1 Introduction

In this paper, we describe and analyze a new fast incremental algorithm for tracking, over time, a solution to problem P_0 (defined below), which offers a natural generalization of correlated pairs to subsets of larger size.

Given a set of time series $a_1, \ldots, a_m$, it is often desirable to find the most highly correlated pairs among them. If the data is properly normalized then this problem is equivalent to finding pairs (i, j) so that

$$||a_i - a_j|| < \epsilon,$$

where ϵ is our threshold parameter (smaller ϵ require that a_i, a_j are more highly correlated).

Let's consider our time series as vectors forming the columns of matrix A; then $Ax = a_i - a_j$ where $x = e_i - e_j$ and e_i is the i^{th} column of the identity matrix. Hence finding highly correlated pairs may be considered as finding certain x such that

$$||Ax|| < \epsilon \quad \text{and} \quad ||x||_0 = 2,$$

where $||x||_0$ denotes the number of nonzero coordinates of vector x.

Here, we will focus on the problem of finding vectors x so that

$$||Ax|| < \epsilon \quad \text{and} \quad ||x||_0 \text{ is small.}$$

In particular, we will build on the work of David Donoho et al. [1, 2, 3] by using problem $\mathsf{P}_0(\tilde{A}, b)$:

$$\begin{cases} \min & ||x||_0 \\ s.t. & \tilde{A}x = b. \end{cases}$$

Equality $\tilde{A}x = b$ is essentially equivalent to $Ax = 0$ if we remove a column b of A to arrive at $\tilde{A}$; then there is an easy correspondence between solutions to $Ax = 0$ and $\tilde{A}x = b$. Problem P_0 involves an exact equality $\tilde{A}x = b$, but we would like to allow for some error tolerance ϵ in this identity. As will be seen below, we will accomplish this tolerance using a dimension reduction idea which we call the ϵ-space of a matrix.

This paper is organized as follows: first, in §2, we initially describe and provide pseudocode for the algorithm. Next, in §3, we explore why the algorithm works and discuss its time complexity. Finally, in §4, we present experimental results.

2 The algorithm

Here we present an algorithm which tracks over time a sparse vector x so that $\tilde{A}x = b$, where $\tilde{A}$ is our sliding window of time series data, and b is a particular target time series. For example, we may set b as a time-delayed stock price for a particular symbol, and choose $\tilde{A}$ as a set of other stock prices. Then any

solution to $\tilde{A}x = b$ in effect is predicting the value of b using a small number of symbols (time series) in $\tilde{A}$.

In the pseudocode, it will be easier to replace the matrix/column pair $(\tilde{A}, b)$ by the single matrix A, in which our target vector (formerly b) is simply the i^{th} column a_i, for a particular i. If we write A_i for matrix A without column a_i, then we could state the objective of this algorithm as approximating, at each step, the solution to $\mathsf{P}_0(A_i, a_i)$.

The algorithm assumes that there is an $m \times n$ sliding window A, which is being updated to the new window $\hat{A}$ (in general, we will use the hat accent $\hat{X}$ to denote the incrementally-updated version of variable X). We assume that α is the new incoming data (as a row), and β is the old outgoing row. We summarize this by writing

$$\begin{pmatrix} A \\ \alpha \end{pmatrix} = \begin{pmatrix} \beta \\ \hat{A} \end{pmatrix}.$$

Our algorithm incrementally maintains matrices B and Q so that $B = A^T A$ is $n \times n$ and the rows of $k \times n$ matrix Q are the k eigenvectors of B with the largest eigenvalues. In the following pseudocode, the subroutine most_sig_eig$(\hat{B}, Q^T, k)$ returns the k most significant eigenvectors of $\hat{B}$ — since a converging iterative method could be used to implement this subroutine, most_sig_eig also accepts the input Q^T as a starting point for this convergence. We will see in the next section why tracking this matrix Q allows us to essentially apply a constraint of the form $A_i x \approx a_i$.

Incrementally Approximate $\mathsf{P}_0(A_i, a_i)$

// $m \times n$ matrix A is the old sliding window
// We have the input:
// α is the new row of data, β is the old row
// τ is the taper factor, $\tau \in (0, 1]$
// $n \times n$ matrix $B = A^T A$
// rows of $k \times n$ matrix Q are the k most significant eigenvectors of B
// Also, $\hat{Q}_i$ denotes matrix $\hat{Q}$ without its i^{th} column $\hat{q}_i$

Let $\hat{B} = \tau B + \alpha^T \alpha - \tau^m \beta^T \beta$
Compute $\hat{Q}^T =$ most_sig_eig$(\hat{B}, Q^T, k)$
Solve $\begin{cases} \min & ||x||_1 \\ \text{s.t.} & \hat{Q}_i x = \hat{q}_i \end{cases}$
Keep $(\hat{B}, \hat{Q}, x)$ for the next time step
Return x

We will see that the above algorithm is guaranteed to provide an upper bound on both the sparsity of x as well as the accuracy of the approximation $Ax \approx 0$ (although this latter bound is data-dependent).

3 Analysis of the algorithm

In the following sections, we'll see why this algorithm works and examine how quickly and accurately it operates.

3.1 Idea of the ϵ-space

We are all familiar with the idea of the *null space* of a matrix (or linear operator in general). The idea of this section is to extend this idea to a natural set of vectors v for which

$$\frac{||Av||}{||v||} < \epsilon \tag{1}$$

for some $\epsilon > 0$ which we may specify.

One might be tempted to consider the set

$$S = \{v : ||Av|| \leq \epsilon ||v||\}.$$

However, it is often the case that S is *not* a vector space. For example, if we let

$$A = \begin{pmatrix} -1 & 3 \\ -2 & 3 \end{pmatrix} \quad x = \begin{pmatrix} -1 \\ 0 \end{pmatrix} \quad y = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

then

$$\frac{||Ax||}{||x||} = \sqrt{5} < 3, \quad \frac{||Ay||}{||y||} = \sqrt{5/2} < 3, \text{ but } \frac{||A(x+y)||}{||x+y||} = 3\sqrt{2} > 3.$$

In other words, when $\epsilon = 3$ we have $x, y \in S$ but $x + y \notin S$.

In order to arrive at a vector space (which is of course closed under addition) that satisfies equation (1) for nonzero v, we will need to select a particular subset of S. The following choice seems natural:

Definition 1 *For any $\epsilon > 0$ and $m \times n$ matrix A, find the singular value decomposition $A = U\Sigma V^T$ for A; let σ_i denote the i^{th} singular value along the diagonal of Σ.*

Then we define the **ϵ-space** *N_ϵ of A as the vector space spanned by those columns of V corresponding to singular values $\sigma_i \leq \epsilon$.*

Notice that, by definition, the null space is always a subset of the ϵ-space.

Property 2 *If $v \in N_\epsilon$, the ϵ-space of A, then $||Av|| \leq \epsilon ||v||$.*

Proof.

It suffices to see that

$$||Ax|| \leq \epsilon \tag{2}$$

for all unit vectors $x \in N_\epsilon$, so this is what we will show.

First, we recall the details of the singular value decomposition for real matrices: $A = U\Sigma V^T$. Here, U and V are real unitary matrices (so $U^T = U^{-1}$,

or, equivalently, the columns of U form an orthonormal basis). Matrix Σ is diagonal, with entries $\sigma_1, \ldots, \sigma_n$ from upper-left to lower-right, with $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$.

If V is unitary, then $||Vx|| = ||\sum_i v_i x_i|| = \sqrt{\sum_i x_i^2} = ||x||$. Let $w = V^T x$. Notice that $V^{-1} = V^T$ is unitary iff V is unitary. Then clearly $||w|| = 1$ because $||x|| = 1$.

Also notice that, for $x \in N_\epsilon$ and w such that $x = Vw$, it must be case that $w_i = 0$ for any i with $\sigma_i > \epsilon$; otherwise, x would not be a linear combination of those vectors in V corresponding with $\sigma_i \leq \epsilon$, as stipulated by the definition of the ϵ-space. In other words, $w_i \neq 0 \implies \sigma_i \leq \epsilon$.

If $w = V^T x$, then $Ax = U\Sigma w = U(\sum_i \sigma_i w_i e_i) = \sum_i (\sigma_i w_i u_i)$. Therefore,

$$||Ax|| = \sqrt{\sum_i \sigma_i^2 w_i^2}. \tag{3}$$

We may now combine these observations to confirm equation (2) for an arbitrary unit vector $x \in N_\epsilon$. We still have $w = V^T x$. Then

$$||x|| = 1 \implies ||w|| = 1$$

and

$$(x \in N_\epsilon \;\&\; w_i \neq 0) \implies \sigma_i \leq \epsilon$$

together, along with (3), give us

$$||Ax|| \;=\; \sqrt{\sum_i \sigma_i^2 w_i^2} \;\leq\; \epsilon \sqrt{\sum_i w_i^2} \;=\; \epsilon,$$

which completes the proof. □

How does the idea of an $\epsilon-$space apply to our algorithm? Let us begin by defining the **first k right singular vectors** of a matrix A as the first k columns of matrix V in the singular value decomposition $A = U\Sigma V^T$. Now notice that, if $A = U\Sigma V^T$, then $B = A^T A = V\Sigma^2 V^T$. It follows that if the singular values — the diagonal entries $\sigma_1, \sigma_2, \ldots$ of Σ — are distinct, then the eigenvectors of B are exactly the *right singular vectors* of A. From this point on, we assume that the singular values of A are distinct, so that the rows of matrix Q, originally defined as the k most significant eigenvectors of A, are also the first k right singular vectors of A.

Now let $\epsilon = \sigma_{k+1}$ and define matrix N_ϵ so that

$$V = \begin{pmatrix} Q^T & N_\epsilon \end{pmatrix}, \tag{4}$$

where V are all the right singular vectors of A (recall that $A = U\Sigma V^T$), and Q are the first k right singular vectors. Notice that x is in the ϵ-space of A, by definition, iff $x \in \mathsf{col}(N_\epsilon)$. Equation (4) also reveals to us that $\mathsf{col}(Q^T)$ and

$\mathsf{col}(N_\epsilon)$ are complementary vector spaces in $\mathbb{R}^n$. This gives us a nice characterization of those vectors x in the ϵ-space of A in terms of Q. We may summarize this by writing

$$x \in \mathsf{col}(N_\epsilon) \quad \Leftrightarrow \quad Qx = 0.$$

Since the algorithm returns a vector x with $Q_i x = q_i$, we may write $\hat{x}$ for the augmented vector with $\hat{x}([n] - i) = x$ and $\hat{x}(i) = -1$ to see that $Q\hat{x} = 0$, and $\hat{x} \in \mathsf{col}(N_\epsilon)$.

Therefore our algorithm bounds the error of the approximation $A_i x \approx a_i$ by

$$||A_i x - a_i|| = ||A\hat{x}|| \leq \epsilon ||\hat{x}|| \leq \epsilon(1 + ||x||_1),$$

where $\epsilon = \sigma_{k+1}$. Since we have minimized the value $||x||_1$, we have also imposed a degree of minimization on this bound as well.

3.2 Time complexity

In this section we will summarize those time complexity bounds which are available for our algorithms. These bounds do not appear to be optimal, primarily due to the difficulty in anticipating how many iterations will be required by our convergence techniques — finding eigenvectors or using the simplex method to solve our linear programming problem.

We claim that a single iteration of our algorithm runs in time $O(ckn^2 + LP_{time})$. As above, k is the number of eigenvectors of B that we track, and n is the number of time series in our data window A. We have also introduced the variable c as the number of iterations needed to compute subroutine most_sig_eig, and LP_{time} as the amount of time needed to solve our linear programming problem.

It is clear that updating $B \to \hat{B}$ takes at most time $O(n^2)$. There are several implementations of most_sig_eig available to use (see, e.g., [6] or [4]). If we use the block power method, then each block power iteration involves a matrix multiplication $S = QB$ followed by an orthonormalization $Q =$orthonormalize_rows(S); together a single step will take time $O(kn^2)$. If there are c block power iterations, then certainly this portion of the algorithm requires time $O(ckn^2)$.

Narendra Karmarkar [5] provides a linear programing algorithm which runs in time $O(n^{3.5}\log(n))$, although in practice we expect even better performance than this. In addition, we can expect better incremental speed if we use a *warm-start* technique, in which the previous time step's coefficient vector x is used as a starting point to converge to the current time step's new coefficient vector. Thus we simply summarize this portion of the time complexity as $O(LP_{time})$, and leave further evidence of incremental speedup to the experiments.

Thus far we have outlined pseudocode for our algorithm, demonstrated bounds on the accuracy, and briefly analyzed the time complexity of both versions. We are now ready to empirically test these ideas on real data.

4 Experiments

In this section we will describe several experiments performed on stock price data in order to test the speed, accuracy, and stability of our algorithm. Our data consists of stock prices obtained from the NYSETAQ database via the Wharton research data services (wrds.upenn.edu). We began with the prices of all stock trades in the NYSETAQ database during the first 10 business days of 2003. We then isolated the 500 most frequently traded stocks. From these 500 stocks, we built synchronized time series with a resolution of about 25 seconds per time step; this resolution was chosen to approximate the actual average time between trades for these stocks.

In each of these experiments, a particular stock, say the i^{th}, is time-shifted by one time step. We then use our algorithms to find, at each step, a sparse coefficient vector x so that $A_i x \approx a_i$. Since column a_i has been time-shifted forward, our vector x can be thought of as predicting the next value of that time series based on the current value of the other time series.

In our first experiment, we tracked one particular stock, chosen at random, amongst all 500 time series. Our sliding window consisted of 5000 time steps. We used the taper factor $\tau = 0.95$.

Figure 1 supports the idea of a balance between error tolerance (larger k give smaller ϵ) and sparsity (smaller k give sparser x). In the plot, we have used the absolute value of the *relative test error* r — if a is the actual stock price being predicted, and p is the predicted value, then

$$r = \frac{(p-a)}{a}.$$

Each point on the plot is the average of the absolute value $|r|$ of several relative test errors from a set of 25 consecutive iterations of our algorithm.

We remind the reader that low k corresponds with high sparsity, while higher values of k give better training accuracy. In this figure, plotting the relative test error of stock predictions against various values of k, we see that the worst (highest) relative test errors are those for which k is either very high or very low. This suggests that a good choice of k will exhibit a balance between extreme sparsity and extreme training accuracy.

Figure 2 illustrates the average percentage change of nonzero coefficients in the vector x between consecutive time steps. In particular, if x_t is the vector found by our algorithm at time t, then the percentage change p between time t and $t+1$ is given by

$$p - \frac{\#\big(\text{supp}(x_t) \,\Delta\, \text{sup}(x_{t+1})\big)}{k}.$$

We remind the reader that, for sets A and B, the symmetric difference

$$A \,\Delta\, B = (A - B) \cup (B - A)$$

is exactly the set of points on which A and B differ. Since the percentages in this figure never exceed 35%, this figure supports the idea that our algorithms give locally stable subset selection results.

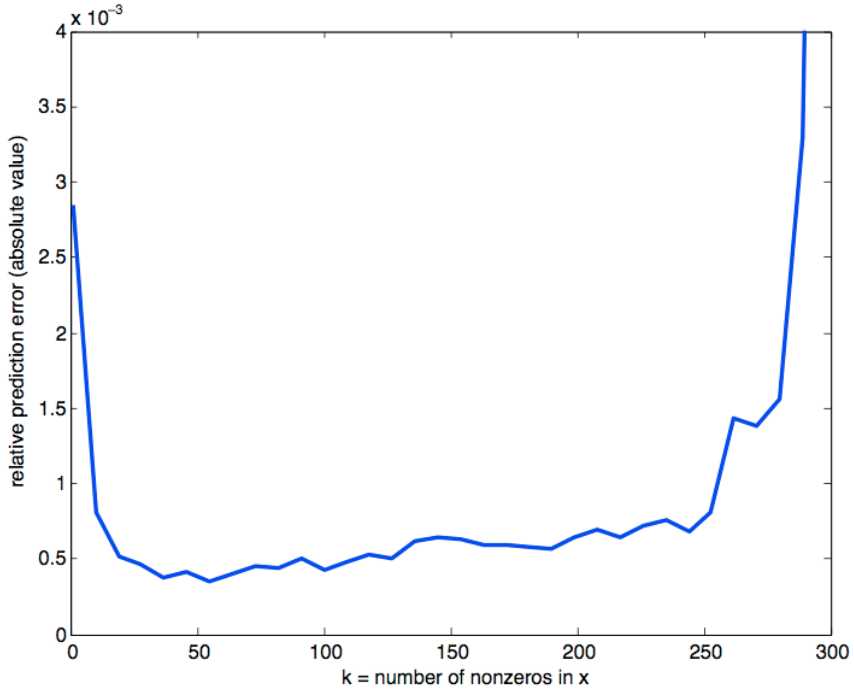

Figure 1: Average relative test error for different k

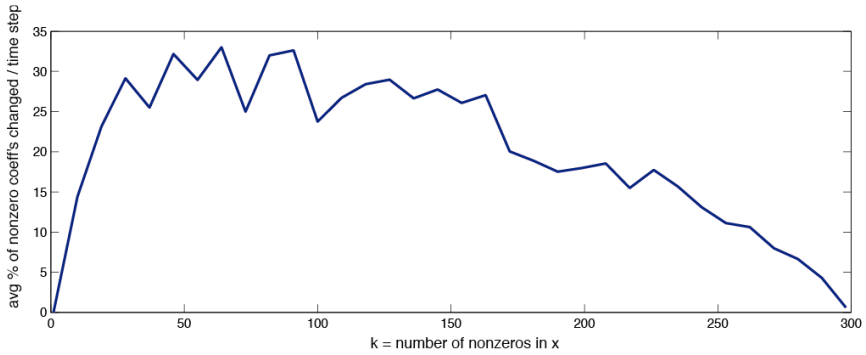

Figure 2: Average percentage change of nonzero coefficients (of x) for different k

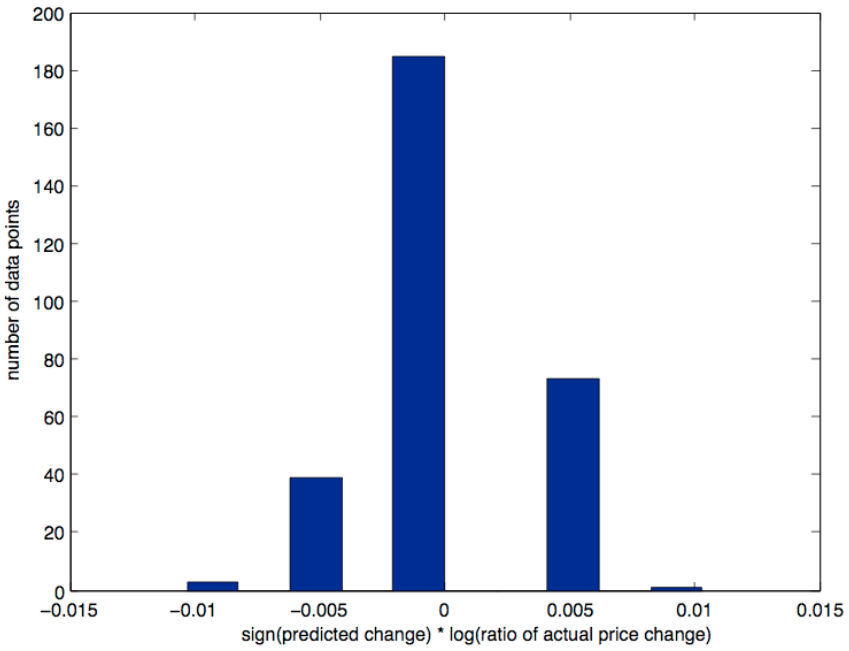

Figure 3: Histogram of classification quality

Our next experiment observed the quality of stock price predictions as a potential basis for a trading strategy. In this experiment, we set $k = 100$ and used the first difference of the natural logarithm of stock prices as our time series. This effectively encouraged our algorithm to try to predict the *change* in stock value (as a ratio) rather than the price itself.

In figure 3, we see a histogram of points from a particular set of 300 predictions. For each time step, we computed the value

$$v = \text{sign(predicted change)} \cdot \log(\text{actual change, as a ratio}).$$

Notice that positive v correspond to correct predictions while negative v to incorrect; moreover, the magnitude of this quantity reflects the "degree of correctness" of this particular prediction. Thus a good predictor will achieve a histogram which is highly skewed toward positive values. And, in figure 3, we do indeed see a positive skew.

Finally, in figure 4, we see the running times for three versions of our algorithm: a relatively naive nonincremental implementation, the stable version, and the fast version. Clearly, the slowest of the three is the nonincremental case. Furthermore, it appears (as much as we can tell from this limited data set) that the asymptotic time complexity of both our very fast and our stable versions is much better than that of the nonincremental version.

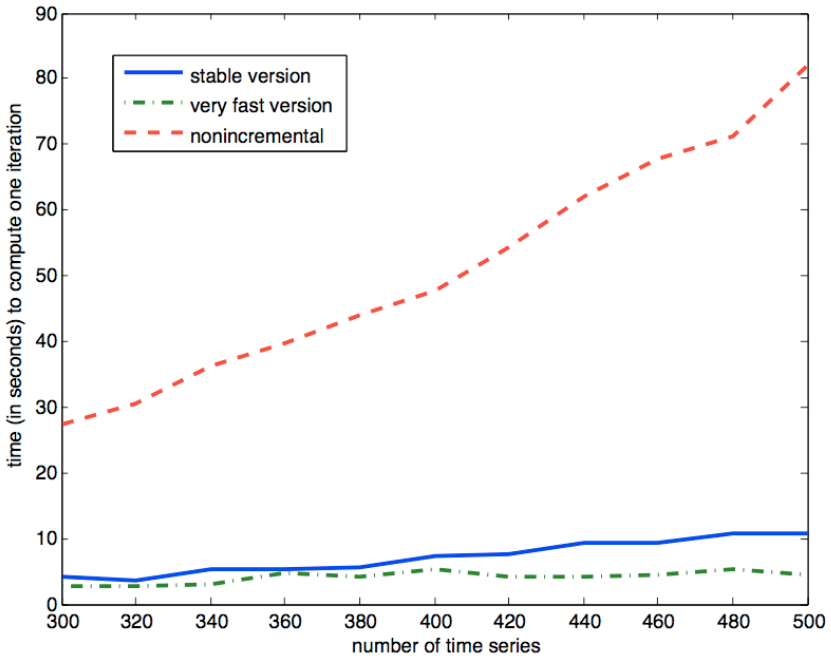

Figure 4: Time (in seconds) per iteration for different versions of our algorithm

References

[1] Scott Shaobing Chen, David L. Donoho, and Michael A. Saunders. Atomic decomposition by basis pursuit. *SIAM Review*, 43(1):129–159, 2001.

[2] David L. Donoho. For most large underdetermined systems of linear equations, the minimal l1-norm solution is also the sparsest. http://www-stat.stanford.edu/˜ donoho/Reports/2004/l1l0EquivCorrected.pdf, 2004.

[3] David L. Donoho and Jared Tanner. Sparse nonnegative solutions of underdetermined linear equations by linear programming. http://www-stat.stanford.edu/˜ donoho/Reports/2005/NonNegative-R5.pdf, 2005.

[4] Gene H. Golub and Charles F. Van Loan. *Matrix Computations, 2nd Ed.* The Johns Hopkins University Press, 1989.

[5] N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the 16th annual ACM symposium on theory of computing*, pages 302–311. ACM press, 1984.

[6] Lloyd N. Trefethen and III David Bau. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematic (SIAM), 1997.