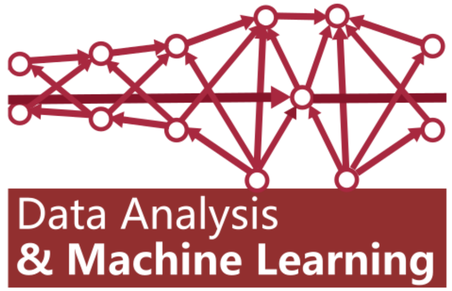


Data Analysis and Machine Learning 4

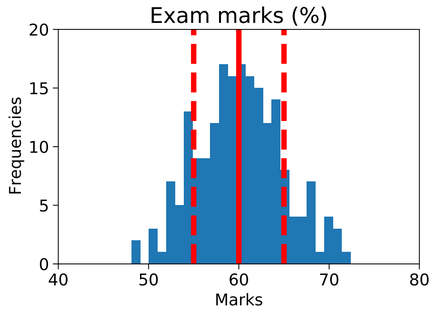
Week 3: Preprocessing, PCA, clustering

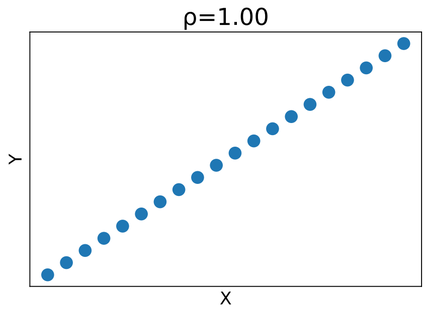
Elliot J. Crowley, 30th January 2023

THE UNIVERSITY *of* EDINBURGH

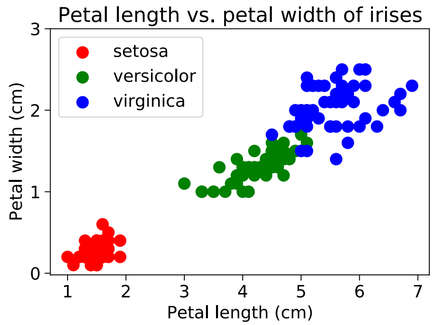
Recap

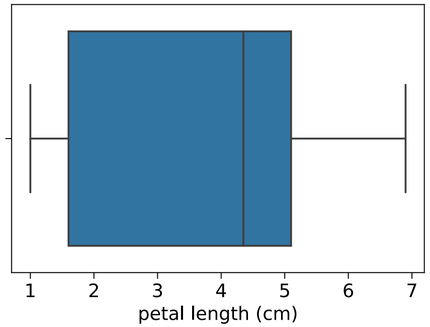
- We reviewed summary statistics for datasets





- We considered different ways to visualise data





This week

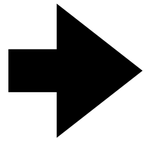
- You will learn how to preprocess data so it can be used for various algorithms
- There will be some linear algebra revision
- You will learn about PCA and how it can be used for dimensionality reduction
- You will find out how to cluster data using the K-means algorithm

Preprocessing

Matrix inputs

- PCA and many machine learning (ML) methods require a matrix input
- Our dataset must be represented by a matrix of real continuous values
- Given tabular data, we need to convert it into such a matrix

	Height (cm)	Age	Favourite colour
0	185	32	blue
1	193	70	red
2	147	77	brown
3	163	26	blue

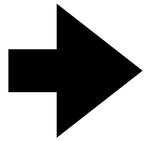


?

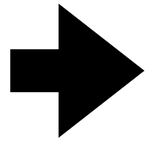
Representing a dataset as a matrix

- We have tabular data with N data items (rows) and C attributes (cols)
- For ease of exposition, we will drop attributes that don't correspond to continuous variables
- If there are now D attributes we can represent the dataset by a $N \times D$ matrix

	Height (cm)	Age	Favourite colour
0	185	32	blue
1	193	70	red
2	147	77	brown
3	163	26	blue



	Height (cm)	Age
0	185	32
1	193	70
2	147	77
3	163	26



$$\mathbf{X} = \begin{bmatrix} 185 & 32 \\ 193 & 70 \\ 147 & 77 \\ 163 & 26 \end{bmatrix}$$

$$\mathbf{X} \in \mathbb{R}^{N \times D}$$

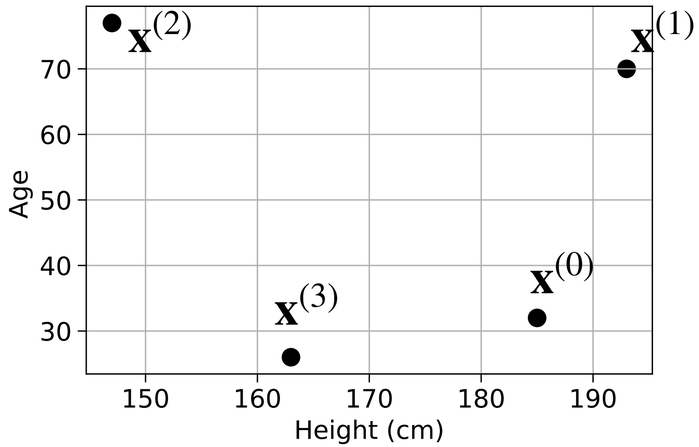
Representing data points as vectors

- We are representing our dataset using a $N \times D$ **dataset matrix** $\mathbf{X}$
- Each row is a data item or **data point** that lives in D-dimensional space
- Let's denote these as $\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N-1)}$ or $\{\mathbf{x}^{(n)}\}_{n=0}^{N-1}$. They are **vectors**

	Height (cm)	Age
0	185	32
1	193	70
2	147	77
3	163	26

$$\mathbf{X} = \begin{bmatrix} 185 & 32 \\ 193 & 70 \\ 147 & 77 \\ 163 & 26 \end{bmatrix}$$

$$\mathbf{X} \in \mathbb{R}^{N \times D}$$



$$\mathbf{x} \in \mathbb{R}^{D}$$

Credit to Iain Murray for the notation

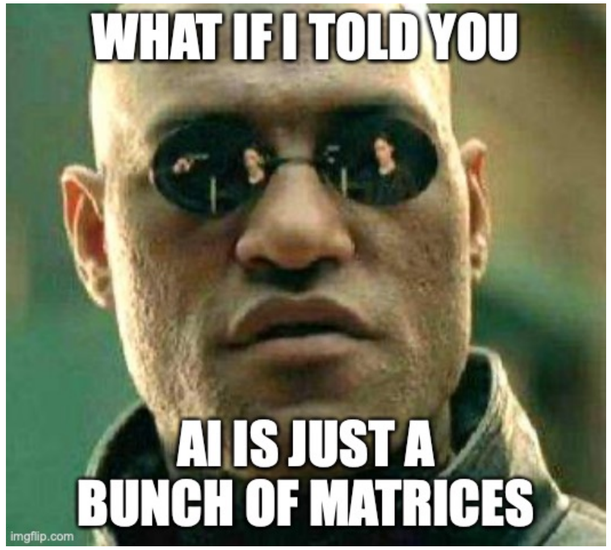
Data points are column vectors

- It is standard with tabular data to have the rows as data points
- But in ML literature it is convention to denote all vectors including data points $\mathbf{x}$ as **column vectors**
- It is also convention to represent a dataset as $\mathbf{X} \in \mathbb{R}^{N \times D}$ (in the same way we just did) where the **rows** are those data points
- **Just be aware of this peculiarity!**

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{D-1} \end{bmatrix} \qquad \mathbf{x}^{(n)} = \begin{bmatrix} x_0^{(n)} \\ x_1^{(n)} \\ \vdots \\ x_{D-1}^{(n)} \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} \mathbf{x}^{(0)\top} \\ \mathbf{x}^{(1)\top} \\ \mathbf{x}^{(2)\top} \\ \vdots \\ \mathbf{x}^{(N-1)\top} \end{bmatrix} = \begin{bmatrix} x_0^{(0)} & x_1^{(0)} & \cdots & x_{D-1}^{(0)} \\ x_0^{(1)} & x_1^{(1)} & \cdots & x_{D-1}^{(1)} \\ x_0^{(2)} & x_1^{(2)} & \cdots & x_{D-1}^{(2)} \\ \cdots & \cdots & \ddots & \vdots \\ x_0^{(N-1)} & x_1^{(N-1)} & \cdots & x_{D-1}^{(N-1)} \end{bmatrix}$$

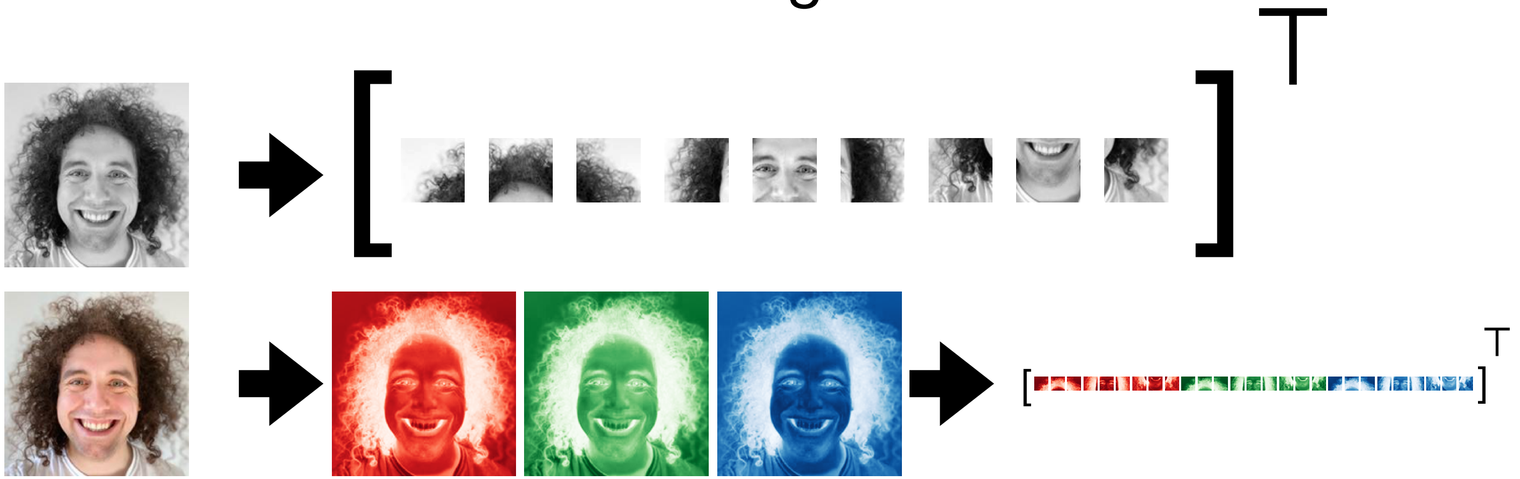
Why vectors?

- We can now use the machinery of linear algebra for PCA and ML
- **Matrices linearly transform vectors**
- Computers are very good at matrix multiplication
- Neural networks consist of multiple matrices (See Week 9!)



Can we represent other types of data as vectors?

- Yes! We can flatten or ***vectorise*** images



- We can represent text data as a histogram of word counts (a bag of words) e.g. [# “I”, # “like”, # “sausage”, # “hate”]

I like sausage

$$[1 \quad 1 \quad 1 \quad 0]^\top$$

I hate sausage

$$[1 \quad 0 \quad 1 \quad 1]^\top$$

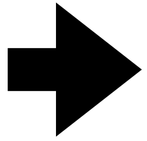
sausage sausage

$$[0 \quad 0 \quad 2 \quad 0]^\top$$

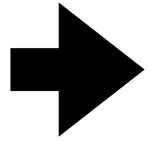
Standardising your data

- Measurements of different variables can have vastly different scales
- We want to compare variables like-for-like and not let those with large values dominate
- The solution is to **standardise** your data
- We want each column of $\mathbf{X}$ to have a mean of 0 and a SD of 1

	Height (cm)	Age	Salary (£)
0	190	44	25000
1	143	36	29000
2	152	20	100000
3	178	56	67000



$$\mathbf{X} = \begin{bmatrix} 190 & 44 & 25000 \\ 143 & 36 & 29000 \\ 152 & 20 & 100000 \\ 178 & 56 & 67000 \end{bmatrix}$$



?

Standardising your data

- We want each column of $\mathbf{X}$ to have a mean of 0 and a SD of 1
- For each column, compute the mean and SD
- Then subtract the mean from each value and divide by SD
- **This is essential for PCA and many ML algorithms**

I will use $\sum_n$ to mean "sum over all n"

$$\mathbf{X}_{old} = \begin{bmatrix} x_0^{(0)} & x_1^{(0)} & \dots & x_{D-1}^{(0)} \\ x_0^{(1)} & x_1^{(1)} & \dots & x_{D-1}^{(1)} \\ \dots & \dots & \ddots & \vdots \\ x_0^{(N-1)} & x_1^{(N-1)} & \dots & x_{D-1}^{(N-1)} \end{bmatrix}$$

$$\mu_j = \frac{1}{N} \sum_n \mathbf{x}_j^{(n)}$$

$$\sigma_j^2 = \frac{1}{N} \sum_n (\mathbf{x}_j^{(n)} - \mu_j)^2$$

$$\mathbf{X}_{new} = \begin{bmatrix} \frac{x_0^{(0)} - \mu_0}{\sigma_0} & \frac{x_1^{(0)} - \mu_1}{\sigma_1} & \dots & \frac{x_{D-1}^{(0)} - \mu_{N-1}}{\sigma_{N-1}} \\ \frac{x_0^{(1)} - \mu_0}{\sigma_0} & \frac{x_1^{(1)} - \mu_1}{\sigma_1} & \dots & \frac{x_{D-1}^{(1)} - \mu_{N-1}}{\sigma_{N-1}} \\ \dots & \dots & \ddots & \vdots \\ \frac{x_0^{(N-1)} - \mu_0}{\sigma_0} & \frac{x_1^{(N-1)} - \mu_1}{\sigma_1} & \dots & \frac{x_{D-1}^{(N-1)} - \mu_{N-1}}{\sigma_{N-1}} \end{bmatrix}$$

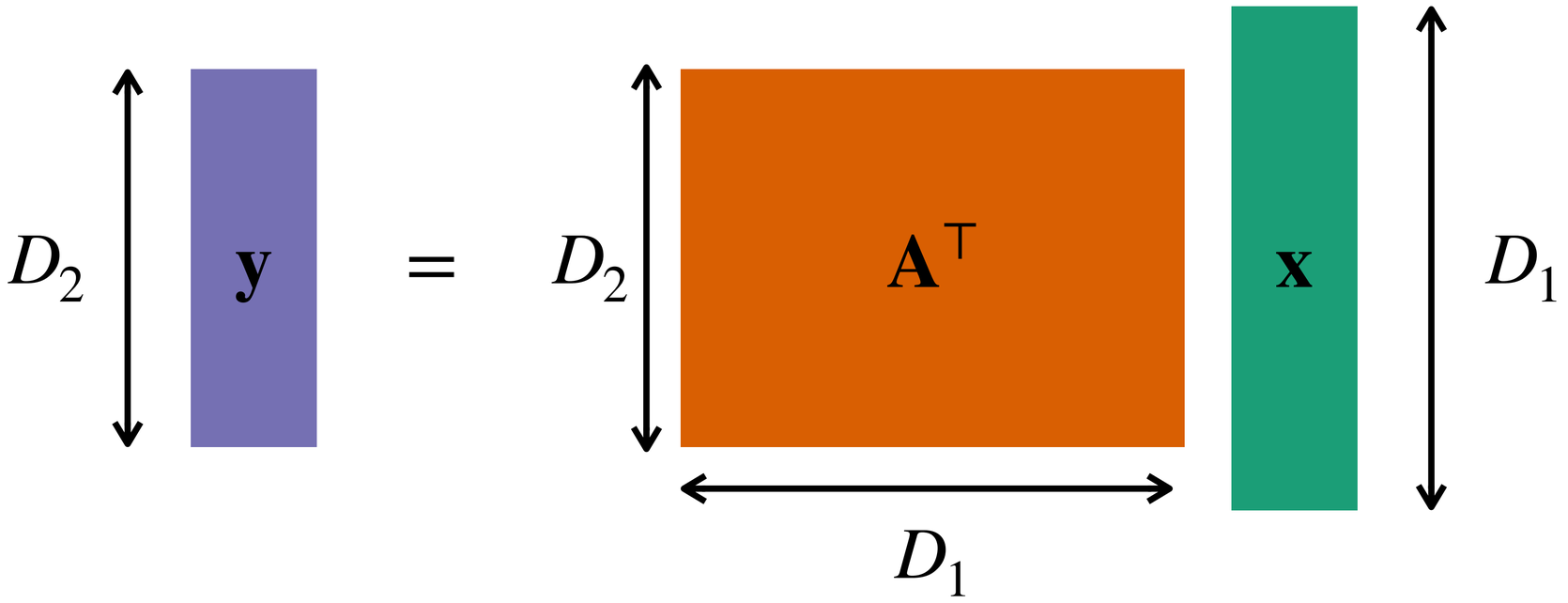
Normalising vs. standardising

- Nomenclature can vary but in this course **standardising** refers to scaling each variable to zero mean and unit variance
- We can do other forms of scaling e.g. divide each variable by its maximum value
- We will refer to other forms of scaling as **normalising**
- Generally, anything that gets different variables to similar ranges is fine **just make sure you do it!**

Transformation matrices (Linear algebra revision)

Using a matrix to transform a vector

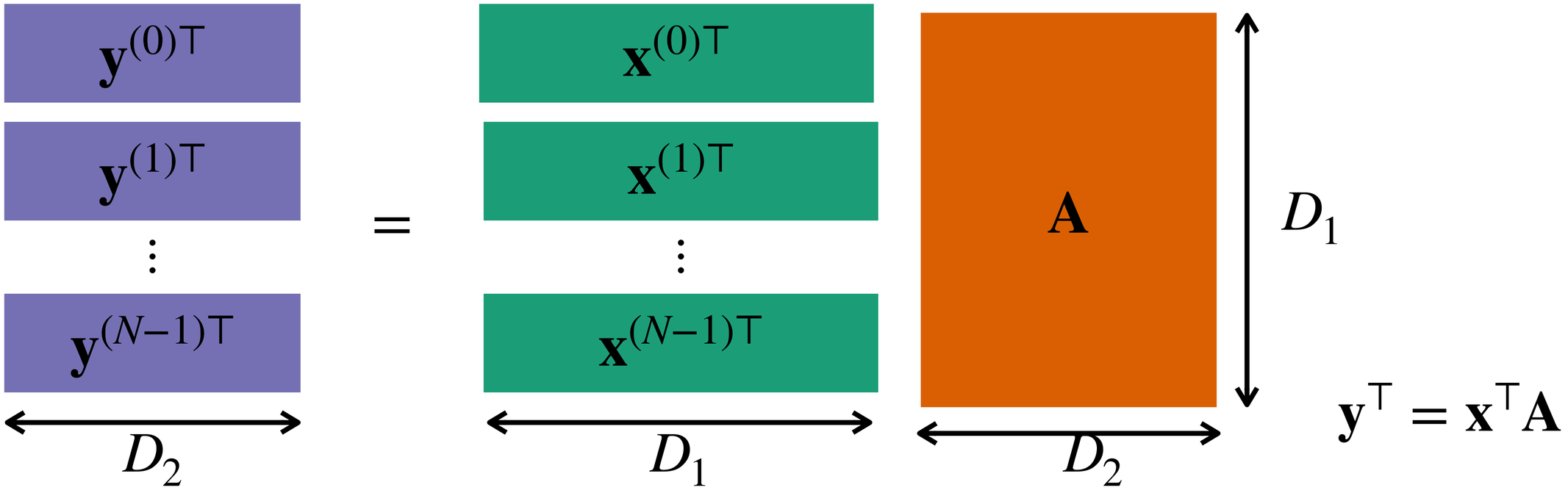
- Assume we have a data point $\mathbf{x} \in \mathbb{R}^{D_1}$ and a matrix $\mathbf{A} \in \mathbb{R}^{D_2 \times D_1}$
- If we multiply $\mathbf{A}^\top$ by $\mathbf{x}$ then we get a transformed data point $\mathbf{y} \in \mathbb{R}^{D_2}$



$$\mathbf{y} = \mathbf{A}^\top \mathbf{x}$$

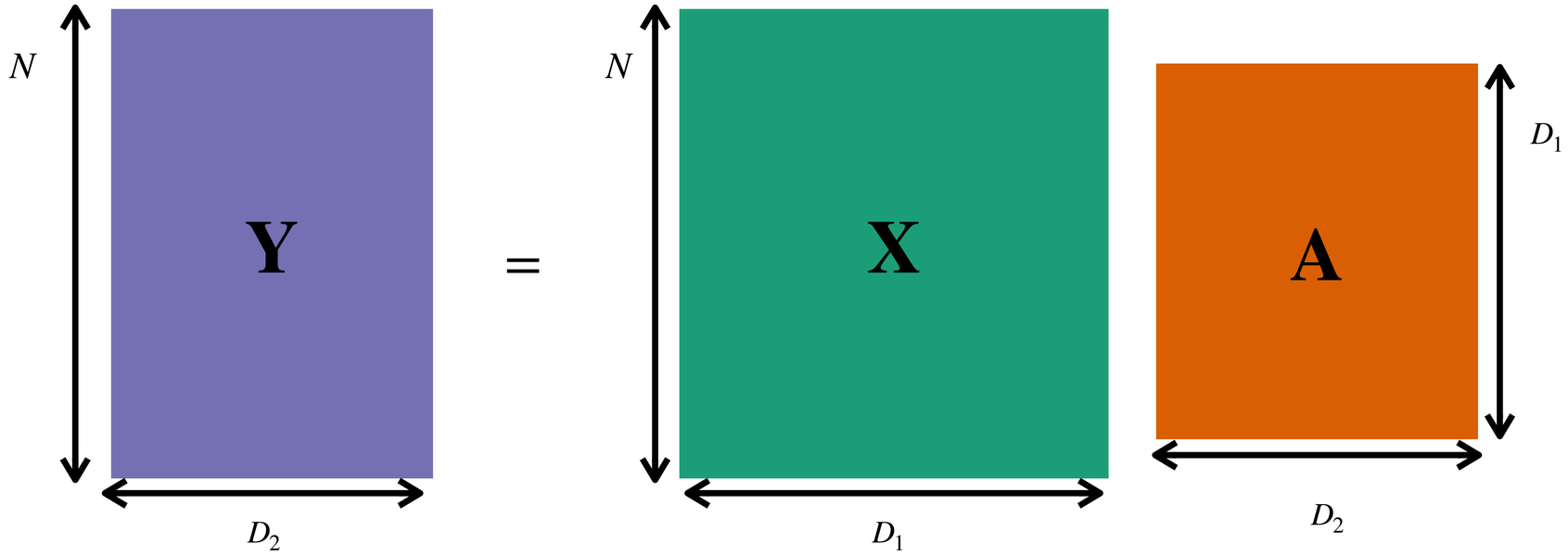
Using a matrix to transform multiple vectors

- If we want do this to multiple data points $\{\mathbf{x}^{(n)}\}_{n=0}^{N-1}$ then we can transpose each, and then post-multiply by $\mathbf{A}$
- This gives us the transposes of transformed data points $\{\mathbf{y}^{(n)}\}_{n=0}^{N-1}$



Matrix multiplication

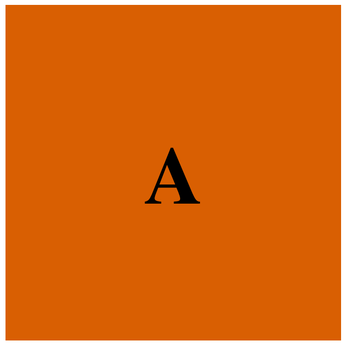
- We want to transform all our $\{\mathbf{x}^{(n)}\}_{n=0}^{N-1}$ into $\{\mathbf{y}^{(n)}\}_{n=0}^{N-1}$ in parallel
- We can form a dataset matrix $\mathbf{X} \in \mathbb{R}^{N \times D_1}$ and post-multiply by $\mathbf{A}$
- This gives us a transformed dataset matrix $\mathbf{Y} \in \mathbb{R}^{N \times D_2}$ whose rows are transformed data points



$$\mathbf{Y} = \mathbf{X}\mathbf{A}$$

Changing dimensionality

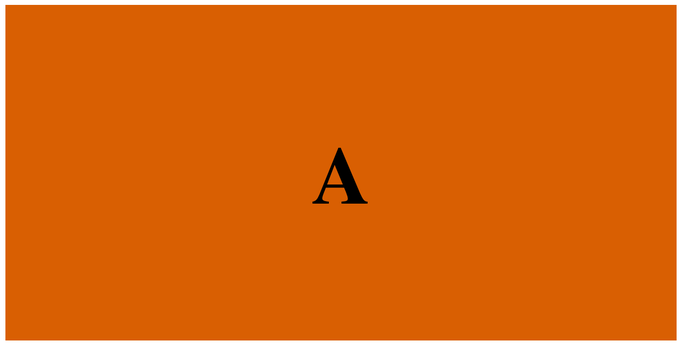
- $\mathbf{Y} = \mathbf{XA}$ transforms the vectors $\{\mathbf{x}^{(n)}\}_{n=0}^{N-1}$ into $\{\mathbf{y}^{(n)}\}_{n=0}^{N-1}$
- Recall that $\mathbf{A} \in \mathbb{R}^{D_2 \times D_1}$, $\mathbf{x} \in \mathbb{R}^{D_1}$, $\mathbf{y} \in \mathbb{R}^{D_2}$
- D_1 is fixed but D_2 can vary



$D_2 = D_1$

$\mathbf{A}$ square

Dimensionality of each $\mathbf{x}$ stays the same



$D_2 > D_1$

$\mathbf{A}$ wide

Dimensionality of each $\mathbf{x}$ goes up

$D_2 < D_1$

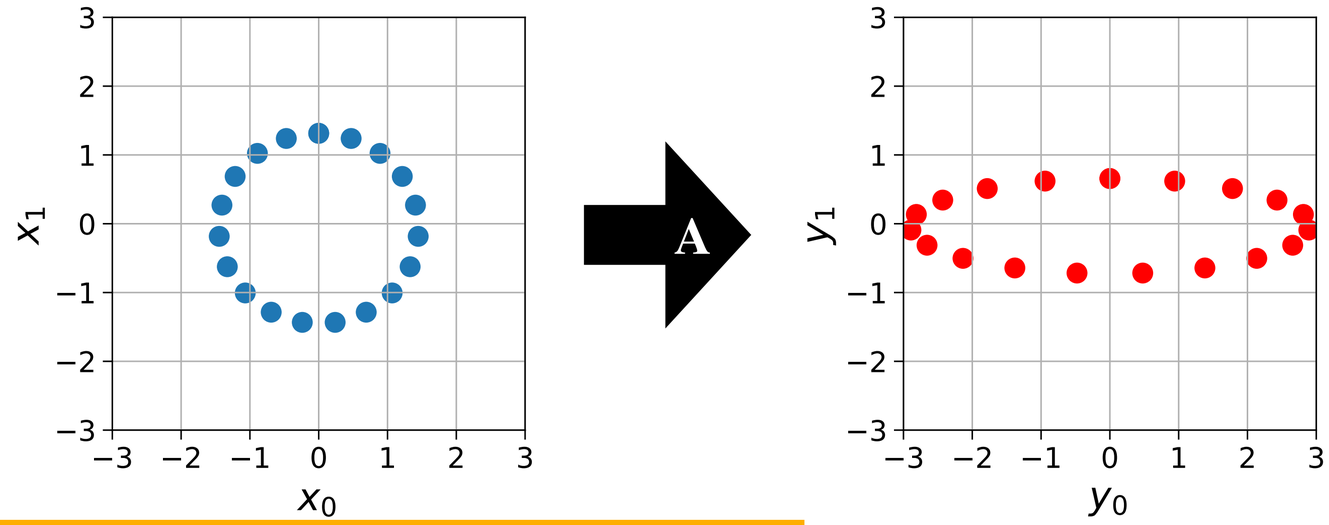
$\mathbf{A}$ thin

Dimensionality of each $\mathbf{x}$ goes down

Mapping in the same space

- Consider a dataset $\mathbf{X} \in \mathbb{R}^{20\times2}$ that consists of 2D points on a circle
- If $\mathbf{A} \in \mathbb{R}^{2\times2}$ then $\mathbf{XA}$ maps each point to a new point within 2D space
- These new points are the rows of $\mathbf{Y} \in \mathbb{R}^{20\times2}$ where $\mathbf{Y} = \mathbf{XA}$

$$\mathbf{A} = \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix}$$



x_0 and x_1 are the dimensions of $\mathbf{x}$ and so on

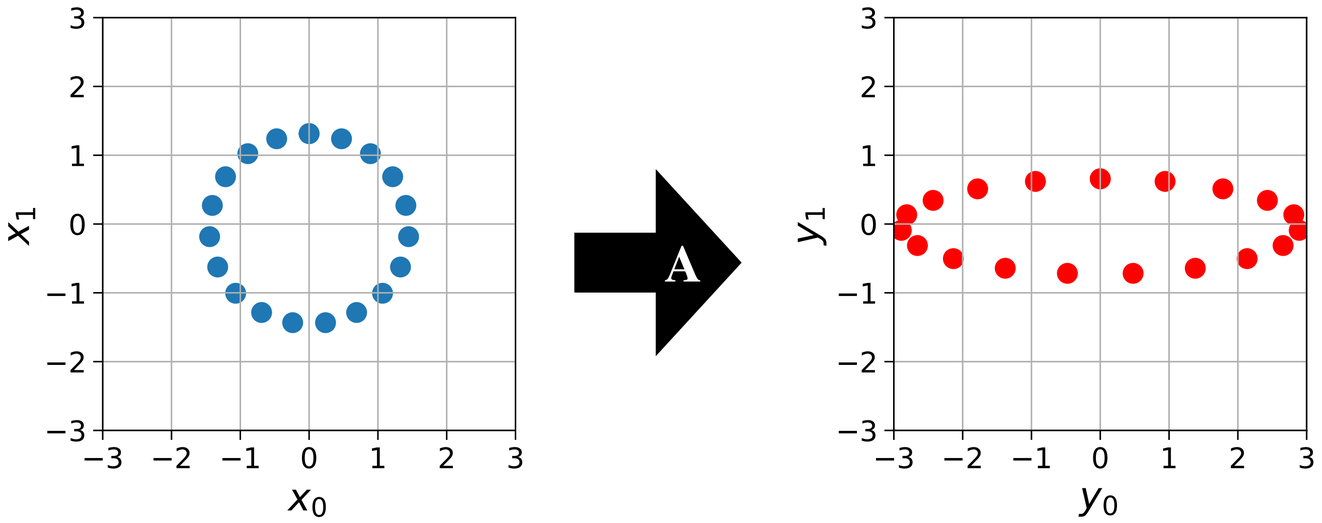
$\mathbf{Y} = \mathbf{XA}$ causes a change of basis

- New dimensions are linear combinations of old dimensions
- The columns of $\mathbf{A}$ are **basis vectors**

$$[y_0 \quad y_1] = [x_0 \quad x_1] \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix}$$

$$y_0 = 2x_0$$

$$y_1 = 0.5x_1$$



Orthogonality and orthonormality

- If the dot product of the basis vectors is 0 then they form an orthogonal basis
- If they are also unit norm then we call this an orthonormal basis

$$\mathbf{A} = [\mathbf{a}_0 \quad \mathbf{a}_1] = \begin{bmatrix} 2 & 0 \\ 0 & 0.5 \end{bmatrix}$$

$$\mathbf{a}_0 \cdot \mathbf{a}_1 = 2 \times 0 + 0 \times 0.5 = 0$$

$$\|\mathbf{a}_0\| = \sqrt{2^2 + 0^2} = 2$$

$$\|\mathbf{a}_1\| = \sqrt{0^2 + 0.5^2} = 0.5$$

Orthogonal

$$\mathbf{A} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\mathbf{a}_0 \cdot \mathbf{a}_1 = 0$$

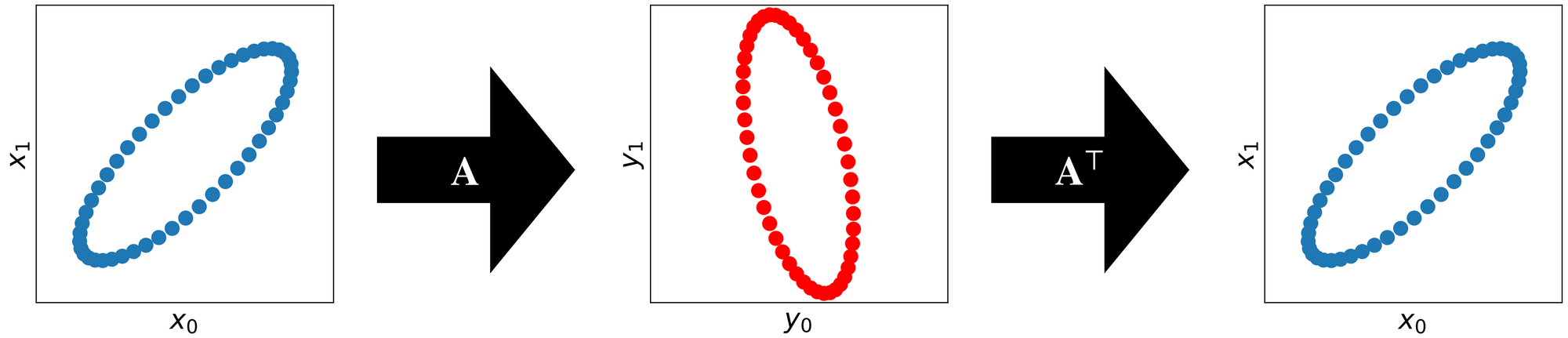
$$\|\mathbf{a}_0\| = 1$$

$$\|\mathbf{a}_1\| = 1$$

Orthonormal

Invertibility

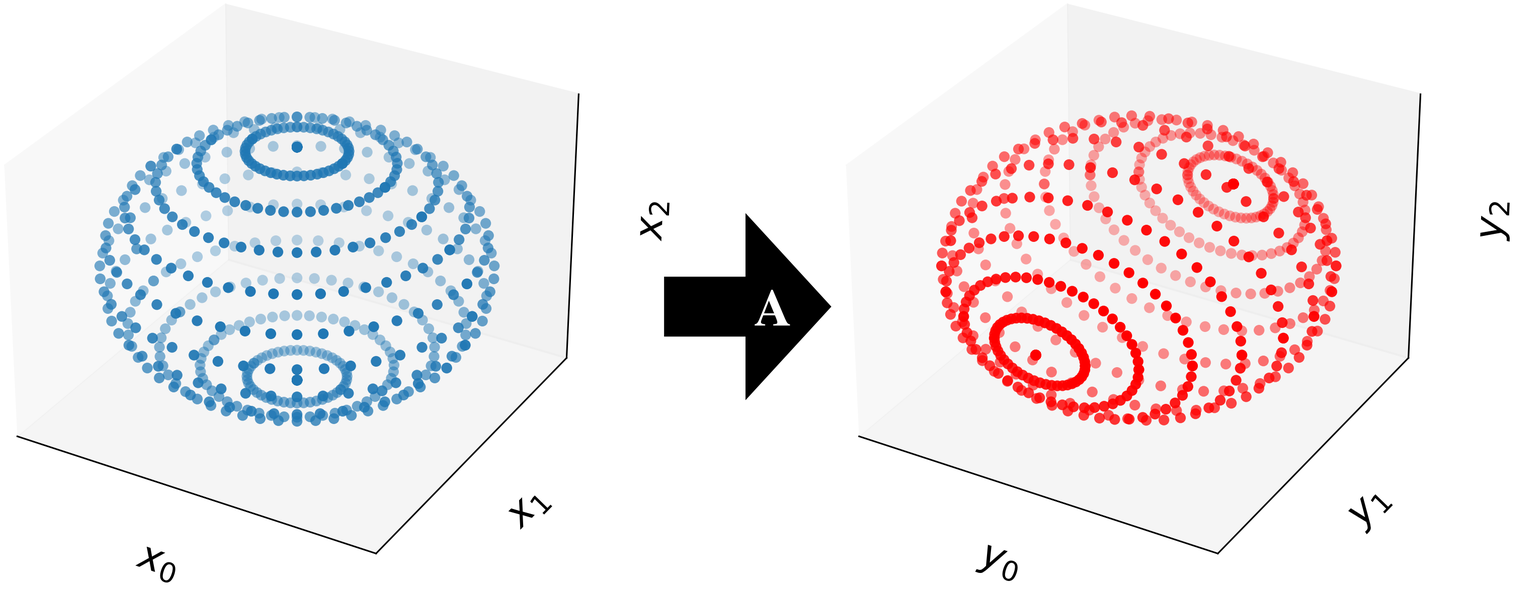
- A matrix $\mathbf{A}$ that forms an orthonormal basis is called an orthonormal matrix
- These are **invertible** which means no information is lost
- Their inverse is also their transpose i.e. $\mathbf{A}^{-1} = \mathbf{A}^{\top}$



Mapping in the same space … in 3D!

- Consider a dataset $\mathbf{X} \in \mathbb{R}^{625\times3}$ that consists of points on a sphere
- We can use a rotation matrix to rotate these points around an axis
- This is mapping each point to a new position within this 3D space

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$



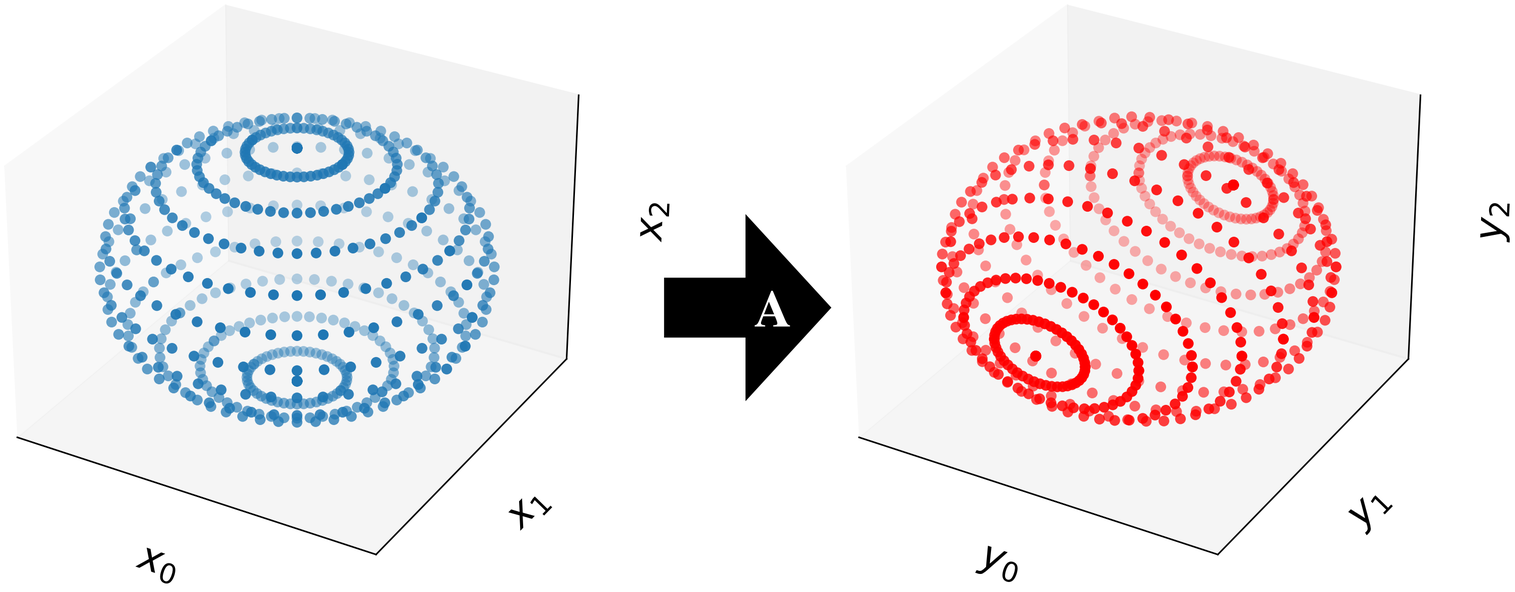
Change of basis

We have $\mathbf{Y} = \mathbf{XA}$ and for 90 degrees $\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$

$$y_0 = x_0$$

$$y_1 = -y_2$$

$$y_2 = x_1$$



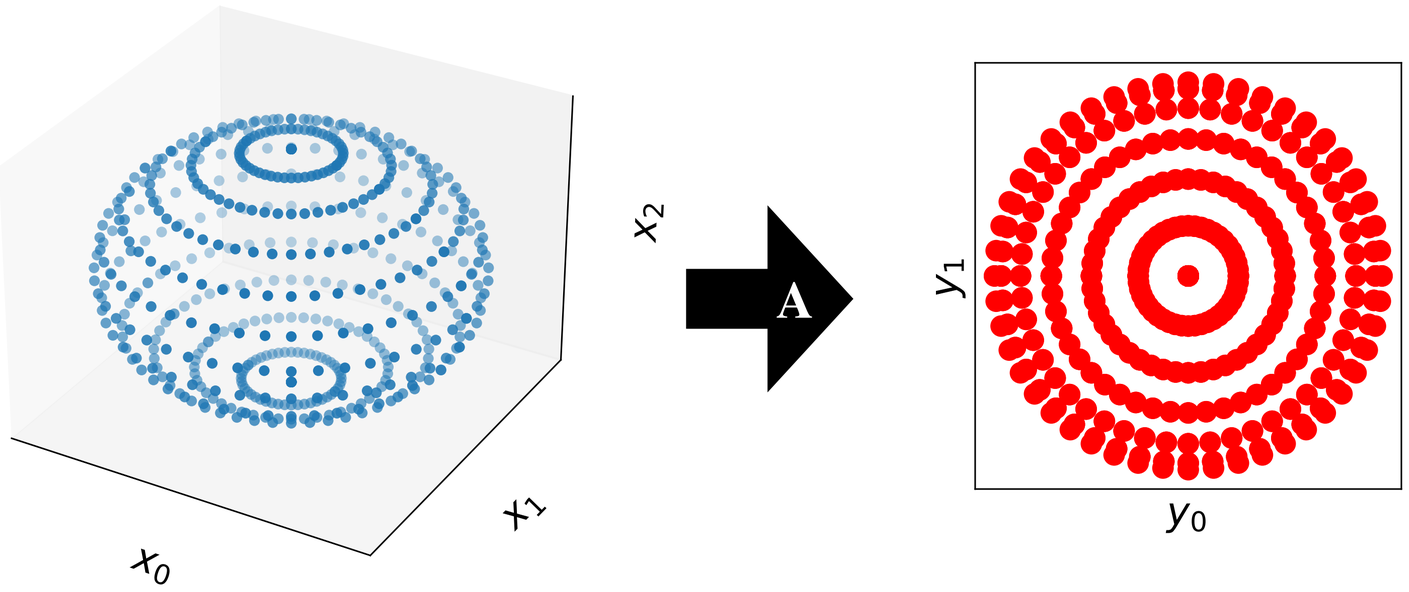
Dimensionality reduction

- If $\mathbf{A} \in \mathbb{R}^{3\times2}$ then $\mathbf{XA}$ maps each 3D point to a 2D point
- This throws away information

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

$$y_0 = x_0$$

$$y_1 = x_1$$



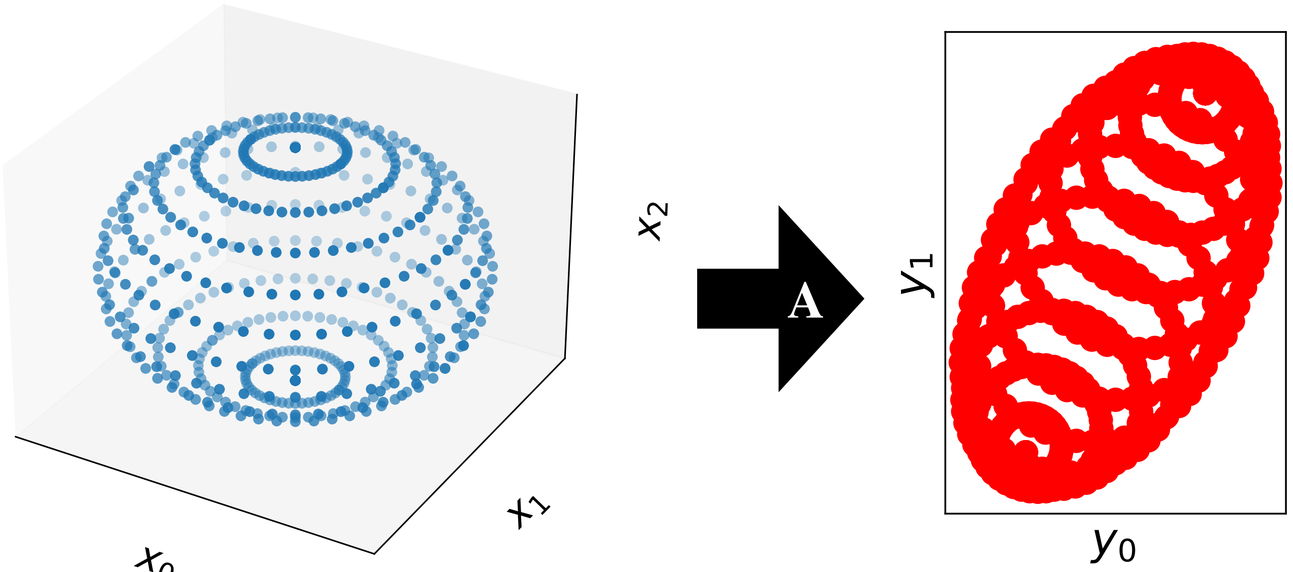
Dimensionality reduction

If $\mathbf{A} \in \mathbb{R}^{3\times2}$ then $\mathbf{XA}$ maps each 3D point to a 2D point

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 2 \end{bmatrix}$$

$$y_0 = x_0 + x_1 + x_2$$

$$y_1 = x_0 - x_1 + 2x_2$$



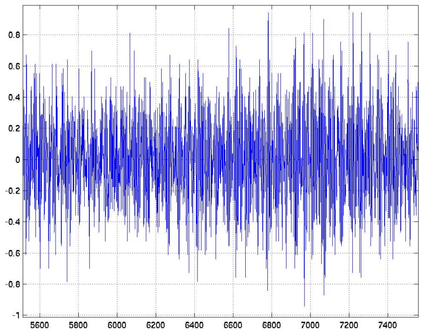
Principal Component Analysis (PCA)

Motivation for PCA

- Most data is high dimensional
- This makes it hard to visualise patterns across a whole dataset

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

Tables with >3 columns

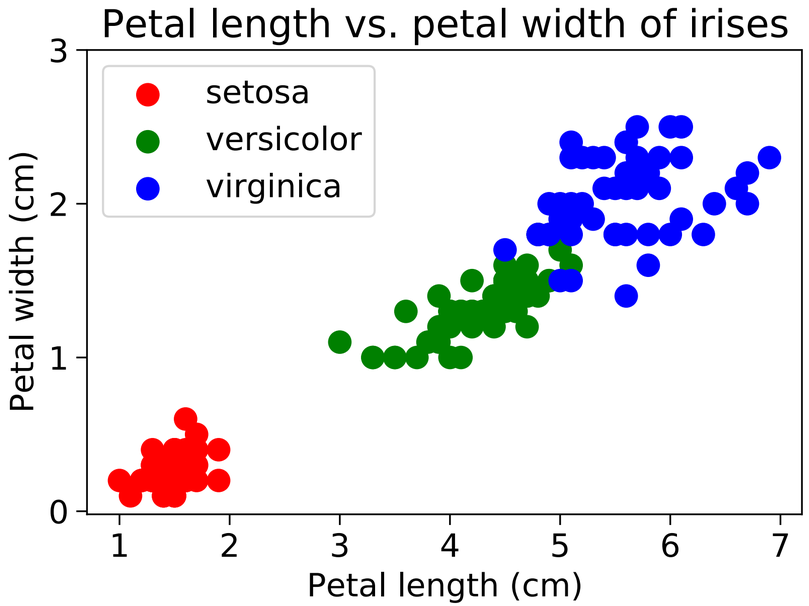


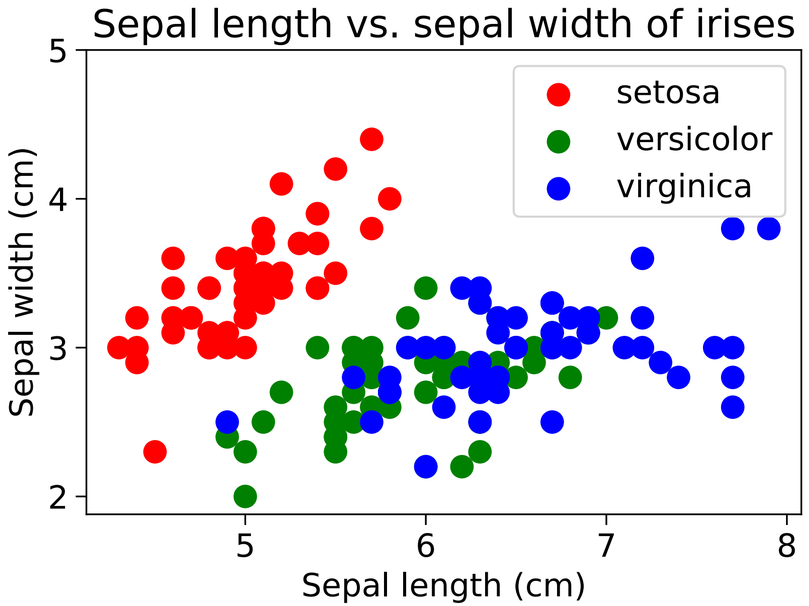
Time series with thousands of points

Images with millions of pixels

Motivation for PCA

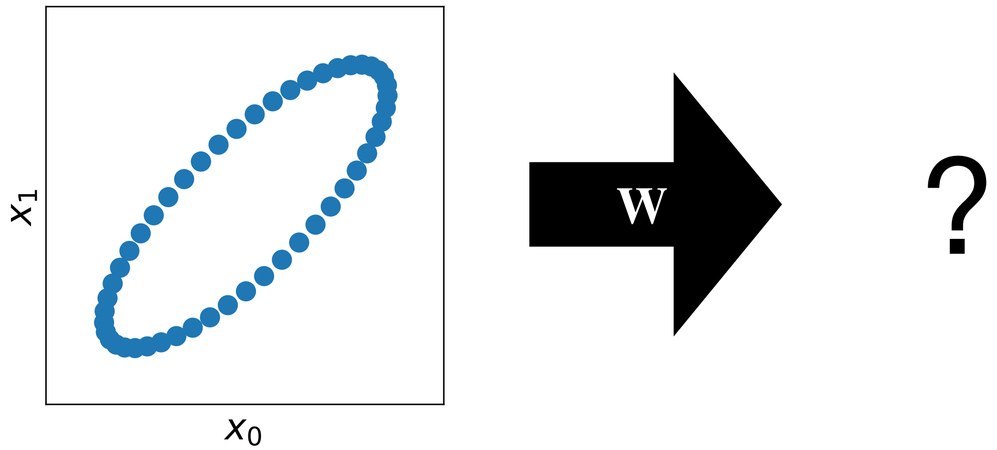
- We could use a transform to reduce the dimensionality of our data e.g. to 2D
- Then we could simply look at a scatter plot to find patterns
- But how do we know what the best transform is? (Spoilers: it involves PCA)





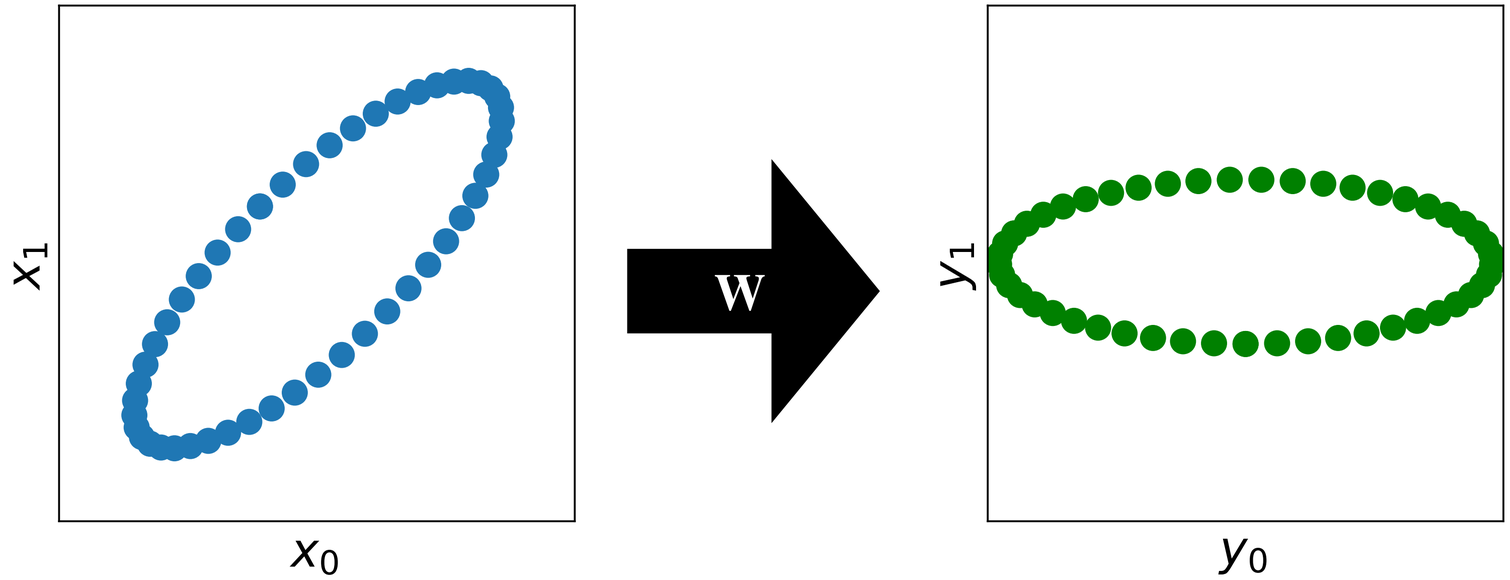
Principal Component Analysis (PCA)

- Consider a **standardised** dataset matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$
- PCA takes $\mathbf{X}$ and returns an **orthonormal** matrix $\mathbf{W} \in \mathbb{R}^{D \times D}$
- We can then transform $\mathbf{X}$ using $\mathbf{Y} = \mathbf{X}\mathbf{W}$
- The new dimensions $y_0, y_1, \ldots$ are linear combinations of the old dimensions $x_0, x_1, \ldots$



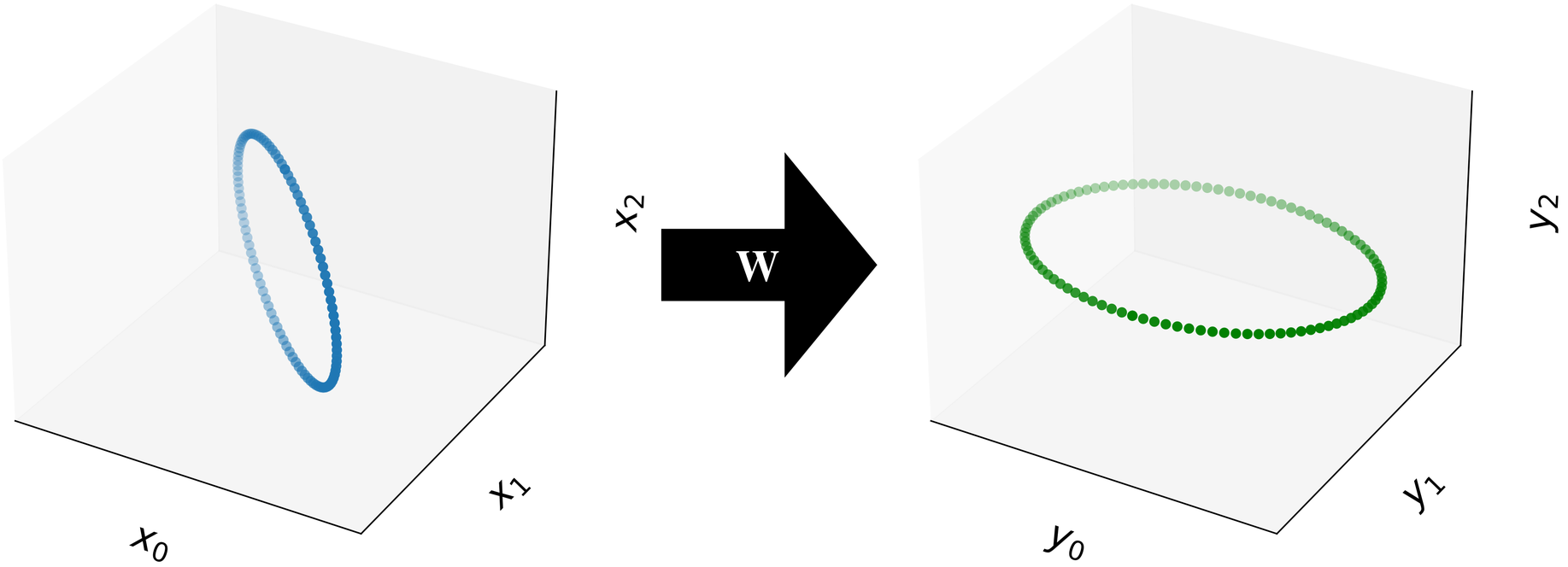
Maximum variance

- The data is transformed so there is as much variance as possible in y_0
- y_0 best explains the data in 1D



Arranging dimensions by decreasing variance

- There is the most variance in y_0 and there is the next most variance in y_1
- y_0 and y_1 best explain the data in 2D
- And so on!



Computing principal components

For a **standardised** dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$, PCA gives you a matrix $\mathbf{W} \in \mathbb{R}^{D \times D}$

The columns of $\mathbf{W}$: $\{\mathbf{w}_d\}_{d=0}^{D-1}$ are the **principal components** of the data

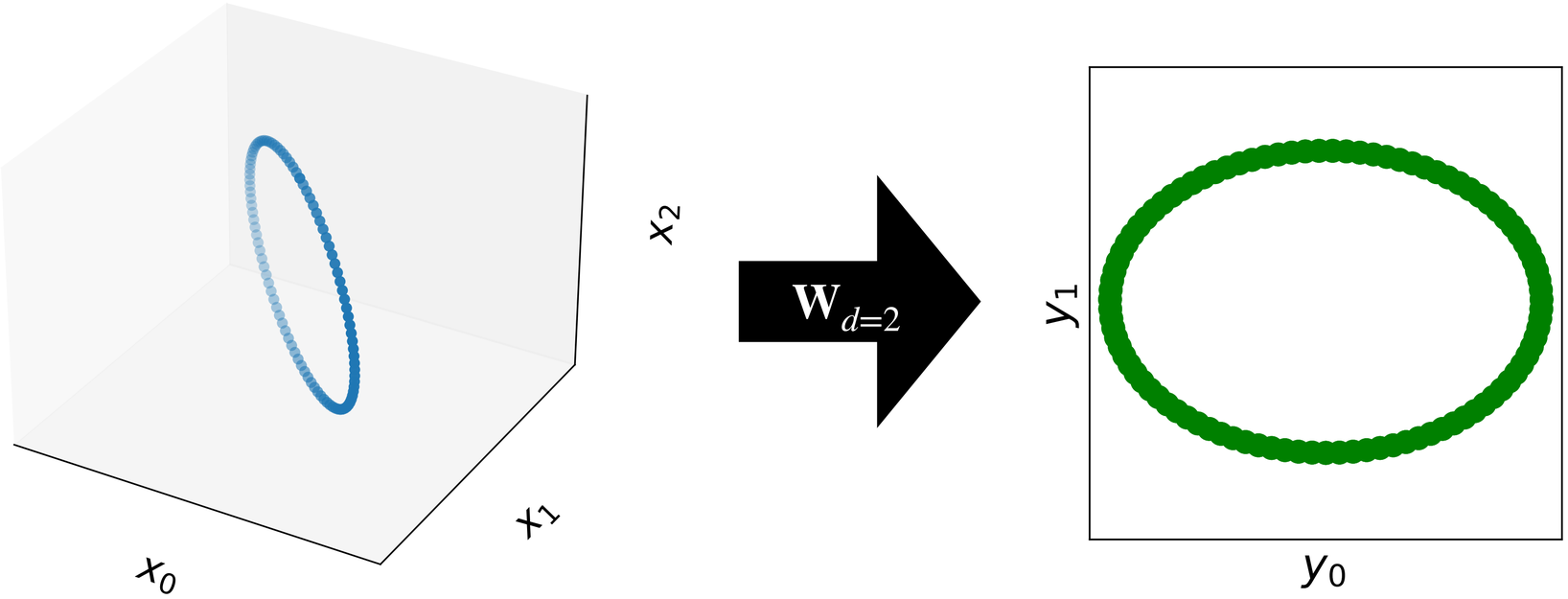
To compute these:

1. Construct the covariance matrix $\mathbf{\Sigma} = \frac{1}{N}\mathbf{X}^\top\mathbf{X}$
2. Eigendecompose $\mathbf{\Sigma}$ to eigenvalue, eigenvector pairs
3. Sort pairs by decreasing eigenvalue and denote as $\{\lambda_d\}_{d=0}^{D-1}, \{\mathbf{w}_d\}_{d=0}^{D-1}$

These vectors are
the principal components

PCA for dimensionality reduction

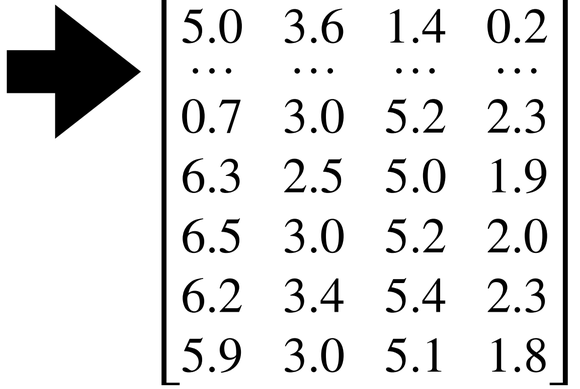
- PCA gives us $\mathbf{W} \in \mathbb{R}^{D \times D}$ where $\mathbf{W} = [\mathbf{w}_0 \quad \mathbf{w}_1 \quad \cdots \quad \mathbf{w}_{D-1}]$
- To reduce to $d < D$ dimensions we can just keep the first d columns
- e.g. $\mathbf{W}_{d=2} = [\mathbf{w}_0 \quad \mathbf{w}_1]$ would take our data to 2D using $\mathbf{Y} = \mathbf{X}\mathbf{W}_{d=2}$



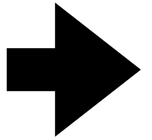
PCA for dimensionality reduction on irises

- The iris dataset contains 150 data points
- Let's take the numeric columns to form a dataset matrix $\mathbf{X} \in \mathbb{R}^{150\times4}$
- Make sure that $\mathbf{X}$ is standardised

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica



$$\begin{bmatrix} 5.1 & 3.5 & 1.4 & 0.2 \\ 4.9 & 3.0 & 1.4 & 0.2 \\ 4.7 & 3.2 & 1.3 & 0.2 \\ 4.6 & 3.1 & 1.5 & 0.2 \\ 5.0 & 3.6 & 1.4 & 0.2 \\ \cdots & \cdots & \cdots & \cdots \\ 0.7 & 3.0 & 5.2 & 2.3 \\ 6.3 & 2.5 & 5.0 & 1.9 \\ 6.5 & 3.0 & 5.2 & 2.0 \\ 6.2 & 3.4 & 5.4 & 2.3 \\ 5.9 & 3.0 & 5.1 & 1.8 \end{bmatrix}$$



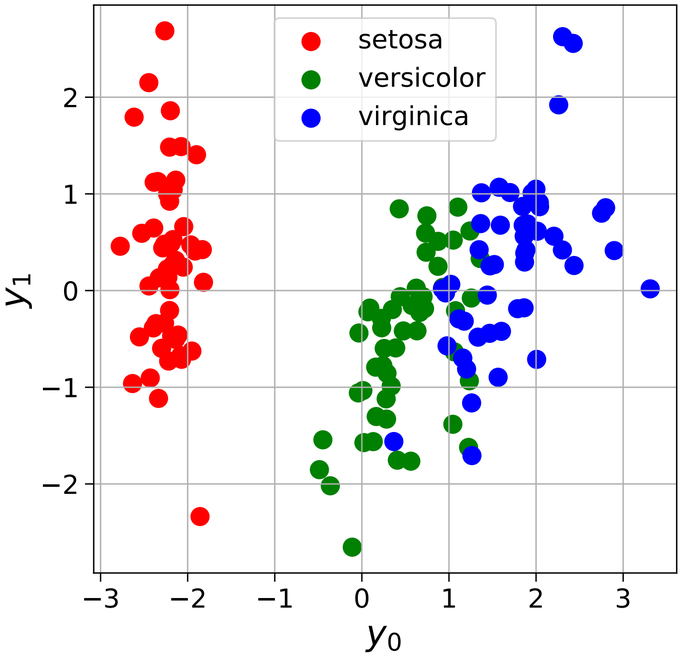
$$\begin{bmatrix} 5.1 & 3.5 & 1.4 & 0.2 \\ 0.9 & 1.0 & -1.3 & -1.3 \\ -1.1 & -0.1 & -1.3 & -1.3 \\ -1.4 & 0.3 & -1.4 & -1.3 \\ -1.5 & 0.1 & -1.3 & -1.3 \\ -1.0 & 1.2 & -1.3 & -1.3 \\ \cdots & \cdots & \cdots & \cdots \\ .0 & -0.1 & 0.8 & 1.4 \\ 0.6 & -1.3 & 0.7 & 0.9 \\ 0.8 & -0.1 & 0.8 & 1.1 \\ 0.4 & 0.8 & 0.9 & 1.4 \\ 0.1 & -0.1 & 0.8 & 0.8 \end{bmatrix}$$

PCA for dimensionality reduction on irises

- Use PCA to form $\mathbf{W} \in \mathbb{R}^{4\times4}$
- Now use $\mathbf{Y} = \mathbf{X}\,[\mathbf{w}_0 \quad \mathbf{w}_1]$ to project down to 2D
- Different species are distinguishable just by looking at y_0
- These new dimensions were found automatically

$$y_0 = -0.52x_0 - 0.27x_1 - 0.58x_2 + 0.56x_3$$

$$y_1 = -0.38x_0 + 0.92x_1 + 0.02x_2 + 0.07x_3$$



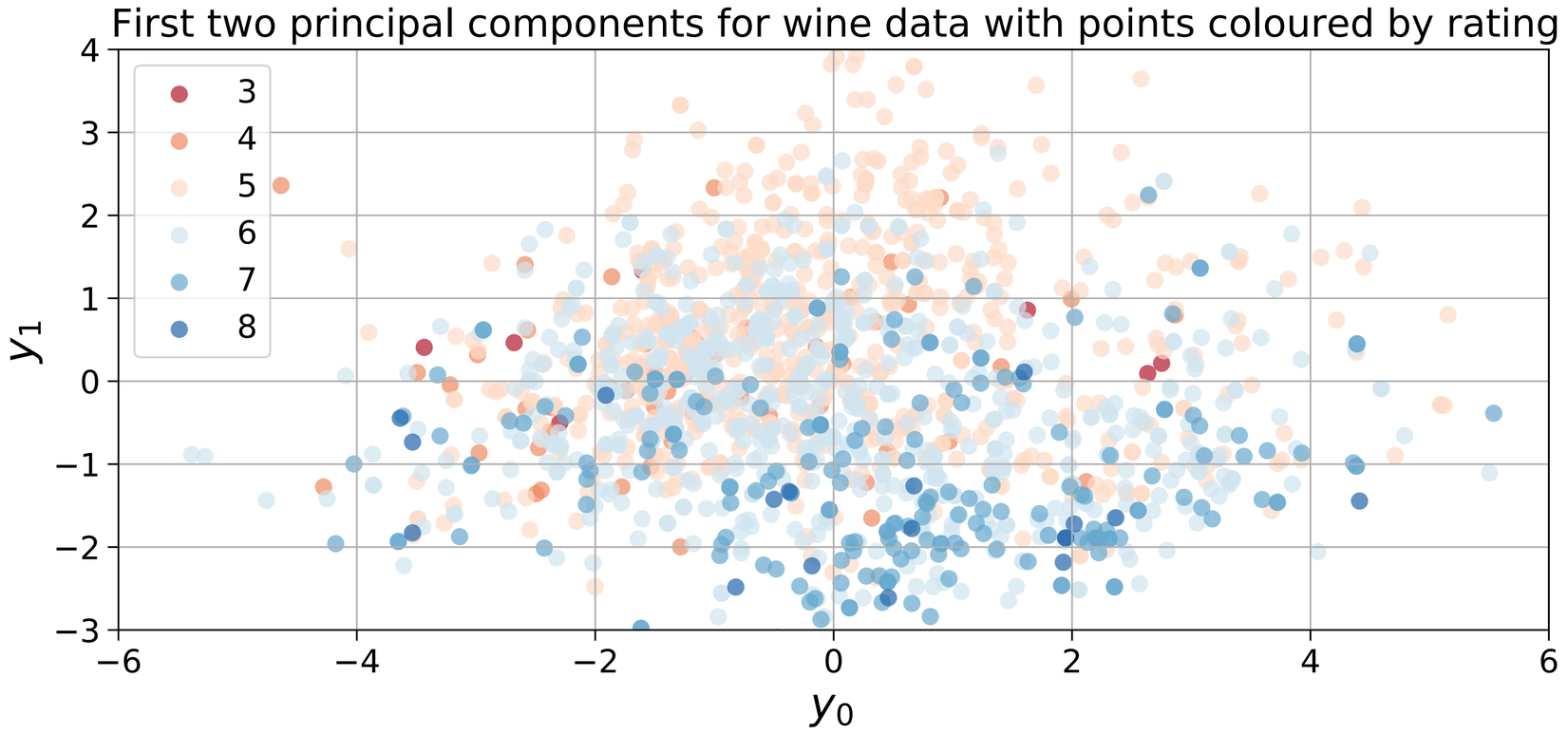
PCA for dimensionality reduction on wine

- We have a red wine dataset $\mathbf{X} \in \mathbb{R}^{1599 \times 11}$
- Each wine has also been scored by an expert between 0 and 10
- We can look at a few examples but it's hard to get the full picture

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

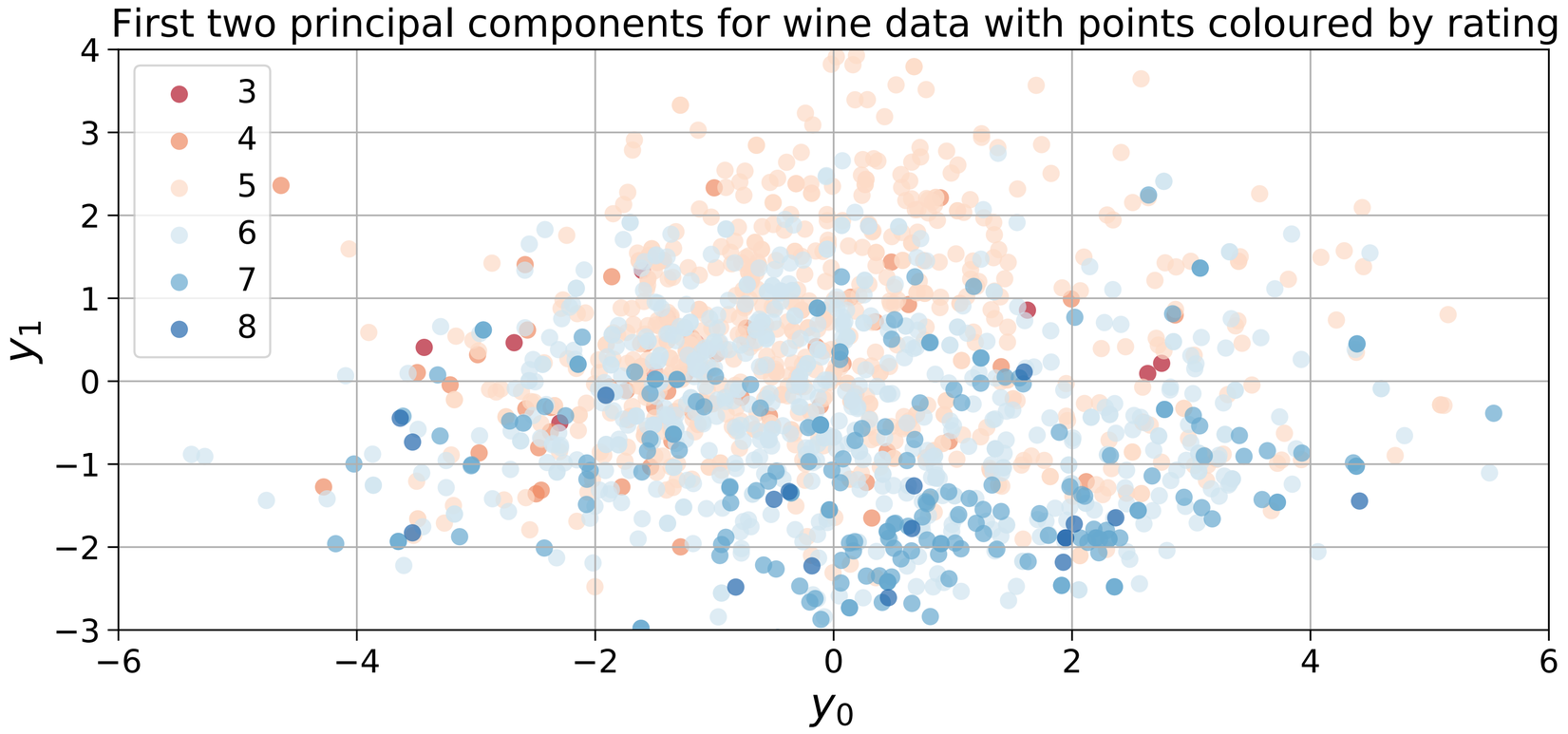
PCA for dimensionality reduction on wine

- Let's standardise our data, and then use PCA to form $\mathbf{W} \in \mathbb{R}^{11\times 11}$
- Now use $\mathbf{Y} = \mathbf{X}\,[\mathbf{w}_0 \quad \mathbf{w}_1]$ to project down to 2D



PCA for dimensionality reduction on wine

- We can see in this space that good wines tend to be near the bottom
- What makes a good wine? A negative y_1 of course!



Good wine recipe - make y_1 negative

- The new dimensions are just linear combinations of the original dimensions

$$y_1 = -0.11x_0 + 0.27x_1 - 0.15x_2 + 0.27x_3 + 0.15x_4 + 0.51x_5 + 0.57x_6 + 0.23x_7 + 0.01x_8 - 0.04x_9 - 0.39x_{10}$$

- In a lot of cases the new dimensions aren't very intuitive
- PCA is best used for exploratory data analysis

Importance of components

- Performing PCA gives us eigenvalue, eigenvector pairs $\{\lambda_d\}_{d=0}^{D-1}, \{\mathbf{w}_d\}_{d=0}^{D-1}$
- The eigenvectors are our principal components
- The eigenvalues are an importance weighting for each component

The first principal component explains $\frac{\lambda_0}{\sum_d \lambda_j}$ % of the variance of the data

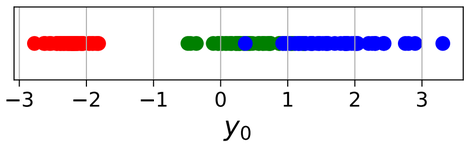
Importance of components

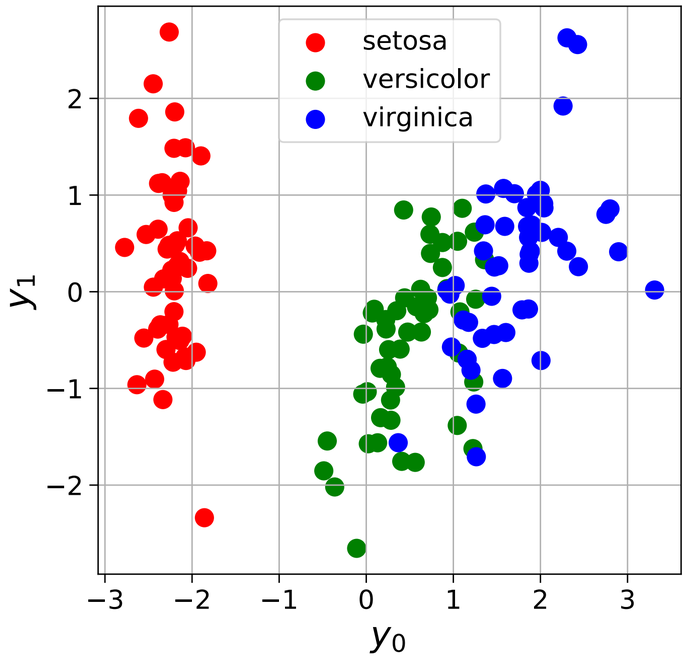
The first principal component explains $\frac{\lambda_0}{\sum_d \lambda_j}$ % of the variance

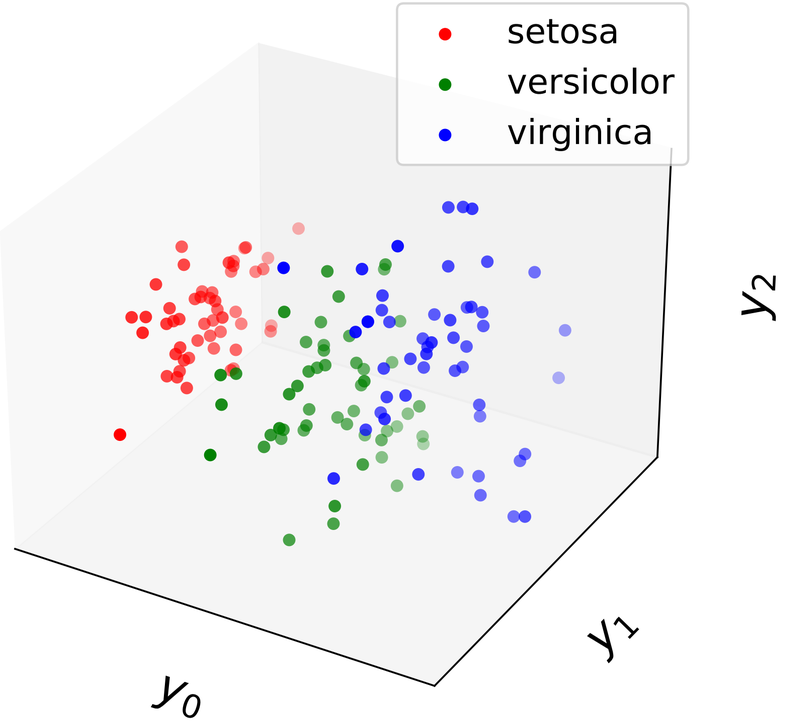
It follows that the first M principal components account for $\frac{\sum_{d=0}^{M-1} \lambda_j}{\sum_d \lambda_j}$ %

Be careful throwing away dimensions if not enough variance is explained

Explaining variance of irises





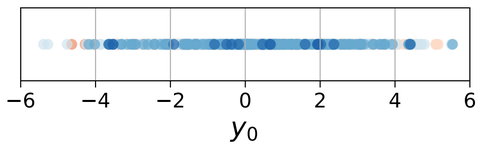


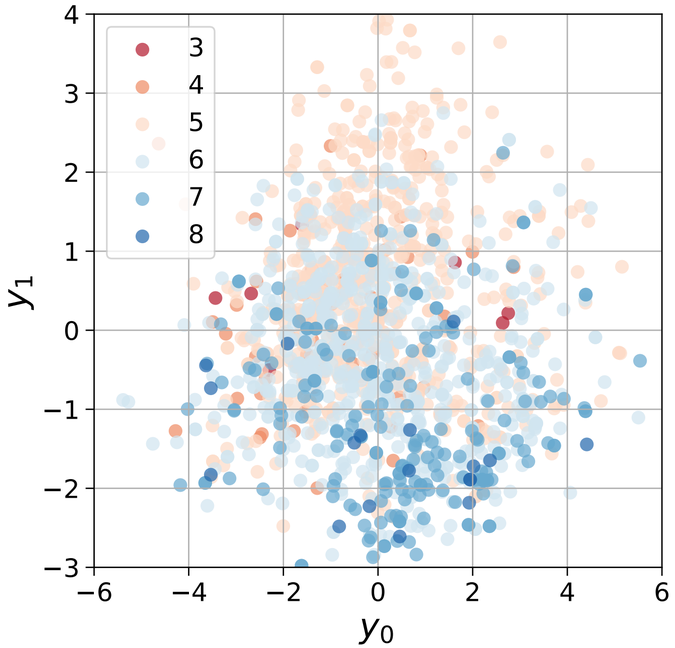
1D: 73%

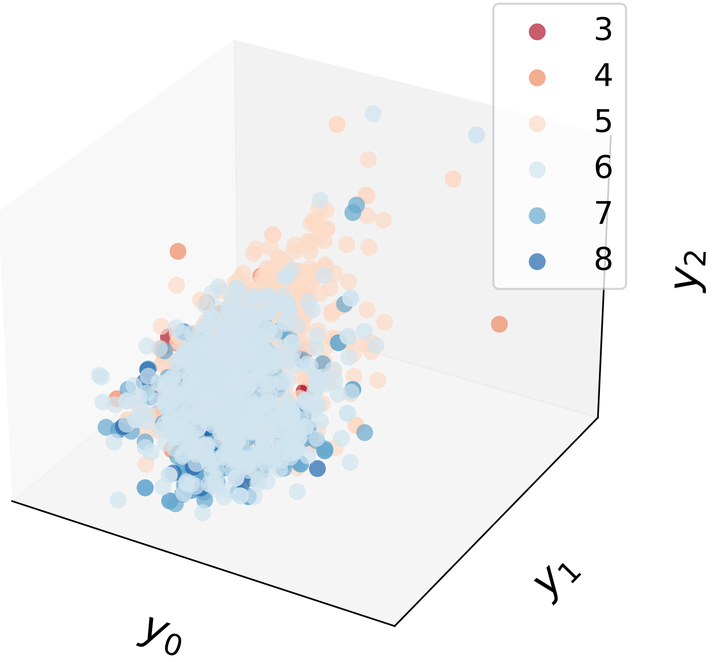
2D: 96%

3D: 99%

Explaining variance of wine







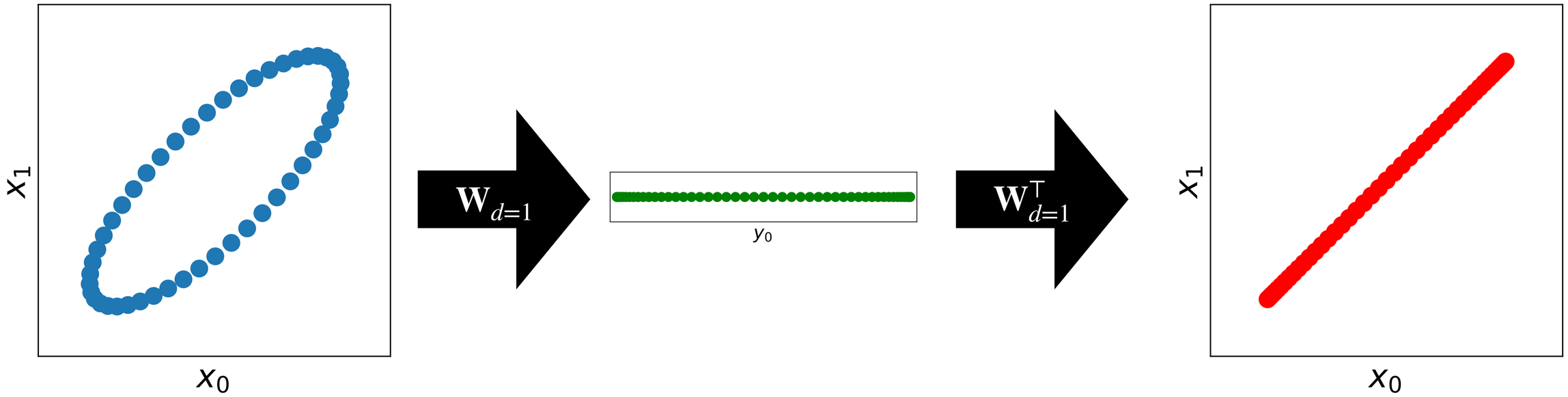
1D: 28%

2D: 45%

3D: 60%

Reconstructing a dataset

- $\mathbf{Y} = \mathbf{X}\mathbf{W}_d$ projects data from D dimensions to d dimensions
- Let's assume we can use $\mathbf{W}_d^\top$ to bring the projected data back up to D dims
- We can then define our **reconstructed dataset** as $\widetilde{\mathbf{X}} = \mathbf{Y}\mathbf{W}_d^\top = \mathbf{X}\mathbf{W}_d\mathbf{W}_d^\top$



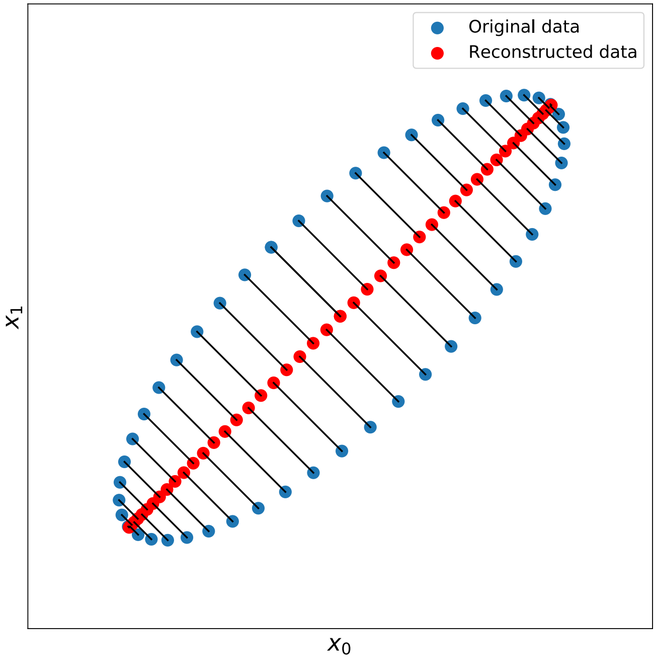
Reconstruction error

- For a good reconstruction, reconstructed points should be near the originals
- i.e. the (average) distance between them should be low
- Our points are $\{\mathbf{x}^{(n)}\}_{n=0}^{N-1}$ with reconstructions $\{\widetilde{\mathbf{x}}^{(n)}\}_{n=0}^{N-1}$

Reconstruction error

$$E_r = \frac{1}{N}\sum_n \|\mathbf{x}^{(n)} - \widetilde{\mathbf{x}}^{(n)}\| = \frac{1}{N}\sum_n \|\mathbf{x}^{(n)} - \mathbf{W}_d\mathbf{W}_d^\top\mathbf{x}^{(n)}\|$$

$\mathbf{X}\mathbf{W}_d\mathbf{W}_d^\top$ for the dataset matrix means $\mathbf{W}_d\mathbf{W}_d^\top\mathbf{x}$ for each column vector data point



Minimising reconstruction error

- We have a reconstruction error that we want to minimise

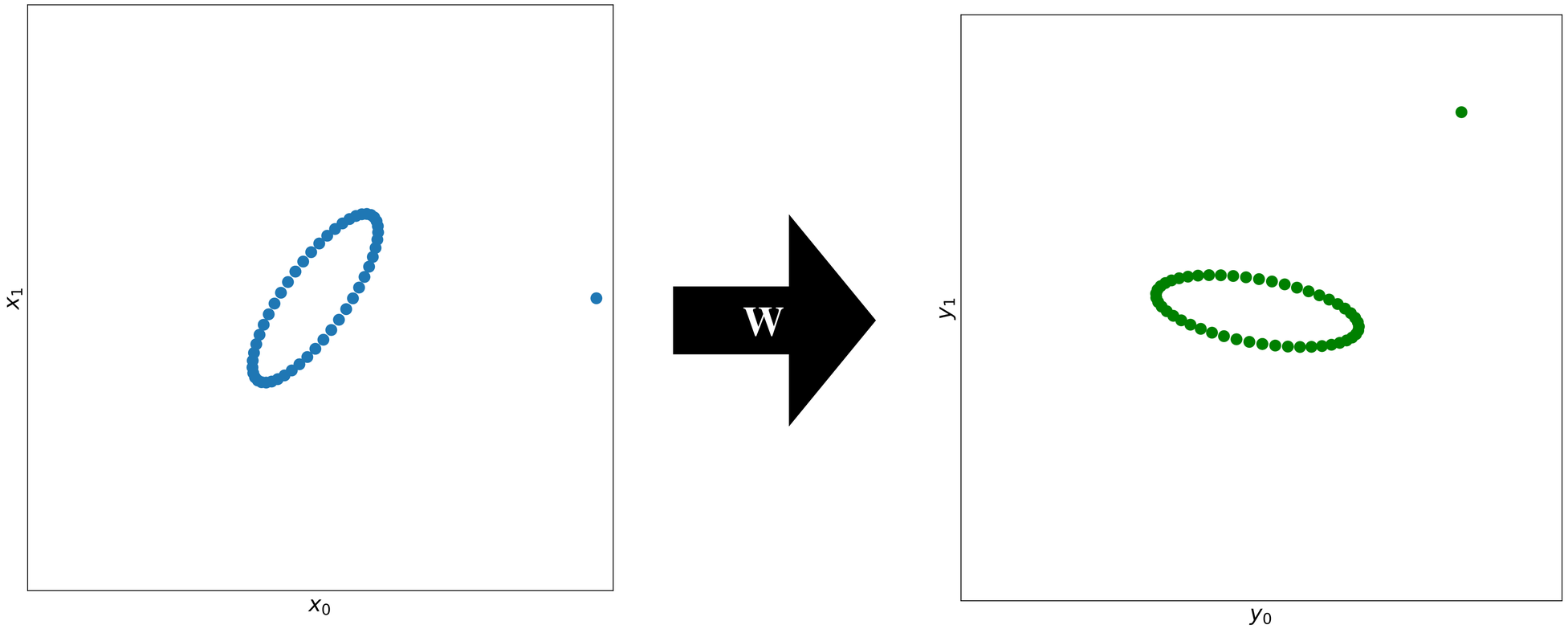
$$E_r = \frac{1}{N}\sum_n \|\mathbf{x}^{(n)} - \widetilde{\mathbf{x}}^{(n)}\| = \frac{1}{N}\sum_n \|\mathbf{x}^{(n)} - \mathbf{W}_d\mathbf{W}_d^\top\mathbf{x}^{(n)}\|$$

- $\mathbf{W}_d$ does minimise this!
- PCA gives you the best possible matrix for making E_r as small as possible
- **Minimising reconstruction error is equivalent to maximising variance**

Proof: http://web4.cs.ucl.ac.uk/staff/D.Barber/textbook/200620.pdf p324

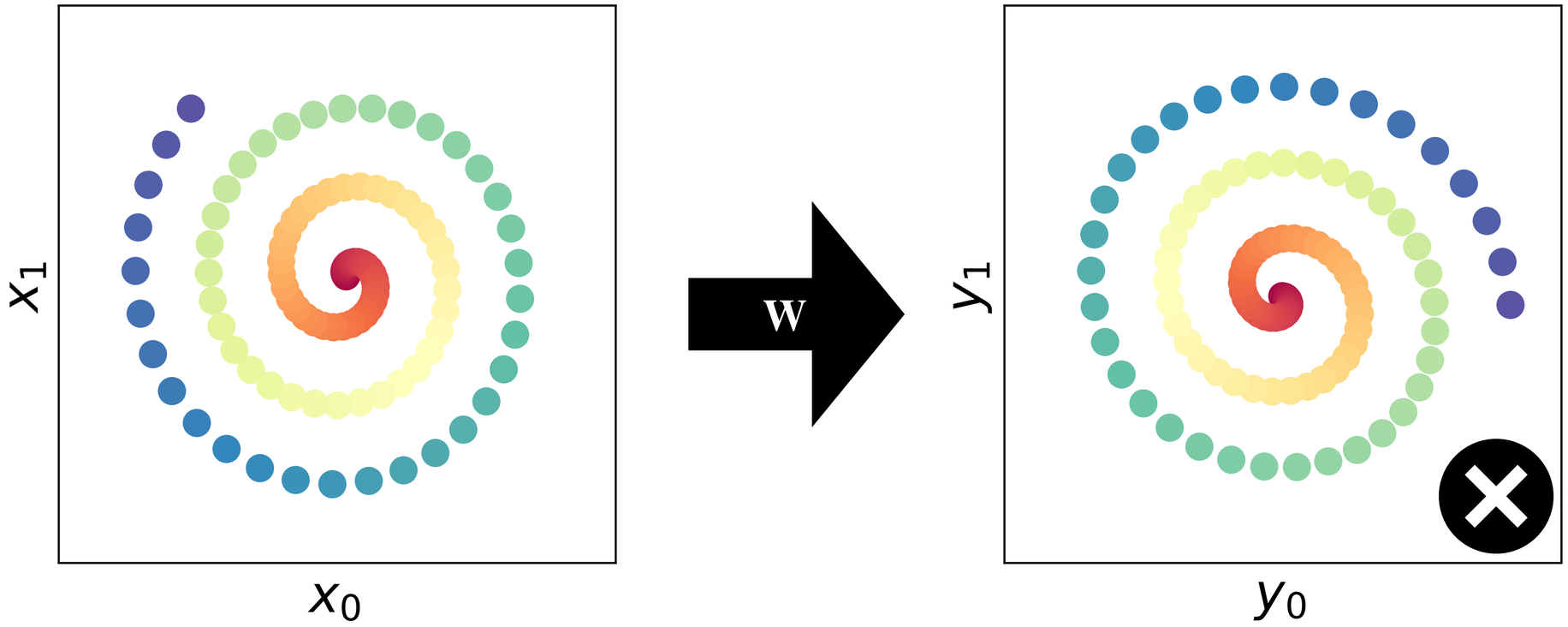
Limitations: PCA is susceptible to outliers

Outliers can change the direction of maximum variance



Limitations: PCA is linear

If the direction of maximum variance isn’t a line, PCA can’t find it



Clustering with K-means

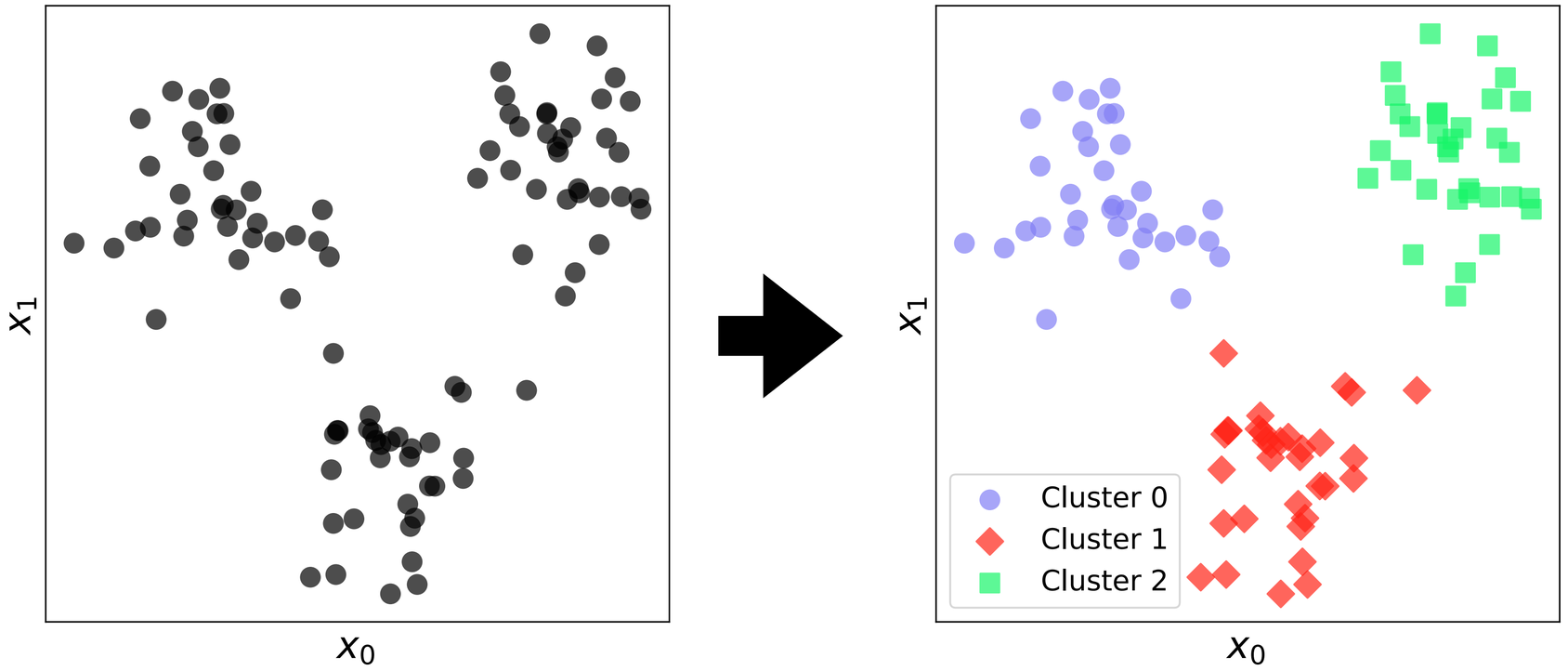
Motivation

You have a dataset that you want to split into groups

- people with low, medium, high income for marketing
- grouping shoppers to recommend products
- identifying personality types for a dating website

K-means

- We can use K-means to automatically split our dataset in groups
- Other clustering algorithms are available!



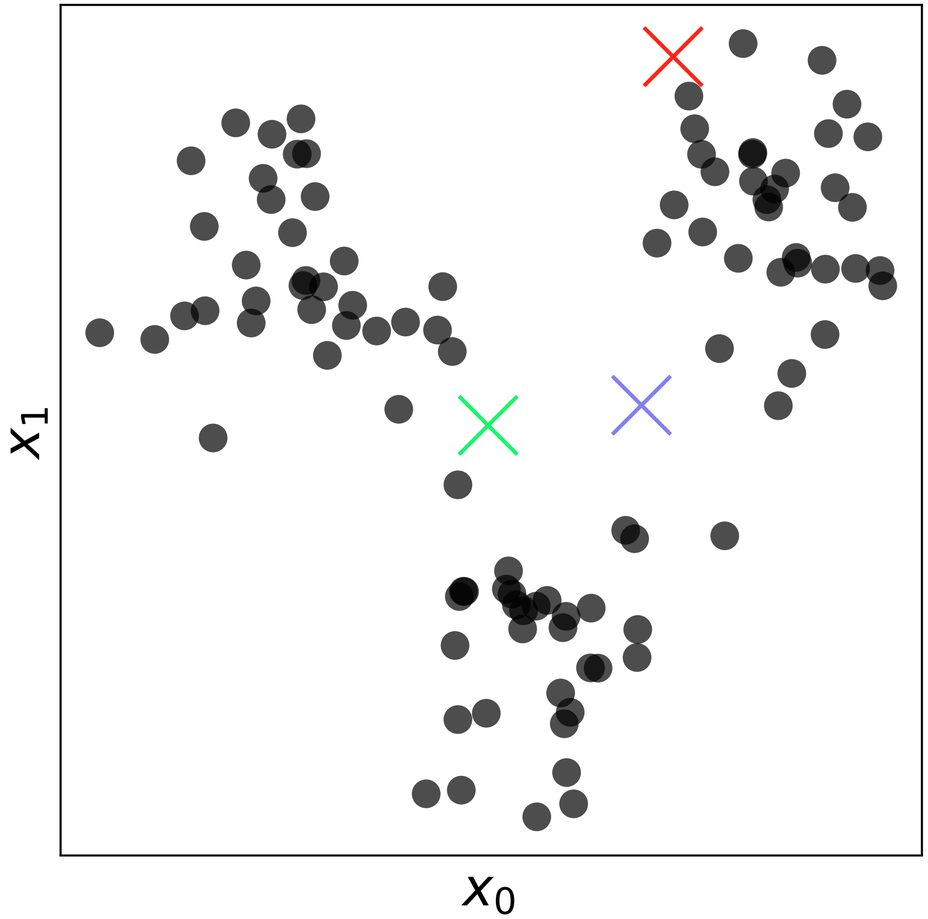
K-means algorithm

- Select the number of clusters K
- Initialise the cluster centres $\{\mathbf{c}_k\}_{k=0}^{K-1}$ at random
- Repeat:
  1. Assign each **standardised** data point to its nearest cluster centre
  2. Update cluster centres as mean of their assigned points
- Until no change

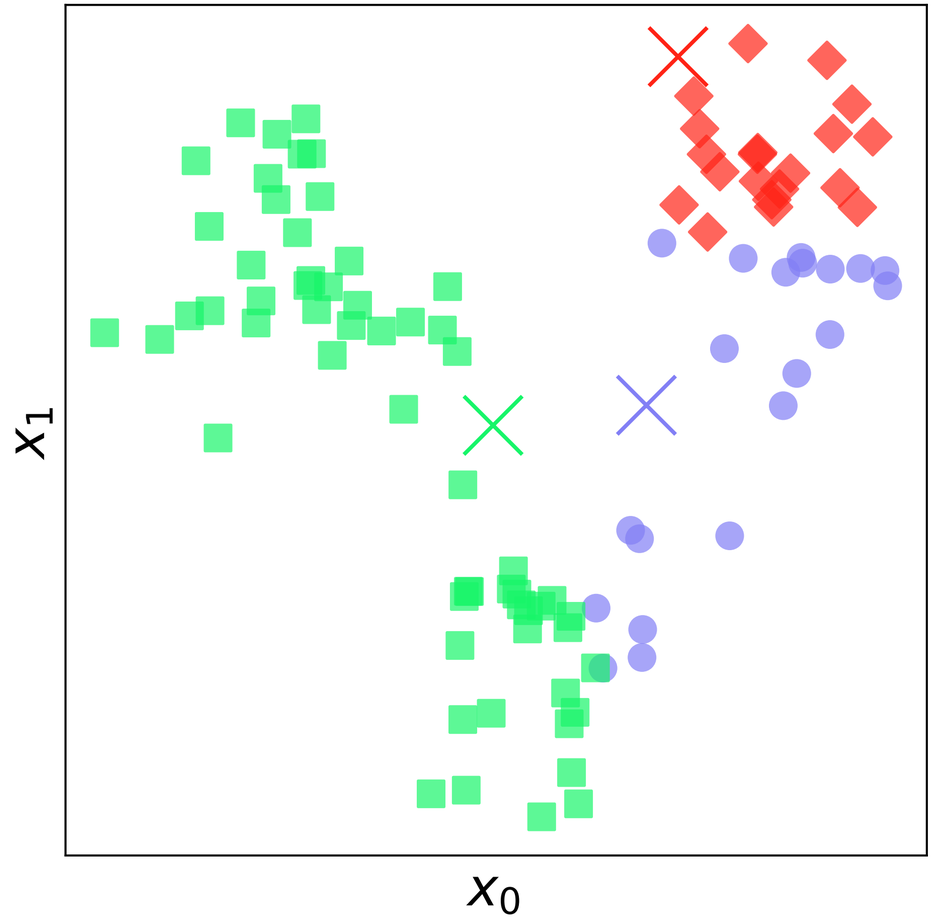
Credit: Andrew Zisserman (for the slide idea, not the algorithm)

K-means walkthrough with $K = 3$

Initialise the cluster centres $\{\mathbf{c}_k\}_{k=0}^{K-1}$ at random

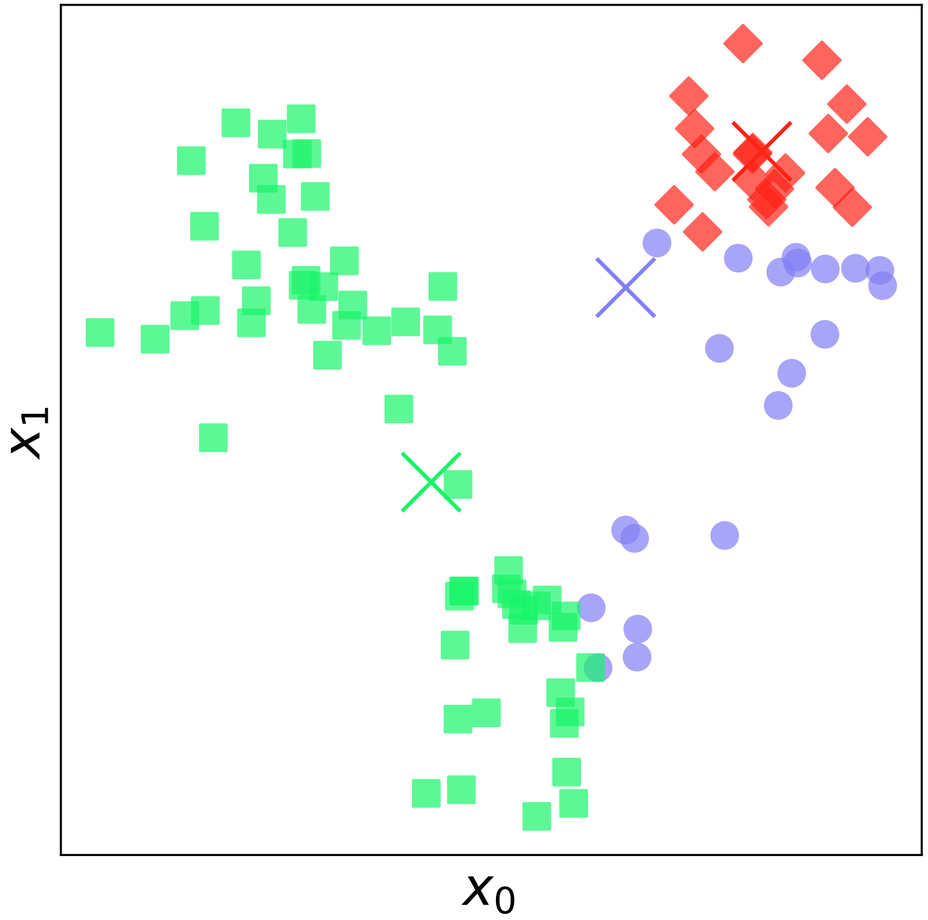


Assign each data point to its nearest cluster centre

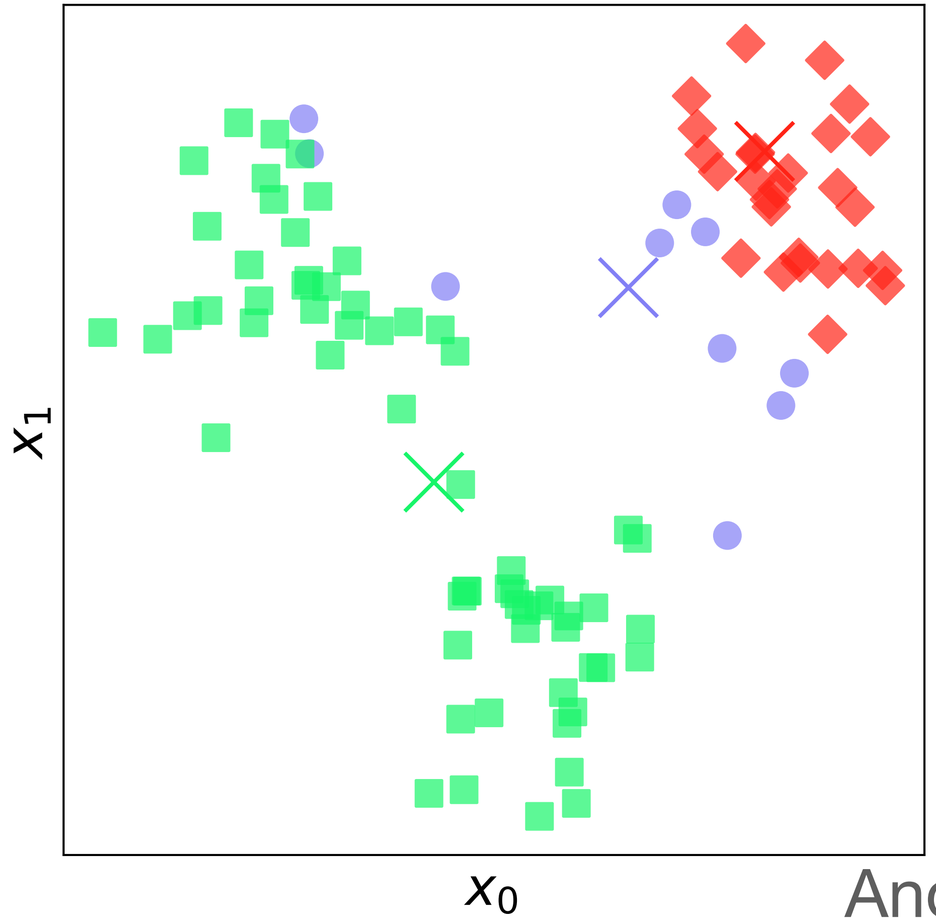


K-means walkthrough with $K = 3$

Update cluster centres as mean of their assigned points



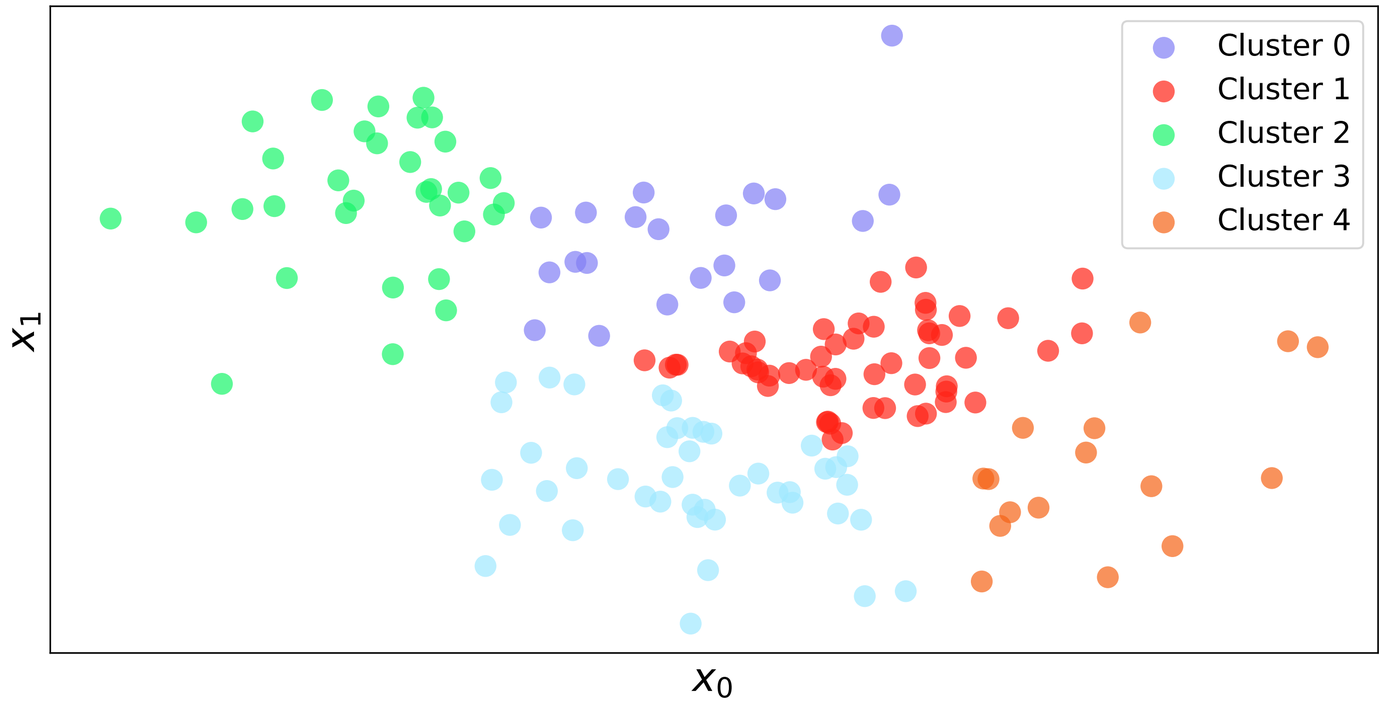
Assign each data point to its nearest cluster centre



And so on!

Warning

- K-means is very sensitive to where the initial cluster centres are placed
- The number of clusters is user defined
- The clusters **might not be meaningful**



This data is just noise!

Summary

- We have revised some linear algebra
- We have learnt how to preprocess data so it can be used for some algorithms
- We have seen how PCA can be used for dimensionality reduction
- We have been introduced to K-means and how it can cluster data