CAB420: Dimension Reduction

DIMENSIONS ARE BAD, MMMKAY

Why Reduce Dimensions?

- True vs Observed Dimensionality
  - Consider 20x20 binary bitmaps of handwritten digits
    - We have $\{0,1\}^{400}$ possible patterns
    - Most of these will never be seen
  - True dimensionality
    - Possible variations of the pen stoke
    - The set of examples we can actually see

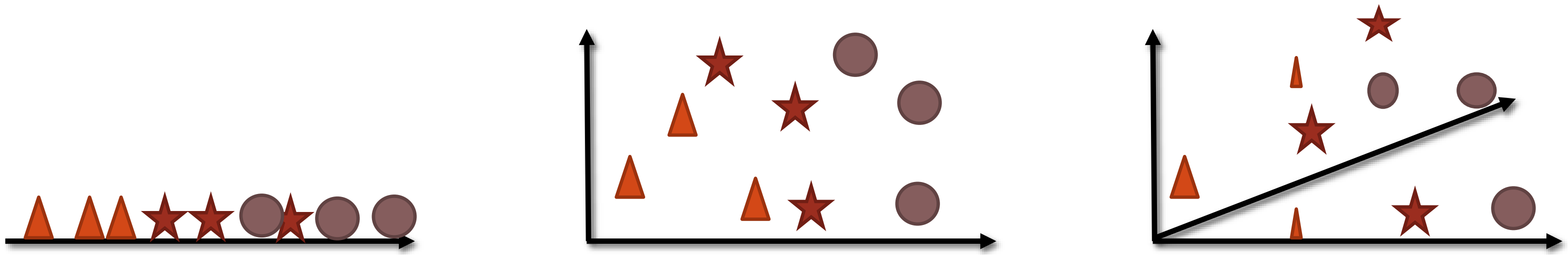
Curse of Dimensionality

Machine learning methods are based on statistics

- More observations means more reliable statistics

But

- More dimensions (without more examples) means that our examples are sparser in our feature space
- Less reliable models



Why Reduce Dimensions?

- Simplify or reduce the data
  - Help better represent the data given a limited number of samples
- Not all dimensions are equal
  - Some are high informative, others are not so useful, it’s good to get rid of the useless ones
- Can make other machine learning tasks easier
  - And faster

Principal Component Analysis

PCA

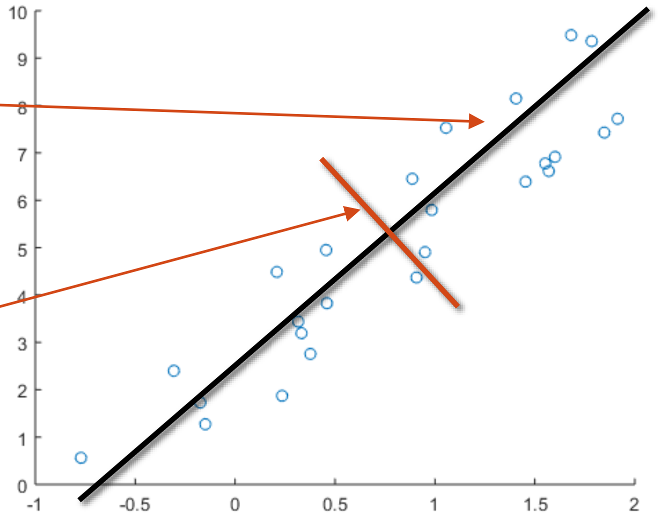
Principal Component Analysis

Principal components are:

- The directions where there is the most variance
- Can be seen as the underlying structure of the data

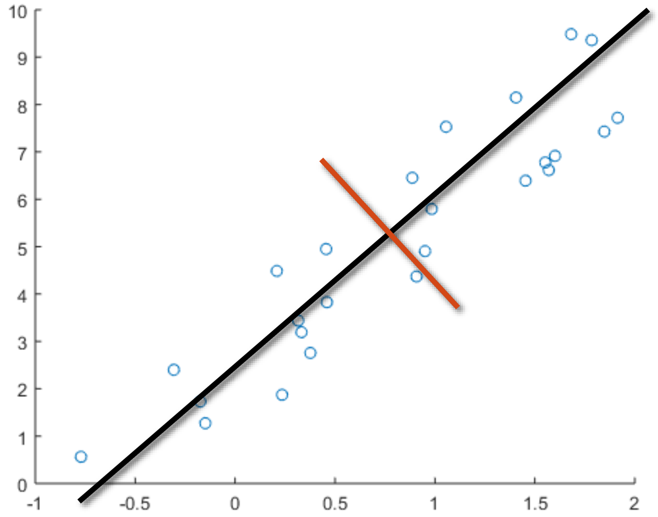
First Principal Component

Second Principal Component



Eigenvectors and values

- Exist in pairs
  - Eigenvectors are the directions of variance
  - Eigenvalues specify the amount of variance in for the eigenvector
- We have as many pairs as we do dimensions in the source data



Finding Eigenvalues and Eigenvectors

- A covariance matrix can be formed between all dimensions in a dataset
- This matrix will be the same whether or not our data is centered (i.e., every entry in a dimensions is reduced by its mean)
- Principal components (PCs) are the directions in which variance is changing the most, not typically along an axis line
- PCs are represented by both eigenvalues and eigenvectors
  - Eigenvalues are related to the amount of variance in a principal component
  - Eigenvectors determine the direction of the PCs variance
  - These eigenvectors must have a magnitude of 1, as they will become unit measures.
- Recovery of Eigenvalues and Eigenvectors is done via linear algebra
  - Outside the scope of this subject
- Can also be found using Singular Value Decomposition
  - Typically, more numerically stable

Principal Component Analaysis

- As the output of our PCA process, we get

$$T = XW$$

  - X is the input data, containing N samples and P dimensions
  - T is the transformed data
    - Same size as X
  - W is the learned transformation
    - $P \times P$ matrix
    - Each column describes how the P dimensions are combined to create a new dimension
- To compute PCA, we need $N > P$
  - More samples than dimensions

Principal Component Analysis

- PCA will also offer a measure of variance for each new dimension
  - Derived from the Eigenvalues
  - New dimensions are returned in order of variance
  - The first few dimensions can be seen as the most important
    - Contain the most variance, thus the most information
    - Dimensions that contain limited variance are closer to being a constant, thus contain less information
- Using data that is not standardised can distort PCA results

Principal Component Analysis

Doesn’t

- Add new dimensions
- Remove dimensions
- Change the data

It does

- Re-project the data along a new of axes such that
  - The first dimension has the most variance
  - The second dimension has the second most variance
  - And so on

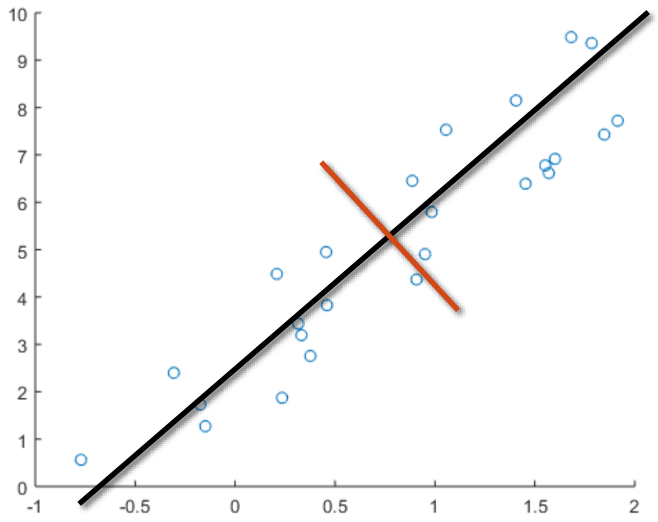
Projecting the Data

- When projecting it also centres the data
  - Translations of the data have no impact on the PCA results
- Projected axes are orthogonal to each other
  - i.e. at right angles
  - Ensures that the new axes can cover the data space as efficiently as possible
- We can also transform from the PCA space back to the original space

$$\hat{X} = TW'$$

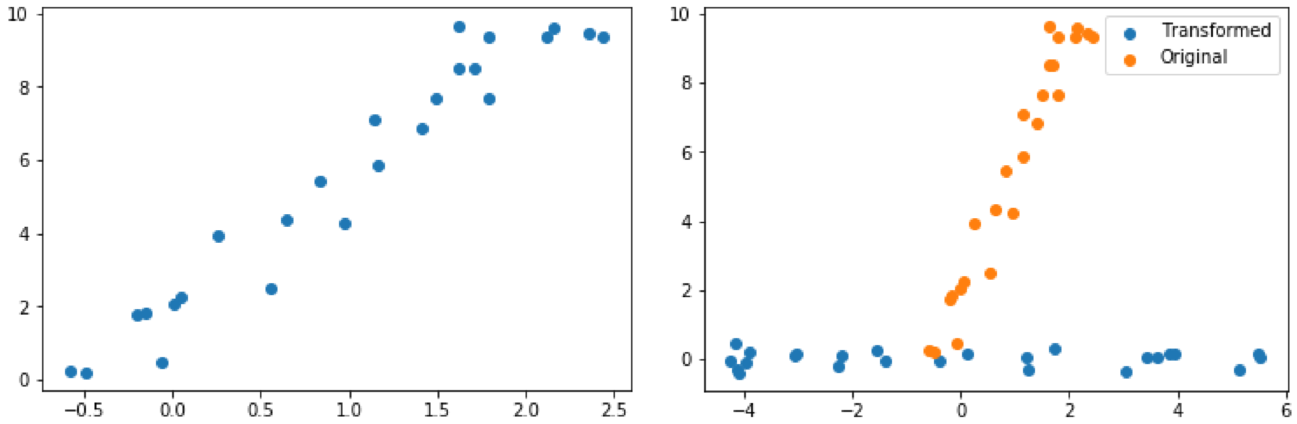
What are the new Dimensions?

- The new dimensions can be seen as weighted sums of the existing ones
  - Our first PC is (roughly)
    - 0.5x + 0.5y
  - Our second PC is (roughly)
    - -0.5x + 0.5y
- PCA can be seen as computing a rotation matrix and rotating the data



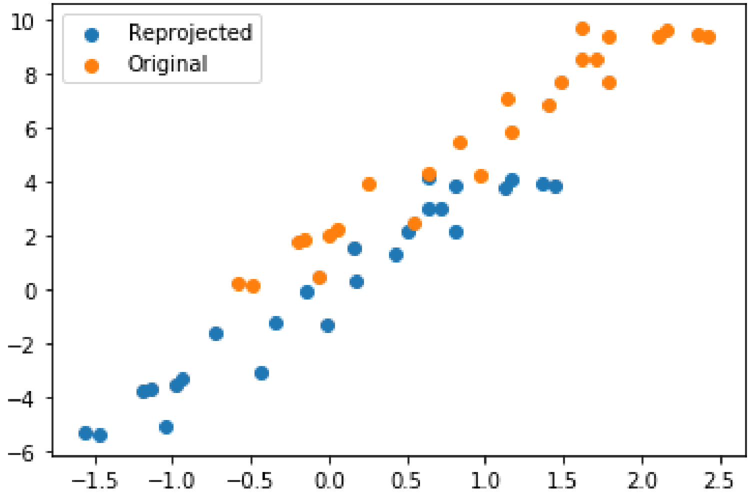
Simple Example

- See ***CAB420_Dimension_Reduction_Example_1_Principal_Component_Analysis.ipynb***
- Our data
  - Some random points on a line
- After applying PCA
  - Points are "lined up" on X and Y dimension
    - Centred
  - The way the points are distributed (their shape) is the same



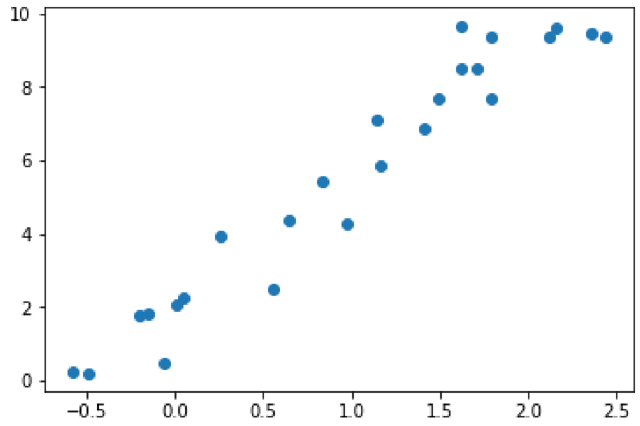
Simple Example: Reprojection

- When we reproject the data, we get a shifted version
  - PCA centers the data for us
    - Set the mean to be 0
    - First step of standardisation
  - Our reprojected version is thus centered at the origin
    - We need to save the mean of the original data to totally recover the original
    - Python will save the mean for us



Simple Example

- Projection matrix, W, shows how new dimensions are created
  - New dimensions are a weighted combination of original dimensions
- Eigenvalues show variance in each new dimension
  - First dimension contains 99.5% of the variance
  - This makes sense, our original data is spread out along one narrow line




```
W = [[-0.260 -0.966]
     [-0.966  0.260]]

Eigenvalues = [12.144
                0.050]
```

CAB420: PCA for Dimension Reduction

WE’VE GOT SOME NEW AXES, NOW WHAT?

Reducing Dimensions

PCA gives us

- A new coordinate frame
- Knowledge about how much of the total variance is captured by each new dimension

So we can just select the top N dimensions

- Retain a given amount of variance
- Remove dimensions that contribute very little

Reducing Dimensions

- We can view this as being
$$T_L = XW_L$$
  - L is the number of dimensions we wish to keep
    - We select the first L columns of W, W_L
    - Our output now has only L dimensions

- We can reproject from the reduced space using
$$\hat{X} = T_L W_L'$$
  - Information will be lost in such a reprojection

An Example

- See ***CAB420_Dimension_Reduction_Example_2_PCA_and_Dimension_Reduction.ipynb***
- Our data:
  - The Iris dataset, measurements of Iris petals, 4 dimensions
- Approach:
  - Apply PCA and transform data
  - Reconstruct the original using only some of the dimensions

Computed Transform

- When we compute PCA, we get
  - A transform
    - We have 4 dimensions in our input, so our transform matrix is 4x4
  - A measure of variance in each new dimension
    - The explained variance ratio is shown below (rather than the raw Eigenvalues)

```
[[ 0.361 -0.085  0.857  0.358 ]
 [ 0.657  0.730 -0.173 -0.075]
 [-0.582  0.598  0.076  0.546]
 [-0.315  0.320  0.480 -0.754]]
```

```
[0.92461872
 0.05306648
 0.01710261
 0.00521218]
```

1 Dimension

- Reconstructed and actual values for first 5 samples
  - Reconstructed samples are close to the original in values
  - We had ~92.5% of variance in the first principal component

```
Reconstructed:
[[4.86247892 3.28673941 1.4328746  0.22688569]
 [4.79929088 3.30151808 1.2830867  0.16423922]
 [4.85120324 3.28937661 1.40614547 0.21570665]
 [4.85721176 3.28797132 1.42038875 0.22166368]
 [5.01906124 3.25011732 1.8040546  0.38212597]]

Actual:
[[4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]]

Error: 0.085604
```

2 Dimensions

- Reconstructed and actual values for first 5 samples
  - Slightly more accurate than 1 dimension

```
Reconstructed:
[[4.7462619  3.15749994 1.46356177 0.24024592]
 [4.70411871 3.1956816  1.30821697 0.17518015]
 [4.6422117  3.05696697 1.46132981 0.23973218]
 [5.07175511 3.52655486 1.36373845 0.19699991]
 [5.50581049 3.79140823 1.67552816 0.32616959]]

Actual:
[[4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]]

Error: 0.025341
```

3 Dimensions

- Reconstructed and actual values for first 5 samples
  - Minimal error with three dimensions included

```
Reconstructed:
[[4.86875839 3.03166108 1.4475168  0.12536791]
 [4.69370023 3.20638436 1.30958161 0.18495067]
 [4.6238432  3.07583667 1.46373578 0.25695828]
 [5.0193263  3.58041421 1.37060574 0.24616799]
 [5.40763506 3.89226243 1.68838749 0.41823916]]

Actual:
[[4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]]

Error: 0.005919
```

Breaking PCA

BREAKING STUFF IS FUN

PCA

- Dimension reduction technique
  - Project the data into a set of new dimensions such that the
    - First new dimension captures the most variance
    - The second captures the second most variance
    - ….
- Can reduce dimensionality by electing to only retain a percentage of total variance
  - Typically most (90% or more) variance is in a small number of dimensions (10% or less)

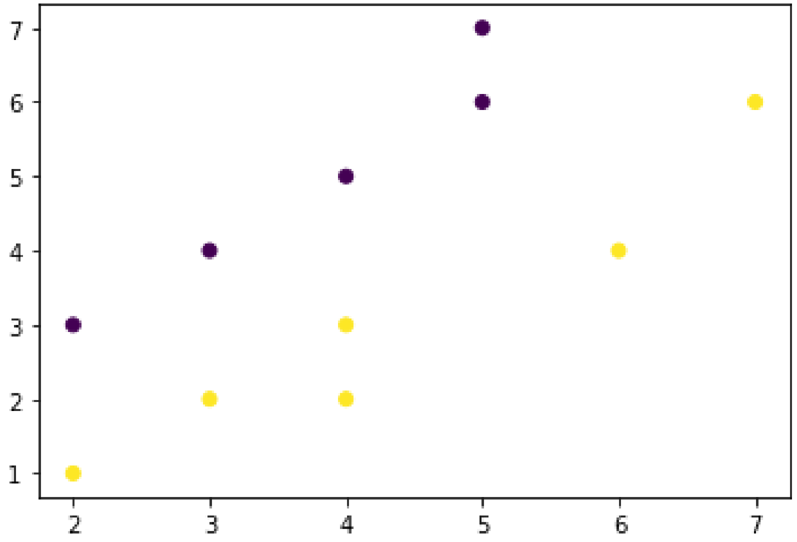
But…

- Is the amount of variance captured the best criteria for determining what to keep?
- Are the most important variables the ones that make different types of object distinct?

Breaking PCA

- See ***CAB420_Dimension_Reduction_Example_3_Linear_Discriminant_Analysis.ipynb*** (the first part)

- Simple dataset
  - Two classes
  - Well separated



Breaking PCA

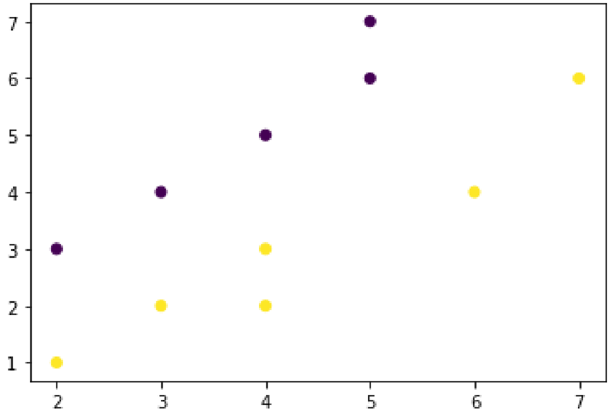
- Computing PCA on our dataset, we get:
  - Explained variance: `[0.85471369 0.14528631]`
    - Most variation is in the first dimension
- If we transform the data using just the first PC, and reproject we get:

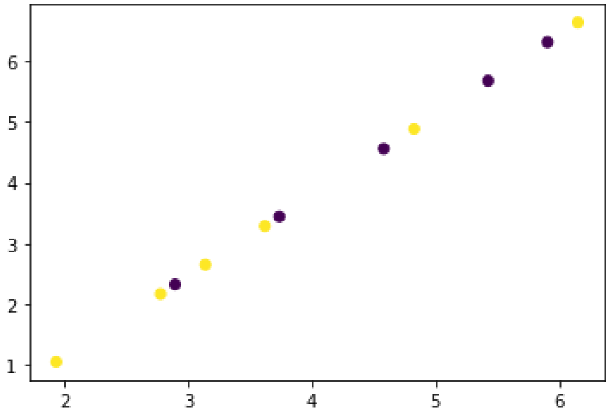
```
Actual:        Reconstructed:
[[3 4]         [[3.73842482 3.44234505]
[4 5]           [4.58253002 4.56007607]
[5 6]           [5.42663522 5.67780709]
[5 7]           [5.90755333 6.31462   ]
[2 1]           [1.9324834  1.05098821]
[3 2]           [2.7765886  2.16871923]
[4 2]           [3.13977569 2.64963734]
[4 3]           [3.6206938  3.28645025]
[6 4]]          [4.82798609 4.88509938]]
```

- Average reconstruction error = 0.408243

Breaking PCA

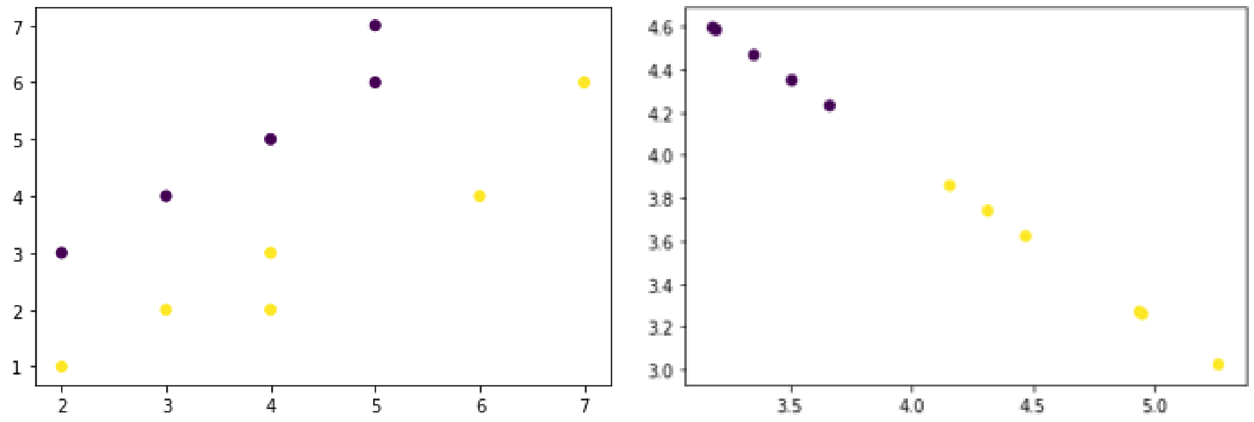
- Left: original data
- Right: reconstructed data (from first PC)
- We've lost any ability to separate the classes





Breaking PCA

- If we used only the second principal component
  - Our reconstruction looks nothing like the original data
  - Class separation is preserved



CAB420: Linear Discriminant Analysis

A BIT LIKE PCA, BUT NOT

Linear Discriminant Analysis

- Also often called Fisher Discriminant Analysis
- LDA seeks to
  - Find a projection that maximises the ratio between the inter class and between class scatter matrices
  - Meaning:
    - Samples in the same class get closer together
    - Samples in different classes get further away
  - Find a projection that best **discriminates** between the classes

LDA Overview

- PCA considers the variance between existing dimensions of the data
  - Finds a new set of dimensions such that the first dimension contains the most variance, etc.
- In Linear Discriminant Analysis, we instead focus on the "variance" relating to classes of the data.
- In order to effectively separate clusters during dimension reduction, we must take into consideration:
  1. The "variance" between classes, and
  2. The "variance" within each class.

LDA Overview

- The difference between each point and its cluster's mean is the crucial part of LDA.
  - We don't need the "divided by n - 1" that we normally associate with variance. In fact, this will only scale our new dimensions.
  - The matrix result without the division is instead called a **scatter matrix**.
- One important note is that we should ensure a sufficient number of points in each class in order for LDA to succeed. This is because LDA is very sensitive to outliers.
- The number of data points in the smallest class should be larger than the number of explanatory variables.

LDA –Scatter Matrices

- LDA is based on scatter matrices
- Within class
  - $S_w = \sum_{i=1}^{n} (x_i - \mu_{y_i})(x_i - \mu_{y_i})^T$
  - where
    - n is the number of points
    - μ_{y_i} is the mean of the ith class
- Between class
  - $S_B = \sum_{k=1}^{m} n_k (\mu_k - \mu)(\mu_k - \mu)^T$
    - where
      - m is the number of classes
      - n_k is the number of points in the kth class
      - μ_k is the mean of the kth class
      - μ is the mean of the data

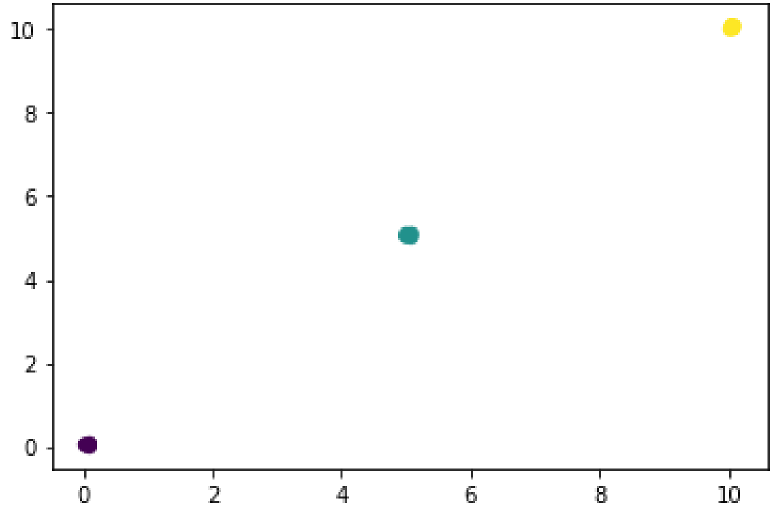
Scatter Matrices Visualised

- See ***CAB420_Dimension_Reduction_Example_3_Linear_Discriminant_Analysis.ipynb*** (the second part)
- 2D Data, 3 Classes
  - Within class scatter

```
[[0.01331477 0.0015448 ]
 [0.0015448  0.01423041]]
```

  - Between class scatter

```
[[248.86709419 249.70285217]
 [249.70285217 250.54892153]]
```



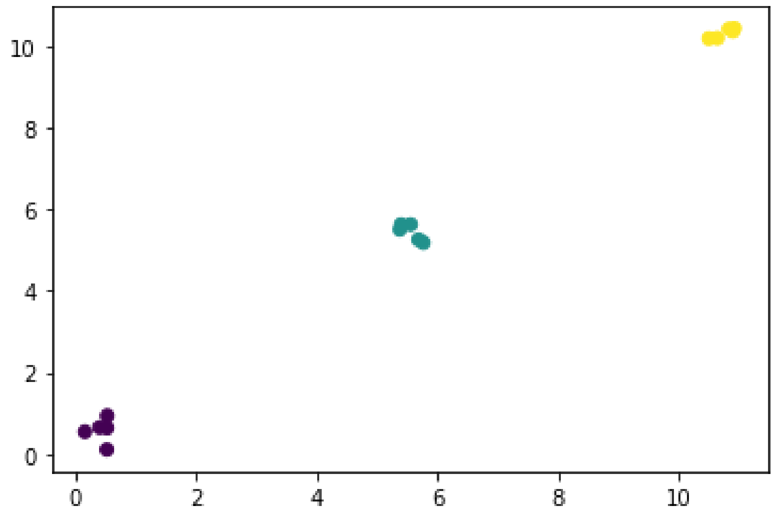
Scatter Matrices Visualised

- 2D Data, 3 Classes
  - Within class scatter

```
[[ 0.35319307 -0.03366045]
 [-0.03366045  0.59361245]]
```

  - Between class scatter

```
[[266.73188433 251.55160505]
 [251.55160505 237.23710806]]
```



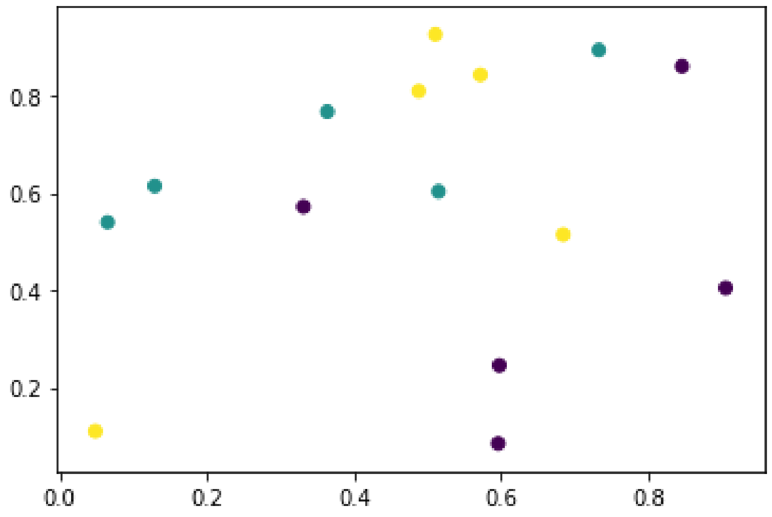
Scatter Matrices Visualised

- 2D Data, 3 Classes
  - Within class scatter

```
[[0.74859984 0.41766819]
 [0.41766819 0.89271018]]
```

  - Between class scatter

```
[[ 0.22332289 -0.19700407]
 [-0.19700407  0.17936888]]
```



Computing LDA

- LDA seeks to maximise ratio of between class to within class scatter

$$\hat{w} = argmax_w \frac{w^T S_B w}{w^T S_W w}$$

- This can be solved by computing the eigenvectors for $\frac{S_B}{S_W}$.
- LDA will only return $C - 1$ meaningful components
  - C is the number of classes in the data

Computing LDA

- Same simple sample data we broke PCA with (on the left)
- Within class scatter matrix

```
[[24.13333333 24.        ]
 [24.         26.        ]]
```

- Between class scatter matrix

```
[[ 0.77575758 -2.90909091]
 [-2.90909091 10.90909091]]
```

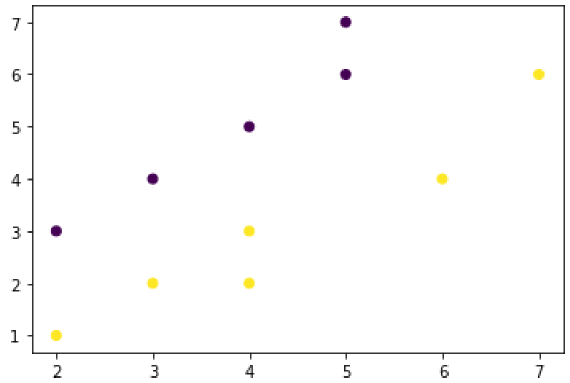
- Transformed Data (on the right)
  - Eigenvectors

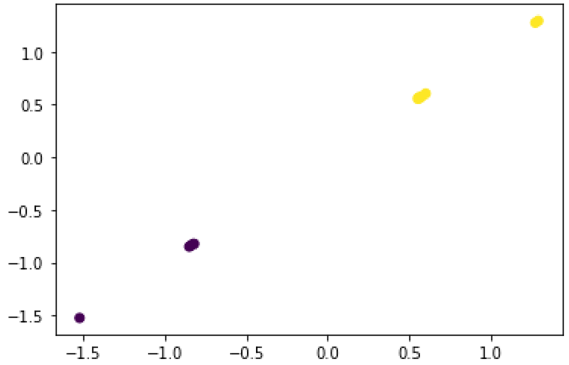
```
[[-0.96623494  0.71169327]
 [-0.25766265 -0.70249034]]
```

  - Eigenvalues

```
[0.          8.22044277]
```

  - Only 1 non-zero Eigenvalue





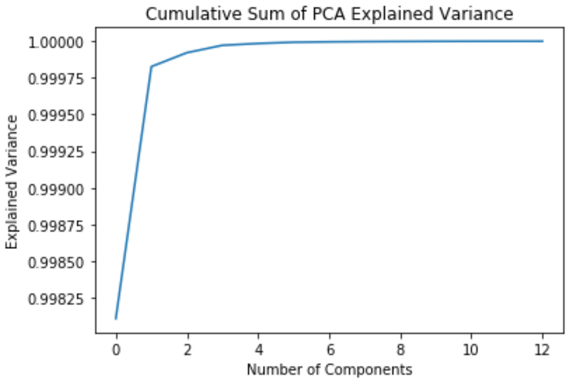
An Example – PCA vs LDA

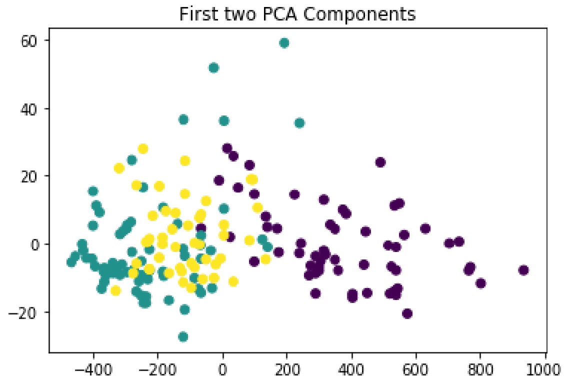
- See ***CAB420_Dimension_Reduction_Example_4_Linear_Discriminant_Analysis_II_Action_Time.ipynb***

- Data
  - Chemical properties of wine from three different cultivars in Italy
    - 12 variables
  - This is ostensibly a classification task, though we'll just analyse the data and visually look at class separation

PCA

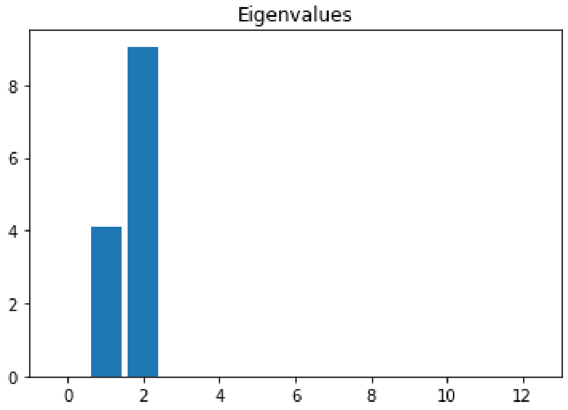
- Almost all the information is in the first dimension
  - 99.8% of the variance in one dimension
- Plotting the top two dimensions (~99.975% of the variance) we see that we have some class sepration
  - Purple is somewhat separated, the other classes not so much

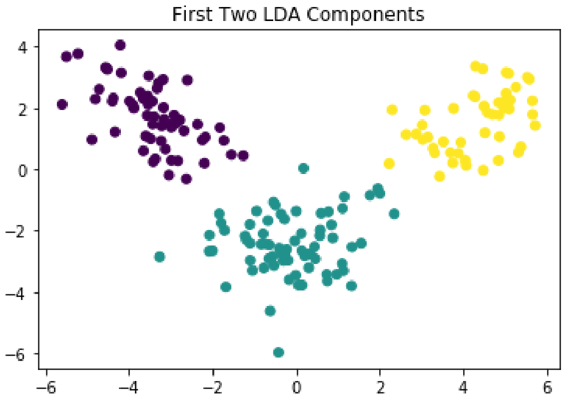




LDA

- Note that I only have two meaningful dimensions
  - LDA returns C – 1 components
  - I have 3 classes, thus 2 components
- Two dimensions result in very good separation of classes



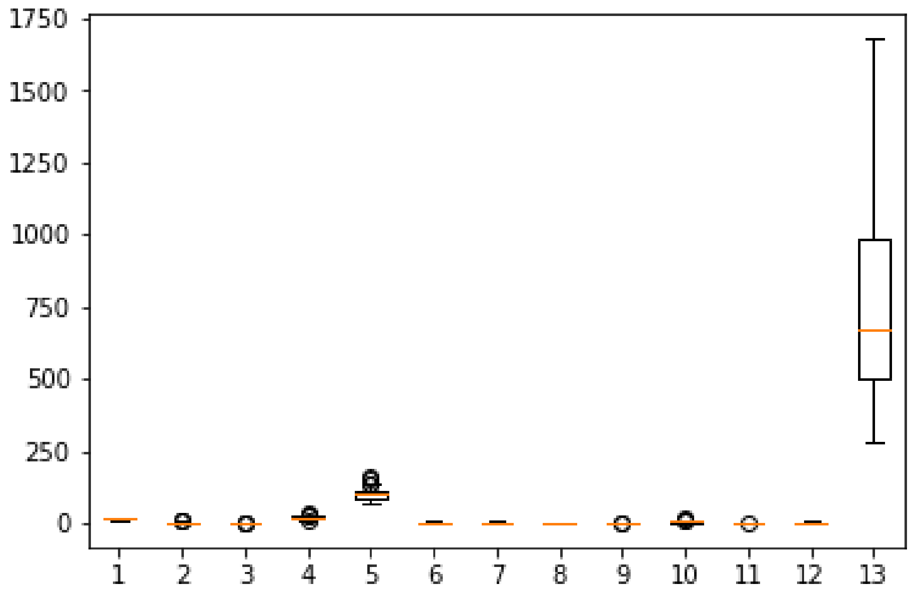


Breaking LDA

- LDA is sensitive to the number of samples per class
- If we have very few samples per class
  - Scatter matrices become hard to predict
  - LDA solution can become unstable
- This problem becomes more pronounced with different solvers
  - SVD (Python's default) is generally more stable and robust in these situations

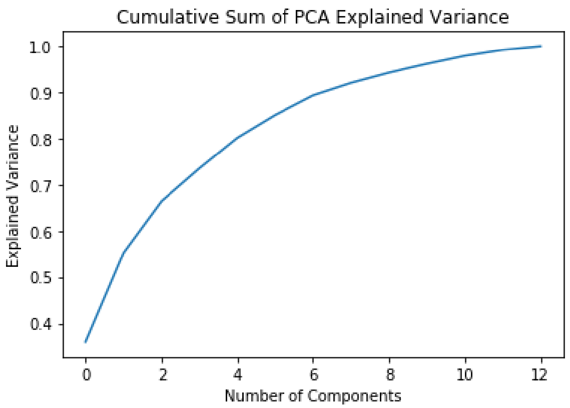
PCA – what went wrong?

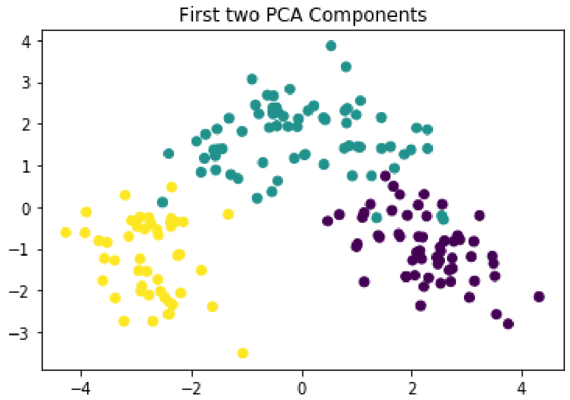
- The boxplot of the data explains our problem
  - The variation in the data is dominated by a single dimension
  - This distrorts our PCA, and hurts it's performance



Improving PCA

- Standardised data
- We no longer have all the variation in one dimension
- Class separation much better on first two dimensions
  - Not as good as LDA, but close





LDA and Acronym Overuse

Warning

- There is another LDA: latent Dirichlet allocation
- Some toolkits/text refer to latent Dirichlet allocation as LDA exclusively
  - If you search for LDA in MATLAB you will get taken to info on latent Dirichlet allocation
  - Google will give you results for both
- "Fisher Discriminant Analysis" is another name for Linear Discriminant Analysis

Incidentally…

- latent Dirichlet allocation is a really cool method for modelling topic distributions
- We'll look at that when we look at sequences

CAB420: t-SNE

YET MORE DIMENSION REDUCTION,
YET ANOTHER ACRONYM

t-distributed Stochastic Neighbour Embeddings

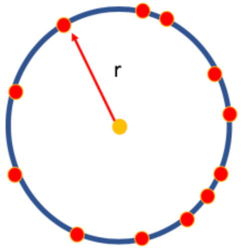
t-SNE is a bit different

- Fairly recent method (2008) developed for visualisation
- Not a linear projection
- Projects data into two dimensions
- Uses local relationships to create a low dimensional mapping that can capture non-linear structure

Why t-SNE?

- Lots of things aren't linear (see left), and PCA (and LDA) struggle to capture such a relationship

- Overcomes the "crowding problem", when projecting from a high to low dimensional space points can become tightly clustered and "crowded"



Why t-SNE?

- Intended for visualisation
  - Often we use dimension reduction to visually analyse data, t-SNE is designed for this

- Becoming widely used
  - Used a lot when visualising neural network intermediate states or outputs
    - We will use it for this and similar purposes
  - Important to understand how it differs from other dimension reduction techniques

t-SNE Overview

Two main steps:

- Step 1: In high dimensional space, create a probability distribution that captures the relationship between points
- Step 2: Create a low dimensional space that follows the created distribution as best as possible
- More details: https://mlexplained.com/2018/09/14/paper-dissected-visualizing-data-using-t-sne-explained/
- https://distill.pub/2016/misread-tsne/

Limitations

- Non-convex optimisation
  - Not guaranteed to reach a global minima (i.e. a constant best solution)
  - Non-deterministic
    - If you run it again, you'll get different results
- Assumes Locally Linear Manifolds
  - Uses Euclidean distance to compare points (which is generally an ok assumption)
  - If local relationships are highly non-linear, may struggle to make sense of things

CAB420: Eigenfaces

FACE REC WITH PCA

Face Recognition: Eigenfaces

- Classic face recognition approach that uses PCA
- See ***CAB420_Dimension_Reduction_Example_5_Eigenfaces.ipynb***
- Problem
  - Identify a person from an image, given an input image and a database of images with known identities

Face Recognition: Naïve Solution

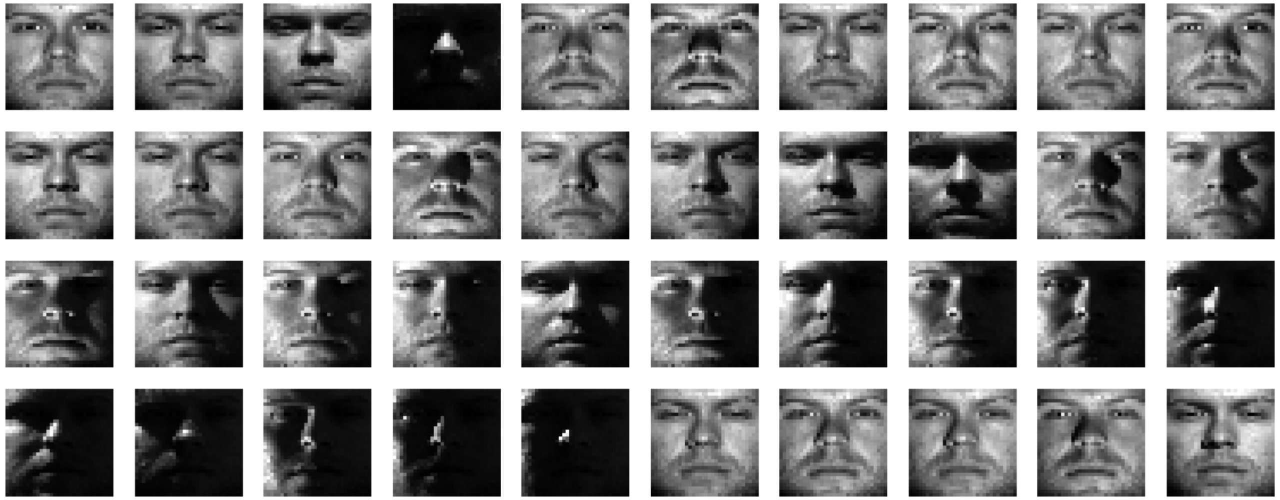
- For input image
  - Align/crop image such that it's at a consistent pose/scale
  - Compare, pixel-wise, to all images in the dataset
  - Select identity of closest image as best match
- Problems with this approach
  - Image are large, even small images have very high dimensionality
  - What happens in the lighting changes, the background changes, etc?

Eigenfaces

- For our database
  - Compute PCA over database, return top N dimensions
- For input image
  - Map to PCA space
  - Compare to database in PCA space and select best match as target ID
- Still sensitive to lighting, background, etc
  - Though less so than a raw pixel approach
- Much quicker due to fewer retained dimensions

Input Data

- Face images
  - 32 x 32 pixels, greyscale
  - Varied lighting
  - Consistent pose
  - Images normalised based on eye locations



PCA and Faces

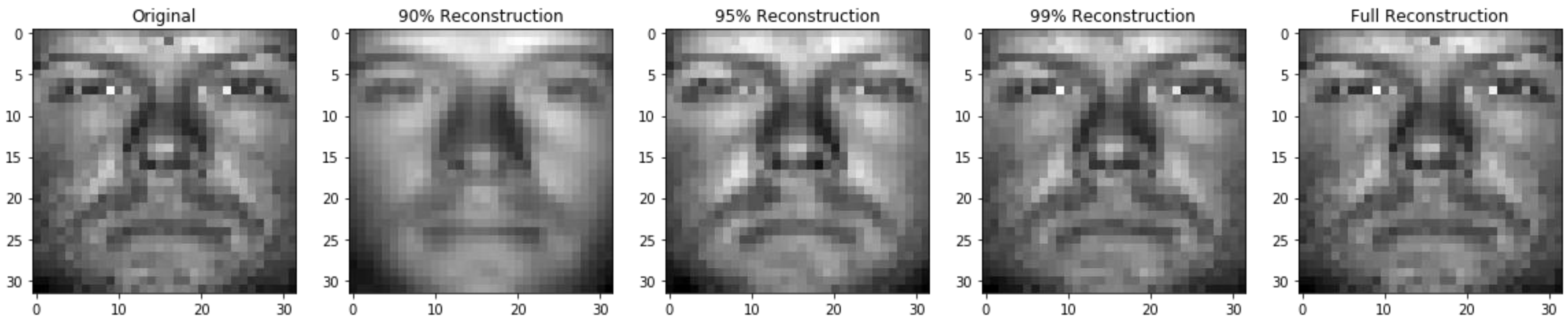
- The mean face
  - Fairly average looking
  - Perhaps more male than female
    - Nature of the dataset, more males than females in the data
  - All lighting variation removed

Eigenfaces

- Our data are images
  - Thus, our eigenvectors (principal components) are images
- Our eigenvectors (eigenfaces) catpure the major variations in faces
  - Early eigenvectors capture the drastic lighting changes

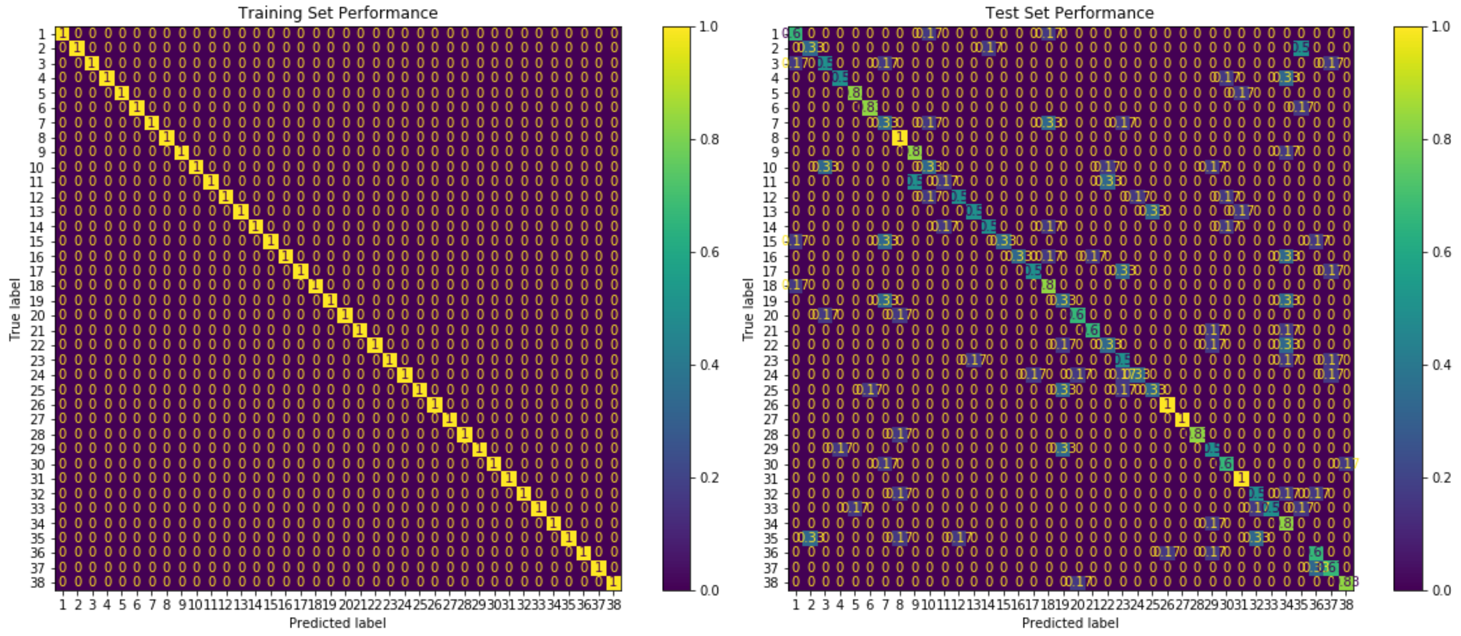
Low Dimensional Faces

- Reconstructions as retained PCs are increased
  - 90% is 23 PCs
  - 95% is 60 PCs
  - 99% is 238 PCs
- More PCs means more fine-grained details
  - Fewer PCs can be (sort-of) seen as a low pass filter



Recognising Low-Res Faces

- Train a classifier
  - CKNN
  - Trained on dimension reduced data



Combining PCA and LDA

LIVING TOGETHER, IN PERFECT HARMONY

(OR SOMETHING LIKE THAT)

Why?

Don’t they do similar things?

- Reduce dimensions, etc?

Sort of, but not really

- PCA is focussed on reconstruction
- LDA focussed on class separation
  - Throws data away

LDA can have problems when you have

- Lots of dimensions and/or classes
- Few sample points per class

When LDA Goes Bad

When we perform LDA with

- Lots of dimensions and/or classes; and/or
- Few sample per class

The estimate of the with within scatter class matrix in particular become poor

- Can lead to overfitting
- Can lead to a singlular or close to singular matrix
  - Singular -> non invertible
  - We need to invert with the within class scatter matrix
- Leads to bad results

PCA as Pre-Processing

A Solution

- Reduce the size of the data space
- i.e. PCA

The process

- Apply PCA, select top N components, removing uninformative dimensions
- Apply LDA to reduced space
- When transforming a new sample point:
  - x_transformed = x*PCA*LDA
  - i.e. transform to PCA space, then to LDA space

This will be explored further in this weeks tutorial

Final Thoughts

SUMMING UP AND ALL THAT

LDA vs PCA

- PCA
  - Unsupervised
  - Aim to preserve as much information as possible
  - Can be inverted

- LDA
  - Supervised
    - We need to know class labels
  - Aim to preserve as much discriminative power as possible
  - Cannot be inverted

PCA Requirements

- PCA is unsupervised
  - No labels needed, just the raw data

- We need to have more samples than we do dimensions
  - If we don’t have this, many PCA methods will transpose the data
  - Or be unstable

- Standardisation can help (a lot)

PCA Requirements

- Ideally, we want many more samples than dimensions
  - PCA may become unstable if we only have slightly more samples than dimensions
- We can do robustness tests by computing multiple PCA transforms on datasets sampled from our overall data
  - If the PCA transform is consistent, we can claim that we have enough data
  - Similar to the idea of the standard error measured when fitting a regression model

LDA Requirements

- Class labels
  - LDA is supervised
- Sufficient examples per class to estimate scatter matrices
  - Ideally, more samples per class than dimensions
- A problem that doesn't require reconstruction
  - We cannot reconstruct the original signal from LDA
- Standardisation not needed
  - The scatter matrices capture this
  - Standardisation may result in an axis being flipped, but that's it

Which one to use?

As the previous slides suggests, it depends on

- What are you trying to do?
  - Classification?
  - Visualisation?
  - Compression?
- What data do you have?
  - Class labels or not?
- Methods can be used in combination
  - FisherFaces: apply PCA, then apply LDA

Lots of other dimension reduction techniques too

The methods that we've look at are not always best. Other methods include:

- Independent Component Analysis
- Factor Analysis (and it's many variants)
- Probabilistic PCA and Probabilistic LDA
- Deep network-based methods
  - Metric learning
  - Auto-encoders
- And many, many, more …