Chapter 6: Dimensionality Reduction

Sophie Sadler

Why dimensionality reduction?

- Cannot visualise more than 3 dimensions easily
- The curse of dimensionality means we need more data in higher dimensions
- Higher dimensions can increase computational cost

x	y
2.00	-1.43
2.37	-2.80
1.00	-3.17
0.63	-1.80

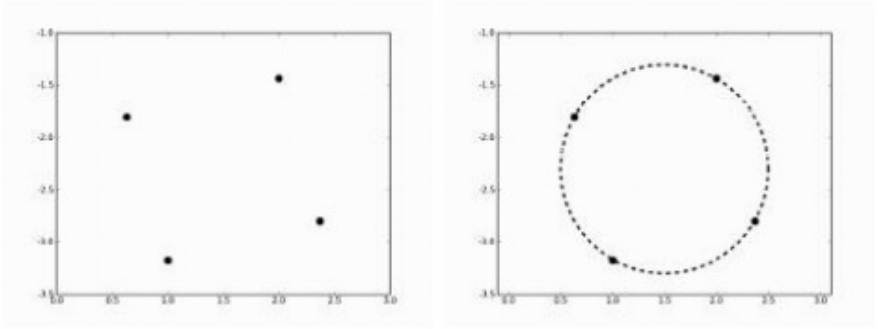

FIGURE 6.1 Three views of the same four points. *Left:* As numbers, where the links are unclear. *Centre:* As four plotted points. *Right:* As four points that lie on a circle.

Types of dimensionality reduction

- Feature selection: simply reduce the number of input features to the ML problem by removing ones which aren't useful.
- Feature derivation: derive new features from combinations of the existing ones
- Clustering: group together similar datapoints to see if this allows fewer features to be used

6.1 Linear Discriminant Analysis (LDA)

- Use the between-class and within-class scatters to reduce dimensions.
- Within-class scatter:

$$S_W = \sum_{\text{classes } c} \sum_{j \in c} p_c(\mathbf{x}_j - \boldsymbol{\mu}_c)(\mathbf{x}_j - \boldsymbol{\mu}_c)^T.$$

- Between-class scatter:

$$S_B = \sum_{\text{classes } c} (\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^T.$$

- We want a small within-class scatter and a large between-class scatter: maximise S_B/S_W

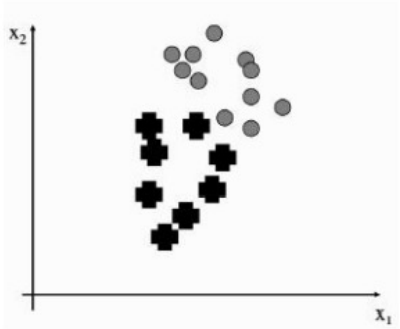

FIGURE 6.2 A set of datapoints in two dimensions, with two classes.

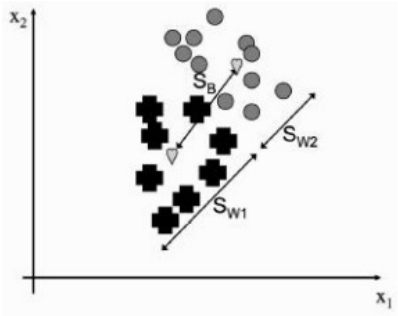

FIGURE 6.3 The meaning of the between-class and within-class scatter. The hearts mark the means of the two classes.

6.1 Linear Discriminant Analysis (LDA)

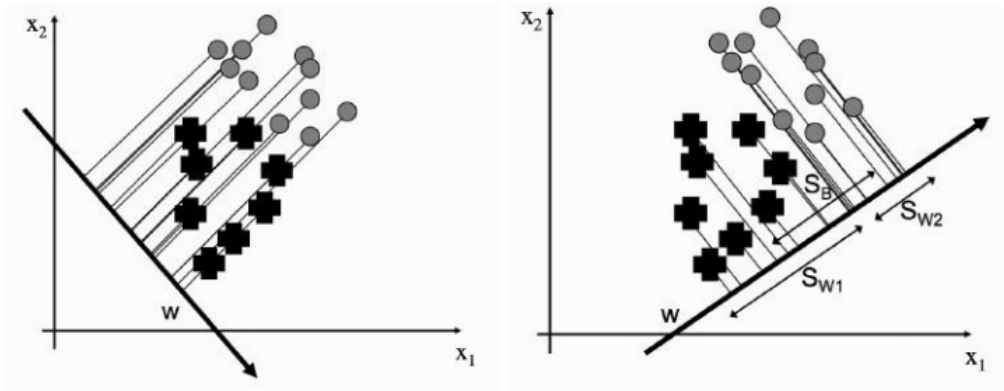

FIGURE 6.4 Two different possible projection lines. The one on the left fails to separate the classes.

- Our "weights" for LDA are the vector **w** which represents the projection line, so our projection of point **x** in the new space is $\mathbf{w}^T \cdot \mathbf{x}$
- So after transformation, the equations for between and within class scatter become:

$$\sum_{\text{classes } c} \sum_{j \in c} p_c(\mathbf{w}^T \cdot (\mathbf{x}_j - \boldsymbol{\mu}_c))(\mathbf{w}^T \cdot (\mathbf{x}_j - \boldsymbol{\mu}_c))^T = \mathbf{w}^T S_W \mathbf{w}$$

$$\sum_{\text{classes } c} \mathbf{w}^T (\boldsymbol{\mu}_c - \boldsymbol{\mu})(\boldsymbol{\mu}_c - \boldsymbol{\mu})^T \mathbf{w} = \mathbf{w}^T S_B \mathbf{w}.$$

- $$\frac{S_B \mathbf{w}(\mathbf{w}^T S_W \mathbf{w}) - S_W \mathbf{w}(\mathbf{w}^T S_B \mathbf{w})}{(\mathbf{w}^T S_W \mathbf{w})^2} = 0. \quad \Rightarrow \quad S_W \mathbf{w} = \frac{\mathbf{w}^T S_W \mathbf{w}}{\mathbf{w}^T S_B \mathbf{w}} S_B \mathbf{w}.$$

6.1 Linear Discriminant Analysis (LDA)

```
C = np.cov(np.transpose(data))

# Loop over classes
classes = np.unique(labels)
for i in range(len(classes)):
    # Find relevant datapoints
    indices = np.squeeze(np.where(labels==classes[i]))
    d = np.squeeze(data[indices,:])
    classcov = np.cov(np.transpose(d))
    Sw += np.float(np.shape(indices)[0])/nData * classcov

Sb = C - Sw
# Now solve for W and compute mapped data
# Compute eigenvalues, eigenvectors and sort into order
evals,evecs = la.eig(Sw,Sb)
indices = np.argsort(evals)
indices = indices[::-1]
evecs = evecs[:,indices]
evals = evals[indices]
w = evecs[:,:redDim]
newData = np.dot(data,w)
```

6.2 Principal Component Analysis (PCA)

- Unlike LDA, this does not incorporate the labels. Hence, it can be used for labelled data, but will not use the information provided by the labels.
- The idea is to find a new set of axes to represent the data:

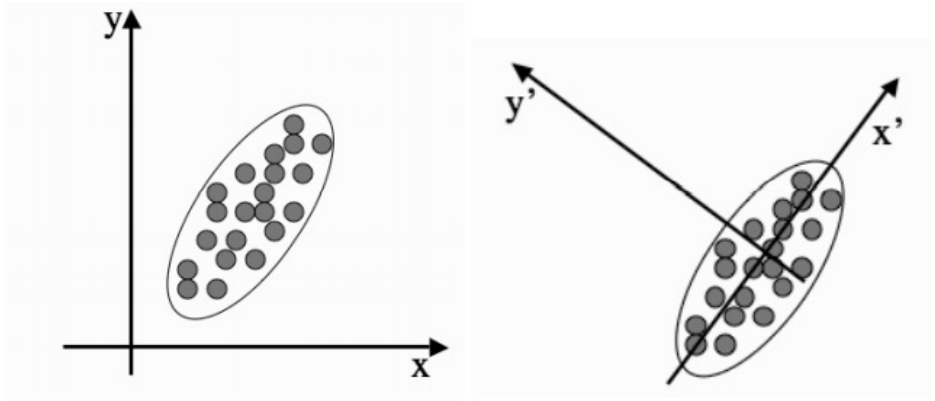

FIGURE 6.6 Two different sets of coordinate axes. The second consists of a rotation and translation of the first and was found using Principal Components Analysis.

- PCA is just the first way to find these new axes that we're going to look at.
- "Principal component" refers to the direction in the data with the largest variation

6.2 Principal Component Analysis (PCA)

- To have our data lying on the direction of maximum variation, we need to rotate it. Thus our data matrix, **X**, is rotated by a rotation matrix, **P**, and the final data is $\mathbf{Y} = \mathbf{P}^T\mathbf{X}$,

The Principal Components Analysis Algorithm

- Write N datapoints $\mathbf{x}_i = (\mathbf{x}_{1i}, \mathbf{x}_{2i}, \ldots, \mathbf{x}_{Mi})$ as row vectors
- Put these vectors into a matrix $\mathbf{X}$ (which will have size $N \times M$)
- Centre the data by subtracting off the mean of each column, putting it into matrix $\mathbf{B}$
- Compute the covariance matrix $\mathbf{C} = \frac{1}{N}\mathbf{B}^T\mathbf{B}$
- Compute the eigenvalues and eigenvectors of $\mathbf{C}$, so $\mathbf{V}^{-1}\mathbf{C}\mathbf{V} = \mathbf{D}$, where $\mathbf{V}$ holds the eigenvectors of $\mathbf{C}$ and $\mathbf{D}$ is the $M \times M$ diagonal eigenvalue matrix
- Sort the columns of $\mathbf{D}$ into order of decreasing eigenvalues, and apply the same order to the columns of V
- Reject those with eigenvalue less than some η, leaving L dimensions in the data

6.2 Principal Component Analysis (PCA)

```
def pca(data,nRedDim=0,normalise=1):

    # Centre data
    m = np.mean(data,axis=0)
    data -= m

    # Covariance matrix
    C = np.cov(np.transpose(data))

    # Compute eigenvalues and sort into descending order
    evals,evecs = np.linalg.eig(C)
    indices = np.argsort(evals)
    indices = indices[::-1]
    evecs = evecs[:,indices]
    evals = evals[indices]

    if nRedDim>0:
        evecs = evecs[:,:nRedDim]

    if normalise:
        for i in range(np.shape(evecs)[1]):
            evecs[:,i] / np.linalg.norm(evecs[:,i]) * np.sqrt(evals[i])

    # Produce the new data matrix
    x = np.dot(np.transpose(evecs),np.transpose(data))
    # Compute the original data again
    y=np.transpose(np.dot(evecs,x))+m
    return x,y,evals,evecs
```

6.2.2 Kernel PCA

- One problem with PCA is that it's merely a linear transformation. The direction of greatest variation may not be a straight line.
- To get around this problem, we can use kernel PCA. This is similar to the trick used by SVM.
- Now the covariance matrix will be:

$$\mathbf{C} = \frac{1}{N} \sum_{n=1}^{N} \Phi(\mathbf{x}_n) \Phi(\mathbf{x}_n)^T,$$

The Kernel PCA Algorithm

- Choose a kernel and apply it to all pairs of points to get matrix $\mathbf{K}$ of distances between the pairs of points in the transformed space
- Compute the eigenvalues and eigenvectors of $\mathbf{K}$
- Normalise the eigenvectors by the square root of the eigenvalues
- Retain the eigenvectors corresponding to the largest eigenvalues

6.2.2 Kernel PCA

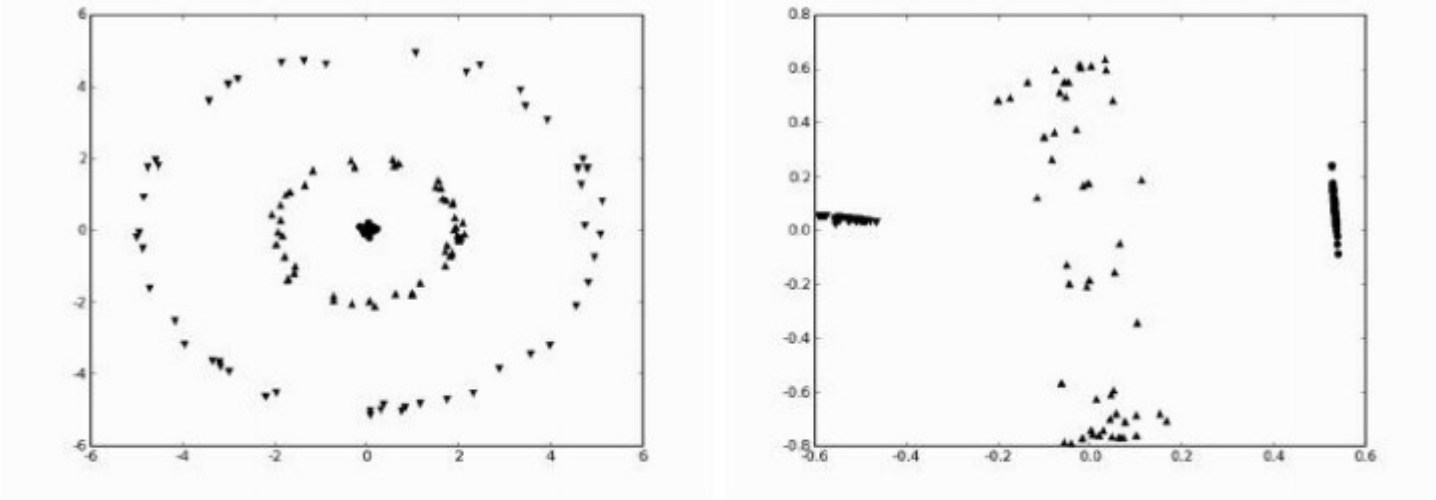

FIGURE 6.10 A very definitely non-linear dataset consisting of three concentric circles, and the (Gaussian) kernel PCA mapping of the iris data, which requires only one component to separate the data.

6.3 Factor Analysis

- For factor analysis, we hypothesise that the data comes from an underlying data source that isn't known, and can be explained by a smaller number of uncorrelated latent variables.
- Factor analysis relies on an expectation-maximisation (EM) algorithm, which is explained in chapter 7 of the book, so full detail is not given here.

Suppose that we have a dataset in the usual $N \times M$ matrix $\mathbf{X}$, i.e., each row of $\mathbf{X}$ is an M-dimensional datapoint, and $\mathbf{X}$ has covariance matrix $\mathbf{\Sigma}$. As, with PCA, we centre the data by subtracting off the mean of each variable (i.e., each column): $\mathbf{b}_j = \mathbf{x}_j - \boldsymbol{\mu}_j$, $j = 1 \ldots M$, so that the mean $E[\mathbf{b}_i] = 0$. Which we've done before, for example for the MLP and many times since.

We can write the model that we are assuming as:

$$\mathbf{X} = \mathbf{WY} + \boldsymbol{\epsilon}, \tag{6.20}$$

The covariance matrix of the original data, $\mathbf{\Sigma}$, can now be broken down into $\text{cov}(\mathbf{Wb} + \boldsymbol{\epsilon}) = \mathbf{WW}^T + \Psi$, where Ψ is the matrix of noise variances and we have used the fact that $\text{cov}(\mathbf{b}) = \mathbf{I}$ since the factors are uncorrelated.

- Then incremental updates are performed:

$$\begin{aligned} \mathbf{W}_{new} &= \left(\mathbf{y}E(\mathbf{x}|\mathbf{y})^T\right)\left(E(\mathbf{x}\mathbf{x}^T|\mathbf{y})\right)^{-1}, \\ \Psi_{new} &= \frac{1}{N}\text{diagonal}\left(\mathbf{x}\mathbf{x}^T - WE(\mathbf{x}|\mathbf{y})\mathbf{y}^T\right), \end{aligned}$$