Week 5 - ML

Start Date	@1 December 2025
Weeks	Week 5

Data Compression With Dimensionality Reduction - Chapter 5

- Alternative approach to feature selection for dimensionality reduction → Feature extraction
- Data compression important in machine learning → helps store + analyse increasing amount of data that are produced and collected

Unsupervised dimensionality reduction via component analysis

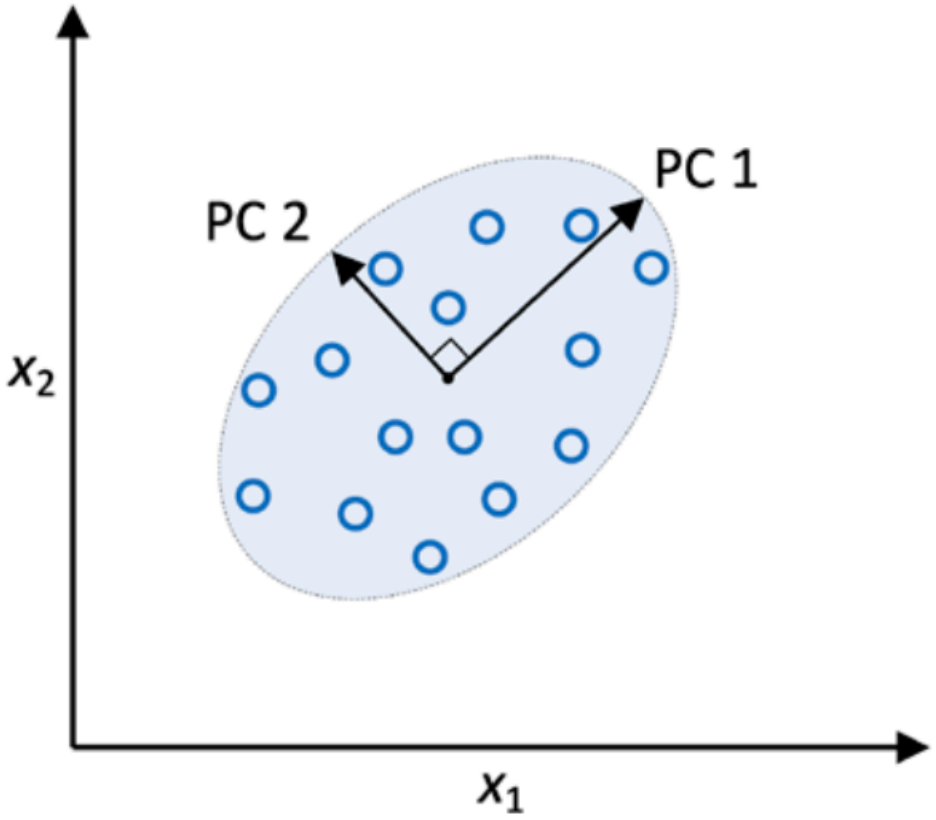
- Feature extraction → transform or project data onto new feature space
- Context of dimensionality reduction → Feature extraction → Data compression with goal to maintain most of the relevant information
    - Not only used to improve storage space or computational efficiency → used to improve predictive performance by reducing Curse of Dimensionality

Principal Component Analysis (PCA)

- PCA → Unsupervised linear transformation technique → used in many different fields → mainly feature extraction + dimensionality reduction
- Some popular application of PCA → Exploratory data analysis, de-noising signals in stock market trading, analysis of genome data + genome expression levels
- PCA → Helps us identify patterns based on correlation between features
    - Looks at high-dimensional data, then asks: "In which directions do the points spread out the most?" → Directions where data varies the most

- Then:
    - Builds new axis (new features) along “maximum spread” directions
    - Keeps only some of those axes (often first few) → end up with fewer dimensions than started with
    - Makes sure new axes are at right angles to each other (orthogonal) → new feature captures a different, non-overlapping aspect of variation

SHORT VERSION: PCA rotates + compresses data into fewer, uncorrelated dimensions that capture as much of original variation as possible



Using PCA to find the directions of maximum variance in a dataset

- x_1, x_2 → Original feature axes → PC1, PC2 → Principal components
- PCA for dimensionality reduction → $d \cdot x$ - dimensional transformation matrix, W

- Allows us to map vector of feature examples, x to new k-dimensional feature subspace → Fewer dimensions than original d-dimensional feature space

$$x = [x_1, x_2, \cdots x_d], x \in R^d$$

Transformed by transformation matrix, $W \in R^{d \cdot k}$

$$xW = z$$

Output vector:

$$x = [z_1, z_2, \ldots, z_k] = z \in R^k$$

- Result of transforming → first principal component will have largest variance
  - **PC1** captures the most information/variation in data → if you look at one axis only → the spread of points is as large as possible
- After **PC1**, PCA looks for second direction (**PC2**) → but, **PC2** must be right angle (orthogonal) to PC1 → so PC1 and PC2 don't overlap.
- This can keep going → **PC3** → has to be orthogonal to PC1 + PC2

> ⭐ **PCA directions → sensitive to scaling → Standardise first**

Approach Summarised:

1. Standardise d-dimensional dataset
2. Construct covariance matrix
3. Decompose covariance matrix into eigenvectors + eigenvalues
4. Sort eigenvalues by decreasing order → to rank corresponding eigenvectors
5. Sort eigenvectors → corresponds with k largest eigenvalue → k is dimensionality of new features subspace $K \leq d$
6. Construct projection matrix, W from "top" k eigenvectors

7. Transform d-dimensional input dataset, X, using projection matrix, W → obtain new k-dimensional feature subspace

Eigendecomposition

- Factorisation of square matrix into eigenvalues + eigenvectors → core of PCA procedure
- Covariance matrix - $A = A^T$
- Decomposed symmetric vector → Eigenvalues are real numbers + eigenvectors are orthogonal to each other
- Decomposed covariance matrix → Eigenvectors associated with highest eigenvalue corresponds to direction of maximum variance

Four Steps of PCA:

1. Standardise the data
2. Constructing the covairance
3. Obtaining the eigenvalues + eigenvectors of covariance matrix
4. Sorting the eigenvalues by decreasing order to rank the eigenvectors

Step 2 of summarised approach

- d x d-dimensional covariance matrix, d = number of dimensions in the dataset
- Covariance between two features, x_j and x_k → on population level:

$$\sigma_{jk} = \frac{1}{n-1}\sum_{i=1}^{n}(x_j^{(i)} - \mu_j)(x_k^{(i)} - \mu_k)$$

- μ_j, μ_k = sample means of feature j, k
  - Sample mean = zero → if dataset is strandardised
- Positive covariance between two features = features increase + decrease together
- Negative covariance → features vary in opposite directions
- Covariance matrix of three features:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_2^2 & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_3^2 \end{bmatrix}$$

- Covariance matrix represents principal component: (direction of maximum variance)
    - Eigenvalues will define their magnitude

Step 3 of summarised approach

- Obtain eigenpairs for covariance matrix
- Eigenvector v:

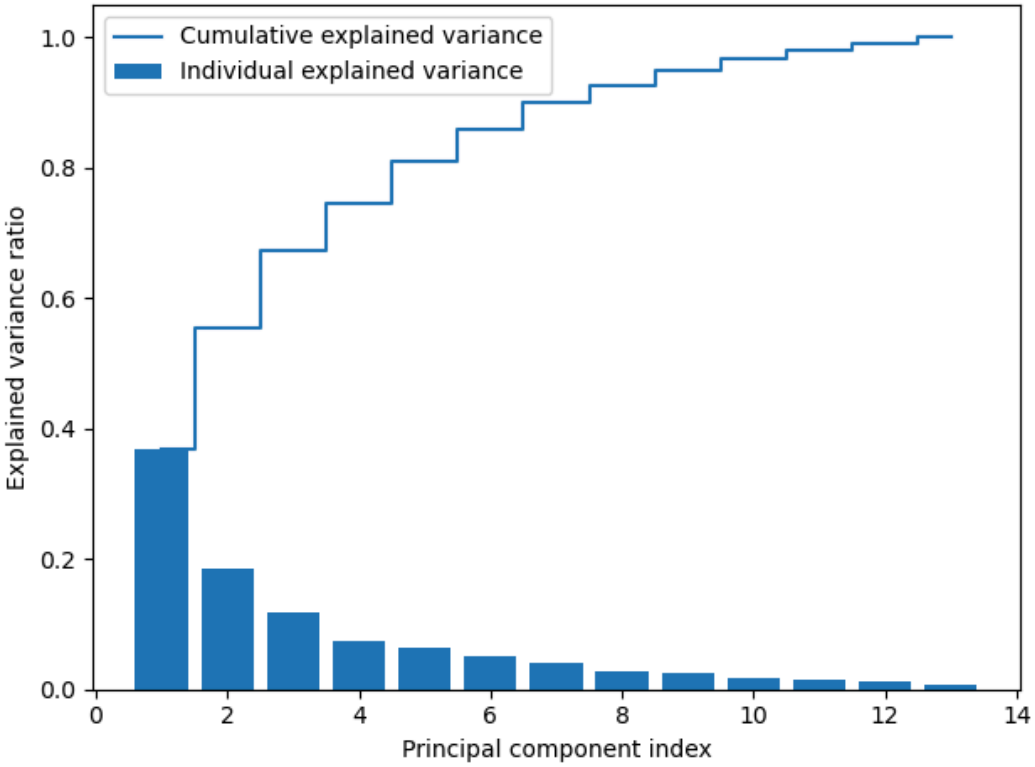
$$\Sigma = \lambda v$$

- λ = Scalar → eigenvalue

Total and explained variance

- Only select subset of eigenvectors (principal component) that contains most of the information (variance)
- Eigenvalues define magnitude → sort by decreasing magnitude → interested in top $k$eigenvectors
- Variance explained ratio

$$\frac{\lambda_j}{\sum_{j=1}^{d} \lambda_j}$$



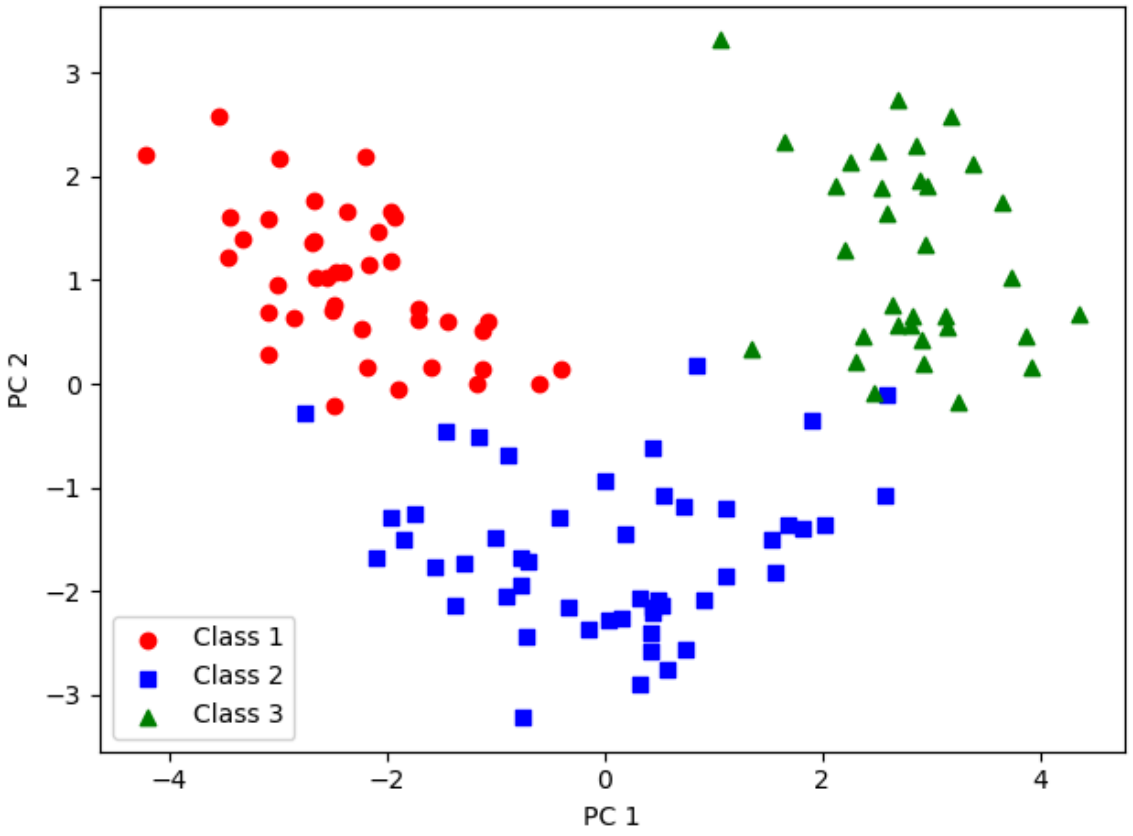
The proportion of the total variance captured by the principal components

- First principal → accounts to approx. **40%** of variance
- First two principal components → combined almost **60%** of variance

Feature Transformation

PCA Step 5, 6, 7 of summarised approach

- Sort eigenpairs by descending order of eigenvalues
- Construct projection matrix from selected eigenvectors
- Projection matrix to transform data onto lower-dimensional subspace



Data records from the Wine dataset projected onto a 2D feature space via PCA

- Data more spread along PC1 → Consistent with variance ratio plot (from above)

> ⭐ PCA is a unsupervised technique that doesn't use any class label information

Assessing feature contribution

- PCA → create principal components → represent linear combinations of features
- **Loading** → How much each original feature contributions to a given principal component
  - Factor loadings computed → Scaling eigenvectors by square root of eigenvalues
  - **Result:** **Correlation between original features + principal components**

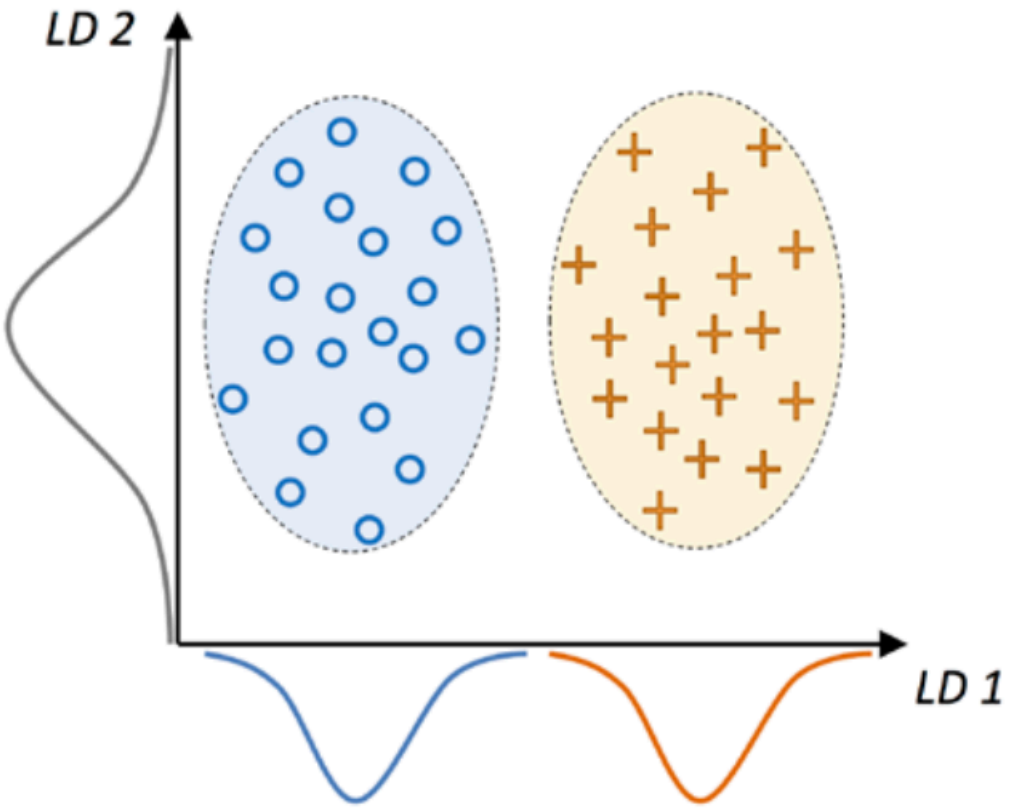
Supervised Data Compression With Linear Discriminant Analysis (LDA)

⭐ Linear Discriminant Analysis (LDA) → Linear transformation technique that takes class label information into account

- General concept → Similar to PCA → LDA finds the feature subspace that optimises class separability

💡 Fisher LDA

- **LDA sometimes called Fisher LDA →** Ronald A. Fisher (1936)
  - Later → Generalised for multi-class problems by **C. Radhakrishnan Rao** → For equal class covariance + normally distributed classes.



The concept of LDA for a two-class problem

- LDA for two-class problem
  - Class 1 - Circles
  - Class 2 - Crosses
- Linear discriminant → x-axis (LD 1) → separate two normal distributed classes
- y-axis (LD 2) → variance in dataset → lot of variance in dataset
  - Would fail at good linear discriminant → doesn't capture any of class-discriminatory information
- **Assumption LDA:**
  - Data is normally distributed
  - Also assumed → classes have identical covariance matrices
  - Training examples are statistically independant.
- If assumptions are slightly violated → LDA for dimensionality reduction can still work reasonably well

LDA Workings:

1. Standardise d-dimensional dataset (d = number of feature)
2. Each class → Compute d-dimensional mean vector → LDA takes labelled information into account
3. Construct between-class scatter matrix S_B + within-class scatter matrix S_W
4. Compute eigenvectors + corresponding eigenvalue of matrix $S_W^{-1}S_B$
5. Sort eigenvalues by decreasing order → rank corresponding eigenvectors
6. Choose k eigenvectors → k largest eigenvalue → construct d x k-dimensional transformation matrix, W → eigenvector are column of matrix
7. Project examples onto new feature subspace using transformation matrix, W

⭐ **LDA is similar to PCA** → Decomposing matrices into eigenvalues + eigenvectors → For new lower-dimension feature space

Computing Scatter Matrices

- **Always standardise the features**
- Calculation of new vectors → construct within-class scatter + between-class scatter matrix

$$m_i = \frac{1}{n_i} \sum_{x \in D_i} x_m$$

- m_i = Mean vector
- μ_m = Mean feature value

$$m_i = \begin{bmatrix} \mu_i alchohol \\ \mu_i malicacid \\ \vdots \\ \mu_i proline \end{bmatrix}^T \quad i \in \{1, 2, 3\}$$

- Mean vectors → Compute within-class scatter matrix, S_W:

$$S_W = \sum_{i=1}^{c} S_i$$

- Calculated by summing up individual scatter matrices:

$$S_i = \sum_{x \in D_i} (x - m_i)(x - m_i)^T$$

- **Assumption: Class labels in training datasets are uniformly distributed.**
  - Violated → Check with class label distribution

Scale individual scatter matrices, S_i → Before we sum them up.

- Divide scatter matrices by number of class-examples, n_i
  - Scatter matrix → Same as computing covariance matrix Σ_i

- Normalised version of scatter matrix

$$\Sigma_i = \frac{1}{n_i} S_i = \frac{1}{n_i} \sum x \in D_i (x - m_i)(x - m_i)^T$$

- Next steps → compute between0class scatter matrix S_B:

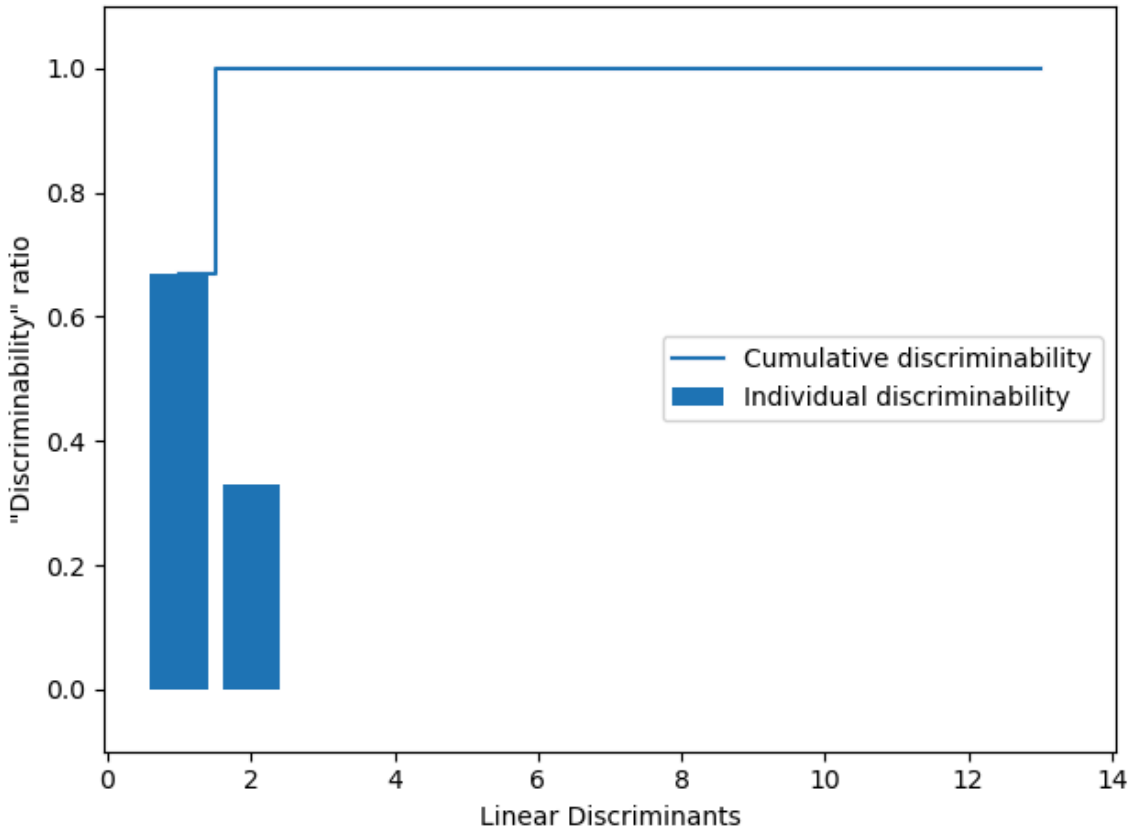
$$S_B = \sum_{i=1}^{c} n_i (m_i - m)(m_i - m)^T$$

Linear Discriminants For New Feature Subspace

- Instead of eigendecomposition → Solve generalised eigenvalue problem of matrix $S_W^i S_B$
- LDA → Number of linear discriminants at most $c - 1$, c = Number of class labels → S_B = Sum of c matrices with rank one or less

Collinearity → Perfect case (All aligned example points fall on straight line)

- Covariance matrix = rank one → eigenvector with nonzero eigenvalue



The top two discriminants capture 100 percent of the useful information

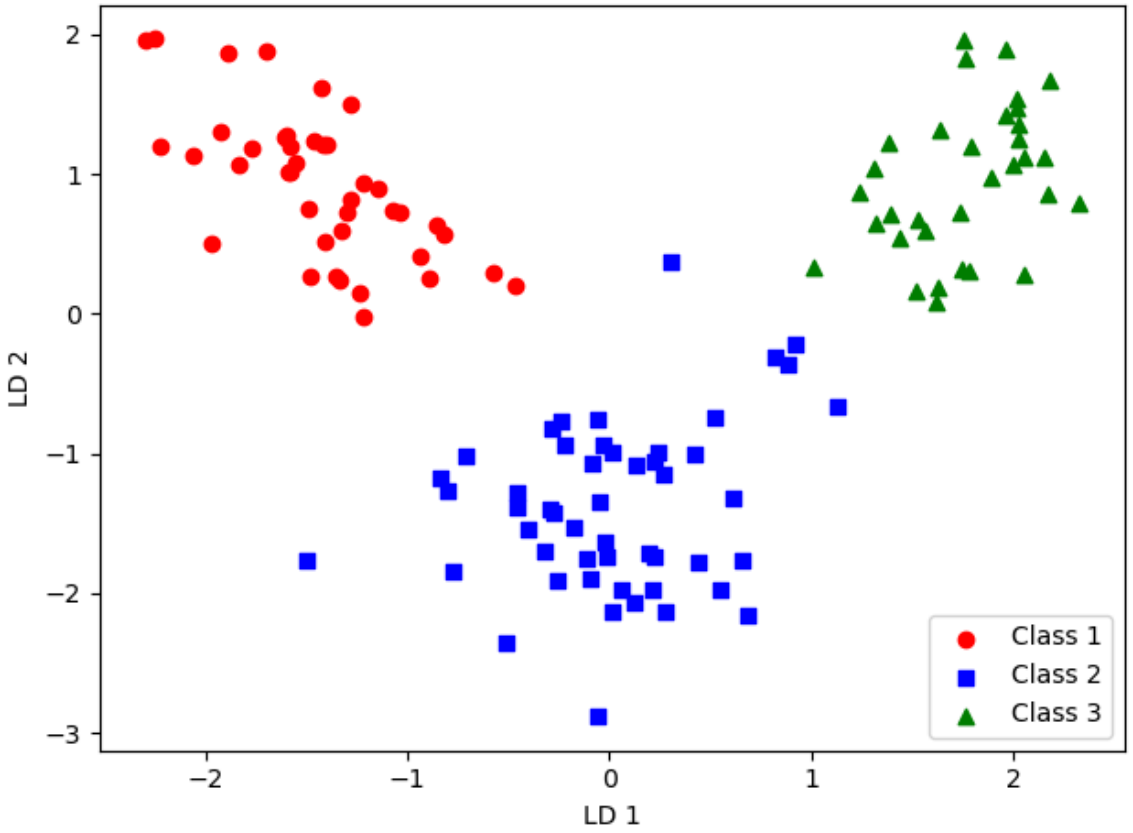
- Measure how much class-discriminatory information captured by linear discriminants (eigenvectors)
- First two linear discriminants = capture 100% of useful information

```
Matrix W:
 [[-0.1481 -0.4092]
 [ 0.0908 -0.1577]
 [-0.0168 -0.3537]
 [ 0.1484  0.3223]
 [-0.0163 -0.0817]
 [ 0.1913  0.0842]
 [-0.7338  0.2823]
 [-0.075  -0.0102]
 [ 0.0018  0.0907]
 [ 0.294  -0.2152]
 [-0.0328  0.2747]
 [-0.3547 -0.0124]
 [-0.3915 -0.5958]]
```

- Two discriminative eigenvector columns stacked → create transformation matrix W

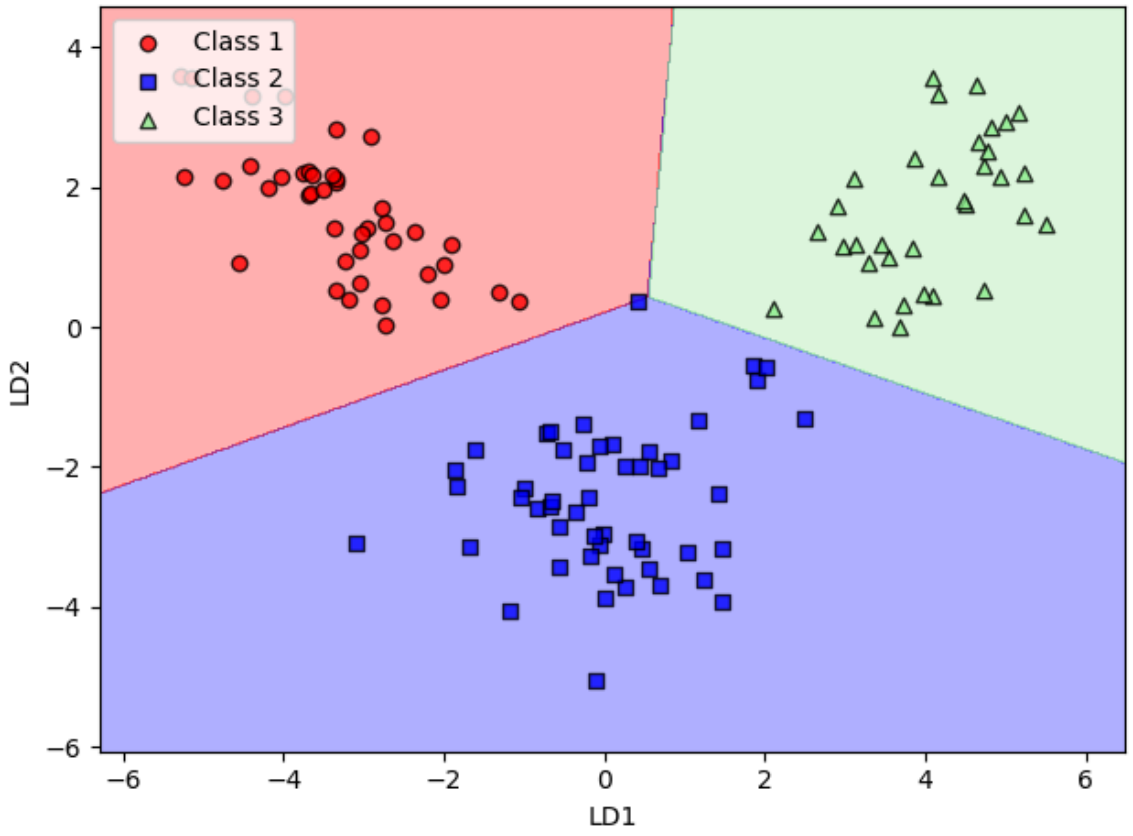
Projecting Examples Onto New Feature Subspace

- Transformation matrix W → Transform training dataset → Multiply matrices → $X' = XW$



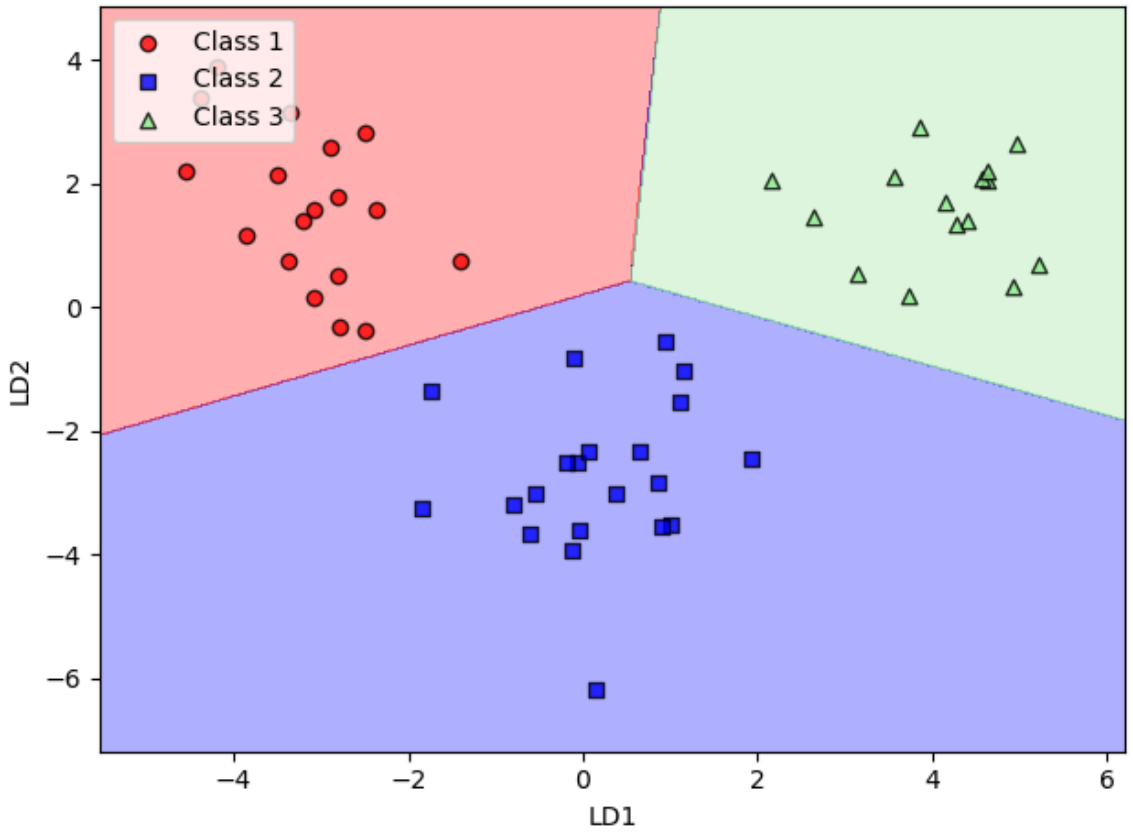
Wine classes perfectly separable after projecting the data onto the first two discriminants

- Wine classes → perfectly linearly separable in new feature subspace



The logistic regression model misclassifies one of the classes

- Lower-dimension dataset → logistic regerssion
  - Only one example is misclassified



The logistic regression model works perfectly on the test data

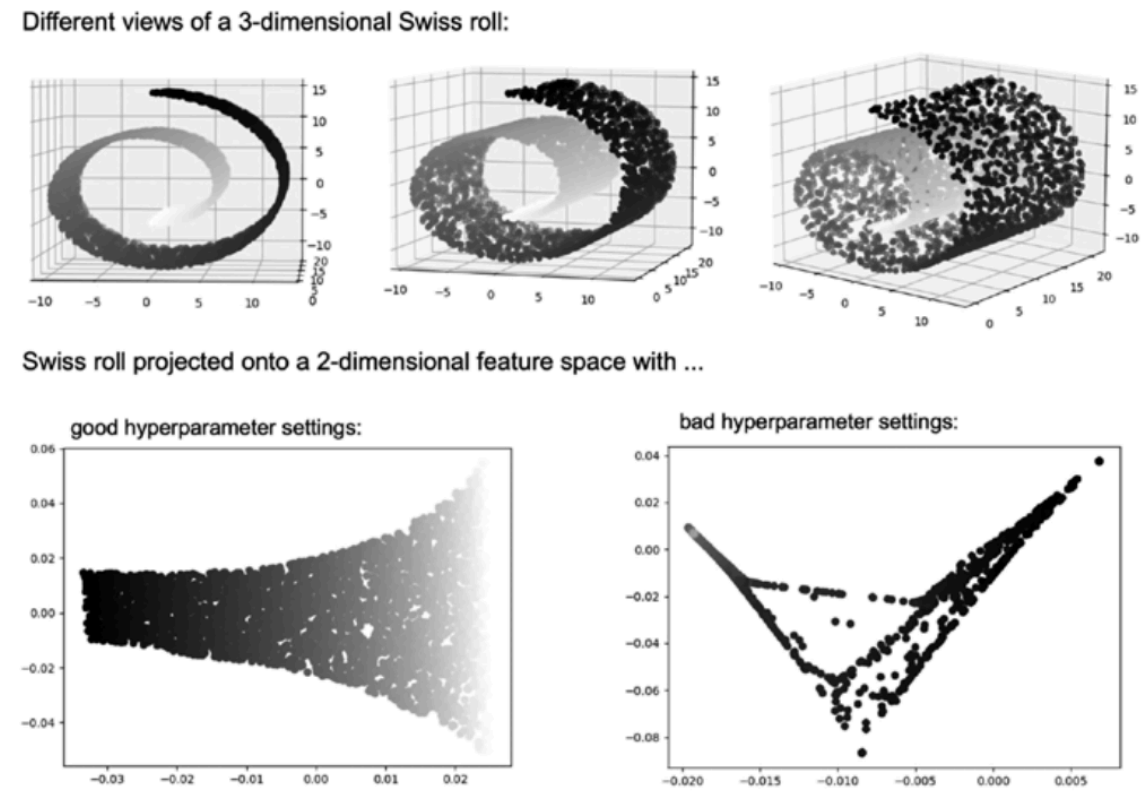
- Lower regularisation strength → shifts decision boundaries and all test data is separated perfectly

Nonlinear Dimensionality Reduction via Visualisation

- **T-distributed Neighbor Embedding (t-SNE)** → Frequently used in literature → Visualise high-dimensional datasets into two or three dimensions
- t-SNE → Plot image of handwritten images in 2-dimensional space

Why consider Nonlinear dimensionality Reduction?

- Development + application of nonlinear dimensionality reduction technique → **Manifold Learning** → Scikit-Learn library covers other complex ones too



Three-dimensional Swiss roll projected into a lower, two-dimensional space

- **Very powerful but also very hard to use**
  - Difficult → Working with high-dimensional datasets that we cannot visualise + structure is not obvious
  - Could try projecting dataset to 2 or 3 dimensions
    - Not enough for capturing complicated relationships
    - **Also for it's hard or impossible to assess quality**

Hence people rely on PCA + LDA for dimensional reduction

Visualising Data via t-Distributed Stochastic Neighbor Embedding

- In a nutshell → t-SNE modelling data points based on pair-wise distances in high-dimensional (original) feature space
  - Then → t-SNE learns to embed data points into a lower-dimensional space → pair-wise distances in original space is preserved
- t-SNE → Intended for visualisation purpose → Requires whole dataset for projection
  - Projects points directly → Cannot apply t-SNE to new data points

Uniform Manifold Approximation And Projection

- Another popular visualisation technique → UMAP
- Faster than t-SNE + produces similarly good results as t-SNE + used to project new data