ITI8730 - Data Mining
Home Assignment 2

LEBARON Maëlie
Student code : 214322IV
maleba@ttu.ee

November 7, 2021

1 Feature Selection : Fisher score

To implement the fisher score I have divided my codes into several functions :

The function `mu` which calculate the mean for a dataset with the feature as a parameter. This function can be used on the whole dataset (to compute μ) or on a subdataset (to compute μ_j).

The function `theta_square` which compute the standard deviation for a dataset.

The function `fisher_score` returns a vector with the fisher score for all features.

2 Classification

2.1 Decision tree

To implement the decision tree, I used the infamous example of *Play Tennis dataset*.

To implement the decision tree, I have split the code into 4 functions :

The function `entropy source`: calculate the entropy of the whole dataset

The function `entropy`: calculate the entropy for a specific feature for a dataset

The function `most useful feature`: compute the information gain for each feature and return the index of the most useful feature

The function `decision tree` which is the main function of the code. It return a `data.tree`. This function is **recursive** and therefore takes *a dataset and a root node as parameters*. The idea is to find the most interesting feature using the `most useful feature` function. We then **split our dataset for each value** the most interesting feature can takes. We apply the function `decision tree` on theses datasets, and attach the tree returned as a child from our root note.

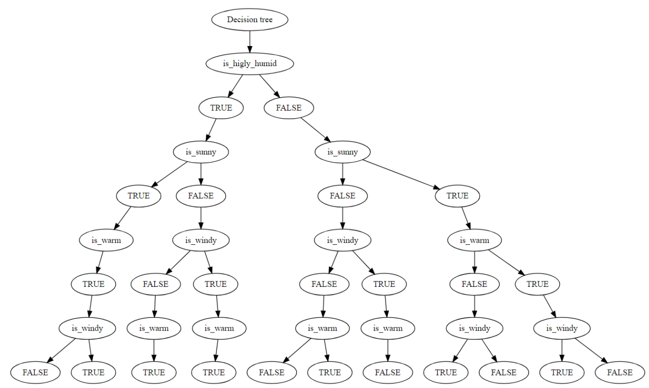

Figure 1: Decision tree for tennis dataset

We can see on the tree that depending on the sub-dataset, the most useful feature is not always the same : when it is *not humid*, depending on if *it is sunny*, the most interesting value after can be *the wind* or the *warmth*.

The fact that *humid→sunny* only have one child is interesting : we could deduce that it is impossible to have *humid→sunny→not warm*. This is a good example of the importance of the data choice, because this case is possible in reality but not represented here.

2.2 K nearest neighbors

To implement the KNN algorithm, I created a function which find the k nearest neighbors using a distance matrix and return the label shared by most of them.

After splitting the dataset in 2 dataset (train dataset is ≈70% of the original dataset and validation dataset is ≈30%). On a simple dataset, with no overlapping, we have 100% accuracy. We the second dataset, we have 98,72% accuracy.

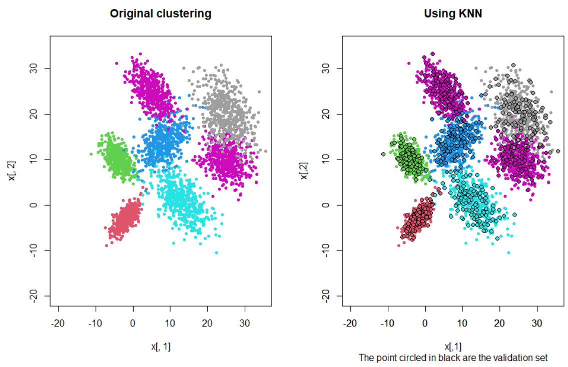

Figure 2: KNN on a overlapping dataset

3 Kernel trick

3.1 Creating the dataset

We need to create a 2D dataset, with cluster in the shape of half-moons. To do that we generate randomly 2 sets of angles in the following intervals : $[0, \pi]$ and $[\pi, 2\pi]$. Using the sets of angles, we create the half moon, adding a "shift" value for one, to avoid creating a circle.

3.2 Applying the kernel trick

The first step is to find a function that will separate our half-moons when going from 2D to 3D, trying different combination and using the plot3D to see the result.

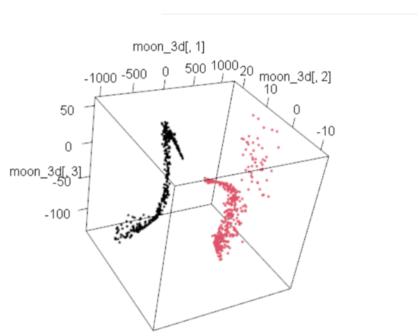

Figure 3: The half moon in 3D

We then have our kernel function, and we can use `ksvm()` function to calculate the support vector. The accuracy of the kernel trick for this particular dataset with the function was 99.125%.

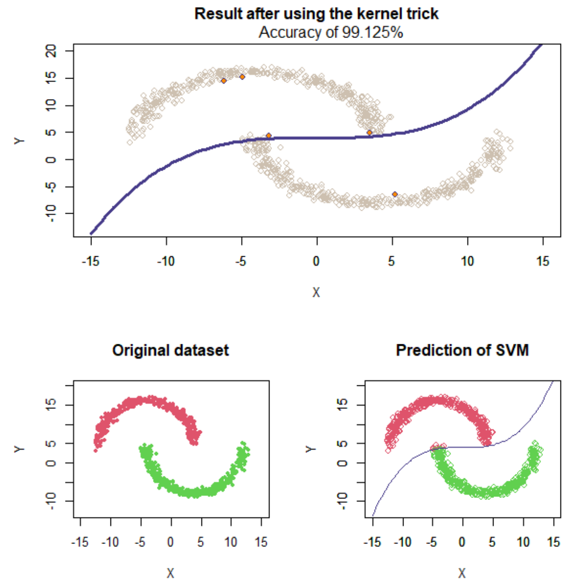

Figure 4: Applying the kernel trick on the half-moons

4 Data preprocessing

To test the dataset, I used the native PCA function. I created 3 features generated randomly (independents) and 2 features that are combinations of the three independents ones. For the PCA to not lead to a reduction of the dataset, the quantity of information of each feature should be approximately the same.

I therefore have a dataset with 5 features, with the following values :

	Standard dev.	Prop. of σ	Cumul. of prop.
1	1.02	0.21	0.21
2	1.01	0.20	0.41
3	1.00	0.20	0.61
4	0.98	0.19	0.81
5	0.98	0.19	1

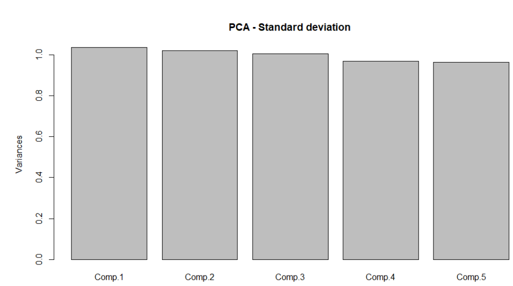

Figure 5: Standard deviation for each feature