Module 14: Support Vector Machines

Learning outcomes:

1. Analyse appropriate SVM methods and hyperplanes for classifying two-dimensional data.
2. Select kernel functions to improve classification performance.
3. Apply soft-margin SVMs to manage noise, outliers, and non-separable data.
4. Implement SVM strategies for multi-class classification and evaluate their feasibility in real-world contexts.
5. Implement SVMs in Python for case study analysis and capstone optimisation.

Support vector machines (SVMs)

- SVMs are supervised learning models used for classification. They find the optimal hyperplane that separates classes with the maximum margin. The data points that define this margin are called support vectors.
- SVMs can handle both linearly separable and non-linearly separable data, using different types of margins and kernel functions.

SVM types and when to use them

Type	Use cases	Notes
Hard-margin SVM	The data is perfectly separable with no noise.	There is no tolerance for misclassifications.
Soft-margin SVM	The data has noise, outliers or imperfect separation.	It allows margin violations, controlled by parameter `C`.

- ***Note:*** `C` is a regularisation parameter, where:
  - High `C` means fewer violations (risk of overfitting).
  - Low `C` means more violations allowed (risk of underfitting).

The kernel trick and kernel types

Kernels allow SVMs to model non-linear boundaries without manually transforming the data. They compute similarity in a higher-dimensional space.

Comparison of kernel types

Kernel	Use cases	Applications
Linear	High-dimensional or linearly separable data	Text classification (e.g. TF-IDF)

Polynomial	Known non-linear interactions, curved boundaries	Image recognition
RBF (Gaussian)	Complex, highly non-linear relationships	Digit recognition, bioinformatics
Sigmoid	Rarely used; mimics neural network behaviour	Some signal processing tasks

Multi-class classification with SVMs

SVMs are naturally binary classifiers. Use strategies to extend to multi-class problems.

One-vs-one and one-vs-all

Strategy	How it works	When to use it
One-vs-one	Builds a classifier for each pair of classes (k(k − 1) / 2)	Few classes, complex boundaries
One-vs-all	Builds one classifier per class vs all others (k)	Simpler tasks with fewer classes