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 $(15.6, 2.2, 23.5) \in \mathbb{R}^4$.

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?

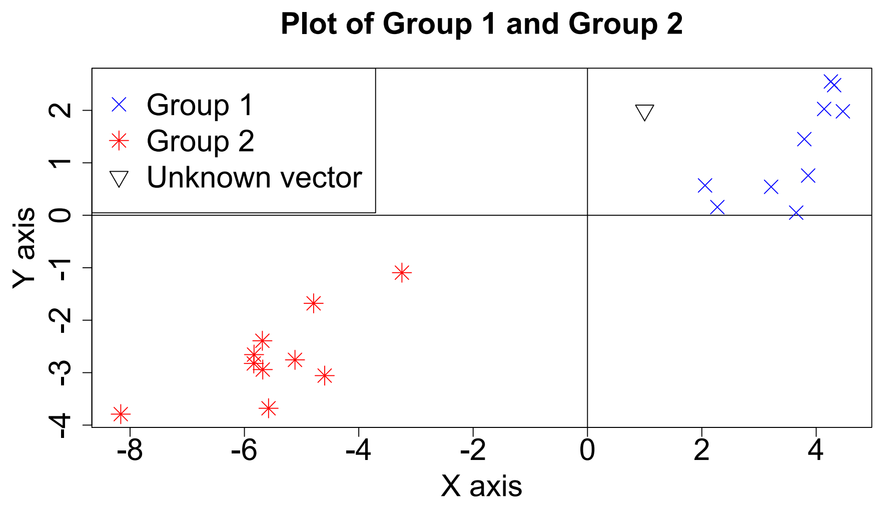

Figure 2: Toy example of two groups and one unknown vector.

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.

References

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