Prelude

Part of my research focuses on random projections, which involves running simulations and evaluating the performance of algorithms using random projections (and other random algorithms) on specific datasets.

I intend this Github folder to serve several purposes:

1. Make code I use in my research public, to encourage reproducible research (or help spot annoying tpyos)
2. Give some motivation to why I do this research
3. Encourage potential students (undergraduates, PhDs) to do similar research

Contents

1 Data and Distances

Most data can be represented as a vector in $\mathbb{R}^D$, where D is some positive integer. Here are some examples.

1. **Book data**

   We could record the width, thickness, and height of a book as a vector in $\mathbb{R}^3$.

   For example, the hard copy of `Simulation, 4th Edition` by [Ross, 2006] in my book collection has a width of 15.6cm, 2.2cm, and height of 23.5cm. This could be recorded as the vector $(15.6, 2.2, 23.5) \in \mathbb{R}^3$.

2. **Pictures of objects**

   This is a picture of a cat, which has dimensions 1000×700 pixels. Each pixel can be represented as a vector in $\mathbb{R}^3$ (storing colour intensity in red, green or blue). Hence this picture can be represented as a vector in $\mathbb{R}^{2100000}$, where $2100000 = 1000 \times 700 \times 3$.

Figure 1: Cat picture from AP/Nestle Purina PetCare

3. **Exam data**

   For a mathematics exam consisting of N questions, we could store a student's score for the exam as a vector of $\mathbb{R}^N$, where the i^{th} element of the vector, $i = 1, \ldots, N$ consists of the student's score for the i^{th} question.

The types of data above are all examples of continuous data, and this is the type of data I currently look at.

In most cases, we can use the notion of distance to do some kind of learning on the data.

1.1 k-nearest neighbours

Example 1.1. *Finding which group an unknown vector belongs to*
Suppose we have vectors (data) in $\mathbb{R}^2$, and we know beforehand whether they belong to Group 1, or Group 2. Suppose further we have an unknown vector in $\mathbb{R}^2$, and we want to figure out whether it belongs to Group 1 or Group 2.

Since the vectors are in $\mathbb{R}^2$, we can plot them as follows in Figure 2. Which group do you think the unknown vector (represented by ∇) belongs to - Group 1 or Group 2?

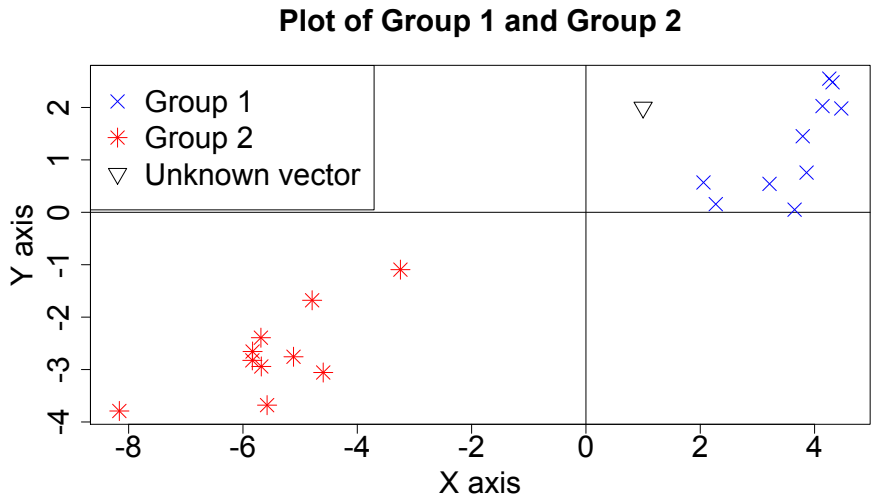

Figure 2: Toy example of two groups and one unknown vector.

Based on the plot, we might reason thus: "*Well, the black triangle is closer to all the* $\times$ *than all the* $*$*, so we say that the unknown vector belongs to Group 1.*"

However, when we reasoned "closer", what we really meant was that the Euclidean distances[1] of our unknown point to the $\times$ points were much shorter than the Euclidean distances of our unknown point to the $*$.

We might be interested in extending this example to higher dimensions $\mathbb{R}^D$ where $D > 2$, but as we[2] cannot visualize plots in 4 dimensions or more, it is useful to look at our 2D example

[1] It is also possible that you might have looked at the angular distance, i.e. the angle between the vector representing our unknown point and the vector of a point in Group 1 / Group 2 rather than the Euclidean distance- that is okay as well.

[2] most humans

in Example 1.1 to come up with some heuristic to classify vectors in higher dimensions.

This gives rise to the k-nearest neighbours algorithm which works like this.

Require vectors $\vec{x}_1^{(1)}, \ldots, \vec{x}_{n_1}^{(1)}$ belonging to Group 1
Require vectors $\vec{x}_1^{(2)}, \ldots, \vec{x}_{n_2}^{(2)}$ belonging to Group 2
Choose a value of k
Have vectors $\vec{x}_1, \ldots, \vec{x}_m$ to classify
for each $\vec{x}_i, 1 = 1, 2, \ldots, m$ **do**
  Compute and store the $n_1 + n_2$ `distances` between $\vec{x}_i$ and the $n_1 + n_2$ vectors $\vec{x}_1^{(1)}, \ldots, \vec{x}_{n_1}^{(1)}, \vec{x}_1^{(2)}, \ldots, \vec{x}_{n_2}^{(2)}$
  Pick the k shortest distances
  Out of the k shortest distances, tabulate the number of vectors belonging to Group 1 which $\vec{x}_i$ is closest to, and the number of vectors belonging to Group 2 which $\vec{x}_i$ is closet to
  Classify $\vec{x}_i$ based on majority vote
end

Algorithm 1: k nearest neighbours algorithm

1.2 Types of Distances ...

There are many other types of distances that we can use apart from the Euclidean distance or angular distance. For example, here is a (non-exhaustive) list of certain types of distances which we can compute

1. Angular distance
2. Dot product
3. Euclidean distance
4. Hamming distance
5. Jaccard similarity
6. l_1 distance
7. l_p distance
8. l_∞ (Chebyshev) distance
9. Resemblance

There is Matlab code in the folder `actual_distances` that computes these distances.

More precisely, suppose we have $N := n_1 + n_2$ vectors in $\mathbb{R}^p$, where n_1 vectors belong to one group, and n_2 vectors belong to another group. Then we can create matrices X_1 of size $n_1 \times p$, and X_2 of size $n_2 \times p$, where the row vectors of X_1 correspond to vectors in the first group, and the row vectors of X_2 correspond to vectors in the second group.

This code in the folder takes in as input the matrices X_1, X_2, a certain distance type, optional parameters corresponding to the distance type, and outputs a distance structure consisting of a distance matrix D of size $n_1 \times n_2$.

The $(i, j)^{\text{th}}$ element in this distance matrix D corresponds to the distance between the i^{th} row of X_1 and the j^{th} row of X_2.

Here are some example inputs and outputs:

```
>X1 = binornd(1,0.5,10,50);  % randomly generate X1
>X2 = binornd(1,0.5,30,50);  % randomly generate X2

% computes l_4 distance between pairs
>get_pairwise_distances(X1, X2, 'lp_distance', 4)

ans =

  struct with fields:

     dist_mat: [10 x 30 double]
    dist_type: 'lp_distance'
       dist_p: 4


>get_pairwise_distances(X1,X2,'hamming_distance')

ans =

  struct with fields:

     dist_mat: [10 x 30 double]
    dist_type: 'hamming_distance'
```

You can see from the code that we output a distance structure rather than just a distance

matrix. This makes it easier for debugging (for example, which distance did we choose to compute?)

1.3 ... and techniques requiring them

Surprisingly, a lot of statistical techniques use the concept of distances. Some brief examples

1. Least squares / projection (normed distance)
2. Linear / Quadratic discriminant analysis (Euclidean distance, Mahalanobis distance)
3. Support vector machines (inner products)
4. Similarity search (Hamming distance, Jaccard similarity, resemblance)

We even use the concept of distance in hypothesis testing as well.

1.4 Where does the research come in?

We could imagine new machine learning techniques to work as such:

1. We throw in data into a black box machine learning algorithm
2. Distances between data are computed
3. Magic happens
4. Machine learning algorithm makes some prediction

and hence there is nothing new to be done (apart from coming up with new algorithms).

However, there are some points to ponder about.

1. *Speed of computation of these distances*

   One can imagine that as we store more and more data, i.e, we store vectors of size $\mathbb{R}^D$ when D is large, computing distances between two vectors will take time proportional to D.

   For example, computing the angular distance between two vectors in $\mathbb{R}^2$ would be much faster than computing the angular distance between two vectors in $\mathbb{R}^{1000000000}$.

   A machine learning algorithm might take drastically more time to run with bigger vectors.

2. **'Storage' of vectors**

   In the above example, we might have had vectors in $\mathbb{R}^{1000000000}$. But suppose we also had lots of these vectors (eg, imagine the number of all cat pictures N on the internet). N and D might be large, and we may not have enough space to store them in memory.

   However, if we aim to do some prediction, and a machine learning algorithm only uses the computed distances for prediction, then we don't really need these original vectors, just the distances between vectors. Maybe there might be a better way to store these vectors?

3. **Privacy**

   People might be concerned about data breaches. Similar to the 'storage' of vectors point above, we might be interested in a way to store these vectors such that we store the "distances" between vectors for any kind of analysis, yet do not retain the original vectors which might store sensitive information.

Random projections (and locality-sensitive hashing) types of algorithms are dimension reduction algorithms, reducing the size of data from dimension D to a smaller dimension k. Distances are preserved under such algorithms in expectation, and the relative error of these distances usually depend on the smaller dimension k, rather than the initial dimension D.

This means - if today we work with vectors in $\mathbb{R}^D$ where $D = 1,000,000$, and we reduce the size of the vectors to $k = 100$ dimensions, we may have a certain relative error E when computing pairwise distances. Tomorrow, if we work with vectors in $\mathbb{R}^D$ where $D = 1,000,000,000$ and we reduce the size of the vectors to $k = 100$ dimensions as well, our relative error is still about the same E.

So the initial large dimension D does not matter. Only the smaller dimension k matters!

My research focuses on **further reducing the relative error of the computation of distances** for such algorithms, and looking at the effects of this reduction with several machine learning algorithms.

2 Construction of Random Projections

Suppose we had vectors $\vec{x}_1, \vec{x}_2 \in \mathbb{R}^m$, and we are interested in some distance D between the two vectors, which we can compute by some function[3] $D = d(\vec{x}_1, \vec{x}_2)$. Some distances we may be interested in include

1. Euclidean distance, given by

$$\begin{aligned} d_{\text{Euclidean distance}}(\vec{x}_1, \vec{x}_2) &:= \sqrt{(x_{11} - x_{21})^2 + \ldots + (x_{1m} - x_{2m})^2} \\ &= \sqrt{\|\vec{x}_1\|^2 - 2\vec{x}_1^T\vec{x}_2 + \|\vec{x}_2\|^2} \end{aligned}$$

2. Dot product, given by

$$\begin{aligned} d_{\text{Dot product}}(\vec{x}_1, \vec{x}_2) &:= x_{11}x_{21} + \ldots + x_{1m}x_{2m} \\ &= \vec{x}_1^T\vec{x}_2 \end{aligned}$$

3. Angular distance, given by

$$d_{\text{Angular distance}}(\vec{x}_1, \vec{x}_2) := \arccos\left(\frac{\vec{x}_1^T\vec{x}_2}{\|\vec{x}_1\|\|\vec{y}_2\|}\right)$$

4. l_p distance, given by

$$d_{l_p \text{ distance}}(\vec{x}_1, \vec{x}_2) := ((x_{11} - x_{21})^p + \ldots + (x_{1m} - x_{2m})^p)^{\frac{1}{p}}$$

where we denote

$$\|\vec{x}\| := x_1^2 + \ldots + x_m^2$$

Consider the number of operations taken to compute any of these above distances. For example, to compute the Euclidean distance, we have to do

- One "subtract" operation, and one "square" operation for each $(x_{1i} - x_{2i})^2$.
- $m - 1$ "adding" operations to sum up all m terms
- One "square root" operation

[3] This is not necessarily a metric. For example, angular distance.

which gives us $2m + m - 1 + 1 = 3m$ operations. So the number of operations we need to do is proportional to m, which is the dimension of the vectors $\vec{x}_1, \vec{x}_2$. One can expect that as m gets large, we need to spend about $O(m)$ of time computing such distances.

That's just for one pair of vectors though. Suppose we have vectors $\vec{x}_1, \ldots, \vec{x}_n$, and we might want to compute all pairwise distances $d(.\ ,\ ,.)$ between them. There's $\binom{n}{2}$ such distances, so in total we might spend about $\frac{n!}{(n-2)!2!}(3m)$ computations, which works out to about $\frac{3n(n-1)m}{2}$ operations, or something which takes about $O(n^2 m)$ of time.

The goal here is to speed up the time taken in computing such distances.

We consider the following steps

1. "Hope" that we can find some map given by $T : \mathbb{R}^m \to \mathbb{R}^k$, where we map vectors from a high dimensional space to a low dimensional space, with $k \ll m$.
2. Instead of computing distances $d(\vec{x}_i, \vec{x}_j)$ which take $O(m)$ of time, we find some (other) function $\widetilde{d}(T\vec{x}_i, T\vec{x}_j)$ which approximates the distance $d(\vec{x}_i, \vec{x}_j)$, i.e

   $$\widetilde{d}(T\vec{x}_i, T\vec{x}_j) \approx d(\vec{x}_i, \vec{x}_j)$$

   which takes only $O(k)$ time to compute.

We can represent such maps T by a random matrix R, where each entry of R is generated from some probability distribution. Figure 3 shows how this works.

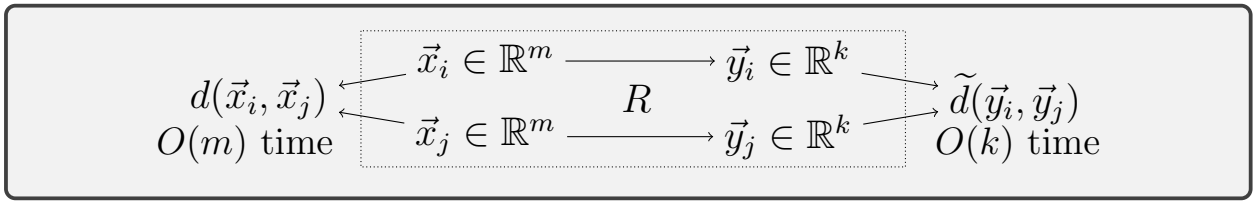

Figure 3: Estimation process for $\widetilde{d}(\vec{y}_i, \vec{y}_j) \approx d(\vec{x}_i, \vec{x}_j)$ where $T\vec{x} = \vec{y}$. The "dotted box" shows what happens for " $\approx$ ".

2.1 An example of R (using $N(0,1)$)

Suppose we look at two measures of distances, the Euclidean distance given by

$$d_1(\vec{x}_1, \vec{x}_2) := \sqrt{\|\vec{x}_1\|^2 - 2\vec{x}_1^T\vec{x}_2 + \|\vec{x}_2\|^2}$$

and the dot product given by

$$d_2(\vec{x}_1, \vec{x}_2) := \vec{x}_1^T\vec{x}_2$$

Consider the map $T : \mathbb{R}^m \to \mathbb{R}^k$ represented by a random matrix $R_{k\times m}$, where each r_{ij} entry in R is i.i.d. $N(0,1)$.

The goal is to find some function $\widetilde{d}_i(R\vec{x}_1, R\vec{x}_2)$ that approximates $d_i(\vec{x}_1, \vec{x}_2)$, for $i = 1, 2$.

2.1.1 Finding estimates of the Euclidean distance and dot product between vector pairs

Let's consider the matrix multiplication of $R\vec{x}_1, R\vec{x}_2$. We have that

$$\begin{pmatrix} r_{11} & r_{12} & \dots & r_{1m} \\ r_{21} & r_{22} & \dots & r_{2m} \\ \dots & \dots & \ddots & \dots \\ r_{k1} & r_{k2} & \dots & r_{km} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{1m} \end{pmatrix} = \begin{pmatrix} r_{11}x_{11} + \ldots + r_{1m}x_{1m} \\ r_{21}x_{11} + \ldots + r_{2m}x_{1m} \\ \vdots \\ r_{k1}x_{11} + \ldots + r_{km}x_{1m} \end{pmatrix} = \begin{pmatrix} \sum_{s=1}^m r_{1s}x_{1s} \\ \sum_{s=1}^m r_{2s}x_{1s} \\ \vdots \\ \sum_{s=1}^m r_{ks}x_{1s} \end{pmatrix} = \begin{pmatrix} y_{11} \\ y_{12} \\ \vdots \\ y_{1k} \end{pmatrix}$$

and similarly, we have that

$$\begin{pmatrix} r_{11} & r_{12} & \dots & r_{1m} \\ r_{21} & r_{22} & \dots & r_{2m} \\ \dots & \dots & \ddots & \dots \\ r_{k1} & r_{k2} & \dots & r_{km} \end{pmatrix} \begin{pmatrix} x_{21} \\ x_{22} \\ \vdots \\ x_{2m} \end{pmatrix} = \begin{pmatrix} r_{11}x_{21} + \ldots + r_{1m}x_{2m} \\ r_{21}x_{21} + \ldots + r_{2m}x_{2m} \\ \vdots \\ r_{k1}x_{21} + \ldots + r_{km}x_{2m} \end{pmatrix} = \begin{pmatrix} \sum_{s=1}^m r_{1s}x_{2s} \\ \sum_{s=1}^m r_{2s}x_{2s} \\ \vdots \\ \sum_{s=1}^m r_{ks}x_{2s} \end{pmatrix} = \begin{pmatrix} y_{21} \\ y_{22} \\ \vdots \\ y_{2k} \end{pmatrix}$$

For our vector $\vec{y}_i = (y_{i1}, y_{i2}, \ldots, y_{ik}), i = 1, 2$, we consider each entry $y_{is}, s = 1, \ldots, k$ as an i.i.d. observations coming from some probability distribution. In fact, each y_{is} is a weighted sum of m random $N(0,1)$ random variables, with the weights given by $x_{i1}, x_{i2}, \ldots, x_{im}$.

Consider $\mathbb{E}[y_{i1}]$. We must have

$$\begin{aligned} \mathbb{E}\left[\sum_{s=1}^m r_{1s}x_{is}\right] &= \mathbb{E}\left[r_{11}x_{i1} + r_{12}x_{i2} + \ldots + r_{1m}x_{im}\right] \\ &= x_{i1}\mathbb{E}[r_{11}] + x_{i2}\mathbb{E}[r_{12}] + \ldots + x_{im}\mathbb{E}[r_{1m}] \\ &= x_{i1} \times 0 + x_{i2} \times 0 + \ldots + x_{im} \times 0 \\ &= 0 \end{aligned}$$

and by independence, each $y_{i1}, y_{i2}, \ldots, y_{ik}$ have a mean of zero for $i = 1, 2$.

Suppose we square our observations, and look at $y_{i1}^2, y_{i2}^2, \ldots, y_{ik}^2$. Each squared observations are also i.i.d. from some distribution. We consider $\mathbb{E}[(y_{i1})^2]$, and must have

$$\begin{aligned}
\mathbb{E}\left[\left(\sum_{s=1}^{m} r_{1s}x_{is}\right)^2\right] &= \mathbb{E}\left[\sum_{s=1}^{m} r_{1s}^2 x_{is}^2 + 2\sum_{0\le s<t}^{m} r_{1s}x_{is}r_{1t}x_{it}\right] \\
&= \sum_{s=1}^{m} \mathbb{E}[r_{1s}^2]x_{is}^2 + 2\sum_{0\le s<t}^{m} \mathbb{E}[r_{1s}]\mathbb{E}[r_{1t}]x_{is}x_{it} \\
&= \sum_{s=1}^{m} 1 \cdot x_{is}^2 + 2\sum_{0\le s<t}^{m} (0)(0)x_{is}x_{it} \\
&= \sum_{s=1}^{m} x_{is}^2 \\
&= \|\vec{x}_i\|^2
\end{aligned}$$

Since the mean of $(y_{i1})^2$ is equal to $\|\vec{x}_i\|^2$, then as each $(y_{i1})^2, (y_{i2})^2, \ldots, (y_{ik})^2$ are i.i.d., we have that

$$\begin{aligned}
\mathbb{E}\left[\|\vec{y}_i\|^2\right] &= \mathbb{E}\left[y_{i1}^2 + y_{i2}^2 + \ldots + y_{ik}^2\right] \\
&= \mathbb{E}\left[\sum_{s=1}^{k} y_{is}^2\right] \\
&= k\|\vec{x}_i\|^2
\end{aligned}$$

This result implies that

$$\begin{aligned}
\mathbb{E}\left[\|\vec{y}_1 - \vec{y}_2\|^2\right] &= \mathbb{E}\left[(y_{11} - y_{21})^2 + (y_{12} - y_{22})^2 + \ldots + (y_{1k} - y_{2k})^2\right] \\
&= \mathbb{E}\left[\sum_{s=1}^{k} (y_{1s} - y_{2s})^2\right] \\
&= k\|\vec{x}_1 - \vec{x}_2\|^2
\end{aligned}$$

Hence, if we are interested in some $\widetilde{d}_1(\vec{y}_1, \vec{y}_2) \approx d_1(\vec{x}_1, \vec{x}_2)$, we can set

$$\widetilde{d}_1(\vec{y}_1, \vec{y}_2) \approx \sqrt{\frac{d_1(\vec{y}_1, \vec{y}_2)}{k}}$$

By considering the mean of $y_{1s}y_{2s}, s = 1, \ldots, k$, and repeating the process above, we also get an expression

$$\widetilde{d}_2(\vec{y}_1, \vec{y}_2) \approx \frac{d_2(\vec{y}_1, \vec{y}_2)}{k}$$

In practice, one can form a matrix $X = [\vec{x}_1|\vec{x}_2|\ldots|\vec{x}_n]$ by concatenating columns of vectors in $\mathbb{R}^m$. The matrix product $V = \frac{1}{\sqrt{k}}RX$ is computed, where R is of size $k \times m$ with entries i.i.d. $N(0,1)$.

Computing the Euclidean distance (or dot product) between any two columns i, j of V gives an estimate of the Euclidean distance (or dot product) between $\vec{x}_i, \vec{x}_j$.

This is the original random projection algorithm [Indyk and Motwani, 1998] which has been used since the late 1990s.

While we can see that $(y_{1s} - y_{2s})^2$ is an unbiased estimator for the squared Euclidean distance between $\vec{x}_1, \vec{x}_2$, and that $y_{1s}y_{2s}$ is an unbiased estimator for the dot product $\vec{x}_1^T\vec{x}_2$, we might be concerned about the variance of our estimator, or interested in probability bounds on the the relative error of our estimated distance.

Before we look at the variances and probability bounds of these estimates, we should plot some graphs to get some intuition on how well the random projection estimate does.

The Matlab file `script_to_try.m` has the code for the following graphics.

We first load the `Colon` dataset [Alon et al., 1999], center the data, and normalize the vectors to have a length of 1. Each vector in the colon dataset is of $\mathbb{R}^{2000}$.

We then compute the estimate of the Euclidean distance as well as the dot product for the first two vectors in the `Colon` dataset when we project the data down to $\mathbb{R}^k$ for $k \in \{10, 20, \ldots, 1000\}$. We find the relative RMSE of these estimates.

We next compute the estimates of all pairwise Euclidean distances and the dot product for the Colon dataset (1891 pairwise distances), and compute their relative RMSE of these estimates as well.

The motivation for doing all pairwise distances is that our first two vectors of the Colon dataset may have been "good" vector pairs or "bad" vector pairs, and we want to avoid cherry picking. Computing the estimates of all pairwise distances gives some form of overall "average" on how well the random projection algorithm does.

Finally, we plot our RMSEs in Figure 4

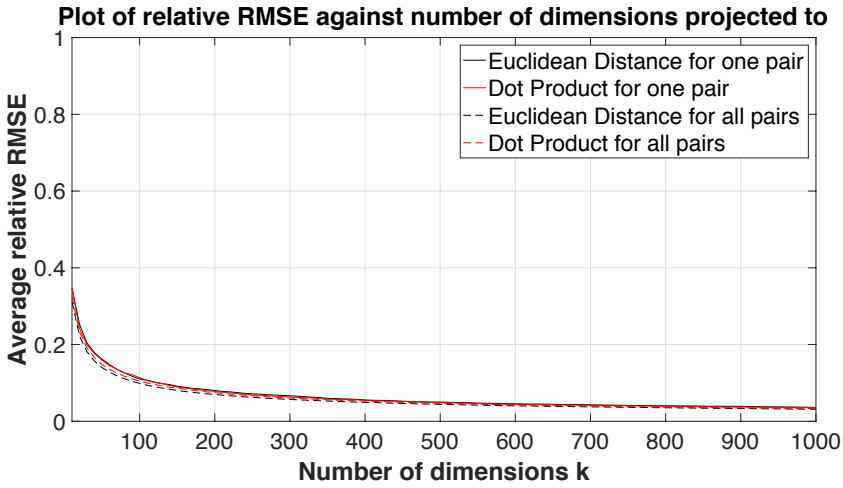

Figure 4: Plot of relative RMSE for all pairwise Euclidean distances and dot products for the Colon Dataset against dimension projected to for 1000 iterations for $k \in \{10, 20, \ldots, 1000\}$

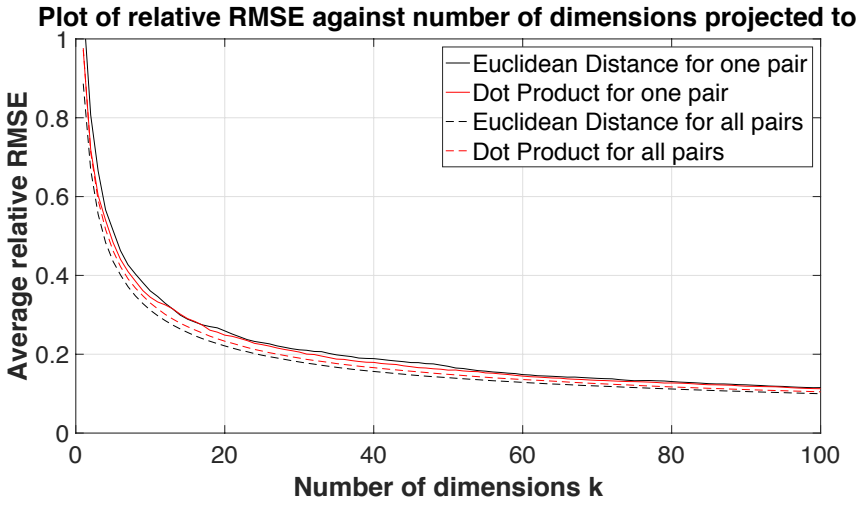

Figure 5: Plot of relative RMSE for all pairwise Euclidean distances and dot products for the Colon Dataset against dimension projected to for 1000 iterations for $k \in \{1, 2, \ldots, 100\}$.

We can see that the relative RMSE quickly reduces to about 0.1 as k increases between $k \in (0, 100)$. Afterwards, the rate of decrease in the relative RMSE slows down a lot. We can't really tell much, so let's zoom in.

We hence repeat our above experiment but for values of $k = \{1, 2, \ldots, 100\}$. The results are shown in Figure 5.

It seems that the relative RMSE of the dot product is slightly higher than the relative RMSE of the Euclidean distance from Figure 5. We will plot the deviation of the relative RMSE instead.

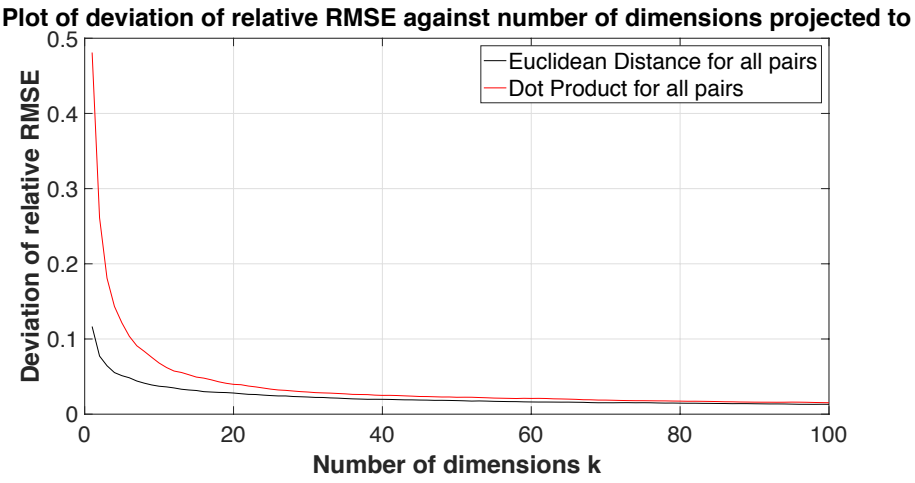

Figure 6: Plot of deviation of relative RMSE for all pairwise Euclidean distances and dot products for the Colon Dataset against dimension projected to for 1000 iterations for $k \in \{1, 2, \ldots, 100\}$

Figure 6 shows that the estimates of the Euclidean distance has a smaller deviation of the RMSE than the estimates of the dot product. This is in fact true in theory, as we will see in the next subsection.

2.1.2 Finding variance and bounds of estimates of the Euclidean distance and dot product between vector pairs

References

[Alon et al., 1999] Alon, U., Barkai, N., Notterman, D., Gish, K., Ybarra, S., Mack, D., and Levine, A. (1999). Broad patterns of gene expression revealed by clustering analysis of tumor and normal colon tissues probed by oligonucleotide arrays. Proceedings of the National Academy of Sciences, 96(12):6745–6750.

[Indyk and Motwani, 1998] Indyk, P. and Motwani, R. (1998). Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, STOC '98, pages 604–613, New York, NY, USA. ACM.

[Ross, 2006] Ross, S. M. (2006). Simulation, Fourth Edition. Academic Press, Inc., Orlando, FL, USA.