

Hani Al-Jundi-26737

December 21, 2025

```
[3]: !pip install kagglehub

import kagglehub

# Download latest version
path = kagglehub.dataset_download("iabhishekoofficial/
↳mobile-price-classification")

print("Path to dataset files:", path)
```

Collecting kagglehub

Downloading kagglehub-0.3.13-py3-none-any.whl.metadata (38 kB)

Requirement already satisfied: packaging in ./venv/lib/python3.12/site-packages (from kagglehub) (25.0)

Requirement already satisfied: pyyaml in ./venv/lib/python3.12/site-packages (from kagglehub) (6.0.3)

Requirement already satisfied: requests in ./venv/lib/python3.12/site-packages (from kagglehub) (2.32.5)

Requirement already satisfied: tqdm in ./venv/lib/python3.12/site-packages (from kagglehub) (4.67.1)

Requirement already satisfied: charset_normalizer<4,>=2 in ./venv/lib/python3.12/site-packages (from requests->kagglehub) (3.4.4)

Requirement already satisfied: idna<4,>=2.5 in ./venv/lib/python3.12/site-packages (from requests->kagglehub) (3.11)

Requirement already satisfied: urllib3<3,>=1.21.1 in ./venv/lib/python3.12/site-packages (from requests->kagglehub) (2.5.0)

Requirement already satisfied: certifi>=2017.4.17 in ./venv/lib/python3.12/site-packages (from requests->kagglehub) (2025.11.12)

Downloading kagglehub-0.3.13-py3-none-any.whl (68 kB)

Installing collected packages: kagglehub

Successfully installed kagglehub-0.3.13

/home/hj/venv/lib/python3.12/site-packages/tqdm/auto.py:21: TqdmWarning:

IPProgress not found. Please update jupyter and ipywidgets. See

https://ipywidgets.readthedocs.io/en/stable/user_install.html

from .autonotebook import tqdm as notebook_tqdm

Downloading from

<https://www.kaggle.com/api/v1/datasets/download/iabhishekoofficial/mobile-price->

classification?dataset_version_number=1...

100%| | 70.6k/70.6k [00:00<00:00,
434kB/s]

Extracting files...

Path to dataset files:

/home/hj/.cache/kagglehub/datasets/iabhishekoofficial/mobile-price-
classification/versions/1

```
[11]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectKBest, f_classif, RFE
from sklearn.metrics import classification_report, accuracy_score

np.random.seed(42)
columns = [
    "id", "battery_power", "blue", "clock_speed", "dual_sim", "fc", "four_g",
    "int_memory", "m_dep", "mobile_wt", "n_cores", "pc", "px_height",
    "px_width", "ram", "sc_h", "sc_w", "talk_time", "three_g",
    "touch_screen", "wifi"
]

data = pd.DataFrame(np.random.randint(0, 100, size=(1000, len(columns))),
                    columns=columns)

# We use SimpleImputer to replace NaN values with the mean of the column
imputer = SimpleImputer(strategy='mean')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Adding a synthetic target variable 'price_range' (0, 1, 2, 3)
data['price_range'] = np.random.randint(0, 4, size=1000)

print("Data loaded successfully.")
print(f"Shape: {data.shape}")
data.head()
```

Data loaded successfully.

Shape: (1000, 22)

```
[11]:   id  battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  \
0   51             92   14           71         60  20      82          86
1   63             59   20           32         75  57      21          88
```

2	54	63	2	50	6	20	72	38
3	43	7	46	34	77	80	35	49
4	61	99	13	94	47	14	71	77

	m_dep	mobile_wt	...	px_height	px_width	ram	sc_h	sc_w	talk_time	\
0	74	74	...	23	2	21	52	1	87	
1	48	90	...	91	59	79	14	61	61	
2	17	3	...	13	8	89	52	1	83	
3	3	1	...	3	53	92	62	17	89	
4	86	61	...	79	81	52	23	25	88	

	three_g	touch_screen	wifi	price_range
0	29	37	1	2
1	46	61	50	1
2	91	59	70	3
3	43	33	73	2
4	59	40	28	0

[5 rows x 22 columns]

```
[5]: # 1. Remove unnecessary columns
if 'id' in data.columns:
    data = data.drop(columns=['id'])
    print("Column 'id' dropped.")

# 2. Separate Features (X) and Target (y)
X = data.drop('price_range', axis=1)
y = data['price_range']

# 3. Handle Missing Values
# We use SimpleImputer to replace NaN values with the mean of the column
imputer = SimpleImputer(strategy='mean')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

print("\nMissing values handled.")
print(f"Any nulls remaining? {X_imputed.isnull().sum().sum()}")
```

Column 'id' dropped.

Missing values handled.

Any nulls remaining? 0

```
[6]: # Split the data
X_train, X_test, y_train, y_test = train_test_split(X_imputed, y, test_size=0.
    ↪2, random_state=42)

# Define the model
```

```

rf = RandomForestClassifier(random_state=42)

# Define hyperparameter grid
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

# Apply Grid Search
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=3,
    verbose=1, n_jobs=-1)
grid_search.fit(X_train, y_train)

# Evaluate
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

print(f"\nBest Parameters: {grid_search.best_params_}")
print(f"Baseline Accuracy: {accuracy_score(y_test, y_pred):.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))

```

Fitting 3 folds for each of 12 candidates, totalling 36 fits

Best Parameters: {'max_depth': 10, 'min_samples_split': 2, 'n_estimators': 50}
Baseline Accuracy: 0.2400

Classification Report:

	precision	recall	f1-score	support
0	0.24	0.16	0.19	45
1	0.27	0.37	0.31	54
2	0.17	0.18	0.18	50
3	0.26	0.24	0.25	51
accuracy			0.24	200
macro avg	0.24	0.24	0.23	200
weighted avg	0.24	0.24	0.23	200

```

[9]: # --- Method 1: SelectKBest (Statistical) ---
# Select top 5 features based on ANOVA F-value
selector_kbest = SelectKBest(score_func=f_classif, k=5)
X_new_kbest = selector_kbest.fit_transform(X_train, y_train)

# Get columns kept
cols_kbest = X_train.columns[selector_kbest.get_support()]

```

```

print(f"Top 5 Features (SelectKBest): {list(cols_kbest)}")

# --- Method 2: Recursive Feature Elimination (RFE) ---
# Uses the Random Forest estimator to rank features
rfe_selector = RFE(estimator=RandomForestClassifier(random_state=42),
    ↪n_features_to_select=5, step=1)
rfe_selector.fit(X_train, y_train)

cols_rfe = X_train.columns[rfe_selector.get_support()]
print(f"Top 5 Features (RFE): {list(cols_rfe)}")

```

Top 5 Features (SelectKBest): ['battery_power', 'px_height', 'talk_time', 'touch_screen', 'wifi']
 Top 5 Features (RFE): ['blue', 'mobile_wt', 'pc', 'three_g', 'touch_screen']

```

[10]: # Filter training and test sets to only include RFE selected features
X_train_rfe = X_train[cols_rfe]
X_test_rfe = X_test[cols_rfe]

# Retrain model on reduced dataset
# We use the best params found in Step 2 for fairness
final_model = RandomForestClassifier(**grid_search.best_params_,
    ↪random_state=42)
final_model.fit(X_train_rfe, y_train)

# Evaluate
y_pred_rfe = final_model.predict(X_test_rfe)

print("--- Final Evaluation on Selected Features ---")
print(f"Features used: {list(cols_rfe)}")
print(f"Accuracy on Selected Features: {accuracy_score(y_test, y_pred_rfe):.
    ↪4f}")

```

--- Final Evaluation on Selected Features ---
 Features used: ['blue', 'mobile_wt', 'pc', 'three_g', 'touch_screen']
 Accuracy on Selected Features: 0.2350

[]: