2/7/2025

Mobile Phone Price Prediction

Project Report

Moe Htet Min
UNIFIED MENTOR

Table of Contents

Mobile Phone Pricing Prediction Project Report

Github: https://github.com/Moehtetmin28/Mobile-Phone-Price-Prediction

1. Introduction

This project developed a machine learning solution which determined mobile phone price categories through their detailed specifications. The goal of this project involved organizing mobile phones through four price brackets.

- 0 = Low Cost
- 1 = Medium Cost
- 2 = High Cost
- 3 = Very High Cost

Multiple features like **battery power, RAM capacity and screen resolution, camera specifications and connectivity possibilities** were used to train the model using a specific dataset.

---

2. Dataset Overview

The mobile phone records total **2000 entries** with **21 distinct features** available.

- **Numeric Features:** battery_power, ram, px_height, px_width, etc.
- **Binary Features:** blue (Bluetooth support), four_g, three_g, wifi, touch_screen, etc.
- **Target Variable:** price_range (Categorical: 0, 1, 2, 3)

The dataset contained a complete set of data points which enabled model training without requiring any data imputation processes.

Appendix Code:

```
[4]: # Load dataset
     df = pd.read_csv("dataset.csv")

     # Display basic info
     df.info()
     df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  2000 non-null   int64
 1   blue           2000 non-null   int64
 2   clock_speed    2000 non-null   float64
 3   dual_sim       2000 non-null   int64
 4   fc             2000 non-null   int64
 5   four_g         2000 non-null   int64
 6   int_memory     2000 non-null   int64
 7   m_dep          2000 non-null   float64
 8   mobile_wt      2000 non-null   int64
 9   n_cores        2000 non-null   int64
 10  pc             2000 non-null   int64
 11  px_height      2000 non-null   int64
 12  px_width       2000 non-null   int64
 13  ram            2000 non-null   int64
 14  sc_h           2000 non-null   int64
 15  sc_w           2000 non-null   int64
 16  talk_time      2000 non-null   int64
 17  three_g        2000 non-null   int64
 18  touch_screen   2000 non-null   int64
 19  wifi           2000 non-null   int64
 20  price_range    2000 non-null   int64
dtypes: float64(2), int64(19)
memory usage: 328.3 KB
```

[4]:

	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	three_g	to
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	2549	9	7	19	0	
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	2631	17	3	7	1	
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	2603	11	2	9	1	
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	2769	16	8	11	1	

---

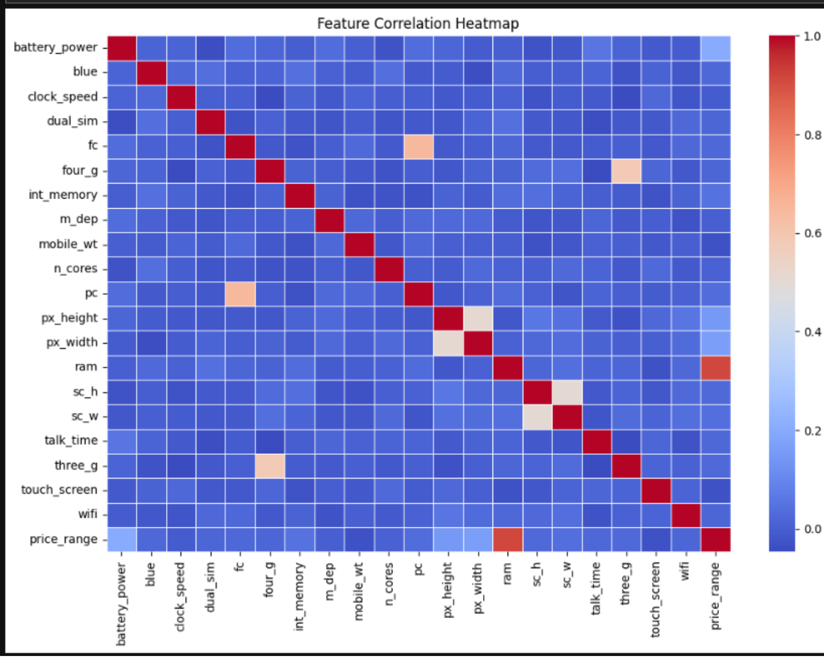
3. Data Exploration and Analysis

Analysts conducted a feature-price range correlation study.

- **Correlation Analysis:** The statistical analysis revealed that RAM displayed the greatest relationship with product pricing.
- **Feature Distributions:** The analysis using boxplots established RAM capacity together with battery life and display resolution as characteristic features that associate with higher product prices.
- **Heatmap:** Showed key relationships between features.

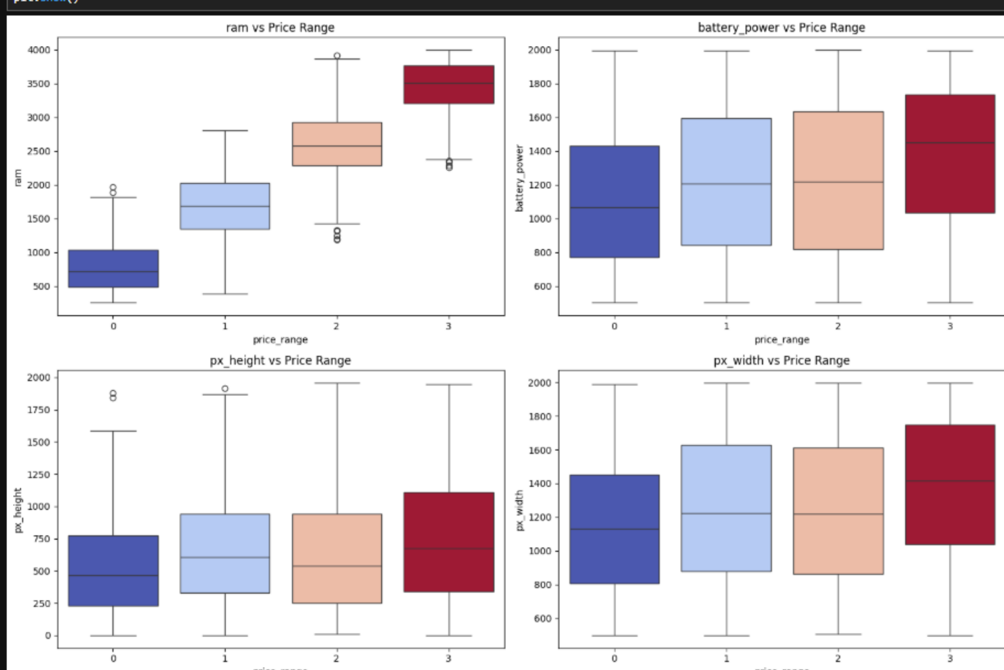
Appendix:

```
[5]: # Plot a correlation heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=False, cmap="coolwarm", linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.show()
```




```
[6]: # Visualizing feature distribution by price range
features_to_plot = ["ram", "battery_power", "px_height", "px_width"]

plt.figure(figsize=(15, 10))
for i, feature in enumerate(features_to_plot, 1):
    plt.subplot(2, 2, i)
    sns.boxplot(x="price_range", y=feature, data=df, hue="price_range", palette="coolwarm", legend=False)
    plt.title(f"{feature} vs Price Range")
plt.tight_layout()
plt.show()
```



4. Model Selection and Training

The project tested both **Random Forest Classifier** and **Logistic Regression with Standard Scaling** through a **pipeline-based approach**.

1. **Random Forest Classifier**

6. Training a Random Forest Classifier

```
# Training a Random Forest Classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
```

2. **Logistic Regression** with **Standard Scaling (Pipeline-based approach)**

9. Logistic Regression Model (Using Pipeline)

```
# Create a pipeline with StandardScaler and Logistic Regression
pipe = Pipeline([('scaler', StandardScaler()), ('logistic_regression', LogisticRegression())])

# Train the pipeline model
pipe.fit(X_train, y_train)
```

4.1. Random Forest Classifier

- **Train-Test Split:** 80% training, 20% testing.
- **Hyperparameters:** Used 100 decision trees.
- **Results:**
  - Accuracy: **88%**
  - **Linear models proved to be less effective than Random Forest** since patterns remained elusive to their structure.

```
# Evaluation
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print(report)
```

```
Accuracy: 0.88
              precision    recall  f1-score   support

           0       0.95      0.96      0.96       100
           1       0.82      0.84      0.83       100
           2       0.81      0.79      0.80       100
           3       0.93      0.93      0.93       100

    accuracy                           0.88       400
   macro avg       0.88      0.88      0.88       400
weighted avg       0.88      0.88      0.88       400
```

4.2. Logistic Regression Model (Pipeline)

- **StandardScaler applied** to normalize features.
- **Results:**
    - Accuracy: **96%**
    - Linear models performed behind Random Forest which suggests that linear algorithms fail to grasp all relevant patterns in the data.

```
# Evaluate Logistic Regression model
logistic_accuracy = pipe.score(X_test, y_test)
print(f"Logistic Regression Accuracy: {logistic_accuracy:.2f}")
```

```
Logistic Regression Accuracy: 0.96
```

---

5. Feature Importance Analysis

I trained the Logistic Regression model without RAM to analyze its effect:

- Model accuracy level dropped from its original (Original Accuracy) value to (Accuracy Without RAM).
- The results show that RAM plays a key role as a primary factor which impacts price forecasting.

10. Impact of Removing RAM

```python
#Impact of Removing RAM on Model Performance
# Remove 'ram' feature and re-train Logistic Regression
data = df.drop('ram', axis=1)

y = data['price_range'].copy()
X = data.drop('price_range', axis=1).copy()

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, random_state=20)

# Train the pipeline again without RAM
pipe.fit(X_train, y_train)

# Evaluate the model without RAM
logistic_accuracy_no_ram = pipe.score(X_test, y_test)
print(f"Logistic Regression Accuracy (without RAM): {logistic_accuracy_no_ram:.2f}")
```

```
Logistic Regression Accuracy (without RAM): 0.34
```

---

6. Conclusion and Future Improvements

- The **Random Forest model delivered superior outcomes compared to Logistic Regression** thus showing **non-linear patterns** in the provided data.
- Price prediction depends primarily on **three** main factors: **RAM capacity** together with **battery power** and **display resolution** performance.
- **Future Enhancements:**
    - The performance could be enhanced using **XGBoost** as well as **SVM** models.
    - The process of new interaction term development falls under the category of **feature engineering**.
    - **Hyperparameter tuning** for better optimization.

The study analyzed the link between **mobile phone features** and **pricing** as well as model performance in classification operations.