Class 5: Unsupervised learning and dimension reduction

Andrew Parnell
`andrew.parnell@mu.ie`

PRESS RECORD

https://andrewcparnell.github.io/intermediate_ML

Learning outcomes

Our first foray into **unsupervised learning**

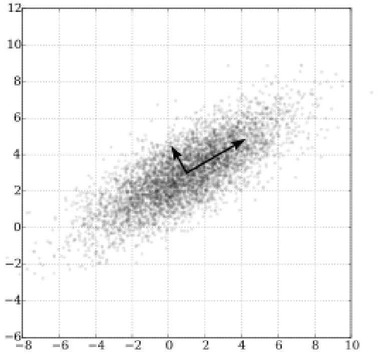
- Reminder on some basic methods of dimension reduction
- An introduction to autoencoders and how they are used
- Some examples to show how they are used

Set-up

- Suppose we have a high dimensional data set that we can write as a matrix X with n rows and p columns
- For example you might have a set of songs in the rows and details about them in the columns (number of streams, length, chord structure, etc)
- Plotting all of these data gets really hard as it's so high dimensional
- Can we extract some kind of meaning from the variability in these data? Are some columns linked? Are some of the row very different from the others
- Idea: approximate the matrix X with $\tilde{X}$ which has far fewer columns but contains most of the information in X

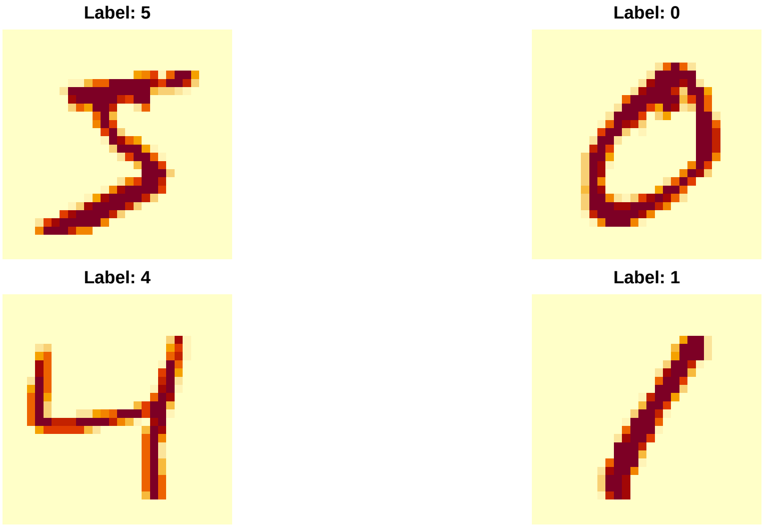
Principal Components Analysis

- The oldest, simplest and most useful version of dimension reduction
- The method plots the data in high-dimensional space (mathematically) and then re-aligns the axes to match the spread of the data, so that the first axis has most of the variability, the second axes the second most, etc

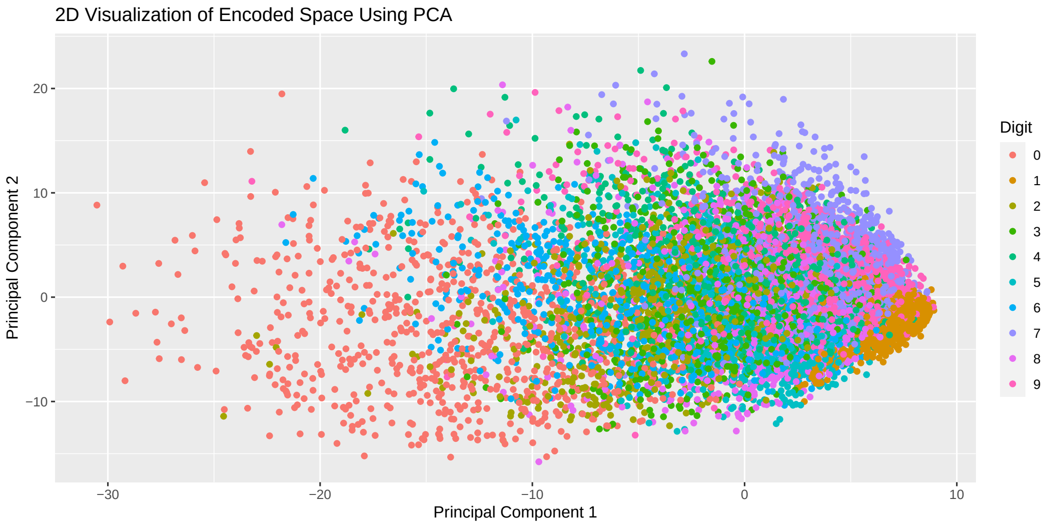


From https://en.wikipedia.org/wiki/Principal_component_analysis

Example: the mnist data



Running PCA with 2 dimensions



Some notes on the previous slide

- The PCA never saw the digit number, it only saw the images (I used the digit numbers just for the final plot)
- I chose (and it's very common to choose) 2 principal components to plot, but you could choose many more
- The algorithm automatically orders the PCs and you can see how much variation each one captured in the data

```r
summary(pca)$importance[,1:5] * 100
```

```
##                             PC1      PC2      PC3      PC4     PC5
## Standard deviation     636.2889 540.7337 517.7845 454.7757 425.138
## Proportion of Variance   6.0790   4.3900   4.0260   3.1050   2.714
## Cumulative Proportion    6.0790  10.4690  14.4950  17.6000  20.314
```

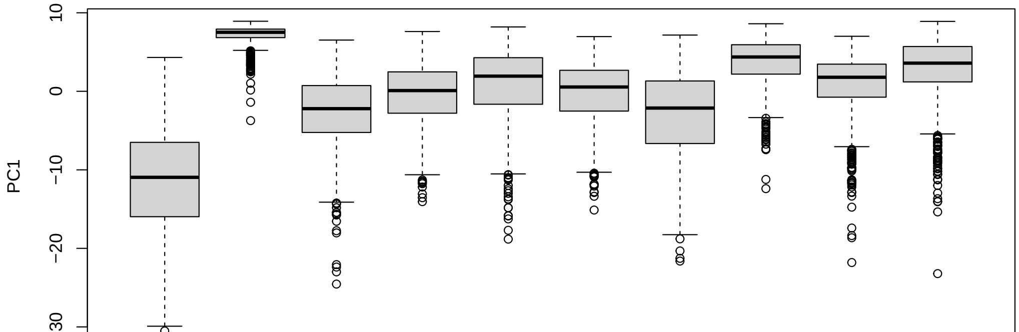
- The command to run the PCA is really simple:

```r
pca <- prcomp(x_train, center = TRUE, scale. = TRUE)
```

Using PCA values as classifiers

- The first principal component is pretty good at classifying some of the digits (especially zero and one):

```
boxplot(projected_df$PC1 ~ projected_df$digit,
        ylab = "PC1", xlab = "Digit")
```

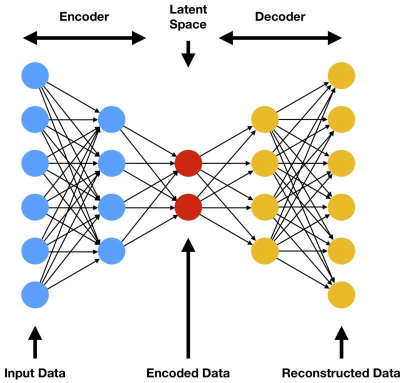


A different way of doing dimension reduction - autoencoders

- A very strange idea: why not fit a standard neural network model but have the features and the target to be the same!?
- How can this work? Rather than adding lots of extra weights in large hidden layers inside the model reduce the number of weights down to something much smaller
- Use those reduced weights as the dimension reduced version of the full data

Structure of an Autoencoder

- An autoencoder consists of an encoder, a code layer, and a decoder
- The encoder compresses the input into a latent-space representation
- The decoder reconstructs the input data from the latent space



From https://www.compthree.com/blog/autoencoder/

Training an Autoencoder

- Because the model is really just a neural network, just like all the others we have fitted, we can use `keras`
- We structure it slightly differently because we want access to the latent space in the middle of the model
- Once fitted, we access the latent space in the middle and plot, or use it for whatever purpose we have

Encoding Layer in R

- The encoding layer reduces the dimensionality of the input data
- We require an activation function, but we're not restricted to just one layer, we can use more if we want:

```r
autoencoder %>%
  layer_dense(units = 64, activation = 'relu',
              input_shape = 784)
```

Code Layer in R

- The code layer represents the compressed form of the input
- We use fewer units depending on how big a set of 'components' we want

```r
autoencoder %>%
  layer_dense(units = 32, activation = 'relu')
```

Decoding Layer in R

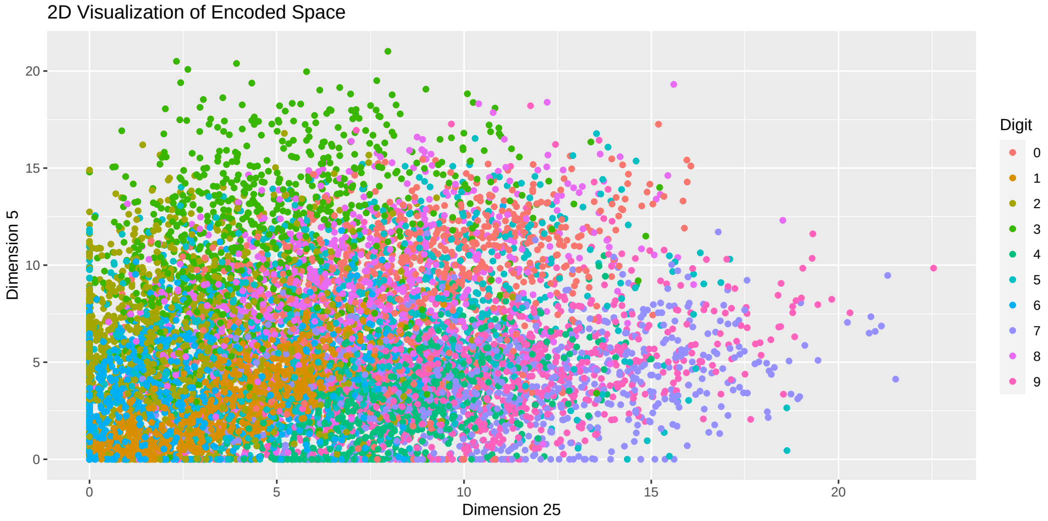
- The decoding layer tries to reconstruct the original data from the code layer
- It usually mirrors the structure of the encoding layer

```r
autoencoder %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dense(units = 784, activation = 'sigmoid')
```

Running the model

- We then compile and fit the model the same as always
- But then we extract the weights of that middle layer and predict (a forward pass) those values for each of the observations (in the training or the test set)
- We can then plot them and see how the values change across the observations

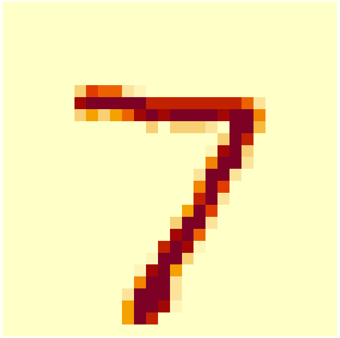
Example plot



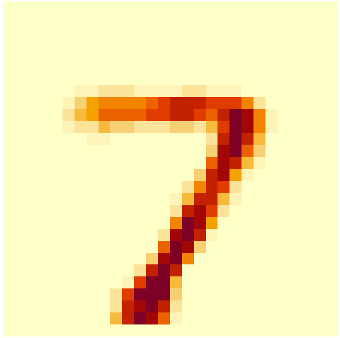
A reconstructed image

```
## 313/313 - 0s - 261ms/epoch - 835us/step
```

Original image. Label: 7



Reconstructed image. Label: 7



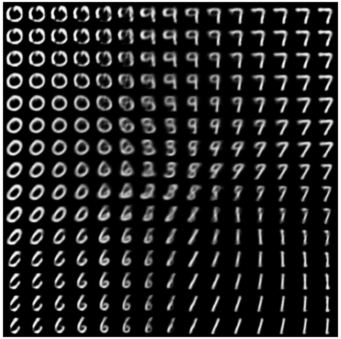
What do use an encoder for?

- You can use it to store a much smaller version of your data that contains most of the information
- You can use it to find observations that lie on the edge of the space and so might be considered anomalies
- If you find that some dimensions are capturing certain components of the data you can remove them (e.g. de-noising images)
- You could re-use the lower dimensional set as the features in another model which might be more efficient
- It can also be used to generate new data

Variational autoencoders

- A cool extension of autoencoders are **variational encoders** (VAEs)
- VAEs extend AEs by introducing a probabilistic framework, where the encoder models the input data as a probability distribution in the latent space
- Usually a normal distribution is used with an estimated mean and variance matrix
- The fitting and the code gets a bit more complicated
- But the cool thing is that you can then use them to **generate** new training data

Using VAEs to generate new images



PCA vs AE vs VAE

- PCA is fast but might not capture complex structures as effectively
- Autoencoders are more flexible and can capture non-linear relationships
- Variational AEs are richer again and allow for the generation of new data
- All methods are available in R for dimension reduction

Summary

- Dimension reduction very useful for all sorts of reasons, but especially if you have a lack of labelled data for supervised learning
- Pick a method that works for your data; I would always start with PCA
- We can use some of these methods later on for anomaly detection