IAML - Week 8

Antonio León Villares

December 2021

Contents

1 Principal Component Analysis

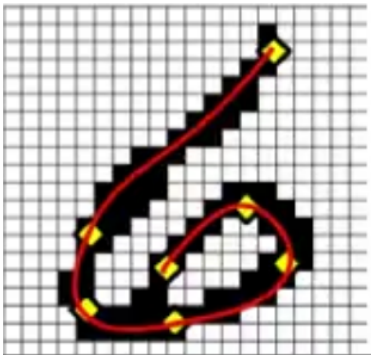
- high dimensional data often results from bad representations which don't provide the underlying cause
- PCA performs feature extraction to reduce dimensionality of data
- PCA generates new basis vectors such that projected data has highest variance
- Principal components can be used to decompose large pieces of data, by using linear combinations

We can also think of PCA in terms of reducing the reconstruction error (i.e error from going from PCA coordinates to origianl coordinates is minimised)

1.1 The Curse of Dimensionality

- **What is observed dimensionality?**
  - the phenomenon by which data, which is relatively low dimensional, seems to be of higher dimension
- **Why is data typically high dimensional**
  - data tends to be high dimensional, because of how we choose to represent it (for example, images which are represented pixel by pixel, or sentences represented word by word)
  - a clear example is considering data of 5 variables, reagrding car accidents, heat strokes, days of school missed, burst water pipes and salt expenditure. This might seem 5 dimensional, but it probably hghly dependent on a single factor: temperature
  - *the key is to understand that data can be made to be highly dimensional, when in fact its underlying structure can be low dimensional*
- **How can image analysis be dimensionally reduced?**
  - consider a 20×20 bitmap, representing numbers
  - the configurations of the bitmap which are *actual* numbers is in fact a very low proportion of the 2^{400} possible bitmap
  - think about it: if we pick any random bitmap, more likely than not it will look like:

- however, a number can be represented based on how many strokes are needed to draw it



- the strokes can be thought of as the "true dimensionality"

- **What is the curse of dimensionality?**
  - the *curse of dimensionality* is used to convey the fact that, in adding dimensions to data, the data becomes more prone to problems
  - a higher number of dimensions theoretically allow more information to be stored, but practically it rarely helps due to the higher possibility of noise and redundancy in the real-world data.
  - this can be easily seen by considering 10 points in 1, 2 and 3 dimensions:

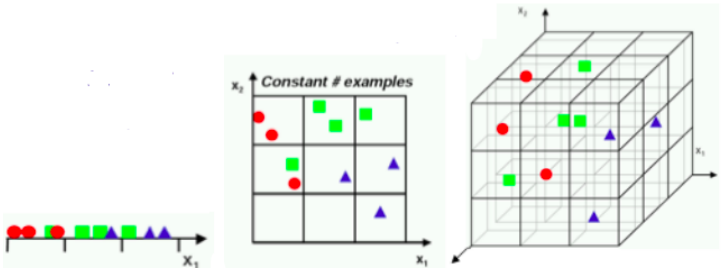


Figure 1: As we increase the dimensions, the distance between the points increases, as more space is present within the same interval. In some way, we are losing observations per unit volume

  - more on the curse:
    * https://stats.stackexchange.com/questions/99171/why-is-euclidean-distance-not-a
    * https://analyticsindiamag.com/curse-of-dimensionality-and-what-beginners-should
    * https://en.wikipedia.org/wiki/Curse_of_dimensionality

- **Why does the curse of dimensionality affect machine learning?**
  - machine learning is built on top of statistics, which is built on top of counting
  - if we have n observations in 5D space, we still have n observations in 10D space
  - however, due to the curse of dimensionality, the n observations will be more sparsely distributed in the space
  - this overall makes it harder to extract significant results, since data will seem "dilluted"
  - high dimensions also affect distance metrics: distances in hgih diemnsions will seem nearly constant

1.2 Dimensionality Reduction

- **How can we deal with high dimensional data?**
  1. **Domain Knowledge**: if we know the format of the data, we can ensure that we only focus on certain features (i.e image processing)
  2. **Dimension Assumptions**: we can increase the counts we observer, by for example assuming that the dimensions are *independent* (only consider the x axis data, and replicate it across the y axis), that the data is *smooth* (nearby regions in space should have similar distributions), or that the data is *symmetric* (if (x, y) is a data point, (y, x) should also be a data point)

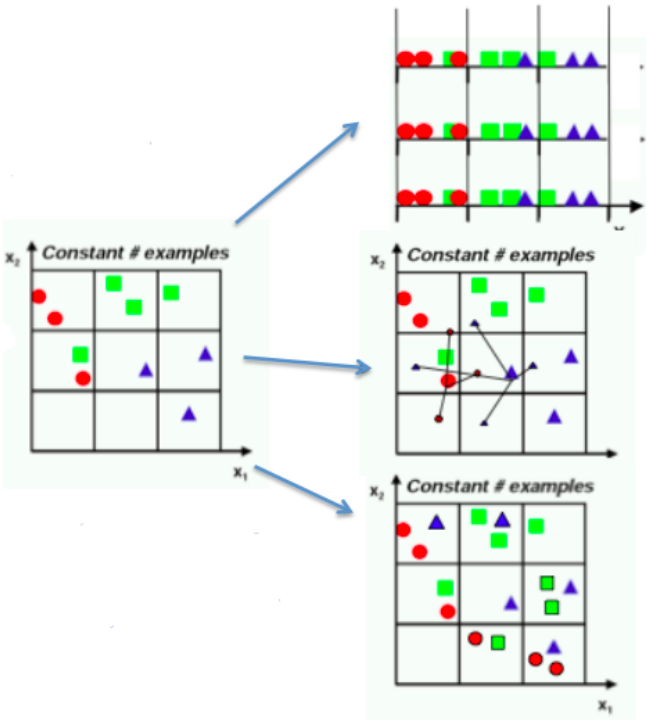


Figure 2: From top to bottom: independent dimension assumption, smoothness assumption and symmetry assumption.

3. **Dimensionality Reduction**: determine a smaller set of dimensions with which to describe the data., whilst preserving structure (depending on how we reduce dimension, "structure preservation" can have different meanings)

- **What are the 2 ways of applying dimensionality reduction?**

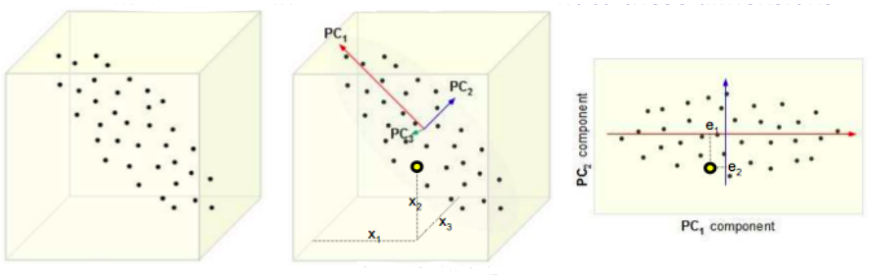
1. **Feature Selection**: use a subset of the original features. How to select the subset depends on the application. If we are interested in classification, we can use information gain as a metric
2. **Feature Extraction**: linearly combine the current features to produce a new set of dimensions.

1.3 Principal Component Analysis

1.3.1 Aim of Principal Component Analysis

- **What form of dimensionality reduction is PCA?**
  - PCA is a form of **feature extraction**

– we can think of points in a 2 dimensional plane in 3 dimensions. PCA will rotate and scale the axes, such that the basis vectors of the plane become the new axes on which the data is represented.



- **What are principal components?**
  - the principal components are the vectors which act as basis vectors for the new set of dimensions
  - PCA produces the principal components, such that:
    * the first principal component is in the direction of **greatest variance** in the data
    * the second principal component is perpendicular to PC1, and is in the direction of **greatest variance** in the **remaining data**
    * the third principal component is perpendicular to PC1 and PC2 and ...

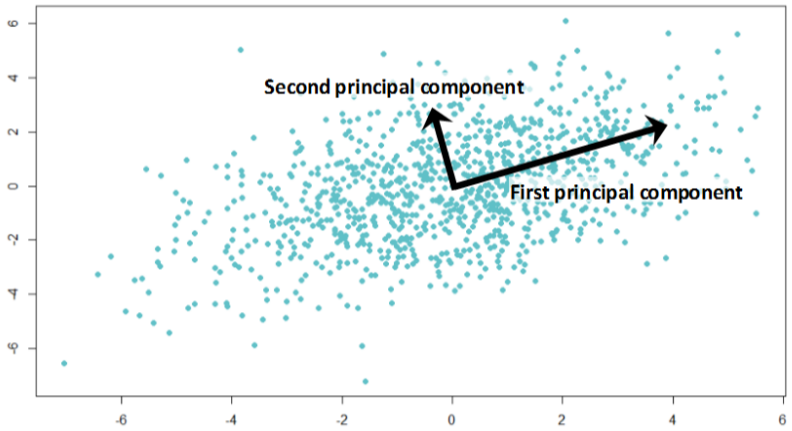


Figure 3: Notice how most of the data spreads along with the first principal component, and in less measure along the second principal component.

- **Why choose to maximise variance across principal components?**

- variance is used as a measure of structure preservation in PCA
- the idea is that by prioritising high variance, we increase the chances that data which is close in the original set of axes, is also close in the new set of axes

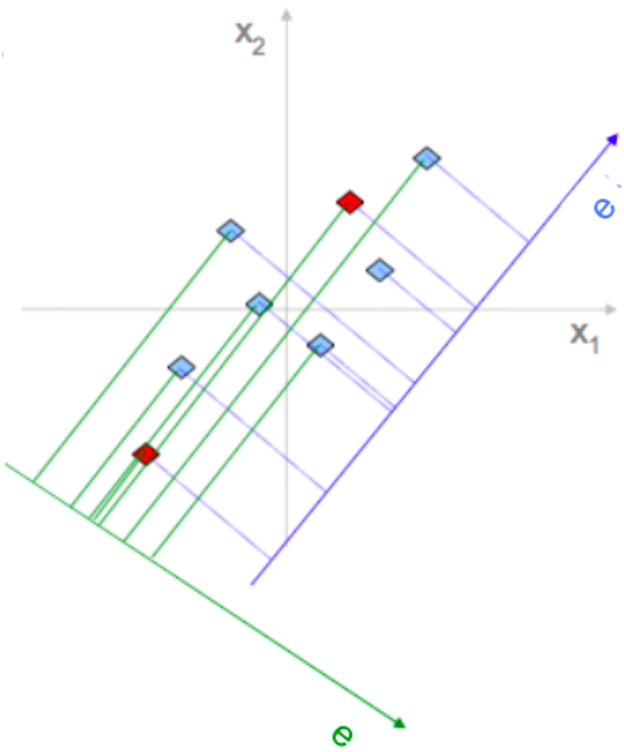


Figure 4: Consider projecting the data points on the blue and the green basis vectors. Notice, the blue vector is in the direction of highest variance (points are spread out the most in its direction). As a consequence, when we project the points onto it, the relative positions of points is preserved. However, if we were to project onto the green vector, all the data would be squished together, increasing the chances of 2 distinct data points overlapping. For example, look at the red points: they are very far away in the original data, but they would be the same in the green space.

- hence, by projecting in the direction of highest variance, we reduce the possibility of 2 distinct points being misrepresented
- alternatively, we can think of the principal components as reducing the distance between the original data, and their projection

1.3.2 Computing the Principal Components

The Covariance Matrix

If $\boldsymbol{X}$ is the matrix of the data (N rows (observations), D columns (attributes), we can compute the covariance matrix using:

$$\boldsymbol{\Sigma} = \frac{1}{N}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}$$

In particular for 2 attributes i, j, and letting $\underline{x}_k$ be the kth data point, and $\underline{x}_{k,a}$ be the the element of $\underline{x}_k$ corresponding to attribute a, then:

$$\boldsymbol{\Sigma}_{ij} = Cov(i,j) = \frac{1}{n}\sum_{k=1}^{N}(x_{k,i} - \mu_i)(x_{k,j} - \mu_j)$$

It is easier to normalise the data, such that:

$$\underline{x}_k = \underline{x}_k - \underline{\mu}, \qquad \underline{\mu} = \frac{1}{N}\sum_{k=1}^{N}\underline{x}_k$$

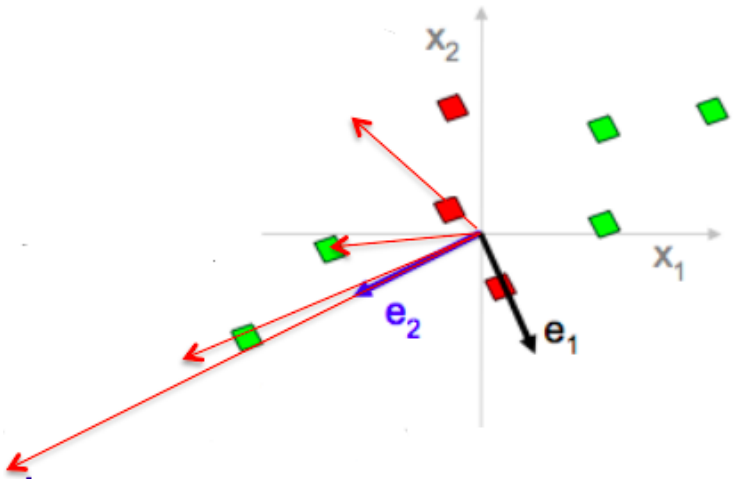
in which case:

$$\Sigma_{ij} = Cov(i,j) = \frac{1}{n}\sum_{k=1}^{N}x_{k,i}x_{k,j}$$

This tells us how much we expect i to change with j. For example, if for a point i and j have the same sign, they contribute a positive. Alternatively, if they are different, they contribute a negative amount to covariance. Hence, if $Cov(i,j) > 0$, we expect both attributes to vary similarly, and otherwise if $Cov(i,j) < 0$.

Multiplying Data by Covariance Matrix

Consider the following distribution of points:



With:

$$\mathbf{\Sigma} = \begin{pmatrix} 2 & 0.8 \\ 0.8 & 0.6 \end{pmatrix}$$

What this tells us is that there is high variance along the x_1 axis. Moreover, x_1 and x_2 have a positive covariance, so they tend to "move together" (this is represented by the green squares; the red squares are those in which for example x_1 is negative but x_2 is positive).

Now, consider applying the covariance matrix as a linear transformation to the point $(-1, 1)$:

$$\begin{pmatrix} 2 & 0.8 \\ 0.8 & 0.6 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1.2 \\ -0.2 \end{pmatrix}$$

The resulting vector is represented in the plot above as the second arrow (when moving anticlockwise). If we continuously apply the matrix to the result, we obtain the following sequence of vectors:

$$\begin{pmatrix} -2.5 \\ -1 \end{pmatrix}, \begin{pmatrix} -6 \\ -2.7 \end{pmatrix}, \begin{pmatrix} -14.1 \\ -6.4 \end{pmatrix}, \begin{pmatrix} -33.3 \\ -15.1 \end{pmatrix}$$

These are shown in the diagram above, where we can see that the vectors tend to approach $\underline{e}_2$. We note 2 things:

- the slope of the lines produced tends towards 0.454 (0.4, 0.45, 0.454, 0.454). This indicates that the vectors are eventually just getting stretched by the covariance matrix.
- $\underline{e}_2$ seems to go through the data, and in the direction of highest variance

What this indicates is that the vector $\underline{e}_2$ could be a principal component of the data. Morover, such a vector seems to act as an **eigenvector** of $\mathbf{\Sigma}$: it is just getting stretched:

$$\mathbf{\Sigma}\underline{e}_2 = \lambda \underline{e}_2$$

Determining the Eigenvector of the Covariance Matrix

We can check this hunch, by computing the eigenvalues of the covariance matrix, using:

$$det(\mathbf{\Sigma} - \boldsymbol{\lambda}\mathbb{I} = 0$$

If we perform the calculations, we get that:

$$\lambda_1, \lambda_2 = \frac{1}{2}(2.6 \pm \sqrt{2.6^2 - 4 * 0.56})$$

Notice, $\lambda_1 + \lambda_2 = 2.6$, which is the sum of variances across x_1 and x_2. Similarly, we can find the (normalised) eigenvectors as:

$$\underline{\xi}_1 = \begin{pmatrix} 0.91 \\ 0.41 \end{pmatrix} \qquad \underline{\xi}_1 = \begin{pmatrix} -0.41 \\ 0.91 \end{pmatrix}$$

(notice the slope of the first eigenvector (if we hadn't rounded) is around 0.454, as expected)

An important thing to notice is that since the covariance matrix is **symmetric**, its eigenvectors will be orthogonal.

Projecting Into a New Coordinate Space

A subspace is spanned by a set of linearly independent vectors. In particular, any point in the subspace can be written as a linear combination of these vectors. If in addition to this, the vectors are *orthogonal*, then if $\underline{v}_i$ are the basis vectors, we can write $\underline{x}$ in the subspace as:

$$\underline{x} = \sum_{i=1}^{n} c_i \underline{v}_i$$

We can find c_j by computing the dot product of the above expression with $\underline{v}_j$:

$$\begin{aligned} & \underline{x} = \sum_{i=1}^{n} c_i \underline{v}_i \\ \implies \quad & \underline{v}_j \cdot \underline{x} = \underline{v}_j \cdot \sum_{i=1}^{n} c_i \underline{v}_i \\ \implies \quad & (\underline{v}_j, \underline{x}) = \sum_{i=1}^{n} c_i (\underline{v}_j, \underline{v}_i) \\ \implies \quad & (\underline{v}_j, \underline{x}) = c_j (\underline{v}_j, \underline{v}_j) \\ \implies \quad & c_j = \frac{(\underline{v}_j, \underline{x})}{\|\underline{v}_j\|^2} \end{aligned}$$

If on top of all this the vectors are **orthonormal**, then:

$$c_j = (\underline{v}_j, \underline{x})$$

This tells us that c_j is the coordinate of the jth attribute of $\underline{x}$ in the subspace (the above formula RHS is equivalent to the vector projection of $\underline{x}$ onto $\underline{v}_j$).

We can use all of the above to determine the coordinates of data when transformed by PCA. If we have a (normalised) data point $\underline{x} \in \mathbb{R}^D$, and we

have m principal components (which are orthonormal), given by $\underline{e}_1, \underline{e}_2, \ldots, \underline{e}_m$, then define:

$$\boldsymbol{E} = \left(\underline{e}_1, \underline{e}_2, \ldots, \underline{e}_m\right)$$

notice, since each principal component is an eigenvector of the covariance matrix, each $\underline{e}_i \in \mathbb{R}^D$, so $\boldsymbol{E} \in \mathbb{R}^{D \times m}$. We can then project to the new space via:

$$\underline{x}_{proj} = \boldsymbol{E}^T \underline{x} = \begin{pmatrix} \underline{e}_1^T \\ \underline{e}_2^T \\ \vdots \\ \underline{e}_m^T \end{pmatrix} \underline{x} = \begin{pmatrix} (\underline{e}_1, \underline{x}) \\ (\underline{e}_2, \underline{x}) \\ \vdots \\ (\underline{e}_m, \underline{x}) \end{pmatrix} = \begin{pmatrix} (\underline{x}, \underline{e}_1,) \\ (\underline{x}, \underline{e}_2,) \\ \vdots \\ (\underline{x}, \underline{e}_m,) \end{pmatrix}$$

Where we have used the fact that the dot product is commutative, and can be defined as:

$$(\underline{e}_i, \underline{x}) = \underline{e}_i^T \underline{x}$$

As a sanity check, we can see that $\boldsymbol{E}^T \in \mathbb{R}^{m \times D}$ and $\underline{x} \in \mathbb{R}^D$, so we expect $\underline{x}_{proj}$ to by a vector in $\mathbb{R}^m$, as required. Using the matrix $\boldsymbol{E}$ is just a more compact way of describing how each component of $\underline{x}_{proj}$ is just the projection of $\underline{x}$ onto each of the basis vectors.

Principal Components as Eigenvectors of the Covariance Matrix

We have defined principal components as basis vectors, such that data along them has maximum variance. Lets assume a vector $\underline{e}$ is a principal component of some mean normalised data. Lets consider the variance resulting from projecting a given point of the data $\underline{x}_i$ onto $\underline{e}$. From the work above, we know that the projection will be given by:

$$\underline{x}_i^T \underline{e} = \sum_{j=1}^{D} x_{ij} e_j$$

Let μ' denote the average of the points projected onto $\underline{e}$. Then, there are N total projections, where the ith projection is given by $\sum_{j=1}^{D} x_{ij} e_j$, so:

$$\mu' = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{j=1}^{D} x_{ij} e_j \right)$$

Interchanging the summations:

$$\mu' = \sum_{j=1}^{D} e_j \left(\frac{1}{N} \sum_{i=1}^{N} x_{ij} \right)$$

But notice, $\frac{1}{N}\sum_{i=1}^{N} x_{ij}$ is the average of attribute j in the normalised data, so by definition, it must be 0 (since the data is mean normalised). Hence:

$$\mu' = \sum_{j=1}^{D} e_j \times 0 = 0$$

In other words, the variance of the points projected onto $\underline{e}$ is given by:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}\left(-\mu' + \sum_{j=1}^{D} x_{ij}e_j)\right)^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{j=1}^{D} x_{ij}e_j\right)^2$$

We can split the squared term into 2 terms, taking care to chang eth eindex of summation:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{j=1}^{D} x_{ij}e_j\right)\left(\sum_{k=1}^{D} x_{ik}e_k\right)$$

We have infinite summations, so we can move them around:

$$\begin{aligned}\sigma^2 &= \frac{1}{N}\sum_{i=1}^{N}\left(\sum_{j=1}^{D} x_{ij}e_j\right)\left(\sum_{k=1}^{D} x_{ik}e_k\right)\\ &= \sum_{i=1}^{N}\sum_{j=1}^{D}\sum_{k=1}^{D}\left(\frac{1}{N}x_{ik}e_k x_{ij}e_j\right)\\ &= \sum_{j=1}^{D}\sum_{k=1}^{D} e_k e_j\left(\frac{1}{N}\sum_{i=1}^{N} x_{ik}x_{ij}\right)\end{aligned}$$

But recall, $\frac{1}{N}\sum_{i=1}^{N} x_{ik}x_{ij}$ is precisely $Cov(k,j)$, so we have:

$$\sigma^2 = \sum_{j=1}^{D}\sum_{k=1}^{D} e_k e_j Cov(k,j)$$

Notice, if we want to maximise this expression, we can simply make $\|\underline{e}\| \to \infty$, which will make each e_j, e_k extremely large. Hence, we shall enforce the constraint $\|\underline{e}\| = 1$ (and for convenience we will use $\|\underline{e}\|^2 = 1$). This now becomes a constrained optimisation problem, which can be solved via **Lagrange Multipliers** (more on them here). The method of Lagrangian Multipliers tells us that to optimise the expression above given a constraint, is equivalent to optimising:

$$V = \sum_{j=1}^{D}\sum_{k=1}^{D} e_k e_j Cov(k,j) - \lambda(\|\underline{e}\| - 1) \implies V = \sum_{j=1}^{D}\sum_{k=1}^{D} e_k e_j Cov(k,j) - \lambda\left(\left[\sum_{c=1}^{D} e_c^2\right] - 1\right)$$

To optimise, lets consider the partial derivative with respect to some attribute a (we want to optimise with respect to each element in $\underline{a}$):

$$
\begin{aligned}
&\frac{\partial V}{\partial e_a} = 0 \\
\implies &\frac{\partial}{\partial e_a}\left(\sum_{j=1}^{D}\sum_{k=1}^{D} e_k(e_j Cov(k,j)) - \lambda\left(\left[\sum_{c=1}^{D} e_c^2\right] - 1\right)\right) = 0 \\
\implies &\sum_{j=1}^{D}\frac{\partial}{\partial e_a}\left(e_j\sum_{k=1}^{D} e_k Cov(k,j))\right) - 2\lambda e_a = 0 \\
\implies &\sum_{j=1}^{D}\left[\frac{\partial e_j}{\partial e_a}\left(\sum_{k=1}^{D} e_k Cov(k,j)\right) + e_j\sum_{k=1}^{D}\frac{\partial}{\partial e_a}(e_k Cov(k,j)))\right] = 2\lambda e_a \\
\implies &\sum_{k=1}^{D} e_k Cov(k,a) + \sum_{j=1}^{D} e_j Cov(a,j) = 2\lambda e_a \\
\implies &\sum_{k=1}^{D} e_k Cov(a,k) + \sum_{j=1}^{D} e_j Cov(a,j) = 2\lambda e_a \\
\implies &2\sum_{j=1}^{D} e_j Cov(a,j) = 2\lambda e_a \\
\implies &\sum_{j=1}^{D} e_j Cov(a,j) = \lambda e_a \\
\implies &\sum_{j=1}^{D} e_j Cov(a,j) = \lambda e_a
\end{aligned}
$$

Since this holds for $a = 1, 2, \ldots, D$, it follows that for the principal component $\underline{a}$ to lead to maximum variance, we must have:

$$
\begin{pmatrix}
\sum_{j=1}^{D} e_j Cov(1,j) \\
\sum_{j=1}^{D} e_j Cov(2,j) \\
\vdots \\
\sum_{j=1}^{D} e_j Cov(D,j)
\end{pmatrix} = \lambda \underline{e}
$$

But notice, each row in the vector above is a dot product. In particular, if $\mathbf{\Sigma}$ is the covariance matrix of the normalised data, if $\underline{\Sigma}_i$ represents the ith row vector

of $\mathbf{\Sigma}$, we have:

$$\begin{pmatrix} \underline{\Sigma}_1^T \underline{e} \\ \underline{\Sigma}_2^T \underline{e} \\ \vdots \\ \underline{\Sigma}_D^T \underline{e} \end{pmatrix} = \lambda \underline{e} \implies \mathbf{\Sigma} \underline{e} = \lambda \underline{e}$$

In other words, it follows that if $\underline{e}$ is a principal component onto which data is projected, $\underline{e}$ must be an eigenvector of the covariance matrix of the data.

Eigenvalues of the Principle Components

Once we know that the eigenvectors of $\mathbf{\Sigma}$ are the principal components, what are their corresponding eigenvalues? Recall, above we derived that the variance of the points projected on a principal component $\underline{e}$ is given by:

$$\sigma^2 = \sum_{j=1}^{D} \sum_{k=1}^{D} e_k e_j Cov(k, j)$$

But in the work above, we showed that:

$$\sum_{k=1}^{D} e_k Cov(j, k) = \lambda e_j$$

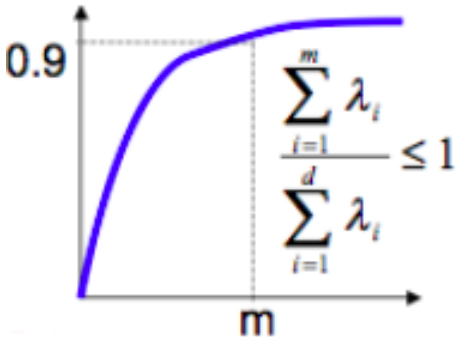
So:

$$\sigma^2 = \lambda \sum_{j=1}^{D} e_j^2 = \lambda \|\underline{e}\| = \lambda$$

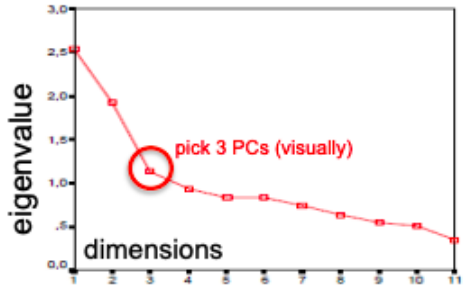
In other words, the eigenvalues of the principal components correspond to the variance of the data along the principal component. Hence, the ith principal component will be the eigenvector with the ith highest eigenvalue.

1.3.3 PCA

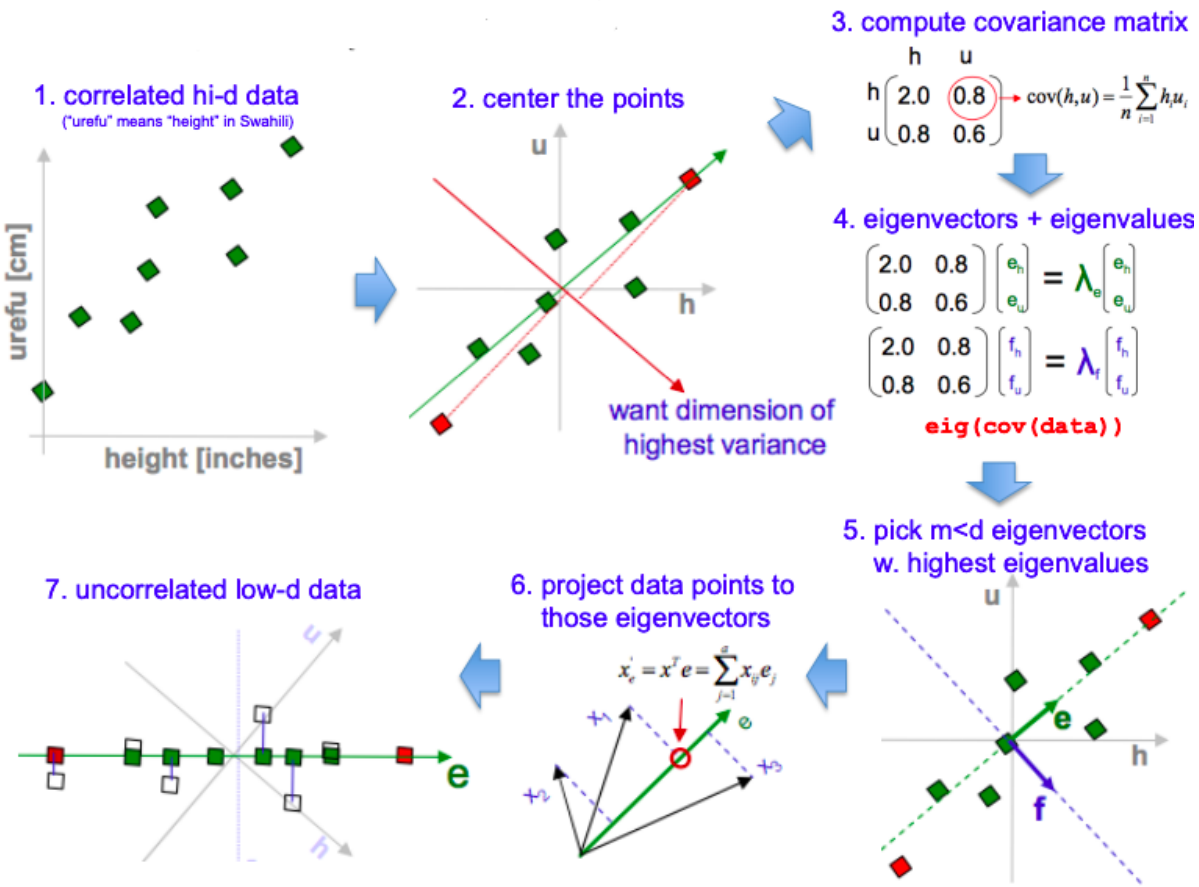
- **How do you select the number of principal components?**
  - when we compute the eigenvectors, we will obtain D new basis vectors
  - however, for dimensionality reduction, we should only pick m of these
  - in order to select what m should be, there are 2 methods:
    - pick the first m eigenvectors, such that the sum of the variance that they explain is more than 90% of the total variance (90% is arbitrary)



* alternatively, a scree plot can be used, alongside the the elbow method



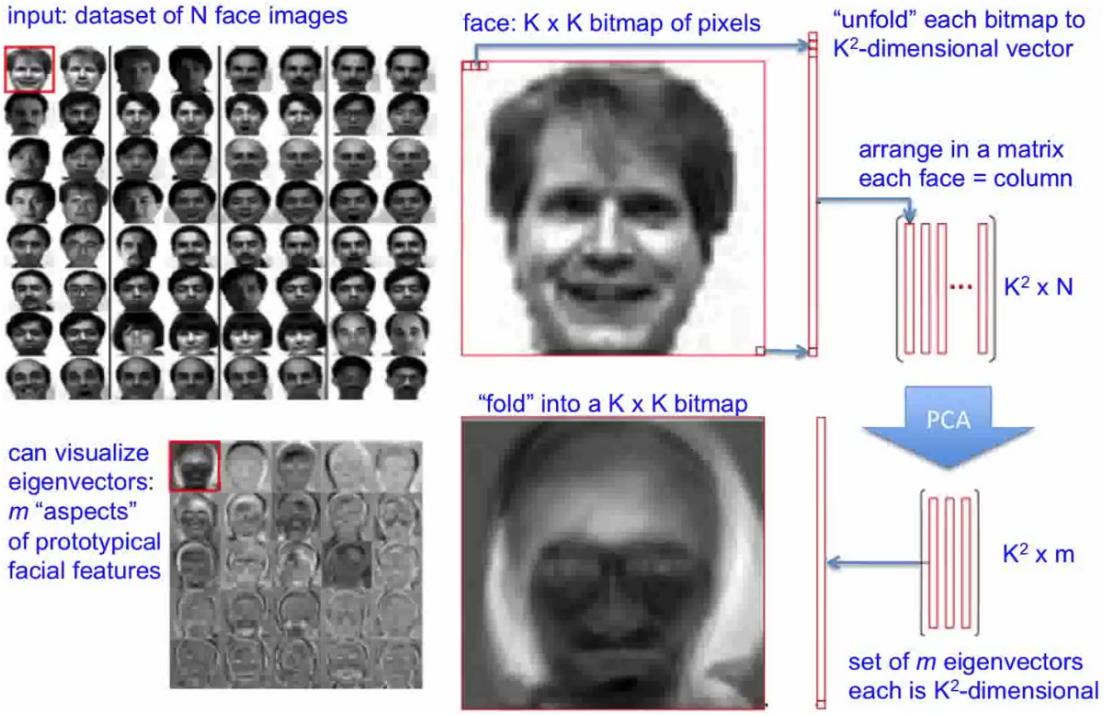
- **How can we use PCA to reduce dimensionality?**



1.3.4 Eigenfaces

PCA on Image Data

We can apply PCA to image data. For example, a $K \times K$ image can be turned into a vector with K^2 elements. From this, we can get the eigenvectors (which will be K^2 dimensional), and unfold them to produce a $K \times K$ image. If we do this with the first principal component, it should outline the areas of high and low variance; in other words, it will tell us the most distinguishing features of the image.



PCA of Transposed Data

In this case, they represent the data using the variables as rows, and the observations as columns. The analysis is similar. If $\boldsymbol{X}$ is the matrix of data (rows represent observations, columns represent variables), the covariance matrix is typically given by $\boldsymbol{\Sigma} = \boldsymbol{X}^T X$. If we however transpose the data, the covariance matrix will be $\boldsymbol{X}\boldsymbol{X}^T$ (we are omitting the $\frac{1}{N}$ factor). Then, assume $\underline{v}$ is an eigenvector (principal component) of $\boldsymbol{\Sigma}$. Then:

$$\begin{aligned} & \boldsymbol{X}^T\boldsymbol{X}\underline{v} = \lambda\underline{v} \\ \implies & \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X}\underline{v}) = \lambda\boldsymbol{X}\underline{v} \\ \implies & \boldsymbol{X}\boldsymbol{X}^T(\boldsymbol{X}\underline{v}) = \lambda(\boldsymbol{X}\underline{v}) \end{aligned}$$

Hence, from the principal coomponents of the original data, we can derive the principal components of the transposed data, by applying the transformation $\boldsymbol{X}\underline{v}$. In a similar way, if $\underline{v}$ were an eigenvector of $\boldsymbol{X}\boldsymbol{X}^T$:

$$
\begin{aligned}
& \boldsymbol{X}\boldsymbol{X}^T\underline{v} = \lambda\underline{v} \\
\implies & \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T\underline{v}) = \lambda\boldsymbol{X}^T\underline{v} \\
\implies & \boldsymbol{\Sigma}(\boldsymbol{X}^T\underline{v}) = \lambda(\boldsymbol{X}^T\underline{v})
\end{aligned}
$$

The Eigenface Space

The eigenvectors form a space, with which we can represent faces, by employing linear combinations of the basis vectors. For example, if $\underline{x}$ is a face, and $\boldsymbol{E}$ is the matrix of eigenvectors (as columns), then as discussed above, the face can be projected into the eigenface space via:

$$\boldsymbol{E}^T\underline{x}$$

If we then take the elements of the projected vector, these are the coefficients of each eigenface basis vector which we need to use to recompose the face. For example:

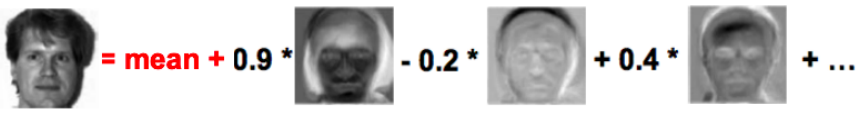


Figure 5: We need to add the mean back, since we normalised the data.

Figure 6: The result of adding a principal component to the mean face. The first face represents the average face in the data. As we add principal components, the face resembles the original a lot more.

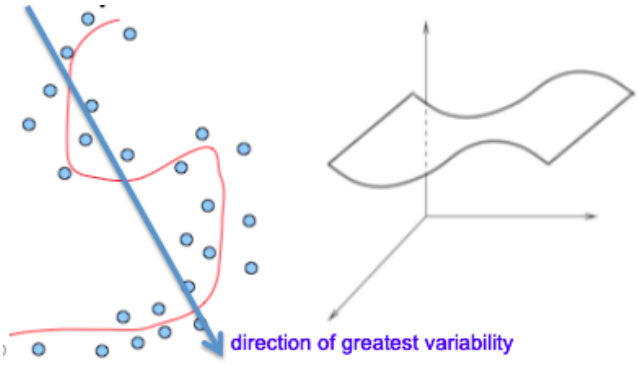
Turns out, this method is relatively robust, in the sense that different lighting or face orientation or facial expression shouldn't strongly affect the eigenface decomposition (since PCA won't pick up on sporadic feature changes, but rather how features differ throughout). Moreover, we can use this to turn normal objects into faces:

1.3.5 Issues with PCA

- **Should we also scale data by its variance?**
  - PCA depends on the covariance matrix, so high covariance can greatly affect the eigenvectors and in particular, the eigenvalues
  - if we have a dimension which has an abnormally large scale, its covariance will be abnormally large, so it will dominate, and that varaible will become the first principal component
  - because of this, it is recommended that we scale data to have 0 mean and unit variance:

$$\frac{\underline{x} - \mu}{\sigma}$$

- **Does PCA perform well with non-linear data?**
  - PCA assumes that data the underlying space which "explains" the data is linear
  - even if the data is non-linear, it will always return an eigenvector in the direction of greatest variance, albeit this will be a straight line
  - this isn't always ideal



1.3.6 PCA for Classification

- **How useful is PCA for classification?**

- reducing dimensions reduces the effect of the curse of dimensionality, which can be usefu for classification
- however, PCA can also damage the task of classification

- **Why can PCA reduce classification power?**
  - PCA only deal with coordinates, so it has no idea about labels (unsupervised)
  - hence, it can be possible that when projecting data into the principal components, data with different labels gets shuffled, making classification impossible

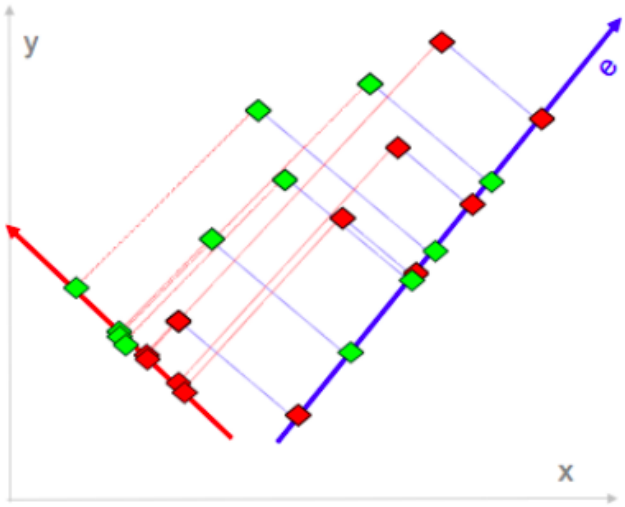
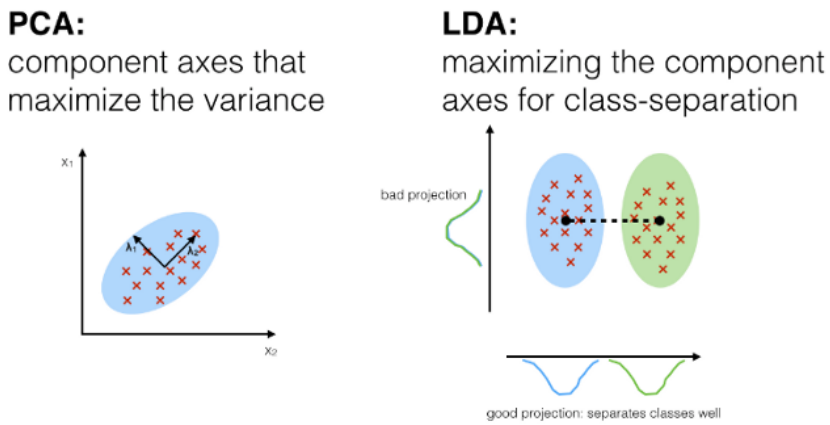
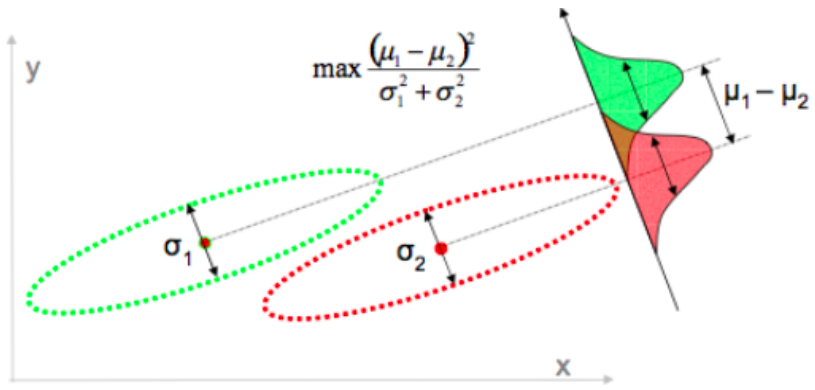


Figure 7: Upon projecting onto PC1, the red and green classes will get jumbled up, so classifiying will be hard. If we had chosen PC2, however, the data would have been fully separated.

1.3.7 PCA vs Linear Discriminant Analysis

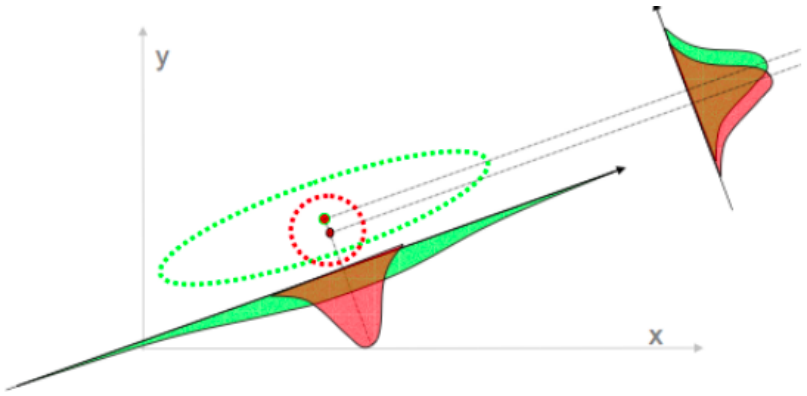
- **How does LDA compare to PCA?**
  - PCA seeks dimensions which maximise variance along them
  - LDA seeks dimensions which, when projected to, data (mean) from different classes have the maximum separation, whilst data from the same class has minimal variance
  - in other words, LDA is built to produce a new space which will be the most useful for a classifier





- **What does LDA assume to produce the new dimensions?**
  - data is Gaussian
  - there exists a boundary between data points of classes
- **Does LDA always perform better than PCA for classification dimensionality reduction?**
  - not necessarily
  - if the data has similar means, but drastically different variance, independently of how big we wnat to make the difference between means, it will never be too much, meaning that data points from classes will be projected close to each other
  - PCA will pick up the big difference in variance between the classes, and much better distribute the classes in the new space

– PCA projected data won't be separable linearly (since the means are close together), but a non-linear classifier would be able to separate the data



1.4 Evaluating Dimensionality Reduction

✓ extracts underlying factors which explain the data (i.e reflects intuition of how temperature can be causing factor of different phenomena)

✓ PCA converts data into data with independent attributes, so dimensions will be uncorrelated, meaning that probabilities in high dimension are easier to compute (independence assumption is valid, so we can use Naive Bayes)

✓ less variables, so less space required to represent data

✗ can be expensive to apply (not good for large dataset)

✗ if classes are fine-grained (close to each other), PCA can do harm

✗ if assumptions don't hold (i.e linear subspace), it won't help

2 Hierarchical Clustering

- Hierarchical clustering removes the need to determine the number of clusters to create
- Top-down methods subdivide the data to generate the hierarchy (hierarchical K-Means), whilst bottom-up methods join small clusters together (agglomerative clustering)
- There are 5 clustering distance measures (single, complete, average, centroid and Ward)
- The Lance-Williams Algorithm provides an efficient way of forming agglomerative hierarchical clustering

2.1 Clustering and Granularity

- **What is inconvenient about K-Means as a clustering method?**
  - the main issue of K-Means is that we need to select the number of clusters
- **What is wrong about the question "how many clusters should we use"?**
  - the number of clusters to select depends on how finely we want to subdivide the data
  - within any real world data, it is always possible to select larger and smaller clusters: it depends on the features of interest
  - however, no clustering algorithm gives us the number of clusters

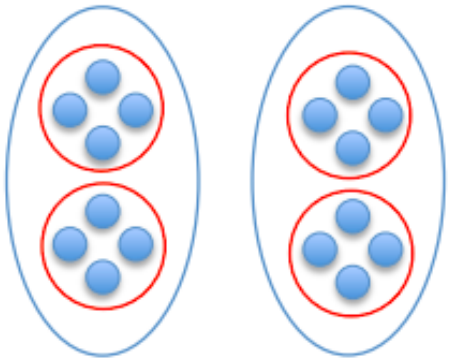


Figure 8: We can think of clustering as analogous to classifying waves in the ocean: we can consider tsunamis or just ripples, but the sea is composed of this and everything in between. The same with clustering: there is no one way which tells us everywthing we need to know about the underlying structure of the data. In this exampl,e we can identify 16, 4 or 2 clusters, depending on what interests us.

- **How does hierarchical clustering solve the issue of choosing K?**
  - instead of choosing the number of clusters, determine a hierarchy of clusters
  - we can then choose a level in the hierarchy to define the granularity of the clusters
- **How is hierarchical clustering organised?**
  - at the top level of the hierarchy, we have coarser clusters - they give us the main, more apparent differences in the data.
  - the **topmost** cluster corresponds to all of the data
  - at the bottom level of the hierarchy, we have a fine grain clustering, looking at the small features that distinguish 2 clusters of data
  - the **bottom** cluster corresponds to singleton clusters of all the data
- **What are the strategies used to build a hierarchical cluster?**
  - **top-down clustering**: hierarchy from recursively splitting all of the data
  - **bottom-up clustering**: hierarchy from merging singleton clusters together

2.2 Hierarchical K-Means

- **What is the hierarchical K-Means algorithm?**

- use K-Means to produce a **top-down** hierarchical clustering
- we use K-Means to split the data into K clusters
- we recursively apply this on each cluster produced to further split it

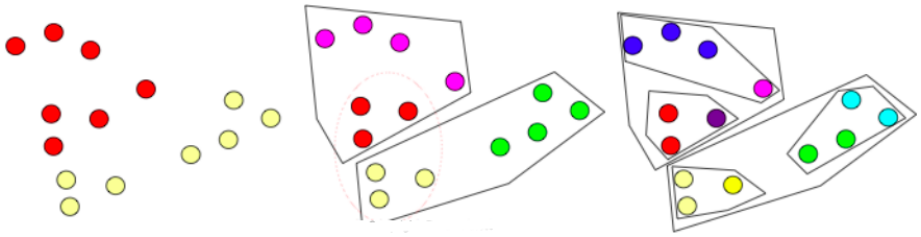


Figure 9: We originally split the data using $K = 2$ into the red and yellow clusters. At the next step, the red cluster is split into the red and pink clusters, whilst the yellow cluster is split into the yellow and green clusters.

- **What are the benefits and drawbacks of hierarchical K-Means?**
  - since at each step it gets recursively called on smaller subsets of the data, the algorithm is **fast**:

$$\mathcal{O}(K \times n \times D \times \log_K n)$$

  - however, once clusters are formed, points from different clusters can never be part of the same cluster. In particular, points which are **close together** might not get put into the same cluster. This is apparent in the diagram above, where the bottom left red and yellow clusters could form a good cluster.

2.3 Agglomerative Clustering

2.3.1 Algorithm

- **What is the purpose of agglomerative clustering?**
  - **agglomerative clustering** is a bottom-up hierarchical clustering algorithm
  - it guarantess that if 2 points are close together, they will be clustered together
  - the definition of "close together" is the most important part of this algorithm
- **What is the agglomerative clustering algorithm?**
  1. Initialise a collection C of clusters, using n singleton clusters c_i from each of the n data points

2. Repeat n times (until only one cluster is left):
   - determine the 2 **clusters** which are **closest together**. This is defined by some distance measure D and:

$$\min_{i,j}\{D(c_i, c_j)\}$$

   - merge c_i and c_j into c_{i+j}
   - remove c_i, c_j from C, add c_{i+j}

- **What are the benefits and drawbacks of agglomerative clustering?**
  - it ensures that if 2 points are close togehter, they are clustered together
  - the hierarchy can be nicely visualised by a **dendrogram**

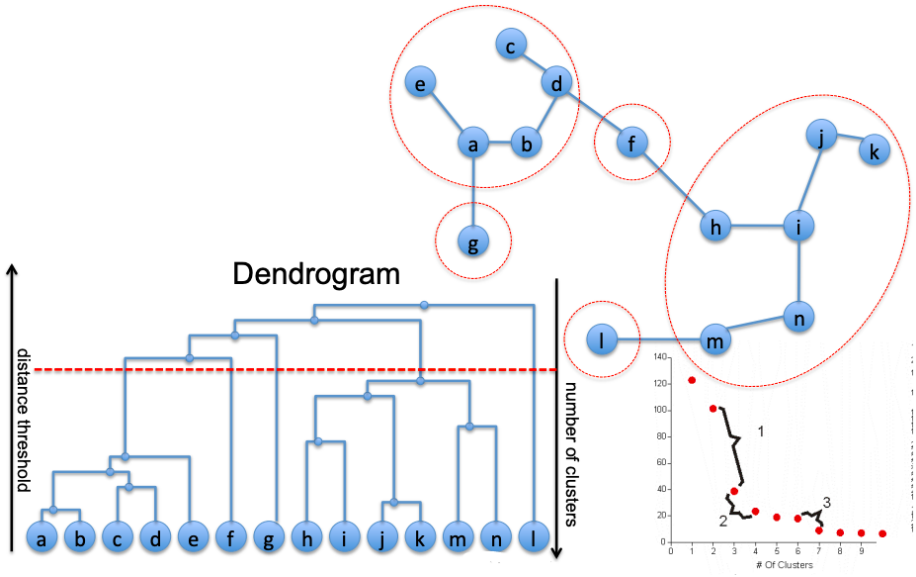


Figure 10: At the bottom left a dendrogram. At the bottom level, we place the singleton clusters. At each step when 2 clusters are merged, a line is drawn joining them. For example, the first cluster formed will be between a and b. The height of the line represents how close the 2 clusters were. The red line "cuts" the dendrogram, and this cut represents a clustering: $\{a, b, c, d, e\}, \{f\}.\{g\}, \{h, i, j, k, m, n\}, \{l\}$. The **distance threshold** defines were the line is placed, such that no further clustering is undertaken. This clustering is shown at the top right by the circled nodes. At the bottom right, a scree plot, which can be used to determine the number of clusters to consider. At the x-axis, we have the number of clusters; at the y-axis, we have the **merge distance/distance threshold**.

– however, the algorithm is very slow: you need to create a distance matrix, and at each step recompute distances:

$$\mathcal{O}(n^2 d + n^3)$$

– moreover, we need to define a distance metric, to determine when 2 clusters are close enough to merge

2.3.2 Distance Measures

There are 5 types of **cluster distance measures** in hierarchical clustering:

1. *Single Link*
2. *Complete Link*
3. *Average Link*
4. *Centroids*
5. *Ward's Method*

- **What is the single link distance?**
  - the *shortest* distance between elements in 2 clusters:

$$D(c_1, c_2) = \min_{x_1 \in c_1, x_2 \in c_2} D(x_1, x_2)$$

  - tends to produce enlongated, flat clusters, since to cluster we consier the smallest distance, and the distance itself is based on the smallest distance between elements in clusters

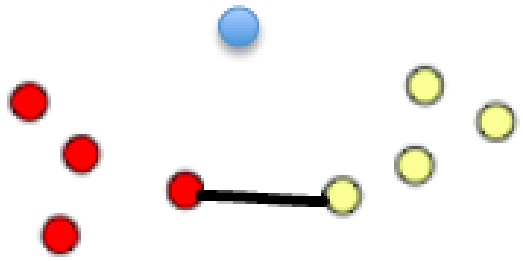


Figure 11: In this case we would cluster the red and yellow clusters.

- **What is the complete link distance?**
  - the *longest* distance between elements in 2 clusters:

$$D(c_1, c_2) = \max_{x_1 \in c_1, x_2 \in c_2} D(x_1, x_2)$$

- tends to produce more spherical clusters
- we then cluster based on the smallest of the largest distances between sub-clusters

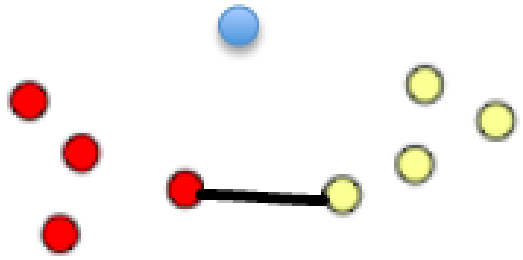
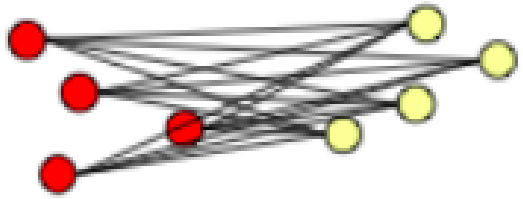


Figure 12: In this case we would merge the yellow and blue clusters.

- **What is the average link distance?**
  - the *average* pairwise distance between elements in the clusters:

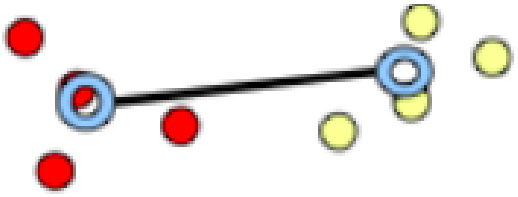
$$D(c_1, c_2) = \frac{1}{|c_1|}\frac{1}{|c_2|}\sum_{x_1 \in c_1}\sum_{x_2 \in c_2} D(x_1, x_2)$$

  - this is useful when clustering to avoid outliers



- **What is the centroid distance?**
  - the distance between cluster *centroids*:

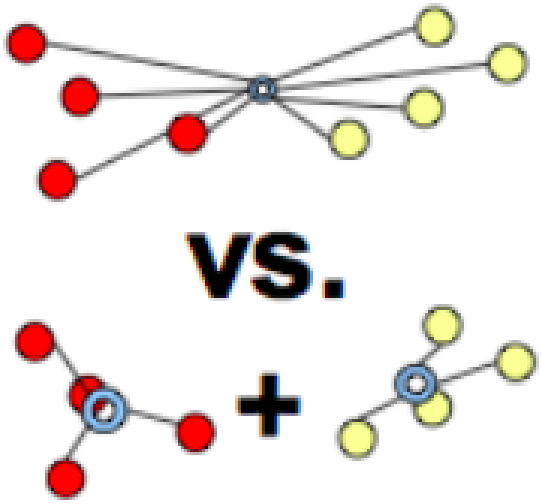
$$D(c_1, c_2) = D\left(\frac{1}{|c_1|}\sum_{x_1 \in c_1} x_1, \frac{1}{|c_2|}\sum_{x_2 \in c_2} x_2\right)$$



- **What is Ward's Method?**
  - for a pair of clusters, we simulate joining them. We then compute the sum of squared distances from each point in the cluster and the cluster centroid:

$$TD_{c_1 \cup c_2} = \sum_{x \in c_1 \cup c_2} D(x, \mu_{c_1 \cup c_2}$$

  - this TD is compared with $TD_{c_1} + TD_{c_2}$
  - Ward's method aims so that when clustering, we minimise the distance of cluster elements from their centroid (as we merge, this distance will increase, since clusters become bigger)
  - more on Ward's Method



2.3.3 Lance-Williams Algorithm

- **What is the Lance-Williams Algorithm?**
  - an efficient way of performing agglomerative clustering (still cubic runtime)
- **How does the algorithm work?**
  1. Define a distance matrix, where $D[i,j] = D(x_i, x_j)$, for $i, j = 1, 2, \ldots, n$. This holds the distances between the original points in the data.
  2. For n iterations (until only 1 cluster left):
     - determine the smallest $D[i,j]$ (so that i, j is the pair of closest clusters)
     - merge the closest clusters $\longrightarrow i + j$

– add the cluster $i + j$, delete the clusters i and j

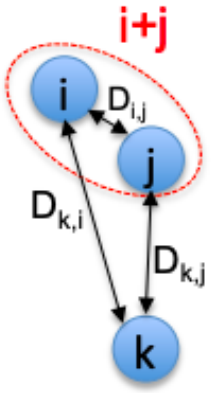


Figure 13: i and j are the closest clusters, so join them as a single cluster

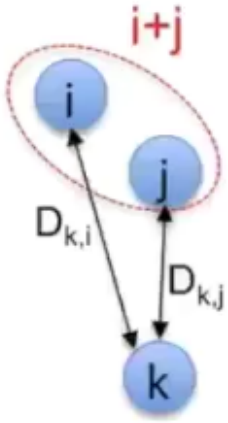


Figure 14: We can then "disconnect" i and j, since they are a single cluster

– now, for each remaining cluster k, we can "disconnect" k from i and j, and we need to compute $D[k, i + j]$. Here is where Lance-Williams obtains an efficiency improvement: instead of brutely recomputing all values, they define:

$$D[k, i + j] = \alpha_i D[k, i] + \alpha_j D[k, j] + \beta D[i, j] + \gamma |D[k, i] - D[k, j]|$$

– turns out, depending on the distance measure, we can set values of the constants, such that we obtain the updated distance measure:

Method	α_i	α_i	β	γ
Single linkage	0.5	0.5	0	-0.5
Complete linkage	0.5	0.5	0	0.5
Group average	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	0	0
Weighted group average	0.5	0.5	0	0
Centroid	$\frac{n_i}{n_i+n_j}$	$\frac{n_j}{n_i+n_j}$	$\frac{-n_i \cdot n_j}{(n_i+n_j)^2}$	0
Ward	$\frac{n_i+n_k}{(n_i+n_j+n_k)}$	$\frac{n_j+n_k}{(n_i+n_j+n_k)}$	$\frac{-n_k}{(n_i+n_j+n_k)}$	0

Figure 15: Errate: α_i in second column shuld be α_j

How does the Lance-Williams formula work?

- lets consider *single link*
- if we were to update the distance, we would just set:

$$D[k, i+j] = \min\{D[k,i], D[k,j]\}$$

since we just want the minimum distance between k and any element of the cluster

- if we apply the Lance-Williams formula, it tells us that:

$$D[k, i+j] = 0.5(D[k,i] + D[k,j] - |D[k,i] - D[k,j]|)$$

- it might not seem immediately obvious that the RHS is indeed the minimum of the 2 distances, so lets consider a numerical example:

$$\begin{aligned} &D[k,i] = 0.8 \\ &D[k,j] = 0.2 \\ &|D[k,i] - D[k,j]| = 0.8 - 0.2 = 0.6 \\ \implies &D[k, i+j] = 0.5(0.8 + 0.2 - 0.6) = 0.2 = \min\{0.8, 0.2\} \end{aligned}$$

- why did this work? Notice, we can write $0.8 = 0.2 + 0.6$. In other words:

$$\max\{a,b\} = \min\{a,b\} + |b-a| \implies \min\{a,b\} = \max\{a,b\} - |b-a|$$

- thus, if we imagine $D[k,i] = \max\{D[k,i], D[k,j]\}$ and $D[k,j] = \min\{D[k,i], D[k,j]\}$, then:

$$\begin{aligned} D[k, i+j] &= 0.5(D[k,i] + D[k,j] - |D[k,i] - D[k,j]|) \\ &= 0.5(2 \times \min\{D[k,i], D[k,j]\} \\ &= \min\{D[k,i], D[k,j]\} \end{aligned}$$

as required.

2.4 Clustering Summary

- Clustering focuses on determining patterns in subpopulations
- **K-Means**: fast, iterative, reaches local minimum. Pick K based on high decrease in intra-cluster variance
- **Mixture Models**: probabilistic K-Means using Expectation-Maximisation to increase likelihood of models
- **Hierarchical Clustering**: top-down (K-Means) or bottom-up (Agglomerative). Performance depends on the measure used to define the dsitance between clusters (single, complete, average, centroids, ward)