MOBILE PHONE PRICE RANGE CLASSIFICATION

GROUP NO: 7

NAME OF STUDENTS:
1) Aldrin Çifliku
2) Noah Paçuku
3) Eno Bendi
4) Elgin Belalla

EPOKA UNIVERSITY
DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

CONTENTS

INTRODUCTION

Picking the price of a mobile phone is not an easy task, especially if you are neither the producer of the phone nor the first hand vendor. Therefore, there is a need to distinguish a way of finding the appropriate price for each mobile phone device. In general each mobile phone can be assigned to one of these four price ranges: low, medium, high, and very high. Moreover, each mobile phone comes with a set of features and specifications such as the RAM memory, CPU, front and back camera quality, and so on. All these features play a role in determining the mobile phone's price range. In this project, machine learning techniques will be applied to try and make predictions about the price range of a phone. Various classification techniques, artificial neural networks and hybrid models will be trained and tested using a dataset which is provided by a shop owner and contains the information about the above mentioned features and their corresponding price ranges. All the models will be trained with different normalized versions of the dataset and with different parameters or structures (in case of MLP, Ensembles) to see in which conditions they perform better. The aim of the project is to find:

1) Under which normalization and feature engineering each model works the best
2) Which model has the best accuracy at predicting price ranges

The models with better accuracy can then be utilized for the prediction of the price range of real life mobile phones by simply providing their features.

METHODOLOGY

In this project, various supervised learning techniques will be used on a dataset of mobile phone instances with the aim of achieving high accuracy prediction of the price range for a mobile phone. The project mainly consists in three steps:

1) Data manipulation which includes feature engineering and application of various normalization techniques.
2) Training each model with specific parameters or structures (eg. trying MLP with different number of nodes in the hidden layer)
3) Testing the model and calculating accuracy metrics to see which of the models perform better, and under which conditions does each model perform the best.
4) Taking the best performing models and applying them to a real life app.

Testing and training data are taken from the same dataset using python libraries to do the random selection. For every model and for the data manipulation steps python will be

used. Mainly the sklearn, pandas, and tensorflow libraries. Other libraries might be employed as well for auxiliary tasks.

STATE OF THE ART

There were no scientific papers that had made use of our dataset however there were a few notebooks in the Kaggle platform. We have chosen to show for comparison a notebook provided by Mukul Saini, Data Scientist. In our study we have excluded kNN and included multiple other classification and hybrid techniques. The table below shows the results achieved by Mukul Saini using 6 classification techniques. It is noteworthy that the normalization technique in his study is z-score normalization. No feature engineering was performed by Saini which makes us believe that our results can be slightly better.

Methodology	Author	Accuracy	Precision	Recall	F1-score
Logistic regression	Mukul Saini	0.9625	0.99	0.96	0.98
Random forest	Mukul Saini	0.87	0.97	0.93	0.95
Decision tree	Mukul Saini	0.8425	0.93	0.9	0.92
SVM	Mukul Saini	0.88	0.98	0.9	0.94
kNN	Mukul Saini	0.5	0.67	0.66	0.66
Naive Bayes	Mukul Saini	0.8	0.96	0.89	0.92

DATA MANIPULATION

Our initial dataset consisted of 2000 instances with 500 instances for each price range. That meant that the dataset was well balanced and contained a reasonable amount of data objects. So, we decided not to perform any sampling or instance reduction on our dataset. We did however notice that there are attributes of the dataset that might have a logical correlation with each other such as for example screen height and screen width. Therefore, it was important to perform some feature engineering and see if the results for each model can be improved. There were 2 proposed versions of the dataset with feature engineering as explained below.

1. A dataset where the features screen_width and screen_height are combined to form a new feature and then are dropped from the dataset. This could be done in 2 different ways: we could create a new column that represents the diagonal length of the screen calculated as sqrt(screen_width^2 + screen_height^2) or we could create a column that represents the area of the screen called screen_size and calculated as screen_width * screen_height. We opted for the second option. This

proposed version of the dataset is focused on improving the accuracy of prediction rather than the complexity.

2. A dataset where the features screen_width, screen_height, px_height, and px_width are combined to form a single attribute with the goal of lowering the complexity of the data. The proposed new attribute is screen_resolution which is calculated using the formula resolution = (pixel_height / screen_height) * (pixel_width / screen_width).

Both these proposed datasets and the raw datasets will undergo normalization before being used to train the models. Since we are using various types of models we will also be using various normalization techniques and recording their performance. The normalization techniques we have chosen are: decimal scaling, z_score normalization and min - max normalization.

DECISION TREE

The decision tree model has one important parameter that can be switched to achieve different results. This parameter is the criterion of splitting nodes. For a decision tree classifier the parameter can take 3 values (in python): gini, entropy, and logistic loss. A decision tree model was built using each of these impurity metrics and was applied to each type of normalized dataset. The results are as shown below:

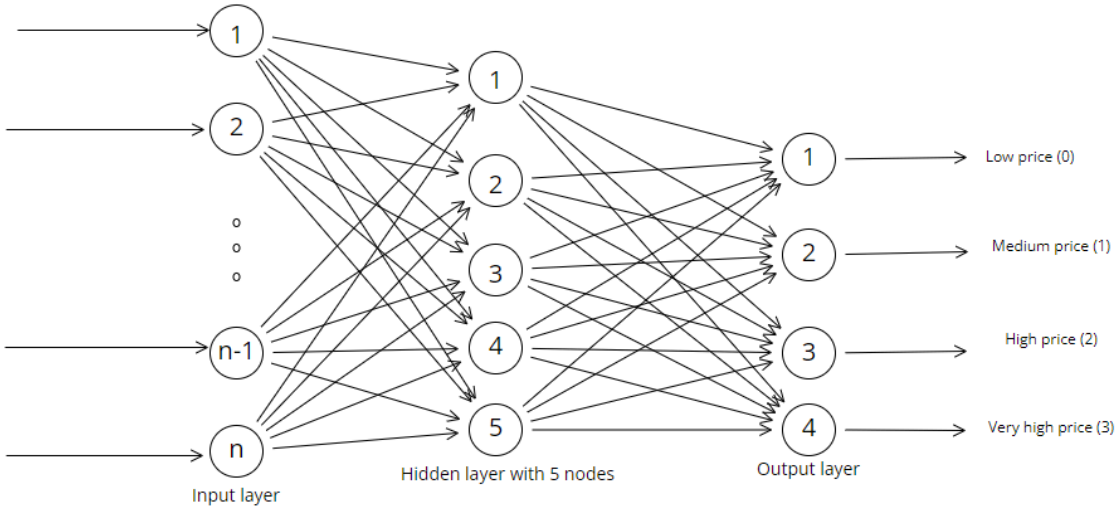
	Gini	Logistic loss	Entropy
Original dataset with min-max normalization	0.85	0.82	**0.84**
Original dataset with z_score normalization	0.8475	0.825	0.8325
Original dataset with decimal scaling	0.85	**0.835**	0.82
Proposed dataset with resolution min-max normalized	0.7475	0.765	0.765
Proposed dataset with resolution z_score normalized	0.735	0.7675	0.77
Proposed dataset with resolution decimal scaling applied	0.745	0.7775	0.765
Proposed dataset with screen size min-max normalized	0.865	0.82	**0.84**
Proposed dataset with screen size z_score normalized	0.8625	0.8325	0.8175

Proposed dataset with screen size decimal scaling applied	**0.8675**	**0.835**	**0.84**
Raw dataset	0.845	0.8225	0.8225

The maximal accuracy achieved by the model was 0.8675 and this was achieved when the model used "gini" as a splitting criterion and was trained on the dataset with the proposed feature engineering which replaces the screen width and height with the screen size. Decimal scaling was the normalization technique. The proposed dataset with the resolution feature did not perform well and made the accuracy of the model lower than it formerly was. Nonetheless, it did improve the speed of execution slightly. The conclusion drawn was that the decision tree model that performs the best is the one that uses gini as a splitting criterion and which is trained on the proposed dataset with screen size and normalized with decimal scaling.

MULTI-LAYER PERCEPTRON

For this study the multi-layer perceptron structure is as shown in the picture below. The number of nodes on the hidden layer was taken as five after performing some trials on the raw dataset and achieving the best result with this number of nodes. For our classification task a single hidden layer was enough. Notice how on the diagram the input layer has been shown to have n nodes. This is because the same MLP structure will be tested on the original dataset and on the datasets on which feature engineering was performed. Thus the value of n can range from 18 to 20.



The back propagation algorithm was used with an adaptive learning rate with initial values from 0.001 to 0.00005 and a momentum which was kept static at 0.9.

The algorithm was already provided by the python sklearn library. The parameters that were changed in order to compare results and improve accuracy were the optimizer, initial learning rate, and activation functions. The choices of optimizer in python are Adam (Adaptive moment estimation) and SGD (Stochastic Gradient Descent). For the activation function the choices were: Identity, Logistic (sigmoid), Hyperbolic tangent, and ReLu (Rectified linear unit). The learning rate as mentioned above was changed in the range 0.001 to 0.00005 The results for each are presented below. Each table specifies at the top the optimizer and the learning rate

MULTI-LAYER PERCEPTRON WITH ADAM OPTIMIZER (0.001)				
	Identity	**Logistic(sigmoid)**	**TanH**	**ReLu**
Original dataset with min-max normalization	0.975	0.97	0.9775	**0.98**
Original dataset with z_score normalization	0.9775	0.975	**0.9825**	0.9775
Original dataset with decimal scaling	**0.985**	**0.98**	0.975	0.97
Proposed dataset with resolution min-max normalized	0.8025	0.8125	0.81	0.8075
Proposed dataset with resolution z_score normalized	0.795	0.8075	0.8	0.81
Proposed dataset with resolution decimal scaling applied	0.815	0.815	0.805	0.815
Proposed dataset with screen size min-max normalized	0.9725	0.9675	0.9725	0.9725
Proposed dataset with screen size z_score normalized	0.9775	0.975	0.9775	0.975
Proposed dataset with screen size decimal scaling applied	0.9775	0.9775	0.975	0.975
Raw dataset	0.75	0.595	0.5925	0.815

MULTI-LAYER PERCEPTRON WITH ADAM OPTIMIZER (0.0005)				
	Identity	**Logistic(sigmoid)**	**TanH**	**ReLu**
Original dataset with min-max normalization	0.975	0.9775	0.9725	0.955
Original dataset with z_score normalization	0.975	**0.985**	**0.975**	0.96
Original dataset with decimal scaling	0.9725	0.97	0.9725	0.96
Proposed dataset with resolution min-max normalized	0.81	0.81	0.8075	0.8025

Proposed dataset with resolution z_score normalized	0.805	0.81	0.8	0.7975
Proposed dataset with resolution decimal scaling applied	0.815	0.815	0.8075	0.8
Proposed dataset with screen size min-max normalized	0.9725	0.97	0.9725	**0.9675**
Proposed dataset with screen size z_score normalized	**0.9775**	0.9725	0.9725	0.9525
Proposed dataset with screen size decimal scaling applied	0.9725	0.97	0.9725	0.9675
Raw dataset	0.6475	0.625	0.61	0.7175

MULTI-LAYER PERCEPTRON WITH ADAM OPTIMIZER (0.00005)				
	Identity	**Logistic(sigmoid)**	**TanH**	**ReLu**
Original dataset with min-max normalization	0.7425	0.2725	0.7175	0.3125
Original dataset with z_score normalization	0.8675	0.2725	0.815	**0.865**
Original dataset with decimal scaling	0.295	0.275	0.3025	0.235
Proposed dataset with resolution min-max normalized	0.7025	0.275	0.7025	0.29
Proposed dataset with resolution z_score normalized	0.7725	0.2625	0.7675	0.7725
Proposed dataset with resolution decimal scaling applied	0.2575	0.2625	0.2675	0.26
Proposed dataset with screen size min-max normalized	0.77	0.29	0.7375	0.345
Proposed dataset with screen size z_score normalized	**0.895**	**0.3475**	**0.8375**	0.85
Proposed dataset with screen size decimal scaling applied	0.26	0.28	0.265	0.215
Raw dataset	0.65	0.515	0.5575	0.3425

While using the adam optimizer the best accuracy achieved was 0.985. This was achieved on 2 occasions. The first one, when using identity activation function with a learning rate of 0.001 and the original dataset with decimal scaling. The second one, while using the logistic activation function with an initial learning rate of 0.0005 and the original dataset with z score normalization. It is important to note that when switching from an initial learning rate of 0.001 to an initial learning rate of 0.0005 there are very small differences

in results. The model is largely consistent in prediction. However, when switching to an initial learning rate of 0.00005 the model becomes unstable, inconsistent and yields very erroneous predictions. As can be seen the accuracy even went as low as 0.215 in one case with that initial learning rate.

MULTI-LAYER PERCEPTRON WITH SGD OPTIMIZER (0.001)				
	Identity	**Logistic(sigmoid)**	**TanH**	**ReLu**
Original dataset with min-max normalization	0.965	0.2925	0.98	0.96
Original dataset with z_score normalization	**0.9775**	**0.95**	**0.985**	**0.9725**
Original dataset with decimal scaling	0.91	0.2425	0.9175	0.23
Proposed dataset with resolution min-max normalized	0.8125	0.2875	0.82	0.815
Proposed dataset with resolution z_score normalized	0.81	0.81	0.8275	0.81
Proposed dataset with resolution decimal scaling applied	0.8025	0.2375	0.8	0.8
Proposed dataset with screen size min-max normalized	0.96	0.2875	0.9625	0.9575
Proposed dataset with screen size z_score normalized	0.975	0.94	0.98	**0.9725**
Proposed dataset with screen size decimal scaling applied	0.9325	0.2475	0.9025	0.915
Raw dataset	0.28	0.23	0.2275	0.23

MULTI-LAYER PERCEPTRON WITH SGD OPTIMIZER (0.0005)				
	Identity	**Logistic(sigmoid)**	**TanH**	**ReLu**
Original dataset with min-max normalization	0.9375	0.2325	0.9375	0.9375
Original dataset with z_score normalization	**0.985**	0.95	0.9725	**0.9675**
Original dataset with decimal scaling	0.855	0.2625	0.845	0.265
Proposed dataset with resolution min-max normalized	0.81	0.2725	0.805	0.8075
Proposed dataset with resolution z_score normalized	0.8	0.8075	0.81	0.7975
Proposed dataset with resolution decimal scaling applied	0.785	0.23	0.785	0.79
Proposed dataset with screen size min-max normalized	0.9475	0.2675	0.9425	0.9375

Proposed dataset with screen size z_score normalized	0.9675	**0.9675**	**0.975**	0.95
Proposed dataset with screen size decimal scaling applied	0.8525	0.2225	0.85	0.86
Raw dataset	0.5425	0.54	0.53	0.2825

MULTI-LAYER PERCEPTRON WITH SGD OPTIMIZER (0.00005)				
	Identity	**Logistic(sigmoid)**	**TanH**	**ReLu**
Original dataset with min-max normalization	0.7425	0.2725	0.7175	0.3125
Original dataset with z_score normalization	0.8675	0.2725	0.815	**0.865**
Original dataset with decimal scaling	0.295	0.275	0.3025	0.235
Proposed dataset with resolution min-max normalized	0.7025	0.275	0.7025	0.29
Proposed dataset with resolution z_score normalized	0.7725	0.2625	0.7675	0.7725
Proposed dataset with resolution decimal scaling applied	0.2575	0.2625	0.2675	0.26
Proposed dataset with screen size min-max normalized	0.77	0.29	0.7375	0.345
Proposed dataset with screen size z_score normalized	**0.895**	0.3475	**0.8375**	0.85
Proposed dataset with screen size decimal scaling applied	0.26	0.28	0.265	0.215
Raw dataset	0.65	**0.5575**	0.515	0.3425

When using the SGD optimizer it can be noticed that the model generally yields good results but is very inconsistent with the logistic activation function. This is due to the fact that this function saturates for very small values of input. As can be seen the z score normalization manages to nerf this negative effect of saturation by distributing values in such a way that the variance is almost 0. The best result achieved is 0.985 when using 0.001 as initial learning rate and TanH activation function on the original dataset with z score normalization. The same result was replicated using an initial learning rate of 0.0005 and an identity activation function on the original dataset with z score normalization. As it happened with the ‘adam’ optimizer the accuracy of the model starts to fall as the initial learning rate is decreased.

Conclusions drawn from this part of the study are:

1) MLP works better without any feature engineering. Feature engineering might slightly improve the time and space efficiency though.

2) The best performing conditions for the model are:
    a) TanH activation function, SGD optimizer applied on a dataset without feature engineering but with z-score normalization (learning rate 0.001).
    b) Identity activation function, Adam optimizer applied on a dataset without feature engineering but with decimal scaling (learning rate 0.001).
    c) Identity activation function, SGD optimizer applied on a dataset without feature engineering but with z -score normalization (learning rate 0.0005).
    d) Logistic activation function, Adam optimizer applied on a dataset without feature engineering but with z-score normalization (learning rate 0.0005).

3) Normalization is crucial for the MLP to work well.This can be noticed by the stark differences in accuracy of the model when trained and tested with the raw dataset and with the other normalized datasets.

SUPPORT VECTOR MACHINE

The Support vector machine model has two parameters that can be switched in order to achieve the highest accuracy: the kernel and the degree (if the kernel is polynomial). The kernels chosen were: polynomial to the third degree, linear, radial basis function and sigmoid. Results are as follow:

	Polynomial (degree=3)	**Linear**	**Radial Basis Function**	**Sigmoid**
Original dataset with min-max normalization	0.895	0.9375	0.8575	0.21
Original dataset with z_score normalization	0.8325	**0.965**	0.875	0.9
Original dataset with decimal scaling	0.7925	0.865	0.65	0.2525
Proposed dataset with resolution min-max normalized	0.775	.84	0.7825	0.2425
Proposed dataset with resolution z_score normalized	0.7575	0.8325	0.7975	0.7775
Proposed dataset with resolution decimal scaling applied	0.7275	0.8	0.64	0.2575
Proposed dataset with screen	0.895	**0.9375**	0.865	0.2125

size min-max normalized				
Proposed dataset with screen size z_score normalized	0.815	0.96	0.8725	0.9025
Proposed dataset with screen size decimal scaling applied	0.7875	0.865	0.65	0.245
Raw dataset	**0.95**	**0.965**	**0.95**	0.2175

The peak accuracy of this model was 0.965 achieved in two instances, both using the linear kernel. On the first occurrence the dataset chosen was the original one, using z_score normalization, whereas on the second occurrence we were dealing with the raw dataset. Runner ups are the values from the raw dataset being trained on the polynomial kernel as well as the RBF kernel, both being 0.95

Throughout all training the linear kernel was the most consistent in achieving high accuracy, but both the polynomial and RBF kernel are consistent in achieving high accuracy as well. On the other hand the sigmoid kernel is quite inconsistent, with values varying from 0.2 to 0.9.

GAUSSIAN NAIVE BAYES

The Naïve Bayes model doesn't really have important parameters that can be switched to achieve different results, such as decision trees or random forest. This is mostly because other Naïve Bayes functions do not work well for multiple classes. So, instead of changing parameters, we use different training sizes to see how it can affect the accuracy of the original and normalized datasets. Results are as shown below:

	0.6	**0.7**	**0.8**	**0.9**
Original dataset with min-max normalization	0.79625	0.79867	0.8	**0.82**
Original dataset with z_score normalization	0.79625	0.79867	0.8	**0.82**
Original dataset with decimal scaling	0.79625	0.79867	0.8	**0.82**
Proposed dataset with resolution min-max normalized	0.75375	0.75874	0.7675	**0.765**
Proposed dataset with resolution z_score normalized	0.75375	0.75874	0.7675	**0.765**
Proposed dataset with resolution decimal scaling applied	0.75375	0.75874	0.7675	**0.765**
Proposed dataset with screen size min-max normalized	0.79125	0.80532	0.7975	**0.82**

Proposed dataset with screen size z_score normalized	0.79125	0.80532	0.7975	**0.82**
Proposed dataset with screen size decimal scaling normalized	0.79125	0.80532	0.7975	**0.82**
Raw Dataset	0.7975	0.80033	0.8	**0.82**

What is interesting about this model is that the different types of normalization do not change the outcome of accuracy within the dataset. So for example, even though we have used min-max, z-score and decimal scaling normalizations, the values in the original dataset remain the same. The maximal accuracy achieved by the model was 0.82 and this was achieved with the original, proposed with the screen size feature and raw datasets all with a training size of 0.9. In contrast, the proposed dataset with resolution feature yields the worst results compared to all others. Close runner ups with a value of 0.8 are again the same models as mentioned above, but with a training size of 0.8 instead of 0.9. We see that training size greatly affects accuracy.

LOGISTIC REGRESSION

The logistic regression model has a crucial parameter that can be adjusted to influence its performance. This parameter is the solver, representing the optimization algorithm used during training. In Python, logistic regression's solver parameter can take various values, each corresponding to a different optimization method. The available solvers include 'newton-cg', 'lbfgs', 'liblinear', 'sag', and 'saga'. Each solver has its characteristic and its suitable for different scenarios, datasets sizes, or regularization requirements. The following table showcases the impact of solver selection on the logistic regression model's performance.

	newton-cg	**lbfgs**	**liblinear**	**sag**	**saga**
Original dataset with min-max normalization	0.915	0.915	0.7875	0.915	0.915
Original dataset with z_score normalization	0.96	0.96	0.8225	0.96	0.96
Original dataset with decimal scaling	0.775	0.775	0.7175	0.7725	0.775
Proposed dataset with resolution min-max normalized	0.825	0.825	0.7625	0.825	0.825
Proposed dataset with resolution z_score normalized	0.8225	0.8225	0.775	0.8225	0.8225

Proposed dataset with resolution decimal scaling applied	0.7425	0.74	0.6975	0.74	0..74
Proposed dataset with screen size min-max normalized	0.9125	0.9125	0.79	0.9125	0.912 5
Proposed dataset with screen size z_score normalized	**0.965**	**0.965**	**0.835**	**0.965**	**0.965**
Proposed dataset with screen size decimal scaling applied	0.7775	0.7775	0.7325	0.7775	0.777 5
Raw dataset	0.675	0.6775	0.79	0.6775	0.675

The peak accuracy achieved by this model was 0.965 and this was achieved when the model was trained on the proposed dataset with screen size, and the normalization technique applied was z-score scaling. Notably, various solvers were employed during the logistic regression modeling process. Despite 'liblinear' being the least accurate among the solvers tested, the alternative solvers showed comparable and consistently high accuracy scores. Notably, these solvers, while different in their optimization approaches, exhibited similar and competitive performances.

It is noteworthy that when the dataset underwent decimal scaling, the model's accuracy witnessed a noticeable decline compared to other normalization techniques.This observation further underscores the critical importance of normalization techniques in influencing logistic regression performance.

RANDOM FOREST

The Random Forest model has in fact 2 important parameters that can be switched to achieve different results. They consist of criterions and estimators. To achieve as accurate of a result as we could, we used these 2 parameters in different forms to get as many combinations as possible, thus achieving the best accuracy. The criterions we used are "Gini" and "Entropy" and the number of estimators used are 100, 500 and 1000. Results are as shown below:

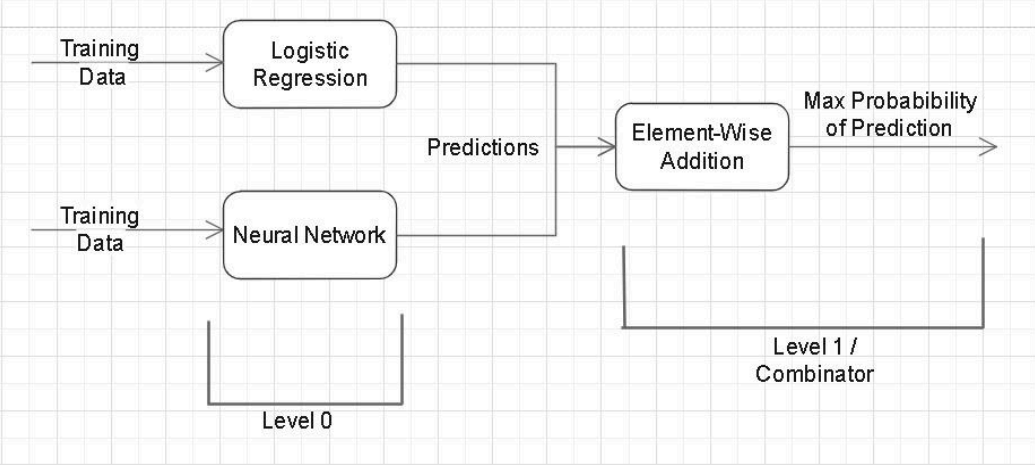
	Gini 100	**Gini 500**	**Gini 1000**	**Entropy 100**	**Entropy 500**	**Entropy 1000**
Original dataset with min-max normalization	0.8775	0.8725	0.875	0.8825	0.885	0.875

Original dataset with z_score normalization	0.875	0.875	0.8875	0.88	0.87	0.885
Original dataset with decimal scaling	0.8775	0.8775	0.8775	0.8625	0.8775	0.88
Proposed dataset with resolution min-max normalized	0.8025	0.8	0.8025	0.7975	0.8025	0.815
Proposed dataset with resolution z_score normalized	0.81	0.805	0.8125	0.8125	0.8	0.8025
Proposed dataset with resolution decimal scaling applied	0.8025	0.81	0.8075	0.8075	0.8025	0.805
Proposed dataset with screen size min-max normalized	0.88	0.875	0.8825	0.8725	0.8775	0.8875
Proposed dataset with screen size z_score normalized	**0.855**	**0.895**	**0.885**	**0.8675**	**0.875**	**0.885**
Proposed dataset with screen size decimal scaling normalized	**0.865**	**0.8775**	**0.885**	**0.8825**	**0.875**	**0.895**
Raw Dataset	0.855	0.88	0.875	0.885	0.88	0.8675

Overall, the level of accuracy is quite high amongst random forest, but there are a couple of distinguishable cases. The maximal accuracy achieved by the model was 0.895 and this was achieved in 2 instances: The 1st being using the proposed dataset with screen size feature and z-score normalization, with "Gini" criterion and 500 estimators. The 2nd being using the proposed dataset with screen size feature and decimal normalization, with "Entropy" criterion and 1000 estimators. It is worth noting that both these instances belonged to the proposed dataset with screen size.

Close runner ups are 2 instances; one of the original dataset with "Gini" criterion and 1000 estimators and the other of the proposed dataset with screen size feature, "Entropy" criterion and 1000 estimators. Again we see that the proposed dataset with resolution feature yields the results with the lowest accuracy.

ENSEMBLE MODEL LOGISTIC REGRESSION PLUS FEED FORWARD NEURAL NETWORK



In the ensemble Logistic Regression in order to observe the changes in accuracy we choose different kinds of solvers (as in the example previously done) for the logistic regression model. Once again the solvers chosen were: newton-cg, lbfgs, liblinear and sag. Results are as shown below:

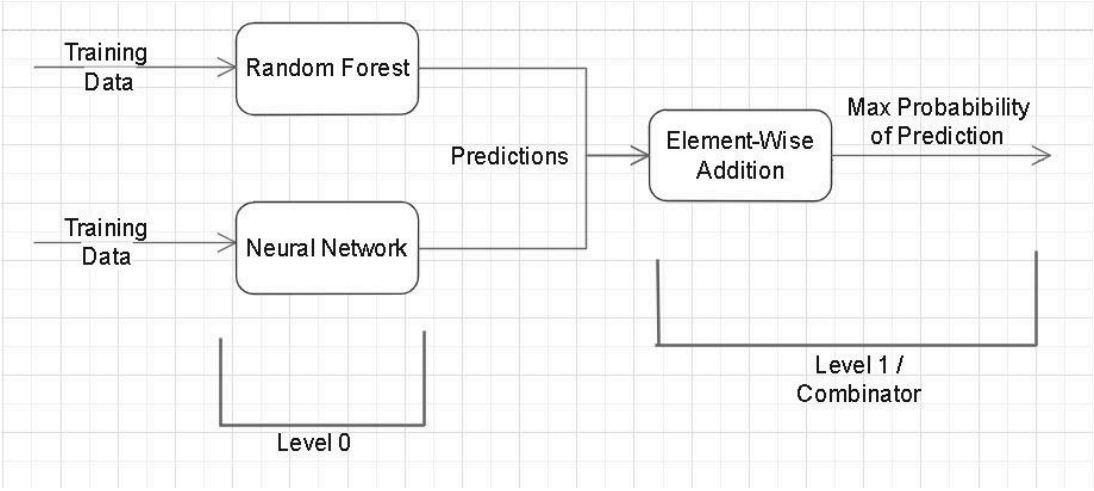
	newton-cg	lbfgs	liblinear	sag
Original dataset with min-max normalization	0.95	0.95	0.9	0.94
Original dataset with z_score normalization	0.96	0.96	0.95	0.96
Original dataset with decimal scaling	0.89	0.9	0.9	0.89
Proposed dataset with resolution min-max normalized	0.82	0.81	0.8	0.81
Proposed dataset with resolution z_score normalized	0.81	0.81	0.8	0.81
Proposed dataset with resolution decimal scaling applied	0.79	0.79	0.8	0.8
Proposed dataset with screen size min-max normalized	**0.97**	0.96	0.95	**0.97**
Proposed dataset with screen size z_score normalized	0.96	**0.97**	0.96	**0.98**

Proposed dataset with screen size decimal scaling normalized	0.9	0.9	0.9	0.9
Raw Dataset	0.85	0.71	0.76	0.69

The highest accuracy, 0.98, achieved was using the proposed screen size dataset, with the z score normalization and using the sig solver. The three runner ups were all a result of this dataset, using a min max normalization with either a newton-cg solver or sag solver, as well as using the z-score normalization with the lbfgs solver.

The solver with the highest accuracy overall was the sag, however the values were pretty consistent throughout all solvers, as it seems the biggest difference in output was a result of the dataset used and to a lesser extent of the normalization used. We can clearly see that the proposed dataset with the resolution feature performs the worst after the raw dataset, meanwhile the proposed dataset with the screen size feature performs the best.

ENSEMBLE MODEL RANDOM FOREST PLUS FEED FORWARD NEURAL NETWORK



The ensemble model, combining Random Forest and Feed Forward Neural Network (FFNN), introduces key parameters that play a significant role in shaping its predictive capabilities. Two fundamental criteria for the decision tree component of the Random Forest—“Gini” and “Entropy”—alongside the number of estimators, set respectively at 100, 500, and 1000, serve as pivotal factors in the model's performance. The choice between Gini impurity and Entropy measures the quality of split decisions, influencing the overall decision tree structure within the ensemble. The number of estimators, representing the count of decision trees in the forest, also holds importance as it directly impacts the model's complexity and predictive power.

	Gini 100	Gini 500	Gini 1000	Entropy 100	Entropy 500	Entropy 1000
Original dataset with min-max normalization	0.95	0.94	0.96	0.94	0.95	0.93
Original dataset with z_score normalization	**0.96**	**0.96**	**0.96**	**0.97**	**0.96**	**0.96**
Original dataset with decimal scaling	0.91	0.91	0.93	0.93	0.93	0.93
Proposed dataset with resolution min-max normalized	0.82	0.81	0.81	0.81	0.81	0.82
Proposed dataset with resolution z_score normalized	0.8	0.81	0.81	0.81	0.81	0.81
Proposed dataset with resolution decimal scaling applied	0.82	0.81	0.81	0.81	0.81	0.82
Proposed dataset with screen size min-max normalized	0.94	0.96	0.94	0.94	0.95	0.95
Proposed dataset with screen size z_score normalized	**0.97**	**0.95**	**0.96**	**0.95**	**0.95**	**0.95**
Proposed dataset with screen size decimal scaling normalized	0.93	0.92	0.91	0.91	0.92	0.91
Raw Dataset	0.69	0.67	0.73	0.65	0.62	0.60

This ensemble model achieved a peak accuracy of 0.97. This was attained under two distinct configurations: first, when the model was trained on the original dataset with z-score normalization and the decision tree criteria set to Entropy with 100 estimators, and secon, when the proposed dataset featuring screen size was utilized with z-score normalization and the decision tree criteria set to Gini with 100 estimators. The lowest accuracy score was observed when utilizing the raw dataset and when employing the proposed dataset with resolution features. This emphasizes the importance of thoughtful feature engineering. Additionally, the dataset's original form without feature engineering exhibited lower accuracy, underlining the significance of feature enhancement in achieving superior predictive performance.

CONCLUSIONS

In conclusion to our project the model that worked the best in this classification task was the Multi-layer perceptron. It reached a peak accuracy of 0.985 in two cases. Logistic regression, Support Vector Machine and the two ensemble models were the models to get closer to MLP in terms of accuracy. Decision trees, Naive Bayes were only slightly worse.
As for the feature engineering performed on the dataset the proposed version with resolution feature failed to improve the accuracy of the model in any of the cases despite providing noticeable improvements to the computational speed of the algorithm. The proposed dataset with screen size feature, on the other hand, was superior in most cases leading us to believe that the feature engineering performed in this case was of high level.
For the normalization techniques it must be emphasized that z-score normalization achieved the best accuracy in most models followed by decimal scaling. Min-max normalization was the best normalization technique only in the Naive Bayes model. It is important to say however that the results with datasets normalized with different techniques are only slightly different. The models that were affected the most by normalization were MLP and Logistic regression where the improvement from the application to the raw dataset is drastic. The model that was affected the least was SVM. This model got the highest accuracy when applied to the raw dataset with a linear kernel.

With all the results in order we decided to pick the following models for our application:

1. Decision tree with gini criterion. Trained on the proposed dataset with screen size feature and decimal scaling.
2. Multi-layer perceptron with 5 nodes in the hidden layer, adam optimizer, identity activation function. Trained on the original dataset with decimal scaling.
3. Support vector machine with linear kernel. Trained on the raw dataset.
4. Gaussian Naive Bayes. Trained on the proposed dataset with screen size feature and decimal scaling.
5. Logistic regression with ‘saga’ solver and max iterations of 10000. Trained on the proposed dataset with screen size feature and z-score normalization.
6. Random forest with gini criterion and 500 estimators. Trained on the proposed dataset with screen size feature and z-score normalization.
7. Ensemble model: Logistic regression with ‘sag’ solver and Feed Forward Neural Network with 5 nodes in the hidden layer. Trained on the proposed dataset with screen size feature and z-score normalization.
8. Ensemble model: Random forest with gini criterion and 100 estimators and Feed Forward Neural Network with 5 nodes in the hidden layer. Trained on the proposed dataset with screen size feature and z-score normalization.

RESULT'S TABLE

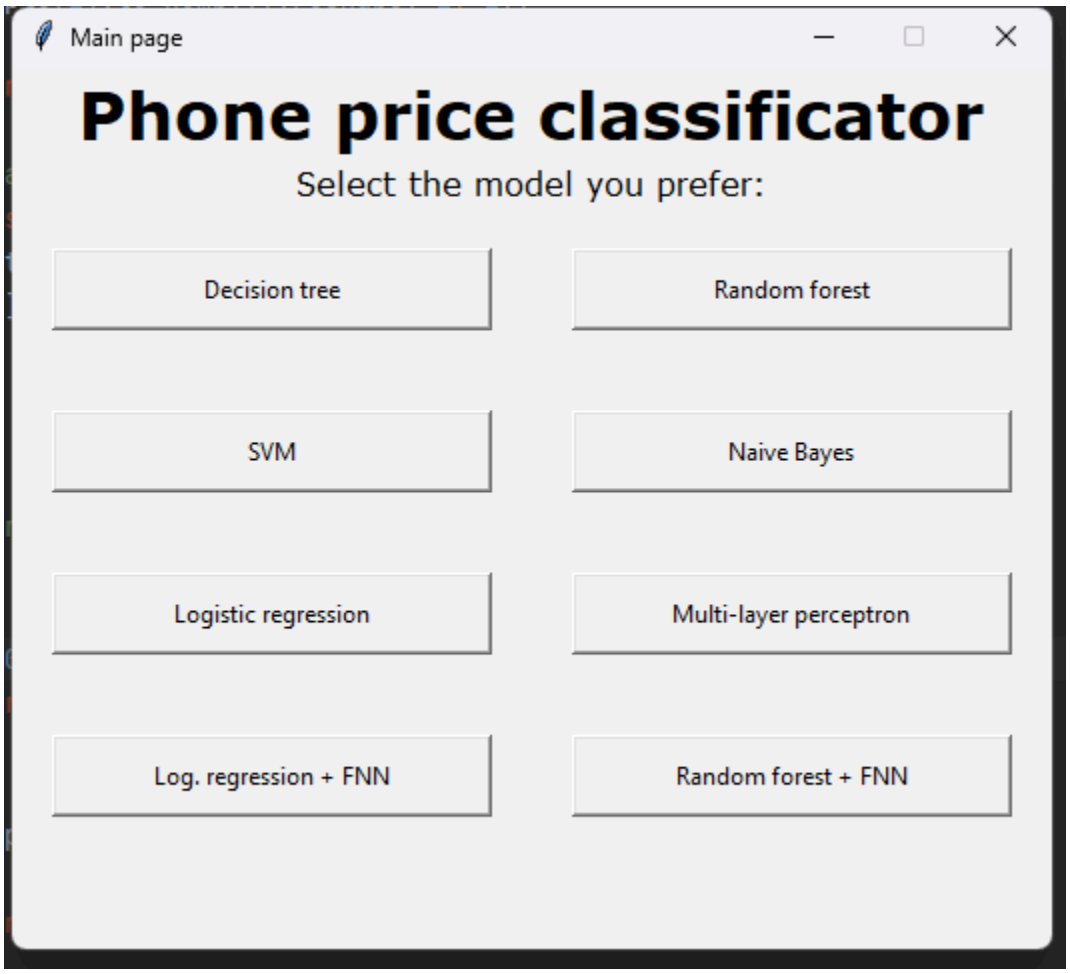
Below can be found a table that compares the best results of all models:

Model	Best accuracy
Decision tree	0.8675
Multi-layer perceptron	**0.985**
SVM	0.965
Naive Bayes	0.82
Logistic regression	0.965
Random forest	0.895
Ensemble: Logistic regression + FFNN	0.98
Ensemble: Random forest + FFNN	0.97

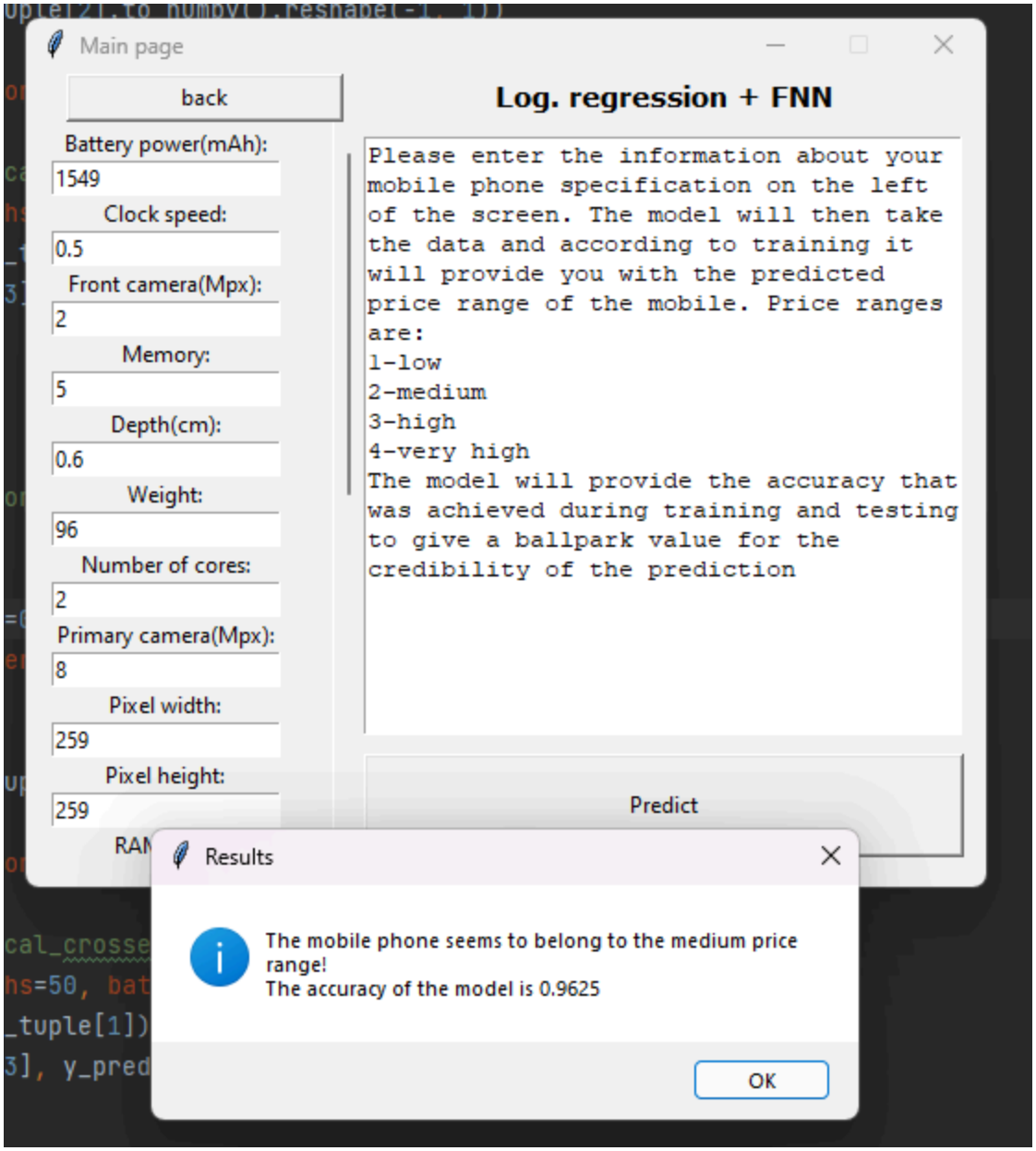
As can be seen the MLP is the best at accuracy followed by the ensemble models. The other models however performed quite well and achieved very good results.

GUI

For the application that should be usable by real users that want to classify new instances of mobile phones into price ranges we have created a simple GUI using tkinter. This app makes use of the models that were selected in the conclusion section based on their performance. The app provides the prediction and the accuracy of the model being used. Below you can find the images of our GUI.



The user can choose from the 8 models that were mentioned above. Of course the user is not provided the details of implementation. He/she should only be provided with the prediction and the accuracy expected. Below you can see how the results are displayed and the menu where the user enters the data of the new instance.



RESOURCES

The resources we have used are the dataset which was provided by Abhishek Sharma, Data Scientist at Grazitti Interactive through the platform Kaggle (Sharma, n.d.).

To code the algorithms and the interface we have used python 3.10 and many libraries of this language such as sklearn, pandas, tkinter, numpy, tensorflow, kerras, and so on. For these libraries we have referred to the official documentation (Python documenation, n.d.).

As our platform of work we have used pycharm for developing and executing algorithms and github for sharing data and keeping track of the code.
Initial information on the classification techniques on this dataset was taken by a Kaggle notebook provided by Mukul Saini (Sai, 2023)

REFERENCES

McSwine, D. (2023, November 7). .,. ScienceDirect. Retrieved January 6, 2024, from https://www.sciencedirect.com/topics/computer-science/ensemble-modeling

Python documenation. (n.d.). scikit-learn: machine learning in Python — scikit-learn 1.3.2 documentation. Retrieved January 6, 2024, from https://scikit-learn.org/stable/

Sai, M. (2023, December 3). *Applying 6 Classificaion Algorithms*. Kaggle. Retrieved January 6, 2024, from https://www.kaggle.com/code/mukulsaini01/applying-6-classificaion-algorithms

Sharma, A. (n.d.). *Mobile Price Classification*. Kaggle. Retrieved January 6, 2024, from https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification?select=train.csv