16 Matrix Sketching

The singular value decomposition (SVD) can be interpreted as finding the most dominant *directions* in an $(n \times d)$ matrix A (or n points in $\mathbb{R}^d$). Typically $n > d$. It is typically easy to call a built in version of the SVD in many programming languages

$$[U, S, V] = \texttt{svd}(A)$$

where $U = [u_1, \ldots, u_n]$, $S = \mathsf{diag}(\sigma_1, \ldots, \sigma_d)$, and $V = [v_1, \ldots, v_d]$. Then $A = USV^T$ and in particular $A = \sum_{j=1}^d \sigma_j u_j v_j^T$. To approximate A we just use the first k components to find $A_k = \sum_{j=1}^k \sigma_j u_j v_j^T = U_k S_k V_k^T$ where $U_k = [u_1, \ldots, u_k]$, $S_k = \mathsf{diag}(\sigma_1, \ldots, \sigma_k)$, and $V_k = [v_1, \ldots, v_k]^T$. Then the vectors v_j (starting with smaller indexes) provide the best subspace representation of A.

But, although SVD has been *heavily* optimized on data sets that fit in memory (via LAPACK, found in Matlab, and just about every other language), it can sometimes be improved. The traditional SVD takes $O(\min\{nd^2, n^2d\})$ time to compute, which can be prohibitive for large n and/or d. Here we highlight two of these ways:

- to provide better interpretability of each v_j.
- to be more efficient on enormous scale, in a stream, or in distributed settings.

We will mainly focus discussion on streaming algorithms as a way to deal with the extreme scale of the data. While other models are available, and we will mention, we will focus on the model where A arrives in the stream, one row a_t at a time t. So our input is $\langle a_1, a_2, \ldots, a_t, \ldots, a_n \rangle$, and at any points a_t in the stream, we would like to maintain a sketch of the matrix B which somehow approximates all rows up to that point.

16.1 Covariance Matrix Summation

The first regime we focus on is when n is extremely large, but d is moderate. For instance $n = 100$ million, and $d = 1000$. The a simple approach in a stream is to make one pass using $O(d^2)$ space, and just maintain the sum of outer products $C_t = \sum_{i=1}^t a_i a_i^t$, the $d \times d$ covariance matrix of A exactly.

Algorithm 16.1.1 Summed Covariance

```
Set $C$ all zeros $(d \times d)$ matrix.
for rows (i.e. points) $a_i \in A$ do
    $C = C + a_i a_i^t$
return $C$
```

We have that any point t, where $A_t = [a_1; a_2; \ldots, a_t]$ in the stream the maintained matrix C is precisely $C = A_t A_t^T$. Thus the eigenvectors of C are the right singular vectors of A, and the eigenvalues of C are the squared singular values of C. This only requires $O(d^2)$ space, and $O(nd^2)$ total time, and incurs no error.

We can choose the top k eigenvectors of C as V_k, and on a second pass of the data, project all vectors on a_i onto V_k to obtain the best k-dimensional embedding of the dataset.

16.2 Frequent Directions

The next regime assumes that n is extremely large (say $n = 100$ million), but that d is also uncomfortably large (say $d = 100$ thousand), and our goal is something like a best rank k-approximation with $k \approx 10$. So

$k \ll d \ll n$. In this regime perhaps d is so large that d^2 space is too much, but something close to dk space and $O(ndk)$ time is reasonable. We will not be able to solve things exactly in the streaming setting under these constraints, but we can provide a provable approximation with slightly more space and time.

This approach, called Frequent Directions [8, 6], can be viewed as an extension of the Misra-Gries trick.

We will consider a matrix A one row (one point a_i) at a time. We will maintain a matrix B that is $2\ell \times d$, that is it only has 2ℓ rows (directions). We maintain that one row is always empty (has all 0s) at the end of each round (this will always be the last row B_ℓ).

We initialize with the first $2\ell - 1$ rows a_i of A as B, again with the last row B_ℓ left as all zeros. Then on each new row, we put a_i in the empty row of B. We set $[U, S, V] = \texttt{svd}(B)$. Now examine $S = \mathsf{diag}(\sigma_1, \ldots, \sigma_{2\ell})$, which is a length 2ℓ diagonal matrix. If $\sigma_{2\ell} = 0$ (then a_i is in the subspace of B), do nothing. Otherwise subtract $\delta = \sigma_\ell^2$ from each (squared) entry in S, that is $\sigma_j' = \sqrt{\max\{0, \sigma_j^2 - \delta\}}$ and in general $S' = \mathsf{diag}(\sqrt{\sigma_1^2 - \delta}, \sqrt{\sigma_2^2 - \delta}, \ldots, \sqrt{\sigma_{\ell-1}^2 - \delta}, 0, \ldots, 0)$.

Now we set $B = S'V^T$. Notice, that since S' only has non-zero elements in the first $\ell - 1$ entries on the diagonal, then B is at most rank $\ell - 1$ and we can then treat V and B as if the ℓth row does not exist.

Algorithm 16.2.1 Frequent Directions

```
Set $B$ all zeros ($2\ell \times d$) matrix.
for rows (i.e. points) $a_i \in A$ do
  Insert $a_i$ into a zero-valued row of $B$
  if ($B$ has no zero-valued rows) then
    $[U, S, V] = \texttt{svd}(B)$
    Set $\delta_i = \sigma_\ell^2$                                    # the $\ell$th entry of $S$
    Set $S' = \mathsf{diag}\left(\sqrt{\sigma_1^2 - \delta}, \sqrt{\sigma_2^2 - \delta}, \ldots, \sqrt{\sigma_{\ell-1}^2 - \delta}, 0, \ldots, 0\right)$.
    Set $B = S'V^T$                          # the last rows of $B$ will again be all zeros
return $B$
```

The result of Algorithm 16.2.1 is a matrix B such that for any (direction) unit vector $x \in \mathbb{R}^d$

$$0 \leq \|Ax\|^2 - \|Bx\|^2 \leq \|A - A_k\|_F^2/(\ell - k)$$

and [7, 6]

$$\|A - A\Pi_{B_k}\|_F^2 \leq \frac{\ell}{\ell - k}\|A - A_k\|_F^2,$$

for any $k < \ell$, including when $k = 0$. So setting $\ell = 1/\varepsilon$, then in any direction in $\mathbb{R}^d$, the squared mass in that direction is preserved up to $\varepsilon\|A\|_F^2$ (that is, ε times the total squared mass) using the first bound. And in the second bound if we set $\ell = \lceil k/\varepsilon + k \rceil$ then we have $\|A - A\Pi_{B_k}\|_F^2 \leq (1 + \varepsilon)\|A - A_k\|_F^2$. Recall that $\|A\|_F = \sqrt{\sum_{a_i \in A} \|a_i\|^2}$.

- *Why does this work?*

  Just like with Misra-Greis [9], when some mass is deleted from one counter it is deleted from all ℓ counters, and none can be negative. So here when one direction has its (squared) mass decreased, at least ℓ directions (with non-zero squared mass) are decreased by the same amount. So no direction can have more than $1/\ell$ fraction of the total squared mass $\|A\|_F^2$ decreased from it.

  Finally, since squared mass can be summed independently along any set of **orthogonal** directions, we can subtract each of them without affecting others. Setting $\ell = 1/\varepsilon$ implies that no direction x (e.g., assume $\|x\| = 1$, and measure $\|Ax\|^2$) decreases is squared norm (as $\|Bx\|^2$) by more than $\|A\|_F^2$.

By a more careful analysis that we only shrink the total norm proportional to the "tail" $\|A - A_k\|_F^2$, then we can obtain the bound described above. See [6] for more details, spelled out in a few lines of linear algebra.

- *Why do we use the* `svd`*?*
  The SVD defines the true axis of the ellipse associated with the norm of B at each step. If we shrink along an basis (or even a set of non-orthogonal vectors) we will warp the ball, and we will not be able to ensure that each direction of B shrinks in squared norm by at most δ_i.

- *Did we* ***need*** *to use the* `svd`*? (its expensive, right)?*
  The cost is amortized. We only call the svd once every ℓ steps, so at most $O(n/\ell)$ times. Since each call takes $O(d\ell^2)$ time, the total cost is $O(nd\ell)$, or only ℓ times as long as reading the matrix.
  It is also possible to call approximate versions of the SVD [5]. This allows versions which have runtime depending on the number of non-zeros in the input matrix. This makes a big difference for very sparse word count or recommendation system matrices.

- *What happened to U in the* `svd` *output?*
  The matrix U just related the main directions to each of the n points (rows) in A. But we don't want to keep around the space for this. In this application, we only care about the directions or subspace that best represents the points; e.g. PCA only cares about the right singular vectors.

16.3 Row Sampling

We next move to a regime where n and d are again both large, and so might be k. But a runtime of $O(ndk)$ may be too large – that is we can read the data, but maybe a factor of k times reading the data is also large. The next algorithms have runtime $\tilde{O}(nd)$ (where $\tilde{O}$ may hide log factors), they are as fast as reading the data. In particular, if there $\mathsf{nnz}(A)$ non-zero entries in a very space matrix, then the runtime is only $\tilde{O}(\mathsf{nnz}(A))$.

The goal is to approximate A up to the accuracy of A_k. But in A_k the directions v_i are *linear combinations of features*.

- What is a linear combination of genes?
- What is a linear combination of typical grocery purchases?

Instead our goal is to choose V so that the columns of V are also columns of A.

For each row of $a_i \in A$, set $w_i = \|a_i\|^2$. Then select $\ell = (k/\varepsilon)^2 \cdot \log(1/\delta)$ rows of A, each proportional to w_i. Let R be the "stacking" of these rows.

These ℓ rows will jointly act in place of V_k^T. However since V was orthogonal, then the columns $v_i, v_j \in V_k$ were orthogonal. This is not the case for R, we need to orthogonalize R. Let $\Pi_R = R^T(RR^T)^{-1}R$ be the projection matrix for R, so that $A_R = A\Pi_R$ describes the *projection* of A onto the subspace of the directions spanned by R. Now

$$\|A - A\Pi_R\|_F \leq \|A - A_k\|_F + \varepsilon\|A\|_F$$

with probability at least $1 - \delta$ [4].

- *Why did we not just choose the t rows of A with the largest w_j values?*
  Some may point along the same "direction" and would be repetitive. This should remind you of the choice to run k-means++ versus the Gonzalez algorithm for greedy point-assignment clustering.

- *Why did we not factor out the directions we already picked?*
  We could, but this allows us to run this in a streaming setting. (See next approach)

- *But $A\Pi_R$ could be rank ℓ, can we get it rank $k \ll \ell$?*
  Yes, you can take its best rank k approximation $[\Pi_R A]_k$ and about the same bounds hold, you may need to increase ℓ slightly.

- *Can we get a better error bound?*
  Yes. First take SVD $[U, S, V] = \texttt{svd}(A)$ and let U_k be the top k left singular vectors. Let $U_k(i)$ be the ith row of U_k. Now the *leverage* score of data point a_i is $s_i = \|U_k(i)\|^2$. Using the leverage scores as weights $w_i = s_i$ allows one to achieve stronger bounds [2]

  $$\|A - A\Pi_R\|_F \leq (1+\varepsilon)\|A - A_k\|_F.$$

  But this requires us to first take the SVD (or other time-consuming procedures), so its is harder to do in a stream; although some newer approaches address this [3]. In many cases, these approaches do not seem to provide tangible benefits over the faster $\|a_i\|^2$-weighted sampling.

  There exist more complicated and slower approaches which achieve the same bound with slightly smaller ℓ [1].

- *Can we also sample columns this way?*
  Yes. All tricks can be run on A^T the same way (in fact most of the literature talks about sampling columns instead of rows). And, both approaches can be combined. This is known as the CUR-decomposition of A.

- *How do we best do this in a stream?*
  The classic analysis assumes that this is done with each row selected independently – some are chosen twice. This can be done in a stream with Reservoir sampling. This requires $O(\ell d)$ space at any point in time, and $O(\ell + d)$ time to process a row. This can be reduced to $O(d + \log \ell)$ using priority sampling, which also reduces the variance.

A significant downside of these row sampling approaches is that the $(1/\varepsilon^2)$ coefficient can be quite large for a small error tolerance. If $\varepsilon = 0.01$, meaning 1% error, then this part of the coefficient alone is 10,000. In practice, the results may be better, but for guarantees, this may only work on very enormous matrices.

16.4 Count Sketch Hashing for Sparse Matrices

This does not give interpretability, but is even more efficient than the column selection, and obtains the strong error guarantees.

The starting point is a JL projection matrix $S \in \mathbb{R}^{n \times l}$ that maps A to a $\ell \times d$ matrix B. This preserves relative error (an oblivious subspace embedding) with $\ell = O(d/\varepsilon^2)$ so, for all x

$$(1-\varepsilon) \leq \frac{\|Ax\|}{\|Bx\|} \leq (1+\varepsilon).$$

A very strong bound, that also ensures results from regression are maintained.

Increasing ℓ to $\ell = O(d^2/\varepsilon^2)$, then a fast count-sketch based approach can be used. Now S has each row s_i as all 0s, except for one randomly chosen entry (a hash to a row of B) that is either -1 or $+1$ at random. This works just like a count sketch but for matrices.

The runtime is only $O(\mathsf{nnz}(A))$, truely as fast as reading the data. But the compression of B is not as interpretable as column selection, or as sparse as Frequent Directions.

Bibliography

[1] Joshua Batson, Daniel A. Spielman, and Nikhil Srivastava. Twice-ramanujan sparsifiers. *SIAM Review*, (315–334), 56. arXiv:0808.0163.

[2] Christos Boutsidis, Michael W Mahoney, and Petros Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 968–977. Society for Industrial and Applied Mathematics, 2009.

[3] Michael B. Cohen, Cameron Musco, and Christopher Musco. Input sparsity time low-rank approximation via ridge leverage score sampling. *SODA*, 2017.

[4] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*. IEEE, 1998.

[5] Mina Ghashami, Edo Liberty, and Jeff M. Phillips. Efficient frequent directions algorithm for sparse matrices. In *KDD*, 2016.

[6] Mina Ghashami, Edo Liberty, Jeff M. Phillips, and David P. Woodruff. Frequent directions: Simple and deterministic matrix sketching. *SICOMP*, 2016.

[7] Mina Ghashami and Jeff M. Phillips. Relative errors for deterministic low-rank matrix approximations. In *ACM-SIAM 25th Symposium on Discrete Algorithms*, 2014.

[8] Edo Liberty. Simple and deterministic matrix sketching. In *Proceedings 19th ACM Conference on Knowledge Discovery and Data Mining*, 2013.

[9] J. Misra and D. Gries. Finding repeated elements. *Sc. Comp. Prog.*, 2:143–152, 1982.