Topic 8: Dimensionality Reductions

8.1 PCA 167
8.2 Methods Closely Related to PCA 179
8.3 Other Classical Dimensional Recreation Methods 180
8.4 Manifold Dimensionality Reduction Methods 183
8.5 SNE and t-SNE 186

When the data has no labels, supervised learning won't work! For example, new viruses or bacterias has no 'labels'. There is no universally accepted mechanism for performing the validation of results on an independent data set [James et al., 2013, Chapter 10], i.e. **there is no performance metrics for unsupervised learning methods**.

The following are classes of unsupervised learnings where we may perform to discover the patterns of data without labels:

- **Dimensionality Reduction**: transform p-dimensional ($p > 2$) data to 2D or 3D for visual inspection of patterns or to a fraction of p dimensional data as a filter for predictive models.
- **Cluster Analysis**: k-means, hierarchical clustering, mixture models, etc.
- **Anomaly Detection**: local outlier factor, isolation forest
- **Latent Variable Models**: expectation-maximisation algorithm, methods of moments, blind source/signal separation.

PCA and t-SNE (t-Distributed Stochastic Neighbour Embedding) are the most effective dimensionality reduction methods and they are discussed in this topic, other dimensionality reduction methods will only be mentioned.

The PCA tries to find a global structure and this can lead to local inconsistencies, i.e. far away point can become nearest neighbours (due to the projection). The t-SNE tries to preserve local structure, i.e. the low dimensional neighbourhood "representation" should be the same as original neighbourhood. Note that potential competitors to t-SNE appear since 2021 such as Ivis (based on Siamese Network and Triplet Loss) and UMAP (Uniform Manifold Approximation Projection). If global structure preservation is required, Ivis is better. However, Ivis requires Keras-tensorflow library and has a lot of dependencies (too large and inefficient in many platforms) while UMAP has an implementation in R with little dependencies and an implementation in Python depending on scipy and scikit-learn.

8.1 PCA

According to Ruppert [2011, Chapter 17], **Principal component analysis** (**PCA**) finds "factors" in the covariance matrix of the data $X = [\boldsymbol{x}_i]$:

$$\text{Cov}(X) = \frac{1}{n-1}\sum_{i=1}^{n}(\boldsymbol{x}_i - \mu_{\boldsymbol{x}})^T(\boldsymbol{x}_i - \mu_{\boldsymbol{x}}) \tag{8.1}$$

and uses this structure to locate low-dimensional subspaces containing most of the variation in the data. Mathematically, the projection of the centred data, $(\boldsymbol{x}_i - \mu_{\boldsymbol{x}})V^T$, is chosen to

maximise the determinant of the transformed vectors [Belhumeur et al., 1997]

$$V_{opt} = \underset{V}{\text{argmax}} |V \,\text{Cov}(X) V^T| = [\boldsymbol{e}_1, ..., \boldsymbol{e}_p].$$

Note that the image data is stored as column in Belhumeur et al. [1997] while we store image data as row.

The data X can then be "decomposed" into "principal components" (which are eigenvectors of the covariance matrix) [Deisenroth et al., 2019, Chapter 10].

PCA may smear the classes of high-dimensional data together until they are no longer linearly separable in the projected space.

Given an $n \times p$ data set x_{ij}, $i = 1, \cdots, n$, $j = 1, \cdots, p$. In PCA, we are interested in the **variance**, therefore, we will centre the data $x_{\cdot j}$ to have a mean of zero $x_{\cdot j} \to x_{\cdot j} - \overline{x}_j$ and denote the "zero mean" data as

$$\widetilde{X} = [x_{\cdot j} - \overline{x}_j] = (\tilde{X}_1, \cdots, \tilde{X}_p).$$

This is a standard operation in R and Python:

```
R      : X = scale(X, scale=FALSE)  # X = sweep(X, 2, colMeans(X))
Python: X = X - np.mean(X,axis=0)
```

We then look for the linear combination of the sample feature such that it is maximised, i.e.

$$\max_{e_{11},\cdots,e_{1p}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} e_{1j} \tilde{x}_{ij} \right)^2 \right\} \text{ subject to } \sum_{j=1}^{p} e_{1j}^2 = 1 \Rightarrow \boxed{\boldsymbol{e}_1 = \text{argmax} \left\{ \frac{\boldsymbol{e}^T \widetilde{X}^T \widetilde{X} \boldsymbol{e}}{\boldsymbol{e}^T \boldsymbol{e}} \right\}}$$

where $\boldsymbol{e}_1 = (e_{11}, e_{12}, \cdots, e_{1p})^T$ is a "normalised" column vector and its right-hand-side fraction above is called the *Rayleigh quotient* (see `https://en.wikipedia.org/wiki/Principal_component_analysis`). The vector $\boldsymbol{e}_1$ is required to be "normalised" because a "unique" answer may be obtained (with a difference of $\pm$ sign).

A *principal component* is a linear combinations of the "featured" columns $X_1, \cdots, X_p$ in $\boldsymbol{X}$. What is amazing is that the **weighted vectors** $\boldsymbol{e}_i$, $i = 1, \cdots$, is the **eigenvectors** corresponding to the **eigenvalues** of the matrix (8.1).

The *first principal component* of a set of features X_1, X_2, $\cdots$, X_p is the **normalised linear combination** of the features that has maximum variance (among all linear combinations):

$$Z_1 = e_{11} X_1 + e_{12} X_2 + \cdots + e_{1p} X_p = \boldsymbol{X} \boldsymbol{e}_1, \quad \sum_{j=1}^{p} e_{1j}^2 = 1. \tag{8.2}$$

The kth principal component ($k \geq 2$), is the linear combination of features that accounts for as much of the remaining variation as possible, with the constraint that the correlation between the kth component and the previous $s = 1, \cdots, k-1$ components are 0:

$$Z_s = e_{s1} X_1 + e_{s2} X_2 + \cdots + e_{sp} X_p = \boldsymbol{X} \boldsymbol{e}_s, \quad \sum_{j=1}^{p} e_{sj}^2 = 1. \tag{8.3}$$

The kth component is calculated by subtracting the first $k-1$ principal components from X:

$$\hat{\mathbf{X}}_k = \boldsymbol{X} - \sum_{s=1}^{k-1} Z_s \, \boldsymbol{e}_s = \boldsymbol{X} - \sum_{s=1}^{k-1} (\boldsymbol{X} \boldsymbol{e}_s) \, \boldsymbol{e}_s$$

and then finding the weight vector $\boldsymbol{e}_k$ which extracts the maximum variance from this new data matrix:

$$\boldsymbol{e}_k = \text{argmax}\left\{\frac{\boldsymbol{e}^T\hat{\mathbf{X}}_k{}^T\hat{\mathbf{X}}_k\boldsymbol{e}}{\boldsymbol{e}^T\boldsymbol{e}}\right\}. \tag{8.4}$$

The theory of linear algebra shows that $\boldsymbol{e}_k$ corresponds to the remaining eigenvectors of $\boldsymbol{X}^T\boldsymbol{X}$ with the maximum values for the quantity in the fraction given by their corresponding eigenvalues.

In summary, when $\boxed{n \geq p}$, PCA computes the eigenvalues $\lambda_1,\ \lambda_2,\ \cdots,\ \lambda_p$ of the sample variance-covariance matrix (8.1) and the corresponding eigenvectors $\boldsymbol{e}_1, \boldsymbol{e}_2, \cdots, \boldsymbol{e}_p$. The principal components of a data $\boldsymbol{x}$ is computed using

$$\begin{aligned} PC_1(\boldsymbol{x}) &= \hat{e}_{11}(x_1-\mu_1) + \hat{e}_{12}(x_2-\mu_2) + \cdots + \hat{e}_{1p}(x_p-\mu_p),\\ PC_2(\boldsymbol{x}) &= \hat{e}_{21}(x_1-\mu_1) + \hat{e}_{22}(x_2-\mu_2) + \cdots + \hat{e}_{2p}(x_p-\mu_p),\\ &\vdots\\ PC_p(\boldsymbol{x}) &= \hat{e}_{p1}(x_1-\mu_1) + \hat{e}_{p2}(x_2-\mu_2) + \cdots + \hat{e}_{pp}(x_p-\mu_p). \end{aligned} \tag{8.5}$$

When $\boxed{n < p}$, the PCA is a bit more complex. Instead of dealing with the covariance matrix (8.1) of the $n \times p$ data X where $\boldsymbol{x}$ is $p \times 1$ matrix:

$$X^TX\boldsymbol{x} = \lambda\boldsymbol{x}, \quad \tilde{X} = X - \overline{X},$$

we work with the 'transpose' of the centred data $\tilde{X}$ where $\boldsymbol{v}$ is $n \times 1$ matrix:

$$XX^T\boldsymbol{v} = \lambda\boldsymbol{v} \tag{8.6}$$

From which we can derive

$$X^T(XX^T\boldsymbol{v}) = X^T(\lambda\boldsymbol{v}) \Rightarrow (X^TX)(X^T\boldsymbol{v}) = \lambda(X^T\boldsymbol{v})$$

which implies the eigenvectors of (8.6) to be $X^T\boldsymbol{v}$. More efficient methods for computing partial SVD for $n < p$ are (i) Lanczos algorithm; (ii) Nonlinear Iterative PArtial Least Square (NIPALS).

The standard implementations of PCA in R are:

```
prcomp(x, retx = TRUE, center = TRUE, scale. = FALSE,
  tol = NULL, rank. = NULL, ...)

# older PCA
princomp(x, cor = FALSE, scores = TRUE, covmat = NULL,
  subset = rep_len(TRUE, nrow(as.matrix(x))), fix_sign = TRUE, ...)
```

The values of `pca=prcomp(X)` include

- `pca$sdev` is the standard deviations of the principal components obtained by SVD, $\sqrt{\lambda_i}$, i.e. the square roots of the eigenvalues of the covariance matrix (8.1);
- `pca$rotation` is the matrix of variable loadings $[\boldsymbol{e}_1, ..., \boldsymbol{e}_p]$, i.e., a matrix whose columns contain the **normalised eigenvectors** (8.4). The function 'princomp' returns this in the element 'loadings';
- `pca$x` is $PC_i(\boldsymbol{x})$ for each row $\boldsymbol{x}$, i.e. (8.5).
- `pca$center`: the column means μ_j;
- `pca$scale`: NULL or the sample variance $\sqrt{\frac{\sum_i (x_j-\mu_j)^2}{n-1}}$.

The standard implementation of PCA in Python is:

```
sklearn.decomposition.PCA(n_components=None, *, copy=True, whiten=False,
  svd_solver='auto', tol=0.0, iterated_power='auto', n_oversamples=10,
  power_iteration_normalizer='auto', random_state=None)

# for large dataset:
sklearn.decomposition.IncrementalPCA(n_components=None, *, whiten=False,
  copy=True, batch_size=None)

# for large and sparse dataset:
sklearn.decomposition.SparsePCA(n_components=None, *, alpha=1,
  ridge_alpha=0.01, max_iter=1000, tol=1e-08, method='lars',
  n_jobs=None, U_init=None, V_init=None, verbose=False,
  random_state=None)

sklearn.decomposition.MiniBatchSparsePCA(n_components=None, *, alpha=1,
  ridge_alpha=0.01, max_iter=None, callback=None, batch_size=3,
  verbose=False, shuffle=True, n_jobs=None, method='lars',
  random_state=None, tol=0.001, max_no_improvement=10)
```

Example 8.1.1. Given an unlabelled data with 15 observations and feature columns (but tabulate in horizontal form to save space). Find all the principal components.

	1	2	3	4	5	6	7	8	9	10
X_1	8.095	6.91	4.119	4.4	4.65	2.329	8.272	4.595	8.071	6.403
X_2	4.104	5.272	4.063	5.366	5.238	4.711	2.46	0.581	5.883	3.624
X_3	2.351	-2.827	-3.786	-0.261	-1.096	-1.456	-1.727	-1.292	0.938	2.949

	11	12	13	14	15
X_1	4.136	7.283	2.744	4.939	4.924
X_2	3.514	-1.301	3.584	3.024	2.754
X_3	2.918	-0.738	3.866	-0.803	5.154

Solution: By using R, it is easy to find the covariance matrix:

$$cov(X) = \frac{1}{15-1}\widetilde{X}^T\widetilde{X} = \begin{bmatrix} 3.700046 & -0.441594 & -0.261248 \\ -0.441594 & 3.63697 & -0.097637 \\ -0.261248 & -0.097637 & 6.82242 \end{bmatrix}$$

and it is not difficult to find the eigenvalues of $cov(X)$ using `eigen(cov(X))`:

```
eigen() decomposition
$values
[1] 6.845300 4.105652 3.208484

$vectors
             [,1]         [,2]       [,3]
[1,] -0.08006773  0.72243802 0.6867841
[2,] -0.01930849 -0.68999103 0.7235603
[3,]  0.99660240  0.04467307 0.0691952
```

from which we can obtain the **eigenvalues**:

$$\lambda_i = 6.845301,\ 4.105652,\ 3.208484$$

and the corresponding **normalised eigenvectors**

$$\boldsymbol{e}_1 = \begin{bmatrix} -0.080068 \\ -0.019308 \\ 0.996602 \end{bmatrix}, \quad \boldsymbol{e}_2 = \begin{bmatrix} 0.722438 \\ -0.689991 \\ 0.044673 \end{bmatrix}, \quad \boldsymbol{e}_3 = \begin{bmatrix} 0.686784 \\ 0.723560 \\ 0.069195 \end{bmatrix}$$

We can compare them to R's `prcomp` which prints the following summary:

```
Standard deviations (1, .., p=3):
[1] 2.616353 2.026241 1.791224

Rotation (n x k) = (3 x 3):
            PC1          PC2         PC3
X1 -0.08006773  0.72243802 -0.6867841
X2 -0.01930849 -0.68999103 -0.7235603
X3  0.99660240  0.04467307 -0.0691952
```

Note that $2.616353^2 \approx \lambda_1$ (theoretically they should be the same), etc. as well as the possible opposite sign in the normalised eigenvectors.

The first, second and third principal components of X are:

$PC_1(\boldsymbol{x}) = -0.080068(x_1 - 5.458) - 0.019308(x_2 - 3.525133) + 0.996602(x_3 - 0.279333)$

$PC_2(\boldsymbol{x}) = 0.722438(x_1 - 5.458) - 0.689991(x_2 - 3.525133) + 0.044673(x_3 - 0.279333)$

$PC_3(\boldsymbol{x}) = 0.686784(x_1 - 5.458) + 0.723560(x_2 - 3.525133) + 0.069195(x_3 - 0.279333)$

Note that PC((8.095, 4.104, 2.351)) = (1.8423, 1.5982, 2.3732) which is different from `pca$x[1,]` by a sign.

Example 8.1.2 (Final Exam Jan 2019, Q1(c))**.** You are given the following information:

- The data set consists of 3000 observations and 2 predictors, X_1 and X_2.
- The covariance matrix, C of X_1 and X_2 is $C = \begin{bmatrix} 2.0 & 0.8 \\ 0.8 & 0.6 \end{bmatrix}$.

(i) Compute the eigenvalues, λ_1 and λ_2. For each of the eigenvalues computed, find the eigenvectors, $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$. (7 marks)

Solution:

$$|C - \lambda I| = \begin{vmatrix} 2.0 - \lambda & 0.8 \\ 0.8 & 0.6 - \lambda \end{vmatrix} = \lambda^2 - 2.6\lambda + 0.56 = 0 \Rightarrow \lambda = 2.363015,\ 0.236985$$

[2 marks]

$\lambda = \lambda_1 = 2.363015$ (eigenvalue of PC1):

$$\begin{bmatrix} 2.0 - 2.363015 & 0.8 \\ 0.8 & 0.6 - 2.363015 \end{bmatrix} \boldsymbol{e}_1 = \boldsymbol{0}$$

$$\Rightarrow \boldsymbol{e}_1 = \frac{1}{\sqrt{0.8^2 + (0.363015)^2}} \begin{bmatrix} 0.8 \\ 0.363015 \end{bmatrix} = \begin{bmatrix} 0.910633 \\ 0.413217 \end{bmatrix}$$

[2.5 marks]

$\lambda = \lambda_2 = 0.236985$ (eigenvalue of PC2):

$$\begin{bmatrix} 2.0 - 0.236985 & 0.8 \\ 0.8 & 0.6 - 0.236985 \end{bmatrix} \boldsymbol{e}_2 = \boldsymbol{0}$$

$$\Rightarrow \boldsymbol{e}_2 = \frac{1}{\sqrt{0.8^2 + (-1.763015)^2}} \begin{bmatrix} 0.8 \\ -1.763015 \end{bmatrix} = \begin{bmatrix} 0.413216 \\ -0.910633 \end{bmatrix}$$

[2.5 marks]

(ii) First principal component, PC1 is the linear combination of predictors that has the maximum variance among all principal components. Based on your answer in (i), write the equation of first principal component for this data set. (2 marks)

Solution: PC1 is the linear combination of predictors with highest eigenvalue $\lambda_1 = 2.363015$, i.e. $PC_1(\boldsymbol{x}) = 0.910633(x_1 - \mu_1) + 0.413217(x_2 - \mu_2)$.

Example 8.1.3 (May 2022 Semester Final Exam, Q5(b))**.** Given the two-dimensional data in Table 5.2.

x_1	x_2
5.1	9.8
4.1	9.6
3.2	8.2
6.2	10.4
5.7	11.0
3.8	9.5

Table 5.2: Two-dimensional data.

Find the eigenvalues and normalised eigenvectors of the covariance matrix of the two-dimensional data and write down the principal components of the data in Table 5.2. (8 marks)

Under the PCA model and assuming the eigenvalues are in a **descending order**, the kth **eigenvalues** for data X with $\boxed{n > p}$ can be regarded as the proportion of the total variation. This allows us to define the ***proportion of variance explained***, PVE_k:

$$PVE_k = \frac{\lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_p}, \quad k = 1, 2, \cdots . \tag{8.7}$$

The first k principal components would explain the variances related to the cummulative sum of the first k proportion of variances explained, leading to the ***cumulative proportion of variance explained***:

$$CPVE_k = \frac{\lambda_1 + \cdots + \lambda_k}{\lambda_1 + \lambda_2 + \cdots + \lambda_p}, \quad k = 1, 2, \cdots . \tag{8.8}$$

If we believe the data has 10% noise, we may choose the smallest k such that

$$CPVE_k \geq 1 - 0.1 = 0.9$$

to 'characterise' the typical variation of the data.

If we plot the descending eigenvalues λ_i against the index i, we will obtain a `https://en.wikipedia.org/wiki/Scree_plot` which may allow us to identify the 'noise' using the elbow method which we will explore in the following example.

Example 8.1.4 (Places Rated Almanac Dataset)**.** Use PCA to analyse the dataset (`https://www.openml.org/d/509`) from the guide "Places Rated Almanac", by Richard Boyer and David Savageau, copyrighted and published by Rand McNally with nine rating criteria: (1) Climate and Terrain, (2) Housing, (3) Health Care & Environment, (4) Crime, (5) Transportation, (6) Education, (7) The Arts, (8) Recreation, (9) Economics.

Note that within the dataset, except for housing and crime, the higher the score the better. For housing and crime, the lower the score the better. Where some communities might rate better in the arts, other communities might rate better in other areas such as having a lower crime rate and good educational opportunities.

Solution: By reading and processing the data with PCA in R as follows:

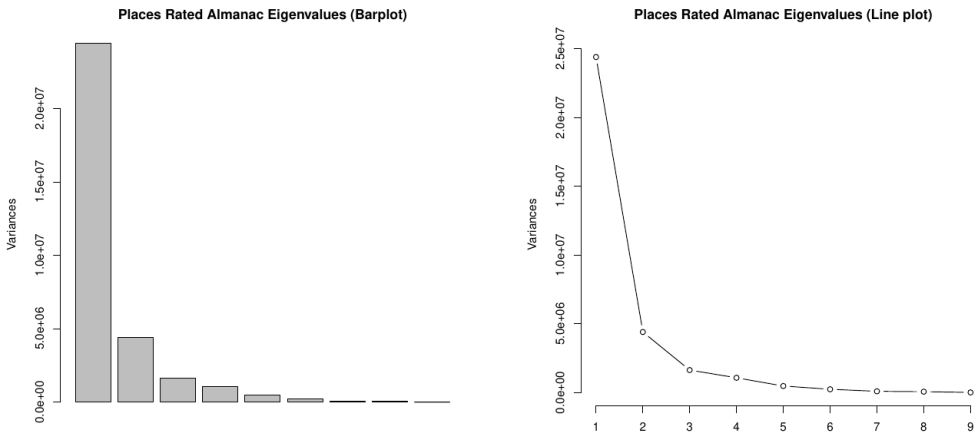
```
X = read.csv("places.arff", skip=91, header=F) # https://www.openml.org/d/509
X = X[,-10]   # Remove last column 'Place'
names(X) = c(
  "Climate+Terrain", "Housing",        "HealthCare+Environment",
  "Crime",           "Transportation", "Education",
  "TheArts",         "Recreation",     "Economics")
print(data.frame(min=sapply(X,min),mean=sapply(X,mean),
                 stdev=sapply(X,sd),max=sapply(X,max)))
pca = prcomp(X)
print(pca)
print(summary(pca))
```

we obtain the following summary:

```
                        min      mean     stdev   max
Climate+Terrain         105  538.7325  120.8083   910
Housing                5159 8346.5593 2385.2626 23640
```

```
HealthCare+Environment   43 1185.7386 1003.0020  7850
Crime                   308  961.0547  357.1542  2498
Transportation         1145 4210.0821 1451.1792  8625
Education              1701 2814.8875  320.7930  3781
TheArts                  52 3150.8845 4642.2837 56745
Recreation              300 1845.9574  807.8882  4800
Economics              3045 5525.3647 1084.4685  9980
Standard deviations (1, .., p=9):
[1] 4941.0190 2099.5249 1279.8592 1037.4757  691.6200  490.7665  304.6472
[8]  258.8357  104.7026

Rotation (n x k) = (9 x 9):
                                PC1           PC2           PC3          PC4
Climate+Terrain         0.006416346  0.015459527  0.006692298 -0.02631066
Housing                 0.269142181  0.937207188  0.082641934 -0.17775057
HealthCare+Environment  0.178318724 -0.020539870 -0.027761041 -0.02656157
Crime                   0.028134276 -0.010901921 -0.037610931  0.09903536
Transportation          0.149302463  0.018757344 -0.971531831 -0.03839697
Education               0.025190912 -0.001395877 -0.041507669  0.02163938
TheArts                 0.930859522 -0.282260587  0.151026851  0.02775471
Recreation              0.069824043  0.103848215 -0.149571984  0.06903276
Economics               0.025130829  0.173359958 -0.012743344  0.97453606
                                PC5           PC6          PC7          PC8
Climate+Terrain         -0.016278231  0.001186617 -0.08140848 -0.04213801
Housing                  0.083842278  0.048638182 -0.02668780 -0.01211847
HealthCare+Environment   0.159075722 -0.929492918 -0.13706121  0.24135975
Crime                   -0.116013534  0.053976191 -0.94477955 -0.26682693
Transportation           0.146649668  0.092235051  0.01354542  0.04150769
Education                0.106255968 -0.253188491  0.24115526 -0.92915944
TheArts                 -0.008673762  0.167554494  0.04296041 -0.01594931
Recreation              -0.954262248 -0.173348306  0.12711706 -0.01878071
Economics                0.102240592 -0.005152175  0.07016097  0.05439799
                                PC9
Climate+Terrain          0.9951449417
Housing                 -0.0229330011
HealthCare+Environment   0.0013718748
Crime                   -0.0876894940
Transportation           0.0094188168
Education               -0.0168655619
TheArts                  0.0005985854
Recreation              -0.0050315892
Economics                0.0327178331
Importance of components:
                             PC1        PC2        PC3        PC4        PC5
Standard deviation     4941.0190 2099.5249 1.280e+03 1.037e+03 691.62003
Proportion of Variance    0.7529    0.1359 5.052e-02 3.319e-02   0.01475
Cumulative Proportion     0.7529    0.8888 9.394e-01 9.726e-01   0.98731
                              PC6        PC7        PC8        PC9
Standard deviation     490.76654 304.64724 258.83567 104.70256
Proportion of Variance   0.00743   0.00286   0.00207   0.00034
Cumulative Proportion    0.99473   0.99760   0.99966   1.00000
```

The first two principle components already explain more than 88% of the variation in the data and this can be 'confirm' with the "scree plot" in either "bar chart" form or the "line plot" form:



At the end of an analysis, we usually need to provide some sort of interpretation. The interpretation of the principal components is based on ***finding which variables are most strongly correlated with each component***, i.e., which of these correlations are large in magnitude, the farthest from zero in either direction. The consideration of a strong correlation is rather subjective, and is determined at what level the correlation is important. As a rule of thumb, a correlation above 0.5 is deemed important.

First Principal Component - PC1

The PC1 is **strongly correlated** with the original variable `TheArts`. Based on the correlation of 0.93086, the first principal component is primarily a measure of `TheArts` and it has $0.93086 \times PVE_1 = 70\%$ influence. The next significant feature in PC1 is the `Housing` which has the same sign as `TheArts`. This may mean that `TheArts` can help boost the `Housing` price.

Second Principal Component - PC2

The PC2 is **dominated** by `Housing` with 12.7% ($0.93721 \times PVE_2$) significance. However, in PC2, `Housing` and `TheArts` have opposite signs which may mean that when the `Housing` price is high, the `TheArts` may be negatively affected.

Third Principal Component - PC3

The PC3 is dominated by `Transportation`. This may suggests that `Transportation` has 4.9% ($0.97153 \times PVE_3$) significance in the influence on people about the places.

Example 8.1.5 (Final Exam May 2019, Q1(c))**.** An investigation is carried out to examine the eating habit of citizens in each state in Malaysia. A total of 8 types of food had been investigated — the average consumption of each type of food (grams) per person per week in each state in Malaysia was recorded. The 8 types of food are listed below:

- x_1 = Fish
- x_2 = Meat
- x_3 = Grain
- x_4 = Dairy
- x_5 = Beans
- x_6 = Egg
- x_7 = Vegetables
- x_8 = Fruit

For advanced analysis on the eating habit, the analyst would like to reduce the dimension of the data. Therefore, principal component analysis has been performed. Eigenvalues computed from the principal component analysis are

$$\lambda = [0.0342, 0.6432, 2.3664, 0.5869, 1.1894, 0.0032, 5.6379, 0.0179]^T$$

(i) State the variance explained by each principal component. With a targeted cumulative proportion of variance explained (CPVE) of 85%, state the number of principal components to be considered.

(4 marks)

Solution: Variance explained = eigenvalue (Highest for PC1 in descending order)

PC	λ	PVE	CPVE
1	5.6379	0.5380	0.5380
2	2.3664	0.2258	0.7638
3	1.1894	0.1135	0.8773
4	0.6432	0.0614	0.9387
5	0.5869	0.0560	0.9947
6	0.0342	0.0033	0.9980
7	0.0179	0.0017	0.9997
8	0.0032	0.0003	1.0000
Total	10.4791		

With targeted CPVE of 85%, 3 principal components (PC1, PC2 and PC3), with CPVE of 87.73%, should be considered.

(ii) Given that the eigenvector for PC1 is

$$\boldsymbol{e}_1 = \begin{bmatrix} 0.174 & -0.626 & -0.146 & 0.667 & -0.249 & 0.743 & 0.598 & 0.484 \end{bmatrix}.$$

Interpret the principal component results with respect to the types of food. You should provide explanation on which types of food are the most and the least contributed to PC1, as well as the correlation between PC1 and that variable. (5 marks)

Solution: Note that the eigenvector may have some problem with the values or the lecturer set the exam question did not normalise the eigenvector.

Based on the given values in tabular form:

Variable, x_i	Type of Food	e_{1i}
x_1	Fish	0.174
x_2	Meat	-0.626
x_3	Grain	-0.146
x_4	Dairy	**0.667**
x_5	Beans	-0.249
x_6	Egg	**0.743**
x_7	Vegetables	0.598
x_8	Fruit	0.484

we find that `Egg` and `Dairy` dominate PC1. This may indicate that for 53.80% of Malaysians has the `Egg` and `Dairy` consumptions as an important part of food consumption while the opposite sign for `Meat` (third dominating feature) may indicate that they have low meat consumption.

When $n > p$, PVE is useful and PCA is a dimensional reduction because PC1, PC2 (and occassionally PC3) can explain the main variations in the data and we can use them to "visually inspect" the data. A ***biplot*** is a 2D "scatter plot" of PC1 and PC2 for a data X.

When $\boxed{n < p}$, the PVEs no longer explain the variation of the data but the biplot provides a purely feature dimension reduction from a mathematical point of view. Let $\widetilde{X}$ be the centred data of the original $n \times p$ ($n < p$) data X. Then eigenvalues and the eigenvectors are calculated as follows:

- Instead of working with $\widetilde{X}^T\widetilde{X}\boldsymbol{e} = \lambda\boldsymbol{e}$ earlier since p is large, we work with an $n \times n$ matrix:

$$\widetilde{X}\widetilde{X}^T\boldsymbol{f} = \lambda\boldsymbol{f} \Rightarrow \widetilde{X}^T(\widetilde{X}\widetilde{X}^T\boldsymbol{f}) = \widetilde{X}^T(\lambda\boldsymbol{f}) \Leftrightarrow (\widetilde{X}^T\widetilde{X})(\widetilde{X}^T\boldsymbol{f}) = \lambda(\widetilde{X}^T\boldsymbol{f}).$$

- The above equation indicates that the eigenvalues of the covariance matrix $\widetilde{X}^T\widetilde{X}$ and the complement-covariance matrix $\widetilde{X}\widetilde{X}^T$ are the same for λ_i, $i = 1, \cdots, n$ for $n < p$. Therefore, we find the eigenvalues and corresponding eigenvectors of the complement-covariance matrix

$$\widetilde{X}\widetilde{X}^T\boldsymbol{f} = \lambda\boldsymbol{f}$$

and obtain the eigenvalues and corresponding eigenvectors for the PCA of X as

$$\lambda_i, \quad \widetilde{X}^T\boldsymbol{f}_i, \quad i = 1, \cdots, n$$

Example 8.1.6 (EATING IN THE UK dataset)**.** Given the table of the average consumption of 17 types of food in grams per person per week for every country in the UK from `http://setosa.io/ev/principal-component-analysis/`.

	England	Wales	Scotland	N.Ireland
Cheese	105	103	103	66
Carcass meat	245	227	242	267
Other meat	685	803	750	586
Fish	147	160	122	93
Fresh potatoes	720	874	566	1033
Processed potatoes	198	203	220	187
Fats and oils	193	235	184	209
Sugars	156	175	147	139
Fresh Veg	253	265	171	143
Other Veg	488	570	418	355
Processed Veg	360	365	337	334
Fresh fruit	1102	1137	957	674
Cereals	1472	1582	1462	1494
Beverages	57	73	53	47
Soft drinks	1374	1256	1572	1506
Alcoholic drinks	375	475	458	135
Confectionery	54	64	62	41

Analyse the given tabular data with PCA and biplot.

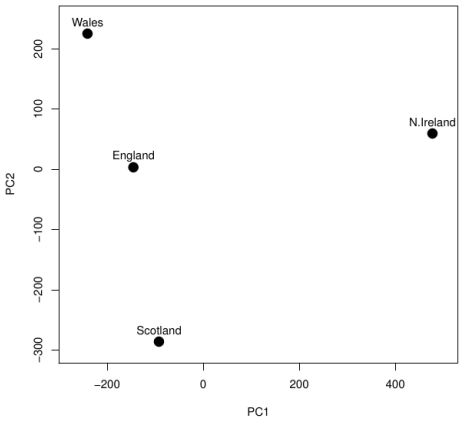
Solution: A script to analyse the data using PCA and produce a biplot is listed below.

```
# From https://bioboot.github.io/bggn213_f17/class-material/UK_food_pca/
X = t(read.csv("UK_foods.csv", row.names=1))
### Print x as a nice table (an alternative is R's View() function)
#knitr::kable(x, caption="The full UK foods data table")
pca = prcomp(X)
print(pca)
print(eigen(cov(X)))
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2",
  xlim=c(-300,550), ylim=c(-250,350), pch=16, cex=2)
text(pca$x[,1], pca$x[,2]+19, rownames(X))
# biplot(pca)
```

The eigenvalues are shown below and it is used in the biplot to find PC1 and PC2.

```
Standard deviations (1, .., p=4):
[1] 3.241502e+02 2.127478e+02 7.387622e+01 3.175833e-14
```

```
Rotation (n x k) = (17 x 4):
                              PC1           PC2          PC3           PC4
Cheese                -0.056955380  0.016012850  0.02394295 -0.694538519
Carcass_meat           0.047927628  0.013915823  0.06367111  0.489884628
Other_meat            -0.258916658 -0.015331138 -0.55384854  0.279023718
Fish                  -0.084414983 -0.050754947  0.03906481 -0.008483145
Fats_and_oils         -0.005193623 -0.095388656 -0.12522257  0.076097502
Sugars                -0.037620983 -0.043021699 -0.03605745  0.034101334
Fresh_potatoes         0.401402060 -0.715017078 -0.20668248 -0.090972715
Fresh_Veg             -0.151849942 -0.144900268  0.21382237 -0.039901917
Other_Veg             -0.243593729 -0.225450923 -0.05332841  0.016719075
Processed_potatoes    -0.026886233  0.042850761 -0.07364902  0.030125166
Processed_Veg         -0.036488269 -0.045451802  0.05289191 -0.013969507
Fresh_fruit           -0.632640898 -0.177740743  0.40012865  0.184072217
Cereals               -0.047702858 -0.212599678 -0.35884921  0.191926714
Beverages             -0.026187756 -0.030560542 -0.04135860  0.004831876
Soft_drinks            0.232244140  0.555124311 -0.16942648  0.103508492
Alcoholic_drinks      -0.463968168  0.113536523 -0.49858320 -0.316290619
Confectionery         -0.029650201  0.005949921 -0.05232164  0.001847469
```



From the biplot, we detect that Northern Ireland is a major outlier. Once we go back and look at the data in the table, this makes sense: the Northern Irish eat way more grams of fresh potatoes and way fewer of fresh fruits, cheese, fish and alcoholic drinks. It's a good sign that structure we've visualised reflects a major fact of real-world geography: Northern Ireland is the only of the four countries not on the island of Great Britain.

Example 8.1.7 (`https://en.wikipedia.org/wiki/Eigenface`)**.** Face image are extremely high-dimensional when they are viewed as vectors of pixel values. For example, a 300x200 image has 60,000 dimensions. (Higher resolution images will be very slow and will take up lots of storage.

The idea behind **eigenface** is to construct a low-dimensional linear subspace that contains most of the face images possible (possibly with small errors) — dimensional reduction.

The **recognition process** with the eigenface method is to project query images into the face-space spanned by eigenfaces calculated, and to find the closest match to a face class in that face-space.

1. Given input image vector $\boldsymbol{x} \in \mathbb{R}^p$, the mean image vector from the database M, calculate the weight of the kth eigenface as:

$$w_k = V_k^T(\boldsymbol{x} - M)$$

   Then form a weight vector $W = [w_1, w_2, ..., w_k, ..., w_n]$.
2. Compare W with weight vectors W_m of images in the database. Find the Euclidean distance.

$$d = \|W - W_m\|^2$$

3. If $d < \epsilon_1$, then the mth entry in the database is a candidate of recognition.
4. If $\epsilon_1 < d < \epsilon_2$, then $\boldsymbol{x}$ may be an unknown face and can be added to the database.
5. If $d > \epsilon_2$, $\boldsymbol{x}$ is not a face image.

PCA has been very successful in a lot of dimensional reduction tasks. For example latent semantic analysis uses PCA, population structure in the genetic data from different geographical locations can be inferred using PCA etc.

8.2 Methods Closely Related to PCA

Multiple Correspondence analysis (MCA) is like a PCA for categorical features.

```
mca(df, nf = 2, abbrev = FALSE)
```

Independent component analysis (ICA) is a computational method for separating a multivariate signal into additive subcomponents. This is done by assuming that at most one subcomponent is Gaussian and that the subcomponents are statistically independent from each other. ICA is a special case of blind source separation. A common example application is the "cocktail party problem" of listening in on one person's speech in a noisy room.

ICA finds the independent components (also called factors, latent variables or sources) by maximising the statistical independence of the estimated components. We may choose one of many ways to define a proxy for independence, and this choice governs the form of the ICA algorithm. The two broadest definitions of independence for ICA are

1. The *Minimisation-of-Mutual information (MMI) family of ICA algorithms* uses measures like Kullback-Leibler Divergence and maximum entropy.
2. The *non-Gaussianity family of ICA algorithms*, motivated by the central limit theorem, uses kurtosis and negentropy.

Typical algorithms for ICA use centering (subtract the mean to create a zero mean signal), whitening (usually with the eigenvalue decomposition), and dimensionality reduction as preprocessing steps in order to simplify and reduce the complexity of the problem for the actual iterative algorithm. Whitening and dimension reduction can be achieved with PCA or SVD. Whitening ensures that all dimensions are treated equally a priori before the algorithm is run. Well-known algorithms for ICA include infomax, FastICA, JADE, and kernel-independent component analysis, among others. In general, ICA cannot identify the actual number of source signals, a uniquely correct ordering of the source signals, nor the proper scaling (including sign) of the source signals.

ICA is important to blind signal separation and has many practical applications. It is closely related to (or even a special case of) the search for a factorial code of the data, i.e., a new vector-valued representation of each data vector such that it gets uniquely encoded by the resulting code vector (loss-free coding), but the code components are statistically independent.

The implementations in R and Python are respectively shown below.

```
# R's FastICA
Rdimtools::do.ica(X, ndim = 2, type = "logcosh", tpar = 1,
  sym = FALSE, tol = 1e-06, redundancy = TRUE, maxiter = 100)
# type = "logcosh", "exp", "poly"

# Python
sklearn.decomposition.FastICA(n_components=None, *, algorithm='parallel',
  whiten='warn', fun='logcosh', fun_args=None, max_iter=200,
  tol=0.0001, w_init=None, whiten_solver='svd', random_state=None)
```

Factor analysis (FA) is a simple linear generative model with Gaussian latent variables:

$$X - M = FL + \epsilon \sim Normal(\mu, WW^T + \Psi)$$

where X is the $n \times p$ data, M is the $n \times p$ column mean matrix, L is the $k \times p$ loading matrix, F is the $n \times k$ factor matrix and ϵ is the $n \times p$ error term matrix.

When $W = 0$, we have the PCA, $X \sim Normal(\mu, \Psi)$.

```
# R
Rdimtools::do.fa(X, ndim = 2, maxiter=10, tolerance=1e-8)

# Python
sklearn.decomposition.FactorAnalysis(n_components=None, *, tol=0.01,
  copy=True, max_iter=1000, noise_variance_init=None, random_state=0,
  svd_method='randomized', iterated_power=3, rotation=None)
```

Kernel PCA is an extension of PCA using techniques of kernel methods, i.e. generalising $\frac{n-1}{n}\text{Cov}(X) = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}_i - \mu)(\boldsymbol{x}_i - \mu)^T$ to $\frac{1}{n}\sum_{i=1}^{n}\Phi(\boldsymbol{x}_i - \mu)\Phi(\boldsymbol{x}_i - \mu)^T$ Using a kernel Φ, the originally linear operations of PCA are performed in a reproducing kernel Hilbert space.

```
Rdimtools::do.kpca(X, ndim = 2, preprocess = c("null", "center",
  "scale", "cscale", "whiten", "decorrelate"),
  kernel = c("gaussian", 1))   # aux.kernelcov
# do.kpca may depend on
kernlab::kpca(x, kernel = "rbfdot", kpar = list(sigma = 0.1),
         features = 0, th = 1e-4, na.action = na.omit, ...)
```

The kpca's kernel can take values from `rbfdot` (gaussian), `vanilladot` (linear kernel), `polydot` (polynomial), `laplacedot` (laplacian), `anovadot` (anova), `splinedot` (spline), `tanhdot` (sigmoid), `besseldot`. To use some of the kernels, we need to set the `kpar` appropriately.

```
# Python
sklearn.decomposition.KernelPCA(n_components=None, *, kernel='linear',
  gamma=None, degree=3, coef0=1, kernel_params=None, alpha=1.0,
  fit_inverse_transform=False, eigen_solver='auto', tol=0,
  max_iter=None, iterated_power='auto', remove_zero_eig=False,
  random_state=None, copy_X=True, n_jobs=None)
```

When `kernel` is `linear`, it is the same as PCA. When kernel is `poly`, `rbf`, `sigmoid`, `cosine`, `precomputed`, they are nonlinear.

8.3 Other Classical Dimensional Recreation Methods

Multidimensional Scaling (MDS), also known as Principal Coordinates Analysis, PCoA), is an algorithm that given a distance matrix with the distances between each pair of objects in a set, and a chosen number of dimensions, k, places each object into k-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. Manhattan distances or geometry. It yields the same result as PCA when Euclidean distances are used.

The classical (metric) MDS is linear and the R implementation is given below. The non-metric MDS is implemented in R's `MASS::isoMDS`, `vegan::metaMDS`, `smacof::mds` and is implemented in Python's `sklearn.manifold.MDS`.

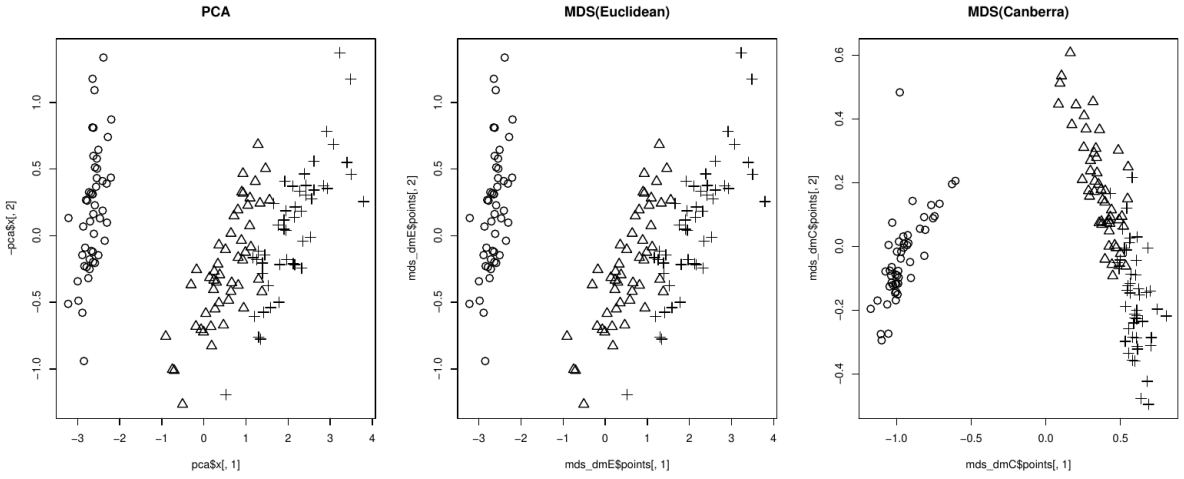
```
cmdscale(d, k = 2, eig = FALSE, add = FALSE, x.ret = FALSE,
  list. = eig || add || x.ret)

# Kruskal's Nonmetric MDS (calls cmdscale). Slow for large dataset
MASS::isoMDS(d, y = cmdscale(d, k), k = 2, maxit = 50,
  trace = TRUE, tol = 1e-3, p = 2)

Rdimtools::do.mds(X, ndim=2)

# X = unique(iris[,1:4])
```

```
# X.dist = dist(X)  # X : n x p unlabelled data
# X.mds  = isoMDS(X.dist)
# plot(X.mds$points, type="n")
# text(X.mds$points, labels=as.character(1:nrow(X)),
                     col=as.integer(iris[,5]))
```

```
sklearn.manifold.MDS(n_components=2, *, metric=True, n_init=4,
  max_iter=300, verbose=0, eps=0.001, n_jobs=None, random_state=None,
  dissimilarity='euclidean', normalized_stress='warn')
```

If `metric` is True, `MDS` performs metric MDS; otherwise, it perform nonmetric MDS (dissimilarities with 0 are considered as missing values).

Example 8.3.1. The following are the biplots for PCA and MDS of iris flower dataset.



From the example, we find the advantages of MDS to be (i) flexible as any distance metric can be used; (ii) preserves global structures.

Howver, the disadvantages of MDS are clearly (i) computationally demanding and extremely inefficient for large numbers of observations; and (ii) suffers from crowding in the presence of large number of observations.

Self-Organising Map (SOM) is a type of ANN that is trained using unsupervised learning to produce a low-dimensional (typically 2D), discretized representation of the input space of the training samples, called a map, and is therefore a method to do dimensionality reduction.

SOM differ from other ANNs as they apply competitive learning as opposed to error-correction learning (such as backpropagation with gradient descent), and in the sense that they use a neighbourhood function to preserve the topological properties of the input space.

It is available in R but the diagram is difficult to use and interpret.

```
library(class)
SOM(data, grid = somgrid(), rlen = 10000,
    alpha, radii, init)
```

The algorithm depends on `somgrid` and returns an object with `code` (containing the column names) and `grid` which is the SOM 2D representation.

```
somgrid(xdim = 8, ydim = 6,
    topo = c("rectangular", "hexagonal"))
```

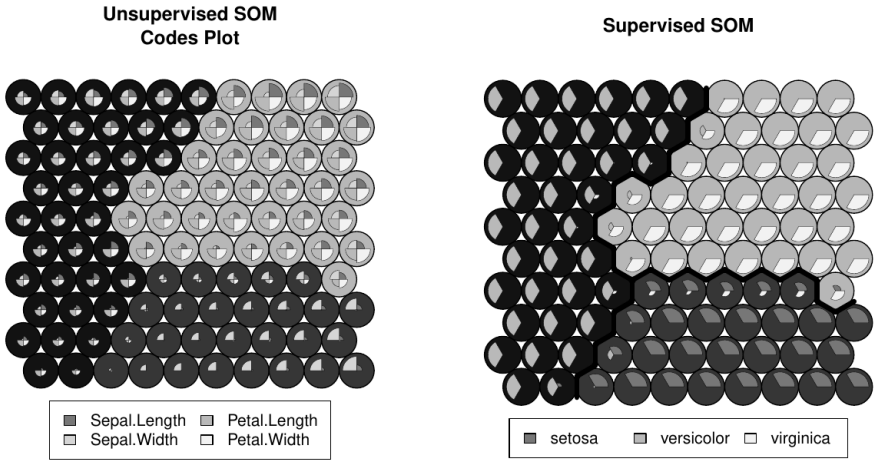
A slightly easier to use SOM is provided by `kohonen`.

```
som(X, ...)
xyf(X, Y, ...)
```

```
supersom(data, grid=somgrid(), rlen = 100, alpha = c(0.05, 0.01),
  radius = quantile(nhbrdist, 2/3), whatmap = NULL, user.weights = 1,
  maxNA.fraction = 0L, keep.data = TRUE, dist.fcts = NULL,
  mode = c("online", "batch", "pbatch"), cores = -1, init,
  normalizeDataLayers = TRUE)
```

Example 8.3.2. The SOM analysis of the iris flower data is given below.

```
# https://rpubs.com/inayatus/som
# https://rstudio-pubs-static.s3.amazonaws.com/492825_4742a19ab5464d37bd572af3792d41
library(kohonen)
d.grid = somgrid(xdim = 10, ydim = 10, topo = "hexagonal")
set.seed(100)
model = som(as.matrix(iris[,1:4]), d.grid, rlen = 500,
            radius = 2.5, keep.data = TRUE,
            dist.fcts = "euclidean")
#plot(model, type="mapping", pchs=19, shape="round")
#plot(model, type="codes", main="Codes Plot", palette.name=rainbow)
#plot(model, type="changes")

# split data
set.seed(100)
idx = sample(nrow(iris), nrow(iris)*0.8)
train = iris[ idx,]
test  = iris[-idx,]
# scaling data
trainX = scale(train[,-5])
testX  = scale(test[,-5], center = attr(trainX, "scaled:center"))
# make label
train.label = factor(train[,5])
test.label  = factor(test[,5])

cl = xyf(trainX, classvec2classmat(train.label), d.grid, rlen=500)
pred = predict(cl, newdata=list(independent=testX,
                                dependent=test.label))
table(Predict = pred$predictions[[2]], Actual = test.label)

pdf("s11_pca_som.pdf", width=10,height=5)
clust = kmeans(cl$codes[[2]], 3)
par(mfrow = c(1,2))
plot(cl, type = "codes", main = c("Unsupervised SOM\nCodes Plot",
     "Supervised SOM"), bgcol = rainbow(3)[clust$cluster])
add.cluster.boundaries(cl, clust$cluster)
```



8.4 Manifold Dimensionality Reduction Methods

Sammon mapping or Sammon projection is an algorithm that maps a high-dimensional space to a space of lower dimensionality by trying to preserve the structure of inter-point distances in high-dimensional space in the lower-dimension projection. The method was proposed by John W. Sammon in 1969.

Denote the distance between ith and jth objects in the original space by d^*_{ij}, and the distance between their projections by d_{ij}. Sammon's mapping aims to minimise the following error function, which is often referred to as Sammon's stress or Sammon's error:

$$E = \frac{1}{\sum_{i<j} d^*_{ij}} \sum_{i<j} \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}}$$

The minimisation can be performed either by gradient descent or other iterative methods. The number of iterations needs to be experimentally determined and convergent solutions are not always guaranteed.

```
# R
Rdimtools::do.sammon(X, ndim=2,
  preprocess = c("null", "center", "scale", "cscale", "decorrelate",
    "whiten"), initialize = c("pca", "random"))
```

Isometric Mapping Embedding (Isomap), an extension of MDS or Kernel PCA, is a nonlinear dimensionality reduction through isometric mapping. It guarantees to asymptotically recover the true dimensionality and geometric structure of a strictly larger class of nonlinear manifolds. Isomap is a combination of the Floyd–Warshall algorithm with classic Multidimensional Scaling MDS. Classic MDS takes a matrix of pair-wise distances between all points and computes a position for each point. Isomap assumes that the pair-wise distances are only known between neighbouring points, and uses the Floyd–Warshall algorithm to compute the pair-wise distances between all other points. This effectively estimates the full matrix of pair-wise geodesic distances between all of the points. Isomap then uses classic MDS to compute the reduced-dimensional positions of all the points. Landmark-Isomap is a variant of this algorithm that uses landmarks to increase speed, at the cost of some accuracy.

In manifold learning, the input data is assumed to be sampled from a low dimensional manifold that is embedded inside of a higher-dimensional vector space. The main intuition behind MVU is to exploit the local linearity of manifolds and create a mapping that preserves local neighbourhoods at every point of the underlying manifold.

It is implemented in R's `Rdimtools` library and in Python as `sklearn.manifold.Isomap`.

```
# R
Rdimtools::do.isomap(X, ndim=2, type=c("proportion", 0.1),
  symmetric = c("union", "intersect", "asymmetric"), weight = FALSE,
  preprocess = c("center", "scale", "cscale", "decorrelate", "whiten"))

# Python
sklearn.manifold.Isomap(*, n_neighbors=5, radius=None, n_components=2,
  eigen_solver='auto', tol=0, max_iter=None, path_method='auto',
  neighbors_algorithm='auto', n_jobs=None, metric='minkowski',
  p=2, metric_params=None)
```

Local linear embedding (LLE) is a nonlinear learning approach for generating low-dimensional neighbour-preserving representations from (unlabeled) high-dimension input. The approach was proposed by Roweis and Saul (2000). The general idea of LLE is to reconstruct the original high-dimensional data using lower-dimensional points while maintaining some geometric properties of the neighbourhoods in the original data set.

LLE consists of two major steps. The first step is for "neighbour-preserving", where each input data point $\boldsymbol{x}_i$ is reconstructed as a weighted sum of K nearest neighbour data points, and the optimal weights are found by minimising the average squared reconstruction error (i.e., difference between an input point and its reconstruction) under the constraint that the weights associated with each point sum up to one. The second step is for "dimension reduction", by looking for vectors in a lower-dimensional space that minimises the representation error using the optimized weights in the first step. Note that in the first step, the weights are optimised with fixed data, which can be solved as a least squares problem. In the second step, lower-dimensional points are optimised with fixed weights, which can be solved via sparse eigenvalue decomposition.

The reconstruction weights obtained in the first step capture the "intrinsic geometric properties" of a neighbourhood in the input data. It is assumed that original data lie on a smooth lower-dimensional manifold, and the "intrinsic geometric properties" captured by the weights of the original data are also expected to be on the manifold. This is why the same weights are used in the second step of LLE. Compared with PCA, LLE is more powerful in exploiting the underlying data structure.

```
# R
Rdimtools::do.lle(X, ndim=2, type = c("proportion", 0.1),
  symmetric="union", weight = TRUE, regtype = FALSE, regparam = 1,
  preprocess=c("null","center","scale","cscale","decorrelate","whiten"))

# Python
sklearn.manifold.LocallyLinearEmbedding(*, n_neighbors=5,
  n_components=2, reg=0.001, eigen_solver='auto', tol=1e-06,
  max_iter=100, method='standard', hessian_tol=0.0001,
  modified_tol=1e-12, neighbors_algorithm='auto',
  random_state=None, n_jobs=None)
```

When `method='standard'`, it uses the standard locally linear embedding algorithm; When `method='standard'`, it uses the Hessian eigenmap method and it requires `n_neighbors > n_components * (1 + (n_components + 1) / 2`; When `method='modified'`, it uses the modified locally linear embedding algorithm; When `method='ltsa'`, it uses local tangent space alignment algorithm.

Laplacian eigenmaps uses spectral techniques to perform dimensionality reduction. This technique relies on the basic assumption that the data lies in a low-dimensional manifold in a high-dimensional space. This algorithm cannot embed out-of-sample points, but techniques based on Reproducing kernel Hilbert space regularisation exist for adding this capability. Such techniques can be applied to other nonlinear dimensionality reduction algorithms as well.

Traditional techniques like PCA do not consider the intrinsic geometry of the data. Laplacian eigenmaps builds a graph from neighbourhood information of the data set. Each data point serves as a node on the graph and connectivity between nodes is governed by the proximity of neighbouring points (using e.g. the kNN algorithm). The graph thus generated can be considered as a discrete approximation of the low-dimensional manifold in the high-dimensional space. Minimisation of a cost function based on the graph ensures that points close to each other on the manifold are mapped close to each other in the low-dimensional space, preserving local distances. The eigenfunctions of the Laplace–Beltrami operator on the manifold serve as the embedding dimensions, since under mild conditions this operator has a countable spectrum that is a basis for square integrable functions on the manifold (compare to Fourier series on the unit circle manifold). Attempts to place Laplacian eigenmaps on solid theoretical ground have met with some success, as under certain nonrestrictive assumptions, the graph Laplacian matrix has been shown to converge to the Laplace–Beltrami operator as the number of points goes to infinity.

```
# R
Rdimtools::do.lapeig(X, ndim=2)

# Python
sklearn.manifold.SpectralEmbedding(n_components=2, *,
  affinity='nearest_neighbors', gamma=None, random_state=None,
  eigen_solver=None, eigen_tol='auto', n_neighbors=None, n_jobs=None)
```

Despite it's popularity, PCA has some obvious shortcomings, most notably is the assumption that *data lie on a linear subspace*. If the data lie along a curled plane, e.g. a swiss roll embedded in 3D Euclidean space, PCA wouldn't be able to find the 2-dimensional representation even though the data is obviously 2d.

Example 8.4.1 (3D swiss roll data)**.** We generate the swiss roll data

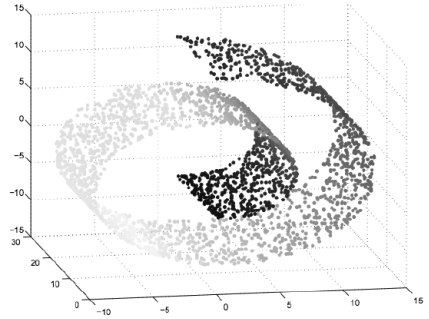


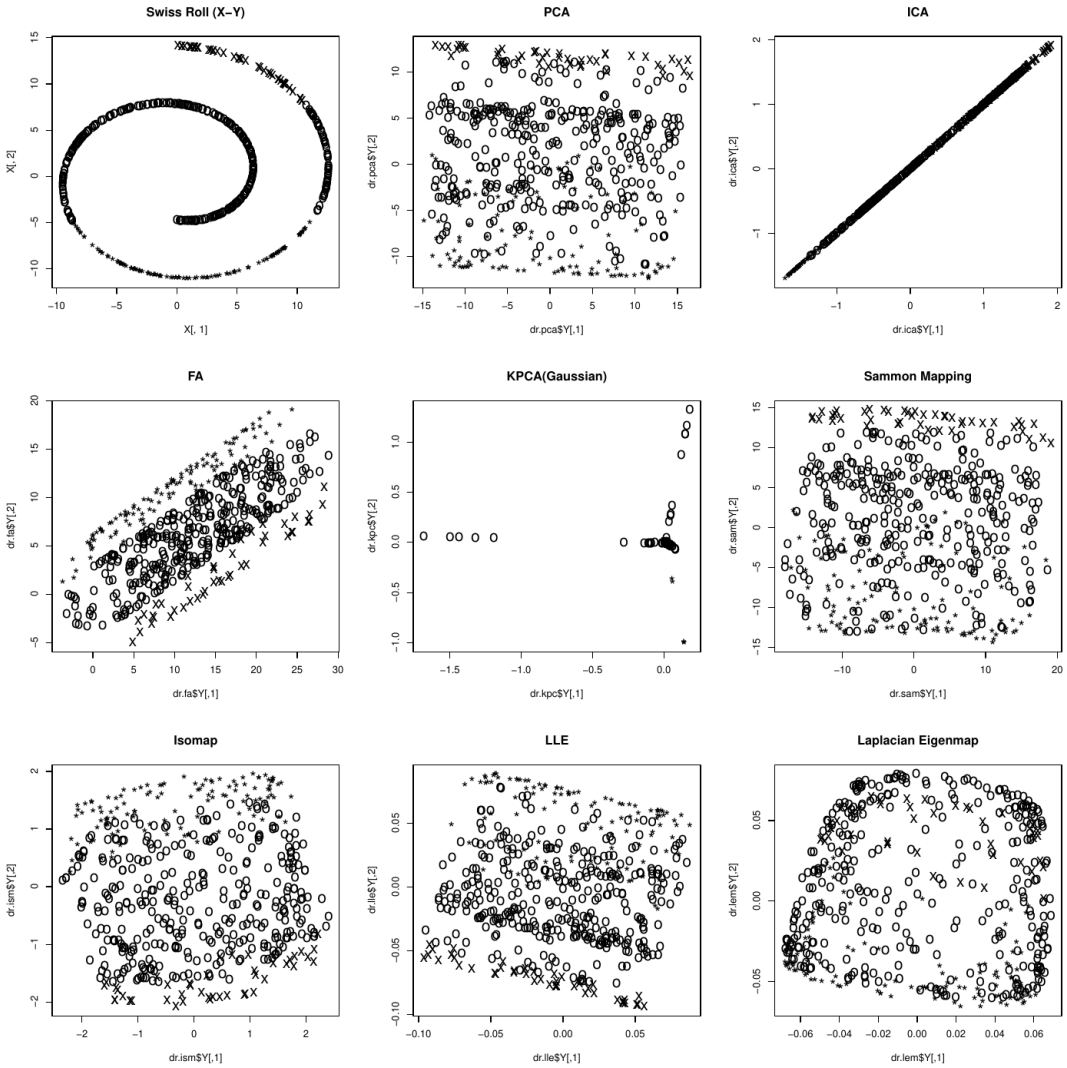
Figure 1: A curled plane: the swiss roll.

and the try to see if dimensional reduction methods can **unfold** our data to 2D.

```
library(Rdimtools)  # too many dependencies due to 'maotai' & 'CVXR'
set.seed(2023)
X = aux.gensamples(500, dname="swiss")   # 500 random 3D data

#library(rgl)
#plot3d(X)

y = cut(X[,2], c(-20,-5,8,15),label=c("*","o","x"))
# dr = Dimension Reduction
dr.pca = do.pca(X)
dr.ica = do.ica(X)
dr.fa  = do.fa(X)
dr.kpc = do.kpca(X)
dr.sam = do.sammon(X)
dr.mds = do.mds(X)
dr.ism = do.isomap(X)
dr.lle = do.lle(X)
dr.lem = do.lapeig(X)

pdf("rdimtools_eg1.pdf", width=12,height=4)
par(mfrow=c(1,3))
plot(X[,1], X[,2], main="Swiss Roll (X-Y)", pch=as.character(y), cex=1.6)
plot(dr.pca$Y, main="PCA",  pch=as.character(y), cex=1.6)
plot(dr.ica$Y, main="ICA",  pch=as.character(y), cex=1.6)
pdf("rdimtools_eg2.pdf", width=12,height=4)
par(mfrow=c(1,3))
plot(dr.fa$Y,  main="FA",   pch=as.character(y), cex=1.6)
plot(dr.kpc$Y, main="KPCA(Gaussian)", pch=as.character(y), cex=1.6)
plot(dr.sam$Y, main="Sammon Mapping",  pch=as.character(y), cex=1.6)
```

```
pdf("rdimtools_eg3.pdf", width=12,height=4)
par(mfrow=c(1,3))
plot(dr.ism$Y, main="Isomap",  pch=as.character(y), cex=1.6)
plot(dr.lle$Y, main="LLE",  pch=as.character(y), cex=1.6)
plot(dr.lem$Y, main="Laplacian Eigenmap",  pch=as.character(y), cex=1.6)
```

However, none of the dimensional reduction methods work well.



8.5 SNE and t-SNE

PCA is a linear algorithm and it cannot "project" the nonlinear relationship between features to low dimensional space well. On the other hand, *SNE (Stochastic Neighbour Embedding)* and *t-SNE (t-Distributed SNE)* use probability distributions with random walk on neighbourhood graphs to identify and try to preserve the "nonlinear" structure of the data.

SNE converts the high-dimensional Euclidean distances between data points into conditional probabilities that represent similarities, i.e.

$$\mathbb{P}(\boldsymbol{X}=\boldsymbol{x}_j|\boldsymbol{X}=\boldsymbol{x}_i)=\frac{\exp(-\frac{\|\boldsymbol{x}_i-\boldsymbol{x}_j\|^2}{2\sigma_i^2})}{\sum_{k\neq i}\exp(-\frac{\|\boldsymbol{x}_i-\boldsymbol{x}_k\|^2}{2\sigma_i^2})}=:p_{j|i}$$

where σ_i is the variance of the Gaussian that is centred on the data point $\boldsymbol{x}_i$.

Assume that the "projected" high-dimensional points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ to a low-dimension "space" is y_i and y_j respectively. The conditional probability of y_i and y_j is

$$\mathbb{P}(\boldsymbol{Y} = \boldsymbol{y}_j | \boldsymbol{Y} = \boldsymbol{y}_i) = \frac{\exp(-\|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2)}{\sum_{k \neq i} \exp(-\|\boldsymbol{y}_i - \boldsymbol{y}_k\|^2)} =: q_{j|i}. \tag{8.9}$$

Note that, we define

$$\mathbb{P}(\boldsymbol{Y} = \boldsymbol{y}_i | \boldsymbol{Y} = \boldsymbol{y}_i) = 0, \quad \text{for all } i$$

since we only want to model pair-wise similarity.

To measure the minimisation of sum of difference of conditional probability, SNE minimises the sum of *Kullback-Leibler divergences* (mentioned in Section 5.2.2) overall data points using a gradient descent method:

$$C = \sum_i \sum_j p_{j|i} \ln \frac{p_{j|i}}{q_{j|i}}. \tag{8.10}$$

In other words, the SNE cost function (8.10) focuses on retaining the local structure of the data in the map (for reasonable values of the variance of the Gaussian in the high-dimensional space, σ_i).

In contrast, t-SNE tries to minimise the sum of the difference in conditional probabilities by using a symmetric version of the SNE cost function:

$$C = \sum_i \sum_j p_{ij} \ln \frac{p_{ij}}{q_{ij}}, \quad p_{ij} := \frac{p_{j|i} + p_{i|j}}{2n}, \quad q_{ij} := \frac{q_{j|i} + q_{i|j}}{2n} \tag{8.11}$$

with simple gradients. Also, t-SNE employs a **heavy-tailed distribution in the low-dimensional space**:

$$\mathbb{P}(\boldsymbol{Y} = \boldsymbol{y}_j | \boldsymbol{Y} = \boldsymbol{y}_i) = \frac{(1 + \|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|\boldsymbol{y}_i - \boldsymbol{y}_k\|^2)^{-1}} =: q_{j|i} \tag{8.12}$$

to alleviate both the crowding problem (the area of the two-dimensional map that is available to accommodate moderately distant data points will not be nearly large enough compared with the area available to accommodate nearby data points) and the optimisation problems of SNE.

The gradient of the t-SNE cost function can be interpreted as a simulation of N-body system:

$$\frac{\partial C}{\partial \boldsymbol{x}_i} = 4 \sum_{j \neq i} \underbrace{(p_{ij} - q_{ij})(1 + \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^{-2}}_{\text{exertion / compression}} \underbrace{(\boldsymbol{x}_i - \boldsymbol{x}_j)}_{\text{spring}} \tag{8.13}$$

To simplify this, Barnes-Hut approximation is introduced, i.e. approximate similar interactions by a single interaction.

In R, t-SNE is implemented in `Rtsne::Rtsne` (based on C++, much faster) and `tsne::tsne`.

```
Rtsne(X, dims = 2, initial_dims = 50, perplexity = 30,
  theta = 0.5, check_duplicates = TRUE, pca = TRUE, partial_pca = FALSE,
  max_iter = 1000, verbose = getOption("verbose", FALSE),
  is_distance = FALSE, Y_init = NULL, pca_center = TRUE,
  pca_scale = FALSE, normalize = TRUE,
  stop_lying_iter = ifelse(is.null(Y_init), 250L, 0L),
  mom_switch_iter = ifelse(is.null(Y_init), 250L, 0L),
  momentum = 0.5, final_momentum = 0.8, eta = 200,
  exaggeration_factor = 12, num_threads = 1, ...)
```

In Python, t-SNE is implemented in scikit-learn. `sklearn.manifold.TSNE`.

```
sklearn.manifold.TSNE(n_components=2, *, perplexity=30.0,
  early_exaggeration=12.0, learning_rate='auto', n_iter=1000,
  n_iter_without_progress=300, min_grad_norm=1e-07,
  metric='euclidean', metric_params=None, init='pca', verbose=0,
  random_state=None, method='barnes_hut', angle=0.5, n_jobs=None)
```

Example 8.5.1. An R script to compare PCA and t-SNE on the MNIST data is listed below.

```
# https://www.analyticsvidhya.com/blog/2017/01/t-sne-implementation-r-python/
library(Rtsne)  # Uses Barnes-Hut-TSNE algorithm instead of the slower t-SNE
train = read.csv("mnist_train.csv")
X = train[,-1]
train$label = as.factor(train$label)
colours = rainbow(length(unique(train$label)))
names(colours) = unique(train$label)

# https://www.youtube.com/watch?v=xPBO-MMxIoQ[R Tutorial: PCA and t-SNE]
pca = prcomp(train[,-1], rank=2) # project data to first two PCs only
plot(pca$x[,1:2], pch=as.character(train$label),
                  col=colours[train$label], main="Biplot")

# Takes a long time to calculate: dim(X) = 10000 x 784
tsne = Rtsne(X, dims=2, perplexity=30, verbose=TRUE, max_iter=500)
time.taken = system.time(Rtsne(X, dims=2, perplexity=30,
                               verbose=TRUE, max_iter=500))
plot(tsne$Y, t='n', main="tsne")
text(tsne$Y, labels=train$label, col=colours[train$label])
```