17 Random Projections and Orthogonal Matching Pursuit

Again we will consider high-dimensional data P. Now we will consider the uses and effects of randomness. We will use it to simplify P (put it in a lower dimensional space) and to recover data after random noise has interfered with it.

The first approach will be through random projections, and we will discuss the *Johnson-Lindenstrauss Lemma*, and the very simple algorithm implied by it.

Then to discuss recovery, we need to model the problem a bit more carefully. So we will define the *compressed sensing* problem. Then we will discuss the simplest way to recover data *orthogonal matching pursuit*. Although this technique does not have the best possible bounds, it is an extremely general approach that can be used in many areas.

17.1 Random Projections

We again start by considering a point set $P \subset \mathbb{R}^d = \{p_1, p_2, \ldots, p_n\}$ 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 $P \subset \mathbb{R}^d$ we want *all* distance preserved so that for all $p, p' \in P$

$$(1-\varepsilon)\|p-p'\| \leq \|\mu(p)-\mu(p')\| \leq (1+\varepsilon)\|p-p'\|. \tag{17.1}$$

This is stricter that the requirements for PCA since we want *all* distance between pairs of points preserved, whereas PCA was asking for more of an average error to be small. 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 17.1.1 Random Projection

for $i = 1$ **to** k **do**
  Let u_i be random element from d-dimensional Gaussian (using $d/2$ Box-Mueller transforms).
  Rescale $u_i = \sqrt{d/k} \cdot u_i / \|u_i\|$.
for each points $p_j \in P$ **do**
  for $i \in [k]$ **do**
    $q_i = \langle p, u_i \rangle$
  p is mapped to $q = (q_1, q_2, \ldots, q_k) = \mu(p) \in \mathbb{R}^k$
return Q

A classic theorem [1], known as the *Johnson-Lindenstrauss Lemma*, shows that if $k = O((1/\varepsilon^2)\log(n/\delta))$ in Algorithm 17.1.1 then for all $p, p' \in P$ then equation (17.1) is satisfied with probability at least $1-\delta$. The proof can almost be seen as a Chernoff-Hoeffding bound plus Union bound, see **L3.2**. 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 17.1.

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. Its not often that d is large enough 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.

Extensions/Advantages. One can also combine this with PCA ideas [2] to get similar bounds and performance as in **L16**.

Another advantage of this technique is that μ is defined independently of P, so if we don't know P ahead of time, we can still create μ and then use it in several different cases. But if we know something of P, then again typically PCA is better.

Typically, the random Gaussian vector u_i can also be replaced with a random vector in $\{-1, 0, +1\}^d$.

17.2 Compressed Sensing

The compressed sensing problem is as follows. The data is S, a *sparse* d-dimension vector; that is, it only has m non-zero entries for $m \ll d$.

For example, let

$$S^T = [0\,1\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,1\,1\,0\,1\,0\,0\,0\,1\,0\,0\,1\,0\,0]$$

for $d = 32$ and $m = 8$. (Perhaps in practice the non-zeros could be larger, and the "zeros" may be small, such as < 0.05.)

Now the goal is to make only $N = K \cdot m \log(d/m))$ (random) measurements of S and recover it exactly (or with high probability). In some settings K is 4, not more than 20 in general. For this to work, each measurement needs to be of (nearly) the entire matrix, otherwise we can miss a non-zero and just never witness it.

Let a measurement x_i be a random vector in $\{-1, 0, +1\}^d$. Example:

$$x_i^T = [\text{-}1\,0\,1\,0\,1\,1\,\text{-}1\,1\,0\,\text{-}1\,0\,0\,1\,\text{-}1\,\text{-}1\,1\,0\,1\,0\,1\,\text{-}1\,\text{-}1\,\text{-}1\,0\,1\,0\,0\,\text{-}1\,0\,1\,0\,0]$$

and the result of the measurement on S is

$$y_i = \langle S, x_i \rangle = 0\text{+}0\text{+}0\text{+}0\text{+}1\text{+}0\text{+}0\text{+}0\text{+}0\text{+}0\text{+}0\text{+}0\text{+}0\text{+}0\text{+}0\text{+}1\text{+}0\text{+}0\text{+}0\text{+}1\text{-}1\text{+}0\text{-}1\text{+}0\text{+}0\text{+}0\text{+}0\text{+}0\text{+}0\text{+}1\text{+}0\text{+}0 = 2.$$

In general we will have a $d \times N$ measurement matrix $X = [x_1; x_2; \ldots; x_N]$ and a measurement $y = XS$. The measurement vector y takes about $N \log N \approx m \log(d/m) \log m$ space, which is much less than S. Since it requires $\log d$ bits just to store the location of each non-zero in S, this is really only an extra $\log m$ space factor over just storing the non-zero bits in S. Hence the great compression.

Examples:

- *single pixel camera:* Instead of 10 Gigapixels (about 25MB), directly sense the 5MB jpg. This is hard, but we can get kind of close. Take N measurements where each y_i is the *sum* of all 10 Gigapixels with a random mask x_i. Each pixel is either taken in "as is" (a $+1$), is blocked (a 0), or is subtracted (a -1).
  Such cameras have been built - they work ok, not as well as regular camera :).
- *Hubble Telescope:* High resolution camera in space (less atmospheric interference). But communication to/from space is expensive. So with fixed (but initial random mask matrix X, that is already know on Earth), can send compressed signals down.
- *MRI on kids:* They squirm a lot. So few angles voxels need to be sensed and this technique gets the best images available on kids. Not as high resolution as on full MRI, but with much much less time.
- *Noisy Data:* Data is often noisy, and have more attributes than actually there. This helps find the true structure. See more next lecture.

17.3 Orthogonal Matching Pursuit (OMP)

This is one of the simplest ways to recover. It is simple and greedy (with some chance to recover). Its like a discrete L_1 version of the technique for SVD, and can be useful for many other hard optimization problems. We assume we know the measurement $(d \times N)$ matrix X, and the N measurements y.

- First, find the measurement column X_j (not the row x_i used to measure).

$$X_j = \arg \max_{X_{j'} \in X} |\langle y, X_{j'} \rangle|.$$

  This represents the single index of S that explains the most about y.

- Next we find the weight

$$\gamma = \arg \min_{\gamma \in \mathbb{R}} \|y - X_j \gamma\|$$

  that represents our guess of entry s_j in S. If S is always 0 or 1, then we may enforce that $\gamma = 1$.

- Finally, we calculate the residual $r = y - X_j\gamma$. This is what remains to be explained by other elements of S.

- Then we repeat for t rounds. We stop when the residual is small enough (nothing left to explain) or γ is small enough (the additional explanation is not that useful).

Algorithm 17.3.1 Orthogonal Matching Pursuit

Set $r = y$.
for $i = 1$ **to** t **do**
  Set $X_j = \arg\max_{X_{j'} \in X} |\langle r, X_{j'} \rangle|$.
  Set $\gamma_j = \arg\min_\gamma \|r - X_j\gamma\|$.
  Set $r = r - X_j\gamma_j$.
Return $\hat{S}$ where $\hat{s}_j = \gamma_j$ (or 0).

Remarks

- Can add a regularization term into loss function

$$X_j = \arg\min_{\gamma} \|r - X_j\gamma\| + \alpha\|\gamma\|$$

  and, as we will see next lecture, this will bias towards sparse solutions.

- Can re-solve for optimal least squares to get better estimate each round, but more work.

$$\gamma_{1,\ldots,i} = \arg\min_{\gamma_1,\ldots,\gamma_i} \|y - [X_{j_1}, \ldots, X_{j_i}]\gamma_{1,\ldots,i}\|$$
$$r = r - [X_{j_1}, \ldots, X_{j_i}]\gamma_{1,\ldots,i}$$

- This *converges* if in each step we restrict $\|r_i\| < \|r_{i-1}\|$. A *Frank-Wolfe* analysis can show that its within ε of optimal after $t = 1/\varepsilon$ steps. Although it may not be a global optimum.

- Term "orthogonal" comes since each X_{j_i} in the ith step is always *linear independent* of $[X_{j_1} \ldots X_{j_{i-1}}]$. Adds an orthogonal explanation of y.

- Roughly, the analysis of why $d\log(m/d)$ measurements is through the Coupon Collectors since we need to hit each of the d measurements. And since X is random and N is large enough, then each $\langle X_j, X_{j'}\rangle$ (for $j \neq j'$) should be small (they are close to orthogonal).

17.3.1 Orthogonal Matching Pursuit Example

We now consider a specific example for running Orthogonal Matching Pursuit, this has $d = 10$, $m = 3$ and $N = 6$. Let the (unknown) input signal be

$$S = [0, 0, 1, 0, 0, 1, 0, 0, 1, 0].$$

Let the measurement matrix by

$$X = \begin{bmatrix} 0 & 1 & 1 & -1 & -1 & 0 & -1 & 0 & -1 & 0 \\ -1 & -1 & 0 & 1 & -1 & 0 & 0 & -1 & 0 & 1 \\ 1 & -1 & 1 & -1 & 0 & -1 & 1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 1 & -1 & -1 & 1 & 1 \\ -1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & -1 \\ 0 & 0 & -1 & -1 & -1 & 0 & -1 & 1 & -1 & 0 \end{bmatrix}$$

so for instance the first row $x_1 = (0, 1, 1, -1, -1, 0, -1, 0, -1, 0)$ yields measurement $\langle S, x_1\rangle = 0 + 0 + 1 + 0 + 0 + 0 + 0 + 0 + (-1) + 0 = 0$.

The observed measurement vector is

$$y = XS^T = [0, 0, 0, 1, 1, -2]^T.$$

Columns 7 and 9 have the most explanatory power towards y, based on X. We let $j_1 = 9$. $X_{j_1} = X_9 = (-1, 0, 0, 1, 1, -1)^T =$ `X(:,9)`. Then $1 = \gamma_1 = \arg\min_\gamma \|y - X_9\gamma\|$. We can then set $r = y - X_9 * \gamma_1 = (1, 0, 0, 0, 0, -1)$.

Next, we observe that columns 3, 4, 5, 7, and 9 have the most explanatory power for the new r. We choose 3 arbitrarily, letting $j_2 = 3$. Let $X_{j_2} = X_3 = (1, 0, 1, -1, 0, -1)^T =$ `X(:,3)`. Then $1 = \gamma_2 = \arg\min_\gamma \|r - X_3\gamma\|$ (actually any value in the range $[0, 1]$ will give same minimum). Using γ_2 we update

$r = r - X_3 * \gamma_2 = (0, 0, -1, 1, 0, 0)$. *Note: This progress seemed sideways at best. It increased our non-zero γ_i values, but did not decrease $\|r - y\|$.*

Finally, we observe columns 1, 3, 6, 7, and 8 have the most explanatory power of the new r. We choose 6 arbitrarily, let $j_3 = 6$. *Note: we could have chosen 3, and then gone an updated our choice of γ_2.* Let $X_{j_3} = X_6 = (0, 0, -1, 1, 0, 0)^T$ = `X(:,6)`. Then $1 = \gamma_3 = \arg\min_\gamma \|r - X_6\gamma\|$. Then using γ_3 we update $r = r - X_6\gamma_3 = (0, 0, 0, 0, 0, 0)$. So we have completely explained y using only 3 data elements.

Remarks:

- This would not have worked so cleanly if we made other arbitrary choices. Using OMP typically needs something like $N = 20 \times m \log d$ measurements (instead of 6). Large measurements would have made it much more likely that at each step we chose the correct variable j_i as most explanatory.
- This still will not always converge to the correct solution. It might get stuck without explaining everything exactly. In that case, we can often guess we still get a good enough explanation (although slightly off) and leave it at that. With much larger d and m, getting a good guess of the m non-zero bits might still be useful. There are other more complex minimization techniques we can alternatively consider in the next lecture.

Bibliography

[1] William B. Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary Mathematics*, 26(189-206):1, 1984.

[2] Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 143–152. IEEE, 2006.