Report Introduction to Data Science – Assignment 2

The data set analysed in this work contains 13 rotation and translation invariant features of 1000 and 574 images respectively for train data and test data.
The images are taken by a drone on a flying height of 30 meters and their features describe 3x3 meters of wheat fields.
Each image has been classified either as weed (class 0) or crop (class 1).
The aim of the work is to build a *k*-NN classifier to classify the test points finding their nearest *k* points in the train set and according to the majority class of their neighbors.
The data has been analysed in two ways. In the file **assignment2.py** the built-in functions from the library scikit-learn are used whereas in the **assignment2_manual_implem.py** there is a manual implementation of the exercise.

Exercise 1 (Nearest neighbors classification)
The 1-NN classifier is been applied to the data and its classification accuracy has been investigated.
The implementation of the 1-NN classifier *using the built-in functions* from the library scikit-learn result in having accuracy score on the test set equals to 0.945 and accuracy score on the train set equal to 1.00.
The accuracy score states how 'well' the labels of the test set points have been predicted by the model comparing these predictions with the labels found in the test set. From the results, we get 94,5% points correctly predicted; the accuracy score has been computed on the labels of the train set as well to confirm the functionality of the model, the result reported is 100% point predicted correctly.
Besides, in the *manual implementation* of the 1NN-classifier the accuracy score of the on the test set is 0.947 and the accuracy score on the train set is equal to 1.00. We get almost the same results as in the automated version for the test set and the same for the train set.

Exercise 2 (Cross-validation)
With the cross-validation method the data are partitioned into a training set of N-1 size and a validation set of size 1. In this exercise, N is equal to 5.
The *built-in function version* of the cross validation has been implemented using the KFold module. Over the list of *k* from {1, 3, 5, …, 25} the '*perform_cross_validation*' function splice the train set in 5 folds. In each iteration, the classifier is applied using one fold as test data and the 4 remaining as training data. Each time a classification error is computed, the function

return the best k with its classification error. In this case, the best k is 3 with classification error of 0.030. In the *manually-implemented version* the *cross_validation* function behaves the same and the classification error reported for each *k* is the average of the errors computed in each different folding. The best k is **3** with classification error of 0.037.

Exercise 3 (Evaluation of classification performance)
In the *built-in function version* to estimate the performance of the classifier on the Test set it was used the k best value found after cross-validation.
The accuracy score of classifier with k=3 on the test set is 0.949 and on the train set is 1.0

In the *manually-implemented version*, the accuracy score of classifier on the test set, with k=3 is 0.949, same as for the other version but the accuracy score for the train set is slightly less: 0.968, compared to the built-in functions version, this could be due to some errors in the code which were not identified. As a matter of fact, this should have been 1.0.

Exercise 4 (Data normalization)
The normalization process aims to generate zero-mean and unit variance input data.
To do that the version chosen is the first one. Standardize features is done by removing the mean and scaling to unit variance. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set.

The code in the other two versions is flawed because: in the second version, the scaler is computed over the test set which is then transformed with it. In version three the scaler is computed over the total set of data, training and test concatenated, and then the subsets are transformed with it.

Using the test set in the model building process could lead to a biased estimate of the generalization performance of the model. The test set should not be used either for training, data normalization, nor hyperparameter selection.

In the *built-in function version*, the best k found after cross validation of normalized data is 3 with classification error of 0.025, this value results to be lower than the value found from the non-normalized data (0.030). The accuracy score of the classifier with k=3 on the test set normalized is 0.959 and the accuracy score on the train set normalized is 1.0. The accuracy score on the test set results to be higher than the one from the non-normalized data (0.949). Both results show that the classifier is more accurate on the normalized data.

In the *manually implemented version*, the best k found with the cross-validation process is 3 and it has classification error of 0.037. This value results to be higher than the one found with the *built-in functions* and even higher than the error on the non-normalized data. The accuracy score of classifier on the test set normalized data, with k=3 is: 0.949. This value is the same as the one from the non-normalized data.

This last exercise gives some results which are odd in this context. It was expected, as shown in the built-in functions version that the classifier was more accurate on the normalized data, instead almost the same results as non-normalized data have been registered.

It would look like the data have not been normalized or something else is going on in the code. The problem has not been identified.

In the *built-in functions version*, the data points from the Train set have been plot on a 3D scatter plot. This was done exclusively to visualize how the data were classified. The plot has been done applying a Principal Component Analysis to reduce from 13 features to 3. The first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components.