Assignment 2 - Introduction to Data Science

Ninell Oldenburg

February 28, 2022

1 Exercise 1

```
Test accuracy:  0.9775
Train accuracy:  1.0
```

As expected for a 1-NN classifier, the train set accuracy is 1.0 and hence has zero error. The test set accuracy, however, has a value of 0.9775 and therefore seems to not underlie an idiosyncrasy to the train set and yet still have a reasonable good score. From the results we can conclude that we can further proceed with the test and training set as well as with the implementation for the algorithm.

2 Exercise 2

```
Cross-validation of XTrain:
kbest:  3
```

My function takes a list of all k-values to be tested as well as the training data and an optional parameter for the number of splits (5 as default) as input and then iterates over all different k-values in the outer loop as well as, in an inner loop, over the different K-Folds just as provided in the assignment description. In each inner loop, we fit the model to the current test-train-split with the current k-value, calculate the accuracy for the current split and accumulate the `1 - accuracy` (the classification error) per k-value in a list. The respective `fit()` function comes from exercise 1, the `accuracy_score` function comes from `sklearn.metrics` as suggested in the assignment description. After having iterated over all splits, I divide the accumulated classification error by the number of splits to get the mean classification error per k-value. Because the list indices of the mean classification error correspond to the indices of the k-values, I can easily read out the index of the smallest classification error and get the corresponding k-value from my input k-value-list. Note: my tests have shown that the classification error is minimum for several k-values. In that case, `np.argmin()` picks the first value in a list, which I assume is a valid procedure for picking k_{best} as it definitely is one of the best k's.

3 Exercise 3

```
Test accuracy (k=3):  0.9875
Train accuracy (k=3):  0.9933333333333333
Computation Time:  1.57 sec
```

Note: I've recorded the computation time for further elaboration in exercise 4.

4 Exercise 4

```
VERSION 1
kbest:  3
Test accuracy:  0.9875
Train accuracy:  0.9933333333333333
Computation Time:  1.39 sec
```

The goal for normalizing data is to make it more useful for algorithms that don't assume any distribution (like k-NN) and to speed up computation time. For this, we want our training and our test set to go through the same transformation with the same μ and σ^2, which should be calculated based on our training data to avoid any bias towards the test set. The only version that does this step correctly is version 1. Here, we calculate the `StandardScaler()` on and only on our training data and apply it to both our training and test set.

In version 2, however, we calculate a different `StandardScaler()` for each of the training and the test set. In Table 1, we can see that μ and σ^2 are different for the test set and the train set scaler, which is what we're trying to avoid in order to avoid overfitting. The problem of overfitting becomes even more clear when we look at the test set accuracy for this version, which is at `0.99` (taken from the code) and consequently unusually high.

Lastly, in version 3 we're calculating the μ and σ^2 for the concatenated train and test set. This also doesn't lead to the correct values, as visible in Table 1, because we're including the test set values into the calculation of the `StandardScaler()` and thus increasing the bias towards the test set (which, again, we want to avoid). This becomes clear when looking at Table 1, where we can see that μ and σ^2 for version 3 lie in between the correct train set value from version 1 and 2, respectively, and the test set value from version 2 that would lead to overfitting.

When we compare our results to the results from exercise 3 (without normalization), we can observe the same test as well as train accuracy, which leads to the conclusion that the normalization was done correctly and did not change the data in any way. For the computation time, I've been recording it for several iterations of exercise 3 and exercise 4 with the result of having 8 out of 10 cases with a lower computation time for the normalized data, which is expected and reflects one of the goals of normalization. Since our data set is with 600 data points relatively small, the decision of which data set (normalized vs. not normalized) takes more computation time is not perfectly constant. Note that I only measured the time for one of the accuracy measures (here: test accuracy) to not distort the results for any kind of overlap. Also note that the examples shown here are the ones that fit the representation goal of a shorter time for the normalized data sets.

Version	μ		σ^2	
	Train	Test	Train	Test
1	2.10204236e+01		1.06733460e+00	
2	2.10204236e+01	2.14570829e+01	1.06733460e+00	1.36419910e+00
3	2.11950873e+01		1.23184152e+00	

Table 1: Mean, μ, and variance, σ^2, of `StandardScaler()` for train and test set (for simplicity only shown for the first feature).