K-Means

Introduction

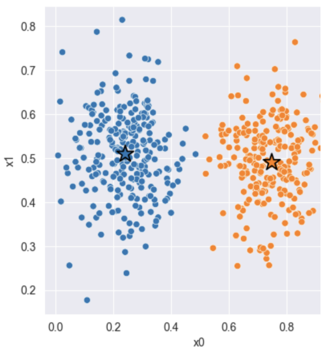
The K-means algorithm is an unsupervised learning algorithm for clustering problems, and widely used in machine learning and data analysis. It is a versatile and efficient algorithm that allows us to group data points into clusters based on their similarity. This is particularly suited for when one has unlabelled data and want to discover underlying patterns or structure within the data. Some examples of such problems can be image compression, market segmentation, and computer vision.

The algorithm starts by initializing a predefined number of clusters. These clusters represent the centers of groups into which the data will be divided. Each data point is assigned to the nearest cluster center based on a distance metric. Then, after the initialization, the algorithm recalculates the centroids of each cluster by taking the mean of all the data points assigned to that cluster. Data points are reassigned to the nearest cluster center based on the updated centroids. These two steps are then repeated iteratively until a stopping cirteria is met. The final output of the K-means algorithm is the set of cluster centers and the assignment of each data point to a specific cluster.

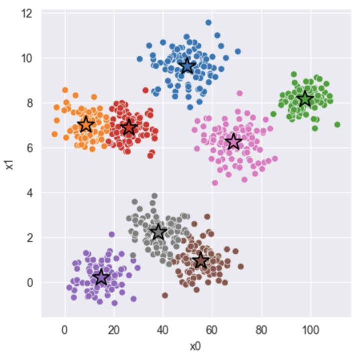
Inductive bias

The algorithm assumes that the clusters in the data are spherical, meaning they have a roughly equal radius in all directions from their centroids. Additionally, it assumes that all clusters have roughly equal sizes and similar densities. This means that the spread of data points within each cluster is roughly uniform. Lastly, there is an underlying assumption that the data can be effectively separated into clusters using linear (or planar) boundaries.

Plots



First dataset



Second dataset

Results and preprocessing

In comparison to the first dataset, the second one had more clusters. Additionally, the clusters were scattered in a seemingly random fashion and did not always have as clear separation between each other. The two axes also had different orders of magnitude, meaning that, although they looked roughly spherical is the plot, the clusters weren't really spherical. These complications also lead to the realization that choosing the initial centroids to be random data-points would not always result in that they would be assigned to different clusters, meaning that this had to be handled differently.

To tackle these issues, I firstly preprocessed the data by normalizing it. This solved the problem of the different magnitudes of the axes. After the normalization, I made two alterations to the original algorithm. The first was implementing a process inspired by the *K-means++*-algorithm. Rather than choosing the initial centroids to be random points, the algorithm instead first chooses the first centroid to be a random point. Afterwards it iteratively chooses the rest to always be the data-point that lies the furthest away from the previously selected centroids. This guarantees that the centroids are well spread throughout the data. The other alteration was implementing the possibility of choosing to fit the data multiple times, and then selecting the fit with the best silouette score. Using the fit with the best score lead to more stable and accurate clusters, but also drastically increased the runtime.

Decision Tree

Introduction

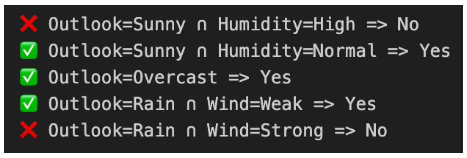
The Decision Tree algorithm is a fundamental and widely used supervised machine learning technique that is primarily employed for both classification and regression tasks. It is a simple yet powerful method that provides interpretable and intuitive solutions to complex decision-making problems. Decision trees are commonly used when faced with problems like feature selection, multi-class classification and risk assessment.

At its core, a Decision Tree is a hierarchical tree-like structure composed of nodes, where each node represents a decision for a particular feature of the dataset. In this case, the Decision Tree is based on the *ID3*-algorithm and works for discrete features. The tree is grown recursively in a top-down greedy approach, where at each recursion one creates a node for the feature that at the present moment yield the best reduction in entropy (highest information gain). The algorithm breaks when all the training data has been fitted to the tree. When doing a prediction, the algorithm searches for a branch in the tree that perfectly corresponds to the features in that data one wants to predict. In the case where no such leaf node is found, there are different ways of handling this problem in order to still perform a prediction.

Inductive bias

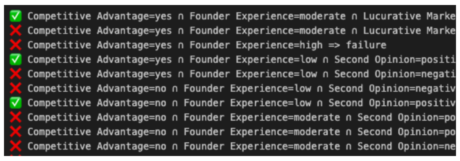
Decision Trees assumes that the data can be effectively partitioned into subsets based on individual feature values. This assumption suggests that the underlying data distribution can be modeled as a series of hierarchical partitions, where each partition corresponds to a node in the tree. Employing a greedy approach for splitting, there also exists an underlying notion that opting for the most informative feature at the current node will result in an optimal decision tree. In relation to this, it is also assumed that some features exert a greater influence on the target variable than others, and should thus be prioritized in the initial stages of the tree construction process.

Plots



```
❌ Outlook=Sunny ∩ Humidity=High => No
✅ Outlook=Sunny ∩ Humidity=Normal => Yes
✅ Outlook=Overcast => Yes
✅ Outlook=Rain ∩ Wind=Weak => Yes
❌ Outlook=Rain ∩ Wind=Strong => No
```

Figure 1: First dataset



```
✅ Competitive Advantage=yes ∩ Founder Experience=moderate ∩ Lucurative Marke
❌ Competitive Advantage=yes ∩ Founder Experience=moderate ∩ Lucurative Marke
❌ Competitive Advantage=yes ∩ Founder Experience=high => failure
✅ Competitive Advantage=yes ∩ Founder Experience=low ∩ Second Opinion=positi
❌ Competitive Advantage=yes ∩ Founder Experience=low ∩ Second Opinion=negati
❌ Competitive Advantage=no ∩ Founder Experience=low ∩ Second Opinion=negativ
✅ Competitive Advantage=no ∩ Founder Experience=low ∩ Second Opinion=positiv
❌ Competitive Advantage=no ∩ Founder Experience=moderate ∩ Second Opinion=po
❌ Competitive Advantage=no ∩ Founder Experience=moderate ∩ Second Opinion=po
❌ Competitive Advantage=no ∩ Founder Experience=moderate ∩ Second Opinion=ne
```

Figure 2: Second dataset

Results and preprocessing

In comparison to the first dataset, the second one included a lot more features. This yielded a more dense and complex set of rules, which did not have as clear of a hierarchical structure. In order to tackle this issue, the data was preprcessed by removing features which were considered to be irrelevant (ie. the *Birth Month*-feature). The algorithm was also modified in such a way that when faced with predicting data without a specific rule, it would search for a rule whose features most closely resembles the data in question. Then it would make a prediction based on this rule instead. This solution worked fine, but has some limitations. One possible improvement is also taking the feature hierarchy (their amount of information gain) into consideration, meaning that it is preferred to choose a rule that have similar features higher up in the hierarchy. Another way of improving the algorithm is by performing pruning after the fitting by using the validation-data. This was implemented, but proved to not yield better results because of the way I chose to handle prediction of data without specific rules.