Pretend I'm a six year old - can you explain the papers and code in this directory to me?

Contents

1 Background information 2

1.1 The Iris Dataset 2

1.1.1 An example of clustering 2

1.1.2 An example of classification 4

1.2 Computing Distances 4

2 Generic random projections 7

2.1 Code 8

3 Statistical Analysis on Random Projections 9

3.1 Improving Random Projections using Marginal Information 9

3.1.1 Code 10

3.2 Random Projections with Control Variates 11

3.2.1 Code 11

4 Appendix 13

4.1 What is PCA / PC / why do we use it? 13

4.2 Maximum Likelihood Estimation 15

4.3 Control Variates 18

1 Background information

People usually want to do analysis on data, such as clustering (can I find interesting groups in the data?), or classification (given n groups in the data, and you give me an object, can I say what group this object belongs to?).

If every observation in the data can be represented as a vector, then one way to do classification or clustering is to consider distances between each vector.

1.1 The Iris Dataset

Let's consider the (Fisher's or Anderson's) iris dataset as an example of data.

This dataset consists of four measurements of 150 flowers - their sepal length, sepal width, petal length, and petal width. Each flower can be thus seen as a vector in four dimensions, with each dimension corresponding to one type of measurement. So for example, you can think of vectors $\mathbf{x}_1, \ldots$ of the form

$$\begin{aligned} \mathbf{x}_1 &= (x_{11}, x_{12}, x_{13}, x_{14}) \\ \mathbf{x}_2 &= (x_{21}, x_{22}, x_{23}, x_{24}) \\ &\vdots \end{aligned}$$

where x_{s1} represents the sepal length, x_{s2} represents the sepal width, x_{s3} represents the petal length, and x_{s4} represents the petal width.

How can we do some classification and/or clustering on this dataset?

We can't visualize vectors in 4 dimensions, so let's reduce the dimension to two dimensions. We do this by *principal component analysis*, and project this down to the subspace represented by the first two principal components of this dataset[1].

1.1.1 An example of clustering

Figure 1 overleaf shows the iris plot in two dimensions. How many clusters can be seen straightaway? Well, two clusters. One on the left, and one on the right. Now, if we *didn't know anything else about the data*, we could go: *Hmm, this is interesting. Let me give this*

[1]Check out the question: What exactly is principal component analysis on Page 13

data about the 150 flowers to a plant biologist, and see if there is anything special about these two clusters.

So you go and knock on the door of the person next to you, and he says: "*Ah, one cluster is entirely the* `setosa` *species! The other cluster consists of the* `versicolor` *and* `virginica` *species.*"

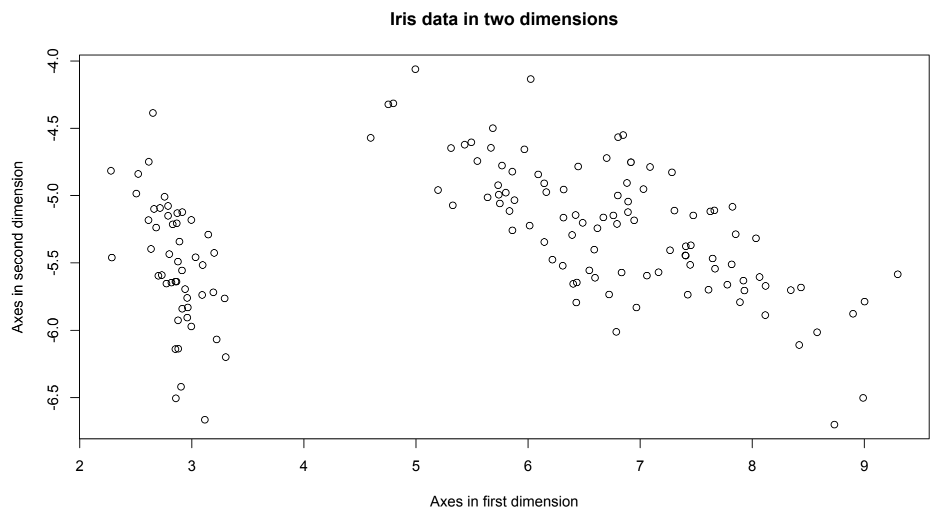

Figure 1: Iris in two dimensions.

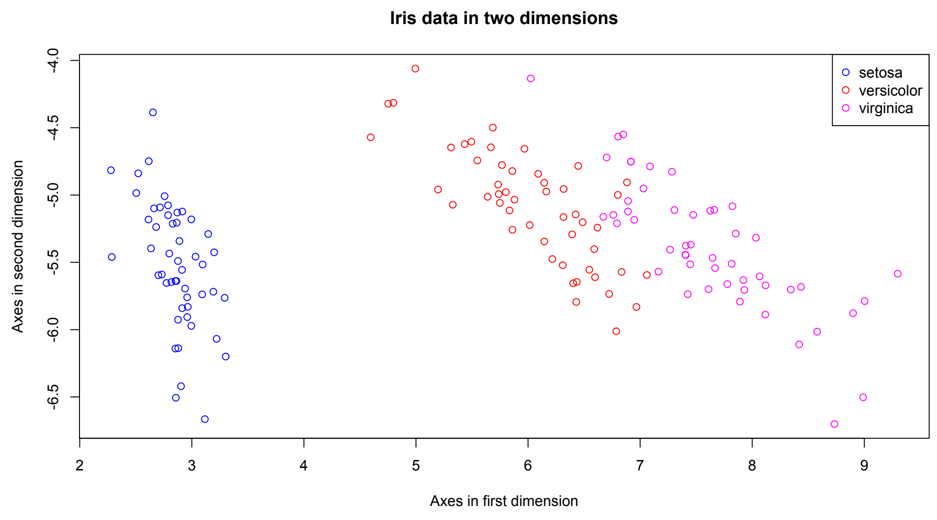

Figure 2: Iris with labels in two dimensions.

Indeed, as shown by Figure 2 above.

Clustering data can lead to the discovery of new groups in the data. In two dimensions, it is easy to see this by eye. In higher dimensions, algorithms like k-means, single-link, hierarchical clustering etc would be used, by computing the distances between every vector, and coming up with clusters.

1.1.2 An example of classification

What about classification?

Suppose I have these labels of the three irises, and you give me a new iris. One way to classify the new iris you gave me is to measure the sepal length, sepal width, petal length, and petal width, project it down to two dimensions, and figure out where it lies exactly.

If the new iris is "nearer" the `setosa` cluster, then I'd say the iris is probably a `setosa`. There are algorithms which can classify data based on distances, such as SVMs, nearest neighbor methods, LDA, etc.

The common thread between these clustering and classification is the computation of distances.

1.2 Computing Distances

But computing distances is easy, right?

Well, no. There's two factors involved in computing distances. The number of vectors n, and the dimensions of vectors p.

Let's tackle the number of vectors n. For n vectors, we'd have to compute *n choose 2* pairwise distances, which means we need to do $\frac{n(n-1)}{2}$ computations. So as n grows, the number of pairwise distances we need to compute grows by n^2.

In practice, if our distances are a metric, we can do some optimization using the triangle inequality to not do all $\frac{n(n-1)}{2}$ computations. But if we want to use other algorithms that rely on other types of distances (not just the simple Euclidean distance), then we might have to do more than $\frac{n(n-1)}{2}$ computations.

This is something we cannot escape - the computations have to be done.

So maybe we can do something about the dimension of the vector, p? For example, if $p = 2$, then finding the Euclidean distance between any two vectors $\mathbf{x}, \mathbf{y}$ would be given by

$$\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

If $p = 10000$, then this would be given by

$$\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_{10000} - y_{10000})^2}$$

The number of computations needed increases linearly as p increases.

Ah, I hear you say. Well, that's simple. We can use PCA to reduce the number of dimensions. If we project down to two dimensions, then $p = 2$, like the iris dataset. Why do we need random projections?

Here's one reason why PCA may not work. Let's go back to the iris dataset. We'll look at the scree plot of the principal components. Figure 3 shows the scree plot of the principal components, and it can be seen that the first principal component explains about 92.5% of variation in the data. So we could in fact project the iris data to *one* dimension instead of two dimensions, as in Figure 4.

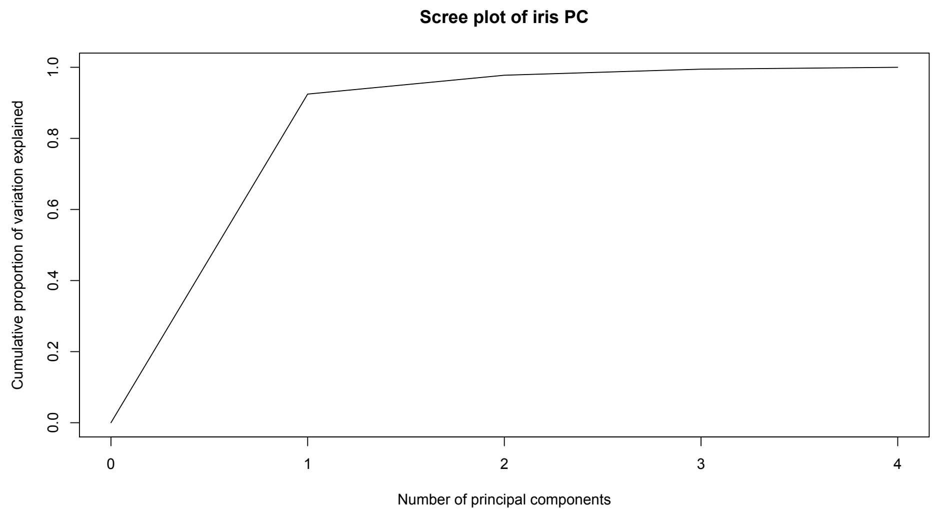

Figure 3: Iris scree plot.

However, not all datasets have the first few principal components explaining most of the variation in the data. In this case, we cannot just reduce the number of dimensions p to a low number. We would have to pick p to be the number of dimensions where the first p principal components explain a significant proportion of the variation in the data, and p could be large.

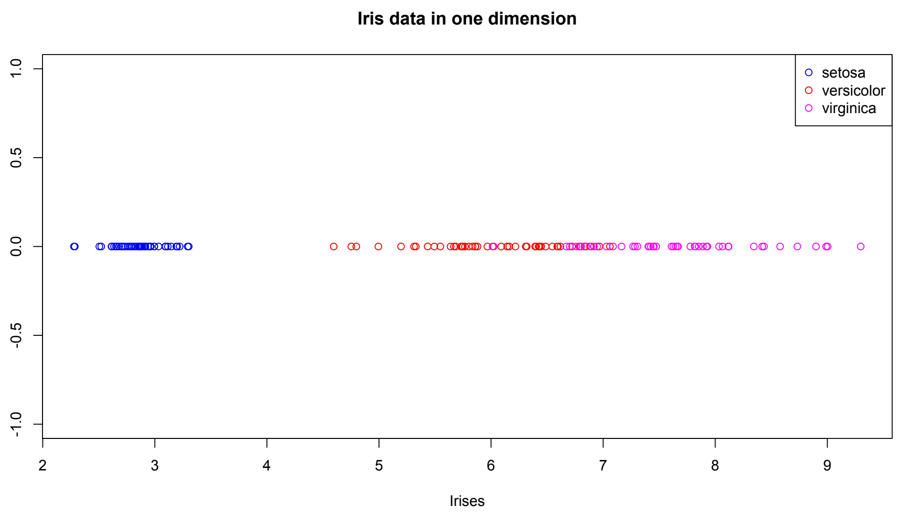

Figure 4: Iris in one dimension.

If p is large, then the data might not be able to fit into memory. This might make computing pairwise distances take a bit more time as well.

Secondly, PCA involves computing the singular vectors of the dataset. This is (somewhat) computationally intensive if we don't want to pick the first few singular vectors, but the first p singular vectors.

Random projections attempt to solve this.

2 Generic random projections

Suppose we have some (Euclidean) distance d_{12} between vectors $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^p$. We come up with some linear map R (our random projection!), and project the vectors down to a lower dimensional space k. So we have some $\mathbf{v}_1 = R\mathbf{x}_1, \mathbf{v}_2 = R\mathbf{x}_2$.

From construction, you can see that R is $k \times p$, with k rows and p columns.

Surprisingly, the Euclidean distance $\tilde{d}_{12}$ between $\mathbf{v}_1$ and $\mathbf{v}_2$ is within some $(1 \pm \epsilon)d_{12}$.

This method works for the inner product, l_p distances for $p = 2, 4, \ldots$ as well.

In fact, probability bounds can be placed on the estimate $\tilde{d}_{12}$ to bound the error ϵ based on the value of k (number of rows of R), *and not based on the value of* p.

This means: No matter how high my p is, *I do not care, since the error of my estimate is only affected by* k.

Let's look at the probability bounds of the squared Euclidean distance (ED) under random projections.

$$\mathbb{P}\left[(1-\epsilon)d_{12} \leq \tilde{d}_{12} \leq (1+\epsilon)d_{12}\rangle\right] < 1 - 2\exp\left\{-(\epsilon^2 - \epsilon^3)k/4\right\}$$

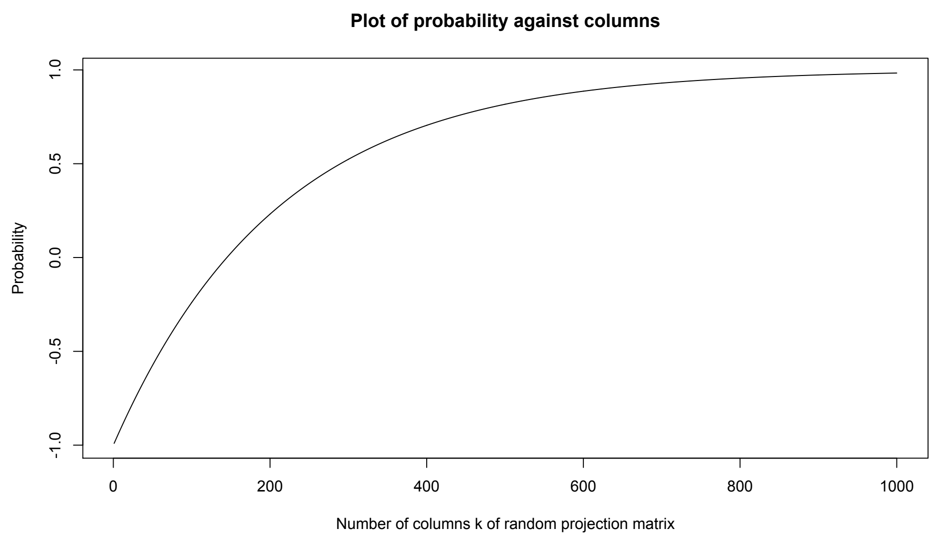

Figure 5: Bounds for Euclidean distance.

Suppose I want my estimate $\tilde{d}_{12}$ to be within $(1 \pm \epsilon)$ of my true distance with $\epsilon = 0.15$. Figure 5 tells me that I would need to project to 800 or 1000 dimensions such that I can be guaranteed this with high probability.

For $p < 1000$, random projections would be a bad idea. But for $p \gg 1000$, eg $p = 2^{32}$, or $p = 2^{64}$ or any p, random projections would be a good idea. *The error remains the same regardless of p!*

My research focuses on reducing such error in estimates.

2.1 Code

The code provided in the folder *Generic_RP_code* allows you to look at the average RMSE (root mean square error) of the random projection estimates of norms, Euclidean distances, and inner products. The code can compute the average error of all vector norm estimates, all pairwise Euclidean distance estimates, or all pairwise inner product estimates for a dataset, and plots the error.

You can use the code for a dataset of your choice to see how well random projections perform on it.

3 Statistical Analysis on Random Projections

Research has been done on improving the *speed* in random projections, as well as improving the *accuracy* of random projections. Part of my research looks at the statistical analysis of random projections, inspired by Ping Li.

The first subsection here looks at a paper which he wrote about a decade ago as first author.

3.1 Improving Random Projections using Marginal Information

I like this paper a lot, because it uses work *already known* in statistics since 1965.

Given $V = XR$, where entries R is i.i.d. $N(0,1)$, and vectors $\mathbf{x}$ in X normalized to have length 1, consider any pair $\mathbf{v}_i, \mathbf{v}_j$, where

$$\begin{aligned}\mathbf{v}_i &= (v_{i1}, v_{i2}, \dots, v_{ik})\\ \mathbf{v}_j &= (v_{j1}, v_{j2}, \dots, v_{jk})\end{aligned}$$

Each tuple $\{(v_{is}, v_{js})\}_{s=1}^{k}$ can be seen as a draw from a bivariate normal with zero mean and $\Sigma = \begin{pmatrix} 1 & a \\ a & 1 \end{pmatrix}$, where $a = \langle \mathbf{x}_1, \mathbf{x}_2 \rangle$.

Using maximum likelihood estimation from statistics, we can find the likelihood of our observations, and express this in terms of a.

The log likelihood is given by:

$$l(a) = -\frac{k}{2}\log(1-a^2) - \frac{k}{2}\frac{1}{1-a^2}\sum_{k=1}^{K}(v_{ik}^2 - 2v_{ik}v_{jk}a + v_{jk}^2)$$

Getting the MLE of a is equivalent to equating $l'(a) = 0$, and finding $\hat{a}$ which gives $l'(\hat{a}) = 0$. This happens to be the solution of this cubic

$$a^3 - a^2(\mathbf{v}_i^T\mathbf{v}_j) + a(\|\mathbf{v}_j\|_2^2 + \|\mathbf{v}_i\|_2^2 - 1) - \mathbf{v}_i^T\mathbf{v}_j = 0$$

It would be useful to look at the variance of Li's inner product estimate with the original variance of the inner product estimate. The variance can be thought of a proxy to the probability bounds - lower variance means better estimates.

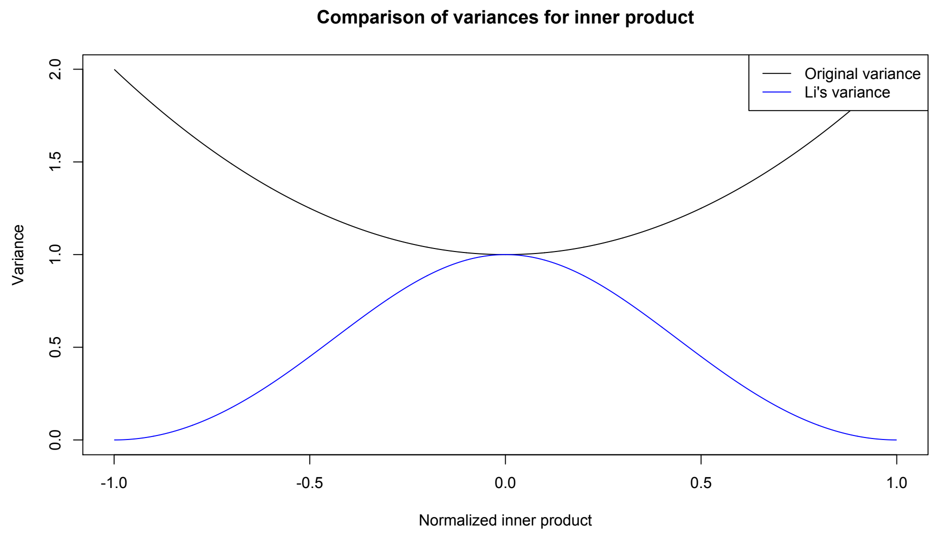

Figure 6: Li's estimate of inner product variance.

Figure 6 shows how Li's estimate is like for normalized data.

Recall that an inner product of 0 for two vectors means they are orthogonal (uncorrelated), an inner product of 1 means they are identical (same direction, strongly correlated), and an inner product of -1 means they are in opposite directions (strongly negatively correlated).

For vectors which are nearly uncorrelated, Li's method does not result in an improved variance. But for vectors which are correlated, Li's method results in an improvement in the estimate.

3.1.1 Code

The code provided in the folder *Improving_RP_with_Marginal_Info* allows you to compare the average RMSE (root mean square error) of the random projection estimates of inner products with the ordinary estimate.

You can use the code for a dataset of your choice to see how well random projections with Li's method performs on it.

Both vectorized Newton Raphson and Cardano's formula is provided - without vectorization, it would be difficult to compute the pairwise inner products efficiently!

I use the code in this folder for several baselines.

3.2 Random Projections with Control Variates

Here, we use the same setup as Li, looking at the bivariate normal

$$\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} \sim N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & a \\ a & 1 \end{pmatrix}\right)$$

Suppose we want to estimate some quadratic form $\mathbf{v}^T A\mathbf{v}$. The Euclidean distance would be $A = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$, and the inner product $A = \begin{pmatrix} 0 & 1/2 \\ 1/2 & 0 \end{pmatrix}$.

Information on control variates can be found on Page 4.3.

We set our control variate to be $\mathbf{v}^T\mathbf{v}$, which we know the value to be 2 since we normalized the data. If the data were not normalized, then the expected value of $\mathbf{v}^T\mathbf{v}$ would be the sum of $\|\mathbf{x}_1\| + \|\mathbf{x}_2\|$.

The theoretical value of the control variate correction $\hat{c}$ is in terms of the inner product a, so the empirical $\hat{c}$ should be computed from the data.

However, substituting a to be the inner product estimate given by random projection (which we need to compute anyway), or Li's estimate into the theoretical value of $\hat{c}$ gives better results than computing empirical $\hat{c}$.

It is also instructive to look at the plot of the new variances after using the control variate shown in Figure 7.

It can be seen that while the control variate for the inner product performs just as well as Li's estimate, the control variate for the Euclidean distance performs well.

When vectors are far apart from each other, the control variate drastically reduces the variance.

3.2.1 Code

The code provided in the folder *Random_Projections_with_Control_Variates* allows you to compare the average RMSE (root mean square error) of the random projection estimates of the Euclidean distances and inner products with control variates against the respective ordinary estimates (and Li's estimate for the inner product).

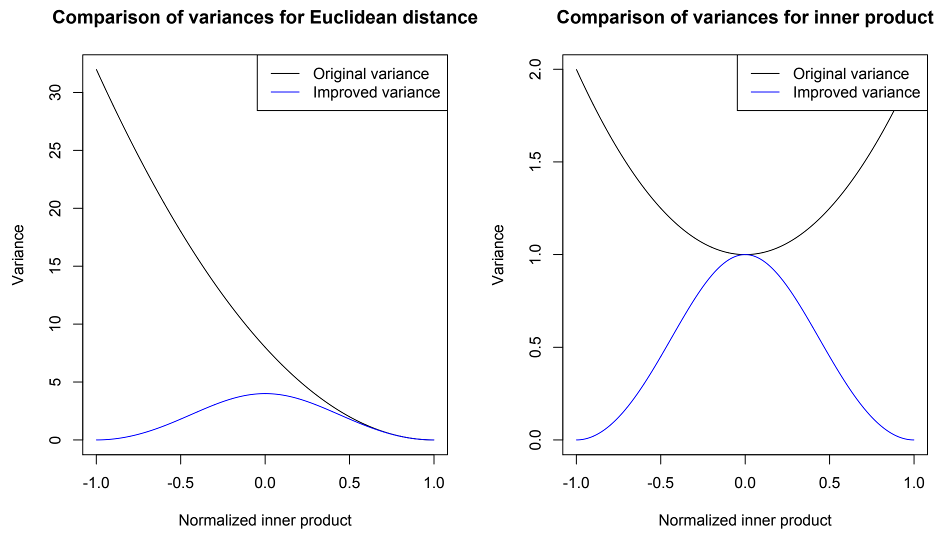

Figure 7: Bivariate normal control variate estimates.

You can use the code for a dataset of your choice to see how well random projections with control variates perform.

Both vectorized versions of computing the empirical covariance is given, otherwise it would be difficult to implement control variates efficiently.

4 Appendix

This section covers everything else not related to random projections.

4.1 What is PCA / PC / why do we use it?

There's a huge wealth of literature out there on principal component analysis (PCA), but in the context of this directory, here's a 2D example.

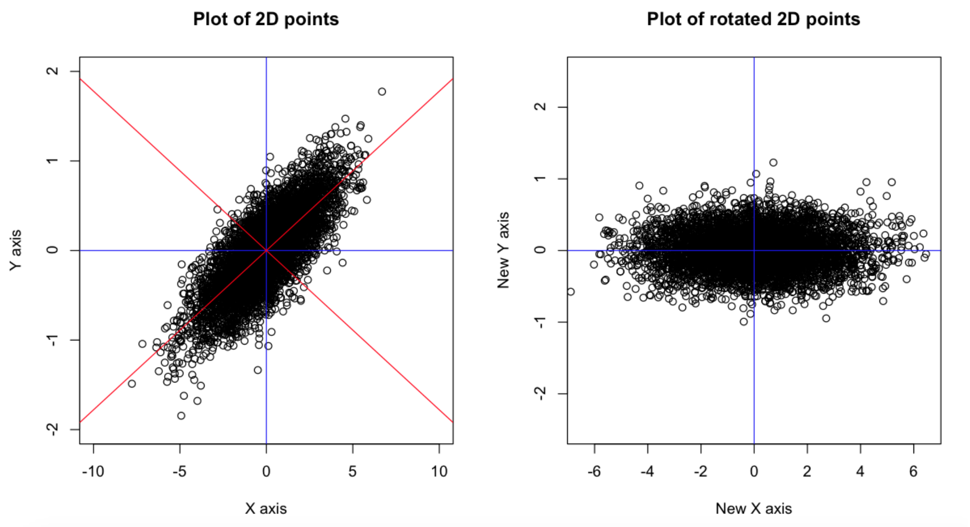

Figure 8: 2D plot of some random points.

Consider the first set of points on the left hand side of Figure 8, and we have the traditional x axis (horizontal line), and y axis (vertical line). Suppose we want to rotate the data to some new axes such that each subsequent axis try to account for most of the remaining variation. Graphically, the red lines show the new set of axes, and if we "rotate" the data, we see something like the plot on the right hand side.

Now, the next question might be why do we want to do this rotation. It might be hard to visualize in the 2D example, but not all dimensions are informative.

To phrase it differently, suppose we had a storage problem. Instead of representing 2D points as $(x_1, y_1), (x_2, y_2), \ldots$, you are told that you can only represent points in 1 dimension, *along any type of axes you choose.*

Which set of axes would you choose, if you want to retain as much information about the data as possible? You'd probably say "I'll choose the axis given by the diagonal red line stretching from the bottom left to the top right".

Of course, this is not the only reason why we do principal component analysis, but it's a nice starting point. You can refer back to the iris data to see how much information (you think) is lost when projecting to a two dimensional space.

There's also something called a scree plot, which measures the variation accounted for by all the principal components. An example of how the scree plot looks like, and what it means is given on Page 5.

4.2 Maximum Likelihood Estimation

Usually, we do not know the (parameters of the) probability distribution which data comes from.

We define a **statistical model** to be a family $\{\mathbb{P}_\theta \mid \theta \in \Theta\}$ of different probability distributions parameterized by some θ belonging to a set of possible values Θ.

We usually choose P_θ which is the "most probable".

Example 4.1. *Coin tossing*
We toss a biased coin three times, and record the sequence of tosses. We denote H for `Heads`, and T for `Tails`.

We model the outcome we see as a Bernoulli distribution with some parameter θ of coming up heads, where each toss is independent.

Then we would say the statistical model in this scenario is a Bernoulli distribution, indexed by $\{P_\theta \mid \theta \in [0,1]\}$.

Suppose we see the sequence HHT.

The probability of seeing this sequence would be $\mathbb{P}_\theta(\text{"HHT"}) = \theta \times \theta \times (1-\theta) = \theta^2(1-\theta)$.

Which choice of θ would be "reasonable"?

For example, if we chose $\theta = \frac{1}{100}$, then $\mathbb{P}_\theta(\text{"HHT"}) = \frac{99}{10^6}$.

On the other hand, if we chose $\theta = \frac{1}{2}$, then $\mathbb{P}_\theta(\text{"HHT"}) = \frac{1}{8}$.

$\theta = \frac{1}{2}$ seems likelier than $\theta = \frac{1}{100}$, but is it the most likely?

We can always denote:

$$L(\theta) = \theta^2(1-\theta)$$

to be the likelihood function, and Figure 9 on Page 16 shows the plot of $L(\theta)$.

The value of $L(\theta)$ is maximized when $\theta = \frac{2}{3}$, and in the absence of all other information, we would say that the most plausible parameter is $\theta = \frac{2}{3}$ given the observed sequence.

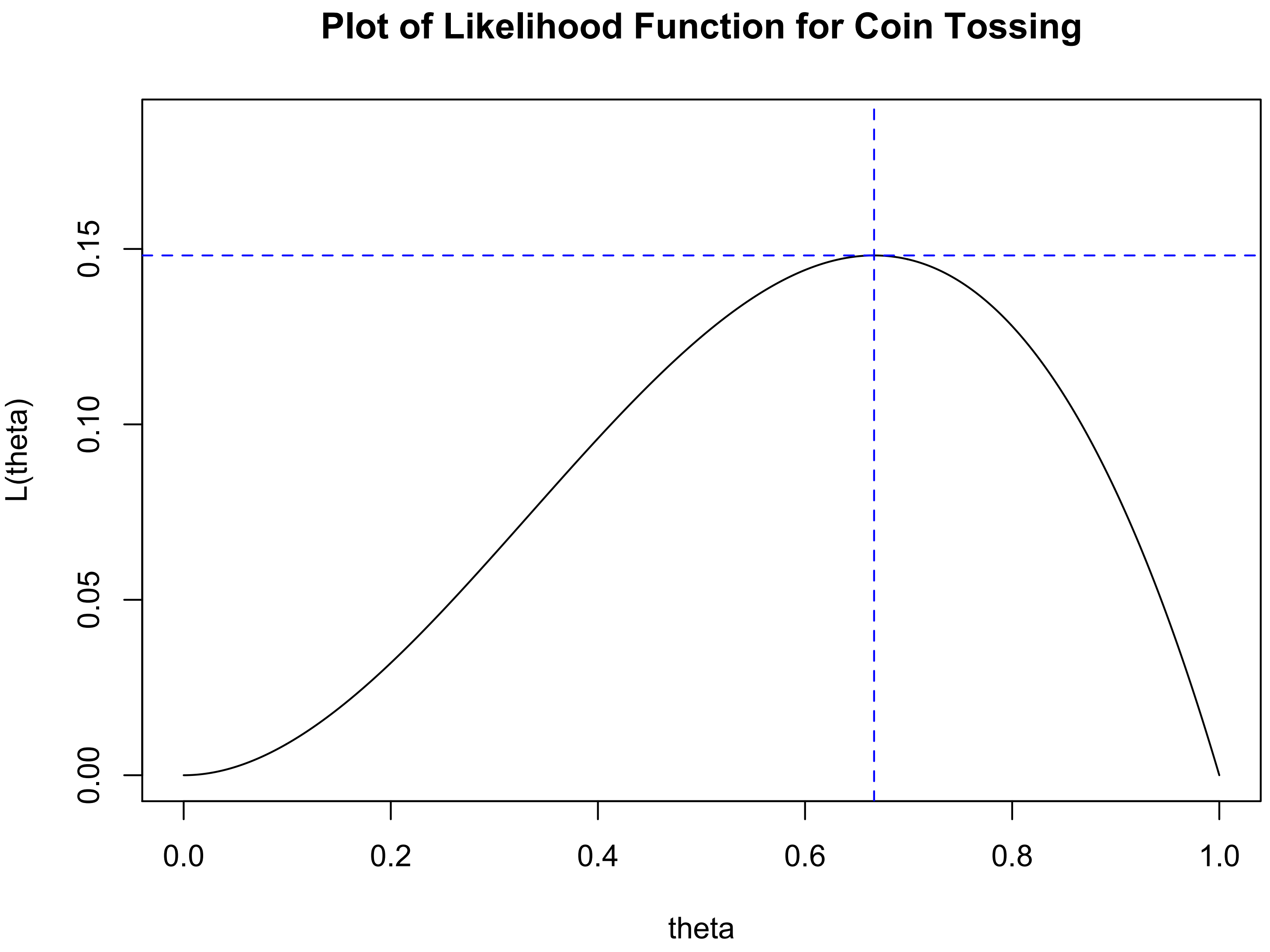

Figure 9: Likelihood Function for Coin Tossing

Thus, a recipe to find the maximum likelihood estimates would be the following.

1. Given data, find a statistical model that could represent it. For coin tossing example, pick “Bernoulli”.
2. Write down the parameters of this statistical model. For coin tossing example, θ (probability of tossing H).
3. Write down the likelihood function *based on the parameters and what you observe.* For coin tossing example, we observed HHT, so likelihood function is $\theta^2(1-\theta)$.
4. Find the maximum of this likelihood function *over plausible range of parameters.* For coin tossing example, $\theta \in [0,1]$, so only look at values in this range.
5. This value of θ which gives the maximum is your maximum likelihood estimate.

It is not always the case that an exact closed form answer can be given by just differentiating and solving for the optimal value of θ(s). Thus, numerical optimization needs to be done. Li’s method on Page 9 is an example of this.

4.3 Control Variates

Control variates uses the same random numbers to estimate a random variable X, as well as to estimate a different variable Y of which the true mean μ_Y is known. As the errors in estimation will be correlated, knowing how far off our empirical observation of Y is from the true mean of Y will allow us to correct our observation for the random variable X. Mathematically, we can write

$$\begin{aligned}\mathbb{E}[X + c(Y - \mu_Y)] &= \mathbb{E}[X] + c(\mathbb{E}[Y] - \mu_Y) \\ &= \mathbb{E}[X]\end{aligned}$$

Y is called the *control variate*, and c is called the *control variate correction.* The goal is to find c such that

$$\text{Var}[X + c(Y - \mu_Y)] = \text{Var}[X] + c^2\text{Var}[Y] + 2c\text{Cov}[X, Y]$$

is minimized. From calculus, the optimal value of c is

$$c = -\frac{\text{Cov}[X, Y]}{\text{Cov}[Y]}$$

and thus we have

$$\text{Var}[X + \hat{c}(Y - \mu_Y)] = \text{Var}[X] - \frac{\text{Cov}[X, Y]^2}{\text{Var}[Y]}$$

The amount of variance reduction depends on the correlation between X and Y, but note that $\frac{\text{Cov}[X,Y]^2}{\text{Var}[Y]}$ is always non-negative.