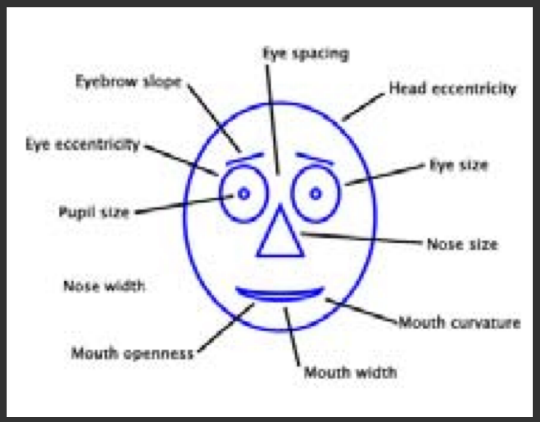
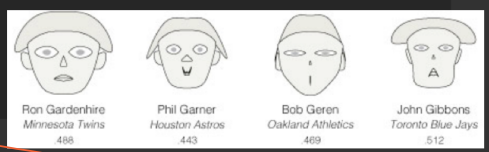
Chernoff Faces (1973)

Insight: We have evolved a sophisticated ability to interpret facial expression.

Idea: Map data variables to facial features.





Do not ever use this method!

1. expect certain coordinations, ratios and patterns among facial attributes

it's bias

⇒ interpret such features not near linear / independent ways

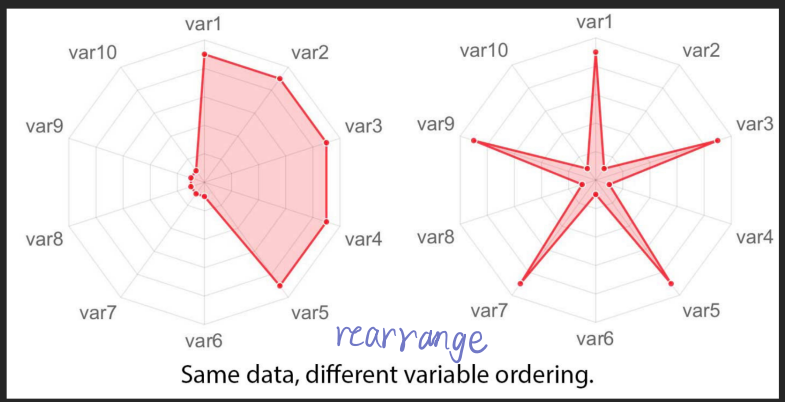
⇒ inaccurate encoding

2. sensitive and political

⇓

should use more abstract way

Radar Plot / Star Graph



rearrange

Same data, different variable ordering.

You have to be careful with radar plots, as the order of dimensions affects the interpretation a lot. For example, note how the visualisation the left conveys a different overall message about the magnitude of the 10-d data object than the visualisation on the right. Vary the order to explore the data, if using this method.

hard to see detail among data items as they overwrite each other

Dimensional reduction

This general approach is *dimensional reduction*

- Reducing data from high-D to low-D
- Reduction to 2D is *far* better than to 3D

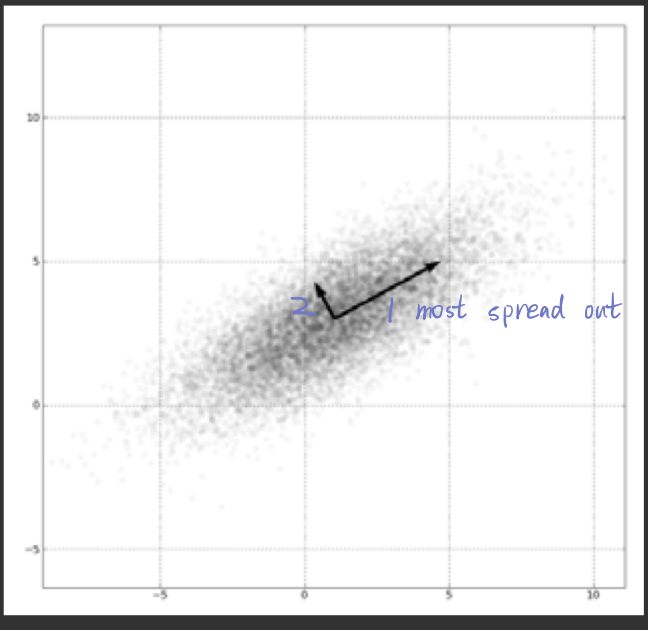
3D plot are generally not useful, scatterplot matrices are also not good (especially with large dimensions)

Sometimes called *multidimensional scaling* (MDS), but that often suggests only a subset of dimensional reduction methods (to me!)

Many methods available, e.g. matrix methods (e.g. PCA), spring models, tSNE, UMAP...

one of MDS (linear model)

Principal Component Analysis

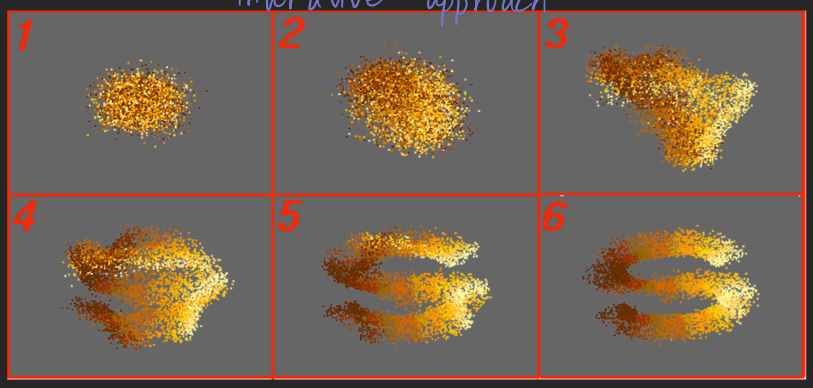


1. Mean-center the data.
2. Find ⊥ basis vectors that maximize the data variance.
3. Plot the data using the top vectors.

Non-linear model (force based model)

Spring Model Layout Process

iterative approach



Snapshots from different stages of a spring model layout. Synthetic "S" data set to help show what's happening. From random initial positions, structure slowly appears as similar (but far) objects pull closer to each other, and dissimilar (but close) ones push apart, in a simulated system of springs, forces and movement.

Spring models

(interative)
run until stable

Simulate a spring between each pair of objects

- Ideal relaxed length of spring proportional to high-D distance between objects

Example: three rows from spreadsheet: call them A, B and C

- Objects A&B quite similar; C quite different to A and to B
- So, AB is a short spring; AC and BC longer springs

CUSTOMER_ID	PRODUCTTYPE	CURRENCY_IS	CUSTOMER_SEG	YIELD	DAYS_TO_MATI	AMOUNT_CHF(	FULLNAME
21	276	AUD	AAA	7.224	1002	250	AUD Eurobonds
21	276	AUD	AAA	7.266	1044	227	AUD Eurobonds
361	2	CHF	CCC	4.204	2609	50	CHF Domestic E

the ideal length of a spring is proportional to the high dimensional distance between 2 objects

Spring models

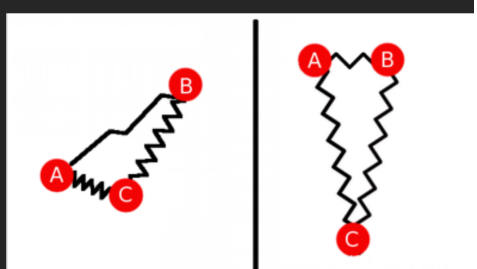
AB is short; AC & BC longer

Start from random positions (left image)

- Some springs too stretched, others too squashed

The spring forces then iteratively push and pull objects until the forces reduce,

- i.e. the layout process reaches



Force-based models

Spring models: one example of force-based models

- far but similar objects *i* and *j* pull each other closer
  - direction of force on *i* is towards *j*, dirn of force on *j* is towards *i*
- close but dissimilar objects *i* and *j* push each other apart
  - dirn of force on *i* is away from *j*, dirn of force on *j* is away from *i*

Other models have also been explored, using simulated forces, velocity, temperature, mass...

- Ideas often based on mechanical systems, the gravitational systems of planets, etc.
- …but simplified so as to be coded more easily, to run more quickly, to suit particular distance metrics, or styles of visualization, etc.

A simple spring model algorithm

```
Initialise 2D positions randomly
for each iteration // run until 'enough' work has been done
  for each object i {
     totalForce = [0,0];  // a 2D vector used to accumulate forces
        for each other object j
        totalForce += force(i, j);  // force() method - see next slide
                  control the unstable
     // ts: a 'time step' constant, that says how strongly to bring
     // in the newly calculated forces.
     velocity(i) += ts * totalForce;   // apply acceleration
     position(i) += ts * velocity(i);
  }
```

Similarity Metrics

The metric is vital to the *force(i,j)* method

$|force(i,j)| \propto |idealDistance(i,j) - currentLayoutDistance(i,j)|$

But what is the 'ideal' (high-dimensional) distance? It depends!

A simple example: treat spreadsheet as matrix of data *Mic*

Each row *i* is an object, e.g. a bond trade, and each column *c* is the values for one data dimension, e.g. *yield*

Similarity of two objects *Mi* and *Mj* uses a combination of similarities for each column

Handle numerical columns (reals, integers…) and categorical columns (strings, names…) separately, using simpler metrics

CUSTOMER_ID	PRODUCTTYPE	CURRENCY_ISC	CUSTOMER_SEG	YIELD	DAYS_TO_MATI	AMOUNT_CHF(	FULLNAME
21	276	AUD	AAA	7.224	1002	250	AUD Eurobon
21	276	AUD	AAA	7.266	1044	227	AUD Eurobon
361	2	CHF	CCC	4.204	2609	50	CHF Domestic

Force between each pair objects *i* and *j* relies on similarity metric *s(i,j)*

Note that spring models have idealDistance = *0* for exact matches, bigger distance for mismatches… so metric might be called '*dis*similarity'

Quantitative: first do sum of normalised differences

Normalise each column to be between 0 and 1, and set *s=0*

For each normalized column *c* :

s += |*Mic* - *Mjc*|

Categorical: each mismatch scales up *s*

For each ordinal column *c*:

if *(Mic ≠ Mjc)* then *s = 1.25 * s*

s = s / D // finally, divide by the number of columns D

So now *s* = ideal, high-dimensional distance between objects *i* and *j*

all of these dimensions build a para

→ then replicate in a spring model

→ move objects to 2D distance

Layout Error

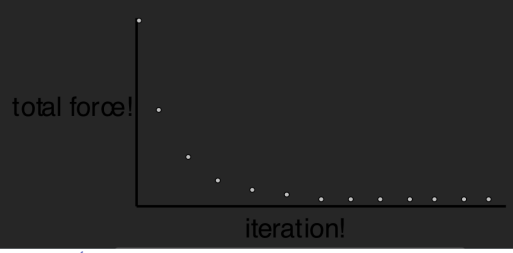
We can measure the current 'error' in the layout: how far from 'perfection' is the current iteration? How many iterations before we stop?

With each iteration, forces act to decrease layout's error (but never to 0)

- Error can be thought of as sum over all pairs *i,j* of |*force(i,j)*|
- Spring equilibrium is a compromise: not all relationships *perfectly* represented
- Error might never be zero, though it should get lower and lower
- We can watch for when it is stable (and low), and then stop iteration

Model may get stuck in a local minimum rather than ideal global minimum (因为 because of randomness or jitter)

- Can do multiple layouts, and choose the layout with the lowest error
- And/or add stochastic 'jitter' or 'heat' to help shake free of such minima



1. cheap way : use the force in the spring model as the total diagram
2. expensive way : run the algorithm multiple times with different random starting position.
   → look for the one with lowest error

Spring Models: Strengths & Weaknesses

Layout positions show global and local structure

- Neighbours on 2D layout are usually high-D neighbours, but distances between non-neighbours are also pretty good

Many (many) dimensions can be combined in a compact 2D vis

- Not for exploring individual dimensions, unlike other techniques

Simple algorithms have quadratic iteration time, and we may need roughly *O(N)* iterations

- *N(N-1)* force calculations so *O(N2)* per iteration, and *O(N3)* overall
- May mean N has to be very small, e.g. 10K, 50K

Faster spring algorithms run at *O(N)* per iteration or less, using various sampling, caching and clustering schemes

Hierarchical Methods for Force Models

haddk bigger numbers

Classic work on this is the *Barnes-Hut approximation*, from 1986

- Divide up the layout space with a binary tree of cells. Each branch combines all the objects lower in the tree. Positioned is centre of mass.
- One can approximate many long-range forces, by one force based on a branch object, to bring force calculation down from *O(N2)* to *O(N.logN)*

just use one force calculation

nearby : exact

For iteration, for each object *i*, find the force from all other objects

- For each nearby cell, do exact force calculations for every other object in those cells
- For each distant cell, do one force calculation using branch node's centre of mass

Note that the tree has to be rebuilt, after each iteration… which can be a substantial cost. (This is where sampling-based approaches win out.)

still faster than basic force / quadratic time force calculation

Sampling Methods for Force Models

much quicker

Early (first?) example is Chalmers' 1996 paper, demo'd in Fsmvis

- Like Barnes-Hut, use fewer force calculations for *dissimilar* objects
- One can use a combination of samples and known high-D neighbours, to bring force calculation down from *O(N2)* to *O(N)*
- Relies on each object *i* having two sets: neighbours, and a random sample set
- Initially each neighbour set is empty, but has a constant maximum size (e.g. 5)

For each iteration

- For each object
  - Add up forces from any/all neighbours
  - Get a new random sample (e.g. 10) of all other objects, and add forces from these samples
  - Check each sample, and add it to neighbour set if it's closer than furthest neighbour (or free space)
  - Adjust velocity and position of this object

Stochastic Neighbour Embedding

Original SNE from Hinton and Roweis in 2008

- Many variants; t-SNE (van der Maaten & Hinton 2008) now the best known

Like springs, all dimensions are combined in 2D layout

- Unlike springs, similarity metric is *KL-divergence*. Attempts to make the *statistical distributions of neighbours* in low-D and in high-D match
- Focus is on local 'neighbour' structure, but not much concern about high-D non-neighbours
- High penalty for neighbours that are far apart in layout, but low penalty for non-neighbours that are close in layout
- So, global structure is poor (or seen as not important)
- Costly initial phase of calculating neighbours and distributions, then *gradient descent* to move objects into position
- Makes for time being quadratic in N, but also problems of very large matrices

not just nearby (metric: chaotic divergence)

not really directly modelled

the square of the number of objects is proportional to the time it takes to do the calculation

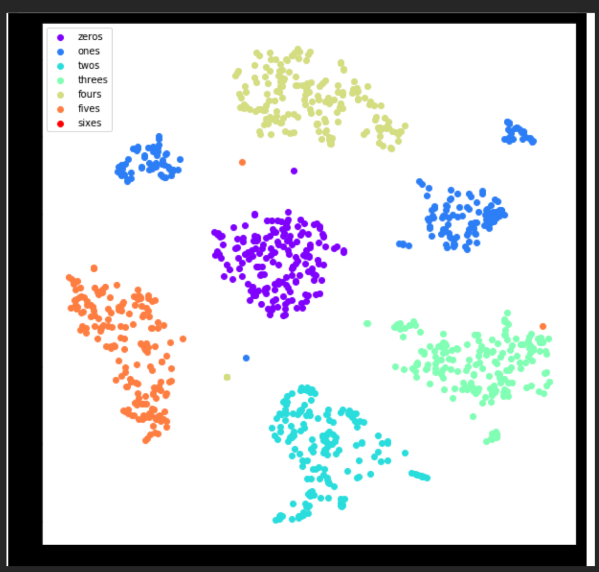
Stochastic Neighbour Embedding

SNE favours clear separate clusters

- People like this, although SNE may make clusters where none exist!
- Very popular in machine learning community, sadly

SNE, tSNE and their variants don't scale well to large data sets

- High memory & computational costs, so N usually tiny, e.g 10k-50k
- Some hierarchical approximations help a lot, though



Layout of a subset of MNIST data set of images of written digits (from blog.paperspace.com)

Excellent interactive overview and critique by Wattenberg and Viegas

UMAP: Uniform Manifold Approximation and Projection (McInnes et al., 2018)

Newer, faster and less biased (than tSNE) algorithm

- Again, out of the ML community. Some very complex maths involved!

Core idea is to find the highD neighbours for each object, but also a model (manifold) that describes connections/distances between objects

- Uses some advanced topological methods to do this
- Also applies very fast algs for finding (approximate) neighbours in high-D spaces (random projection trees, and nearest neighbor descent), and fast variants on stochastic gradient descent.

Python code available on Github here

Best overview is perhaps this video from SciPy conf 2018, on YouTube

UMAP

Balances local & global structure

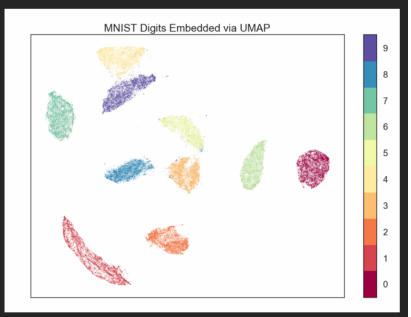
- Keeps similar objects close, but dissimilar ones far
- Rising fast in machine learning community

Ultimately, it's very similar to fast sampling-based spring models

- But UMAP tailors its model to represent inter-object relationships in a more in-depth way

Scales well to large N

- Up to a few million, anyway
- Order of complexity not proven yet, but McInnes guesses O(d*N1.14)



Layout of the 70000 MNIST data set of images of written digits (from McInnes' Github page)

Visualising Multidimensional Data

Strategies

- First, visualise individual dimensions (maybe of a sample)
- Avoid 'over-encoding' too many dimensions onto one visualisation
- Use space and small multiples (e.g. scatterplot matrix) intelligently
- Dimensional reduction is useful when there are too many dimensions for small multiples
- Use interaction to explore linked *relevant* views

There is rarely a single visualisation that answers all questions. Instead, the ability to create/combine/explore appropriate visualisations quickly is key