18 Random Projections

In this lecture we discuss random project techniques, and some of its applications. We already saw some applications within locality sensitive hashing, here we will discuss more applications towards matrix sketching.

We will again start with a $d \times n$ matrix A, or in this setting it will also be useful to think of this as a set of n points in d dimensions. Note in the previous lecture we considered an $n \times d$ matrix A, this is just a change of variables. However, now these n points are the columns of A, not the rows.

This will also relate to the *Johnson-Lindenstrauss Lemma* (and its variants), which is one of the key results about random projections and embeddings.

18.1 Random Projections

Considering a point set $A \subset \mathbb{R}^d = \{a_1, a_2, \ldots, a_n\}$ with n points. Often the dimension d can be quite large, and we want to find patterns that exist in much lower dimensional space. We will explore a curious phenomenon, that in some sense the "true" dimension of the data depends only on the number of points, not on the value d (we will see the "true" dimension is about $\ln n$).

In particular we want a mapping $\mu : \mathbb{R}^d \to \mathbb{R}^k$ (with $k \ll d$) so it compressed all of $\mathbb{R}^d$ to $\mathbb{R}^k$. And in particular for any point set $A \subset \mathbb{R}^d$ we want *all* distance preserved so that for all $a, a' \in A$

$$(1-\varepsilon)\|a - a'\| \leq \|\mu(a) - \mu(a')\| \leq (1+\varepsilon)\|a - a'\|. \tag{18.1}$$

This is different than the requirements for PCA since we want *all* distance between pairs of points preserved, whereas PCA was asking for an average error to be small – at it was the average cost of the projection to the subspace, and did not directly relate to the distance between points. This allowed some points to have large error as long as most did not.

The idea to create μ is very simple: *choose one at random*!

To create μ, we create k random unit vectors $u_1, u_2, \ldots, u_k$, then project onto the subspace spanned by these vectors. Finally we need to re-normalize by $\sqrt{d/k}$ so the expected norm is preserved.

Algorithm 18.1.1 Random Projection

for $i = 1$ **to** k **do**
  Let u_j be random element from d-dimensional Gaussian (using $d/2$ Box-Mueller transforms).
  Rescale $u_j = \sqrt{d/k} \cdot u_j / \|u_j\|$.
for each points $a_i \in A$ **do**
  for $i \in [k]$ **do**
    $q_i[j] = \langle p, u_j \rangle$ $\leftarrow j$th coordinate of q
  a_i is mapped to $q_i = (q_i[1], q_i[2], \ldots, q_i[k]) = \sqrt{d/k} \cdot \bar{\pi}_{\{u_1,\ldots,u_k\}}(a_i) = \mu(a_i) \in \mathbb{R}^k$
return Q

A classic theorem, known as the *Johnson-Lindenstrauss Lemma*, shows that if $k = O((1/\varepsilon^2)\log(n/\delta))$ in Algorithm 18.1.1 then for all $a, a' \in A$ then equation (18.1) is satisfied with probability at least $1 - \delta$. The proof can almost be seen as a Chernoff-Hoeffding bound plus Union bound. For each distance, each random projection (after appropriate normalization) gives an unbiased estimate; this requires the $1/\varepsilon^2$ term to make the difference from the unbiased estimate to be small. Then we take the union bound over all $\binom{n}{2} = O(n^2)$ distances (this yields the $\log n$ term).

Interpretation of bounds. It is pretty amazing that this bound *does not depend on* d. Moreover, it is essentially tight; that is, there are known point sets such that it requires $\Omega(1/\varepsilon^2)$ dimensions to satisfy equation (18.1).

Although the $\log n$ component can be quite reasonable, the $1/\varepsilon^2$ part can be quite onerous. For instance, if we want error to be within 1% error, we may need k at about $\log n$ times 10,000. It requires *very large* d so that setting $k = 10{,}000$ is useful.

However, we can sometimes get about 10% error (recall this is the worst case error) when $k = 200$ or so. Also the $\log n$ term may not be required if the data does actually lie in a lower dimensional space naturally (or is very clustered); this component is really a worst case analysis.

A rule of thumb: use JL when $d > 100{,}000$ *and the desired* $k > 500$.

In conclusion, this may be useful when $k = 200$ ok, and not too much precision is needed, and PCA is too slow. Otherwise, SVD or its approximations may be a better technique.

Connection to Matrix Sketching. One can also combine this with PCA ideas to get similar bounds and performance as other sketching approaches. In this setting, we create a $k \times d$ matrix S, where each d-dimensional column corresponds to an independent random project vector u_i. Then the $k \times n$ (recall we swapped meaning of n and d in this lecture) matrix B, we simply compute $B = SA$.

To preserve the approximate norm of every column of A, we again require $k = O((1/\varepsilon^2)\log(n/\delta))$. Namely with probability at least $1 - \delta$, then for each column a_i in A, then for the corresponding column b_i in B we have

$$(1 - \varepsilon)\|a_i\| \leq \|b_i\| \leq (1 + \varepsilon)\|a_i\|.$$

However, often a more powerful bound is desired. Namely that for any text vector $x \in \mathbb{R}^n$ (not even necessarily in the span of A) that

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

This requires $k = O(n/\varepsilon^2)$ to hold with constant probability. Note now the number of rows in B are larger than the number of rows in A, but the total size is independent of d. So if d was enormous and n was bounded, this can be a big win.

Extensions/Advantages. Another advantage of this technique is that μ is defined independently of A, so if we don't know A ahead of time, we can still create μ and then use it in several different cases. But if we know something of A, then again typically PCA, or Frequent Directions approaches, are better.

The initial bound by Johnson and Lindenstrauss actually required that the random project vectors were orthogonal. This was removed in a series of papers. Other variants allow for the Gaussian random variables to be replaced with (rescaled) random variables from $\{-1, 0, +1\}$ (with the 0 chosen up to $2/3$ of the time), or other "sub-Gaussian" random variables.

Another variant useful for matrix sketching, for each column of S it is all 0s, except for one entry (chosen at random) that is randomly either a -1 or $+1$ (no rescaling needed). Each row is chosen independently. This requires an increase of k to $O(n^2/\varepsilon^2)$.
Note: In that setting, this was applied on data points to reduce $n \to \ell = O(d^2/\varepsilon^2)$.