Melbourne Bioinformatics

BIOINFORMATICS + DATA SERVICES + INFRASTRUCTURE, FOR LIFE SCIENCES TODAY

COMP90014

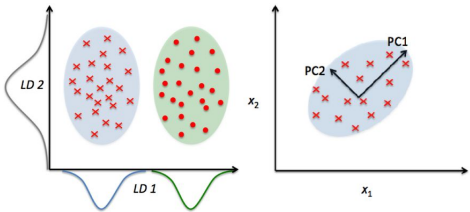
Algorithms for Bioinformatics

Week 9A: Dimensionality Reduction I

Dimensionality Reduction

1. Why do we need dimensionality reduction?
2. Approaches for dimensionality reduction
3. Principal Component Analysis (PCA)
4. Multi-dimensional scaling (MDS)

Why do we need dimensionality reduction?



Feature selection

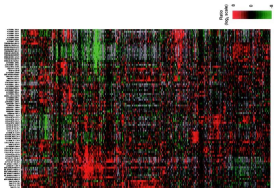
- greedy feature selection

Feature extraction

- unsupervised methods
- supervised methods
- principal component analysis (PCA)
- singular value decomposition (SVD)

Datasets have huge numbers of features (dimensions)

Documents: thousands of words
Images: thousands to millions of pixels
Genomics: thousands of genes, millions of DNA variants



e.g. a gene expression microarray

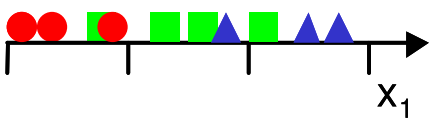
- 100 samples from *Arabidopsis thaliana*
- data from 27 000 genes
- what is the dimensionality of this dataset?
- 27 000

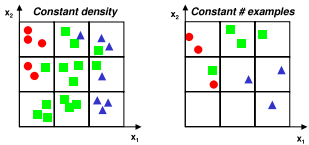
Wikimedia Commons; Zembutsu *et al.*, 2002. Cancer Res 62 (2): 518–527.

High dimensionality has many costs

- data interpretation and visualisation
- redundant and irrelevant features degrade performance of ML algorithms
- computational cost may become intractable

The curse of dimensionality





- as the number of dimensions increases the data become ***sparse***
- an huge amount of data is needed to "cover" all the dimensions
  - number of data points needed **grows exponentially with the number of dimensions**

Goal of dimensionality reduction

1. transform data from high-dimensional space into low-dimensional space
2. retain meaningful properties of the original data

Австралиец via Wikimedia Commons

Dimensionality Reduction

1. Why do we need dimensionality reduction?
2. Approaches for dimensionality reduction
3. Principal Component Analysis (PCA)
4. Multi-dimensional scaling (MDS)

Two approaches to perform dimensionality reduction

$$\mathbb{R}^N \rightarrow \mathbb{R}^M \quad (M < N)$$

- the goal is to find a low-dimensional representation of the data
- preserving (most of) the information or structure in the data

Feature selection: choosing a subset of all existing features

$$[x_1, x_2, ..., x_N] \xrightarrow{\text{feature selection}} [x_{i_1}, x_{i_2}, ..., x_{i_M}]$$

Feature extraction: creating new features by combining existing ones

$$[x_1, x_2, ..., x_N] \xrightarrow{\text{feature extraction}} [y_1, y_2, ..., y_M] = \mathrm{f}\left([x_{i_1}, x_{i_2}, ..., x_{i_m}]\right)$$

Feature selection

Forward Greedy Feature Selection

$\text{FS}^{(0)} = \varnothing$; $F^{(0)} = \{f_1, f_2, ..., f_n\}$;
$i = 0$; opt $= 0$; iter $= 0$;
while $i < n$ **do**
  $k = \text{size}\left(F^{(i)}\right)$
  max $= 0$; feature $= 0$
  for $j = 1$ **to** k **do**
    score $= \text{eval}\left(F_j^{(i)}\right)$
    if score $>$ max **then**
      max $=$ score; feature $= F_j^{(i)}$
  if max $>$ opt **then**
    opt $=$ max; iter $= i$
  $\text{FS}^{i+1} \leftarrow \text{FS}^{(i)} +$ feature
  $F^{i+1} = F^{(i)} -$ feature
  $i++$

Alogirithm: Liu & Hong, 2014. 10.1371/journal.pone.0061567

Supervised feature selection

- optimise an objective function
  e.g. F1 metric

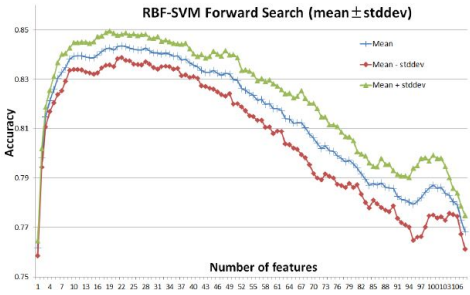
Algorithms for supervised feature selection

- Backward Greedy Feature Elimination
- Forward Greedy Feature Selection

Forward Greedy Feature Selection

1. choose the feature with the highest score
2. add feature to the pool
3. re-score other features together with feature in the pool
4. repeat the process

Feature selection



$$[x_1, x_2, \ldots, x_N] \xrightarrow{\text{feature selection}} [x_{i_1}, x_{i_2}, \ldots, x_{i_M}]$$

Time complexity

- forward / backward: $O(n^2)$
- consider evaluation time: $O(n^2 \times \text{eval})$

- might be unfeasible, if
  - you have a lot of features
  - it takes a long time to evaluate features
- stop after selecting / removing K features
- stop when objective function stops improving

Nguyen & Allebach, 2017. 10.2352/ISSN.2470-1173.2017.12.IQSP-238

Feature extraction

Define a mapping function $y = \mathrm{f}(x)$ over all features:

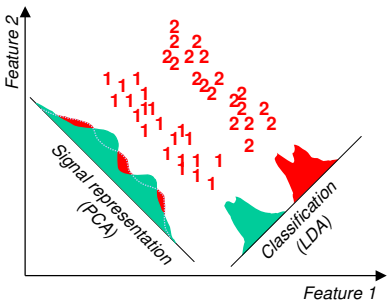
$$[x_1, x_2, ..., x_N] \xrightarrow{\text{feature extraction}} [y_1, y_2, ..., y_M] = \mathrm{f}\left([x_{i_1}, x_{i_2}, ..., x_{i_m}]\right)$$

- $y = \mathrm{f}(x)$ is, in general, non-linear and problem-dependent
- *e.g.* $\mathrm{BMI} = \dfrac{\text{weight}}{\text{height}^2}$

We usually use linear projections $y = Wx$

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \xrightarrow{\text{linear feature extraction}} \begin{bmatrix} y_1 \\ y_2 \\ \\ y_M \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & & w_{MN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

Feature extraction



Two criteria to find the best mapping function $y = \mathrm{f}(x)$

Supervised

Classification: maximize separation among classes
e.g. Linear Discriminant Analysis (LDA)

Regression: maximize correlation between projected data and target variable
e.g. Partial Least Squares (PLS)

Signal representation

Unsupervised: retain as much data variance as possible
Principal Component Analysis (PCA)
Singular Value Decomposition (SVD)

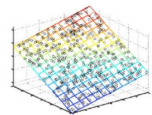
Dimensionality Reduction

1. Why do we need dimensionality reduction?
2. Approaches for dimensionality reduction
3. Principal Component Analysis (PCA)
4. Multi-dimensional scaling (MDS)

Principal Component Analysis

- widely used method for unsupervised, linear dimensionality reduction
- find a k-dimensional set of vectors (***components***) such that, if we project our data into this subspace, as much as possible of the variance is retained

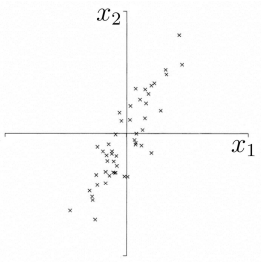
Input: points scattered in multidimensional space
Output: a more compact representation of the data

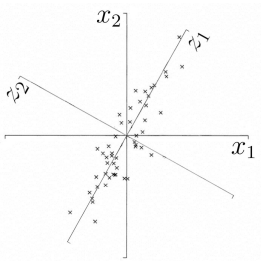


Applications:

- Data visualisation
- Data compression
- Signal processing
- Assist other learning algorithms

PCA





Mathematical procedure:

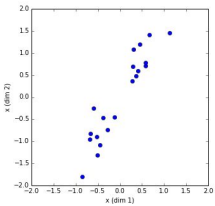
- transform a number of (possibly correlated) variables into a (smaller) number of uncorrelated variables
- the uncorrelated variables are called ***principal components***

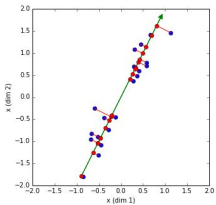
Principal components (PC):

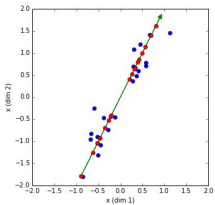
- first PC is the projection direction that maximizes the variance of the projected data
- second PC is the projection direction that is orthogonal to the first PC and maximizes variance of the projected data

- Spatial rearrangements may reveal relationships that were hidden in higher dimension space

PCA concept



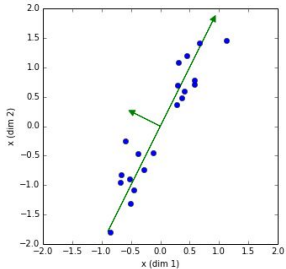




1. Find a line, such that when the data is projected onto that line, it has the **maximum variance**.
   This is the same as **minimising** the orthogonal projection error.

- Projecting the data onto the PC(s) gives us an approximate representation in fewer dimensions.
- Equivalent to rotating the data onto new axes, then discarding some dimensions.

PCA concept



2. Find another line, orthogonal to the first, that has maximum projected remaining variance.
3. Repeat until we have *k* orthogonal lines (PCs)

- The projected position of a point on the PCs is its coordinates in the *k*-dimensional space

PCs are ordered

- PC1 has the most variance
- PC2 has the second most, *etc*.

Principal components are uncorrelated

- Original data may contain correlated features
- Principal components are orthogonal

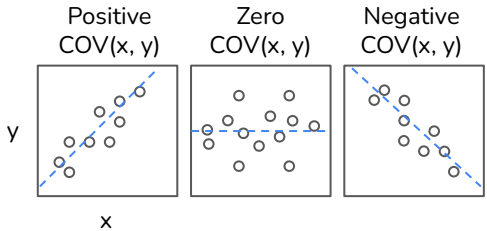
PCA Main Terminology

Covariance

- Measure of relationship between variables
- Positive = trend the same
- Zero = no relationship
- Negative = trend inversely

Linear combination

- Multiplying each variable by a scalar
- Eg. 1.2x + 3.4y + ...
- Used by PCA to define new axes which best explain the data in a smaller number of dims



PCA Main Terminology

(For a given PC)

Eigenvector

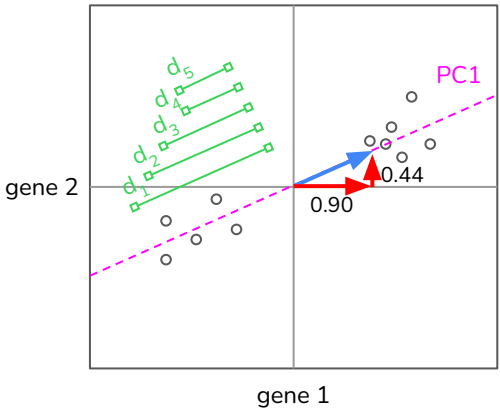
- Defines the line of best fit
- Unit vector (length = 1)
- Also known as singular vector

Loading Scores

- Combination of each variable to create the PC
- 0.9xgene1 + 0.44xgene2
- A linear combination!

Eigenvalue

- Measure of variance explained by this PC
- Average of SS(distances for PC)
- Eigenvalue = $(d_1^2 + d_2^2 + ...) / n-1$



PCA algorithm

Input: Data matrix X, integer K
Output: Projected matrix Z

1. Calculate the covariance between each dimension of the data.
2. Find ***eigenvectors*** of the covariance matrix. This:
   a. Finds basis vectors which are uncorrelated
   b. Finds ***eigenvalues***
      i. Give a measure of variance along each eigenvector
3. Choose K: how many principal components to keep
4. $Z \leftarrow$ Project data points onto K basis vectors

- Principal Components chosen are the eigenvectors with largest eigenvalues

$$\text{Cov}_{xy} = \frac{\sum (x - \bar{x})(y - \bar{y})}{n - 1}$$

$$Ax = \lambda x$$

For a matrix A, there is a vector x (eigenvector) such that the product of Ax is equal to a scalar multiplied by x

PCA Output

Output

- Data is centered
- Axes are now the PCs

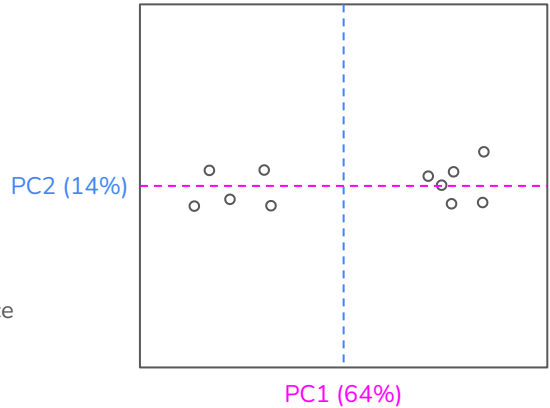
Variation explained by each PC

- Can determine using Eigenvalues
- Eigenvalues: measure of variance
- Eg `PC1=9, PC2=2, ..., Sum=14`
- Variation explained by PC1 = 9/14 = 0.64

- First 2 PCs explain 78% of the total variance
- The higher the better!

How many components will be produced?

- Min(num variables, num samples)
- Whichever is smaller



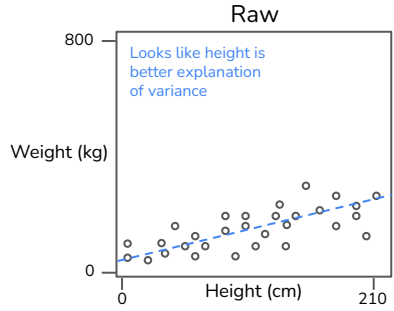
PCA Preprocessing

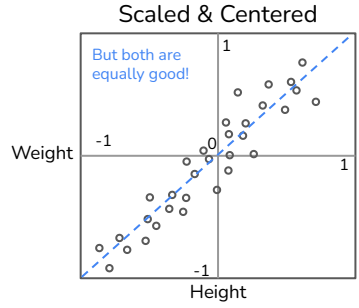
Scaling the data

- Variables should be on same scale
- RNAseq: Counts per million (CPM), Log CPM
- Log scale often useful for biological count data!

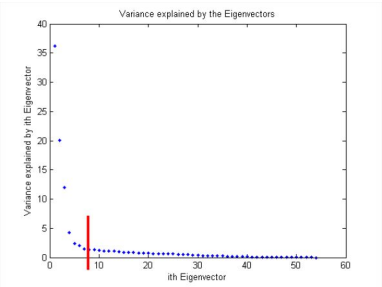
Centering the data

- PCA needs the data to be centered first
- Some libraries will do this for you, others won't
- Imagine centering about the "origin"
- Why? Conceptually, PCA finds best fit lines which go through the origin.





Using PCA



How many components to choose?

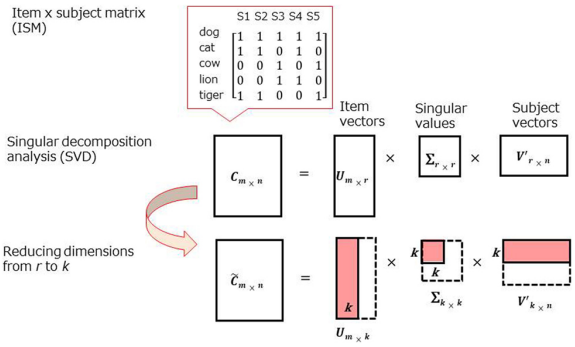
- plot the proportion of variance explained by each eigenvector
  - eigenvalue divided by the sum of eigenvalues
- *e.g.* use eigenvectors that cover at least X % of the variance

Time complexity (n dimensions and m observations)

- covariance matrix computation is $O(n^2 m)$
- eigenvalue decomposition is $O(n^3)$
- total for PCA is $O(n^2 m + n^3)$

Alternatives?

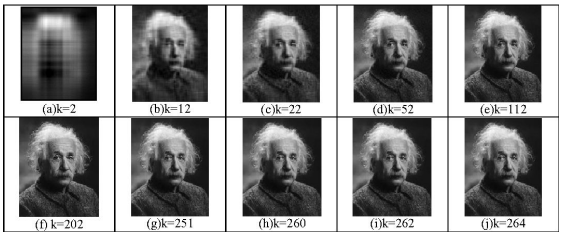
Singular Value Decomposition (SVD)



- data matrix C
- factors U, Σ and V^T
  - Σ (diagonal matrix) with singular values (σ)
  - singular values (σ) are the square roots of the eigenvalues
- U and V^T are orthogonal eigenvectors
- can use SVD to calculate PCA

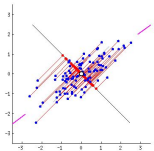
Sumiyoshi *et al*., 2018. 10.3389/fpsyt.2018.00087.

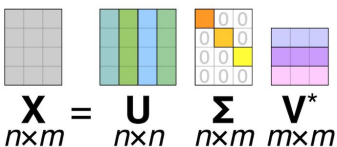
PCA and SVD



- can interpret PCA as a SVD on the centered covariance matrix
- SVD doesn't require you to calculate covariance matrix: more efficient

Linear projections: PCA and SVD





PCA: Finds orthogonal vectors (components) to represent as much variance is as possible. Decomposing the covariance matrix, $C = W\Sigma W^T$.
See this animation.

SVD: More efficient
Can interpret PCA as a SVD on the centered covariance matrix

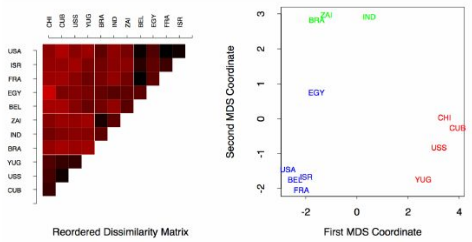
- StatQuest: Principal Component Analysis (20 minute video from Josh Starmer)

amoeba via stackexchange; Cmglee via Wikimedia Commons

Dimensionality Reduction

1. Why do we need dimensionality reduction?
2. Approaches for dimensionality reduction
3. Principal Component Analysis (PCA)
4. Multi-dimensional scaling (MDS)

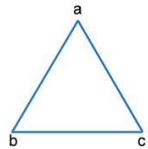
Multi-Dimensional Scaling (MDS)



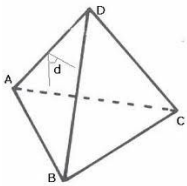
- feature extraction
- produces a lower-dimensional representation that ***preserves pairwise distances or dissimilarities***
- for visualising data
- sometimes called Principal Coordinates Analysis, but it's **not the same as PCA**!

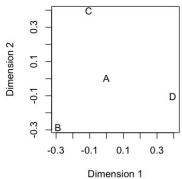
MDS

$$D(x_i, x_j) = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$



$$D(x_i, x_j) = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array}$$





- start with a pairwise distance matrix or dissimilarity matrix
- we can represent three points that are equally-spaced in 3D **exactly in 2D**
- we can represent four points that are equally-spaced in 3D **exactly in 3D** ...
- **... but not in 2D**
- in general, we need $N - 1$ dimensions to exactly represent pairwise distances between N samples

Types of MDS

Metric (distance-based) MDS

- Minimise ***stress***
- Stress is the error in the distances (lack-of-fit measure)
- Try to preserve distance between each vector x_i, x_j

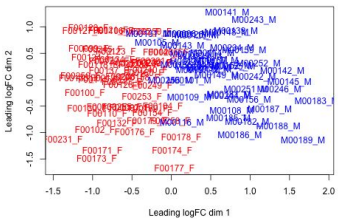
$$\text{Cost} = \sum_{i<j} \left(d_{ij} - \hat{d}_{ij} \right)^2$$

- actual cost/stress used may be more complex

Non-metric MDS

- Try to maintain original ranking of pairwise distances

Interpreting low-dimensionality visualisation



- clusters may represent real clusters
- outliers may represent real outliers
- the position of points is ***not useful*** in MDS
- resulting dimensions are not ordered by importance/variance
- can't reconstruct original data

Thank you!

Today: Dimensionality Reduction I

Next time: Dimensionality Reduction II