NTNU

Kunnskap for en bedre verden

TDT4173 - Machine Learning

Individual assignment

Author:
Martin Skatvedt

Date: 17.09.2023

1 K-means

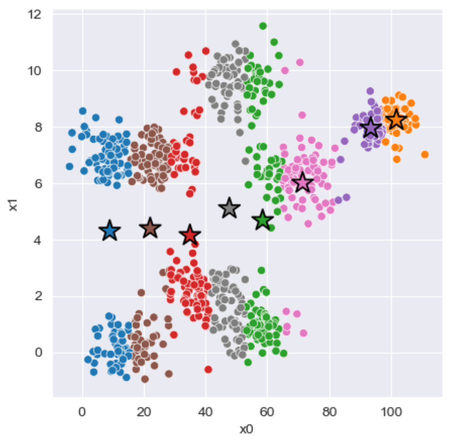
The k-means algorithm is form of unsupervised learning, it tries to classify data points into clusters without labeled data. The algorithm start by selecting k clusters, this can either be done manually, but there are also several ways to find the optimal k. Then it selects k points in the space as its initial centroids at random.

Then the algorithm iterates over all data points and calculates the distance (euclidean distance) to each centroid. Each point gets assigned to the cluster with the shortest distance. For each cluster a new entroid is calculated and gets assigned as the new cluster centroids. The algorithm will then reassign and update its points until it converges. This happens when there is no reassignment of centroids.

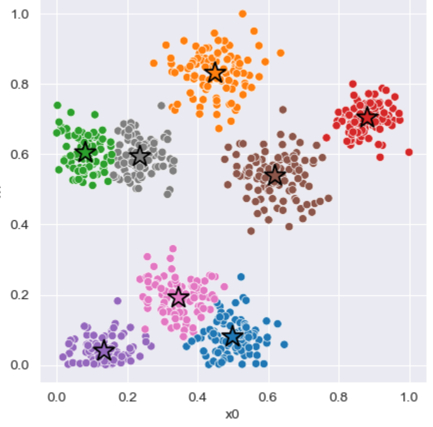
K-means have many uses and is widely used. It is used when you have data which is numerical, or can be converted to numerical values. Also when you have data of a lower dimension, because euclidian distance works well in two or three dimensions, but looses its meaning in higher dimensions. The data should also be continious. It is suited for segmentation, for example segmenting customers based on their purchase history or interests. It is also used in computer vision where we can group similar pixels with each other for object detection.

Its inductive bias is the same as for k-nearest neighbors. Its assumes that the classification of one point will be most similar to other points which are nearby in euclidean distance.

Several factor make the second dataset much harder to cluster than the first. Tthere are more clusters, and some of them are much closer than in the first datset. Secondly the data is not normalized. On the x axis the range is between 0-110 and on the y axis the range is between 0-11. This makes points on the x-axis much farther apart then on the y-axis. If we take k-means inductive bias into account here, it believes that points on the y-axis are in the same cluster because it is as close as it is in the x-axis as can be seen in figure 1a.



(a) Non-normalized data



(b) Final result with normalized data

To solve this problem I normalized the data so that the scale on both axis's was between 0-1. This made the algorithm identify the clusters. I also added the *k-means++* initialization technique to avoid local minimums, and detect the cluster correctly each time.

The algorithm usually converges in under 8 iterations and with a score of:

- Distortion: 4.174
- Silhouette Score: 0.595

2 Decision trees

A decision tree consists of a root node, internal nodes, branches and leaf nodes. The *ID3* algorithm generates a decision tree recursively, it splits on which attribute *gains the most information.* To calculate the best attribute to split on, you can use its information gain, or its Gini-index. When the best attribute has been computed, the training data gets split and branches are created. The algorithm continues recursively. This will create rules which are easy to interpret.

Decision trees are suited for tasks where your data is represented by attribute-value pairs. Also when your output has to be discrete, in this case *yes* or *no*, but can also have more outputs. Decision trees are widely used for classification or regression, as they are visually easy to explain.

Decision trees inductive bias is that shorter trees are preferred over longer trees. It also prefers trees that place higher information gain closer to the root node.

The second dataset introduces attributes which are irrelevant to the the problem, such as the birth month of the founder, as opposed to founder experience. Since birth months can split the dataset into much smaller parts, it has the highest information gain and will set it as the root node as its inductive bias says. Also by placing the birth month as the root node it will produce a much shorter tree.

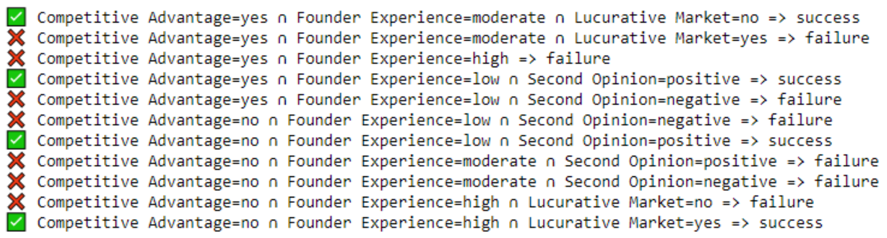
For the second dataset and without any modification from the first dataset my result were:

- Train: 98.0%
- Valid: 64.0%
- Test: 60.0%

Which meant that my tree was overfitting, with a high training accuracy, but low validation and test results. As decision trees are very prone to overfitting, I implemented pre-pruning with a max-depth, minimum samples for splitting and minimum samples for leaf nodes, however this did not produce very good results either. As my best results where with *max-depth=1* which only split on the birth month.

In the end I ended up removing the *Birth Month* column from the data, I also used *max-depth=3* which produced very good results. As can be seen below, with its rules in figure 2.

- Train: 92.0%
- Valid: 92.0%
- Test: 86.0%



```
✅ Competitive Advantage=yes ∩ Founder Experience=moderate ∩ Lucurative Market=no => success
❌ Competitive Advantage=yes ∩ Founder Experience=moderate ∩ Lucurative Market=yes => failure
❌ Competitive Advantage=yes ∩ Founder Experience=high => failure
✅ Competitive Advantage=yes ∩ Founder Experience=low ∩ Second Opinion=positive => success
❌ Competitive Advantage=yes ∩ Founder Experience=low ∩ Second Opinion=negative => failure
❌ Competitive Advantage=no ∩ Founder Experience=low ∩ Second Opinion=negative => failure
✅ Competitive Advantage=no ∩ Founder Experience=low ∩ Second Opinion=positive => success
❌ Competitive Advantage=no ∩ Founder Experience=moderate ∩ Second Opinion=positive => failure
❌ Competitive Advantage=no ∩ Founder Experience=moderate ∩ Second Opinion=negative => failure
❌ Competitive Advantage=no ∩ Founder Experience=high ∩ Lucurative Market=no => failure
✅ Competitive Advantage=no ∩ Founder Experience=high ∩ Lucurative Market=yes => success
```

Figure 2: Rules of decision tree