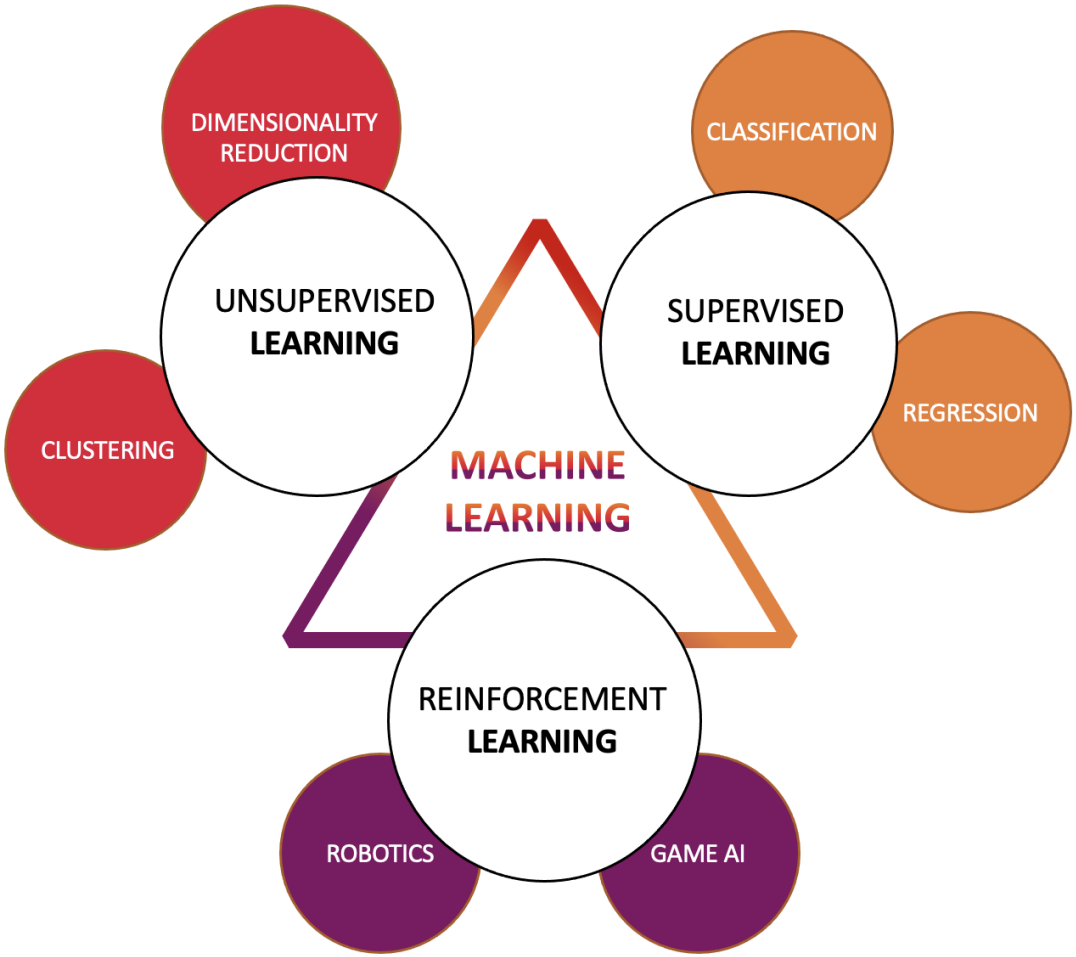
AIML421/COMP309: ML Tools and Techniques
Lecture 4: Regression, Dimensionality Reduction

Ali Knott
School of Engineering and Computer Science, VUW

Recap: Types of machine learning



Today:

1. Regression $\leftarrow$ supervised
2. Dimensionality reduction $\leftarrow$ unsupervised

1. Regression

Classification and regression algorithms

There are two types of supervised ML algorithm.

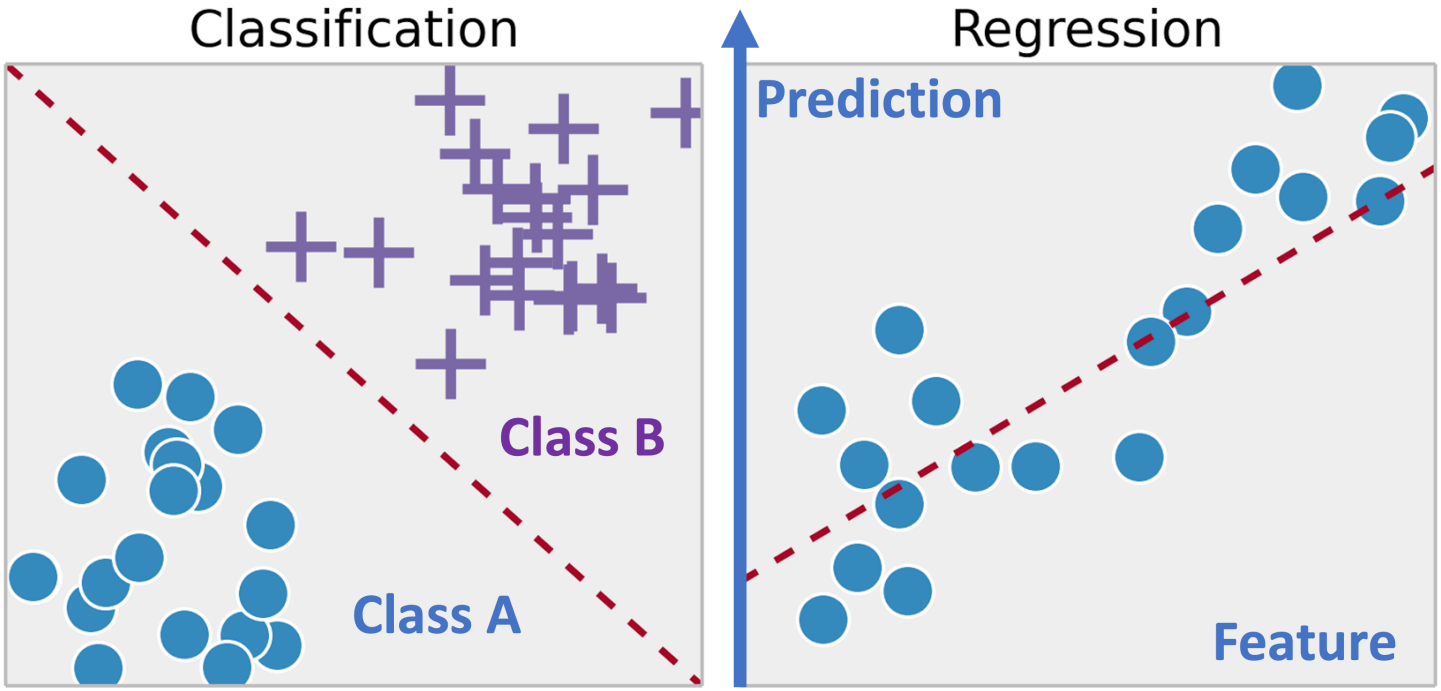
- Classification algorithms have a *discrete* (categorical) output.
  - One class, from a set of discrete alternatives.
- Regression algorithms have a *continuous* (numerical) output.
  - They might be used to predict *price*, *temperature*, and so on.

Both classification and regression algorithms work with the same kinds of *input* features. (Which can be categorical or numerical.)

Classification and regression algorithms

Both kinds of algorithm identify *lines* (or planes) in feature space.

- Classification algorithms find lines that *separate output classes*.
- Regression algorithms find lines that *represent the output value*.
  - In simple cases, the line can be described as a maths function $y = f(x)$.



Examples of regression tasks

House Price Prediction:

- Given features such as location, size, age, number of rooms, predict price

Stock Market Prediction:

- Predicting future stock prices based on historical data

Weather Forecasting:

- Given climate data, predict future temperatures in a given area

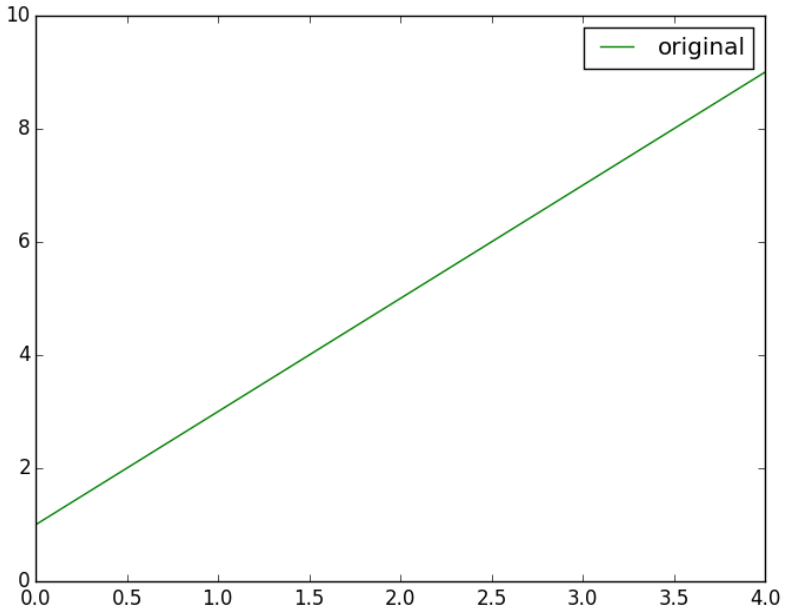
Linear regression

The simplest regression algorithm is linear regression.

- Linear regression assumes the line to be found is a *straight* one.

Here's the simplest scenario:

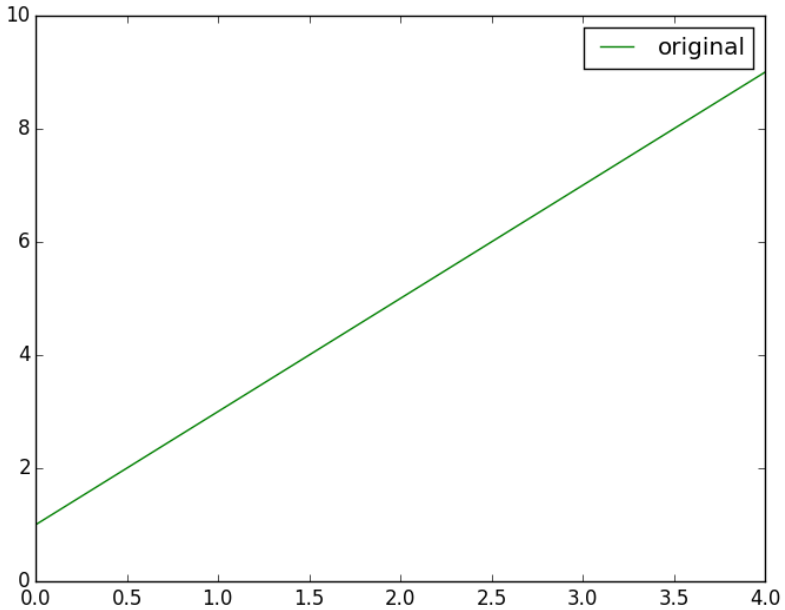
- The line is described by a maths function, on a single continuous input feature x. $y = f(x)$



Linear regression

This example straight-line function is $y = 1 + 2x$.

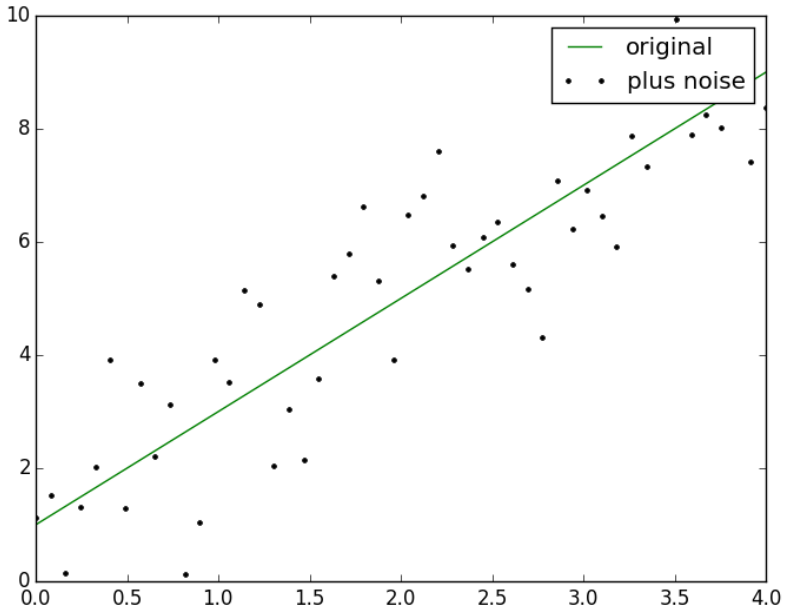
- The 1 term is a constant, that gives the output for $x = 0$.
- The 2 term is the slope of the line.
  - 'If you increase x by 1, you increase y by 2.'
- Linear regression finds the constant and slope that *best models* the training data.



Linear regression

Let's create some datapoints around our line.

- Regression has to find the line that *best 'fits'* those datapoints.
- It does that by searching a *space* of possible constants and slopes.
- It's looking for the line that *minimises the sum of errors* for the datapoints.



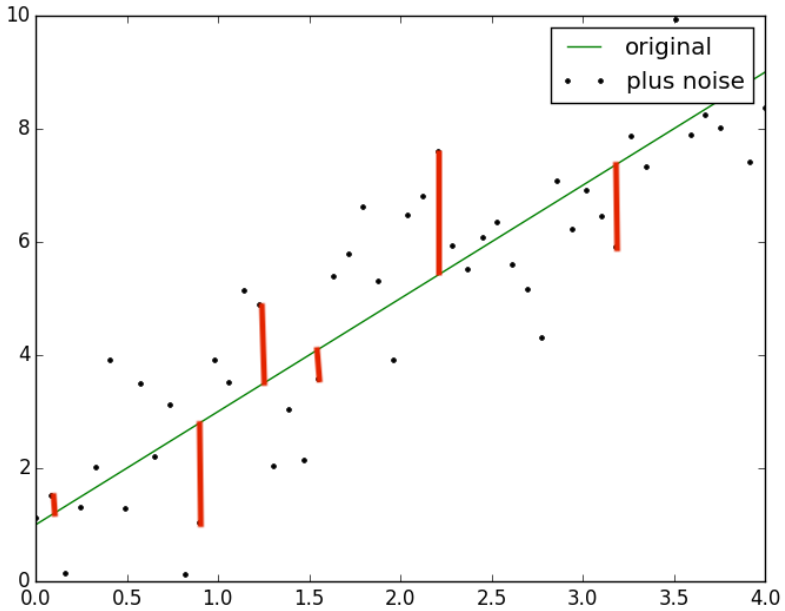
Linear regression

For a line $f(x)$, the error for a *single datapoint* (x, y)... is $(y - f(x))^2$.

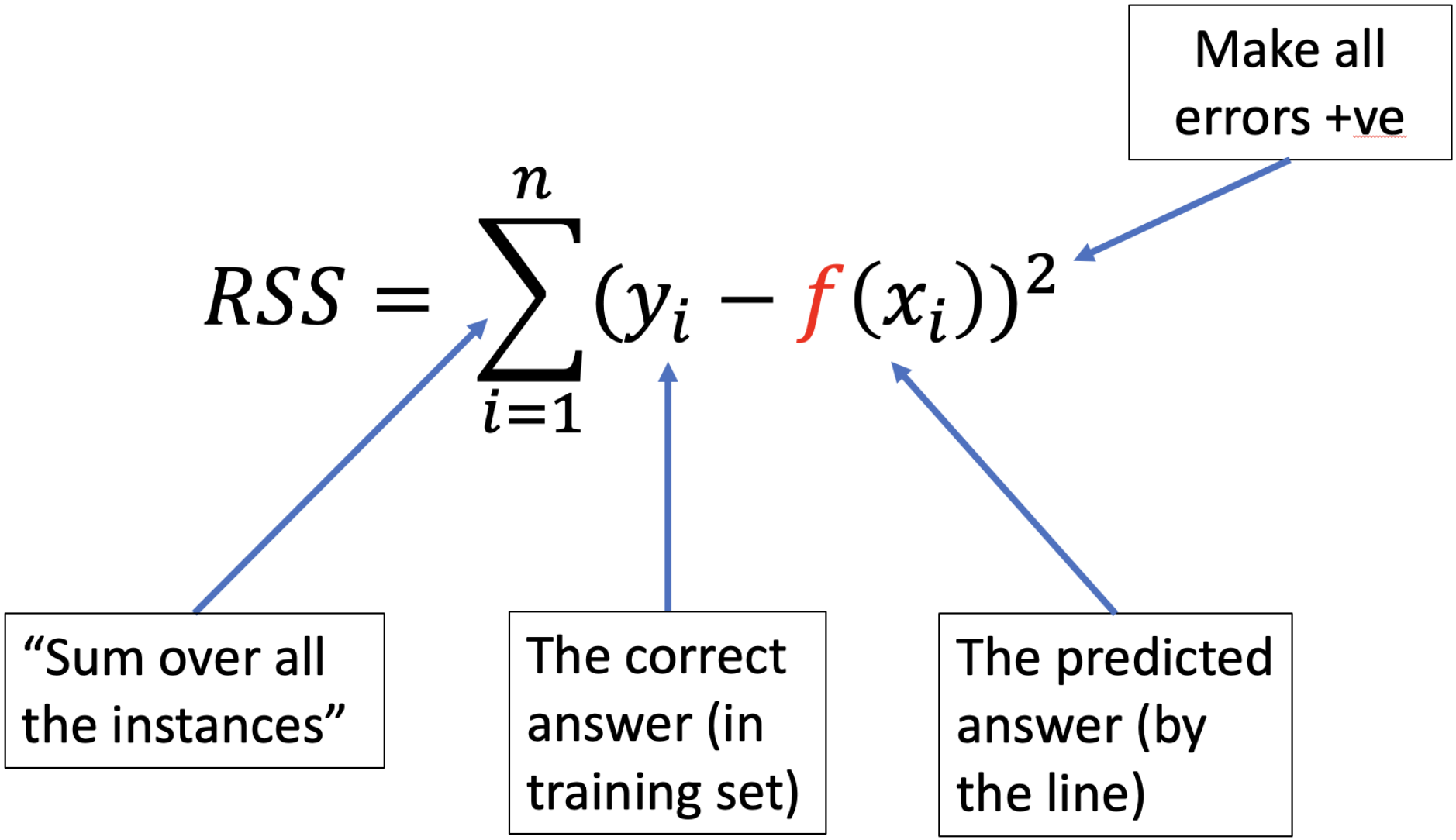
- That's called the residual error. (We square it so it's always +ve.)

For *all n* datapoints in the training set, the error is given by a *sum*:

$$\sum_{i=1}^{n} (y_i - f(x_i))^2 \leftarrow \text{the 'Residual Sum of Squares' (RSS)}$$



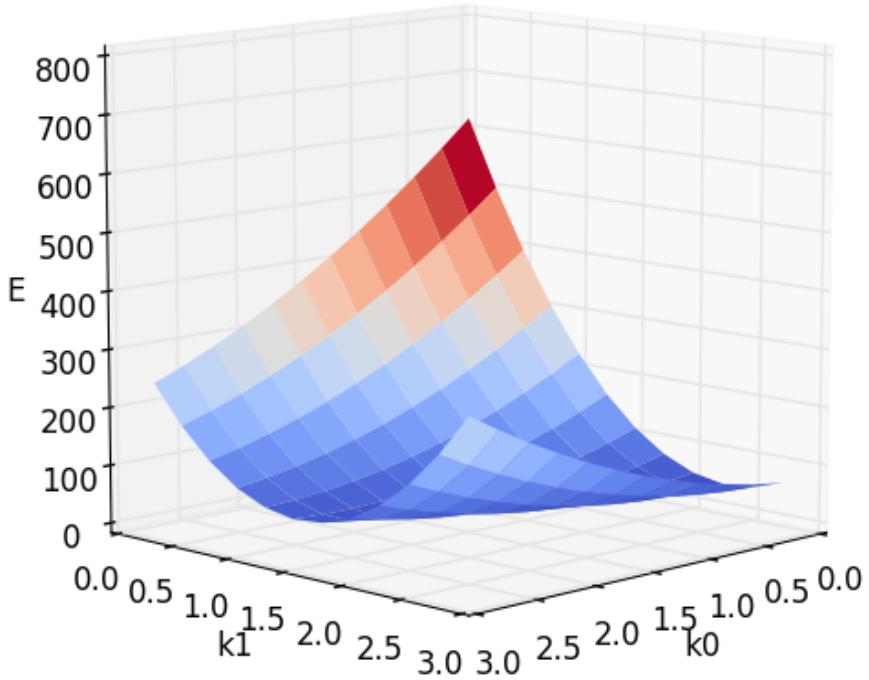
The Residual Sum of Squares formula



How do you find the line that *minimises* the RSS?

Remember the line is defined by its constant and its slope:
$y = 1 + 2xy = k_0 + k_1 x$.

- Regression works by defining an error landscape (recall Week 2).
  - This is a *function*, showing shows the RSS error for *each possible combination* of constant and slope.
- The regression equation finds the *minimum* of that function.



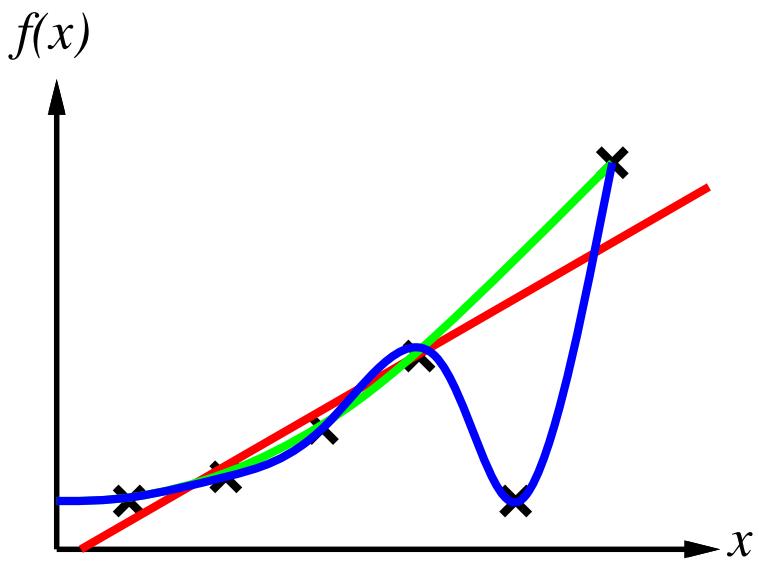
Regression can also fit *curved* lines to a dataset.

A straight line is a function of x.

- To get a curved line, we can use a function including x^2...
- For a curvier line, we can add x^3...

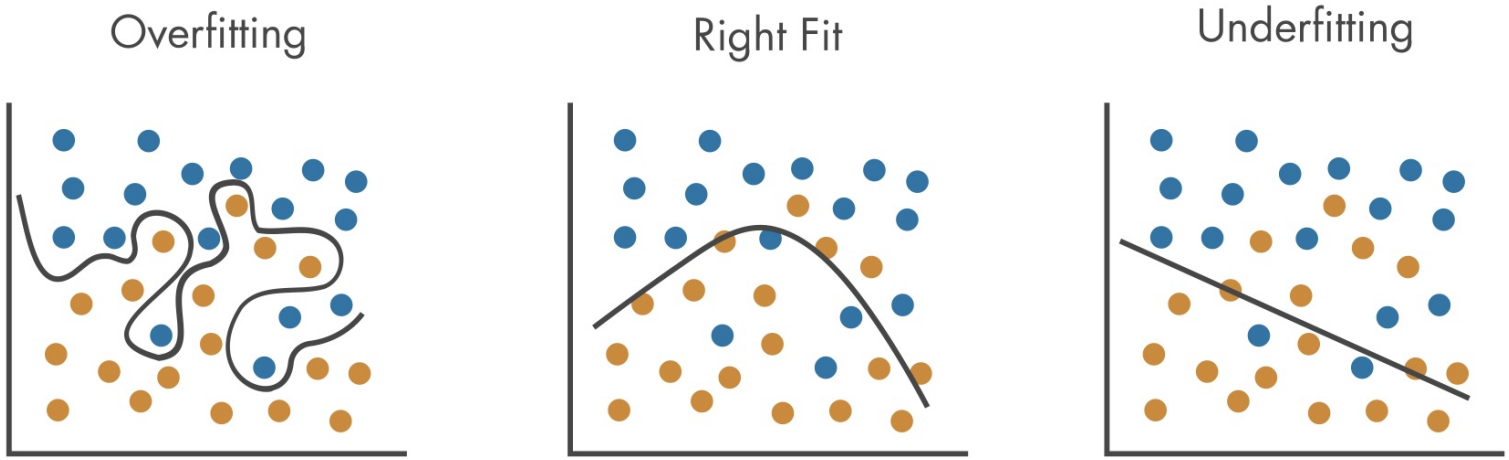
Polynomial regression works with *any polynomial fn*, to find the best fit:

$$y = k_0 + k_1 x + k_2 x^2 + k_3 x^3 + \ldots + k_n x^n$$

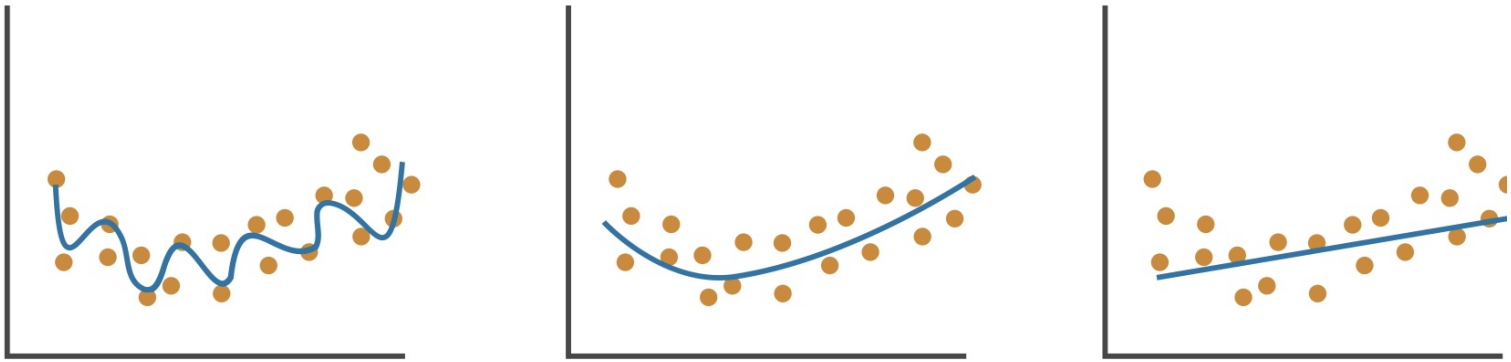


Overfitting & underfitting happens with regression too!

Remember what these look like for classification:



Here's what they look like for regression:



Regression also works with multiple input features...

Until now our function has been $y = f(x)$

- But we can also have $y = f(x_1, x_2, \ldots, x_n)$.
- Now the training points lie in a *n+1-dimensional space*...
  And regression fits a *plane* to these points.

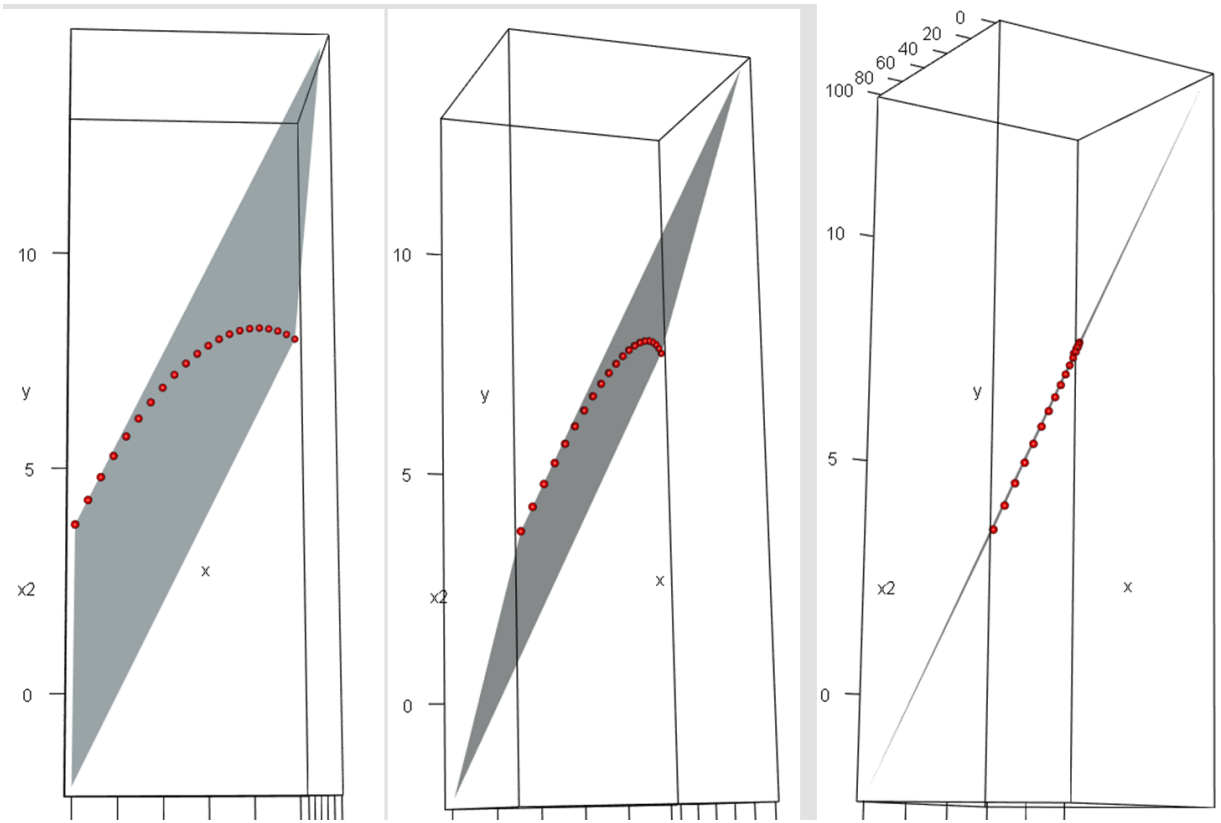
Multiple linear regression finds a *linear* relationship with each input feature x_i.

$$y = k_0 + k_1 x_1 + k_2 x_2 + \ldots + k_n x_n$$

Regression also works with multiple input features...

There's no need to look for polynomial relationships if you have multiple dimensions!

- Multiple regression just treats x, x_2, x_3 etc as separate variables.
- The polynomial curves end up falling into a flat plane...



Collinearity

In multiple linear regression, there are sometimes *relationships between input features*.

- The polynomial function 'living in' the flat plane on the previous slide is an example.
- In general, there can be *correlations* between input features. (Called collinearity.)

Say our dataset has two inputs, x_1 and x_2—but x_2 is just a *copy* of x_1!

If we're trying to learn the function $y = k_0 + k_1 x_1 + k_2 x_2 \ldots$

- We could let x_1 do all the work (setting k_2 to 0).
- Or we could let x_2 do all the work (setting k_1 to 0).
- Any *sum* of k_1 and k_2 that gives the actual slope will work!

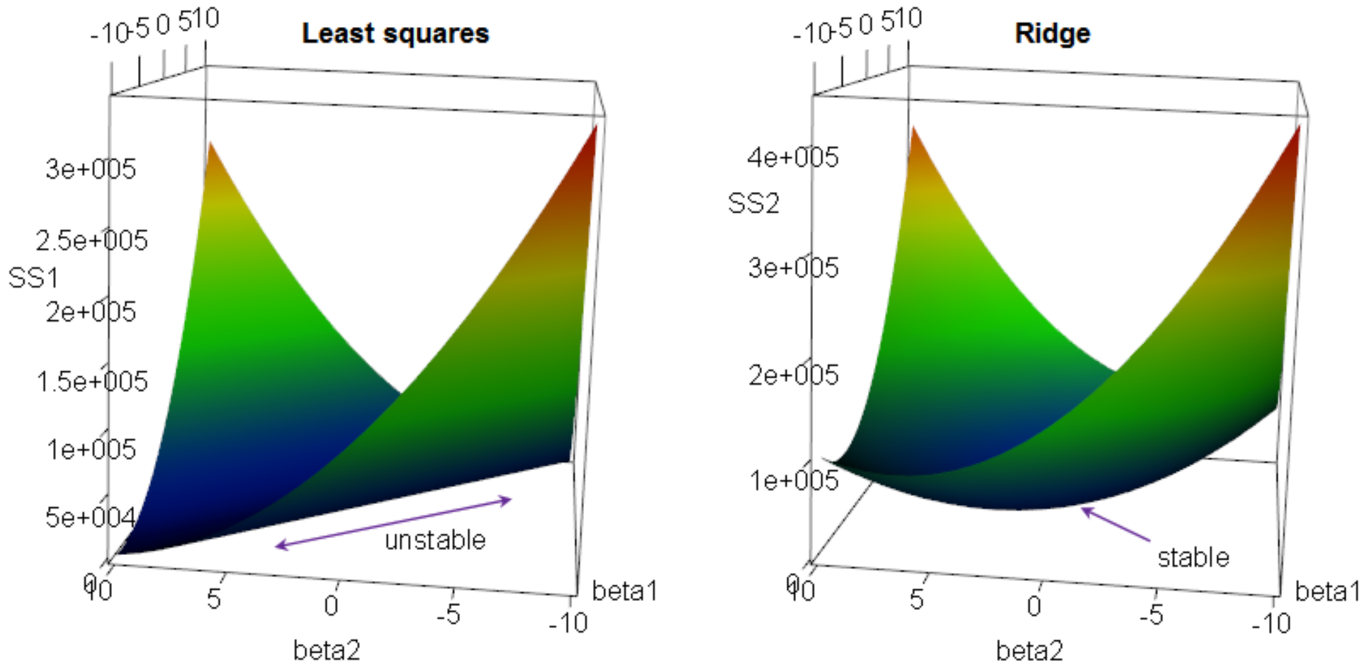
Ridge regression

If we have two collinear input variables, the error landscape contains a ridge of minimum values.

- Regression in this context is *unstable*: won't settle in a clear place!

We can fix this problem with ridge regression.

- Ridge regression encourages *small regression coefficients* k_i.
- This creates a single minimum in the error landscape.



Ridge regression and regularisation

Recall from Week 2: in neural networks, regularisation is a device that encourages a network's weights w_i to *stay small*.

- The loss function we minimise is a *sum* of 'training error' and 'combined weight size'.
- The weights could be minimised with L1 (Lasso), or L2 (Ridge).

These same methods can be used in regression.

- In neural networks, the parameters are weights (w_i)...
- In regression, the parameters are regression coefficients (k_i).

Regression addresses model *complexity*, but also model *stability*.

Measures for evaluating regression models

We can evaluate *classification* models with several measures:

- Accuracy and Error rate
- Confusion matrices (for binary classifiers)

For regression models, there are also several measures.

- Sum squared error (SSE)
- Mean squared error (MSE)
- Mean absolute error (MAE)
- Root mean squared error (RMSE)

They are all measures of *error* (small is good)—because it's hard to achieve perfect performance with continuous outputs.

Sum Squared Error

Sum Sqared Error is identical to the error measure regression optimises.

$$SSE = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

The difference is that this error is computed on items *from the test set*. (Or validation set.)

Mean Squared Error and Mean Absolute Error

Here's SSE again:

$$SSE = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

Mean Squared Error (MSE) just divides SSE by the number of examples n.

- $MSE = SSE/n$
- This gives an *average*, that's independent of dataset size.

Mean Absolute Error (MAE) doesn't *square* the errors.

- $MAE = \frac{(\sum_{i=1}^{n} |y_{true} - y_{predicted}|)}{n}$
- More 'interpretable': preserves the units used in the task.

Root Mean Squared Error

Another way of restoring the units from the task is just to take the square root of the Mean Squared Error.

That's the Root Mean Squared Error (RMSE).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_{true} - y_{predicted})^2}{n}}$$

This one pays more attention to large errors.

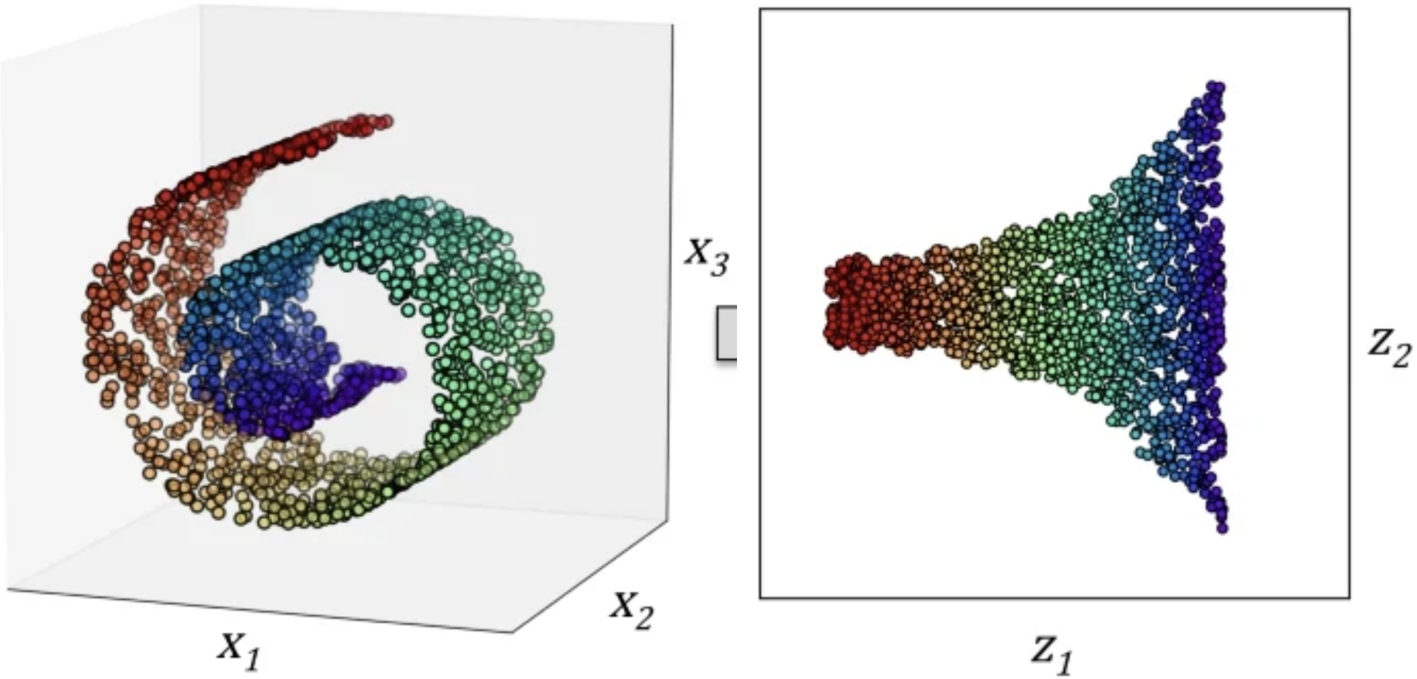
2. Unsupervised learning methods (part 1)

. . . Today, we'll look at some methods for dimensionality reduction.

Dimensionality reduction methods aim to simplify a dataset by re-representing it in a *smaller feature space*.

To illustrate dimensionality reduction...

Say you had some datapoints in a 3D space, that happen to look like this:



You can re-represent this in two dimensions...

- This *simplifies*, while capturing lots of what's important.

What we'll look at

I'll describe three dimensionality reduction methods:

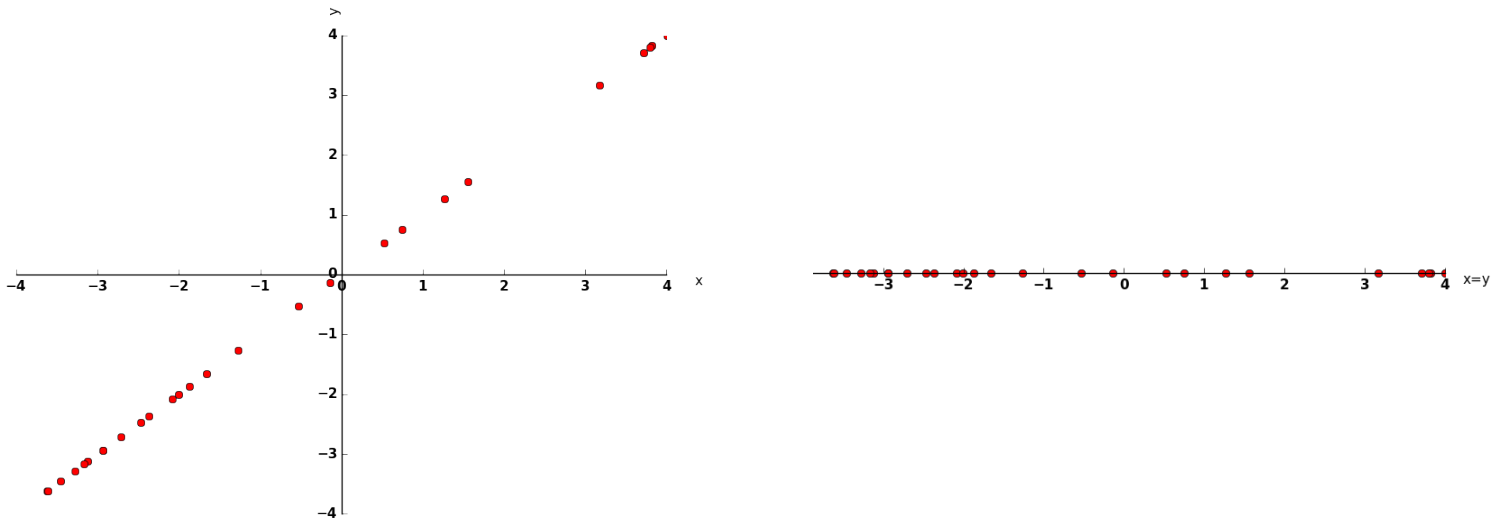
- Principal component analysis (PCA)
- Independent component analysis (ICA)
- Autoencoders.

2.1. Principal component analysis (PCA)

Principal component analysis (PCA) is a technique which finds new axes for representing a set of input data points.

Consider a two-dimensional dataset, where there are strong *correlations* between the two dimensions.

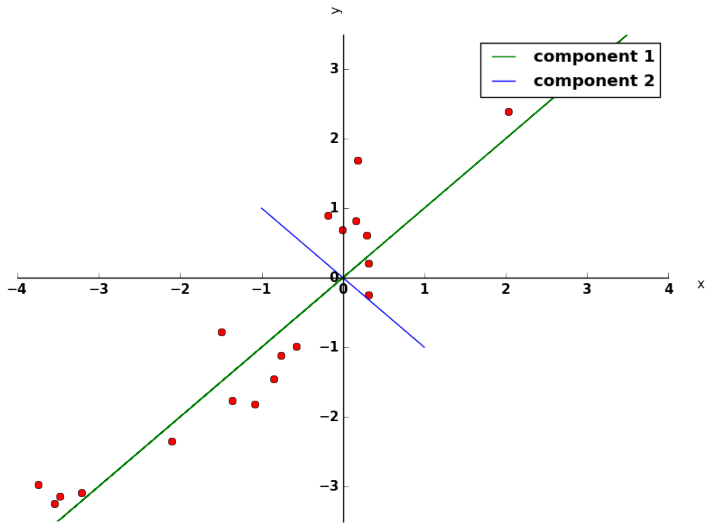
This dataset really only has one dimension! We could re-represent it with a 1D graph, as on the right.



Principal component analysis (PCA)

Now say there's some other dimension of variation in our data, that's orthogonal to the main correlation.

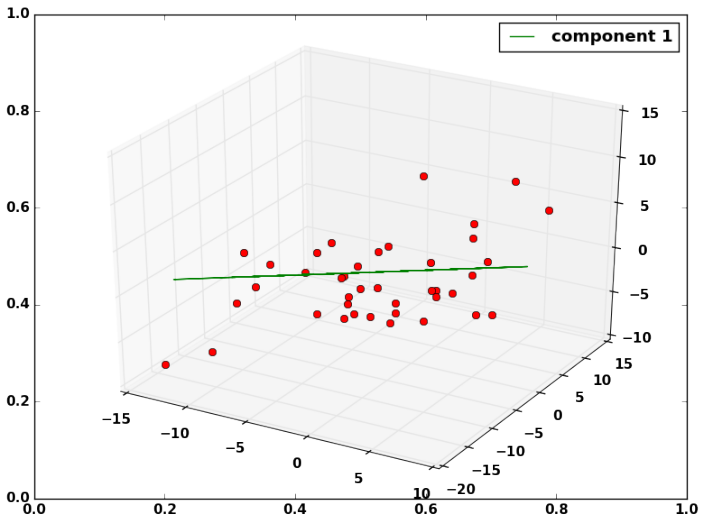
It's still useful to re-represent the data, using axes that explicitly identify the *components* of the variation.

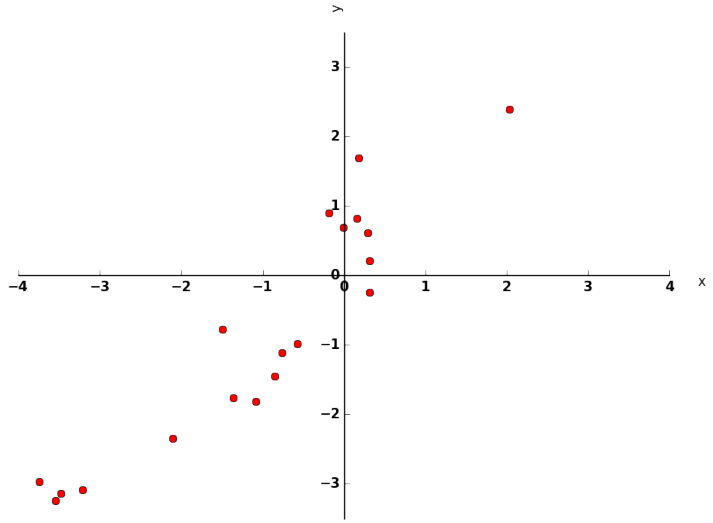


That's what principal component analysis is.

Principal component analysis

If the raw data is 3-dimensional, the main dimension of variation will be a *3d line*.





After this line is found, imagine looking along it: you'll see the data in a new 2D space, whose 2 dimensions are *orthogonal* to the line.

In this space, you can find the *next* dimension of variation.

Principal component analysis

For a dataset with N dimensions, PCA creates a new representation in dimensions $D_1 \ldots D_n$, where

- D_1 is the main component of variation in the data
- D_2 is the main component of variation *after* D_1 *is removed...*
- D_3 is the main component of variation *after* D_2 *is removed...*

In practice, the first couple of components often account for nearly all the variance.

- So you can *approximate* by ignoring the smallest components. (That's how PCA is a dimensionality reduction technique.)

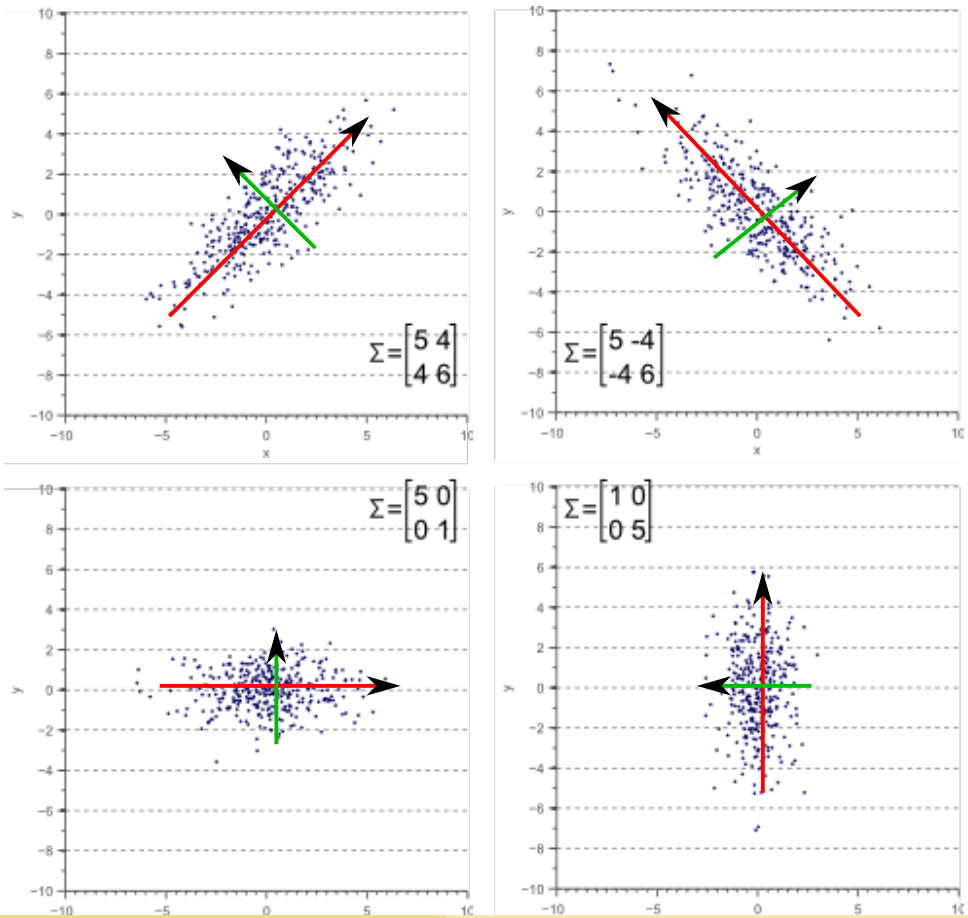
The maths of PCA

To do PCA we first compute the covariance matrix of the input data.

The covariance matrix can be thought of as defining the *transformation* that best maps a cloud of Gaussian noise (in N dimensions) onto the input data.

Then we compute the eigenvectors and associated eigenvalues of this matrix.

The eigenvectors of a transformation are the vectors whose direction is invariant under the transformation.

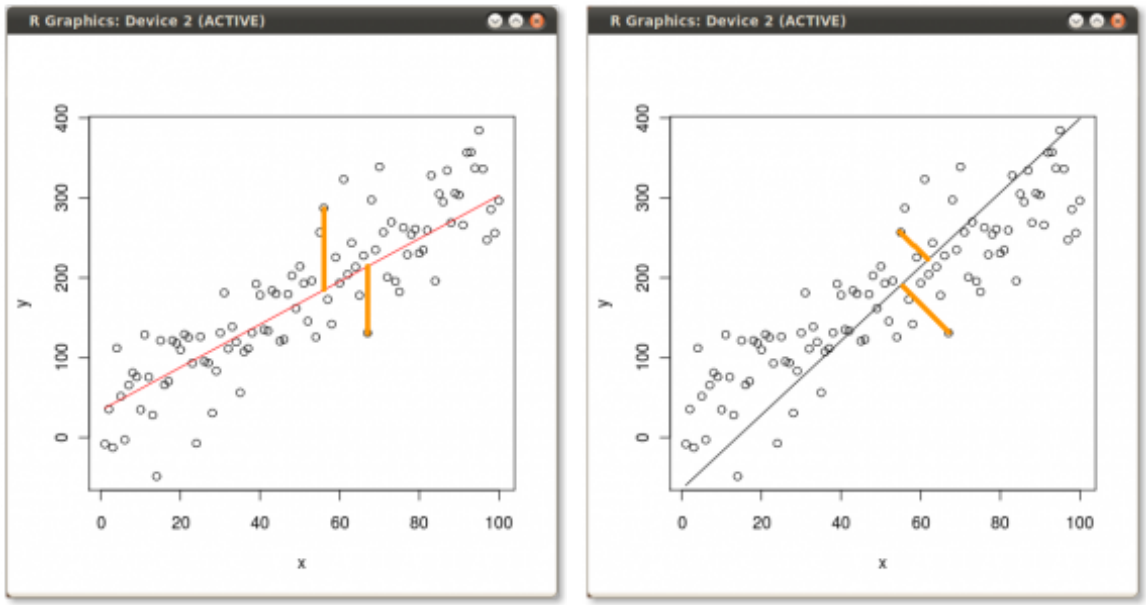


PCA and regression

PCA is a bit like doing several linear regressions, one by one.

The main difference is that in regression, we minimise the sum squared error in one dimension only...

While in PCA, the lines we minimise the sum squared error in all dimensions.

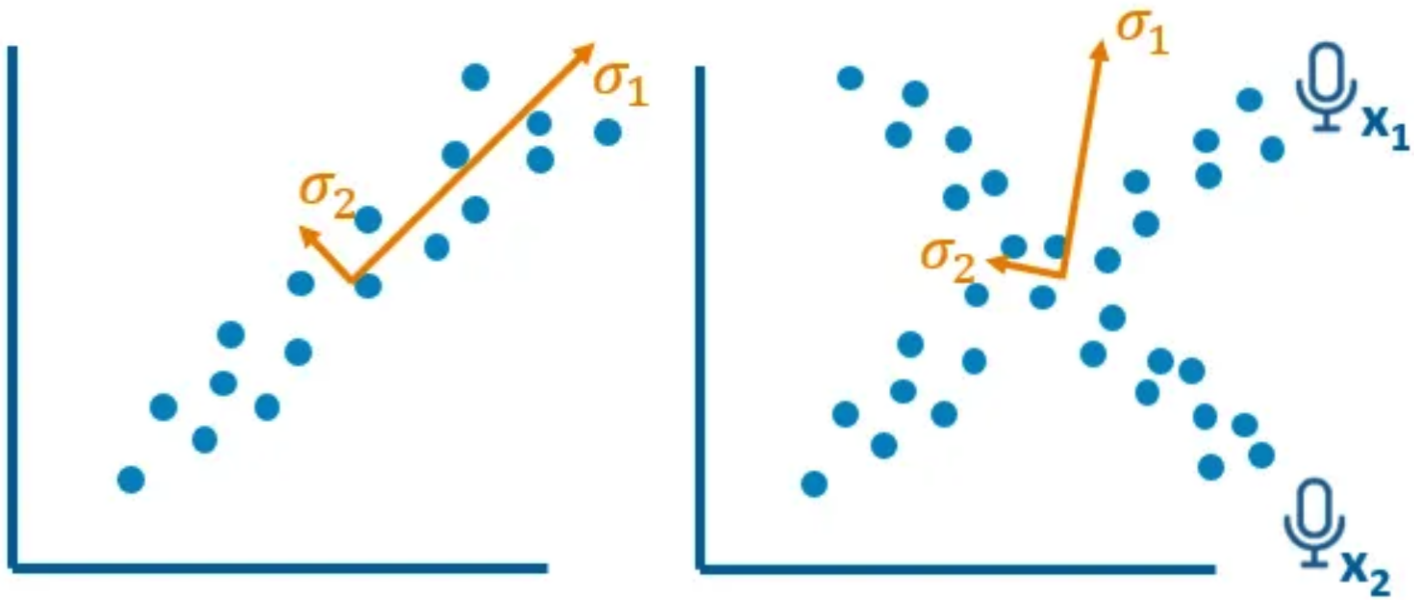


2.2. Independent component analysis (ICA)

Sometimes, there's *more than one thing going on* in our data.

- E.g. 2 people talking, 2 instruments playing, 2 sine waves...
- Independent component analysis (ICA) is a way to identify the different things.

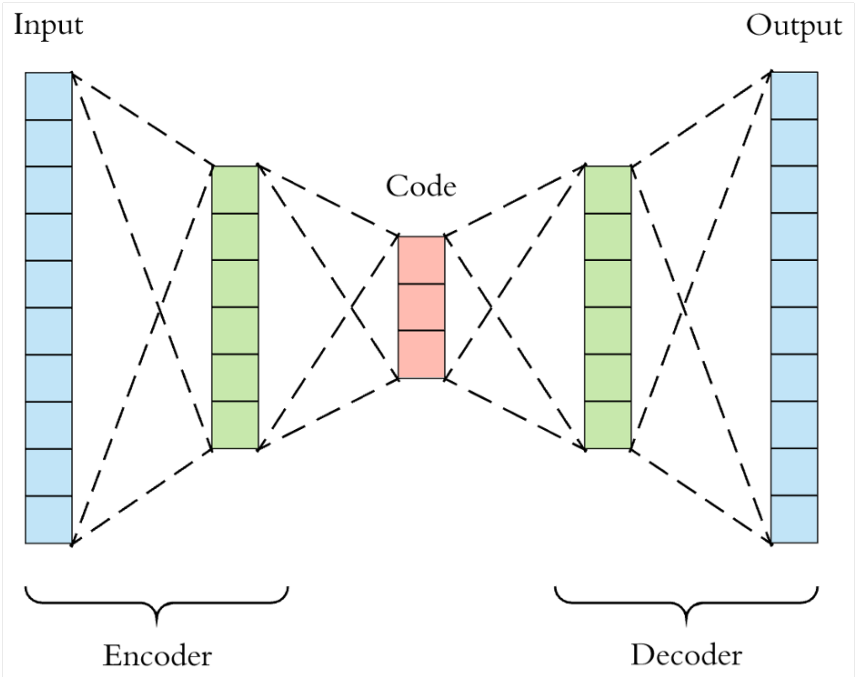
PCA is good when there's one main component...
ICA is good if there's *more than one*.



2.3. Autoencoders

A final dimensionality reduction technique uses a neural network: a multi-layer perceptron (MLP).

- An autoencoder is a MLP which learns to map each input *back onto itself*.
- It can create its own training data. . . it learns by self-supervision.



2.3. Autoencoders

An autoencoder has a bottleneck layer in the middle, which is *smaller* than its input/output layers.

- If the network learns well, this middle layer holds a *compressed representation* of the input/outputs.
- The input/output layers encode items in full feature space. . .
  The bottleneck layer represents items in a *smaller* feature space.