COSC343: Artificial Intelligence

Lecture 16: Unsupervised learning

Alistair Knott

Dept. of Computer Science, University of Otago

In today's lecture

Some unsupervised learning algorithms:

- Principal components analysis
- Clustering: k-means and dendrograms
- Autoencoders

Supervised vs unsupervised learning

In the learning algorithms we've seen so far, the training data specifies inputs and outputs of a function.

- Linear regression: for input x, you should get output y (a number)
- Classification: for input x, you should get output C (a class)

But sometimes, the training data is just a bunch of data: there's no special 'output' dimension. We just want our machine learning algorithm to *organise* the data, or make sense of it somehow.

- Algorithms of this kind are called unsupervised learning algorithms.

Types of unsupervised learning algorithm

There are many different methods for 'making sense' of an arbibtrary set of data.

- Some of these involve *re-representing* the data.
- Some involve finding *patterns* in the data.

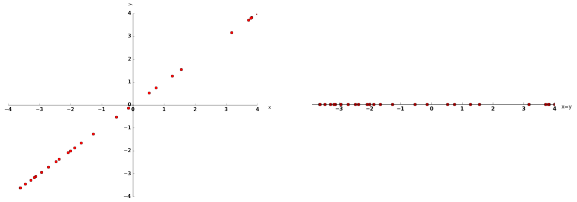
I'll give an example of each.

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.

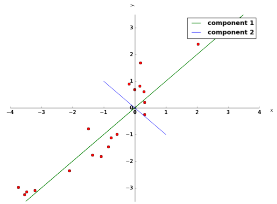


That's called dimension reduction.

Principal component analysis (PCA)

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

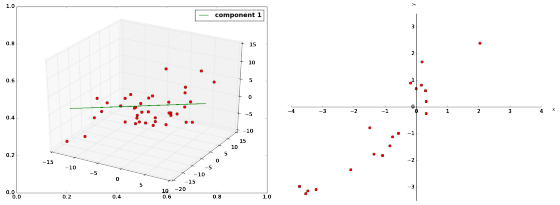
It's still useful to re-represent the data, using axes that explicitly identify the *components* of the variation.



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. (PCA is a dimensionality reduction technique.)

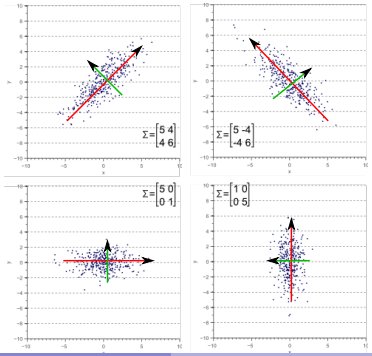
The maths of PCA

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.

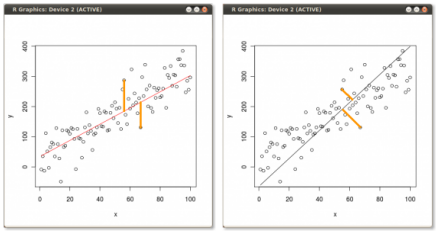


PCA and regression

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.



PCA in practice

Let's look at some high-dimensional data: a collection of images.

Each image is an item in our training set.

- Each image has 780 pixels (each representing a shade of grey).
- So the data occupies a space with 780 dimensions!

PCA in practice

We can compute the covariance matrix for this data, and identify its eigenvectors and associated eigenvalues.

- The top 30-40 eigenvectors account for nearly all the variation in the dataset.
- So we can re-represent our data in 780 dimensions in around 30-40 dimensions.
- These can be used as input to other algorithms (e.g. classification algorithms).

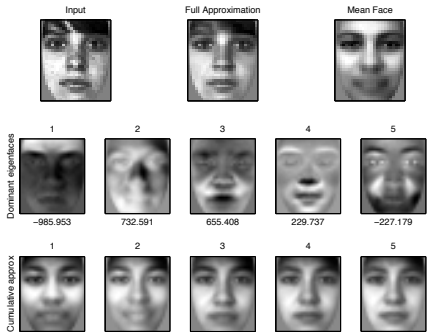
PCA in practice

We can actually visualise these eigenvectors, as points in our original 780-dimensional space (i.e. as images).

PCA in practice

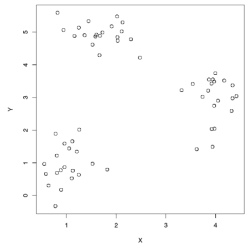
We can now take any image (from the same dataset) and re-represent it in a space based on the top *n* principal components.

Here are the dominant components for an example face:



Clustering

Sometimes items in a dataset seem to belong to different discrete categories, even if they're not labelled.



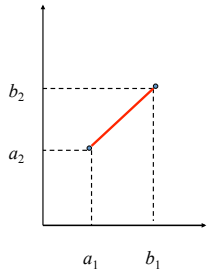
In this case, we want a way of identifying important *discrete regions* in the data, rather than important continuous axes.

The techniques that do this are called clustering techniques.

K-means clustering

K-means clustering is the simplest clustering algorithm.

- It partitions the dataset into *K* clusters. (You have to choose *K* in advance.)
- Clustering is done based on some measure of *similarity*. When the data occupies a continuous *n*-dimensional space, a common measure is Euclidean distance.



For *n* dimensions, the Euclidean distance between points $(a_1 \ldots a_n)$ and $(b_n \ldots b_n)$ is:

$$\sqrt{(a_1 - b_1)^2 + \ldots + (a_n - b_n)^2}$$

The *K*-means clustering algorithm

1. Randomly choose *k* points from the input space to be the starting centroids of the clusters ($c_1 \ldots c_k$).

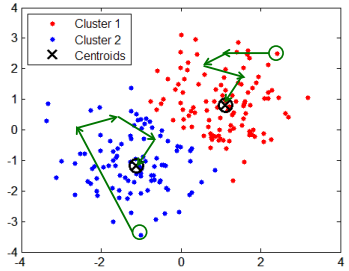
2. Then for each point in the dataset p_i:
   - Find the centroid c_j that's closest to p_i.
     (This identifies the cluster that it belongs to.)
   - Move the centroid c_j *towards* p_i in each dimension $d = 1 \ldots n$:

$$c_j^d \leftarrow c_j^d + \alpha(p_i^d - c_j^d)$$

3. Repeat from 2 until some stopping criterion. (E.g when the total distance moved by the centroids is less than some threshold.)

K-means clustering

Here's an example of the algorithm, for a 2-dimensional input dataset, and with $k = 2$.



Evaluating *K*-means clustering

We want to find a set of centroids c_j that minimise the sum of distances d between each point p and its assigned cluster C_j.

$$\sum_{j=1}^{k} \sum_{p \in C_j} d(p, c_j)$$

The *K*-means algorithm isn't guaranteed to minimise this sum.

- It's prone to finding *local* minima, based on the initial centroids.
- To alleviate the problem we can run the algorithm repeatedly, with different initial centroids.

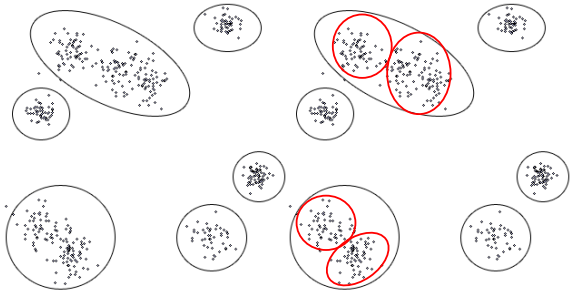
How many clusters should there be?

- Again we can run the algorithm several times with different *k*s...
- NB the algorithm 'finds' clusters in the data even if there are none!

Hierarchical clustering

Our data may be clustered at several different *hierarchical levels*.

Take a look at this data: how many clusters are there?



Hierarchical clustering algorithms

Hierarchical clustering algorithms work *recursively*.

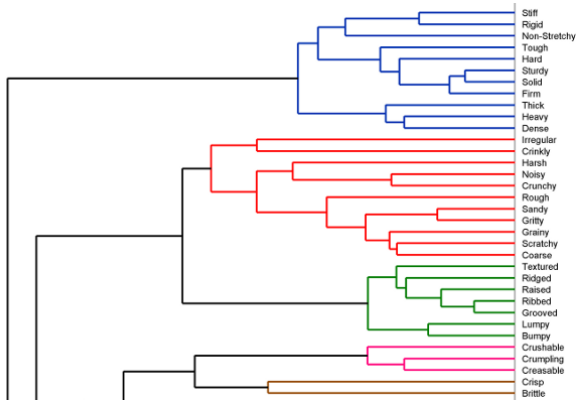
Here's a bottom up approach:

- Arrange the n closest pairs of points into n mini-clusters...
- Progressively form bigger clusters.

There are top-down methods too.

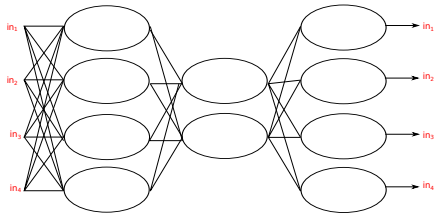
Dendrograms

The results of hierarchical clustering analysis can be presented in a dendrogram. Here's one clustering adjectives by semantic similarity:



Autoencoders

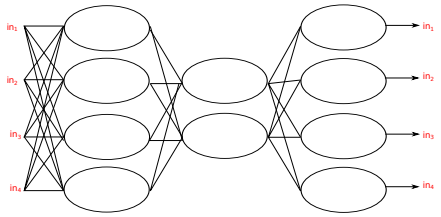
A nice trick is to build a multi-layer perceptron with one hidden layer, and train it to map each training item *onto itself*.



- The hidden layer is set to be smaller than the input/output layers.
- So the network has to learn to *compress* each data item.

Autoencoders

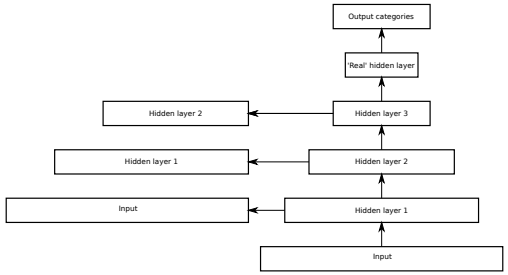
When you do this, what does the hidden layer learn?



- It can learn *correlations* in the input data (just like PCA)...
- It can learn *clusters* in the input data (just like k-means clustering).

Stacked autoencoders

It's possible to have several layers of autoencoders.
At the end, we can add a single output layer, for supervised training.



This is much easier to train than propagating errors all the way from the output units to the input units.

Summary

- Principal components analysis: good for reducing the dimensionality of data
- K-means clustering: a way of 'discovering' categories in the data
- Hierarchical clustering: a way of 'discovering' hierarchical categories in the data
- Autoencoders: a useful neural network technique for finding regularities in data.

Reading

No reading for this lecture.

Reading for next lecture: AIMA Sections 3.1–3.4.