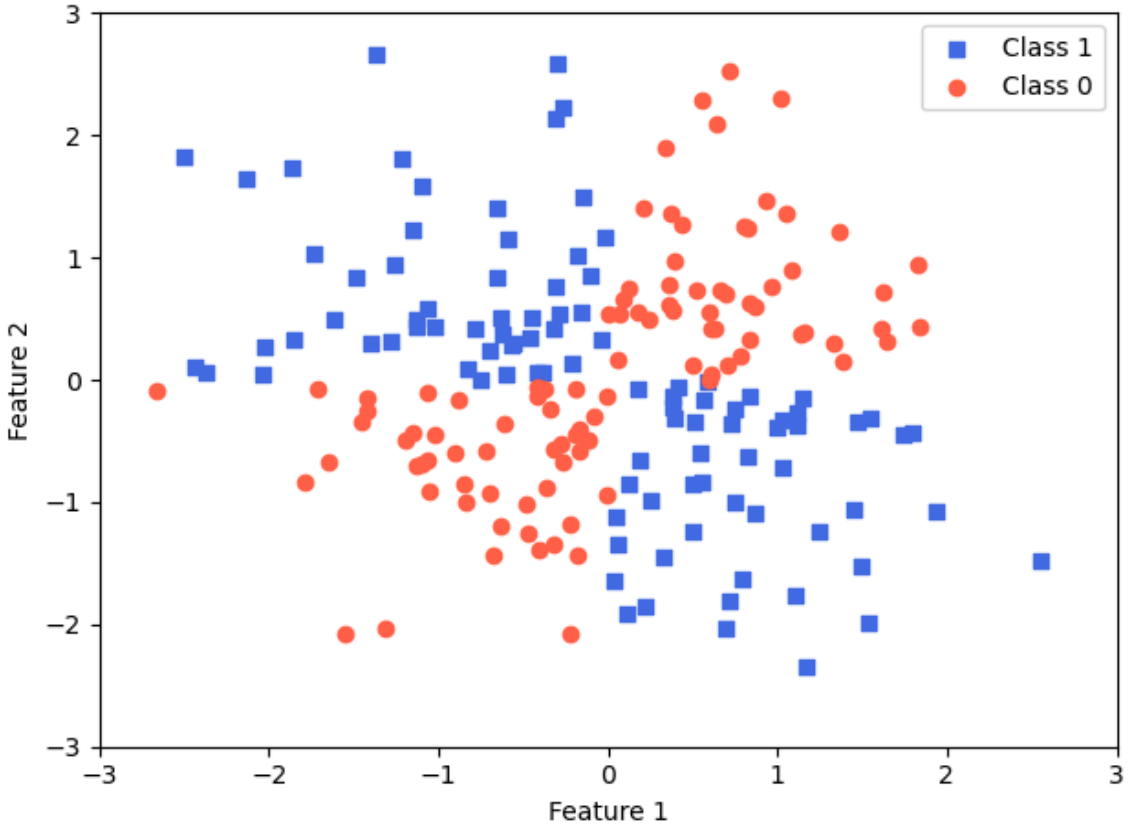
Week 3 - ML

Start Date	@17 November 2025
Weeks	Week 3

still chapter 3

Kernel Methods for Linearly Inseparable Data

- SVMs can be kernelized for non-linear classification
- Kernel SVM → Most common variant of SVM



- XOR dataset with noise

    - Can’t separate examples from negative + positive classes
- Idea of Kernel Method → Create non-linear combinations of original features to project them onto higher-dimensional space via mapping fuction (ϕ)

$$\phi(x_1, x_2) = (z_1, z_2, z_3) = (x_1 \cdot x_2, x_1^2 + x_2^2)$$

Using Kernel Trick to Find Separating Hyperplanes In A High-dimensional Space

- Non-linear problem → Mapping Function (ϕ) → Train Linear SVM to classify data in new feature space → ϕ (Mapping Function) → Transform new unseen data to classify it using Linear SVM.
- However, Mapping Function is computationally very expensive, especially high-dimensional data
    - Use Kernel Trick
- Quadratic programming to train SVM → replace dot product $x^{(i)}x^{(j)} = \phi(x^{(i)})^T\phi(x^{(j)})$
- To save expensive dot product → we define kernel function

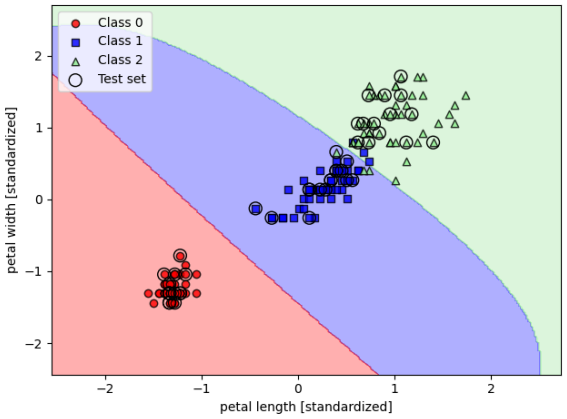
$$K(x^{(i)}, x^{(i)}) = \phi(x^{(i)})^T\phi(x^{(j)})$$

- One of the most widely used Kernels → **Radial Basis Function (RBF)** → Called **Gaussian Kernel**

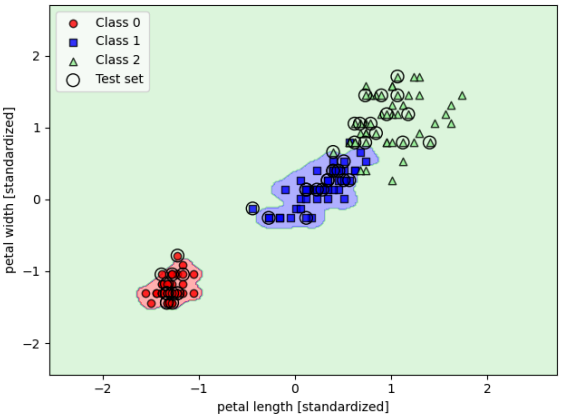
$$K(x^{(i)}, x^{(i)}) = exp\left(-\frac{||x^{(i)} - x^{(j)}||^2}{2\sigma^2}\right)$$

Simplified → $K(x^{(i)}, x^{(i)}) = exp(-\gamma||x^{(i)} - x^{(j)}||^2)$

- $\gamma = \frac{1}{2\sigma^2}$ → Free parameter to be optimised
- Kernel → Interpreted as similarity function between pair of examples
- Minus sign → Inverts distance into similarity score → falls between range (0,1)



γ is small → RBF Kernel SVM model relatively soft



γ is high → Fits data but will have high generalization error

> ⭐ γ plays an important role in controlling overfitting or variance when algorithm is too sensitive to fluctuation in training dataset

Decision Tree Learning

- Decision Tree Classifier → Attractive model when we care about interpretability
    - Break down data by making decisions based on questions → shape like a tree
- Model learns class labels from features by series of questions asked
- Start at root → split data on feature that results in largest Information Gain (IG)
- Training examples in each node belong to the same class
    - Splitting happens until all leaves are “pure”
- We want to prune tree by setting limits → tree can become very deep → prone to overfitting

Maximising Information (IG)

- Splitting tree at most informative features → define objective function → optimise algorithm

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right})$$

- Three impurity measures or splitting criteria used in binary decision trees:
  - Gini Impurity (I_G)
  - Entropy (I_n)
  - Classification Error (I_E)

Entropy

- For all non-empty classes ($P(i|t) \neq 0$)

$$I_H(t) = -\sum_{i=1}^{c} P(i|t) log_2 P(i|t)$$

- $P(i|t)$ = proportion of examples that belong to i for particular node, t
- Entropy = 0 if all examples at a node belong to same class
- Entropy = maximal - uniform class distribution

Gini Impurity

- Criterion to minimise the probability of misclassification:

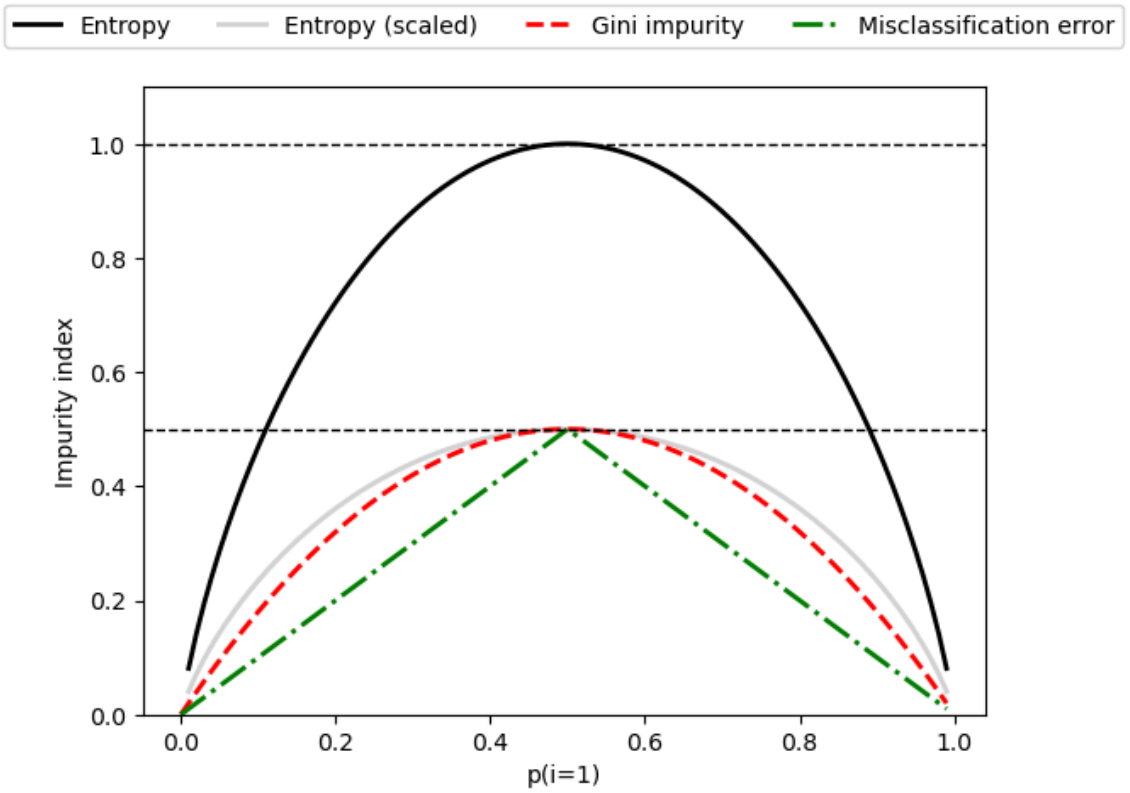
$$I_G(t) = \sum_{i=1}^{c} P(i|t)(1 - P(i|t) = 1 - \sum_{i=1}^{c} P(i|t)^2$$

- Gini Impurity = maximal if all the classes are perfectly mixed
- In practice both Gini Impurity and Entropy → Typically yield very similar results
- Not worth evaluating trees using different impurity criteria → Experiment rather with different pruning cut-offs

Classification Error

$$I_E(t) = 1 - max\{P(i|t)\}$$

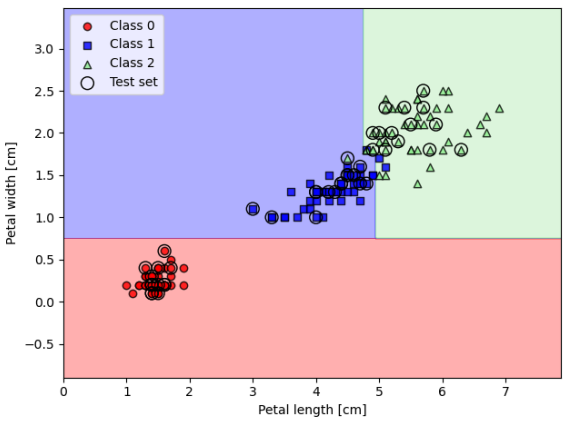
- Useful for purning → not recommended for growing decision trees → less sensitive to changes in class probabilities of the node

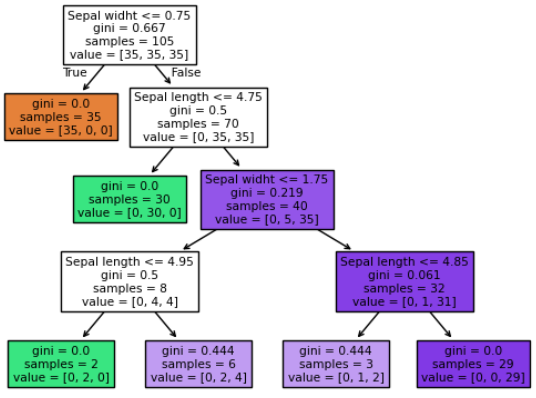


- Different impurity indicies for different class-membership probabilities in range 0,1

Building a Decision Tree (Scikit-Learn)

- Decision Trees can build complex decision boundaries → dividing feature space into rectangles





- Scikit-Learn Decision Tree → max depth of 4, Gini Impurity
- Feature scaling not required for decision trees
- Can visualise decision tree model after training with Scikit-Learn

Combining Multiple Decision Trees via Random Forest

- Ensemble method → became popular → Good classification performance + Robustness towards overfitting
- Random Forest → Good Scalability + Easy to use
  - Ensemble of Decision Trees
  - Idea behind Random Forest

> ⭐ Average multiple (deep) Decision Trees that individually suffer from high variance
> Build a more robust model → better generalisation performance + less susceptible for overfitting

Four Simple Steps:

1. Draw random bootstrap sample size n (randomly chose a n examples from training dataset with replacement)
2. Grow Decision Tree for bootstrap sample
   - Each node:
     a. Randomly select d features without replacement
     b. Split node using Objective Function
3. Repeat 1-2 k times
4. Aggregate predictions of each tree → majority voting

With and Without Replacement

- With → Chosen feature is put back → can be chosen multiple times
- Without → Chosen feature not put back → feature can't be drawn again

- Random Forest → Advantage → Don't have to worry about choosing good hyperparameters
  - However → Doesn't offer same level of interpretability as decision tree
- Don't need to **prune** Random Forest

> ⭐ Only parameter to case about → number of trees, k → larger number of trees = better performance of random forest classifier → at cost of increased computational cost

- Some hyperparameters can be optimised (less common in practice) → Size, n + number of features, d
  - n controls bias-variance tradeoff

Bootstrap sample size

- Increasing bootstrap sample → increased diversity among individual trees
- Shrinking bootstrap sample → increase randomness → reduce overfitting → But, lower overall performance

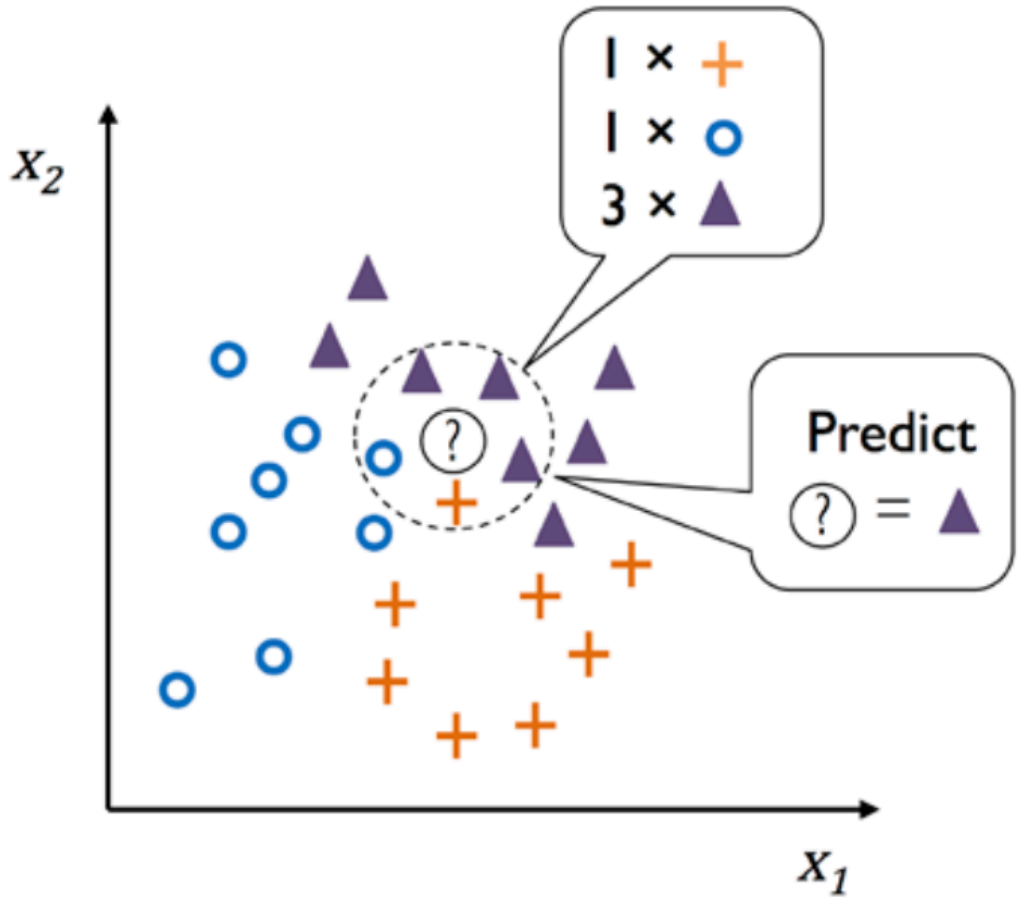
K-Nearest Neighbors

- K-Nearest Neighbor (KNN) classifier → 'lazy' → not because of its simplicity
  - KNN doesn't learn a discriminative function, memorises training dataset instead

Parametric vs. Non-parametric models

- Parametric model → Estimate parameters from training data → learn function to classify new data points (without need of original training set)
  - **Examples:** Perceptron, Logistic Regression, Linear SVM
- Non-Parametric → Can't be characterised by a fixed set of parameters + number of parameters change with amount of training data
  - **Examples:** Decision Tree Classifier/Random Forest, Kernel SVM

- KNN → Sub-category of non-parametric models → Instance based learning
  - Memorise datasets
  - Lazy learning in special case → zero cost during learning process



KNN Algorithm in 3 Steps:

1. Choose number of k + distance metric
2. Find K-Nearest Neighbor of data that we want to classify
3. Assign new class by majority vote

- Finds k examples in training dataset that are closest to the point we want to classify
- Class label of data point → determined by majority vote

Advantages + Disadvantages of Memory-based Approach

- **Advantage:** New training data → classifier adapts asap.
- **Disadvantage:** Computational complexity grows linearly with number of examples in training dataset → limited storage capabilities
- Efficient data structures for memory based approach = K-D Tree, Ball Tree

> ⭐ Most of the times, we'll work with small-medium sized datasets
>
> - Memory based approach → Good choice for real-world problems

- **In case of ties** → Scikit-Learn → Prefer neighbor with closer distance to data record to be classified
  - If it's similar lengths → What comes first in order will be chosen first
- Important to find good balance for k
  - Euclidean Distance measure real-value examples
    - Have to standardise data so that each feature contributes equally to distance
- **Minkowski Distance** → Generalisation of Euclidean + Manhattan Distance

$$d(x^{(i)}, x^{(j)}) = \sqrt[p]{\sum k |x_k^{(i)} - x_k^{(j)}|^p}$$

- If p = 1 ⇛ Euclidean
- If p = 2 ⇛ Manhattan

Curse of Dimensionality

- KNN is very susceptible to overfitting → **Curse of Dimensionality**
- Feature space becomes increasingly sparse for increasing number of dimensions of a fixed dataset
- Even your closest neighbors are far away
- Use feature selection + Dimensionality reduction → Avoid curse of dimensionality