16 Approximate PCA

Recall that PCA is the process to find the most dominant *directions* in an $(n \times d)$ matrix A (or n points in $\mathbb{R}^d$). Typically $n > d$. We explored before using the SVD

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

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 $V_k = [v_1, \ldots, v_k]^T$. The 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. Here we highlight two of these ways:

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

The SVD takes $O(\min\{nd^2, d^2 n\})$ time to compute.

16.1 Row Sampling

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 $t = (k/\varepsilon)^2 \cdot \log(1/\delta)$ rows of A, each proportional to w_i. Let the "stacking" of these rows be R.

These t 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 $1 - \delta$ [2].

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

- *Can we get a better error bound?*
  Yes. First take SVD $[U, S, V] = \text{svd}(A)$ 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 $\ell_i = \|U_k(i)\|^2$. Using the leverage scores as weights $w_i = \ell_i$ allows one to achieve stronger bounds [1]

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

  But this requires us to first take the SVD (or other time-consuming procedures), so its is harder to do in a stream.

- *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.

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.

16.2 Frequent Directions

Another efficient solution is provided by using a Misra-Gries trick. It is called Frequent Directions [5, 3].

We will consider a matrix A one row (one point a_i) at a time. We will maintain a matrix B that is $\ell \times d$, that is it only has ℓ 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 $\ell - 1$ 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 $S = \textsf{diag}(\sigma_1, \ldots, \sigma_\ell)$, which is a length ℓ diagonal matrix. If $\sigma_\ell = 0$ (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 $\sigma'_j = \sqrt{\sigma_j^2 - \delta}$ and in general $S' = \textsf{diag}(\sqrt{\sigma_1^2 - \delta}, \sqrt{\sigma_2^2 - \delta}, \ldots, \sqrt{\sigma_{\ell-1}^2 - \delta}, 0)$.

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 $(\ell \times d)$ matrix.
for rows (i.e. points) $a_i \in A$ **do**
  Set $B_\ell \leftarrow a_i$ # put a_i in all-zero row B_ℓ of B
  $[U, S, V] = \texttt{svd}(B)$
  Set $\delta = \sigma_\ell^2$ # the ℓth entry of S
  Set $S' = \textsf{diag}\left(\sqrt{\sigma_1^2 - \delta}, \sqrt{\sigma_2^2 - \delta}, \ldots, \sqrt{\sigma_{\ell-1}^2 - \delta}, 0\right)$.
  Set $B = S'V^T$ # the last row 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 [4, 3]

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

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 [6], 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, all ℓ directions (with non-zero squared mass) are decreased. 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.

- *Why do we use the* `svd`*?*
  Because we can choose any orthogonal basis, we find the one that has the smallest δ value to decrease by. This is what the `svd` gives us.

- *Did we **need** to use the* `svd`*? (its expensive, right)?*
  Well, we only need to run it on a matrix of size $\ell \times d$, so $d\ell^2$ might not be too bad ... although this is repeated n times. We can decrease the total runtime to $O(nd\ell)$ by waiting until we have 2ℓ non-empty rows, and then shrinking to decrease to $\ell - 1$ non-empty rows again. This retains the same error bounds, but only calls the `svd` about n/ℓ times.

- *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.

Bibliography

[1] 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.

[2] 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.

[3] Mina Ghashami, Edo Liberty, Jeff M. Phillips, and David P. Woodruff. Frequent directions: Simple and deterministic matrix sketching. Technical report, arXiv:1501.01711, 2015.

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

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

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