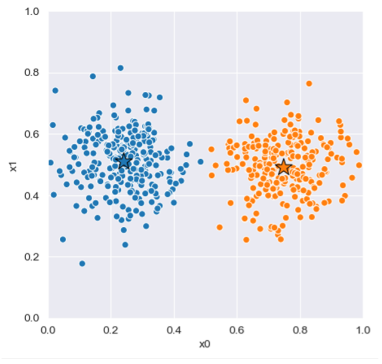
k-means

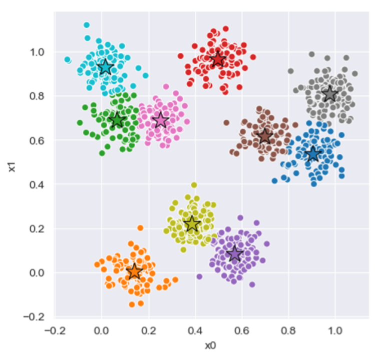
The k-means clustering algorithm is used to divide observed data into k clusters. No labels are given for the data, and hence the clustering is an unsupervised learning problem. In short, the algorithm begins with randomly selecting k initial cluster centroids (centers) and then assigning each training sample to the centroid they are closest to with regard to the euclidean distance. Then new cluster centroids are calculated taking the mean of the samples assigned to each cluster, and the samples are then reassigned to their closest centroid. This is repeated until convergence. The two cluster assignments found in the first dataset using an implementation of this algorithm are shown in figure 1a.

Convergence may be measured using the distortion function, calculating the sum of squared distances between the training samples and the centroid they are assigned to. The distortion is minimized when performing the k-means loop of reassigning samples and recalculating centroids. However, the distortion function is only guaranteed to converge to a local minimum, causing the k-means algorithm to be sensitive to the selection of the initial centroids. Different cluster assignments may therefore be found in different runs of the algorithm when the initial centroids are randomly selected. This was an issue in the second dataset, where the algorithm used on the first dataset generated different cluster assignments in different runs. To avoid this, two measures were taken. First, the algorithm was run multiple times, each time calculating the distortion and storing the cluster assignment if the distortion was lower than that of the cluster assignment previously stored. The second measure was to choose the initial clusters in a more well thought out way than just randomly. This was done using a method similar to k-means++, where at first one centroid was randomly chosen and then the next centroids were chosen one at a time at random but with regard to a probability distribution of weights proportional to the distances to already chosen centroids. Some further explanation of this can be found in comments in the code.

Another difficulty in the second dataset was that the 'x0' and 'x1' features of the data were not in the same range, meaning that the clusters were not approximately circular when seen in a plot with the same range on both axes. Since k-means is based on euclidean distance, this affects the clusters found. In order to find the clusters as seen by eye when plotting the data as done in the Jupyter notebook, the 'x0' values where divided by 10 so that the features were in the same range. The 10 clusters found in the second dataset are shown in figure 1b.



(a) First dataset.



(b) Second dataset.

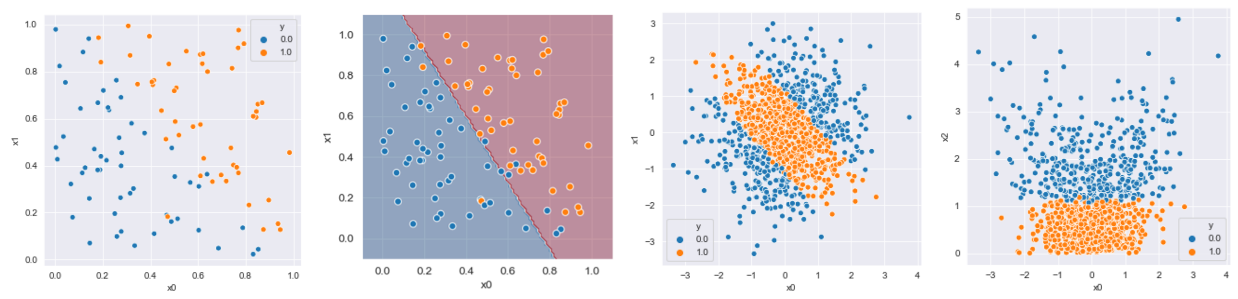
Figure 1: The results of the k-means implementation on the two datasets.

Logistic regression

Logistic regression is an algorithm used for binary classification problems where the values to predict (the labels) may take only one of two discrete values. It is similar to linear regression, but the model takes into account that the labels are discrete valued as well as that there are only two possible values. In order to do so we use the sigmoid function that takes all input data and generates values between 0 and 1. We take a linear function $\theta^T x + b$ of features x as the argument for the sigmoid function. θ contains two parameters, one to be multiplied with each feature of the datapoints. The bias b is needed to allow the line generated by the linear function to not go through (0,0). θ and b are optimized with gradient descent on a loss derived from maximum likelihood estimation with the probability of a certain predicted value 0 or 1, given the datapoints and parameters, is Bernoulli distributed. This optimization means that we generate some θ and b and use them to calculate predictions using the sigmoid function, and next we calculate the gradients of the loss function with respect to θ and b. These gradients multiplied with some learning rate α are then subtracted from the current estimations of θ and b in order to get the next estimations. This subtraction, $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$, is known as the gradient descent rule. Here, h_θ denotes the prediction values (hypotheses) while y refers to the labels. When the parameters and bias have been optimized, the final predictions are calculated. The sigmoid function gives 'soft' predictions between 0 and 1, which are converted into the final predictions by generating a 1 if the soft prediction was ≥ 0.5 and 0 if the soft prediction was < 0.5.

The inductive bias of logistic regression is that it assumes that the data with one label can be easily separated by a hyperplane from the data with the other, which is the case of the first dataset in the assignment (see figure 2a). There, we can draw a line (the hyperplane in two dimensions), separating the datapoints by their different predicted values. This hyperplane is found through the logistic regression algorithm and plotted along with the datapoints in figure 2b. The accuracy of the classification of the first dataset using the implementation was 0.900.

In the second dataset however, the datapoints are not as easily separable, as can be seen in figure 2c. In order to still use the same logistic regression algorithm implementation on the second dataset as on the first, the datapoints were mapped into a space where they are separable with a hyperplane. This was done by passing all datapoints through a function where the 'x0' and 'x1' coordinates where added and the absolute value of the sum was calculated, as written in the code in the 'experiment.ipynb' file. A plot of these mapped datapoints are shown in figure 2d. Running the algorithm on these data points now gives a classification with an accuracy of 0.990 on the 'train' set and 0.984 on the 'test' set.



(a) (b) (c) (d)

Figure 2: The results of the logistic regression implementation on the two datasets.