Discovering Groups

Summary

Being able to group data into clusters using clustering techniques is a good basis for understanding that data, however, in many cases these clusters can be difficult to interpret. An alternative approach is to attempt to produce 2D visualisations (images) that highlight the key relationships in the data by projecting the data from a high dimensionality to two dimensions. We'll look at four key algorithms: Principal Component Analysis (PCA), Self Organising Maps (SOMs), Multidimensional Scaling (MDS) and Stochastic Neighbour Embeddings (SNE & t-SNE).

Key points

Visualising Data in Two Dimensions

- Sometimes rather than clustering data explicitly, we just want a way to visualise (through an image or diagram) which items are similar to each other and which are highly dissimilar
    - Basically we want to map high dimensional data into a lower dimensional space in a meaningful way
    - Lots of techniques allow us to do this:

Principal Component Analysis (PCA)

- Principal component analysis allows us to project hight dimensional data into a lower dimensional space
- Could use PCA to create a scatter plot where the x and y axis are the first and second principal components
    - No control over distance measure
    - Just because axes are the oriented along greatest variances, doesn't mean similar items will appear close to each other

Self Organising Maps

- A self-organizing map (SOM) or a *Kohonen Map* is a type of artificial neural network (ANN) that is trained using unsupervised learning to produce a low-dimensional (typically two-dimensional), discretised representation of the input space of the training samples, called a map.
- Self-organizing maps are different from other artificial neural networks
    - they apply competitive learning as opposed to error-correction learning (such as backpropagation with gradient descent),
    - they use a neighbourhood function to preserve the topological properties of the input space.
- However, it's best not to think of SOMs in terms of neural networks!
    - Consider a SOM as an n by m grid of *units* where each unit has a weight vector with dimensionality equal to the dimensionality of the input vectors
        - This is known as the *map*
        - Units that are spatially close together are considered to be neighbours
        - The *location* of a unit in the grid can be considered to be a coordinate in 2D space
            - The SOM maps high dimensional vectors to a 2D coordinate given by the unit which has a weight vector which is most similar to the input vector (typically in terms of Euclidean distance); this unit is called the *best matching unit*
- The are two parts to using a SOM

        - The training process in which the weights of the units are learned
        - The projection process in which an vector is assigned to the best matching unit (BMU)
            - The coordinate of this unit is the projection of the input vector onto the 2D plane
- Training a SOM:
    - Prerequisite definitions - Let:

        - s define the current iteration
        - λ define the maximum number of iterations
        - t define the index of the target vector in the input dataset $\mathbf{D}$
        - $\mathbf{D}(t)$ defines the target input vector
        - v define an the index of a unit in the map
        - $\mathbf{W}_v$ define the weight vector of unit v
        - u is the BMU in the map
        - $\Theta(u,v,s)$ defines the neighbourhood weighting function
            - This produces a weight for the update of a neighbouring node based on its distance from the BMU
            - Common to use a Gaussian function
        - $\alpha(s)$ defines the learning rate
            - Typically this is a function that falls off as iterations increase
            - for example: $r_{initial}\exp(-s/\lambda)$, where $r_{initial}$ is the initial learning rate (usually a small value between 0.1 and 0.001).

    - Algorithm:

        1. Randomly assign weights to each unit
        2. Traverse each input vector in the input data set
            1. Find the BMU by computing the Euclidean distance of the input vector to each unit and picking the unit with the smallest distance
            2. Update the units in the neighbourhood of the BMU (including the BMU itself) by pulling them closer to the input vector: $\mathbf{W}_v(s+1) = \mathbf{W}_v(s) + \Theta(u, v, s)\,\alpha(s)(\mathbf{D}(t) - \mathbf{W}_v(s))$
        3. Increase s and repeat from step 2 while $s < \lambda$

- The SOM idea generalises in a number of ways:
    - The map can be modified to have more (or fewer dimensions)
    - The map needn't be a regular grid; any lattice structure upon which a neighbourhood function can be defined will work
        - Hexagonal lattices are fairly popular

Multidimensional Scaling (MDS)

- Multidimensional Scaling (MDS) is an alternate approach to embedding high-dimensional data in a lower dimensional (typically 2D) space.
- Two main categories:
    - Metric MDS: tries to optimise layout of points so that Euclidean distances in the lower dimensional space match original distances
    - Non-metric MDS: attempts to directly maintain the ordering or rank between items in the 2D projection compared to the ordering of the original distances
- Only requires distances between items as input
    - Unlike PCA and SOM there is no explicit mapping from points in the higher dimensional space to points in the lower dimensional space
- Irrespective of the category of the MDS algorithm, the key idea is to minimise a stress function
    - The stress function describes how well the interpoint dissimilarities in the low-dimensional space preserve those in the original space

- Depending on the choice of stress function, the embedding can be non-linear
- A popular stress function for non-linear metric scaling is the Sammon Mapping:

$$S(\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_n) = \sum_{i \neq j} \frac{(\delta_{ij} - \|\mathbf{z}_i - \mathbf{z}_j\|)^2}{\delta_{ij}}$$

  where z_i is the lower-dimensional vector for the i-th item and δ_{ij} is the original distance between items i and j
- Other popular stress functions include the
  - *Least-squares scaling* or *Kruskal-Shepard scaling*
  - *Shepard-Kruskal non-metric scaling*
- Some stress functions can be solved using Eigendecomposition, however many must be solved using *gradient descent* based optimisation
  - e.g. for Sammon Mapping:
    - Each point z_j can be iteratively updated by:

$$\mathbf{z}_j(k+1) = \mathbf{z}_j(k) - \gamma_k \nabla_{\mathbf{z}_j} S(\mathbf{z}_1(k), \mathbf{z}_2(k), ..., \mathbf{z}_n(k))$$

      where γ is a scalar *learning rate* (Sammon's original paper refers to this as the "magic factor"!) and the derivative of the stress function w.r.t z_j is:

$$\nabla_{\mathbf{z}_j} S(\mathbf{z}_1(k), \mathbf{z}_2(k), ..., \mathbf{z}_n(k)) = 2 \sum_{i \neq j} \left(\frac{\|\mathbf{z}_i(k) - \mathbf{z}_j(k)\| - \delta_{ij}}{\delta_{ij}} \right) \left(\frac{\mathbf{z}_j(k) - \mathbf{z}_i(k)}{\|\mathbf{z}_i(k) - \mathbf{z}_j(k)\|} \right)$$

Stochastic Neighbour Embedding

- Stochastic Neighbour Embedding (SNE) is very much like MDS, however instead of optimising distances, SNE attempts to optimise the distribution of the data points in the high and low dimensional spaces to be similar.
  - SNE models the distribution of points in the high dimensional space by placing Gaussians over each point (with an optimised variance for each Gaussian).
    - It defines the conditional probability that a high-dimensional point 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:

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

  - The target distribution in the low dimensional space is defined similarly (with respect to low-dimensional points y_i and y_j):

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

    - Note the distribution for the low-dimensional space assumes all Gaussians have variance 1/sqrt()
  - A standard way to quantify the difference between two distributions is the Kullback Liebler Divergence:

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

  - The SNE cost function is the sum of KL Divergences over all data points:

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

- The projection from the high dimensional space to the low dimensional space is found by minimising the cost using Stochastic Gradient Descent.
    - There are some pitfalls:
        - It's difficult to optimise
        - It leads to "crowded" visualisations in which things clump together in the centre (this is also true of Sammon mapping)
- t-distributed Stochastic Neighbour Embedding works around the problems of SNE:
    - Uses a symmetric cost function:

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

        - this makes the gradients easier to compute
    - Changes the target distribution from Gaussian to *Student's t distribution*:

$$q_{ij} = \frac{f(\|x_i - x_j\|)}{\sum_{k \neq i} f(\|x_i - x_k\|)} \quad \text{with} \quad f(z) = \frac{1}{1 + z^2}$$

        - the 'fatter tails' of the t distribution alleviate the crowding problem by better matching of the properties of the volume of Gaussian distributions in higher dimensional spaces.

Other techniques

- Examples of other and more modern approaches to non-linear dimensionality reduction include
    - ISOMAP
    - Locally Linear Embedding (LLE)
    - Principal curves

Embedding data

- Rather than using a technique to project high dimensional data in a 2D or 3D space, could we target a medium dimensionality which captures the key features?
    - Yes! This is called an embedding.
    - We'll see examples of some different embeddings in later lectures; here we'll focus on word embeddings.

Word Embeddings

- Problem: we want vector representations of words
    - We've already seen how multiple words can be encoded using a Bag of Words
        - A bag of words with a single word in it produces a One Hot Encoding; a vector in which everything is 0, with the exception of a single 1 in one element.
        - This representation has problems:
            - Independence of words is implied (all words are orthogonal in the space); this isn't however true in real languages, where *synonymous* words exist (words with similar meanings)
            - The vectors have *very* high dimensionality (although they are maximally sparse)
        - Word Embeddings aim to build vectors that:
            - capture semantic relationships between words (words with similar meanings have similar vectors)
            - Encode words in dense vectors with much lower-dimensionality (e.g. typically 300 dimensions)
            - Potentially aim to capture the semantics (and/or possibly syntax) of the

language in an algebraic fashion

        - allowing arithmetic on the language to produce meaningful results - e.g: word2vec("Brother") - word2vec("Man") + word2vec("Woman") = word2vec("Sister")
    - There are many models that create word embeddings in the literature:
        - word2vec (uses shallow neural nets to learn mappings from words to vectors)
        - GLoVe (uses matrix factorisation to learn a mapping)
        - etc...

Further Reading

- Chapter 3 of "Programming Collective Intelligence" gives a good overview of the some of the basic techniques.
- Relevant sections of Chapter 14 of The Elements of Statistical Learning (https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print10.pdf) provide a good academic introduction
- Wikipedia has reasonable commentary (and good links to the original research) on a number of the topics:
    - https://en.wikipedia.org/wiki/Multidimensional_scaling (https://en.wikipedia.org/wiki/Multidimensional_scaling)
    - https://en.wikipedia.org/wiki/Self-organizing_map (https://en.wikipedia.org/wiki/Self-organizing_map)
    - https://en.wikipedia.org/wiki/Word2vec (https://en.wikipedia.org/wiki/Word2vec)
- k-means++: the advantages of careful seeding (http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf) . Arthur and Vassilvitskii. Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics Philadelphia, PA, USA. pp. 1027–1035. 2007.
- Mean shift: A robust approach toward feature space analysis (http://www.caip.rutgers.edu/riul/research/papers/pdf/mnshft.pdf) . Comaniciu and Meer. IEEE Trans. Pattern Anal. Machine Intell., 24:603–619, 2002.
- Good descriptions of dimensionality reduction techniques and clustering:
    - Learning from Data: Concepts, Theory, and Methods (2nd ed.). Cherkassky and Mulier. John Wiley & Sons, Inc., New York, NY, USA.
- A Nonlinear Mapping for Data Structure Analysis (http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1671271) . Sammon. in Computers, IEEE Transactions on , vol.C-18, no.5, pp.401-409, May 1969
- The O'Reilly blog has an informative post on how t-SNE works: An illustrated introduction to the t-SNE algorithm (https://www.oreilly.com/learning/an-illustrated-introduction-to-the-t-sne-algorithm) .
- How to use t-SNE effectively (https://distill.pub/2016/misread-tsne/)
- Visualizing Data using t-SNE (http://jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf) . van der Maaten and Hinton. JMLR. 9 (2008) 2579-2605.
- Efficient Estimation of Word Representations in Vector Space (https://arxiv.org/pdf/1301.3781.pdf) . Mikolov, Chen, Corrado and Dean.