Project: Devices Price Classification System

Project Description:
Build a Devices Price Classification System (AI System) using Python and SpringBoot. Mainly the system will include two small projects:
● **Python project:** will allow you to predict the prices, allowing the sellers to classify the device's prices according to their characteristics
● **SpringBoot project:** Will contain a simple entity, and a few endpoints, to call the service from the Python project for a bunch of test cases, and store them.

I) **Python project:**

1) Data preparation:

As we have training data and testing data, after we read the data, we need to check it, and this check includes:

- ✓ Duplication check and remove if exist, but no Duplication was found in our data.
- ✓ Nulls check and remove, with function 'check_remove_nulls', after this function was applied, I found that 0.45 % of data rows contains null, which is less than 1%, so this rows with nulls can be deleted as it is a small sample, only 9 rows were deleted.
- ✓ data type check (Numerical or categorical), with function 'check_categorical_exist', No categorical data was founded.
- ✓ data columns view: there is device id in testing data, which is not existed in training data, this column will be removed from testing data when it will be used for testing, as it is not related to training.
- ✓ data statistics: we need to show some statistics, to know our data more, and to see if we need to scale it. Fig1 show that there is high difference between values, for example col[0] & col [1], this data should be scaled, to get better performance in classification. Standard scaler was used later training part, mean value subtraction and dividing by std.

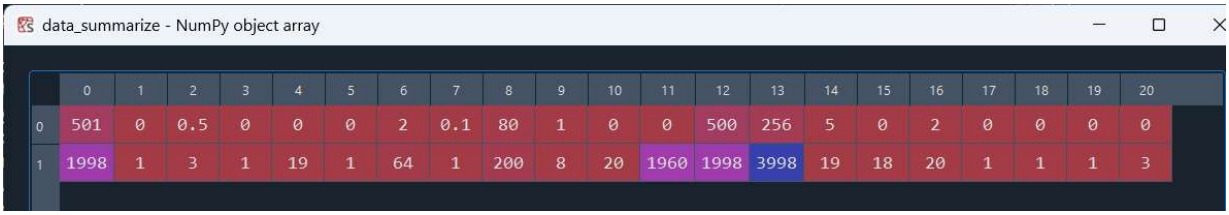

Fig1: min, max data values for reach column

- ✓ Check the balance of classes: with some classifiers like Neural Networks classifiers, the balance of samples in each class is very important. Neural Networks classifiers are famous for their accuracy, so we would like to try them. In order to check the balance of classes, we visualized the Histogram in Fig2, this figure show that all classes have close numbers of samples.

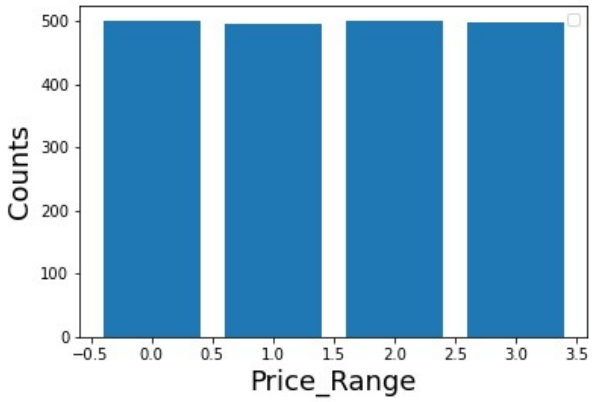

Fig2 Histogram of classes

- ✓ Visualize Features: for more analysis, we use a function to plot 2 features, on x,y, with colored classes. The function 'plot_two_features' will do that. But no interested coorelation were detected between features to delete any of them.

Now after all data were processed, we can go to training

2) Training and validation

we will train two classifiers, compare results, and choose the best, the first one is Feed Forward Neural Network 'ffnn', its known with its high accuracy. The second one is random forest 'rf', a Decision Tree based classifer, it gives fast results.

As we have linear data, with one dimension, we think there is no need for deep learning and CNN, Feed Forward Neural Network or FC with two layers will be enough.

We choose a Feed Forward Neural Network with two layers (one hidden layer), more details for the used model can be shown in function 'get_ffnn_model', and we change the parameters: number of hidden units, optimizer, activation function, e_pochs, batch_size, to get the best performance for FFNN. For example, optimizing hidden units' number, has increased the accuracy 1.5%.

We used **5 folds cross validation** to check the performance, so training data will be divided into 5 parts, in each time we train for 4 parts, and testing the remaining part, then we collect the results.

After the end of cross validation, we retrain the classifier with all training data and save the model in desired folder.

In functions.py code, 'train_validate_and_store_model' function will do all training, cross validation, metrics calculation, model storing.

Figure 3 shows confusion matrix for Random Forest Classifier, with total accuracy 88%.

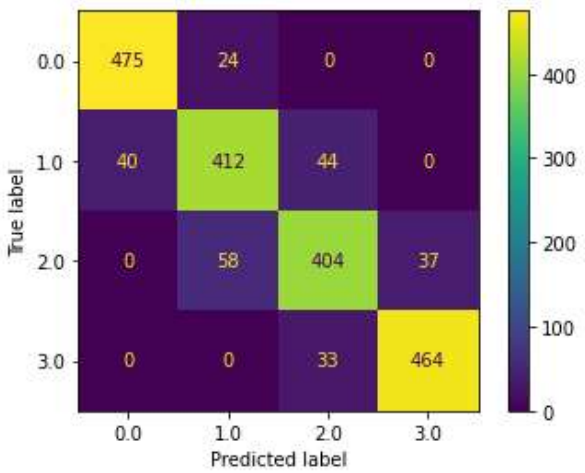

Fig3: RF Cross Validation Accuracy 88%

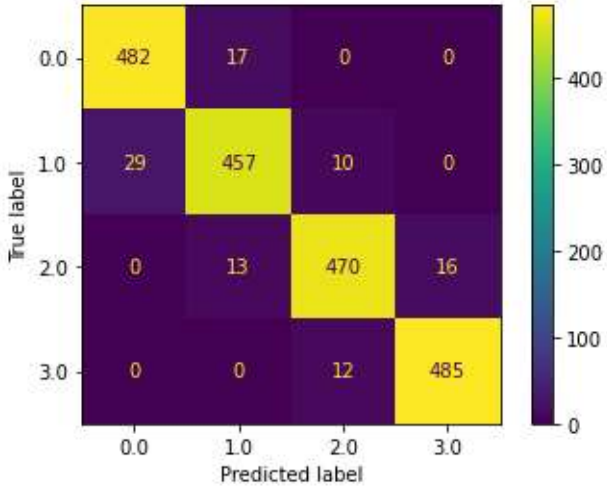

Fig4: FFNN Cross Validation Accuracy 95%

FFNN classifier outperforms the RF classifier. Also confusion matrix in Fig4 show as important result: the confusion is only with neighbor classes, exa: very_high_cost device(class 3) might ,with low probability, be classified as high_cost device (class2), but can never classified as medium_cost or low_cost, and that result is the same for all classes, this zeros in the corners are great for our problem. Besides the high accuracy.

So we decided to use FFNN classifier for this problem, as it gives better accuracy, and it is fast enough for our problem where there is no complicated data.

3) Testing

The last part in the main code will show the results for 10 devices from testing set, Fig5.

```
the prediction of sample 100 is : very_high_cost
the prediction of sample 101 is : medium_cost
the prediction of sample 102 is : low_cost
the prediction of sample 103 is : low_cost
the prediction of sample 104 is : very_high_cost
the prediction of sample 105 is : low_cost
the prediction of sample 106 is : high_cost
the prediction of sample 107 is : medium_cost
the prediction of sample 108 is : medium_cost
the prediction of sample 109 is : high_cost
```

Fig5: prediction results for 10 samples from testing set

4) RESTfulAPI

To make RESTful API with python, we can use Flask, Django, or FastAOI. We will just use Flask.

Let us put the input in json format, to simulate real case of input:


```
{
"id":1,
"battery_power ":1043,
"blue": 1,
"clock_speed": 1.8,
"dual_sim": 1,
"fc":14,
"four_g": 0,
"int_memory":5,
"m_dep":0.1,
"mobile_wt":193,
"n_cores":3,
"pc":16,
"px_height":226,
"px_width":1412,
"ram":3476,
"sc_h":12,
"sc_w":7,
"talk_time":2,
"three_g":0,
"touch_screen":1,
"wifi":0
}
```


Fig6: Json input, desired device to predict feature.

By running app.py with flask, we can see on browser the features of desired device, can be changed from json file, and the expected price calculated by our model, would appear at the end, it is “very_high_cost” for this example.