1 Introduction to K-Nearest Neighbours (KNN)

The KNN algorithm is a non-parametric (no strong assumptions), lazy learning algorithm used for classification and regression tasks in supervised learning. Unlike model-based algorithms that learn explicit functional forms, KNN operates by memorising the training dataset and making predictions based on the similarity of new instances to known instances.

1.1 KNN Classification and Regression

The **KNN classifier** attempts to estimate the conditional distribution of Y given X and classifies a test observation x_0 to the class with the highest estimated probability. Given a positive integer K and a test observation x_0, the KNN classifier first identifies the K points in the training data that are closest to x_0. Let N_0 represent these neighbours. It then estimates the conditional probability for class j as the fraction of points in N_0 whose response values equal j:

$$\Pr(Y = j|X = x_0) = \frac{1}{K}\sum_{i\in N_0} I(y_i = j)$$

Finally, KNN classifies the test observation x_0 to the class with the largest probability from the above equation.

KNN regression is employed when the target variable is continuous rather than categorical. The goal is to predict a continuous value for the target variable Y based on the predictor variables X. The process for KNN regression is similar to KNN classification, with the following adjustments:

1. **Identify Neighbours**: It still identifies the K points in the training data that are closest to x_0.
2. **Calculate Predicted Value**: Instead of estimating probabilities, KNN regression computes the predicted value for Y as the average (or weighted average) of the Y values of its K nearest neighbours:

   $$\hat{Y}(x_0) = \frac{1}{K}\sum_{i\in N_0} y_i$$

   where y_i represents the response value of the i-th neighbour.
3. **Regression**: The predicted value $\hat{Y}(x_0)$ is then assigned as the prediction for the test observation x_0.

In summary, while KNN classification estimates probabilities for categorical outcomes and assigns the class with the highest probability, KNN regression estimates the continuous value of the target variable by averaging the values of its nearest neighbours. Both methods rely on the concept of proximity in the feature space to make predictions.

1.2 Distance Metrics

The choice of distance metric is crucial in the KNN algorithm. Apart from the **Euclidean distance**, other commonly used distance metrics include:

1. **Manhattan Distance (City Block Distance)**:

$$d(x_i, x_j) = \sum_{l=1}^{d} |x_i^{(l)} - x_j^{(l)}|$$

2. **Chebyshev Distance**:

$$d(x_i, x_j) = \max_l |x_i^{(l)} - x_j^{(l)}|$$

3. **Minkowski Distance**:

$$d(x_i, x_j) = \left(\sum_{l=1}^{d} |x_i^{(l)} - x_j^{(l)}|^p \right)^{\frac{1}{p}}$$

   where p is a positive real number. When $p = 1$, it reduces to the Manhattan distance; when $p = 2$, it is the Euclidean distance.

1.3 Selection of K

Selecting the parameter K in KNN involves a nuanced consideration of the bias-variance trade-off. A small K value provides a highly flexible fit with low bias but high variance (i.e., over-fitting), as predictions rely heavily on individual observations. Conversely, larger K values yield a smoother and more stable fit by averaging predictions from multiple neighbours, reducing variance but potentially introducing bias (under-fitting) by over-smoothing the data and masking underlying structures. Therefore, determining the optimal K requires a balance between flexibility and stability, tailored to the dataset's characteristics and the desired trade-off between bias and variance. Experimentation, cross-validation, and a deep understanding of the dataset's dynamics are essential in selecting the most appropriate K value for effective model generalisation and accurate capturing of underlying patterns.

KNN is a simple yet effective algorithm for both classification and regression tasks. Its non-parametric nature makes it particularly useful in scenarios where the underlying data distribution is complex or unknown. However, it is important to carefully select the distance metric and parameter K to achieve optimal performance.

For further reading: *Curse of dimensionality* - Reading material: Cornell CS4780 2018

2 Introduction to Decision Trees

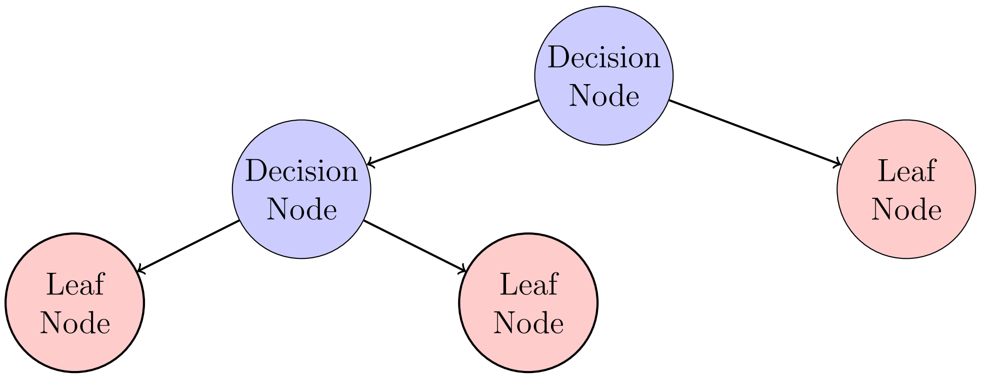
Recommended preparatory materials: *k-dimensional tree* - CMU CMSC 420 Prof. Carl Kingsford and Cornell CS4780 2018

Decision trees are often deemed as sub-optimal machine learning algorithms, particularly when used in isolation, for both classification and regression tasks in machine learning and statistics. Unlike model-based algorithms, which learn explicit functional forms, decision trees take a different approach. They operate by recursively partitioning the feature space based on feature values, resulting in a hierarchical structure of nodes and directed edges. In contrast to k-dimensional trees (KD trees) that partition data into equal sizes, decision trees partition the feature space based on various criteria, such as purity measures like *Gini impurity* or *entropy*. These criteria help in determining the best feature and threshold combination for splitting the data at each node.

However, the standalone use of decision trees often leads to issues with *bias and variance*. Decision trees tend to suffer from high variance, which can result in over-fitting to the training data, and they may also exhibit bias due to the simplicity of their decision boundaries. To address these problems, practitioners often resort to ensemble methods like *boosting* and *bagging*. Boosting techniques, such as AdaBoost and Gradient Boosting, combine multiple weak learners (typically shallow decision trees) to produce a strong learner with improved predictive performance. Boosting focuses on reducing bias by emphasising the misclassified instances in subsequent iterations. On the other hand, bagging methods like Random Forests aggregate predictions from multiple independently trained decision trees. Bagging reduces variance by averaging predictions across diverse models, thereby stabilising the overall prediction and mitigating over fitting.

2.1 Basic Concepts

- A decision tree embodies a series of decisions, where each internal node represents a decision based on a feature, and each leaf node represents a class label or a numerical value.



- The tree is constructed using a top-down, greedy approach, aiming to maximise certain criteria at each node, such as purity or information gain.
- The splitting process continues until certain stopping criteria are met, such as

reaching a maximum depth or having nodes with a minimum number of samples.

2.2 Classification Trees

For classification tasks, given a set of N observations $\{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, where $x_i \in X$ and $y_i \in Y$, the decision tree partitions the feature space X into disjoint regions $R_1, R_2, \ldots, R_J$.

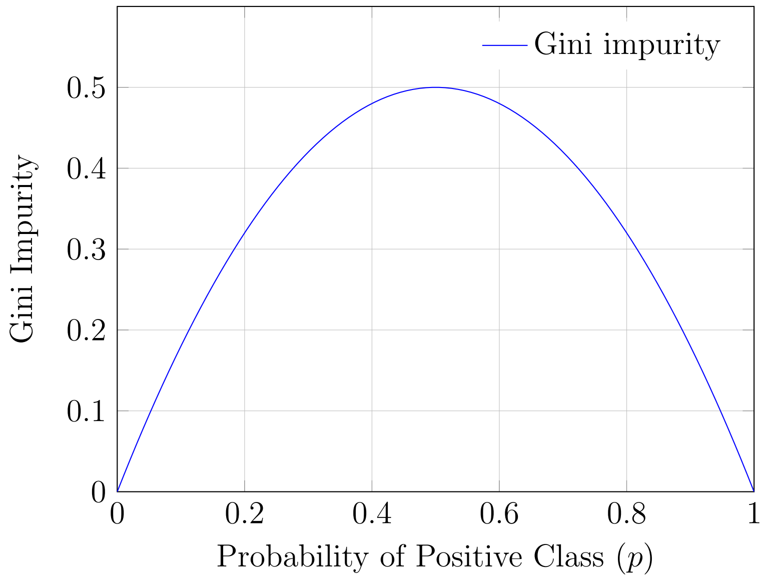
At each internal node m, a feature X_j and a threshold t_m are selected to split the data into two child nodes m_{left} and m_{right}.

The splitting criterion aims to maximise the purity of the resulting subsets. Common impurity measures include:

- **Gini impurity:** $Gini(p) = \sum_{k=1}^{K} p_k(1 - p_k)$, where p_k is the proportion of class k observations in the node. Here K is the number of classes, and p_k is defined as the fraction of inputs in N with label k, i.e., $p_k = N_k/N$, where N_k is the number of observations with label k, and N is the total number of observations in the node. In the binary case, p_k is defined as the fraction of inputs in N with label k, where N_k is the number of observations with label k, and N is the total number of observations in the node. Mathematically, $p_k = \frac{N_k}{N}$. The Gini impurity in a binary classification scenario, denoted as $Gini(p)$, can be expressed as:

$$Gini(p) = 2p(1 - p)$$

  In this equation, p denotes the probability of the positive class, while $(1-p)$ signifies the probability of the negative class.



  Question: Determine the distribution of the Gini impurity for a dataset containing three distinct labels of equal size.

- **Entropy:** Entropy serves as a metric for the impurity or disorder within a dataset. It quantifies the uncertainty associated with the distribution of class labels at a particular node in a decision tree. In decision tree contexts, the most problematic scenario to avoid is when all classes are equally probable, i.e., $p_1 = p_2 = \ldots = p_k = 1/K$, where p_i denotes the probability of the i^{th} class and K represents the total number of classes.

  In a leaf node, this problematic scenario should be avoided, therefore a leaf node with maximum divergence from this scenario is taken. For measuring the differneces in the class distributions KL divergence is used.

  Mathematically, KL divergence between two probability distributions P and Q is defined as:

$$D_{\mathrm{KL}}(P \| Q) = \sum_k P(k) \log\left(\frac{P(k)}{Q(k)}\right)$$

  where $P(k)$ and $Q(k)$ are the probabilities of class k under distributions P and Q respectively.

  Question: Derive the loss function of a leaf node, i.e., $\min_p -\sum_k p_k \log(p_k)$ *Hints:* Use Q as $\frac{1}{K}$.

  Entropy Over Tree:

  The entropy over a tree can be represented by the following equation:

$$H(S) = p_L H(S_L) + p_R H(S_R)$$

  where $H(S)$ represents the entropy of the entire dataset S, and p_L and p_R represent the proportions of the dataset S that are partitioned into subsets S_L and S_R respectively. $H(S_L)$ and $H(S_R)$ represent the entropy of subsets S_L and S_R respectively.

  The proportions p_L and p_R are defined as:

$$p_L = \frac{|S_L|}{|S|}, \quad p_R = \frac{|S_R|}{|S|}$$

  where $|S_L|$ and $|S_R|$ are the cardinalities (sizes) of subsets S_L and S_R respectively, and $|S|$ is the cardinality of the entire dataset S.

The goal is to minimise these impurity measures by selecting the best feature and threshold combination.

The recursive splitting continues until certain stopping criteria are met, such as reaching a maximum depth or having nodes with a minimum number of samples.