Principal Component Analysis.

Linear algebra is one of the great superpowers! A lot of what we do in data science, indeed, a lot of mathematical modelling is either a direct example of linear algebra, or, something more complicated we manage to solve because we can break it up into bits that look like linear algebra. Linear algebra isn't difficult, at the start it just looks like a clever way to organise some complicated calculations, you might say that mathematics is always just a clever way to organise some calculations. There is, however, a deep idea that emerges in linear algebra which we will touch on here and exploit: matrices can be thought of in terms of their components, the individual a_{ij} or in terms of their eigenvectors and eigenvalues, what is sometimes called their *spectral content*; we will use this to find diagonal structures in the matrices which we can use to find structure in data.

In this section we are going to look at principal component analysis (PCA). You will actually encounter PCA twice, in this unit and when you are learning about visualization. This reflects the importance of PCA, from a visualization point-of-view because it is often the first step in any examination of data and from a AI point-of-view because it is an early demonstration of the idea that data usually has a structure and this is why AI works as well as it does!

PCA is an unsupervised technique but a good supervised learning algorithm will always have a hidden unsupervised aspect, it discovers the structure of data on the way to learning how you want it classified. As well as being useful example of unsupervised learning, it reminds us a bit about linear algebra and it teaches us a bit about how to think about data. However, although we won't go into this explicitly, keep in mind that many interesting aspects of neural networks are understandable as non-linear generalizations of PCA! Anyway, I hope you won't mind this small piece of repetition between units, it is worth doing twice and I am sure it will be done with different emphasis in both cases, I am also sure the other lecturer will also show more skill in teaching the material, so you are lucky not to have to rely on just my version!

The first thing is *unsupervised learning*; by unsupervised learning we mean, broadly, we don't start with a set of data points and some labels and try to learn how to map from data to labels, that's supervised learning. In unsupervised learning we have some data points and we try to decide what structure the points have, what is interesting about them, before actually

thinking about labelling. Old fashioned textbooks make a big deal about the distinction between supervised and unsupervised learning; often, in real life, we find that distinction is not so clear; a neural network, for example, is typically trained on labelled data, so they are exampled of supervised systems, but, as noted above, often earlier layers do something more akin to unsupervised learning, breaking input down by its intrinsic structure, all the better to be classified by later layer.

I am overly inclined to talk about things in terms of what the brain does, but this is a good place to do that. The brain clearly performs some mixture of unsupervised and supervised learning. A baby, when it first is learning to see, when its visual system is wiring itself up, is being presented with visual data, this data has structure, there are edges, the edges surround objects, objects appear in front of each other in a sort of consistent way, they have shading which indicates something about orientation, convexity or concavity; the same object can be large or small according to how far away or how close it is. This the baby learns before, or while, it starts to learn to recognize different things. There is unsupervised learning of the structure of the visual world that allows the brain to understand the existence and the nature of objects, as well as supervised learning of what these objects are called. There is an interesting part of the story somewhat related to PCA, an elaboration of PCA called independent component analysis, or ICA, allows us to interpret the dynamics of the visual system in terms of the edge structure of images.

Some linear algebra!

I don't want to spend ages reminding you how linear algebra works; you possibly know this well and if you don't, you should look this up. However, it might be useful to have a quick recap. There are lots of ways to think about what matrices do, in fact, it is a clue to what a powerful concept they are that they can be introduced in some many different ways!

Here is one version! Consider the simultaneous equations:

$$\begin{aligned} x + y &= 3 \\ x - y &= 1 \end{aligned} \tag{1}$$

We can solve this easily by adding the two equation, this give $2x = 4$ or $x = 2$, substituting that back in gives $y = 1$ and it is easy to check that this solve both equations. The geometrical explanation is that both equations

are equations for lines and the point $(2, 1)$ is the unique point that lies on both lines solving both equations. Of course, it would be trick to write down how we did this for a computer algorithm, to write it down so it works even when the y has less convenient coefficients and when there are not two but tens, or hundreds, of variables.

In fact, this is how we make it plain. We can rewrite the equation in matrix form:

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \end{pmatrix} \tag{2}$$

which is in the matrix form

$$A\mathbf{x} = \mathbf{y} \tag{3}$$

Provided we can find an inverse matrix A^{-1} so that $A^{-1}A = \mathbf{1}$ then we can solve the equation:

$$A^{-1}A\mathbf{x} = A^{-1}\mathbf{y} \tag{4}$$

or

$$\mathbf{x} = A^{-1}\mathbf{y} \tag{5}$$

In fact if

$$A = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{6}$$

as above, then

$$A^{-1} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{7}$$

or

$$\begin{pmatrix} x \\ y \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \tag{8}$$

So what did that palaver with the matrix get us? In a way it deskilled the solving of the simultaneous equation; the first time around, when we solved it without using matrices we began by adding the two equations so the y terms cancelled and we got a value for x. This didn't require much skill since it was a very easy example, but you can see that some thinking is required. The matrix method, in contrast, only relied on the step I sort of skipped, inverting the matrix; if you write the equation in terms of matrices then you can solve them by inverting the matrix. In fact, any method of solving the matrix is equivalent to inverting the matrix, so solving the equation is exactly as hard as inverting the matrix. We have, over the last 100 years put a lot

of effort into finding ways to invert matrices, it is a long-winded calculation, but one we are very good at, and by "we" I mean linear algebra packages.

Not all matrices can be inverted: this matrix

$$A = \begin{pmatrix} 2 & 3 \\ -4 & -6 \end{pmatrix} \tag{9}$$

for example, has no inverse. That's ok though because not all simultaneous equations can be solved, for example

$$\begin{aligned} 2x + 3y &= 3 \\ -4x - 6y &= 1 \end{aligned} \tag{10}$$

has no solution because the two lines, the $2x+3y = 3$ line and the $-4x-6y = 1$ line are parallel. In fact, I can tell straight away that the matrix can't be inverted by working out its determinant. The determinant is a quantity associated with matrices, in the 2×2 case it has a simple formula:

$$\det \begin{pmatrix} a & c \\ d & b \end{pmatrix} = ab - cd \tag{11}$$

So

$$\det \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = -2 \tag{12}$$

and

$$\det \begin{pmatrix} 2 & 3 \\ -4 & -6 \end{pmatrix} = 0 \tag{13}$$

The rule is that a matrix has an inverse if its determinant isn't zero. The formula for working out the determinant of bigger matrices is tedious to explain, but easily programmed, we can easily work out determinants, though for anything bigger than 2×2 or certainly 3×3 it's best to get a computer to do it for you!

Eigenvectors and eigenvalues

Ok so this bit is the special secret bit. Matrices can be thought of as being made up of their eigenvectors. The word *eigen* means equal in German and an eigenvector for a matrix is a direction that isn't changed by the matrix. It satisfies:

$$A\mathbf{e} = \lambda\mathbf{e} \tag{14}$$

In other words, multiplying the eigenvector by the matrix can make it longer or shorter, according to whether λ is bigger or smaller than one, but it stays pointing in the same direction. It isn't clear yet why this is a big deal, but it will turn out to be a nice way of thinking how matrices work and we'll use it to do PCA!

How do you find the eigenvectors? Well this is a lovely piece of using mathematics in a kind of wrong way around trick. We want

$$A\mathbf{e} = \lambda\mathbf{e} \tag{15}$$

so we can write that as

$$(A - \lambda\mathbf{1})\,\mathbf{e} = \mathbf{0} \tag{16}$$

For simple example we aren't going to use the example from earlier, it isn't hard but it does have a $\sqrt{2}$ as an eigenvector so lets instead use

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \tag{17}$$

then

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} = \lambda \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} \tag{18}$$

then we rewrite this as

$$\begin{pmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{19}$$

Back though to the general version

$$(A - \lambda\mathbf{1})\,\mathbf{e} = \mathbf{0} \tag{20}$$

for convenience write $A_\lambda = A - \lambda\mathbf{1}$ so our equation becomes

$$A_\lambda\mathbf{e} = \mathbf{0} \tag{21}$$

Now if we invert the matrix A_λ everything goes wrong:

$$\mathbf{e} = A_\lambda^{-1}\mathbf{0} = \mathbf{0} \tag{22}$$

so for the eigenvector to be something interesting, that is not just zero, the matrix A_λ must have no inverse! Luckily we know how to check that, it means

$$\det A_\lambda = 0 \tag{23}$$

which we'll see in a second is just an equation for λ; this equation is called the *characteristic equation*.

Let's try this for a simple example:

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \tag{24}$$

so

$$A_\lambda = \begin{pmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{pmatrix} \tag{25}$$

then

$$\det A_\lambda = (2-\lambda)^2 - 1 = \lambda^2 - 4\lambda + 3 \tag{26}$$

so the characteristic equation is

$$\lambda^2 - 4\lambda + 3 = 0 \tag{27}$$

which has solutions $\lambda = 3$ and $\lambda = 1$.

It turns out once you know the eigenvalues you can easily work out the eigenvectors by substituting back into the original equation. You can look up how to do this, it isn't hard. In this case for $\lambda = 3$ the eigenvector is, from the equation

$$-e_1 + e_2 = 0 \tag{28}$$

so, any example would be given by picking $e_2 = 1$

$$\mathbf{e} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{29}$$

and for $\lambda = 1$

$$\mathbf{e} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{30}$$

There is a small subtlety here which we say in the 'an example would be given' but above. Basically an eigenvector is a direction not a vector, if $\mathbf{e}$ is an eigenvector then so is $\mu\mathbf{e}$ for any non-zero μ, this hinges on the linearity of matrices, say $A\mathbf{e} = \lambda\mathbf{e}$ then $A(\mu\mathbf{e}) = \mu A\mathbf{e} = \mu\lambda\mathbf{e} = \lambda(\mu\mathbf{e})$, so if $\mathbf{e}$ is an eigenvector, so is $\mu\mathbf{e}$. Often we are going to use unit-length eigenvectors and when we want to remind ourselves of that we will give them a hat $\hat{\mathbf{e}}$:

$$|\hat{\mathbf{e}}| = 1 \tag{31}$$

Diagonalization

What good are eigenvectors? Well, as I said before, you can think of matrices as being made up of their eigenvectors; I don't want to go into this in detail, there are a lot of details, but the example I want to look at is diagonalization, this is a related sort of idea and then one we need for doing PCA!

To talk about diagonalization, I am going to assume the matrix is symmetric. This isn't required, there are diagonalizable non-symmetric matrices, but the example we need for PCA is symmetric and the symmetric case lacks all the caveats that we need for the more general case, by focusing on diagonalizable we get to avoid some "as long as blah blah blah" type statements.

To remind you about symmetry, the transpose of a matrix involves flipping around the diagonal, so if $A = [a_{ij}]$ then $A^\mathsf{T} = [a_{ji}]$, for 2×2 is

$$A = \begin{pmatrix} a & c \\ d & b \end{pmatrix} \tag{32}$$

then

$$A^T = \begin{pmatrix} a & d \\ c & b \end{pmatrix} \tag{33}$$

A matrix is symmetric if $A = A^T$ so

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix} \tag{34}$$

is symmetric,

$$A = \begin{pmatrix} 1 & 2 \\ -2 & 1 \end{pmatrix} \tag{35}$$

is not.

Now, what do we mean by diagonalization, it basically means that we can rewrite a symmetric matrix in the form

$$A = PDP^{-1} \tag{36}$$

where D is a diagonal matrix made up of the eigenvalues:

$$D = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) \tag{37}$$

and the P's are matrices you make out of the eigenvalues.

This is both deep and useful; to deal with the useful aspect first, if

$$A = PDP^{-1} \tag{38}$$

then

$$A^2 = PDP^{-1}PDP^{-1} = PD^2P^{-1} \tag{39}$$

and, diagonal matrices are easy to deal with, if $D = \text{diag}(d_1, d_2, \ldots, d_n)$ then $D^2 = \text{diag}(d_1^2, d_2^2, \ldots, d_n^2)$. In this way diagonalizing a matrix can make lots of things you might want to do with the matrix easier.

The deep part is harder to see but it says in a way that for any matrix there is a set of axes where the matrix just scales along the axes. P^{-1} is a matrix, so it makes a linear transformation on your data, like a change of axes, so the equation is saying, first use P^{-1} to changes axes, then use D to scale the axes and then use P to go back to your original axes: every matrix has a system of axes where all it does is scale things!

In actual computational terms, we make P using the eigenvectors as columns, so

$$P = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_n] \tag{40}$$

As we pointed out there is some question of the length of the eigenvectors, it is convenient in this context to make them unit length. There are technical details here that I am sort of hiding but are actually interesting and fascinating, basically for a symmetric matrix with an inverse, the example we are dealing with here, the eigenvectors are orthogonal, lets first make them have unit length so:

$$|\mathbf{e}_i| = 1 \tag{41}$$

for all i, now

$$\mathbf{e}_i^\mathsf{T}\mathbf{e}_j = \delta_{ij} \tag{42}$$

where I amn't explaining what happens if two eigenvectors have the same eigenvalue, in that case you can make the eigenvectors orthogonal, they aren't automatically orthogonal as they are when the eigenvalues are different, but they can be orthogonalized. Again these are details you can read about. The key thing is that in this case P has the special property that its transpose is its inverse:

$$P^\mathsf{T}P = \mathbf{1} \tag{43}$$

because

$$P^\mathsf{T} = \begin{pmatrix} \mathbf{e}_1^\mathsf{T} \\ \mathbf{e}_2^\mathsf{T} \\ \dots \\ \mathbf{e}_n^\mathsf{T} \end{pmatrix} \tag{44}$$

and so the individual entries of $P^\mathsf{T}P$ are dot-products of eigenvectors.

As a point of terminology this means P is an *orthogonal* matrix. You can check why that might be the case if you want. Another point to learn more about if you want is that an orthogonal matrix represents a rotation, or a rotation and a reflection, so

$$\mathbf{y} = P\mathbf{x} \tag{45}$$

means that $\mathbf{y}$ is a rotation of $\mathbf{x}$, or a rotation and reflection.

Anyhow, for the example above then:

$$P = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{46}$$

and

$$\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^\mathsf{T} \tag{47}$$

This is telling us that we can express A as a rotation and reflection, followed by a scaling of the axes, followed by rotating and reflecting back.

A first look at PCA

In Figure 1 we have data points with a clear structure, they are spread out along the $x = y$ axis and have a bit of "blurring" along the $x = -y$ direction. In an experiment it might be that the $x = y$ represented the phenomena, and $x = -y$ noise, you might actually want to ignore the $x = -y$ direction and just project the points onto the $x = y$ direction; in any case, discovering this structure would be interesting. That's the business of PCA.

We should emphasise that this is a particularly clean example, the 'blur' is completely independent of the 'signal'. In many cases this won't be true; often PCA is a best guess or approximation to finding directions along which things happen in an independent way, but they these directions may not exist, or may not be perpendicular and PCA will only be the first step before

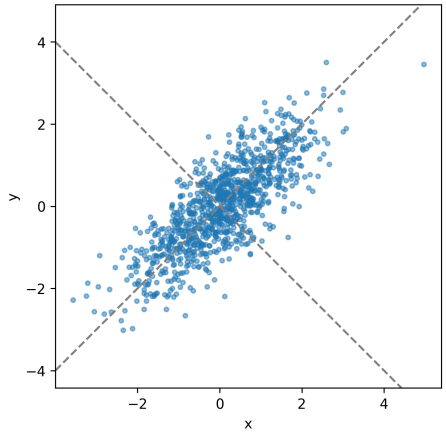

Figure 1: Anisotropic Gaussian with principal axes $(1, 1)$ and $(1, -1)$. You can come up with whatever story you like about these data, lets say that the ear size of lots of people has been measured, the x axis is the size of the left ear, the y axis the size of the right ear. For simplicity the average has been removed. Now, clearly the $x = y$ direction measures "how big is the person" and the $x = -y$ measures "how symmetric is the person".

using other, more non-linear methods, such as ICA or a neural network of some sort, an autoencoder of some sort, for example.

Back the PCA! If we are presented with data the first thing we might want to examine is the covariance matrix for the data:

$$\Sigma = \mathbb{E}\big[(x - \mu)(x - \mu)^{\mathsf{T}}\big] \,. \tag{48}$$

where μ is the mean and in our example we've already gotten rid of that. In practice we work out the sample covariance:

$$\hat{\Sigma}_{ij} = \frac{1}{n-1} \sum_{k=1}^{n} \big(x_i^{(k)} - \bar{x}_i\big)\big(x_j^{(k)} - \bar{x}_j\big). \tag{49}$$

The little hat is there to distinguish the theoretical concept with expectation values and what we actually work out from data. The covariance is a measure of, well covariance, "co-" often means with and covariance measure how much things vary together, Σ_{12} measure how much X_1 varies as X_2 varies for example. If two variables are independent then the corresponding entry in the covariance matrix will be zero, though, of course, the sample covariance

might give a small value. In passing, the converse is not true, zero covariance does not imply independence, you need the mutual information for that!

For the data in Fig. 1 this give

$$\hat{\Sigma} = \begin{pmatrix} 1.20532775 & 0.92028907 \\ 0.92028907 & 1.12193513 \end{pmatrix} \tag{50}$$

and so the 1.20 is the variance in the x direction, 1.12 in the y direction and the 0.92 is how much does x depend on y. The data we measure, x and y is covariant, what we are interested in discovering, which in this simple example is clear from the graph, is the direction $x = y$ where the real action is happening; the data is something like "every data point has some value that determines how big x and y both are, except there is a little bit of noise that makes them a bit different". In this simple example the noise is completely unrelated to the signal, so knowing how far you are in the $x = y$ direction does not tell you anything about what is happening in the $x = -y$ direction. For real data this is not often true, but it is often approximately true. In any case, we can see that we are looking for directions where the covariance matrix is diagonal! This is diagonalization, what we looked at before.

If we work out the characteristic equation of Σ above we get $\lambda_1 = 2.07$ with eigenvector $\mathbf{e}_1 = (1, 1)^T$ and $\lambda_2 = 0.26$ with eigenvector $\mathbf{e}_2 = (1, -1)^T$, so, basically

$$\begin{aligned} \hat{\Sigma} &= \begin{pmatrix} 1.20532775 & 0.92028907 \\ 0.92028907 & 1.12193513 \end{pmatrix} \\ &= \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 2.07 & 0 \\ 0 & 0.26 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^{\mathsf{T}} \end{aligned} \tag{51}$$

So what's happened here: basically if we have already removed the average, the covariance matrix is

$$\Sigma_{\mathbf{x}} = \mathbb{E}\left[\mathbf{x}\mathbf{x}^{\mathsf{T}}\right]. \tag{52}$$

What happens if we do a change of variables, imagine $\mathbf{x} = P\mathbf{y}$

$$\begin{aligned} \Sigma_{\mathbf{x}} &= \mathbb{E}\left[\mathbf{x}\mathbf{x}^{\mathsf{T}}\right] \\ &= \mathbb{E}\left[P\mathbf{y}\mathbf{y}^{\mathsf{T}}P^{\mathsf{T}}\right] = P\mathbb{E}\left[\mathbf{y}\mathbf{y}^{\mathsf{T}}\right]P^{\mathsf{T}} \\ &= P\Sigma_{\mathbf{y}}P^{\mathsf{T}} \end{aligned} \tag{53}$$

In otherwords, diagonalizing the covariance matrix is just a way to find new axes whose covariance matrix is diagonal. These new axes are the *principal components*.

A practical way to think about this is in terms of *loadings* and *scores*. Recall the unit eigenvectors of the covariance matrix are used to make the diagonalizing matrix P:

$$P = [\hat{\mathbf{e}}_1, \hat{\mathbf{e}}_2, \ldots, \hat{\mathbf{e}}_n], \tag{54}$$

The columns of P are often called the *loading vectors*: each loading tells you which original variables contribute positively or negatively to that principal component, and how strongly. For a centred data vector $\mathbf{x}$, its coordinates in the new PCA system are

$$\mathbf{y} = P^{\mathsf{T}}\mathbf{x}, \tag{55}$$

and these numbers $\mathbf{y}$ are called the *principal component scores*; they give the values the data points have in the new axes. Basically instead of the original data $\mathbf{x}^k$ we have new data $\mathbf{y}^k$, this is the same data but in a new, rotated, coordinate system. We still need to interpret these new axes. In the stupid story we had for the data in Fig. 1 we knew what x_1 and x_2 were[1], the sizes of the left and right ears. The new coordinates y_1 and y_2 we still need to interpret, they might be "how big the person is" and "how symmetric the person is". PCA doesn't give you the interpretation, it just finds the components. You might want to ignore some of the scores, so in this case we might ignore y_2 since most of the variance in the data comes from y_1.

Conversely, you can reconstruct the original data from the scores by rotating back:

$$\mathbf{x} = P\mathbf{y}. \tag{56}$$

If we keep only the first m components, the ones with the largest eigenvalues, and write $P_m = [\hat{\mathbf{e}}_1, \ldots, \hat{\mathbf{e}}_m]$, then

$$\mathbf{y}_m = P_m^{\mathsf{T}}\mathbf{x}, \qquad \mathbf{x} \approx P_m\mathbf{y}_m, \tag{57}$$

which is the basic dimensionality reduction idea: we represent each data point using only m numbers while losing as little variance as possible. This I am sure you'll hear about in more detail and more coherently in your data visualization unit!

[1] I am playing the usual mathematicians trick here of not being careful about the names of the variables, before I called these x and y but now I want $\mathbf{x} = (x_1, x_2)^{\mathsf{T}}$ for the data in the original coordinates and $\mathbf{y} = (y_1, y_2)^{\mathsf{T}}$ for the scores, the data in the new coordinates.

The 2024 Irish Election

Elections to the Dáil, the Irish parliament uses the greatest voting system every created: single transferable vote. Clearly England uses the worst, first past the post; which is obviously terrible and not worth discussing. The advantage of STV over the sort of list systems employed in Europe is more subtle; a list system tries to match the number of representative to the proportion of people voting for them, if 25% of people support party A then party A should have roughly 25% of the seats. STV does something better, it represents the division inside the hearts of individuals, rather than the division across the electorate; it is a beautiful idea, the make up of the Dáil represents not just how views change from person to person, but the conflicting views inside each individual. I urge you to look at how STV works. For our purposes though STV has the advantage over first past the post that it allows for lots of political parties and gives no reason to vote "strategically" so the vote share is a good measure of view-point. At the same time, unlike a list system, the voting is all on a constituency basis, so there are data points: the 43 constituencies.

We are going to ignore for now the details of STV and look at only the first preference vote, each person marks with a one the person they would mostly like to see elected in their constituency. Ireland has 43 constituencies and elects 174 representatives, called TDs or Teachtaí Dála, to the Dáil. There are a lot of parties, the two historic parties of power Fianna Fáil and Fine Gael, both just right of centre but in different ways, historically one represented the Catholic working class and professionals, the other wealthy farmers and business people. There is the insurgent Sinn Féin, actually the oldest party in the state but never in power since it lost the Irish Civil War in 1922, a few left wing parties, such as the Labour Party, the old party of unionised labour, the Social Democrats and People before Profit, along with some right wing parties such as Aontú and Independent Ireland. Many constituencies also have independent candidates who focus on local issues, for example politics in Kerry is dominated by a single family, the Healy-Raes who have few principles other than always wearing cloth caps and trying to get money for Kerry.

There was an election in 2024; it saw the Green Party lose most of its seats since it had annoyed its supporters by joining a coalition government; Sinn Féin didn't do as well as had been expected, in short, their peculiar relationship to armed violence and the support of liberal politics left them

with a peculiar, and unstable, mixture of supporters. A joint government formed of Fianna Fáil and Fine Gael remained in power, this coalition included some independents and dropped the Green Party. All this means that the data for this election includes 43 data points, one for each constituency and each data point is eleven dimensional, corresponding to voter share for 10 political parties and a final category of fringe parties and independents. Not every party runs candidates in every constituency, for convenience the blank entries are replaced with zeros.

If you do PCA on these data you end up with 11 eigenvalues, strongly dominated by the first three or four, after that the get small. This indicates that there are probably three or four main factors that determine how people vote. By doing PCA, in this case using the correlation matrix rather than the covariance data, this is a technical detail made necessary by the fact the numbers always add up to one. Fig. 2 shows the result for the first and second component coloured by the third. Anyone who isn't Irish will have to trust me, but the three components are very open to interpretation.

The up-down component, the second, is a conservative to liberal axis; Dublin Rathdown is a very bougie area, historically liberal on social issues, Cavan-Monaghan is a farming area, hard land too and problems compounded by proximity with the border which during the Troubles led, well, to trouble. It is a socially conservative area. The left-right axes measures how mixed between town and country the constituency is; Galway West, where I am from, includes the city of Galway, a fun loving music filled place popular with tourists and poets, and Connemara, a desolate country area, empty, rural and also popular with tourists and poets, but for different reasons. Dublin North West in contrast is all streets and semi-detached houses and corner shops. The third component picks out the presence of a strong local candidate; in Tipperary North that's Seamus Healy, in Dublin Bay South it is Ivana Bacik, who isn't an independent, but whose personal popularity means she did much better than you'd expect.

Glossary

1 and **0** these are an 'abused notation', I am relying on your common sense to guess what they mean in context, **1** might mean

$$\mathbf{1} = \mathrm{diag}(1, 1, \ldots, 1) \tag{58}$$

or it might be a vector with just ones, similarly $\mathbf{0}$ might be a matrix with only zeros or it might be vector with only zeros. Mathematics, the supposed exactest of sciences, is often filled with 'abused notation'; it is an interesting development that mathematicians these days are creating a computer language approach called `Leen` that allows you, or forces you, to write down mathematics in an exact way, which is useful and good for mathematics, but not for communicating mathematics where stopping all the time to give definitions can make things very dry and boring to read.

So called blackboard bold is sometimes used for unit matrices, so $\mathbb{1}_n$ would stand for the unit matrix. There is a lot of variability on whether matrices are written in bold; I did a mathematics degree so I don't, but I use bold for vectors in type and an underline on the page; engineers, and often computer scientists, have different conventions. You just need to accept this!

There is a handy approach to switching between matrix notation and component notation:

$$A = [a_{ij}] \tag{59}$$

means we are going to call the matrix A and its components a_{ij} so for 2×2:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \tag{60}$$

and, of course, this isn't a linear algebra unit so I'll remark in passing that

$$A^{-1} = \frac{1}{\det A} \begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix} \tag{61}$$

where $\det A = a_{11}a_{22} - a_{12}a_{21}$ with more complicated formulas for bigger matrix and the usual general statement that you should never code you own matrix algebra code, always use a library, a lot of people have spent a lot of time making them fast because more of computation is matrix computation.

The symbol δ_{ij} is the *Kronecker delta*:

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases} \tag{62}$$

The *covariance matrix*, if $\mathbf{X} = (X_1, X_2, \ldots, X_n)^\mathsf{T}$ is a vector of random variables

$$\Sigma = \mathrm{Cov}(\mathbf{X}) = \mathbb{E}\left[(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})^\top\right], \qquad \mu_i = \mathbb{E}[X_i]. \tag{63}$$

or, component-wise

$$\Sigma_{ij} = \mathrm{Cov}(X_i, X_j) = \mathbb{E}[(X_i - \mu_i)(X_j - \mu_j)]\,. \tag{64}$$

The *correlation matrix* is a *whitened* version of this:

$$\sigma_i^2 = \mathrm{Var}(X_i) = \Sigma_{ii}, \qquad \sigma_i = \sqrt{\Sigma_{ii}}. \tag{65}$$

and let

$$D = \mathrm{diag}(\sigma_1, \ldots, \sigma_n). \tag{66}$$

then

$$R = D^{-1}\Sigma D^{-1}. \tag{67}$$

or component-wise

$$R_{ij} = \frac{\mathrm{Cov}(X_i, X_j)}{\sigma_i\,\sigma_j} = \frac{\Sigma_{ij}}{\sqrt{\Sigma_{ii}\Sigma_{jj}}}. \tag{68}$$

When you are doing PCA you should use the covariance if that units are the same for all the dimensions and the data variance is somehow comparable, and correlation otherwise. The election data is a sort of edge case, the correlation matrix produces better results because some of the parties have a much bigger vote than others, there is also a technical complication related to the entries in the data vector always adding up to one.

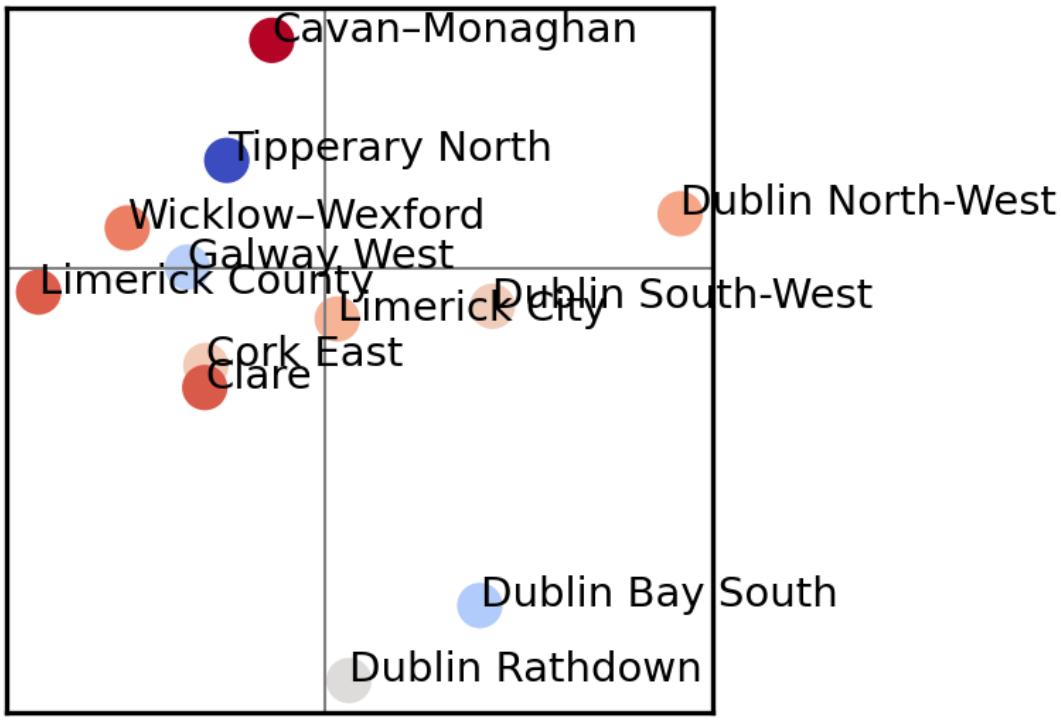

Figure 2: This shows the first three principal components, the first in the x-axis, the second, the y-axis and the third by colour. Only 12 data points, picked randomly, are shown otherwise it is hard to read. The y-axis is clearly a left-right axis, Dublin-Rathdown is a very liberal place, Cavan-Monaghan very conservative; the x-axis is not as clear but seems to measure diversity of opinion, the lefter points correspond to more complex places mixing rural and urban areas, for example. The third component, illustrated by colour, is interesting, it seems to correspond to independent candidates doing well, the two blues places, Galway West and Tipperary North have strong local candidates, Seamus Healy in Tipperary North and Catherine Connolly, who subsequently ran successfully for the presidency, in Galway West.