Stat260/CS294: Randomized Algorithms for Matrices and Data

Lecture 10 - 10/07/2013

Lecture 10: Fast Random Projections and FJLT, cont.

Lecturer: Michael Mahoney *Scribe: Michael Mahoney*

Warning: these notes are still very rough. They provide more details on what we discussed in class, but there may still be some errors, incomplete/imprecise statements, etc. in them.

10 Fast Random Projections and FJLT, Cont.

We continue with the discussion from last time. There is no new reading, just the same as last class.

Today, we will cover the following.

- We will describe a fast algorithm to compute very fine approximations to the leverage scores of an arbitrary tall matrix.
- We will describe a few subtleties to extend this basic algorithm to non-tall matrices, which is of interest in extending these LS ideas to low-rank matrix approximation.
- We will describe how to use this algorithm in a fast random sampling algorithm for the LS problem (that has a running time that is essentially the same as the fast random projection algorithm that we discussed last time).

Today will basically wrap up our worst-case theory discussion for the LS problem. Next time, we will start to discuss how these ideas can be used in practice in high-quality implementations for the overdetermined LS problem.

10.1 Computing leverage scores quickly for a tall matrix

The leverage scores for a tall $n \times d$ matrix A, with $n \gg d$, are the diagonal elements of the projection matrix onto the column span of A. Equivalently, they are the Euclidean norms of the rows of *any* $n \times d$ orthogonal matrix U spanning the column span of A. Thus, a straw-man algorithm is to perform a QR decomposition of the SVD on A and read off the leverage scores. This takes $O(nd^2)$ time, and so this represents what our fast algorithm should beat.

The basic idea behind the fast algorithm to approximate leverage quickly is to use the "R" matrix, not from a QR decomposition of the original matrix A, but from a random sketch of A of the form $\Pi_1 A$, where Π_1 is a FJLT. Equivalently, the idea is to use a "randomized sketch" of the form $A(\Pi_1 A)^\dagger \Pi_2$, where Π_1 is an FJLT and Π_2 is a JLT. To see this, recall that if ℓ_i is the i^{th} leverage score, then

$$\ell_i = \left\|U_{(i)}\right\|_2^2 = \left\|e_i^T U\right\|_2^2 = \left\|e_i U U^T\right\|_2^2 \tag{1}$$

$$= \left\|e_i A A^\dagger\right\|_2^2 = \left\|\left(AA^\dagger\right)_{(i)}\right\|_2^2. \tag{2}$$

Viewed this way, the hard part of computing ℓ_i via Eqn. (1) is to compute the U matrix, which takes $O(nd^2)$ time; and that hard part of computing ℓ_i via Eqn. (2) is first to compute $A^\dagger$ and second to do the matrix multiplication between A and $A^\dagger$, each of which take $O(nd^2)$ time.

While Eqn. (2) might seem more difficult to work with, we can insert random projections at appropriate places to speed up both steps. In particular, we have the following.

$$\begin{aligned} \ell_i &= \left\|e_i A A^\dagger\right\|_2^2 \\ &\approx \left\|e_i A \left(\Pi_1 A\right)^\dagger\right\|_2^2 = \hat{\ell}_i \\ &\approx \left\|e_i A \left(\Pi_1 A\right)^\dagger \Pi_2\right\|_2^2 = \tilde{\ell}_i. \end{aligned}$$

In these expressions, $A \in \mathbb{R}^{n\times d}$, $\Pi_1 \in \mathbb{R}^{r_1\times n}$, where $r_1 = O\left(\frac{d\log(d)}{\epsilon}\right)$, in which case $\Pi_1 A \in \mathbb{R}^{r_1\times d}$ and $(\Pi_1 A)^\dagger \in \mathbb{R}^{d\times r_1}$.

That is, we compute the pseudo inverse of the smaller matrix $\Pi_1 A$, rather than A.

But, computing the product of A and $(\Pi_1 A)^\dagger$ takes $O(ndr)$ time, which is $\Omega(nd^2)$ time, since $r \approx \frac{d\log(d)}{\epsilon}$. On the other hand, we only need estimates of the Euclidean norms of the rows, and so we can do a second random projection by $\Pi_2 \in \mathbb{R}^{r_1\times r_2}$, where $r_2 = O\left(\log(n)\right)$, which if the matrices are multiplied in the proper order is faster.

(As an aside, we note that the sketch $A\left(\Pi_1 A\right)^\dagger \Pi_2$ can be used in other ways, e.g., to estimate the dot products between different rows of U, which is of interest since the so-called cross leverage scores are defined to be $c_{ij} = U_{(i)}^T U_{(j)}$, i.e., the off-diagonal elements of the projection matrix. Also, the so-called coherence is $\gamma = \max_i \ell_i$, although it is sometimes defined to be $\max_{ij} c_{ij}$.)

With this motivation, here is the main algorithm for computing approximations to the leverage scores quickly.

Algorithm 1 The `FastApproximateLeverageScores` algorithm.

```
Input: $A \in \mathbb{R}^{n\times d}$, with SVD $A = U\Sigma V^T$, and $\epsilon \in (0, 1/2]$.
Output: $\tilde{\ell}_i$, for $i \in [n]$
1: Let $\Pi_1 \in \mathbb{R}^{r_1\times n}$ be an $\epsilon$-FJLT for $U$, with $r_1 = \Omega\left(\frac{d\log(n)}{\epsilon^2}\log\left(\frac{d\log(n)}{\epsilon^2}\right)\right)$.
2: Compute $\Pi_1 A \in \mathbb{R}^{r_1\times d}$ and its QR decomposition or SVD. Let $R \in \mathbb{R}^{r\times r}$ be the "R" matrix
   from QR or $\Sigma_{\Pi_1 A} V_{\Pi_1 A}^T$ from SVD.
3: View the rows of $AR^{-1} \in \mathbb{R}^{n\times d}$ as $n$ vectors in $\mathbb{R}^d$, and let $\Pi_2 \in \mathbb{R}^{d\times r_2}$ be an $\epsilon$-JLT for $n^2$
   vectors (the $n$ vectors and the $\binom{n}{2}$pairwise sums), with $r_2 = O\left(\frac{\log(n)}{\epsilon^2}\right)$.
4: Construct $\Omega = AR^{-1}\Pi_2$.
5: For all $i \in [n]$, compute and return $\tilde{\ell}_i = \left\|\Omega_{(i)}\right\|_2^2$.
```

Here is the main theorem that we can establish for this algorithm.

Theorem 1 *The* `FastApproximateLeverageScores` *algorithm returns $\tilde{\ell}_{(i)}$ such that*

$$\left|\ell_i - \tilde{\ell}_i\right| \le \epsilon \ell_i,$$

for all $i \in [n]$, with constant probability. The running time is

$$O\left(nd \log\left(d/\epsilon\right) + nd\epsilon^{-2}\log(n) + d^3\epsilon^{-2}\log(n)\log\left(d\epsilon^{-1}\right)\right)$$

(assuming that $d \le n \le e^d$, with a more complicated expression otherwise).

Before presenting the main proof, let's start with an outline of the proof, which will follow the discussion above. The algorithm compute $\tilde{\ell}_i = \left\|\tilde{U}_{(i)}\right\|_2^2$, where $\tilde{U}_{(i)} = e_i^T A \left(\Pi_1 A\right)^{\dagger} \Pi_2$. The first thing to note is that we need to be not naive about the order of operations, e.g., don't multiply $AR^{-1}\Pi_2$ in the wrong order; and also that Π_2 is included only to improve the (worst case FLOPS) running time. That is, the quality-of-approximation result holds if Π_2 is not there, and in certain implementations the algorithm might be faster/better without it. So, if we can establish what we want without the Π_2, then including it is a JL transformation and doesn't change the distances (in particular the norms of the rows) too much.

So, the result will follow if

$$e_i = A\left(\Pi_1 A\right)^{\dagger}\left(\left(\Pi_1 A\right)^{\dagger}\right)^T e_j = e_i U U^T e_j$$

and if $\Pi_1 A$ is efficient to compute. But since Π_1 is an FJLT, to compute the quantity $\left(\Pi_1 A\right)^{\dagger}$ takes $O\left(nd\log\left(r_1\right) + r_1 d^2\right)$ time. Then, by the structural lemma from the end of last class, we have that

$$\left(\Pi_1 A\right)^{\dagger} = V\Sigma^{-1}\left(\Pi_1 U\right)^{\dagger},$$

which recall is false in general but holds when rank is preserved. So,

$$e_i A\left(\Pi_1 A\right)^{\dagger}\left(\left(\Pi_1 A\right)^{\dagger}\right)^T A^T e_j = e_i U\left(\Pi_1 U\right)^{\dagger}\left(\left(\Pi_1 U\right)^{\dagger}\right)^T U^T e_j.$$

Since Π_1 is an FJLT, we have that

$$\left(\Pi_1 U\right)^{\dagger}\left(\left(\Pi_1 U\right)^{\dagger}\right)^T \approx I_d,$$

from which the theorem will follow.

Note also that this analysis is more general, e.g., it shows that we can compute the large cross-leverage scores. To extract them, however, takes additional work, which basically amounts to using a "heavy hitter" algorithm to find the large ones without looking at all $\binom{n}{2}$ possibilities. This can be done—see DMMW—but we won't describe it here.

Now onto the proof of Theorem 1.

Proof: We will condition our analysis on the following two events, each of which holds with constant probability if we choose parameters correctly.

$$\begin{aligned} \text{Event 1}: &\quad \Pi_1 \in \mathbb{R}^{r_1 \times n} \text{ is an } \epsilon\text{-FJLT for } U \\ \text{Event 2}: &\quad \Pi_2 \in \mathbb{R}^{r_1 \times r_2} \text{ is an } \epsilon\text{-JLT for the } \binom{n}{2} \text{ points we described.} \end{aligned}$$

Then, let's define the following "hatted" and "tilded" quantities

$$\hat{U}_{(i)} = e_i A \left(\Pi_1 A\right)^{\dagger} \qquad \hat{\ell}_i = \left\|\hat{U}_{(i)}\right\|_2^2$$
$$\tilde{U}_{(i)} = e_i A \left(\Pi_1 A\right)^{\dagger} \Pi_2 \qquad \tilde{\ell}_i = \left\|\tilde{U}_{(i)}\right\|_2^2,$$

where the hatted quantities correspond to one level of approximation (with Π_1) and the tilded quantities correspond to a second level of approximation (with Π_2). The theorem will follow if we show that

$$U_{(i)}^T U_{(j)} \approx \hat{U}_{(i)}^T \hat{U}_{(j)}$$

and also that

$$\hat{U}_{(i)}^T \hat{U}_{(j)} \approx \tilde{U}_{(i)}^T \tilde{U}_{(j)}$$

with appropriate parameters.

More precisely, we need to establish the following two results.

Lemma 1 *For all $i, j \in [n]$, we have that*

$$\left|U_{(i)}^T U_{(j)} - \hat{U}_{(i)}^T \hat{U}_{(j)}\right| \leq \frac{\epsilon}{1-\epsilon} \left\|U_{(i)}\right\|_2 \left\|U_{(j)}\right\|_2$$

Lemma 2 *For all $i, j \in [n]$, we have that*

$$\left|\hat{U}_{(i)}^T \hat{U}_{(j)} - \tilde{U}_{(i)}^T \tilde{U}_{(j)}\right| \leq 2\epsilon \left\|\hat{U}_{(i)}\right\|_2 \left\|\hat{U}_{(j)}\right\|_2$$

Before proving these two lemmas, let's say what they mean and how they imply the results of the theorem.

First, observe that Lemma 1 says that the leverage scores (for $i = j$; respectively, the cross-leverage scores for $i \neq j$) are preserved to within relative (respectively, additive) error, i.e., the action of Π_1 doesn't distort them too much. (Viewing this in terms of approximate matrix multiplication, if $i = j$ then the two matrices/vectors are aligned with no cancellation, while if $i \neq j$ they will in general not be, and so there might be cancellation, in which case obtaining relative error bounds isn't possible.)

Second, Lemma 2 says the same thing for the action of applying Π_2. In particular, by Lemma 1 it follows that

$$\left|\ell_i - \hat{\ell}_i\right| \leq \frac{\epsilon}{1-\epsilon} \ell_i,$$

and by Lemma 2 it follows that

$$\left|\hat{\ell}_i - \tilde{\ell}_i\right| \leq 2\epsilon \hat{\ell}_i.$$

Finally, by combining these results, it follows that

$$\begin{aligned} \left|\ell_i - \tilde{\ell}_i\right| &\leq \left|\ell_i - \hat{\ell}_i\right| + \left|\hat{\ell}_i - \tilde{\ell}_i\right| \\ &\leq \left(\frac{\epsilon}{1-\epsilon} + 2\epsilon\right) \ell_i \\ &\leq 4\epsilon \ell_i, \end{aligned}$$

where the first inequality follows from the triangle inequality, the second inequality follows by applying the two lemmas, and the last inequality follows since $\epsilon \leq 1/2$. From this, the quality-of-approximation claims of the theorem theorem follow. So, let's prove those two lemmas.

Proof:[of Lemma 1] Let $A = U\Sigma V^T$, and recall that $(\Pi_1 A)^\dagger = V\Sigma^{-1}(\Pi_1 U)^\dagger$. Then,

$$\begin{aligned}\hat{U}_{(i)}^T \hat{U}_{(j)} &= e_i A (\Pi_1 A)^\dagger \left((\Pi_1 A)^\dagger\right)^T A^T e_j \\ &= e_i U\Sigma V^T V \Sigma^{-1} (\Pi_1 U)^\dagger \left((\Pi_1 U)^\dagger\right)^T \Sigma^{-1} V^T V \Sigma U^T e_j \\ &= e_i U (\Pi_1 U)^\dagger \left((\Pi_1 U)^\dagger\right)^T U^T e_j\end{aligned}$$

Thiu, it follows that

$$\begin{aligned}\left|U_{(i)}^T U_{(j)} - \hat{U}_{(i)}^T \hat{U}_{(j)}\right| &= \left|e_i U U^T e_j - e_i U (\Pi_1 U)^\dagger \left((\Pi_1 U)^\dagger\right)^T U^T e_j\right| \\ &= \left|e_i U \left(I - (\Pi_1 U)^\dagger \left((\Pi_1 U)^\dagger\right)^T\right) U^T e_j\right| \\ &\leq \left\|I - (\Pi_1 U)^\dagger \left((\Pi_1 U)^\dagger\right)^T\right\|_2 \left\|U_{(i)}\right\|_2 \left\|U_{(j)}\right\|_2 .\end{aligned}$$

Since $\Pi_1 U = U_{\Pi_1 U}\Sigma_{\Pi_1 U} V_{\Pi_1 U}^T$ and $(\Pi_1 U)^\dagger \left((\Pi_1 U)^\dagger\right)^T = V_{\Pi_1 U}\Sigma_{\Pi_1 U}^{-2} V_{\Pi_1 U}^T$, it then follows that

$$\begin{aligned}\left|U_{(i)}^T U_{(j)} - \hat{U}_{(i)}^T \hat{U}_{(j)}\right| &= \left\|I - V_{\Pi_1 U}\Sigma_{\Pi_1 U}^{-2} V_{\Pi_1 U}^T\right\|_2 \left\|U_{(i)}\right\|_2 \left\|U_{(j)}\right\|_2 \\ &= \left\|I - \Sigma_{\Pi_1 U}^{-2}\right\|_2 \left\|U_{(i)}\right\|_2 \left\|U_{(j)}\right\|_2 \\ &= \frac{\epsilon}{1-\epsilon} \left\|U_{(i)}\right\|_2 \left\|U_{(j)}\right\|_2 ,\end{aligned}$$

which establishes the lemma.

$\diamond$

Proof:[of Lemma 2] Since Π_2 is and ϵ-JLT, it preserves the norm of n^2 vectors. Let $x_i = \hat{U}_{(i)}/\left\|\hat{U}_{(i)}\right\|_2$, and consider the following n^2 vectors:

$$\begin{array}{cc} x_i & i \in [n] \\ x_i + x_j & i,j \in [n], i \neq j. \end{array}$$

By the ϵ-JLT property, and since $\|x_i\|_2 = 1$, it follows that:

$$\begin{array}{rl} 1-\epsilon \leq \|x_i \Pi_2\|_2 \leq 1+\epsilon & \forall i \\ (1-\epsilon)\|x_i + x_j\|_2^2 \leq \|(x_i + x_j)\Pi_2\|_2^2 \leq (1+\epsilon)\|x_i + x_j\|_2^2 & \forall i.j \text{ s.t. } i \neq j. \end{array}$$

If we combine these, expand the squares, and use that $\|\alpha + \beta\|_2^2 = \|\alpha\|_2^2 + \|\beta\|_2^2 + 2\alpha^T\beta$, for vectors α and β, and use that $\|x\|_2 = 1$, we have that

$$x_i^T x_j - 2\epsilon \leq (x_i\Pi_2)^T (x_j \Pi_2) \leq x_i^T x_j + 2\epsilon.$$

If we multiply through by $\left\|\hat{U}_{(i)}\right\|_2 \left\|\hat{U}_{(j)}\right\|_2$ and then use the homogeneity of the inner product, we get that

$$\hat{U}_{(i)}^T \hat{U}_{(j)} - 2\epsilon \left\|\hat{U}_{(i)}\right\|_2 \left\|\hat{U}_{(j)}\right\|_2 \leq \left(\hat{U}_{(i)}\Pi_2\right)^T (U_{(j)}\Pi_2) \leq \hat{U}_{(i)}^T \hat{U}_{(j)} + 2\epsilon \left\|\hat{U}_{(i)}\right\|_2 \left\|\hat{U}_{(j)}\right\|_2 ,$$

which establishes the lemma.

◇

For the running time, here is a summary.

- Computing $\Pi_1 A$ takes time $O\left(nd\log(r_1)\right)$.
- Computing the SVD of $\Pi_1 A$ takes time $O\left(r_1 d^2\right)$.
- Computing $R^{-1}\Pi_2$ takes time $O\left(r_2 d^2\right)$.
- Premultiplying by A takes time $O\left(ndr\right)$.

Thus, the overall running time is $O\left(nd\log(r_1) + ndr_2 + r_1 d^2 + r_2 d^2\right)$, which by the choice of the various parameters is $O\left(nd\log(n) + d^3\log(n)\log(d)\right)$ time.

Combining these results establishes the theorem.

◇

10.2 Using the fast leverage score approximation algorithm for a fast random sampling algorithm for the LS problem

We have already seen a fast random projection-based algorithm for the LS problem that runs in $o(nd^2)$ time as well as a slow random sampling algorithm that used the exact leverage scores as an importance sampling distribution. Not surprisingly, we can use the fast approximations to the leverage scores to get a fast random sampling-based algorithm for the LS problem that runs in $o(nd^2)$ time. Here is that algorithm.

Algorithm 2 A "fast" random sampling algorithm for the LS problem.

```
Input: An $n \times d$ matrix $A$, with $n \gg d$, an $n$-vector $b$, and an error parameter $\epsilon \in (0, 1/2)$.
Output: A $d$-vector $\tilde{x}_{opt}$
1: Run the FastApproximateLeverageScores algorithm (with $\epsilon = 1/2$ as input to that algorithm)
   to get 2 approximations to all of the leverage scores of $A$; rescale them to form an importance
   sampling distribution $\{p_i\}_{i=1}^n$.
2: Randomly sample $r \gtrsim O(\frac{d\log d}{\epsilon})$ rows of $A$ and elements of $b$, using $\{p_i\}_{i=1}^n$ as the importance
   sampling distribution, rescaling each by $\frac{1}{rp_{i_t}}$, i.e., form $SA$ and $Sb$.
3: Solve $\min_{x\in\mathbb{R}^d} \|SAx - Sb\|_2$ with a black box to get $\tilde{x}_{opt}$.
4: Return $\tilde{x}_{opt}$.
```

For this algorithm, we can prove the following.

Theorem 2 *For this algorithm, the output is a vector $\tilde{x}_{opt}$ such that with probability ≥ 0.8:*

- $\|A\tilde{x}_{opt} - b\|_2 \leq (1+\epsilon)\mathcal{Z}$
- $\|\tilde{x}_{opt} - x_{opt}\|_2 \leq \sqrt{\epsilon}\kappa(A)\sqrt{\gamma^{-2}-1}\,\|x_{opt}\|_2$

In addition, the running time is $O\left(nd\log(n) + d^3\log(n)\log(d)\right)$.

The quality-of-approximation claims follow since using the approximate leverage scores an an importance sampling distribution leads to the two structural conditions being satisfied. The running time claims follow since the running time bottleneck for this algorithm is the running time of the `FastApproximateLeverageScores` algorithm (and the running time bottleneck of that algorithm is applying the random projection), which is $O\left(nd\log(n) + d^3\log(n)\log(d)\right)$ (it is this since we have set $\epsilon = 1/2$ in the input to that algorithm).

10.3 Computing leverage scores quickly for an arbitrary matrix

A question that will arise once we consider RandNLA algorithms for low-rank matrix approximation is whether this fast leverage score approximation algorithm extends to compute the leverage scores of general "fat" matrices, i.e., matrices where both dimensions are large and we are interested in an approximation with respect to a low dimension defined by a low rank parameter.

The short answer is yes. The longer answer is that there are some subtleties. (There are also subtleties in applying random projections to "fat" matrices that we will also consider. In both cases, the subtleties have to do with the interaction between the space spanning the top part of the spectrum of the matrix and the space spanning the bottom part of the spectrum of the matrix.) Here, we will briefly describe some of those subtleties, without going into too much detail (if you want more detail, see the DMMW paper).

We have seen that, when applied to a tall matrix, random projections flatten out leverage scores to permit sparse projections or uniform sampling. It is also the case that, when applied to fat matrices, random projections uniformize things. But, what things? And, in particular, is there a notion of leverage, so we can view a random projection as preprocessing or preconditioning so uniform sampling is appropriate?

Consider a general $n \times d$ matrix A, i.e., for which $n \approx d$, and write $A = U\Sigma V^T$. In this case, U and V are in general square, and so have both orthonormal rows and columns. Thus, the definition of leverage we have presented before is uninteresting, since it is always uniform. On the other hand, if we project that matrix A onto $\ell \gtrsim k$ dimensions, where k is some explicit or implicit rank parameter, then we are really interested in the non uniformity structure on the top part of the spectrum of A.

This motivates the following definition.

Definition 1 *Given a matrix $A \in \mathbb{R}^{n\times d}$, where $n \approx d$ and $k \ll \min\{n, d\}$. Then, the* leverage scores, relative to the best rank k approximation to A *are*

$$p_i = \frac{1}{k}\left\|\left(U_k\right)_{(i)}\right\|_2^2,$$

if $A = U_k\Sigma_k V_k^T + U_k^\perp \Sigma_k^\perp V_k^\perp$ is the decomposition of A into the best rank k approximation and the residual.

Note that the way we have defined it, $\sum_{i=1}^n p_i = 1$. We could have defined quantities ℓ_i without the normalization, in which case we would have $\sum_{i=1}^n \ell_1 = \|U_k\|_F^2 = k$, which is where the k in the denominator of the equation in the definition comes from.

The basic idea of the approximate leverage score algorithm extends to this case, with the following caveat: the problem of computing the leverage scores relative to the best rank k approximation to

A is an ill-posed problem, in that a minor change in the problem input can completely change the answer. (On the other hand, if we used those perturbed leverage scores in a low-rank approximation algorithm, they would still obtain good quality of approximation bounds, but they would identify a somewhat different subspace.) To see this, consider

$$A = I_n \quad \Rightarrow \quad U_k \text{ is not unique since there are } \binom{n}{k} \text{ equivalent choices}$$

More generally, consider the matrix

$$A = \begin{pmatrix} I_k & o \\ 0 & (1-\gamma) I_{n-k} \end{pmatrix},$$

which is parameterized by a number $\gamma > 0$. As $\gamma \to 0$, it isn't possible to distinguish the top k directions.

There are two common solutions to this.

- Parameterize the problem in terms of the "spectral gap," i.e., in terms of $\gamma = \sigma_k^2 - \sigma_{k+1}^2$. This is theoretically convenient, but it is awkward and typically represents an unrealistic assumption.
- Compute the leverage for a space that is "near" the best rank k space. This is more involved, but it uses ideas from subspace iteration methods that make the connections with random projection (and in particular high-precision random projection) algorithms clearer.

Here is a definition of nearness that DMMW used in the latter approach.

Definition 2 *Given a matrix $A \in \mathbb{R}^{n\times d}$, a rank parameter $k \ll \min\{n,d\}$, let A_k be the best rank k approximation to A. Then,*

$$S_\epsilon = \{X \in \mathbb{R}^{n\times d} : \mathit{rank}(X) = k \text{ and } \|A - X\|_\xi \leq (1+\epsilon)\, \|A - A_k\|_\xi\}$$

is a set of subspaces near (in a sense quantified by $\|\cdot\|_\xi$) the best rank k approximation to A.

Here $\|\cdot\|_\xi$ represents a unitarily-invariant norm; clearly, for different norms, one can expect different quality-of-approximations.

Given this, we can define a weaker notion of leverage as follows.

Definition 3 *Call the numbers $\hat{p}_i$, for all $i \in [n]$* β-approximations to the normalized leverage scores of A_k *if there exists an $X \in S_\epsilon$ such that $\hat{p}_i \geq \frac{\beta}{k} \left\|(U_X)_{(i)}\right\|_2^2$ and $\sum_{i=1}^n \hat{p}_i = 1$, where $U_X \in \mathbb{R}^{n\times k}$ is the matrix of left singular vectors of X.*

Given this notion, we can compute approximations to these scores in "random projection time." We haven't described fat matrices and low-rank matrix approximation yet, but this time to implement a random projection can often be made (in theory and/or in practice) to be faster than an SVD or QR computation. In addition, random projections uniformize these quantities—and the extent to which these are uniformized depends on details like the number of iterations in subspace iterative methods that also determine the quality of low-rank random projection methods. We won't go into detail on these topics, but we will return to some of them below.