TDT4173 Individual Assignment

H. Friis

September 2022

1 Decision Tree

1.1 Theory and Inductive Bias

Decision trees are used for classification, where one gives the model a sample of features for the model to map to a specific class of the target variable. The sample is mapped according to rules the model has learned during training with labeled samples. It is thus considered supervised learning. The tree is built by splitting the data on a feature that gives maximum information about the remaining data. The remaining data are then further split upon a new feature giving the maximum amount of information for that specific subset, forming nodes and vertices greedily in the process, ultimately leaving a tree when all training data is classified or a stop criterion like maximum depth is reached. Decision trees are well suited for predicting instances represented by attribute-value pairs and problems where the target function has discrete output values.

The inductive bias of decision trees (or more specific the ID3) is that shorter trees are preferred over larger trees. They also favour trees that place high information gain attributes close to the root.

1.2 Results and the Second Dataset

The model achieved an accuracy of 100% on dataset 1 without any data-preprocessing or hyper-parameter tuning. For dataset 2, an accuracy of $\approx 90\%$ was achieved by setting the maximum depth to 3 and the maximum amount of branches from a node to 3.

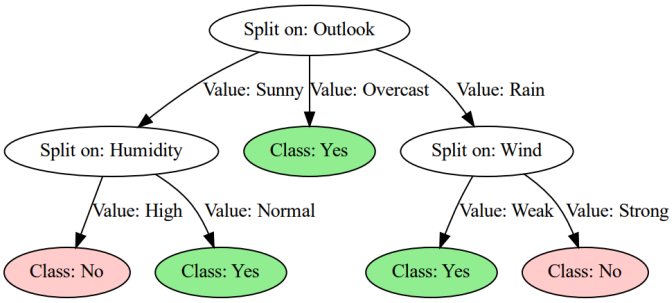


Figure 1: The graph representing the decision tree made for dataset 1. An equivalent was made for dataset 2, but it was too big too be nicely displayed. The tree was made using graphviz.

The second dataset is harder because it introduces irrelevant features like "Founder Zodiac" or "Founder Favourite Color". When fitting a tree without any constraints on this data, it becomes big and complex. Large trees means that it differs from ID3s inductive bias and will generalize poorly, which was the reason for restricting the depth and number of branches. Similarly, feature engineering could have been done to mitigate the problem with overfitting.

2 K-means clustering

2.1 Theory and Inductive Bias

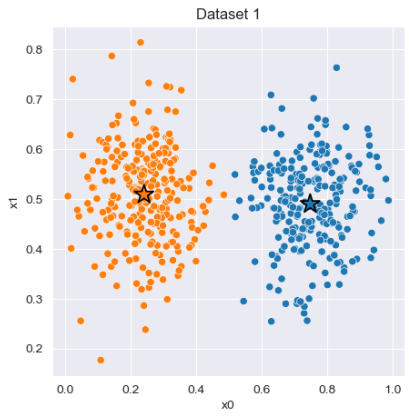
The K-means algorithm tries to partition N observations (of potentially high dimensionality, but 2D is used in the datasets, so the observations will be referred to as points.) into k clusters. This is done by: 1: Initializing the cluster (centroid) positions. This can be done randomly or by trying to fill the space. 2: Calculating the distance (for instance the Euclidian distance) from every point to every cluster center (centroid) and assigning the points to the closest centroid. 3: Each centroid is moved such that its new position corresponds to the mean of the points assigned to it.

Steps (2) and (3) are repeated until convergence or a stop criterion is satisfied. The K-means algorithm is suitable for problems where the aim is to segment data into groups. The use of Euclidian distances limits the data variables to numerical values and cannot be used on categorical labels without data-preprocessing or substantial modifications to the algorithm. The inductive bias of K-means is that points that have similar properties (i.e. x1 and x2) are likely to belong to the same group. It also assumes that features have the same scale.

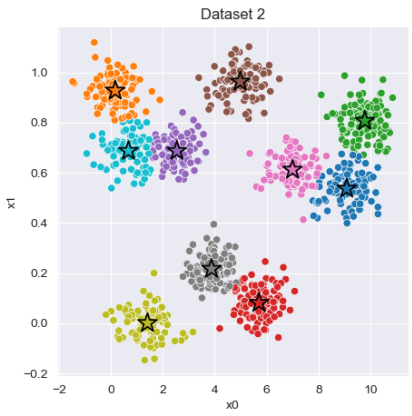
2.2 Results and the Second Dataset

Correct clusters were assigned to the first dataset 100% of the times. For the second dataset, the performance varied from 0% to $\approx$ 100%, depending on the data-preprocessing and modifications to the algorithm. The second dataset is harder to cluster due to at least two factors related to the inductive bias: the scale of the features are different by an order of 10 (in this case favoring long vertical groups) and the clusters have small spread which makes the algorithm highly dependant upon its initial conditions, often leading to local minima and high distortion compared to the optimal solution. (i.e. points with similar features do not belong to the same cluster, which is in contrast to the inductive bias.)

To mitigate this problem, the features were scaled by dividing all elements in a column by the columns maximum value and changes were made to the algorithm. Small improvements were made by creating a k-means++ like initialization of centroids and further improvements were made by implementing some random movement exponentially scaled by the time-step. This alone achieved around 20% performance, so it was necessary to let the algorithm run a user-defined amount of times and pick the best solution in order to consistently score above 50%. By picking the best solution of 20, it will classify the clusters correctly 100% of the times.



(a) The clusters found with $k = 2$ in dataset 1.



(b) The clusters found with $k = 10$ in dataset 2.