Project Mobile Phone Price Classification

Modul:	Machine Learning	
Participants:	Felix Möcks	1395594
	Peer Kröll	1395578
Trial date:	10.07.2024	

Content

1. Introduktion

The "Mobile Phone Price Classification" project aims to find the price category of smartphones, based on its technical specifications. This can be used by smartphone manufacturers, to get an estimate of where the smartphone can be categorized in terms of price. It can also be used by customers who want to see how expensive smartphones are, that fulfill the aspects that are important to them.

2. Decision-making for the used algorithm and its problems

2.1. Chosen Algorithm

For this project, the k-Nearest Neighbors algorithm is used. There are several reasons why kNN is useful for this kind of task. Since we have a clear sample size with specified classes and the exact labels for each of them, the kNN algorithm is effective for sorting the unknown price into the given price ranges using the Euclidean distance. Furthermore, the sample size is quite small, so although kNN is not the fastest algorithm, it takes almost no time to compute the data and the prediction. When using the kNN algorithm, the weighting of the data is not so different, so there is no feature that is more important than the other if we do not talk about its numerical value.

2.2. Progression

To be able to review our code we decided to split the training data with the known price ranges. With the split data we can calculate the accuracy of our algorithm. For this data we created an algorithm to predict the price classification of the split test data. This algorithm created a lot of problems with the data preprocessing, but more on that later. After all the problems were solved the prediction of the test.csv could be implemented and first predictions could be done. Because we have no opportunity to control the predictions of the test.csv because there were no “price_ranges” given, the next step was creating a GUI for a better interaction. We decided to keep the overlay of the GUI very simple. We implemented only a few buttons, a selection box and a widow to show the data and the predicted price ranges. We added a button to detect the accuracy of the predictions of the trainings data. In this case the training data is split in 80% training data and 20% test data. The training data is used to “train” the model and the test data is used to become predicted. After prediction the price ranges are controlled, and an accuracy become calculated. This accuracy is then shown next to the button. We also added a selection box, were the entries “all”, or a specific id of the test.csv could be selected.

Appending on the choice the chosen ids are shown in a window below in the form of a table. First the table shows nothing in the column of the “price_range”. With pressing the button “detect pricing” the price range of the ids shown in the window are predicted. These predicted values are then added in the table and shown in the column on the rightest side.

After some tests and a corrected version of the code we were able to document the code and create the necessary documents.

2.3. Challenges of the project

The biggest challenge we faced was pre-processing the data. As some properties of the raw data have a greater influence on the result than others, so the data must be pre-processed. Due to the Euclidean distance the "battery_power" in the raw data has a significantly greater influence on the prediction than the properties that are either present or not, i.e. are represented in the form of a Boolean.

2.4. Different solution approaches

Even though the kNN algorithm has a way to determine the correct prediction with the Euclidean distance, it happens that the accuracy and therefore the solution differs depending on the data pre-processing. This means that using a different method, such as the standard deviation, or even not changing the raw data set, will significantly change the result. So, to get the most accurate solution, we tried different methods to decide which one to use for the best accuracy in our use case.

2.5. Finding the best pre-processing method

To get the most accurate price prediction, it is necessary to use the right number of neighbors to get the best solution, since a different number of neighbors could change the categorization. To do this, we go through a large number of nearest neighbors (up to 200) and plot them on a graph to see the highest accuracy. Furthermore, to make sure we picked the right one, we stored all the accuracy data in an array and used the "argmax" function to see the best "n" neighbors. Then we used this value directly to get the accuracy of the training data.

The explained method is shown below with the corresponding plots.

For reference: For this case we used a test_size of 0.2 and the random_state = 45.

Attempt 1:

Here, all values were changed to a value between 1 and 0 to secure the influence of the Boolean values. This was achieved by dividing the values by the largest of the specific feature.

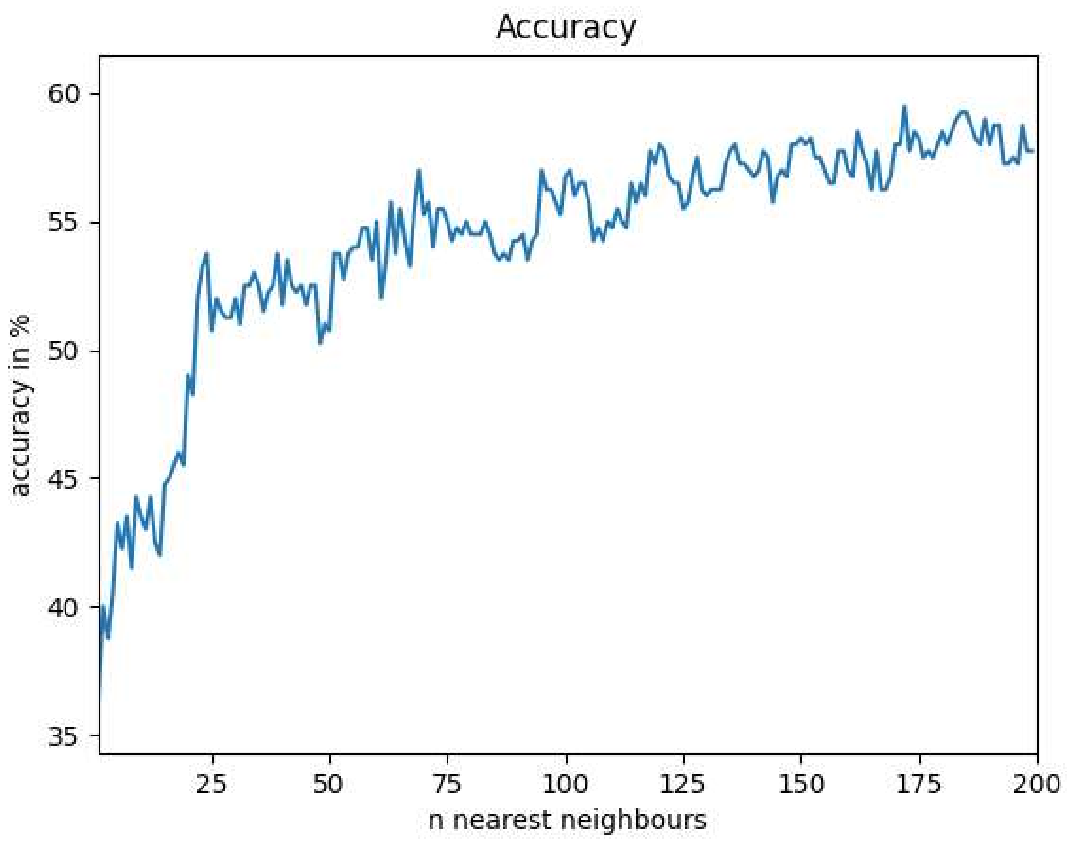
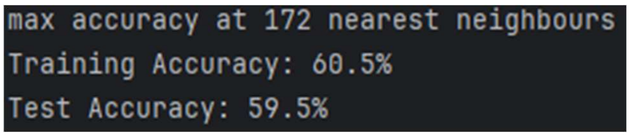


Figure 1:Accuracy histogram of attempt 1

As you can see, the accuracy of the prediction is not good. The accuracy increases with the number of nearest neighbors, but even with 200 neighbors the accuracy is still poor. The highest accuracy is achieved with 172 neighbors, with an accuracy of 59.5% for the test data.



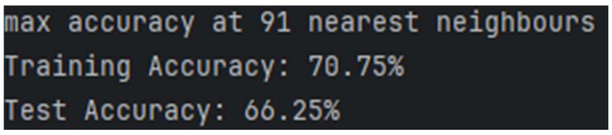
```
max accuracy at 172 nearest neighbours
Training Accuracy: 60.5%
Test Accuracy: 59.5%
```

Figure 2: Highest accuracy in attempt 1

Attempt 2:

Using the standard deviation vertically.

In this attempt, all values were standardized using the standard deviation formula for each class. This was also the case when the described problem occurred with the different Boolean types, which are difficult to standardize.



```
max accuracy at 91 nearest neighbours
Training Accuracy: 70.75%
Test Accuracy: 66.25%
```

Figure 3: Highest accuracy in attempt 2

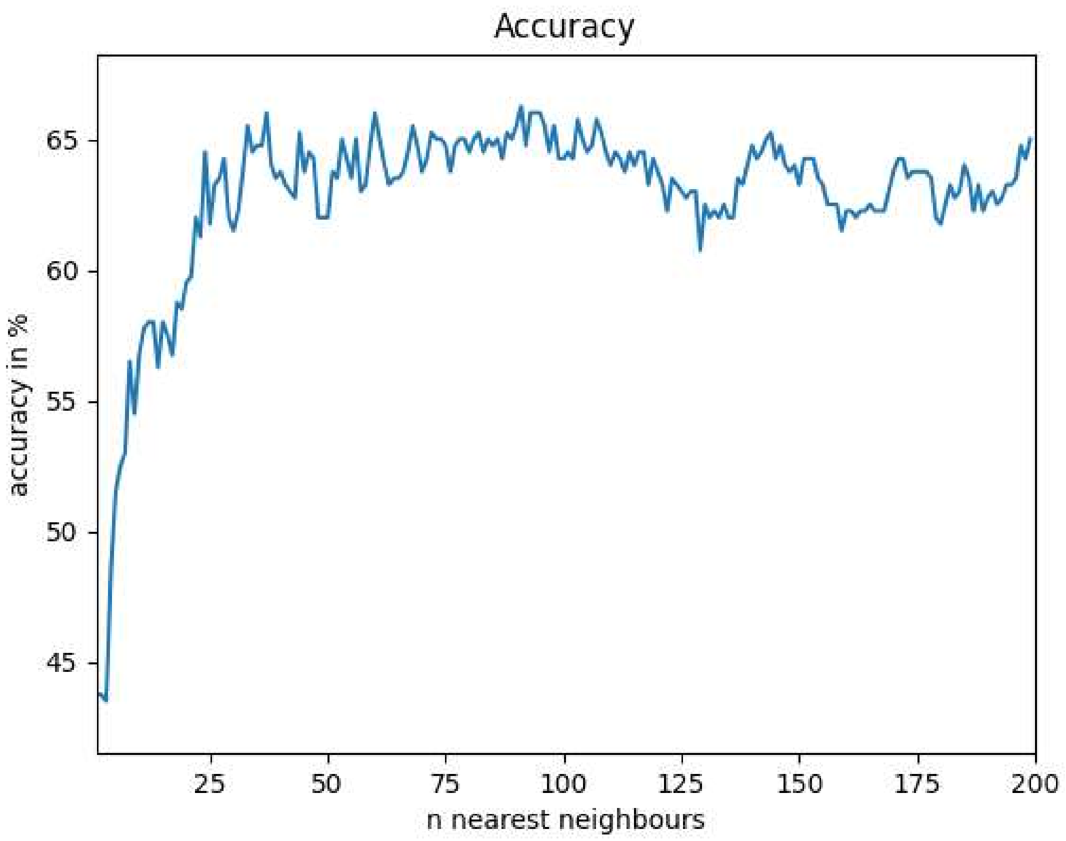


Figure 4: Accuracy histogram of attempt 2

The plotting shows that using the standard deviation has a poor accuracy for any n nearest neighbors. Therefore, this method could not be used well for the training and test data because it only has an accuracy of 66.25% for the test data set and an accuracy of 70.75% for the training data set with 91 nearest neighbors.

Attempt 3

Here the data was not preprocessed. With this method the values have a different weight in detecting the pricing.

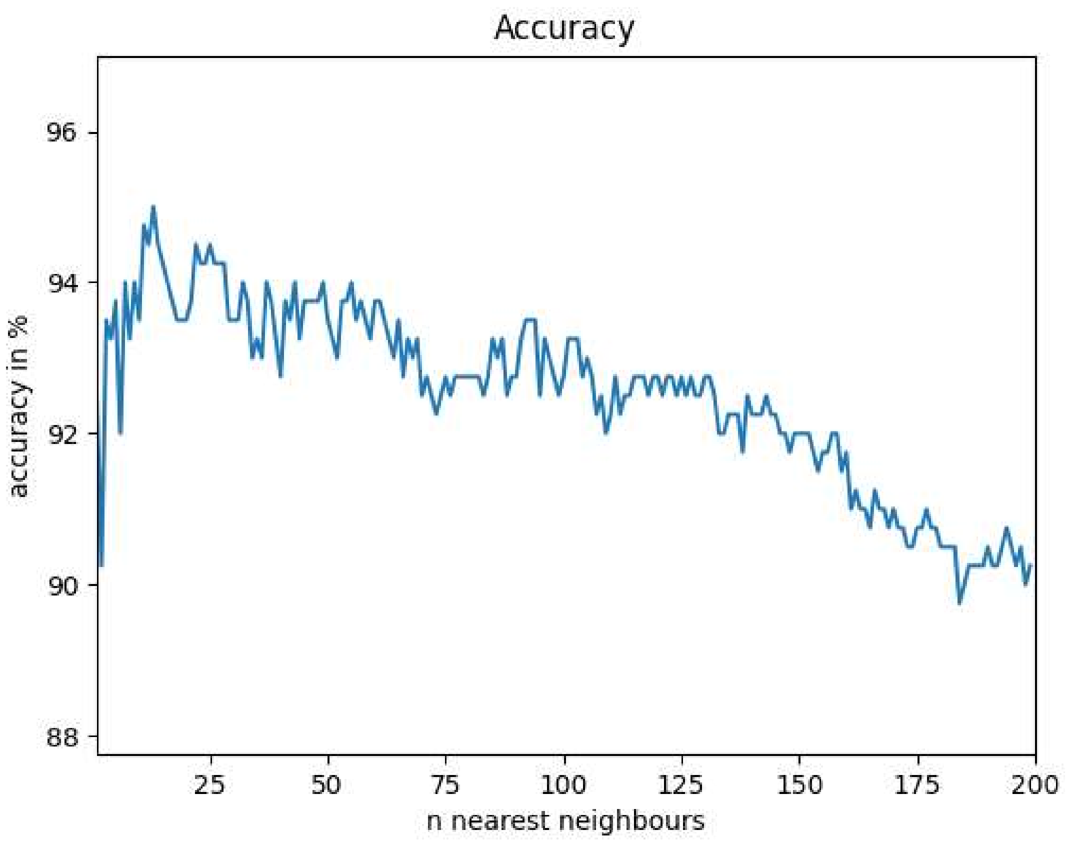
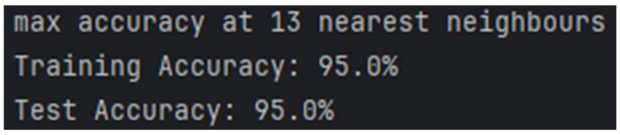


Figure 5: Accuracy histogram of attempt 3



```
max accuracy at 13 nearest neighbours
Training Accuracy: 95.0%
Test Accuracy: 95.0%
```

Figure 6: Highest accuracy in Attempt 3

With this method the accuracy was way better than with the other methods. It achieved an accuracy of 95% with the train data and the test data.

It has the maximum accuracy at n = 13. When looking at the graph above, you can see a small advantage using the raw data. If we compare the raw data with the other preprocessing methods, we can see that the raw data can use a smaller n nearest neighbors, so it needs a little less computing power than the other methods.

2.6. Finding the best n nearest neighbors

The best method for preprocessing has now been identified. In general, the train- test split random state has an impact of the accuracy, but because of the significant difference in the accuracy it was not necessary to test it with another random state. To find the best number of nearest neighbors the data from the train.csv was split 200 times with random "randoms_states". Because the best number of nearest neighbors is somewhere in the lower range as you can see in Figure 5 it was not necessary to test all nearest neighbors up to 200 again. To minimize the workload for the program, the accuracy of the nearest neighbors was detected in a range of 1 to 40. The nearest neighbor with the highest accuracy was then plotted in a graph. If several nearest neighbors achieved the best accuracy, these were also included in the evaluation. In the end, the following graph was created:

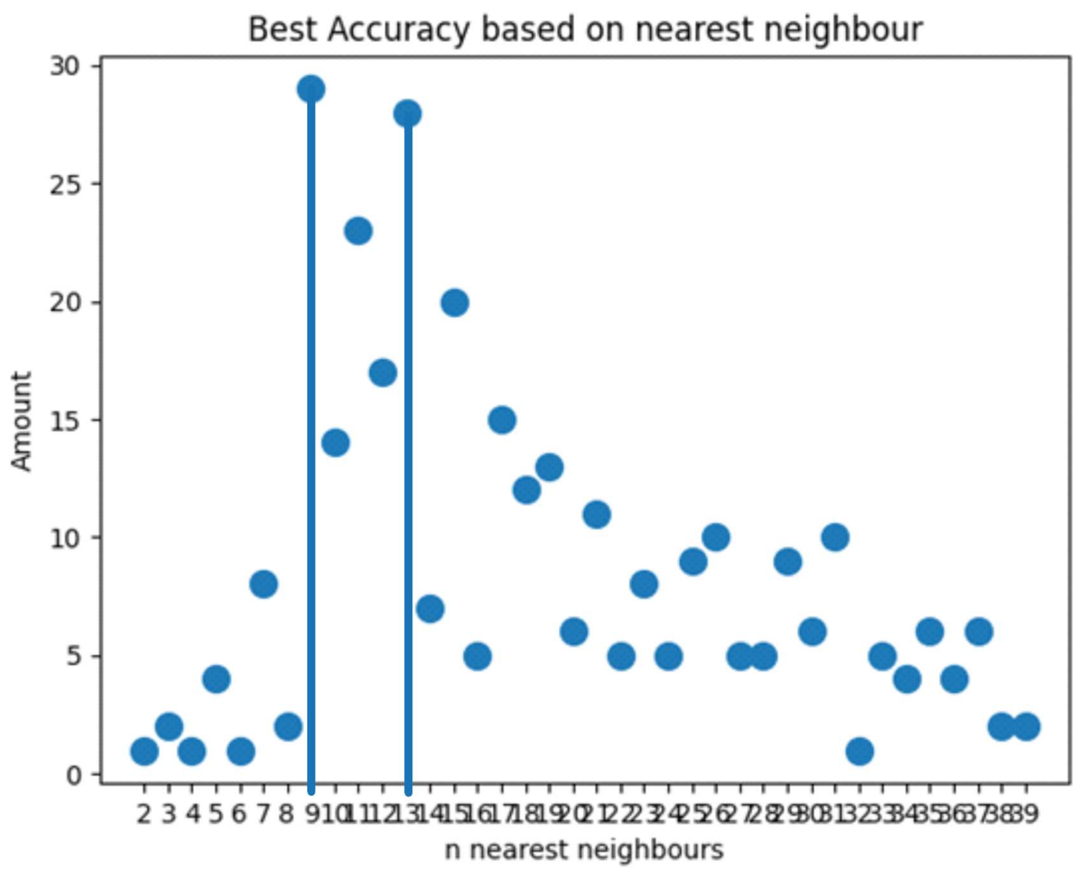


Figure 7: Nearest neighbors with the highest accuracy

As the diagram shows the highest accuracy is reached the most times with 9 nearest neighbors. Also, a good number of nearest neighbors is 13. So, the tests we did above were nearly the best amount we could choose for testing.

2.7. Mobile Phone Price Correlation

The following graph shows the heatmap distribution of the Mobile Phone Price correlating to its features of the training data.

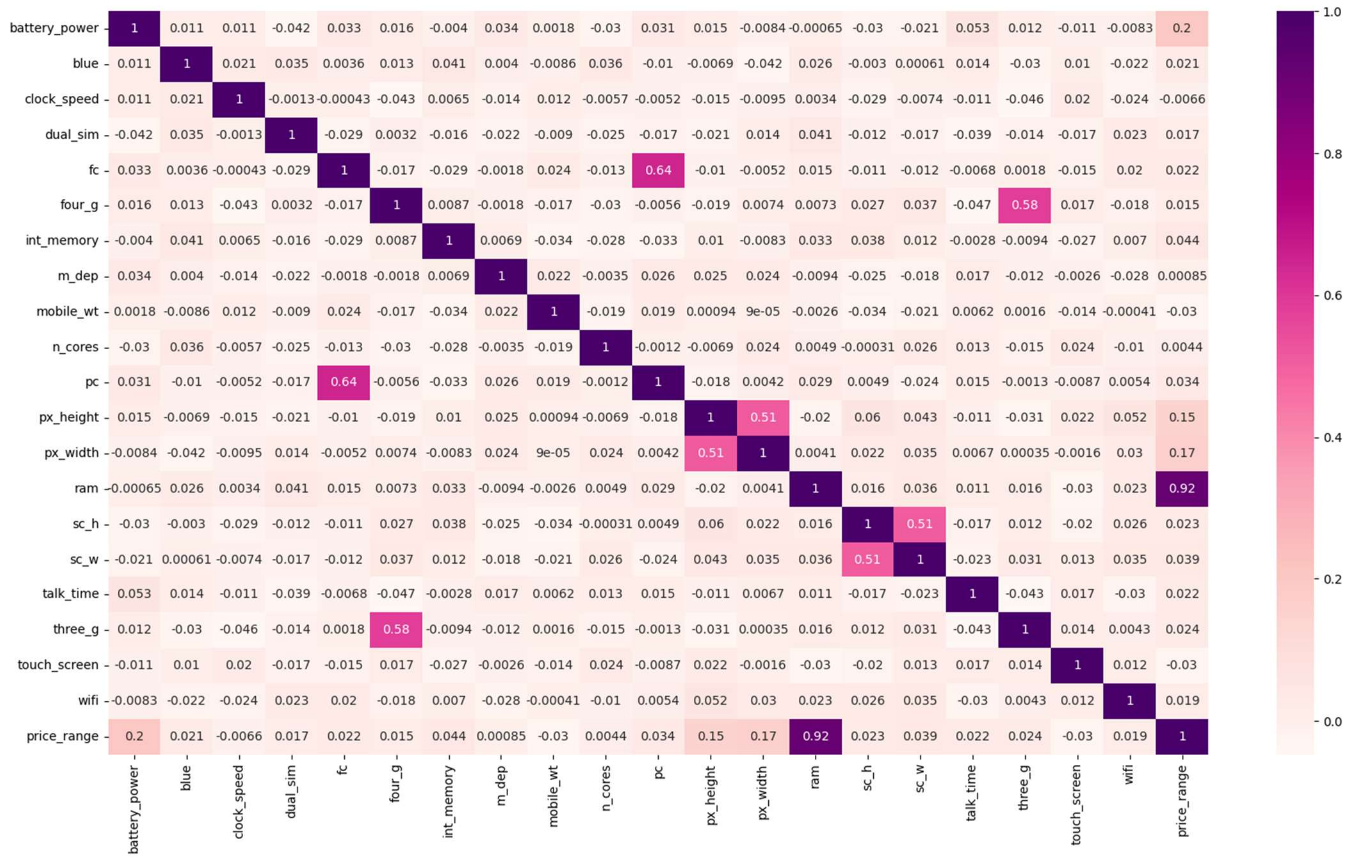


Figure 8: Price correlation

As we expected and described earlier, the features that have a wider range of values are more correlated with the phone price than the Boolean ones. This also means that these values are more fitting and therefore a better indicator for the price classification. This even shows that the Booleans with, for example, the value one (which is the maximum value) don't say anything about a high price value and therefore don't correlate with the pricing.

3. Prediction distribution

After using the raw data and the most accurate "n", it is possible to see the prediction distribution and the price range with the least accurate predictions using a heat map. The confusion matrix shows whether the actual price matches the predicted price. In this case the random_seed 45 is used.

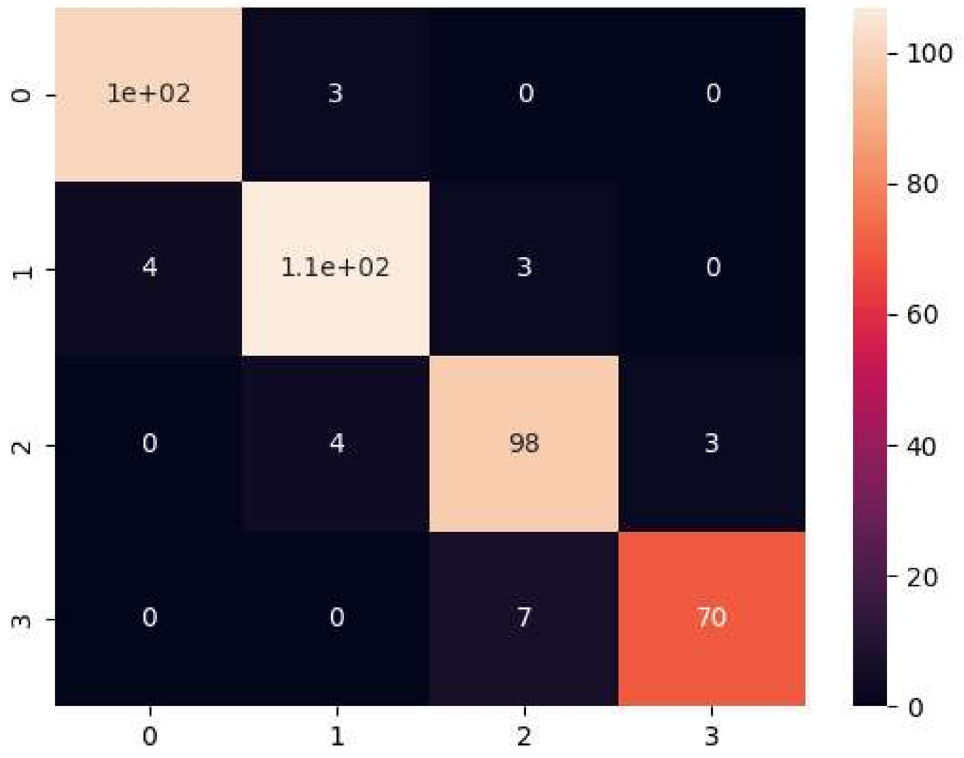


Figure 9: Price Prediction confusion matrix

As you can see, the price predictions are equally good for each price class. It also shows that there isn't a big problem with the data for a specific price range, so there is no need for a specific adjustment.

4. Use Case Diagram

With the following diagram the possibility of the program is shown as a use-case diagram.

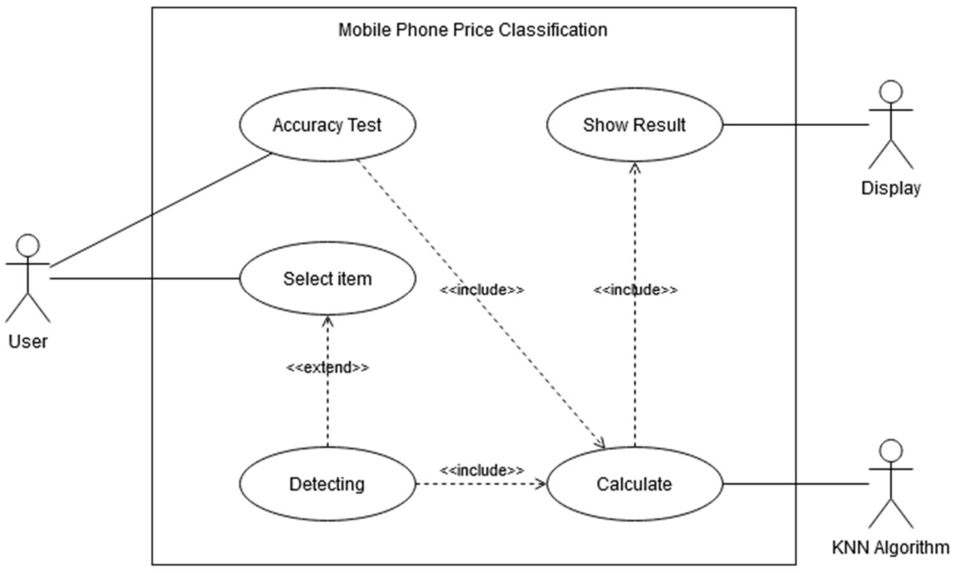


Figure 10: Use-case diagram

The user initially has several options. They can test the accuracy of the KNN algorithm or select an ID from the test.csv file. A single ID or the entire file can be selected here. After selecting the data, the user can use the "detect pricing" button to have the price predicted. This data is then predicted by the KNN algorithm and shown on the display in the form of a table.

5. Activity Diagram

The following diagram shows the program as an activity diagram.

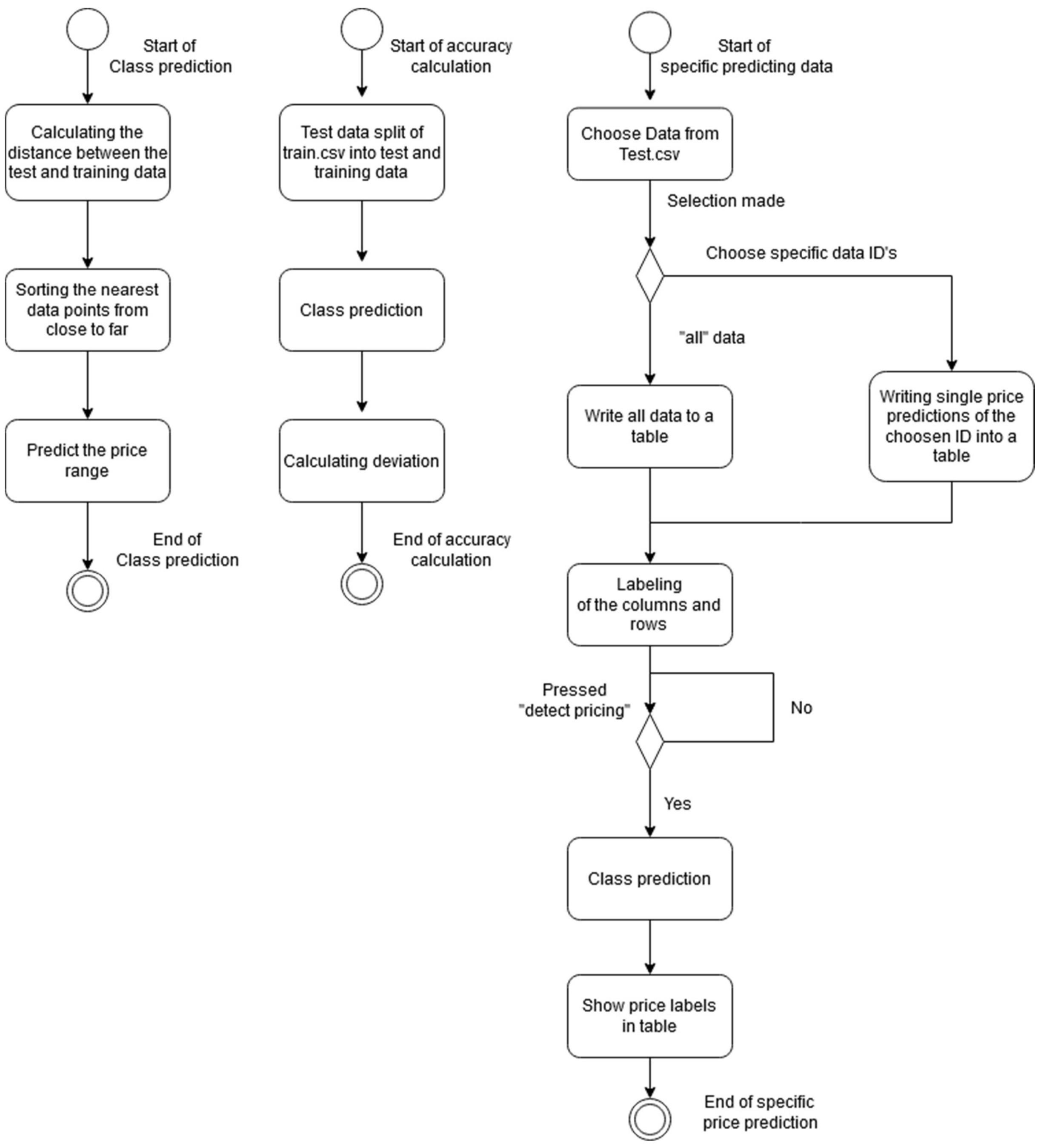

Start of Class prediction
Calculating the distance between the test and training data
Sorting the nearest data points from close to far
Predict the price range
End of Class prediction
Start of accuracy calculation
Test data split of train.csv into test and training data
Class prediction
Calculating deviation
End of accuracy calculation
Start of specific predicting data
Choose Data from Test.csv
Selection made
Choose specific data ID's
"all" data
Write all data to a table
Writing single price predictions of the choosen ID into a table
Labeling of the columns and rows
Pressed "detect pricing"
No
Yes
Class prediction
Show price labels in table
End of specific price prediction

Figure 11: Activity diagram

With the activity diagram the written code is presented in more detail than with a use-case diagram. Therefore, it is more structured regarding the code and shows what happens within each use-case and the result of it.

6. Conclusion

Even though the k-Nearest Neighbor algorithm didn't work as expected using the standard deviation with a maximum of 66.25%, it worked well with the raw data with an accuracy of around 94% using 9 nearest neighbors. For this reason, kNN is a good algorithm for analyzing data that has a clear structure, features and classes, and where the data is informative for predicting its corresponding price category. After fixing a few problems regarding the weight of the classes, we can say that the kNN algorithm is quite simple and useful to apply to a given dataset like the Mobile Phone features.