Topic: Price Classification of Mobile Phones

Data Science Capstone Project Report

George Washington University

Zhen Zhang

December 3,2018

1. Introduction

For this project, I examined the relationship between mobile phone features and price scale. I used the machine learning technique to predict the price range of a mobile phone in the competitive mobile phone market. In this project, I used Python to analyze and visualize data, and to build a model to predict the price range of mobile phones based on their features. In this report, I will introduce the data, describe the process of the project, and introduce the experimental setup, as well as presenting the results of the experiment.

2. Description of Data Set

The dataset that I used was taken from Kaggle. This dataset was a record of the phone prices of various companies. There are two datasets within this data; one is train.csv (2000 rows and 21 columns), another is test.csv (1000 rows and 21 columns). The limitation of this dataset is that the dataset only provided us a price range for the mobile phones rather than specific prices.

3. Experimental Setup

Initial Statistics Analysis and Exploratory Data Analysis

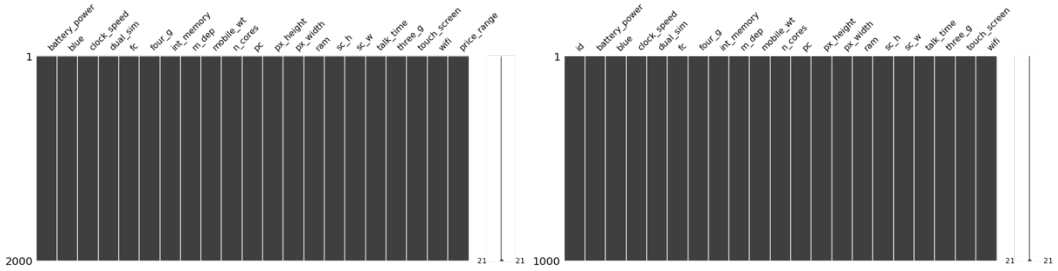
First, I completed a data overview and initial statistics analysis.

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ram	sc_h	sc_w	talk_time
0	842	no	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19
1	1021	yes	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7
2	563	yes	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9
3	615	yes	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11
4	1821	yes	1.2	0	13	1	44	0.6	141	2	...	1208	1212	1411	8	2	15

	id	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	...	pc	px_height	px_width	ram	sc_h	sc_w	talk_time
0	1	1043	yes	1.8	1	14	0	5	0.1	193	...	16	226	1412	3476	12	7	2
1	2	841	yes	0.5	1	4	1	61	0.8	191	...	12	746	857	3895	6	0	7
2	3	1807	yes	2.8	0	1	0	27	0.9	186	...	4	1270	1366	2396	17	10	10
3	4	1546	no	0.5	1	18	1	25	0.5	96	...	20	295	1752	3893	10	0	7
4	5	1434	no	1.4	0	11	1	49	0.5	108	...	18	749	810	1773	15	8	7

	count	mean	std	min	25%	50%	75%	max
battery_power	2000.0	1238.51850	439.418206	501.0	851.75	1226.0	1615.25	1998.0
clock_speed	2000.0	1.52225	0.816004	0.5	0.70	1.5	2.20	3.0
dual_sim	2000.0	0.50950	0.500035	0.0	0.00	1.0	1.00	1.0
fc	2000.0	4.30950	4.341444	0.0	1.00	3.0	7.00	19.0
four_g	2000.0	0.52150	0.499662	0.0	0.00	1.0	1.00	1.0
int_memory	2000.0	32.04650	18.145715	2.0	16.00	32.0	48.00	64.0
m_dep	2000.0	0.50175	0.288416	0.1	0.20	0.5	0.80	1.0
mobile_wt	2000.0	140.24900	35.399655	80.0	109.00	141.0	170.00	200.0
n_cores	2000.0	4.52050	2.287837	1.0	3.00	4.0	7.00	8.0
pc	2000.0	9.91650	6.064315	0.0	5.00	10.0	15.00	20.0
px_height	2000.0	645.10800	443.780811	0.0	282.75	564.0	947.25	1960.0
px_width	2000.0	1251.51550	432.199447	500.0	874.75	1247.0	1633.00	1998.0
ram	2000.0	2124.21300	1084.732044	256.0	1207.50	2146.5	3064.50	3998.0
sc_h	2000.0	12.30650	4.213245	5.0	9.00	12.0	16.00	19.0
sc_w	2000.0	5.76700	4.356398	0.0	2.00	5.0	9.00	18.0
talk_time	2000.0	11.01100	5.463955	2.0	6.00	11.0	16.00	20.0
three_g	2000.0	0.76150	0.426273	0.0	1.00	1.0	1.00	1.0
touch_screen	2000.0	0.50300	0.500116	0.0	0.00	1.0	1.00	1.0
price_range	2000.0	1.50000	1.118314	0.0	0.75	1.5	2.25	3.0

Then, I performed an exploratory data analysis. During this step, I checked for missing values initially by using the ‘is.null()’ function. Then, I used the ‘msno.matrix’ function to build a nullity matrix to visualize the patterns of missing values.



Fortunately, there were no missing values in my data that needed to be addressed.

In the data overview section, I found that the “wifi” and “blue” columns had categorical features. Therefore, I needed to convert these columns to numeric columns. I got the format for values in each object column and explored the number of unique values in each object column. From this step, I found that the values in the object columns were the nominal values as following:

```
blue     no
wifi    yes
Name: 0, dtype: object
```

Therefore, I encoded them as dummy variables by using the pandas function ‘pd.get_dummies’ to create a new dataframe which consists of zeros and ones.

	blue_no	blue_yes	wifi_no	wifi_yes
0	1	0	0	1
1	0	1	1	0
2	0	1	1	0
3	0	1	1	0
4	0	1	1	0

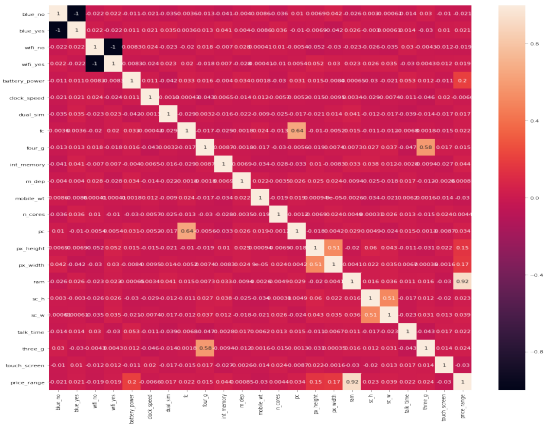
After getting the new dataframe, I put it together with the old dataset, using the ‘pd.cancat’ function to concentrate them.

Because there were too many columns in the dataset, I tried to narrow the columns. First, I worked on the deletion of columns. In this step, I trained the random forest model for raw data to check whether or not there were useless columns.

	ram	battery_power	px_height	px_width	int_memory	mobile_wt	talk_time	pc	sc_w	clock_speed	...
0	0.408939	0.071638	0.0637	0.06189	0.044909	0.041321	0.040772	0.033706	0.03242	0.031636	...

sc_h	n_cores	dual_sim	four_g	touch_screen	wifi_no	blue_yes	wifi_yes	blue_no	three_g
0.027858	0.027434	0.008902	0.008318	0.007771	0.007354	0.006522	0.006326	0.005779	0.005191

From the output, we can clearly see that there are no useless columns, and that “ram” and “battery_power” are the most important feature for the price range from the random forest classifier. I then calculated the correlation of each indicator to see whether or not there were linearly correlated columns and used a heatmap to visualize the result.

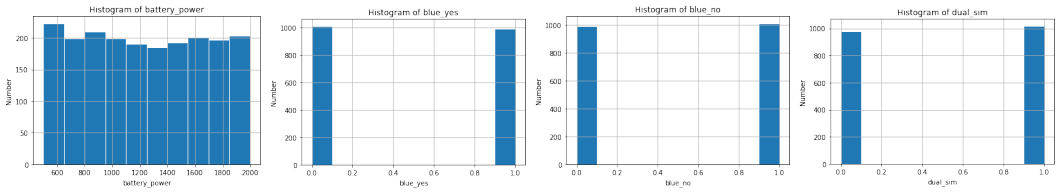


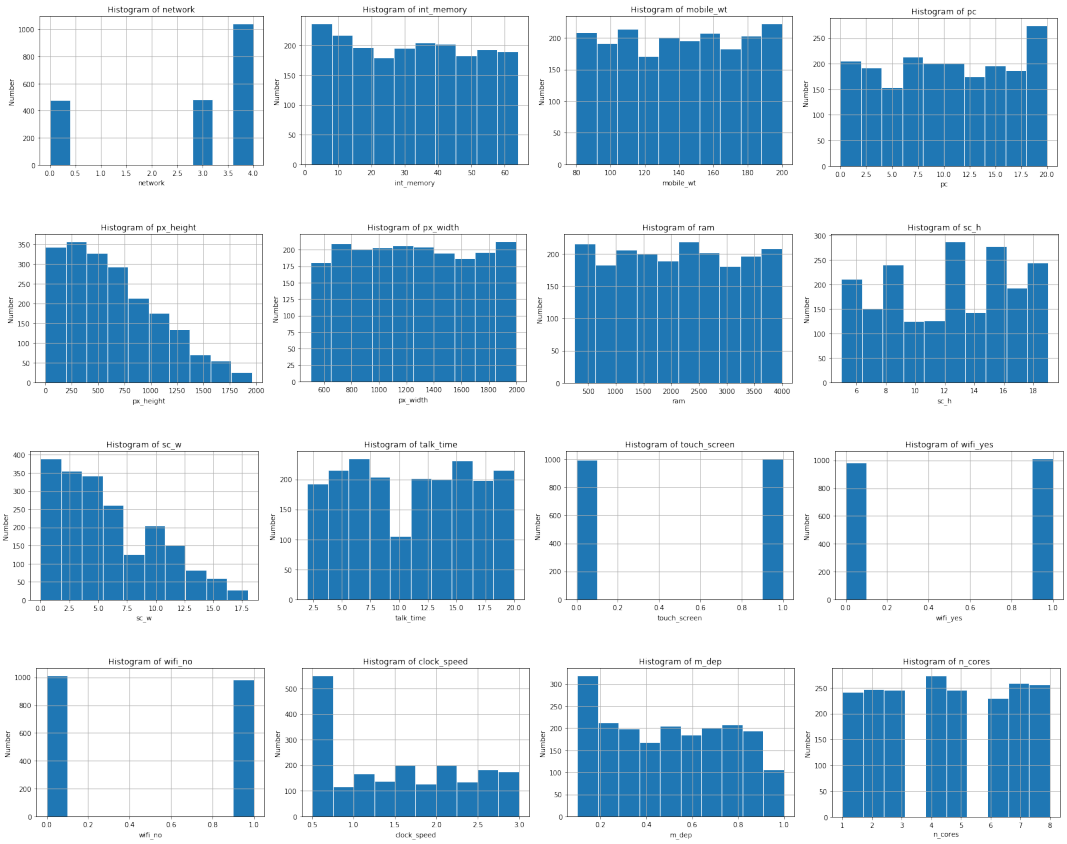
In this step, I found that the indicators "pc" and "fc" have a strong correlation (absolute value is greater than 0.6), which meant that I can use one of them to predict the other. Therefore, I only needed to keep one of them, and I decided to drop the "fc" indicator. I also encoded the “three_g” and “four_g” into “network.”

network
0
4
4
3
4

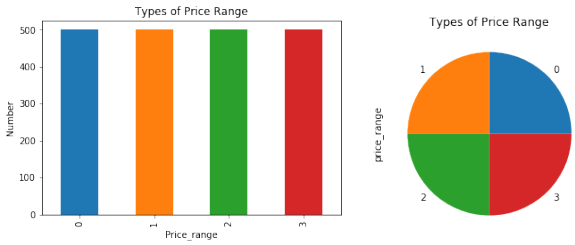
In the output, “3” indicates “three_g,” “4” indicates “four_g,” and “0” indicates neither “three_g” nor “four_g.”

In addition to this, I created the data visualization. At first, I visualized the univariate features:



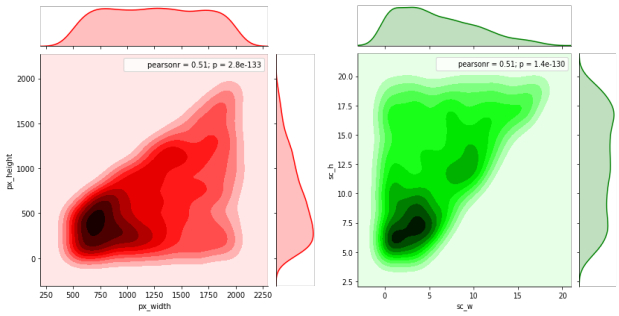


Then, I visualized the target variable by using the bar chart and the pie chart. It clearly showed that each price range has the same number, which means the data is balanced.



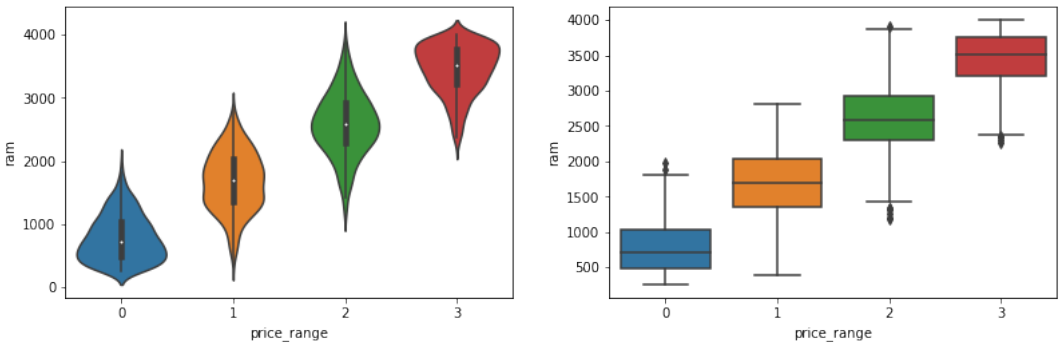
In addition, I used a heatmap to find the correlations between the processed data. From the heatmap, we can easily detect that there are moderate correlations between “px_width” and “px_height,” “sc_w” and “sc_h.” Therefore, I want to further explore the relationship between

these features. For further study of the binary relationship between these variables, I used the jointplot display. The darker the color, the more data there is.



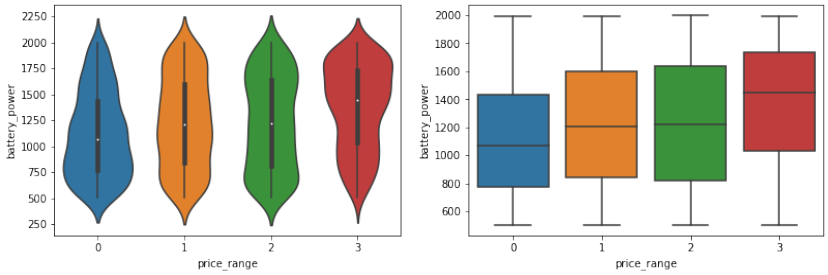
In addition to showing the relationship between each feature, the heatmap also clearly shows that the features “ram,” “battery_power,” are most related to our target variable, “price_range.” Therefore, I will explore the distribution between these variables and price range by using the violin plot and boxplot as follows:

1 How is price_range affected by ram?

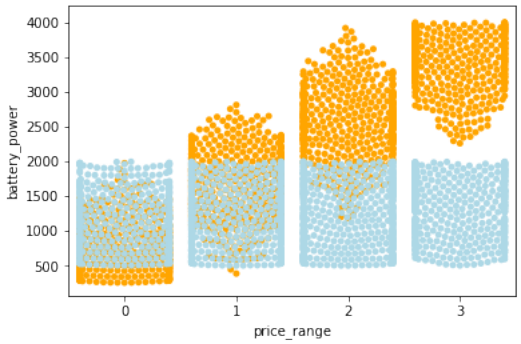


For the violin plot, the wider space represents more data in the interval. The image clearly shows that for the lower price (price range = 0), the ram of the mobile phone is also small; for the higher price (price range = 3), the ram of the mobile phone is also large. From the boxplot, we can see that the price ranges “0,” “2,” and “3” have outliers, but not too many. In addition, we can see that the ram for each price range are nearly normally distributed.

2 How is price_range affected by battery_power?



From the violin plot, we can see that the positive correlation is not particularly obvious between battery power and price range, especially for the price intervals "1" and "2". For the price "0," most samples have small battery capacity; in contrast, most samples with a large battery capacity are in the price range "3." From the boxplot, we can see that there is no outlier. I also used the swarmplot to show the overall relationship between the "ram" (orange), "battery_power" (blue), and "price_range" to add diversity.



4. Data Pre-Processing

Before building the models, I set X as my independent variable (regular variable), and Y as my dependent variable (target variable). Also, I partitioned the training set. I randomly chose 70% of data as the training dataset and 30% of data as the testing dataset. The training dataset can assist in making predictions and the test dataset can assist in checking the accuracy of the model. I also converted the data to the same scale by using the 'standardScaler()' function, which is from the Scikit-learn library.

Description of Models and Training Algorithm

Machine Learning Part

Because the problem is a multiclass problem, I decided to use Naïve Bayes, Support Vector Machine (SVM), and Decision Tree as my models for the machine learning part. I will introduce them here in detail.

Naïve Bayes

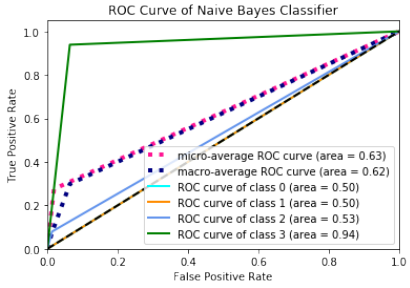
First, I used the Naïve Bayes classifier, which assumes that each feature is independent. I used the Gaussian Naïve Bayes algorithm because the features have continuous values. I used the GaussianNB module from sklearn library. After I built the Gaussian Naïve Bayes classifier, I used the 'fit()' function to train it. Then, I used the 'predict()' function to test the model. Next, I tested the accuracy of the Naïve Bayes model by comparing the prediction with the target value in the test dataset. I also evaluated the performance of the model by using the Precision, Recall, and F1-Scores as follows:

	Class 0	Class 1	Class 2	Class 3
Accuracy	93.38	71.85	71.52	92.02
Precision Score	93.38	74.05	72.48	88.76
Recall Score	93.38	71.85	71.52	92.02
F1-Score	93.38	72.93	72.00	90.36

By analyzing the output of Accuracy, Precision, Recall, and F1-Score, I found that the Naïve Bayes model had a great performance with the price range "1."

In order to make the ROC curve, I set the 'label_binarize()' function to binarize the output in the main script. Then, I used the OneVsRestClassifier to predict each class against the

others. In addition, I computed a ROC curve and ROC area for each class, micro-average ROC curve and ROC area, and macro-average ROC curve and ROC area. Finally, I got my ROC curve for the Naïve Bayes Classifier:

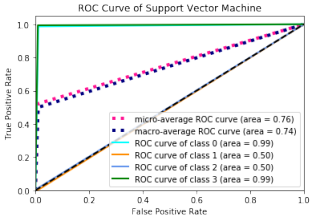


The area under the ROC curve is AUC. The bigger AUC, the better performance. The ROC curve clearly shows that the Naïve Bayes Classifier works well for the price range "1."

Support Vector Machine (SVM)

At first, I used the C-Support Vector Classification to build the model. However, the performance of the model was extremely bad, with the accuracy of the model below 50%. I believe that the reason for this was that there were too many features in the dataset. Thus, I changed the C-Support Vector Classification to LinearSVC. With this change, the accuracy greatly improved. I used the same method to build a SVM model, drawing the ROC curve with the Naïve Bayes model. The performance and ROC curve of the SVM model were as follows:

	Class 0	Class 1	Class 2	Class 3
Accuracy	100	72.59	72.85	99.39
Precision Score	94.97	73.13	78.57	97.01
Recall Score	100	72.59	72.85	99.39
F1-Score	97.42	72.86	75.76	98.18

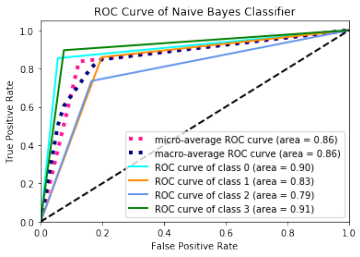


The results show that the SVM classifier has the better performance for price range “0” and price range “3.”

Decision Tree

I chose the ‘DecisionTreeClassifier()’ function as my classifier function. To train the Decision Tree model, I drew the ROC curve in the same way as with previous models. The performance and ROC curve of the Decision Tree model was as follows:

	Class 0	Class 1	Class 2	Class 3
Accuracy	82.12	71.85	60.26	79.14
Precision Score	87.32	61.39	62.76	83.23
Recall Score	82.12	71.85	60.26	79.14
F1-Score	84.64	66.21	61.49	81.13



The results show that the Decision Tree classifier had better performance for price range “0” and price range “3” compared to other price ranges.

Neural Network Part

Considering it is a classification problem, CNN (suitable for pattern features like recognizing an image), or RNN (suitable for time series data) are not very suitable for this case. I employed a multilayer perceptron (MLP) to solve this problem for the neural network part. Here are the initial parameters for my model:

BATCH_SIZE = 50

NUM_EPOCHS = 2000

LEARNING_RATE = 1

HIDDEN_NODE = 100

SEED_START = 10

LOSS_TARGET = 0.01

SEED_STEP = 5

After testing several times, I changed the learning rate from 1 to 0.005. The net architecture of the MLP that I built is shown below:

```
MLP(
  (inputLayer): Linear(in_features=20, out_features=100, bias=True)
  (hiddenLayer1): Linear(in_features=100, out_features=100, bias=True)
  (hiddenLayer2): Linear(in_features=100, out_features=100, bias=True)
  (hiddenLayer3): Linear(in_features=100, out_features=100, bias=True)
  (hiddenLayer4): Linear(in_features=100, out_features=100, bias=True)
  (hiddenLayer5): Linear(in_features=100, out_features=100, bias=True)
  (outputLayer): Linear(in_features=100, out_features=4, bias=True)
)
```

My model has both input and output layers, with an additional five hidden layers. The activation function that I chose was the RELU function. Moreover, I used Stochastic Gradient Descent as my model optimizer in order to prevent disappearing gradient. For the training, I defined the loss function to calculate the difference between the prediction and actual value first. Then, I employed the cross-entropy function as the loss metric. I prepared the data for calculating the gradient by changing them to variables. In addition, I used the 'optimizer.zero_grad()' function to set the gradient to 0. In this way, I was able to prevent the gradient increase when I did the backpropagation. Next, I did the forward, backward, and optimize sequence to update the weight. The updated weight was equal to weight minus learning rate multiplied by the gradient. Finally, I applied and tested the model that I built and get the Accuracy, Precision, Recall, and F1-Scores of the MLP as shown below:

	Class 0	Class 1	Class 2	Class 3
Accuracy	95	88	88	94
Precision Score	94.12	88015	89.26	94.48
Recall Score	95.36	88.15	88.08	94.48
F1-Score	94.74	88.15	88.67	94.48

5. Summary and Conclusion

Models Comparison

Finally, I compared all of the models that I built by calculating the average number of each indictor as follows:

	Accuracy	Precision	Recall Score	F1 Score
Decision Tree	73.3425	73.6750	73.3425	73.3675
MLP	91.5175	91.5025	91.5175	91.5100
Naive Bayes	82.1925	82.1675	82.1925	82.1675
SVM	86.2075	85.9200	86.2075	86.0150

The results clearly show that the performance of MLP is better than other models, and every indictor for MLP is higher than 90%. Therefore, I chose the MLP as my final model to predict the price range for the test file.

In conclusion, I found “ram” and “battery_power” to be the most important variable for predicting the price range, and MLP to be the best model for predicting price range among all of the models that I built.