Data Mining
Lecture 4: Embedding Data

Jo Grundy

ECS Southampton

March 2, 2023

Embedding Data

Understanding large data sets is *hard*
Especially when the data are highly dimensional
It would help if we knew:

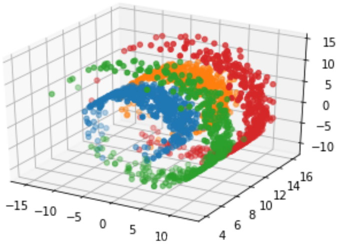
- which data items are similar
- which features are similar

Embedding Data

With 2D data, we can plot it to easily visualise relationships
This is not possible with highly dimensional data
However: PCA can reduce the dimensionality to 2, based on the first and second principle axes

Embedding Data - PCA

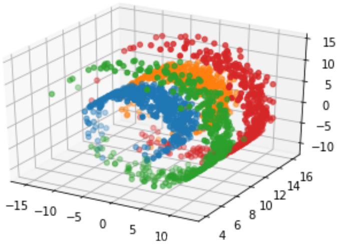
We can use the so called 'Swiss Roll' data set to exemplify this:

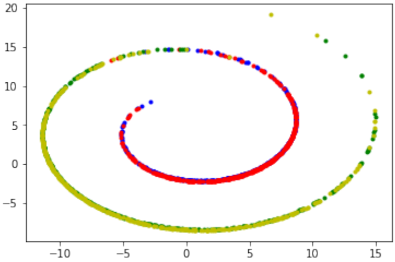


The data in 3D has 4 clearly separate groups

Embedding Data - PCA

We can use the so called ‘Swiss Roll’ data set to exemplify this:





The data in 3D has 4 clearly separate groups

Using PCA, it does not separate the data well at all

Embedding Data - PCA

Unfortunately there is no control over the distance measure

Using axes of greatest variance does not mean similar things appear close together.

PCA is only rotation of original space, followed by removal of less significant dimensions

Embedding Data - Self-Organising Maps

Kohonen 1982: Self-Organising Maps (SOM)

- Inspired by neural networks
- 2D n by m array of nodes
- Units close to each other are considered to be neighbours
- Maps high dimensional vectors to unit with coordinates closest (Euclidean Distance)
- This is the *best matching unit*

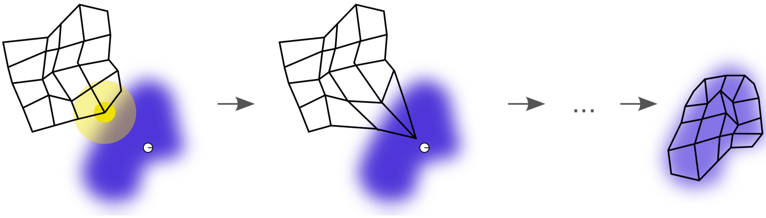
Embedding Data - Self-Organising Maps

SOMs: two phases

- training
- mapping

To start training, the set of nodes each has a random starting position defined in the feature space.

This is then updated by taking one feature vector, finding which unit is the *best matching unit* (BMU) then moving that unit and, to a lesser extent, its neighbours, closer to that data point



Embedding Data - Self-Organising Maps

Algorithm 1: Training Self-Organising Maps

Data: N data points with d dimensional feature vectors X_i
$i = 1 \ldots N$, number of iterations λ

$\boldsymbol{w}$ = randomly initialise n xm units with weight vector ;
t = 0;
while $t < \lambda$ **do**
    for *each* x_i **do**
        BMU =w_{nm} with min distance;
        Update BMU and its neighbours by moving closer to x_i;
    end
    $t = t + 1$
end

Embedding Data - Self-Organising Maps

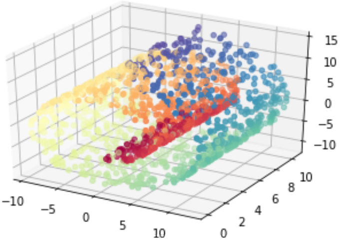
To update the weight vector $\boldsymbol{w}$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \theta(u, v, t)\alpha(t)(x_i - \boldsymbol{w}(t))$$

where θ is the neighbourhood weighting function (usually Gaussian)
α is the learning rate
x_i is the input vector
u is the unit, v is the

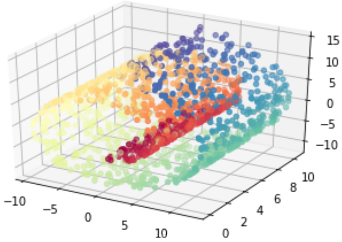
Both the learning rate and the neighbourhood weighting function get smaller over time
Java SOM demo

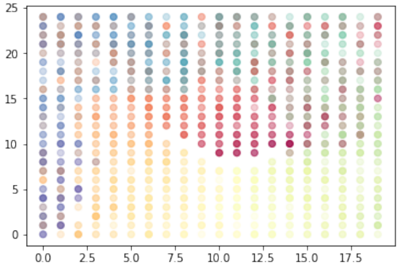
Embedding Data - Self-Organising Maps



The data in 3D has a clear structure

Embedding Data - Self-Organising Maps



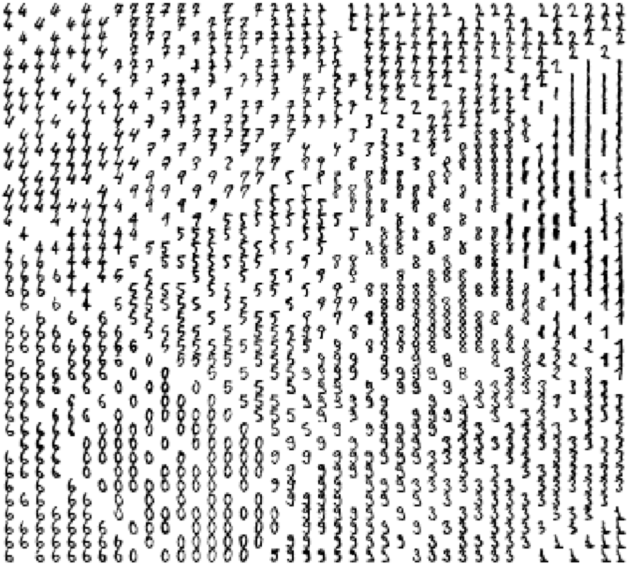


The data in 3D has a clear structure

Using SOM, it can much better group the data

Embedding Data - Self-Organising Maps

With the MNIST digits, the results are quite impressive



Embedding Data - Multidimensional Scaling

Multi Dimensional Scaling involves:

- Start with data in a high dimensional space and a set of corresponding points in a lower dimensional space
- Optimise the positions of points in lower dimensional space so their Euclidean distances are *like* the distances between the high dimensional points
- Can use any distance measure in the high D space

Embedding Data - Multidimensional Scaling

There are two main sorts of multidimensional scaling:

- Metric MDS - Tries to match distances
- Non-metric MDS - tries to match rankings

Only requires distances between items as input
Unlike PCA and SOM, there is no explicit mapping

Both metric and non-metric measure goodness of fit between two spaces
They try to minimise a *stress function*

Embedding Data - Multidimensional Scaling

Stress functions:

- Least-squares scaling / Kruskal-Shepard scaling
- Shepard-Kruskal non-metric scaling
- Sammon Mapping

Sammon Mapping is given by:

$$S(z_1, z_2, \ldots, z_n) = \sum_{i \neq j} \frac{(\delta_{ij} - ||z_i - z_j||)^2}{\delta_{ij}}$$

where $\delta_{i,j}$ is the distance in high dimensional space
and $||z_i - z_j||$ is the distance in low dimensional space
Looks at all combinations of points with all different points.

Embedding Data - Multidimensional Scaling

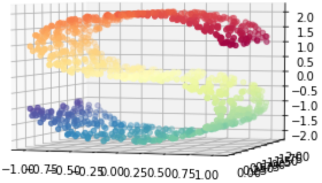
For non-linear, need to use gradient descent
start at arbitrary point, take steps in direction of gradient, with step size a proportion of the gradient magnitude.

$$z_j(k+1) = z_j(k) - \gamma_k \Delta_{z_j} S(z_1(k), z_2(k), \ldots, z_n(k))$$

Where the derivative of the Sammon stress is:

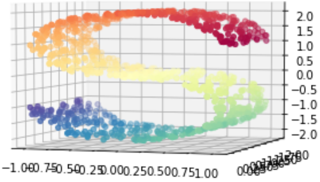
$$\Delta_{z_j} S() = 2 \sum_{i \neq j} \Big(\frac{||z_i(k) - z_j(k)|| - \delta_{ij}}{\delta_{ij}} \Big) \Big(\frac{z_j(k) - z_i(k)}{||z_i(k) - z_j(k)||} \Big)$$

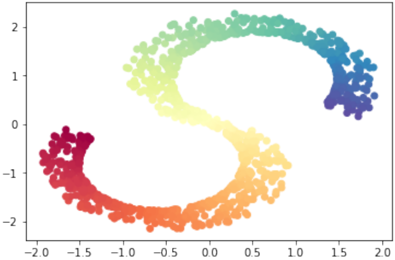
Embedding Data - Multidimensional Scaling



The data in 3D has a clear structure.
MDS gives..

Embedding Data - Multidimensional Scaling





The data in 3D has a clear structure.
MDS gives..

Not particularly brilliant, has red the same distance from yellow as orange and green
Has however preserved the structure from 3D in to 2D

Embedding Data - Stochastic Neighbour Embedding

Stochastic Neighbour Embedding (SNE)
Works in a similar way to MDS
MDS optimises distances, SNE optimises the distribution of data
Aims to make the distribution of the projected data in low dimensional space close to the actual distribution in high dimensional space

Embedding Data - Stochastic Neighbour Embedding

To calculate the source distribution:

We define a conditional probability that high-dimensional x_i would pick x_j as a neighbour if the neighbours were picked in proportion to their probability density under a Gaussian centred at x_i

$$p_{j|i} = \frac{exp^{-||x_i - x_j||^2/2\sigma_i^2}}{\sum_{k \neq i} exp^{-||x_i - x_k||^2/2\sigma_i^2}}$$

The SNE algorithm chooses σ for each data point such that smaller σ is chosen for points in dense parts of the space, and larger σ is chosen for points in sparse parts

Embedding Data - Stochastic Neighbour Embedding

To calculate the target distribution:

Define a conditional probability that low-dimensional y_i would pick y_j as a neighbour if the neighbours were picked in proportion to their probability density under a Gaussian centred at y_i

$$q_{j|i} = \frac{exp\big(-||y_i - y_j||^2\big)}{\sum_{k \neq i} exp\big(-||y_i - y_k||^2\big)}$$

In this space we assume the variance of all Gaussians is $1/\sqrt{2}$ in this space

Embedding Data - Stochastic Neighbour Embedding

To measure the difference between two probability distributions we use the *Kullback-Leibler* (KL) Divergence:

$$D_{KL}(P|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

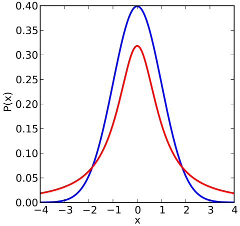
The cost function is the KL divergence summed over all data points

$$C = \sum_i \sum_j p_{i|j} \log \frac{p_{j|i}}{q_{j|i}}$$

C can be minimised using *gradient descent* - but..
difficult to optimise, leads to crowded visualisations, big clumps of data together in the center

Embedding Data - t- Distributed Stochastic Neighbour Embedding

In 2008 Maaten and Hinton came up with a way to improve SNE, by replacing the Gaussian distribution for the lower dimensional space with a Student's t distribution.



Student's t distribution in red, Gaussian distribution in blue

The Student's t distribution has a much longer tail, helps avoid clumping in the middle.

Embedding Data - t- Distributed Stochastic Neighbour Embedding

The cost function is also modified, making gradients simpler, so faster to compute

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

To alleviate crowding using Student's t distribution in the lower dimensional space:

$$q_{ij} = \frac{(1 + ||x_i - x_j||)^{-1}}{\sum_{k \neq i}(1 + ||x_i - x_k||)^{-1}}$$

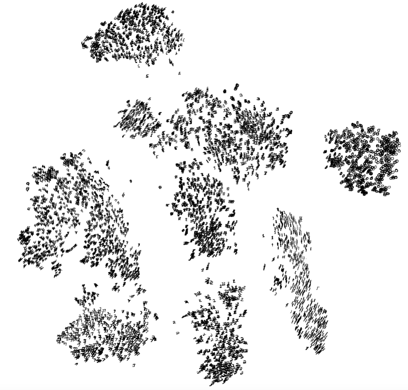
we use 1 degree of freedom with the Student's t distribution, equivalent to a Cauchy distribution

Embedding Data - t- Distributed Stochastic Neighbour Embedding

SNE gives good groupings, but without clear separation

Embedding Data - t- Distributed Stochastic Neighbour Embedding

SNE gives good groupings, but without clear separation

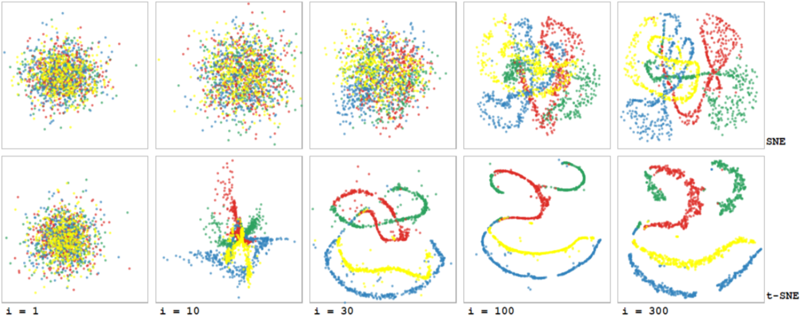


t-SNE gives clear separated groups

van der Maaten and Hinton, JMLR (2008) 2579

Embedding Data - t- Distributed Stochastic Neighbour Embedding

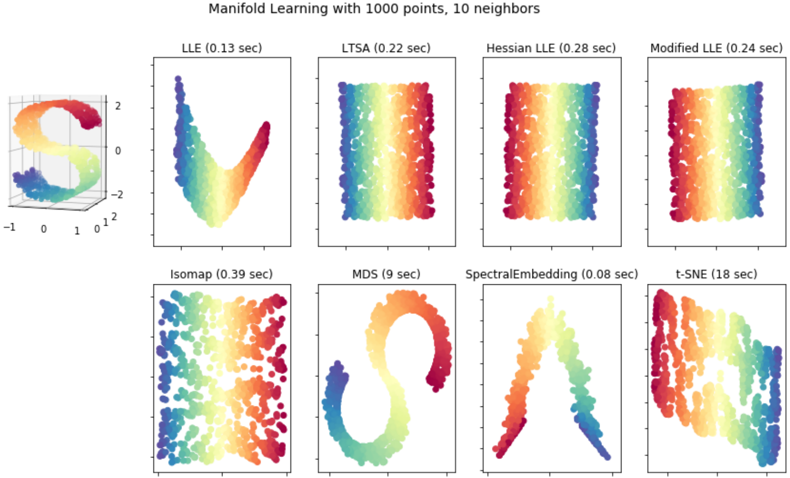
For the Swiss Roll data:



Lorenzo Amabili, `http://lorenzoamabili.github.io`

Embedding Data

Many embedding techniques are available!



Code from sklearn.manifold

`https://scikit-learn.org/stable/modules/manifold.html`

Embedding Data

Instead of projecting high dimensional data down in to 2 or 3 dimensions, we can use a medium dimensionality, keeping the useful information, capturing the key distinguishing features

This is called an *embedding*

For example: word2vec

Embedding Data - One Hot Encoding

For example: Documents

We use a ‘Bag of Words’, where each word is a vector:

- a $\rightarrow [1, 0, 0, 0, 0, 0, 0, 0, \ldots, 0]$
- aa $\rightarrow [0, 1, 0, 0, 0, 0, 0, 0, \ldots, 0]$
- aardvark $\rightarrow [0, 0, 1, 0, 0, 0, 0, 0, \ldots, 0]$
- aardwolf $\rightarrow [0, 0, 0, 1, 0, 0, 0, 0, \ldots, 0]$

This is called *One Hot Encoding*
(also seen in the Discovering Groups lecture)

Embedding Data - One Hot Encoding

What problems does this encoding have?

- all vectors are *orthogonal*, i.e. unrelated

But we know that many words in English are related, e.g. ‘rain’, ‘drizzle’, ‘downpour’, ‘shower’, ‘squall’ all mean pretty much the same thing.

- vectors are very long and *very* sparse

English has over 250,000 words (depending on how you count them) so each vector that could fully describe a document should be 250,000 long for every word.

Embedding Data - word2vec

Boiling this data down to a manageable vector size, while still retaining meaning is not a simple task

word2vec was proposed in 2013 by Mikolov *et al* (although the paper was rejected by the ICLR conference!)

It involved using a simple neural net to predict the words on either side of the target word

Embedding Data - word2vec

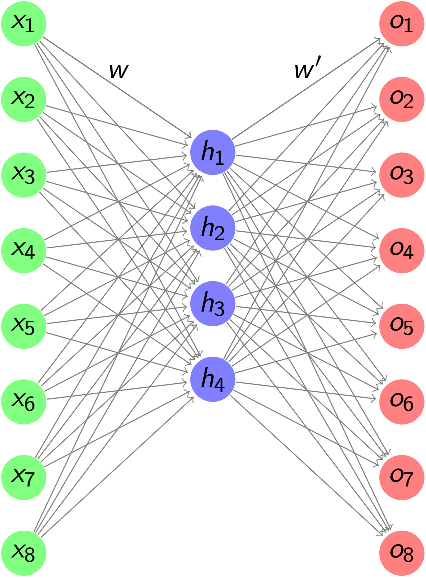
For example:
"the quick brown fox jumps over the lazy dog"

The word 'brown' has the words 'the', 'quick', 'fox' and 'jumps' close by
This gives training samples:

- 'the', 'brown'
- 'quick', 'brown'
- 'fox', 'brown'
- 'jumps', 'brown'

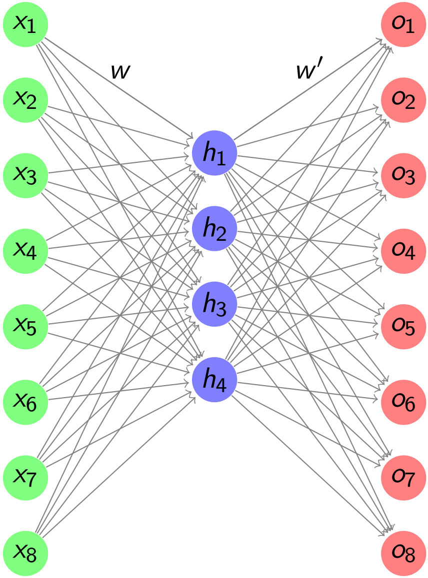
which we train a simple neural network with.

Embedding Data - word2vec



input vector is the 'one hot' encoding of the word in question

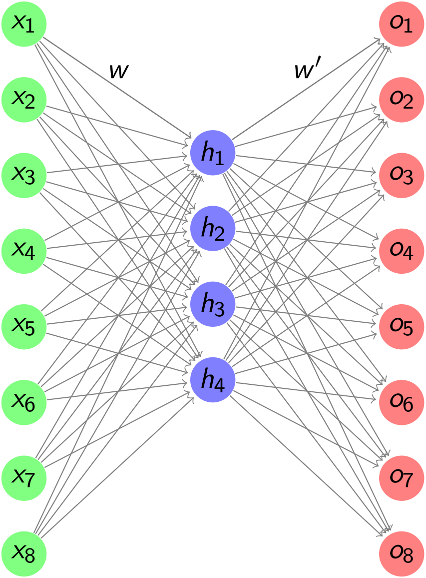
Embedding Data - word2vec



input vector is the 'one hot' encoding of the word in question

the output vector is the vector for the predicted word.

Embedding Data - word2vec

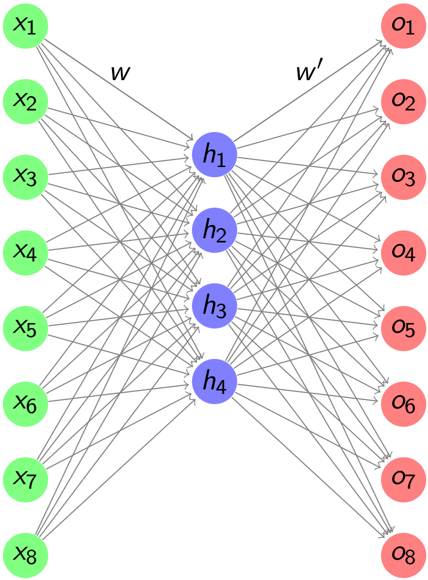


input vector is the ‘one hot’ encoding of the word in question

the output vector is the vector for the predicted word.

the hidden layer learns a lower dimensional encoding of each word

Embedding Data - word2vec



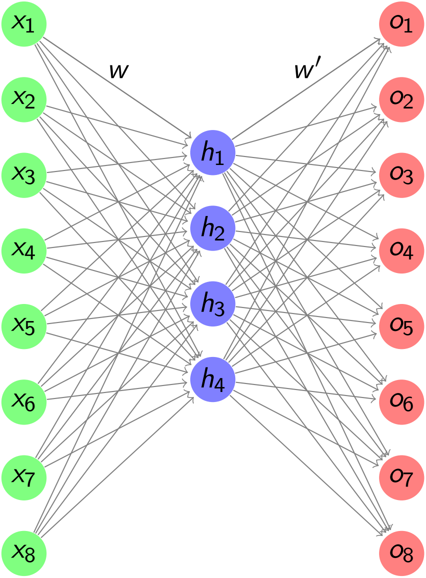
input vector is the 'one hot' encoding of the word in question

the output vector is the vector for the predicted word.

the hidden layer learns a lower dimensional encoding of each word

in training the correct word is used as the teaching signal, and back propagation is used to learn the weights to the hidden layer.

Embedding Data - word2vec



input vector is the 'one hot' encoding of the word in question

the output vector is the vector for the predicted word.

the hidden layer learns a lower dimensional encoding of each word

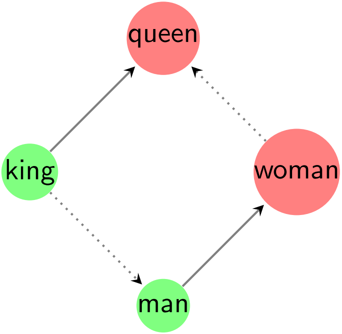
in training the correct word is used as the teaching signal, and back propagation is used to learn the weights to the hidden layer.

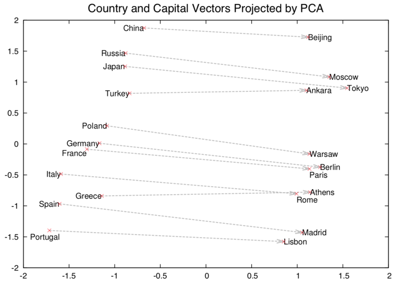
the output from the hidden layer after training is used as the lower dimensional representation

Embedding Data - word2vec

These lower dimensional representations can include a good deal of semantic meaning

i.e. vector(king) - vector(man) + vector(woman) $\approx$ vector(queen)





From Mikolov *et al* NIPS 2013

Embedding Data - Summary

Dimensionality reduction and visualisation is key to understanding the data

Useful for your coursework

There are many ways to do this, with the `sklearn.manifold` library: