain a fast algorithm the paper establishes the CM, an optimized SOLS algorithm.

The proposed optimization retains in memory the values of the dot products computed in Step 1.1 of the SOLS, so that they need only be updated after each iteration. There are nN dot products to be maintained, and computing their initial values has an associated cost of $O(mnN)$. The paper shows that these updates can be done fast, with an associated cost of $O(nN)$ per iteration. Thus, the updates cost a total of $O(knN)$. Adding the initialization cost and the updates cost gives an algorithm with run time of $O((k+m)nN)$. The sparsity of X can be used during the initialization, but not during the updates where X is partially orthogonalized. This reduces the run time to $O(knN + z_x N)$ for a sparse X.

Examining Fig. 3 we see that the CM improves the run time by a factor of k over the naive SOLS. The improvement for a sparse X is even more impressive. When $z_x \leq kn$ the run time of the CM is $O(knN)$, better than the naive SOLS by a factor of m.

2.8 Our results

As mentioned in Section 1 the algorithms we describe are based on formulas that allow recursive efficient calculations of the selection criterion. The algorithms differ by how they evaluate the initial values that are needed to select the first dictionary column. The exact version of the algorithm that we call "ISOLS-exact" produces the same output as the SOLS and the CM. It runs in time $O(mnN)$ on dense dictionaries, and in $O(kmN + z_x N)$ on sparse dictionaries. The algorithm keeps two values for each column of the dictionary in addition to the matrix Q, that is of size $m \times k$.

The second version of the algorithm that we call "ISOLS-lowrank" approximates the data matrix and produces results that may not be identical to the "ISOLS-exact". But it has a much faster run time.

Comparing these algorithms to the CM we observe the following.

- The memory requirements of the CM are heavier than the memory requirements of our algorithms.
- Assuming the same low rank approximation for the data matrix by the CM, that can reduce the term N to d, we consider the case where $z_x \approx n$. The complexity of the CM is roughly $O(dkn)$, which is worse than the $O(dn)$ of the ISOLS-lowrank by a factor of k.

3 Recursive SOLS criterion

Our main result is described in this section. We analyze the selection criterion in Line 1.1 of the SOLS algorithm and show that it can be calculated recursively.

The matrix $Q_{j-1} = (q_1, \dots q_{j-1})$ is known at Line 1.1 of the SOLS. The vector q_i^j is computed for a candidate vector x_i at Iteration-j by:

$$r_j^i = x_i - Q_{j-1}Q_{j-1}^T x_i, \quad q_j^i = \frac{r_j^i}{|r_j^i|} \tag{7}$$

Applying the criterion in (5) the selection is computed by:

$$\begin{aligned} x_{s_j} &= \arg\max_{x_i} |Y^T q_j^i|^2 \\ &= \arg\max_{x_i} \frac{|Y^T r_j^i|^2}{|r_j^i|^2} = \arg\max_{x_i} \frac{u_j^i}{v_j^i} \end{aligned} \tag{8}$$

where $u_j^i = |Y^T r_j^i|^2, \quad v_j^i = |r_j^i|^2$. We proceed to show that both u_j^i and v_j^i can be computed recursively. The recursion for v_j^i is well known, but to the best of our knowledge the recursion for u_j^i is new. We state the result as a theorem.

Theorem. Let Y be $m \times N$, let Q_j be $m \times j$ with orthonormal columns, and let q_j be the last column of Q_j, so that $Q_j = (Q_{j-1}, q_j)$. For any vector x define:

$$\begin{aligned} r_j &= x - Q_{j-1}Q_{j-1}^T x \\ u_j &= |Y r_j|^2, \quad v_j = |r_j|^2, \quad f_j = \frac{u_j}{v_j} \end{aligned}$$

If u_j and v_j are known then f_{j+1} can be computed by steps 1-5 described below, where Step 1 is independent of x, and steps 2-5 are independent of N.

1. $c_j = YY^T q_j, \quad d_j = c_j - Q_{j-1}Q_{j-1}^T c_j, \quad \beta_j = q_j^T c_j$
2. $\alpha_j = q_j^T x, \quad \gamma_j = d_j^T x$
3. $u_{j+1} = u_j + \alpha_j^2 \beta_j - 2\alpha_j \gamma_j$
4. $v_{j+1} = v_j - \alpha_j^2$
5. $f_{j+1} = \dfrac{u_{j+1}}{v_{j+1}}$

Proof. The value of r_j as defined in (7) satisfies:

$$r_{j+1} = x - Q_j Q_j^T x = r_j - q_j q_j^T x$$

This gives the following fundamental recursion:

$$r_{j+1} = r_j - \alpha_j q_j \tag{9}$$

Observe that q_j and r_{j+1} are orthogonal. Writing (9) as $r_{j+1} + \alpha_j q_j = r_j$ and taking squared norms of both sides gives: $|r_{j+1}|^2 + \alpha_j^2 = |r_j|^2$. This proves the recursion on line 4. above. This recursion is well known. It was previously used, for example, in [7, 31].

We proceed to derive the recursion for u_j, given by the formula in line 3., that to the best of our knowledge is new. From the fundamental recursion (9):

$$Y^T r_{j+1} = Y^T r_j - \alpha_j Y^T q_j$$

Taking norms:

$$\begin{aligned} u_{j+1} &= |Y^T r_{j+1}|^2 = |Y^T r_j - \alpha_j Y^T q_j|^2 \\ &= u_j + \alpha_j^2 \beta_j - 2\alpha_j c_j^T r_j \end{aligned}$$

It remains to show that $c_j^T r_j = d_j^T x$, and this follows by direct observation. □

3.1 Complexity

Step 1. The computation of c_j as specified in Step 1 in the theorem can be done in one pass as follows:

$$c_j = \sum_{i=1}^{N} y_i (y_i^T q_j)$$

The cost is roughly $4mN$ flops. Similarly, the cost of calculating d_j is roughly $5(j-1)m$, and the cost of calculating β_j is $2m$. We observe that summing it up for $j = 1, \ldots, k$ gives roughly $4km(N+1)$ flops.

Step 2. The cost of the two dot products is roughly $4m$ flops.

Steps 3,4,5. These steps take roughly 8 flops.

4 The ISOLS-exact algorithm

In this section we describe the ISOLS-exact, a straightforward implementation of the theorem formulas. In order to apply the recursion in the theorem we need the values of u_0, v_0 for all columns of X. The ISOLS-exact computes these values, then selects the first column, and then applies the recursion to select the rest of the columns. The detailed algorithm is shown in Fig.4 The values needed for selecting the first column of X are calculated in Step 0.1, and the initial best selection is identified in Step 0.2. The remaining k-1 selections are determined by the k-1 iterations of Step 1.

4.1 Complexity

There are three distinct components that affect the run time: the global initialization in Step 0.1, the initial calculation in each iteration, performed in Step 1.0, and the two dot products in Step 1.1.

Dense dictionary. The expensive computation in Step 0.1 is the calculation of u_0^i that takes roughly $2mnN$ flops. The cost of Step 1.0 is dominated by the evaluation of c_j. Taking into account the evaluation of both c_j and d_j gives roughly $4(k-1)m(N+1)$ flops for the run time of Step 0.1. The cost of Step 1.1 is dominated by the calculation of α_j^i, γ_j^i which is roughly $4mn$ flops. This gives an asymptotic run time of $O(mnN)$, dominated by the global initialization.

Sparse dictionary. The number of flops needed to perform a dot product between a non-sparse and a sparse vector is the number of nonzero entries in the sparse vector. Therefore, when X is sparse

Input: X, Y, k. Output: S containing k atoms.

0. Initial steps

0.1 for $i = 1, \ldots, n$:
$u_0^i = |Y^T x_i|^2, \;\; v_0^i = |x_i|^2, \;\; f_0^i = u_0^i / v_0^i$

0.2 $i^* = \arg\max_i f_0^i, \; x_{s_1} = x_{i^*}$.

0.3 $S = \{x_{s_1}\}, \; q_1 = x_{s_1}/|x_{s_1}|, \; Q_0 = (\,), \; Q_1 = (q_1)$.

1. Run iterations for $j = 1, \ldots, k-1$:

1.0 $c_j = YY^T q_j, \; d_j = c_j - Q_{j-1} Q_{j-1}^T c_j, \; \beta_j = q_j^T c_j$.

1.1 for $i = 1, \ldots, n$:
$\alpha_j^i = q_j^T x_i, \; \gamma_j^i = d_j^T x_i$.
$u_{j+1}^i = u_j^i + (\alpha_j^i)^2 \beta_j^i - 2\alpha_j^i \gamma_j^i, \; v_{j+1}^i = v_j^i - (\alpha_j^i)^2$
$f_{j+1}^i = u_{j+1}^i / v_{j+1}^i$

1.2 $i^* = \arg\max_i f_{j+1}^i, \; x_{s_{j+1}} = x_{i^*}$.

1.3 Add $x_{s_{j+1}}$ to S
$q_{j+1} = \mathsf{Orthonormal}(x_{s_{j+1}}, Q_j)$
$Q_{j+1} = (Q_j, q_{j+1})$

Figure 4: The ISOLS-exact algorithm

the complexity of Step 0.1 is reduced to $O(z_x N)$, where z_x is the number of non-zeros in the matrix X. Similarly, the complexity of Step 1.1 is reduced to $4z_x$. But there is no change in the complexity of Step 1.0 since it is independent of X. This shows that the run time of Step 1.1 cannot be ignored since km may be larger than z_x. The asymptotic run time including both Step 0.1 and Step 1.0 is $O(z_x N + kmN)$.

Memory. The algorithm requires a temporary storage of km for the matrix Q, and a temporary storage of $2n$ for keeping the values u_i, v_i for each column of the dictionary X.

5 The ISOLS-lowrank algorithm

The analysis in Section 4 shows that the asymptotic run time of the ISOLS-exact is proportional to N, the number of columns of the data matrix Y. Observe that Y appears only in steps 0.1 and 1.0 of the ISOLS-exact, and that in both cases the computation depends only on YY^T. This suggests an approximate algorithm where Y is replaced by the matrix H that has fewer columns, but can still approximate Y in the sense that $HH^T x \approx YY^T x$ for an arbitrary vector x. More specifically we need HH^T to approximate YY^T in the spectral norm. The resulting algorithm is described below as a modification of the ISOLS-exact. The changes needed in Algorithm ISOLS-exact are specified blow.

Input: X, Y, k, d. Output: S.

0.0 Compute an $m \times d$ matrix H such that $HH^T \approx YY^T$.

0.1 Compute u_0^i by: $u_0^i = |H^T x_i|^2$

1.0 Compute c_j by: $c_j = HH^T q_j$.

Modifications to the ISOLS-exact algorithm that give the ISOLS-lowrank algorithm

5.1 Complexity

There are four distinct parts that affect the run time of the ISOLS-lowrank algorithm: Step 0.0, where the low rank approximation is computed, and steps 0.1,1.0,1.1, that were discussed in Section 4.1. We write $T_{\text{LR}}(Y)$ for the run time of computing the low rank approximation of Y. It is discussed in detail in Section 5.2.

Dense dictionary. It is clear from the analysis in Section 4.1 that Step 0.1 dominates steps 1.0 and 1.1 with asymptotic complexity of $O(dmn)$. This gives asymptotic complexity of $O(dmn + T_{\text{LR}}(Y))$ for running the ISOLS-lowrank with a dense dictionary.

Sparse dictionary. Using same arguments as in Section 4.1 the asymptotic run time with sparse dictionaries is $O(d(km + z_x) + T_{\text{LR}}(Y))$.

Memory. The algorithm requires storage of km for the matrix Q, and $2n$ for keeping the values u_i, v_i for each column of the dictionary X.

5.2 Low rank approximations to the data matrix

The ISOLS-lowrank replaces the $m \times N$ data matrix Y with the $m \times d$ matrix H. The matrix H approximates the matrix Y in the sense that $\|HH^T - YY^T\|_2$ is small. It is known that the best low rank approximation can be obtained from the truncated SVD of Y. Suppose the SVD of Y is given by: $Y = U\Sigma V^T$, then $YY^T = U\Sigma^2 U^T$, and the best possible HH^T is $\sum_{j=1}^{d} \sigma_j^2 u_j u_j^T$, where the vectors u_j are the columns of the orthogonal matrix U. This is clearly satisfied if the matrix H is chosen to be:

$$H = (h_1, \ldots, h_d), \quad h_j = \sigma_j u_j \quad \text{for } j = 1, \ldots, d$$

This shows that the matrix H can be calculated from the truncated SVD of the matrix Y. Classical iterative approaches for calculating the truncated SVD have run time of $O(dmN)$ (e.g., [23]).

Recently proposed algorithms that use randomization can compute the SVD much faster. See, e.g., [13, 14, 15], and the experimental evaluation in [16]. Since our goal is computing the matrix H and not the SVD, we observe that in some of these algorithms it is not necessary to pass through the SVD, as the desired matrix H is computed in early or intermediate stages of the algorithm. We illustrate this with the algorithm of [13].

To apply the algorithm of [13] for computing H we need the squared norms of all the columns of Y. This is sometimes known in advance, or can be calculated in one pass at the cost of roughly $2mN$. Given these norms, the algorithm selects d columns at random, where the probability of selecting the column y_i is proportional to $|y_i|^2$. Without loss of generality suppose the selection is $y_1, \ldots, y_d$. The desired matrix H is formed by:

$$H = (h_1, \ldots, h_d), \quad h_j = y_j/|y_j|^2 \quad \text{for } j = 1, \ldots, d$$

To obtain the same accuracy as with the SVD method, one would need a larger value of d. Still, it is shown in [13] that with high probability the increase in the value of d is only by a factor of $O(\log N)$.

6 Experimental results

In this section we describe experimental results with the proposed algorithms. Our goal was to evaluate the run time, stability, and accuracy of the algorithms. The relevant parameters for run time are the amount of speedup for sparse dictionaries, and the amount of speedup gained by reducing the rank of the data matrix. By stability we refer to a known problem with algorithms that require orthogonalization of large matrices. The loss of orthogonality among the vectors may lead to the selection of redundant columns not contributing to the reduction of the error (2). The accuracy is related to the performance of the ISOLS-lowrank as compared with that of the ISOLS-exact.

6.1 Some details about the implementation

In our implementation the low rank approximation of the data matrix Y was computed using the technique of [15]. The idea is to compute a low rank orthonormal matrix Q of dimensions $m \times d$ that approximately spans the column space of Y. Using Q one can compute the $d \times d$ matrix W=$Q^T Y Y^T Q$ in one pass over the data. Let W=SS^T be the Cholesky decomposition of W, and set H^T=$S^T Q^T$, where H is $m \times d$. It is easy to show (see [15]) that if Q is a good in the sense of $QQ^T Y \approx Y$ then H is good in the sense of $HH^T \approx YY^T$.

Instead of keeping the matrix Q explicitly we use the well known technique of Householder matrices [23] to indirectly represent it. Observe that the matrix Q is used in steps 1.0 and 1.3 of the algorithm. Both steps can be implemented with the Householder representation, so that only the calculations involving the vector q_j need to be performed outside of the Householder representation.

6.2 The experiments

We conducted two experiments with each dataset. Each experiment involved 7 runs, as explained below.

Experiment 1. In this experiment we took both the data matrix Y and the dictionary matrix X to be the same as the given dataset. This is the classical column subset selection problem, as discussed in Section 1. We ran this experiment with k in the range of $1 - 100$, and for various values of d, the reduced rank of the data matrix. In particular the following values of d were used: $\{1, 5, 10, 20, 50, 75, 100\}$.

Experiment 2. The data matrix Y was taken as identical to the dataset, and the value of d=50 was used as the rank of H in ISOLS-lowrank. The dictionary matrix X was created by randomly zeroing out entries in the dataset so that the sparsity of the dictionary was increased. In particular, for p in the range of $\{1, 5, 10, 20, 50, 75, 100\}$ we zeroed out entries with probability of $p/100$. Thus, p is the percentage of retained values. Observe that $p = 100$ is a special case of Experiment 1 with d=50. But for smaller values of p the data matrix and the dictionary matrix are distinct.

Reported measurements. In each experiment we measured the run time and the approximation error as a function of d and p. In the case of ISOLS-lowrank the reported run time does not include the time taken to compute the low rank approximation to Y. Clearly, the measurements of run time depend on the hardware, and may be influenced by background processes. But we believe that the rate of change that was observed is properly reflecting the run time dependency on the above parameters.

The approximation error was calculated according to Equation (2), and normalized so that the highest possible value (the squared norm of Y) is 100. Thus, the reported approximation error values are the percentage of the error as given by Equation (2).

6.2.1 Experiments with the TechTC01 dataset

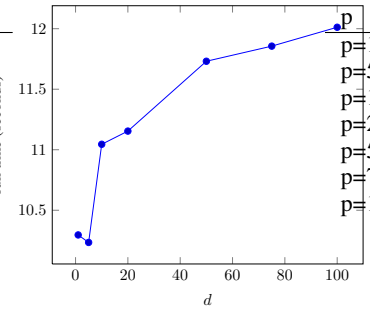
This dataset is part of the Technion repository of text categorization datasets. It has m=163 rows and N=29,261 columns. The results of the experiments are summarized in Fig.5.

Running the ISOLS-exact for selecting 100 dictionary columns took 774 seconds. Using ISOLS-lowrank with d=100 took 12 seconds, an improvement by a factor of 64. In the asymptotic limit the run time is expected to be linearly related to d. The reported results show a reduction in time for smaller d, but not linear dependency. (For example, the run time for $d = 100$ is not 10 times the run time for $d = 10$.) This was observed with other datasets as well, and indicates that the computation overhead is quite significant.

The dependency on p, the dictionary sparsity, is exactly as expected, with the run time going down as the value of p is decreased.

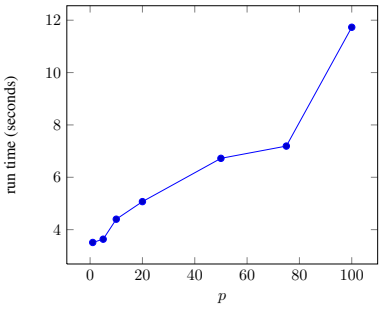
The table and plots show the error of predicting the entire data matrix from selected dictionary columns. Because of the rapid decrease in the error we describe the results of each experiment with two plots. The first plot (on the left) shows the results for all tested values of k, while the second

d	time
d=1	10.296
d=5	10.234
d=10	11.045
d=20	11.154
d=50	11.731
d=75	11.856
d=100	12.012
exact	774.340



Run time as a function of H rank in ISOLS-approx

p	time
p=1	3.510
p=5	3.635
p=10	4.399
p=20	5.070
p=50	6.724
p=75	7.192
p=100	11.731



Run time as a function of the dictionary sparsity (d=50)

	k=10	k=20	k=30	k=40	k=50	k=60	k=70	k=80	k=90	k=100
d=50 p=1	33.394	20.242	13.005	8.945	6.008	3.964	2.500	1.767	1.315	0.974
d=50 p=50	21.241	10.995	7.270	4.992	3.392	2.290	1.478	1.015	0.725	0.439
d=1	19.088	12.192	8.557	7.061	5.947	4.802	4.157	3.586	3.167	2.781
d=50	10.957	6.137	3.962	2.605	1.758	1.208	0.825	0.578	0.391	0.267
d=100	10.957	6.137	3.952	2.608	1.752	1.151	0.720	0.445	0.288	0.189
exact	10.957	6.137	3.952	2.608	1.752	1.151	0.720	0.445	0.288	0.188

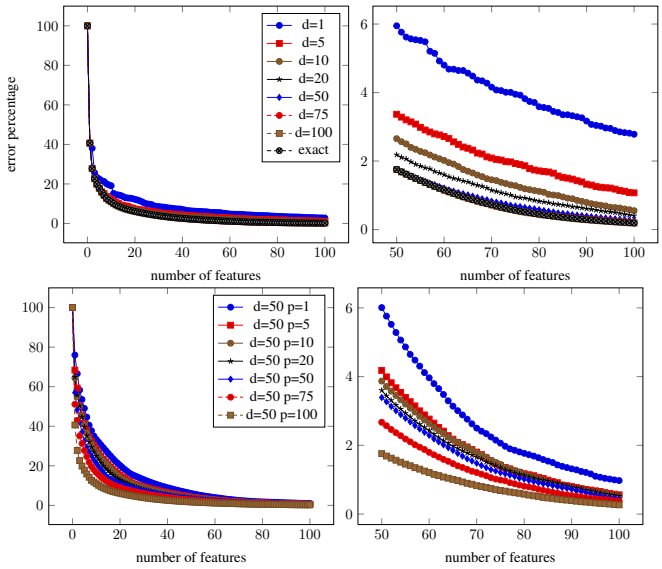
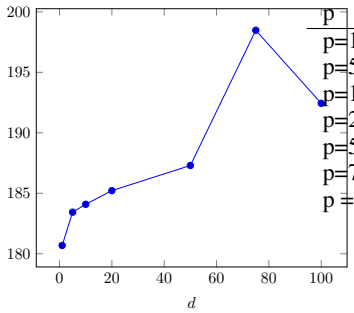


Figure 5: Experiments with the TechTC01 dataset. The top part shows time dependency on d, the degree of H in ISOLS-lowrank, and on p, the sparsity percentage. The table and the plots show how the various parameters affect the accuracy. The error is specified as percentage of the initial error.

plot shows the results only for k in the range $50 - 100$. Observe that there is very little difference between the error of the columns selected by ISOLS-exact and those computed by ISOLS-lowrank . Even in the extreme case where the entire data matrix is approximated by a single vector, the IOLS-lowrank finds 100 columns that reduce the prediction error to less than 3%.

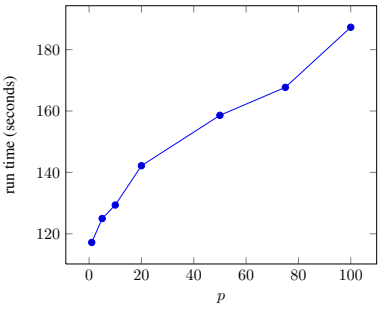
The experiments with very sparse dictionary show a decrease in the prediction accuracy, but the decrease is much less than what we expected. We find it surprising that a small fraction of the data can produce such accurate prediction of the entire data matrix. These observations are specific to the TechTC01 dataset. They did not show up with other datasets.

d	time
d=1	180.690
d=5	183.440
d=10	184.090
d=20	185.220
d=50	187.300
d=75	198.480
d=100	192.440



Run time as a function of H rank in ISOLS-approx

p	time
p=1	117.170
p=5	124.960
p=10	129.370
p=20	142.170
p=50	158.600
p=75	167.710
p = 100	187.300



Run time as a function of the dictionary sparsity (d=50)

	k=10	k=20	k=30	k=40	k=50	k=60	k=70	k=80	k=90	k=100
d=50 p=1	93.410	88.290	84.110	80.649	77.740	75.297	73.310	71.818	70.743	70.039
d=50 p=50	39.417	34.450	31.870	30.070	28.680	27.543	26.633	25.860	25.190	24.594
d=1	27.520	23.870	22.282	20.491	19.128	18.321	17.746	17.074	16.456	16.098
d=50	24.887	21.780	19.792	18.447	17.400	16.569	15.906	15.354	14.907	14.493
d=100	24.887	21.777	19.787	18.441	17.365	16.533	15.840	15.264	14.738	14.288

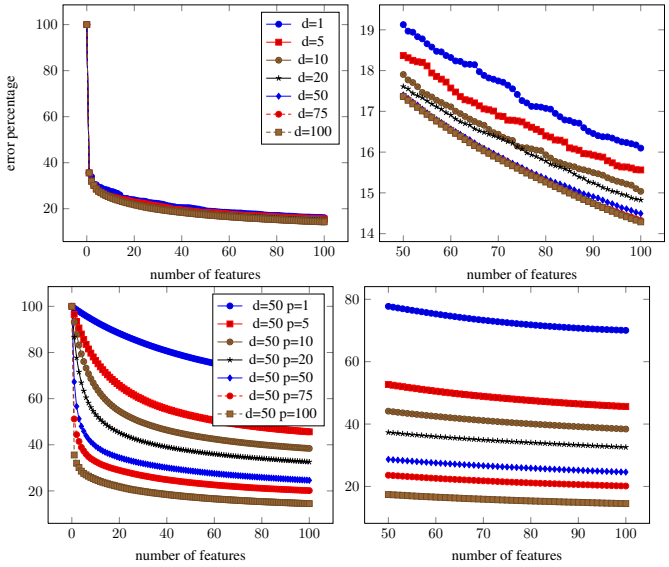


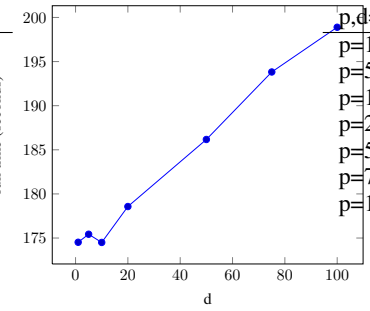
Figure 6: Experiments with the Day1 dataset. The top part shows time dependency on d, the degree of H in ISOLS-lowrank, and on p, the sparsity percentage. The table and the plots show how the various parameters affect the accuracy. The error is specified as percentage of the initial error.

6.2.2 Experiments with the Day1 dataset

This dataset is part of the "URL_reputation" collection at the UCI Repository. It has m=20, 000 rows and N=3, 231, 957 columns. The data is very sparse. It occupies about 40 megabytes of memory, while a non-sparse representation would require about 200 gigabytes. IOLS-exact ran too slow on this dataset and all results were obtained with the IOLS-lowrank.

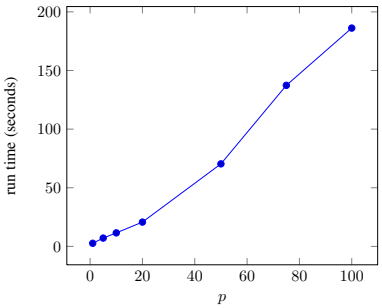
We ran the same set of experiments with this dataset as those applied to the TechTC01 dataset. The results are very similar. They are summarized in Fig.6. The main difference is that the prediction error did not go down as rapidly as in the TechTC01 case.

d	time
d=1	174.520
d=5	175.430
d=10	174.500
d=20	178.570
d=50	186.170
d=75	193.820
d=100	198.890



Run time as a function of H rank in ISOLS-approx

p,d=50	time
p=1	2.670
p=5	7.100
p=10	11.520
p=20	20.730
p=50	70.370
p=75	137.340
p=100	186.170



Run time as a function of the dictionary sparsity (d=50)

	k=10	k=20	k=30	k=40	k=50	k=60	k=70	k=80	k=90	k=100
d=50 p=1	97.210	95.096	93.336	91.760	90.320	88.996	87.804	86.709	85.717	84.800
d=50 p=50	71.130	66.640	63.809	61.839	60.328	59.080	58.019	57.088	56.280	55.550
d=1	63.333	57.782	54.667	52.442	50.958	49.915	48.793	48.042	47.227	46.514
d=50	61.555	56.298	53.349	51.319	49.774	48.575	47.539	46.707	45.967	45.310
d=100	61.555	56.298	53.337	51.301	49.742	48.520	47.527	46.698	45.971	45.311

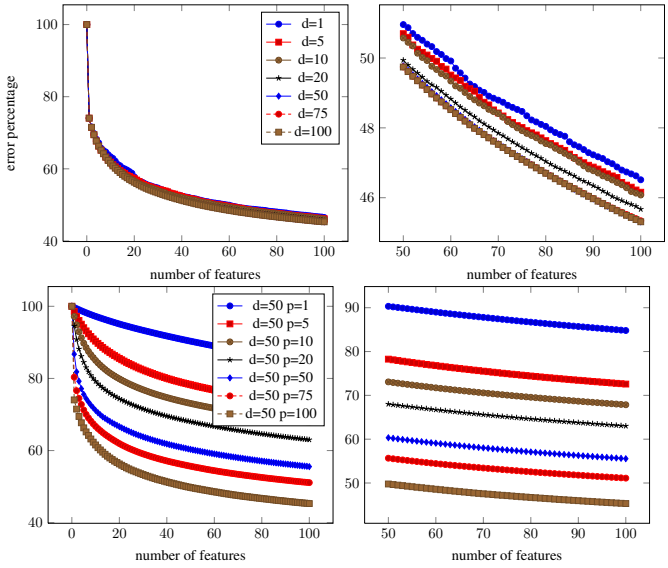


Figure 7: Experiments with the gisette dataset. The top part shows time dependency on d, the degree of H in ISOLS-lowrank, and on p, the sparsity percentage. The table and the plots show how the various parameters affect the accuracy. The error is specified as percentage of the initial error.

6.2.3 Experiments with the gisette dataset

This dataset was used in the NIPS 2003 selection challenge. It has m=6,000 rows and N=5,000 columns. We ran the same set of experiments as with the previous two datasets, and the results are summarized in Fig.7. IOLS-exact ran too slow on this dataset and all results were obtained with the IOLS-lowrank. The results are similar to what was observed with the other datasets.

6.2.4 Summary

The experimental results mostly agree with theoretical analysis. The run time is improved with the reduction in d and with the increase in sparsity. We observed that the dependency on d is

monotone but not linear. Another unexpected observation is the level of prediction accuracy that can be achieved when the ISOLS-lowrank uses small values of d.

We did not observe any stability issues. The prediction error kept improving with additionally selected features, indicating that no redundant features were selected.

7 Concluding remarks

The problem that was discussed in this paper, approximating one matrix in terms some columns of another matrix, is well known and appears to have many practical applications. The majority of previous studies address the accuracy of the proposed scheme, but not the resources that are needed to apply it to large matrices. We consider the CM algorithm of Civril and Magdon-Ismail, established in [7], to be the current state-of-the-art. It reduces the $O(kmnN)$ run time of the naive approach to $O(mnN)$, and further to $O(knN)$ in the sparse case. Using a low rank approximation to the data matrix the run time of the CM can be further reduced to $O(dkn)$. On the Other hand, the CM does not improve on the memory requirements of the naive approach. Both require $O(mn)$ memory to store a partially orthogonalized dictionary matrix, and the CM requires an additional storage of nN.

The algorithms presented in this paper use an entirely different approach than the CM. They have the same asymptotic complexity for the non-sparse case, and a better asymptotic complexity of $O(nN)$ for the sparse case. Using a low rank approximation to the data matrix the run time is further reduces to $O(dn)$. In addition to the run time our algorithms also require a significantly smaller amount of memory, roughly $km + 2n$ floats.

To illustrate the difference consider the specific example of the Day1 dataset of Section 6.2.2. In our experiments it took less than 4 minutes to select 100 columns, and the algorithm used less than 150 megabytes of memory. In this example $k = 100$, $d = 100$, m=20,000, N=3,231,957, N=3,231,957. The naive method would be roughly $mN \approx 610^{10}$ time slower, taking more than 27 thousand years. It would also require roughly 200 gigabytes of memory. Using the CM the run time would be only 100 times slower than in our approach, taking roughly 7 hours, but it would still require around 200 gigabytes of memory.

References

[1] S. Mallat, *A wavelet Tour of Signal Processing*. Academic Press, 1999.

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Springer, 2009.

[3] A. Miller, *Subset Selection in Regression*, 2nd ed. Chapman & Hall/CRC, 2002.

[4] S. Chen, A. Billings, and W. Luo, "Orthogonal least squares methods and their application to non-linear system identification," *International Journal of Control*, vol. 50, no. 5, pp. 1873–1896, 1989.

[5] A. R. B. A. C. W. Dahmen and R. A. DeVore, "Approximation and learning by greedy algorithms," *Annals of Statistics*, vol. 36, no. 1, pp. 64–94, 2008.

[6] J. A. Tropp, "Greed is good: algorithmic results for sparse approximation," *IEEE Transactions on Information Theory*, vol. 50, no. 10, pp. 2231–2242, 2004.

[7] A. Çivril and M. Magdon-Ismail, "Column subset selection via sparse approximation of SVD," *Theoretical Computer Science*, vol. 421, pp. 1–14, Mar. 2012.

[8] J. A. Tropp, A. C. Gilbert, and M. J. Strauss, "Algorithms for simultaneous sparse approximation. part I: Greedy pursuit," *Signal Processing*, vol. 86, no. 3, pp. 572–588, 2006.

[9] J. Chen and X. Huo, "Theoretical results of sparse representations of multiple measurement vectors," *IEEE Transactions on Signal processing*, vol. 54, no. 12, pp. 4634–4643, 2006.

[10] C. Soussen, R. Gribonval, J. Idier, and C. Herzet, "Joint k-step analysis of orthogonal matching pursuit and orthogonal least squares," *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 3158–3174, 2013.

[11] A. Joseph, "Variable selection in high-dimension with random designs and orthogonal matching pursuit," *Journal of Machine Learning Research*, vol. 14, pp. 1771–1800, 2013. [Online]. Available: http://jmlr.org/papers/v14/joseph13a.html

[12] D. L. Donoho, Y. Tsaig, I. Drori, and J. L. Starck, "Sparse solutions of undetermined systems of linear equations by stagewise orthogonal matching pursuit," *IEEE Transactions on Information Theory*, vol. 58, no. 2, pp. 1094–1896, 2012.

[13] A. M. Frieze, R. Kannan, and S. Vempala, "Fast monte-carlo algorithms for finding low-rank approximations," *Journal of the ACM*, vol. 51, no. 6, pp. 1025–1041, 2004.

[14] P. Drineas, R. Kannan, and M. W. Mahoney, "Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix," *SIAM Journal on Computing*, vol. 36, no. 1, pp. 158–183, 2006.

[15] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Review*, vol. 53, no. 2, pp. 217–288, 2011.

[16] A. K. Menon and C. Elkan, "Fast algorithms for approximating the singular value decomposition," *ACM Transactions on Knowledge Discovery from Data*, vol. 5, no. 2, p. 13, 2011, experimental evaluation of fast svd algorithms.

[17] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM Journal of Computing*, vol. 25, no. 2, pp. 227–234, 1995.

[18] E. Amaldi and V. Kann, "On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems," *Theoretical Computer Science*, vol. 209, no. 1–2, pp. 237–260, Dec. 1998.

[19] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust principal component analysis?" *Journal of the ACM*, vol. 58, no. 3, pp. 11:1–11:37, May 2011.

[20] J. A. Tropp, "Algorithms for simultaneous sparse approximation. part II: Convex relaxation," *Signal Processing*, vol. 86, no. 3, pp. 589–602, 2006.

[21] S. Mallat and Z. Zhang, "Matching pursuit in a time-frequency dictionary," *IEEE Transactions on Signal Processing*, vol. 41, pp. 3397–3415, 1993.

[22] L. Rebollo-Neira and D. Lowe, "Optimized orthogonal matching pursuit approach," *Signal Processing Letters, IEEE*, vol. 9, no. 4, pp. 137–140, 2002.

[23] G. H. Golub and C. F. Van-Loan, *Matrix computations*, 3rd ed. The Johns Hopkins University Press, 1996.

[24] A. Çivril and M. Magdon-Ismail, "On selecting a maximum volume sub-matrix of a matrix and related problems," *Theoretical Computer Science*, vol. 410, no. 47-49, pp. 4801–4811, 2009.

[25] M. Gu and S. C. Eisenstat, "Efficient algorithms for computing a strong rank-revealing QR factorization," *SIAM J. Computing*, vol. 17, no. 4, pp. 848–869, 1996.

[26] C. Boutsidis, M. W. Mahoney, and P. Drineas, "An improved approximation algorithm for the column subset selection problem," in *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, C. Mathieu, Ed. SIAM, 2009, pp. 968–977.

[27] S. F. Cotter, B. D. Rao, K. Engen, and K. Kreutz-Delgado, "Sparse solutions to linear inverse problems with multiple measurement vectors," *ASP*, vol. 53, no. 7, pp. 2477–2488, 2005.

[28] T. Blumensath and M. E. Davies, "On the difference between orthonormal matching pursuit and orthogonal least squares," University of Edinburgh, Technical Report, Mar. 2007.

[29] a. T. S. H. M. Gharavi-Alkhansari, "A fast orthogonal matching pursuit algorithm," in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. 3, 1998, pp. 1389–1392 vol.3.

[30] F. Tang, R. Crabb, and H. Tao, "Representing images using nonorthogonal haar-like bases," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 12, pp. 2120–2134, Dec. 2007.

[31] P. Businger and G. H. Golub, "Linear least squares solutions by Householder transformations," *Numer. Math.*, vol. 7, pp. 269–276, 1965.