Notes on Dimensionality Reduction Methods

Rei Sanchez-Arias, Ph.D.

Other Dimensionality Reduction Methods

PCA review

Comments on PCA

Strengths:

- can be efficiently applied to large data sets
- has many **extensions** that deal with special situation such as sparse PCA when data has many missing values, kernel PCA can provide more robust solutions
- preserves **global structures**

Weaknesses:

- may suffer from scale problems, i.e. when one variable dominates the variance simply because it is in a higher scale (for example if one variable is measured in mm instead of km); some of these scale problems can simply be dealt with by centering and scaling.
- may be susceptible to big outliers

Multidimensional Scaling

Multidimensional Scaling (MDS)

Multidimensional scaling (MDS) is an algorithm in which given a **distance matrix** with the distances between each pair of objects in a set, and a chosen number of dimensions, N, places each object into N-dimensional space such that the **between-object distances are preserved** as well as possible.

MDS is most often used as a *visualization tool*. It is best suited to the problem that involve real distances, i.e. city distances or geometry. It yields the same result as PCA when Euclidean distances are used.

An implementation of MDS is available via the `cmdscale()` function from the `stats` package.

Notes on MDS

Strengths:

- flexible as any distance metric can be used
- preserves global structures

Weaknesses:

- computationally demanding and inefficient for large numbers of observations.
- suffers from crowding in the presence of large number of observations.

MDS example

```
# read distance data
library(tidyverse)
cities_dist <- read_csv("https://raw.githubusercontent.com/reisanar/datasets/master/cities_dis
```

This data only tells you about the *dissimilarities* between two cities but does not easily provide a way to *visualize* any **underlying structure** that is meaningful, for example the location of those cities in a map.

The goal is to use the data in the distance matrix to **build a map** of those cities. The challenge is to produce *coordinates* for each city that *best approximate the distances* in the distance table.

To learn more about a formulation of MDS as an optimization problem, check the Jupyter notebook by Cindy Nguyen and Rei Sanchez-Arias available at nanoHUB.org

Explore the dataset

	cities	vancouver	portland	new_york	miami	mexico_city	los_angeles	toronto	panama_
1	vancouver	0	76.11003381	659.9761793	696.3038864	599.673906	290.9056746	579.0356219	871.547
2	portland	76.11003381	0	639.7049827	649.0683294	533.1480917	215.9081194	563.8392681	811.585
3	new_york	659.9761793	639.7049827	0	263.5734512	476.149107	587.6310161	88.96778672	501.588
4	miami	696.3038864	649.0683294	263.5734512	0	263.7530034	512.1960576	305.0794933	245.685
5	mexico_city	599.673906	533.1480917	476.149107	263.7530034	0	337.5951037	475.2850866	295.58
6	los_angeles	290.9056746	215.9081194	587.6310161	512.1960576	337.5951037	0	533.531043	629.721
7	toronto	579.0356219	563.8392681	88.96778672	305.0794933	475.2850866	533.531043	0	550.437
8	panama_city	871.5474944	811.5857933	501.5883803	245.6856642	295.582614	629.7219623	550.4376621	
9	winnipeg	338.1000679	344.4801984	351.0465898	484.341793	527.5223303	405.7753233	262.8345183	714.613
10	montreal	648.2166343	638.3000855	89.98795989	351.3495838	547.9762329	616.7099359	83.64411016	591.27

Solution using `stats::cmdscale()`

Let us define a distance matrix (notice this is a symmetric matrix) using the `as.matrix()` R function, with the distances between the cities considered in this example.

```
# distance matrix: remove column 1 since that simply has the names of the cities
Dist_Mat <- as.matrix(cities_dist[ , -1])
```

We could use the function `cmdscale(Dist_Mat)` to apply MDS to the distance matrix. The result is a data frame with (x, y) coordinates for each of the cities in the map. Creating a **scatterplot** confirms the geographical location expected based on the name of the cities and what we know about their place in a map.

Visualization

Independent Component Analysis

Independent Component Analysis (ICA)

Independent Component Analysis (ICA) attempts to decompose a multivariate signal into **statistically independent non-Gaussian signals**.

ICA is important to blind signal separation and has many practical applications. The most common is applications for sound signals. A simple application of ICA is the *"cocktail party problem"*, where the underlying speech signals are separated from a sample data consisting of people talking simultaneously in a room. Usually the problem is simplified by assuming no time delays or echoes. Note that a filtered and delayed signal is a copy of a dependent component, and thus the statistical independence assumption is not violated.

Check the package `ica` for a collection of functions implementing ICA.

Notes on ICA

Strengths:

- separating multiple independent sources of signal.
- can be efficiently applied to large data sets.
- preserves global structures.

Weaknesses:

- has very specific assumptions, especially that none of the independent sources are normally distributed.
- suffers from crowding in the presence of large number of observations.

t-distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE

t-distributed Stochastic Neighbor Embedding (t-SNE) is a nonlinear dimension reduction technique well-suited for embedding high-dimensional data for *visualization* in a low-dimensional space of two or three dimensions.

Specifically, it models each high-dimensional object by a two- or three-dimensional point in such a way that **similar objects are modeled by nearby points** and **dissimilar objects are modeled by distant points** with high probability. t-SNE is a (prize-winning) technique for dimensionality reduction that is particularly well suited for the **visualization of high-dimensional datasets**.

`Rtsne` is a good R package for implementing t-SNE.

How to Use t-SNE Effectively
Distill interactive visualization

Laurens van der Maaten website
Journal of Machine Learning Research

Notes on t-SNE

Strengths:

- recovers well-separated clusters
- preserves local neighborhoods
- no crowding of observations

Weaknesses:

- stochastic, so unless you set a seed your results are not reproducible
- computationally expensive with extremely large number of observations
- can produce artifacts
- no preservation of global structures

Uniform Manifold Approximation and Projection (UMAP)

UMAP

Uniform Manifold Approximation and Projection (UMAP) produces a low dimensional representation of high dimensional data that *preserves relevant structure*. It searches for a low dimensional projection of the data that has the **closest possible equivalent fuzzy topological structure** to the original high dimensional data.

UMAP is similar to t-SNE but preserves more global structure in the data. Its main use is visualization of a large amount of observations as it recovers well-separated clusters.

The `umap` R package contains relevant function for implementing UMAP. `uwot` is another option.

Notes on UMAP

Strengths:

- recovers well-separated clusters
- computationally efficient
- preserves global distances

Weaknesses:

- somewhat stochastic
- parameters to optimize
- can produce artifacts

Autoencoders

Autoencoders

Autoencoders are *neural networks* that are trained to **reconstruct their original inputs**. The aim of an autoencoder is to **learn a representation** (encoding) for a set of data, typically for dimension reduction. This is achieved by designing a neural network with a bottleneck (to avoid overfitting) and training the network to ignore signal noise.

The idea of autoencoders has been popular in the field of neural networks for decades, and is commonly used for denoising pictures or sound. As with any neural network there is a lot of flexibility in how autoencoders can be constructed such as the number of hidden layers and the training strategy (i.e. variational autoencoders).

Notes on Autoencoders

Strengths:

- preserves global data structures
- new points can be added without recomputing

Weaknesses:

- large number of observations required for effective training
- computationally more expensive
- stochastic
- parameter selection, especially neural network selection and training

Final note and references

Choosing the right technique ...

- A difficulty in applying dimension reduction is that each dimension reduction technique is designed to maintain certain aspects of the original data and therefore may be appropriate for one task and inappropriate for another.

- For many applications it can be difficult to objectively find the right method or parameterization for the dimension reduction task. To find the right dimension reduction technique for your data consider the nature and resolution of your data.

- Are you interested or able to capture only small-scale relationships? or are you focusing on long-range relationships?. In the first case you may want to only consider dimension reduction techniques that preserve local structure and in the second case you should focus on dimension reduction techniques that preserve global structure.

Further Reading and References

1. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: data mining, inference, and prediction. Springer Science & Business Media.
2. Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020). Mathematics for machine learning. Cambridge University Press.
3. Maaten, L. V. D., & Hinton, G. (2008). Visualizing data using t-SNE. Journal of machine learning research, 9(Nov), 2579-2605.
4. Becht, E., McInnes, L., Healy, J., Dutertre, C. A., Kwok, I. W., Ng, L. G., & Newell, E. W. (2019). Dimensionality reduction for visualizing single-cell data using UMAP. Nature biotechnology, 37(1), 38-44.
5. Cox, M. A., & Cox, T. F. (2008). Multidimensional scaling. In Handbook of data visualization (pp. 315-347). Springer, Berlin, Heidelberg.

Videos