Mihhail Daniljuk 02.04.20222

EXERCISE 1

In this task I had to implement a whole model building process algorithm, which includes finding coefficients, checking the model for significance, and checking if adding variables was justified or not. Firstly, to get intuition on what variables to use I computed correlations between a dependent variable and all other features. On each iteration, I choose the feature with the greatest correlation value, compute the model, then perform a significance test, and check if adding a variable was justified, if some of the tests are not passed, I am returning to the previous model and going to the next iteration. The loop converges if model goodness parameters are above a pre-defined threshold, or if the iteration number exceeds the number of features.

Algorithm returns coefficient values, t_stat, and pValue for each coefficient (Table 1). Dependent variable was created by this formula: $y = 5x_1 + 4x_4 - 3x_6 + 0.5x_{11} + \text{noise}$. The returned model gives a very close prediction on these coefficients for x_n. Coefficients that are not included in the final model are zeros. I am not sure in my calculation of t_stat and $pValue$ for each coeff, because it returns NaN for intercept.

Table 1. Result of applying linear regression

	x_n	value	t_stat	p_value
1	0	25.1432	NaN	NaN
2	1	4.9951	22.2969	0
3	4	3.9306	14.3539	0
4	6	-2.9288	-10.2163	2.2561e-22
5	11	0.5410	2.2172	0.0271

EXERCISE 2

In this exercise, I've implemented the Naïve Bayes classifier with feature selection, filtering the data, and cross-validation step. The initial dataset is composed of two gaussians, and I also added some noise features, so that I can filter the data according to the fisher's scores. Filtering is done according to the threshold, which is a hyperparameter in my algorithm. In the cross-validation step, I split train data into *k* folds and apply the same naïve Bayes classifier to compute the total accuracy. There is a possibility to plot a confusion matrix for each iteration of cross-validation. To get an intuition about the goodness of classifier I decided to use a confusion matrix with misclassified and correctly classified labels. To see, which data points were misclassified I also plot the validation dataset with new labels, which determine point's edge color (Fig. 1).

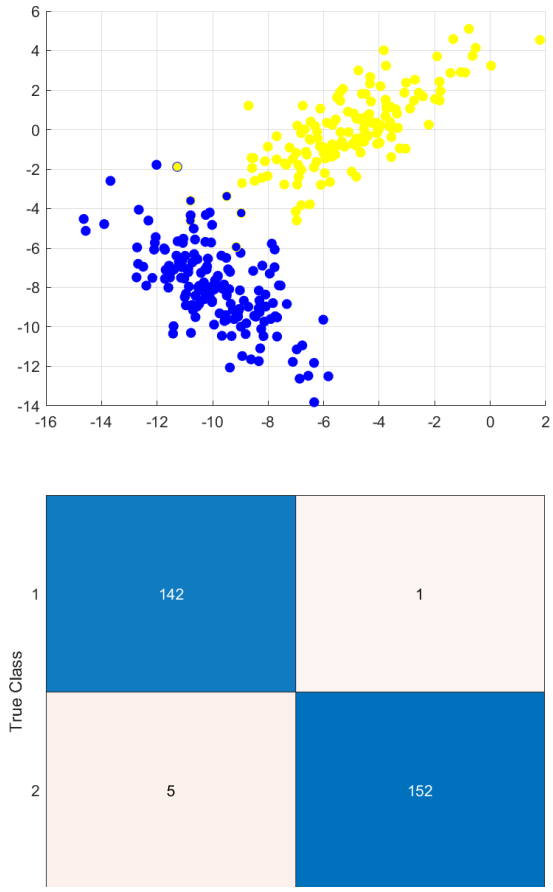

Figure 1. Results of using Naive Bayes classifier

EXERCISE 3

This exercise was one of the hardest ones, as it was the most uncertain for me. I still don't know how to systematically find transformation functions that later could be usable in kernel tricks to easily classify complicated datasets. By doing small research I've found out, that by using a "polynomial" kernel with different dimension parameters, I can find out to which dimensionality I would have to transform data, so it is separable.

I've written a small script that runs a polynomial kernel on my dataset with different dimensionality parameters and found out that it returns 3 as a result. So, it gave me the idea that I have to add 1 dimension to my dataset. After that, I've started searching for this function by intuition and some mathematical knowledge.

2 half-moons in 2 dimensions are separable by the "sinus" or "cubic parabola" function. I've tried both, and as you can see (Fig. 2) the sinus function gave a better result. So, I've decided to use it. Kernel is defined in MATLAB code

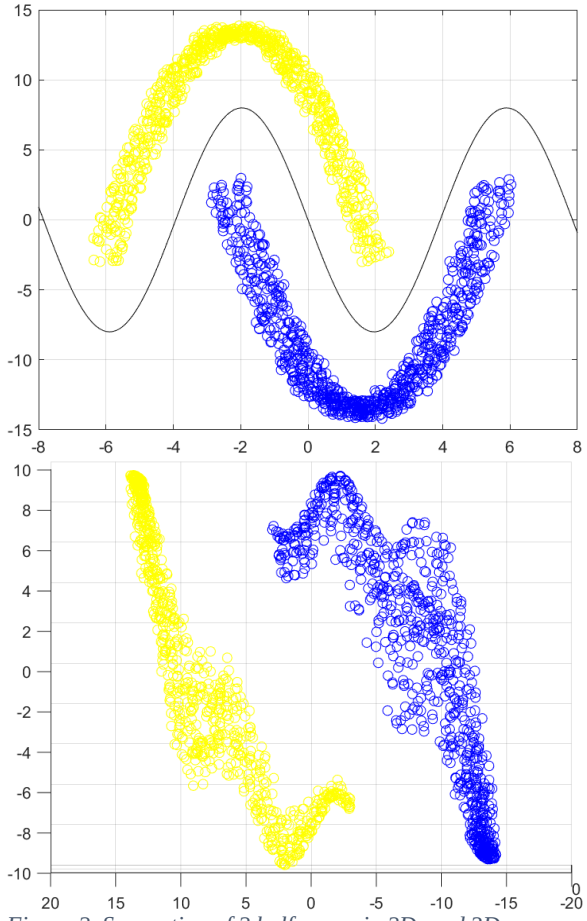

Figure 2. Separation of 2 halfmoons in 2D, and 3D

EXERCISE 4

This exercise was about implementing the Adaboost algorithm, which was very interesting, because, by default, it uses decision trees. In this task I used my previous implementation of the decision tree, the only difference is that I do splitting in 1 dimension. As you can see from the resulting graphs (Fig. 3), the number of trees strongly affects the results of clustering. Trying different variations of this dataset I observed that the optimal number of trees is about 15-17. With this number, the accuracy of the model becomes near 100%. Although even with 10 learners performance of the model is nearly 95%.

As I understood there are two ways for training the Adaboost model, one is to always use the same set with a weighted Gini index, and the second approach is to randomly compose a new set from the previous one, considering the point's probabilistic distribution. I used second approach.

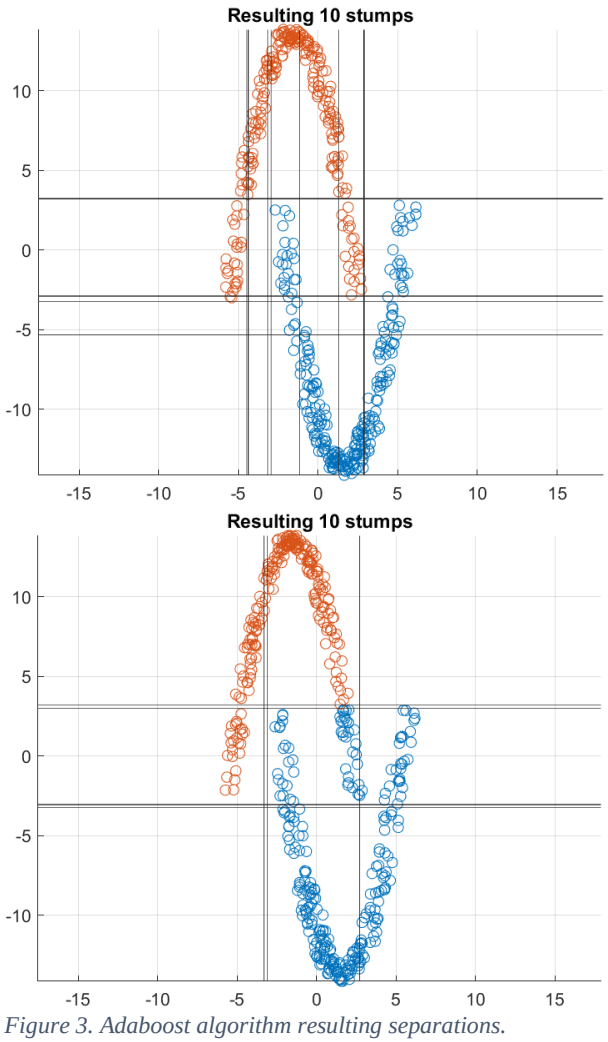

Figure 3. Adaboost algorithm resulting separations.