IBM Machine Learning Project

Mobile Price Range Prediction

Yu-Hsuan HSIEH
March 2023

Summary & Objective

This is a project for analyzing and predicting phone prices.

The dataset used in this project is provided by Kaggle.

We'll first observe the dataset, cleaning will be performed if needed, then fit the dataset into 3 different models, logistic regression, K-nearest neighbors and random forest.

Models will be evaluated and compared to find out the best model that fits the data.

The main objective of this analysis will be trying to train a model that best fits the mobile price dataset, the column that is predicted is the 'price range' column, every column left are for training.

Dataset Observation

Dataset Observation

The dataset consists of 20 columns and 2000 rows, below is the screenshot of df.info().

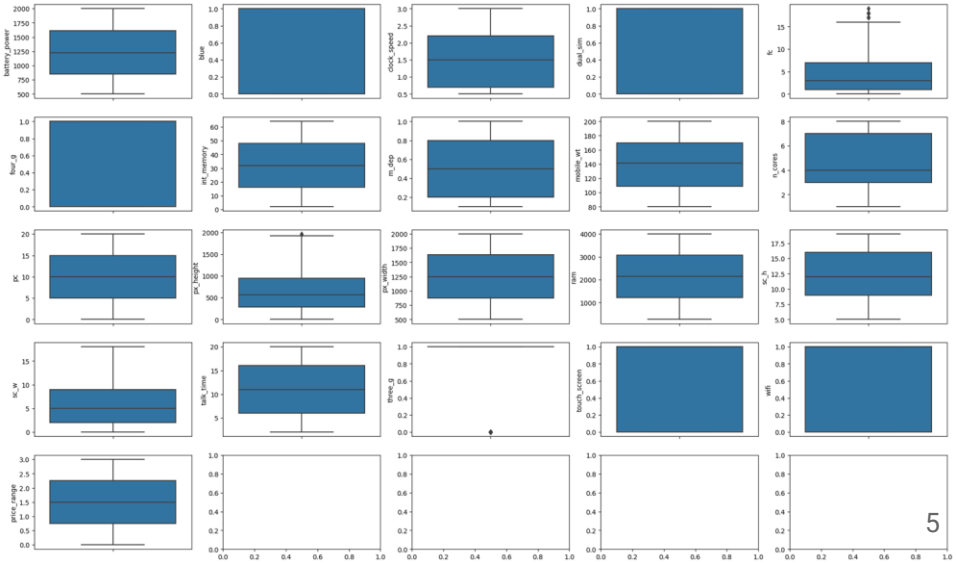
```
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  2000 non-null   int64
 1   blue           2000 non-null   int64
 2   clock_speed    2000 non-null   float64
 3   dual_sim       2000 non-null   int64
 4   fc             2000 non-null   int64
 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
```

Data Cleaning

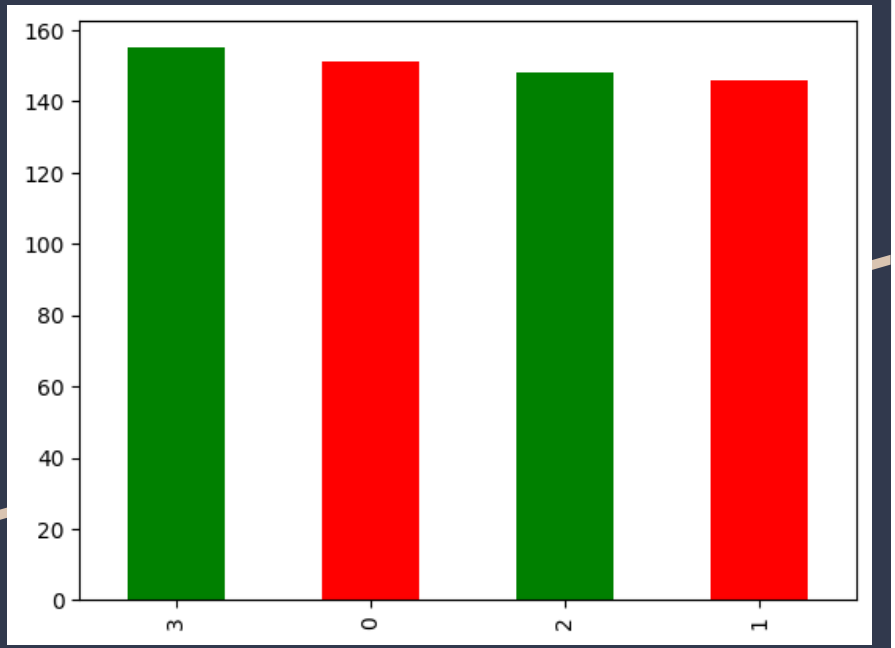
From observing the data types of each column, there is not data cleaning needed. Data are all in proper format.

We then try to plot each column into individual boxplot to seek for outliers.

From the boxplot below, there are no significant outliers, thus it’s not necessary to perform data cleaning.



Data Spliting



We split the data with the following code.

```
from sklearn.model_selection import
train_test_split

X_train, X_test, y_train, y_test =
train_test_split(X, y, train_size = 0.7,
test_size = 0.3, random_state = 42)
```

Then, we evaluate the results to see if the data is balanced.

```
y_test.value_counts()

y_test.value_counts().plot.bar(color=['gree
n', 'red'])
```

We can see that the y_test data is balanced and no improvement is required.

```
3    155
0    151
2    148
1    146
```

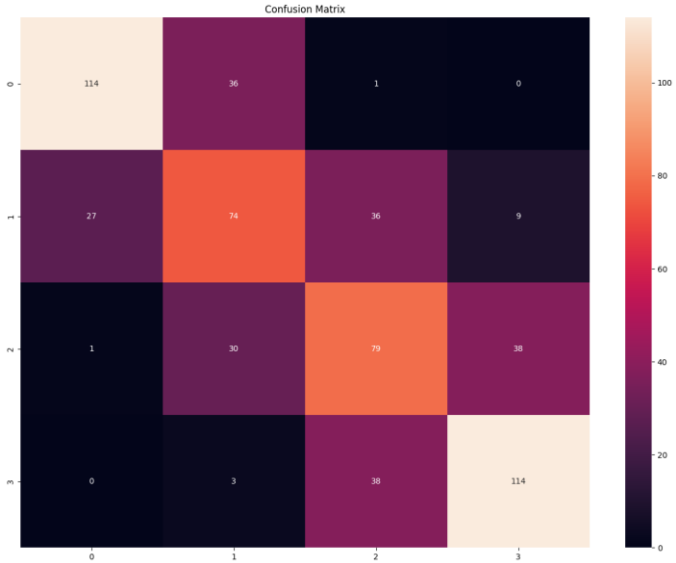
Model I: Logistic Regression

Logistic Regression

```
from sklearn.linear_model import
LogisticRegression

lr = LogisticRegression()

lr.fit(X_train, y_train)

preds_lr = lr.predict(X_test)
```

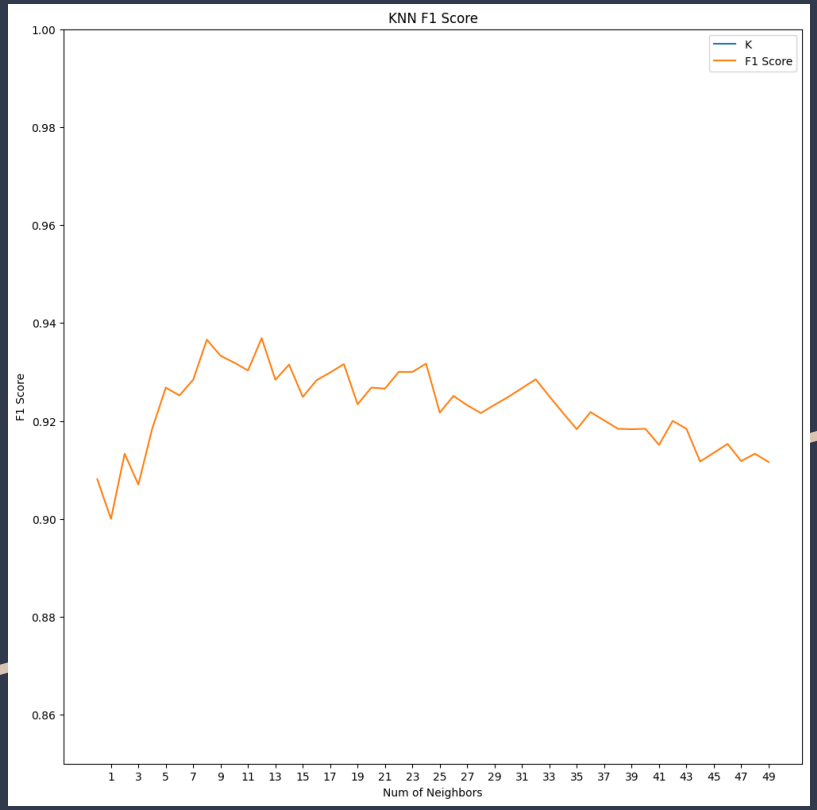
The results are shown below:

```
{'accuracy': 0.635,
 'recall': array([0.75496689, 0.50684932, 0.53378378, 0.73548387]),
 'precision': array([0.8028169 , 0.51748252, 0.51298701, 0.70807453]),
 'f1score': array([0.778157  , 0.51211073, 0.52317881, 0.72151899])}
```



Model II: K-nearest Neighbor

K-nearest Neighbors




```
from sklearn.neighbors import
KNeighborsClassifier
```

We want to find the optimal k value from range 1 - 50.

```
for k in range(1, max_k + 1):
    knn = KNeighborsClassifier(n_neighbors
= k)
    knn = knn.fit(X_train, y_train)
    preds_knn = knn.predict(X_test)
    f1 = f1_score(y_test, preds_knn,
average='weighted')
    f1_scores.append((k, round(f1, 4)))
```

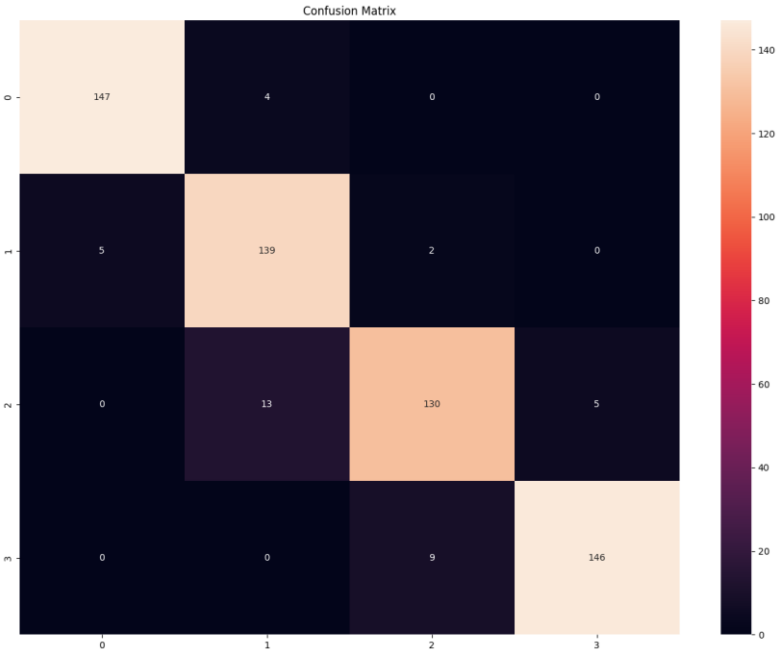
From the chart left, we can see that the optimal k value is 9 and 13, we'll apply 9 in later model training.

K-nearest Neighbors

```
knn_opt =
KNeighborsClassifier(n_neighbors = 9)
knn_opt = knn_opt.fit(X_train, y_train)
preds_opt_knn = knn_opt.predict(X_test)
evaluate_metrics(y_test, preds_opt_knn)
```

The results are shown below:

```
{'accuracy': 0.9366666666666666,
 'recall': array([0.97350993, 0.95205479, 0.87837838, 0.94193548]),
 'precision': array([0.96710526, 0.89102564, 0.92198582, 0.96688742]),
 'f1score': array([0.97029703, 0.9205298 , 0.89965398, 0.95424837])}
```



Model III: Random Forest

Random Forest

```
from sklearn.ensemble import
RandomForestClassifier
from sklearn.model_selection import
GridSearchCV
model = RandomForestClassifier()
param_grid = {'n_estimators': [2*n+1 for
n in range(20)],
              'max_depth' : [2*n+1 for n
in range(10) ],
              'max_features':["auto",
"sqrt", "log2"]}
search = GridSearchCV(estimator=model,
param_grid=param_grid,scoring='accuracy'
)
search.fit(X_train, y_train)
```

We try to find the best parameters for random forest.

Outputs of `print(search.best_params_)` are shown below.

```
{'max_depth': 19, 'max_features': 'auto', 'n_estimators': 33}
```

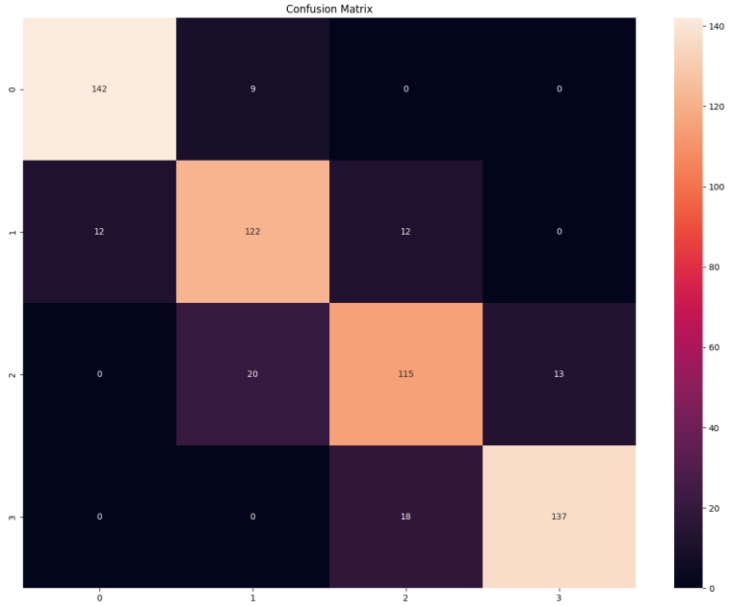
We will apply these parameters for later evaluation.

Random Forest

```
rf = RandomForestClassifier( max_depth =
19, max_features = 'auto', n_estimators
= 33)
rf.fit(X_train,y_train)
preds_rf = rf.predict(X_test)
```

The results are shown below:

```
{'accuracy': 0.86,
 'recall': array([0.94039735, 0.83561644, 0.77702703, 0.88387097]),
 'precision': array([0.92207792, 0.80794702, 0.79310345, 0.91333333]),
 'f1score': array([0.93114754, 0.82154882, 0.78498294, 0.89836066])}
```



Conclusion

Evaluation

From these three models, we can conclude the following table.

	Logistic Regression	K-nearest Neighbor	Random Forest
Accuracy	0.635	**0.94**	0.86
Avg Recall	0.64	**0.935**	0.86
Avg Precision	0.635	**0.9375**	0.8575
Avg F1 score	0.6325	**0.96**	0.8575

We can thus conclude that **KNN** performs the best in this dataset.

Possible Flaw

In this project, we concluded that KNN is the model that best fits the dataset, however, KNN has its own disadvantages and flaws.

- **Sensitivity to the choice of k**: The performance of k-NN can be sensitive to the choice of the number of nearest neighbors (k). If k is too small, the algorithm can be sensitive to noise, while if k is too large, it may lead to overfitting and poor generalization.
- **Computationally expensive**: The k-NN algorithm has to compute distances between the query point and all training data points for each prediction, which can be computationally expensive for large datasets.
- **Curse of dimensionality**: As the number of features or dimensions increases, the distance between the nearest neighbors becomes less informative and less discriminating, which can lead to degraded performance.

Thus, some suggestions can be further made.

Suggestions

To further improve the models' performance, we recommend exploring additional features combination or evaluate different models.

For example, we could consider using principal component analysis (PCA) to reduce the dimensionality of the dataset. We could also try different kernel functions in the SVM model.

Conclusion

Our study demonstrates the effectiveness of the **logistic regression**, **KNN,** and **random forest** models for classifying the mobile price dataset. We recommend using the KNN model for the highest accuracy and other evaluation parameters, but the other models also have their advantages and may be suitable for specific use cases.

Overall, our findings provide useful insights for anyone interested in using classification models for the mobile price dataset, and highlight the potential for further research in this area.

Thank you.