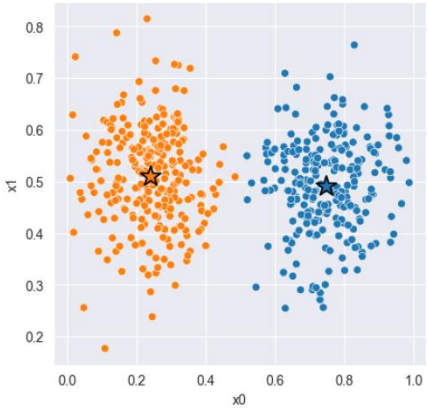
K-means clustering

K-means clustering is an unsupervised learning algorightm. It is optimal for classifying data which can be fit in distinct groups. K-means assumes that the clusters are spherical and that they do not overlap.

Part 1

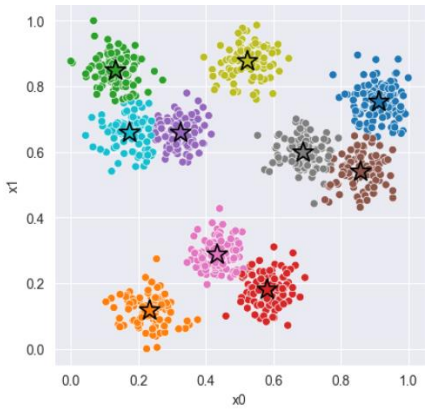
K-means starts by selecting N random centroids. In this case I manually set 2 as the starting point since you can see in the graph that there is most likely two clusters to classify. With the N centroids, you assign each point in the graph to its nearest centroid. Then reposition the centroid to the center of all the points belonging to each cluster. Then you repeat until the centroid positions have not changed for two iterations in a row.



Part 2

For part 2 I normalized the between 0 and 1 using min-max normalization. This allowed the distance calculations to scale properly. To further improve the results, I used k-means++ to set the initial centroid positions. This works by first randomly selecting a centroid, then for the following centroids choose the point which is further away from all the previous centroids.

Implementing this decreased the chance of the algorithm getting stuck in a local minima and I was able to identify 10/10 clusters most of the time.



ID3 decision tree

ID3 is a supervised learning algorithm. It is used for generating a decision tree from a given labeled dataset. It has an inductive bias for shorter trees and for trees with higher information gain closer to the root.

Part 1

ID3 starts by calculating the information gain for every attribute in the dataset. This is the difference in entropy for every value of an attribute calculated before and after a split.

The attribute with the highest information gain is selected as the root node of the tree, and each value of the attribute is added as children. For each value the dataset is split, and the attribute with the highest information gain for each split is added to the tree. This continues recursively until there are no more rows in the dataset to split on.

```
print(f'Accuracy: {dt.a
```

```
Accuracy: 100.0%
```

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

Part 2

The algorithm created for part 1 worked badly for the 2nd dataset. The poor results can be explained by overfitting of the tree. To reduce this overfitting, I tried 2 different things. Using information gain ratio instead of just information gain to decide the splits helped slightly but did not give any significant improvements, which I thought it should. Then I tried pruning the dataset using feature engineering. The “Zodiac sign” attribute does not make any sense to give an impact of the success or failure of a business, but when I removed it from the dataset my accuracy tanked. Then I tried removing each feature, and every pair of features – all of which decreased my accuracy drastically. So unfortunately it seems I have a bug which I have been unable to fix for the part 2. Other features which could give improvements is limiting the max depth of the tree, or setting a minimum number of attributes for each split. However due to me registering late for the course – and therefore not dedicating enough time to the task I was unable to test this in time.

```
Train: 100.0%
Valid: 56.0%
Test: 51.0%
```