Assignment 2
Introduction to Data Science

Magnus Lindberg Christensen (fwz302)

February 26, 2024

In the following assignment I will work with the *nearest neighbors* classifier. For this, I will explore data which can be used for occupancy detection.

Exercise 1

For this exercise I will explore the data to get a better intuition of its structure. This will be done by visualising (1) the distribution of the occupancy status, which is our target variable and (2) the correlation between different features and the target variable

Distribution of Target Variable

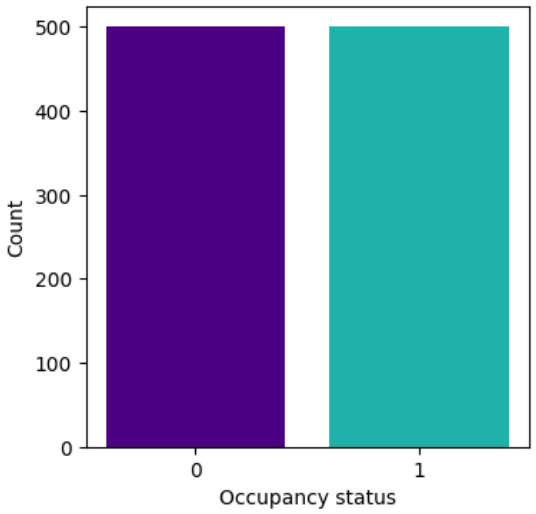

Figure 1: Distribution of Occupancy (0: No occupancy, 1: occupancy)

In Figure 1 the occupancy status in the training sets are depicted using a bar plot, as these types of visualisations are good at showing and comparing data based on its frequencies, and we only wanted to consider the two target values, 0.0 and 1.0. From this visualisation it is clear, that the frequency of *no occupancy* and *occupancy* are both the same, 500, showing that this part of the dataset is well balanced.

Features vs. Target Variable Correlation

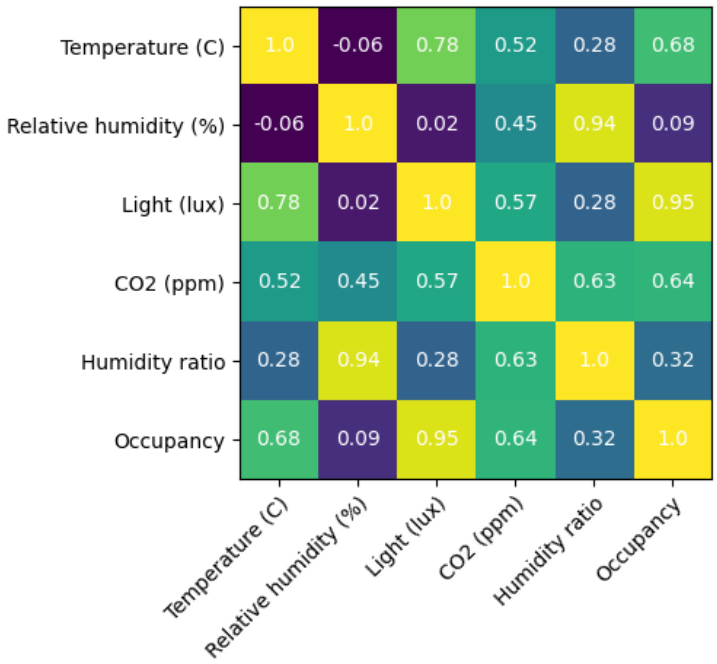

Figure 2: Pearson correlation between all features and the target variable)

To show the correlation between the features in the dataset and the target value I have used a heatmap depicting a standard Pearson correlation between all features, including the target variable. This is used as it provides a visually easy way of showing the correlation between all the features without cluttering. From the heatmap we can see, that the *temperature*, *light*, and *CO2* are highly correlated with the target value (.68; .95; .64). In particular, the light feature are almost perfectly correlated (.95) with the target value. What this means is that if any of these features increase, the probability of a room being occupied increases as well.

Exercise 2

In the following exercise I will present discuss the results from my 1NN classifier. The results are as seen in Figure 3

Figure 3: Accuracy scores of my 1NN algorithm for train and test

From the results it is obvious that the accuracy from the training set are perfect, which is intuitive since we are measuring the distances of the same values. However, it is as well be an indication that our classifier might be overfitting. On the other hand it still shows that the model has a good generalisation to unseen data by correctly classifying 97.75% of the test data. Yet, with this particular assignment we are only working with a 1000 examples, and the model has only been trained on 600 of these which indicates that the dataset might have been too small, and therefore fits almost perfectly. Of course, we are also only considering the examples of $k = 1$ which can potentially increase the accuracy in some cases.

Exercise 3

In the following exercise I will report the best choice of k and describe the process towards this choice. Further, I will justify whether larger values of k are preferred or not.

Figure 4: Best value of k

In Figure 4 we can see the best value of k, which is the choice of k = 3. To find this, I used a 5 fold cross validation. More particularly I created a list of six potential values of k, that I wanted to test. I then created a script which went through each of these values. For each value I created a list to hold the accuracies of each cross validation split. I then split the training set into train and test indices, using the ***from*** *sklearn.model_selection* ***import*** *Kfold* model with 5 folds. After this, I defined the train and test sets from my original training set.

After having constructed the train and test sets, I initialised a knn model using my developed **kNearest** class, and defined the current k to be the current k in the loop. Hereafter, I used my **fit** function to fit the currently indiced training data to my model, and then I predicted the labels for the test set using the **predict** function in the class.

After predicting all the labels, I calculated the accuracy by comparing the predicted labels to the ones defined by the KFold model, using the ***from*** *sklearn.metrics* ***import*** *accuracy_score* metric.

This was repeated for all of the folds, and afterwards the average accuracies was calculated using **np.mean()**.

To find the best value of k, I then used **np.argmax()** on all the average accuracy scores to find the index of the highest average, and hereafter used this to index the list with the potential k's, which led to the optimal value of k being $k = 3$.

Is it always better to consider more neighbours?

The immediate answer to this question is, no not necessarily. From the results we see that the best k value her is $k = 3$, hereby eliminating all k's larger than this. The values of these is either the same or decreases which shows two things; (1) that larger values of this particular example will have a decrease in accuracy, and (2) that even if the higher values of k have the same accuracy, its computationally more efficient to stay with the lowest value of k.

Exercise 4

In the following exercise, I will prevent the general performance based on the full dataset and using the best k of $k = 3$ as found in Exercise 3. Further I will present the accuracies when predicting based on the training set and test set, respectively. The results from these predictions can be seen in Figure 5.

```
Accuracy k_best train   : 0.9933333333333333
Accuracy k_best test    : 0.9875
```

Figure 5: Accuracy comparison between train and test predictions using the best k value $k = 3$

For the results we can see that the predictions based on the training data performs slightly better ($train \approx .993$) compared to the predictions based on the test data ($test \approx .986$). This is of course not surprising, as the training data is, as the name suggests, what the model is fitted to. However, it is worth considering why accuracy of the train predictions isn't $train = 1.0$ as with the 1NN. This has to do with the fact that we are comparing more data-points to our new data, and therefore its not unlikely that some of the values in the training set will experiencing a majority vote favouring some of the test values in a few cases. However, the results are still somewhat near perfect. Also, the comparison between the two shows that the difference in accuracies between the two isn't significant when classifying, indicating that our model confidently predicts the labels of newly occurred data.

Exercise 5

In the following exercise I will center and normalize the data, and compare the results between non-normalized and normalized based predictions. First, I will discuss the three proposed variants of data normalization.

Considering the three versions of data normalization I believe that **version 1** is a suitable way of processing the data, as this is the only version that ensure that the training input data has a mean of 0 and a variance of 1, by centering and normalizing all the data only based on the mean and standard deviation of the training data. In version 2 the data is correctly processed for X_train but not X_test as the result would produce two different scales, and thereby not a measure of comparison, as it scales based on two different means and standard deviation. Lastly, version 3 is not possible to use, as this calculates the mean and standard deviation from all of the data. Hereby, the training data presumably wouldn't have a mean of 0 and variance of 1, and it bases the values on 400 rows of extra data from the test set as well. Hereby as a conclusion, version 1 is the only one that can produce the result searched for.

After centering and normalizing the data, I computed the best k through kfold cross validation.

```
[0.9833333333333334, 0.9916666666666666, 0.9916666666666666, 0.9916666666666666, 0.9899999999999999, 0.9916666666666668]
Best k has been found... k=11
```

Figure 6: Best value of k from kfold cross validation on normalized data

In Figure 6 we see that the optimal k once again is $k = 11$, with an accuracy of $\approx .9917$

Once again I ran the k_best model with $k = 11$, now on the normalized data. The results of this can be seen in Figure 7

```
Accuracy k_best train normalized        :       0.9916666666666667
Accuracy k_best test normalized         :       0.98
```

Figure 7: Accuracies from running k_best algorithm on the normalized data with the optimal $k = 11$

Comparing the results without normalization ($train \approx .9933$; $test = .9875$) and with normalization ($train \approx .9918$; $test = .98$), there is a slight decrease in the accuracy for both examples. All examples seems to capture the underlying pattern. This is not surprising for the training data, since this could be just a result from overfitting. Yet, it still seems that overfitting is not an issue, as the testing data more or less captures the pattern just as well.

Where the main difference with and without normalization are found, is the optimal k. Without normalization the optimal k is $k = 3$, where this becomes $k = 11$ with normalization. Normalization hereby indicates to have an effect on the choice of the hyperparameter of the model, assuming that it requires more examples to correctly classify the data. This as well indicates that

working with normalized data are computationally more expensive than working without normalized data. The higher choice of parameter as well indicates that the normalization effects our distance metric, as the variance becomes 1. Hereby, the higher coherence across the data, requires a larger value of the hyperparameter k to confidently make a majority vote to classify the data.

Assignment Statements

Google Gemini 1.5 was used as a tool to create the overall structure when developing plots, for instance by setting up the figure size and colouring. Also It helped me find the correct numpy functions that I was searching for, eg. np.bincount, np.argsort etc.