Contents

In `syo.pdf`, we mentioned that our random projection matrix was a map which mapped $\vec{x} \in \mathbb{R}^m$ to vectors $\vec{v} \in \mathbb{R}^k$. Hence we can form a matrix $X = [\vec{x}_1|\vec{x}_2|\ldots|\vec{x}_n]$, and compute $V = \frac{1}{\sqrt{k}}RX$, where $V = [\vec{v}_1|\vec{v}_2|\ldots|\vec{v}_n]$.

Using matrix-vector notation, each vector $\vec{x}$ is assumed to be a column vector, for the ordinary $\vec{v} = R\vec{x}$. Moreover, we used the index m to avoid using p (for l_p distances).

However: usually we are given data X encoded in matrices of size $n \times p$, where each observation is a row. Thus, if we wanted to have "rows" of matrices corresponding to observations, then we would have to compute $V = (\frac{1}{\sqrt{k}}RX^T)^T$, where T denotes the transpose.

This is messy, but we can also compute the equivalent: $V = \frac{1}{\sqrt{k}}XR$, where $X_{n\times p}$, and $R_{p\times k}$, giving $V_{n\times k}$, where each row $\vec{v}_i, 1 \leq i \leq n$ in V corresponds to the transformed vector $\vec{x}_i, 1 \leq i \leq n$.

We will also use p here to denote number of parameters, rather than m (since l_p distances do not appear in this context).

0.1 `[R_struct] = generate_normal_R(X_struct, k)`

We create a structure `R_struct` with two fields.

- `R_struct.R_mat`: the matrix $R_{p\times k}$ with entries is i.i.d. $N(0, 1)$.
- `R_struct.scaling_factor`: the scaling factor is set to k.

0.2 `[V_struct] = compute_V_for_random_projection(X_struct, R_struct)`

This code creates a structure `V_struct` with three fields.

- `V_struct.V_mat`: the matrix $V_{n\times k} = X_{n\times p}R_{p\times k}$.
- `V_struct.scaling_factor`: the scaling factor such that when the Euclidean distance of any two rows of V (or inner product of any two rows of V) is computed, the scaling factor is then used to scale the computed value to be an estimate of the Euclidean distance (or inner product) of the respective rows of X.
- `V_struct.num_obs`: the number of observations (or rows) of this matrix.

0.3 [estimated_dist_mat] = estimate_pairwise_ordinary_ED_for_random_projections(V_struct)

This code computes the estimated pairwise Euclidean distance of the matrix V, and then scales it by the scaling factor in `V_struct`.

0.4 [estimated_dist_mat] = estimate_pairwise_ordinary_IP_for_random_projections(V_struct)

This code computes the estimated pairwise dot product of the matrix V, and then scales it by the scaling factor in `V_struct`.

0.5 [Estimated_Array] = create_array_for_cumsum(X_struct, V_struct, kvals, typeof)

This code is used in `compare_generic_distance_simulation.m` to compute the estimated values of random projections for different values of k.

The motivation for this code is that the user generally wants to see (empirically) how the error varies with the dimension $\mathbb{R}^k$ the initial vectors $\mathbb{R}^p$ are projected to. For example, if the user had set `kvals = [5 10 15 20 ... 100]`, the user would project the vectors down to $\mathbb{R}^5$, $\mathbb{R}^{10}, \ldots, \mathbb{R}^{100}$ and compare the error in the estimated pairwise distances for each dimensions.

One naive way to do this is to consider the following loop

for k in `kvals` **do**
  Generate random matrix $R_{p\times k}$
  Compute $V_{n\times k} = X_{n\times p}R_{p\times k}$
  Compute distance of interest and scale by k in the end
  Find error
end

Algorithm 1: Naive loop

In total, we can see this as generating $5 + 10 + 15 + \ldots + 100$ number of columns of random variables.

However, since our variables are i.i.d, a better way would be to consider this loop.

Generate random matrix $R_{p\times 100}$
for k in `kvals` **do**
  Set `current_R` $= R_{p,1:\mathtt{k}}$
  Compute $V = X *$ `current_R`
  Compute distance of interest and scale by k in the end
  Find error
end

Algorithm 2: Better loop

Here, we are only generating our random variables **once**, as we are subsetting from this big matrix. But we can do better as there are still some redundant computations.

Recall from `syo.pdf` that our estimate of distance (eg, squared Euclidean distance) between $\vec{x}_1, \vec{x}_2$ is given by

$$\frac{(y_{11} - y_{21})^2 + (y_{12} - y_{22})^2 + \ldots + (y_{1k} - y_{2k})^2}{k}$$

For `kvals = [5 10 15 20 ... 100]`, suppose we compute the set $I_1, I_2, \ldots, I_{20}$ to be indices, where $I_1 = \{1, 2, 3, 4, 5\}$, $I_2 = \{6, 7, 8, 9, 10\}, \ldots, I_{20} = \{96, 97, 98, 99, 100\}$. Here, we choose the sets based on the divisions in `kvals`.

Then we compute the following sums (along the indices $I_1, \ldots, I_{20}$).

$$\begin{aligned} S_1 &= (y_{11} - y_{21})^2 + (y_{12} - y_{22})^2 + \ldots + (y_{15} - y_{25})^2 \\ S_2 &= (y_{16} - y_{26})^2 + (y_{17} - y_{27})^2 + \ldots + (y_{1,10} - y_{2,10})^2 \\ &\vdots \\ S_{20} &= (y_{1,96} - y_{2,96})^2 + (y_{1,97} - y_{2,97})^2 + \ldots + (y_{1,100} - y_{2,100})^2 \end{aligned}$$

This leads us to a better loop:

Generate random matrix $R_{p\times 100}$
Initialize `sum` $= 0$
Compute indices $I_1, I_2, \ldots, I_K$
for `index_num` in `1:K` **do**
  Set `current_R` $= R_{p,\mathtt{I\_index\_num}}$
  Compute $V = X *$ `current_R`
  Set $S_{\mathtt{index\_num}} =$ distance of interest.
  Set `sum` $=$ `sum` $+ S_{\mathtt{index\_num}}$
  Scale `sum` by $|I_1 + \ldots + I_{\mathtt{index\_num}}|$
  Find error of estimate given by `sum`
end

Algorithm 3: Even better loop

The code in `create_array_for_cumsum.m` does this, but for all pairwise distances, not just

distances between a vector pair $\vec{y}_1, \vec{y}_2$. `cumsum` is used in Matlab to minimize loops. All unnecessary computations are minimized.

The user can choose `typeof` to range from comparing the random projection estimates for Euclidean distance, squared Euclidean distance, and the dot product.

More information can be found in the function comments.

0.6 `[results_mat] = compare_generic_distance_simulation(X_struct, kvals, numiter, typeof)`

This allows the user to compare the RMSE (root mean square error) of all pairwise distances in a specific data matrix inside the structure `X_struct`. This makes use of functions in the folder `actual_distances` as well.

The user can choose inputs

1. `kvals` (which is a vector of increasing values)
   to compare the RMSE of all pairwise distances of varying dimensions $\mathbb{R}^k$.
2. `numiter`, which is the number of iterations this experiment is repeated.
3. `typeof`, which is the type of distance comparison to be done. Valid distances are
   (a) `squared_euclidean_distance`
   (b) `euclidean_distance`
   (c) `dot_product`

The output is a matrix of `numiter` rows and `length(kvals)` columns. Each element in the matrix is the RMSE of all pairwise estimates for that particular iteration and that particular dimension projected down to.