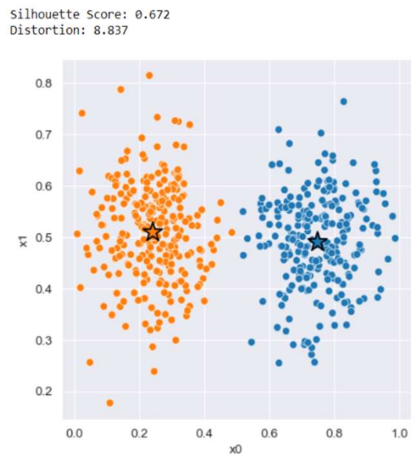
# K-Means

K-means is one of the fastest and most popular clustering algorithms in machine learning. The algorithm manages to partition a multidimensional dataset into k clusters, returning the mean value of each cluster. This value is called a centroid and is often initialized randomly. For each iteration, every point is assigned to the centroid with the closest euclidean distance to construct k initial clusters. Then the cluster centroid is moved to the mean value of the corresponding cluster. The algorithm continues to iterate and adjust the cluster centroids until convergence.

K-means is an unsupervised learning algorithm, meaning the data-points do not contain labels with the correct class under training.  A standard implementation for k-means was sufficient to identify the clusters in the first dataset (*Figure 1*).



```
Silhouette Score: 0.672
Distortion: 8.837
```

*Figure 1: result of the first dataset*

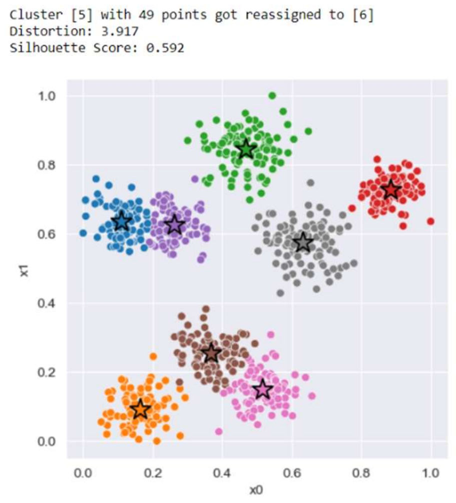
## Inductive bias

To achieve desirable results, k-means include two assumptions about the dataset which had to be considered to solve the second dataset:

1. The variance is the same for all attributes in a data-point representing a class.
2. The data-points are uniformly distributed between the clusters.

## Results/preprocessing

Regarding the second dataset, the x1-attribute had a greater range than the x0-attribute. This was accounted for by normalizing the dataset, which solved the first inductive bias and greatly increased the success rate. I tried implementing k-means++ centroid initialization to take account of the multiple local minima, but this was not sufficient. My solution was therefore to take advantage of the uniform distribution of the classes. After convergence, my algorithm counts the number of classifications per cluster. If the result is not uniformly distributed, the cluster centroid with the least number of classifications gets assigned close to the cluster centroid with highest count. This process repeats until uniform distribution is achieved, finding all clusters over 9/10 times.



```
Cluster [5] with 49 points got reassigned to [6]
Distortion: 3.917
Silhouette Score: 0.592
```

*Figure 2: result of the second dataset*

# Logistic Regression

Logistic regression is a machine learning algorithm suited for binary classification. The training dataset used by the algorithm includes answer attributes which are used to improve the result, meaning it is supervised. When the algorithm is fully trained, it can calculate the probability of a new observation belonging to a particular class.

The goal of the algorithm is to find the best fitting weights, $\boldsymbol{\theta}_i$, for each attribute *i.* These will be implemented in the cost function,

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

where *g(z)* is called the sigmoid function. By iteratively applying gradient descent to the cost function, we can update the weights to maximize prediction probability. The weight parameters can then be defined as

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}.$$

The probability gradient for the first dataset is shown in *Figure 3.*

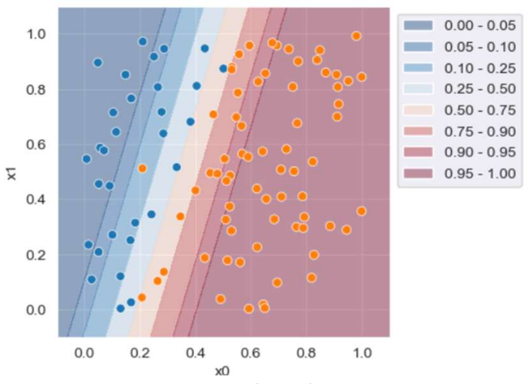


*Figure 3: result of the first dataset*

## Inductive bias

Logistic Regression assumes the observations are independent, which is safe to assume for this assignment. The algorithm also requires a large enough dataset for training, so that patterns can be recognized. This is also fulfilled for both datasets. Thirdly, it assumes that the classes can be separated by a hyperplane.

## Results/preprocessing

```
Train
Accuracy: 0.914
Cross Entropy:  0.249

Test
Accuracy: 0.918
Cross Entropy:  0.281
```

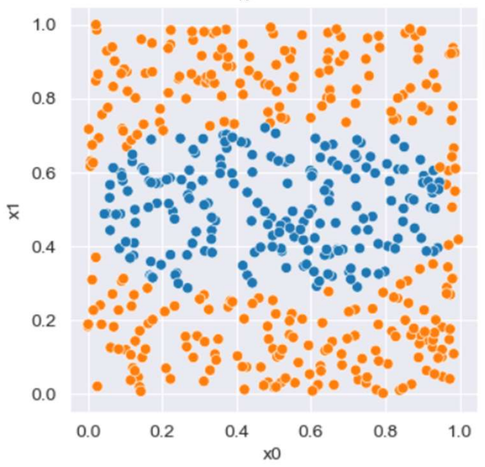


*Figure 4: result and visualization of the second dataset*

As you can see in *Figure 4*, the third assumption is wrong for the second dataset. The classes are not linearly separated, but seemingly separated by an ellipse. To solve this problem, my solution was to transform the dataset. One solution could be to transform it into polar coordinates around the mean (x0,x1) = (0.5, 0.5), but I decided to flip the dataset around the mean value of x1, thereby only transforming the x1 attribute. This resulted in an accuracy of about 90%, which I found sufficient.