Minor DS Project

August 16, 2022

# Create a classification model to predict whether price range of mobile based on certain specification.

```
[2]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[2]: df = pd.read_csv("mobile_price_range_data.csv",sep=",")
     df
```

```
[2]:       battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  \
     0               842     0          2.2         0   1       0           7
     1              1021     1          0.5         1   0       1          53
     2               563     1          0.5         1   2       1          41
     3               615     1          2.5         0   0       0          10
     4              1821     1          1.2         0  13       1          44
     …             …   …          …       …   ..     …         …
     1995            794     1          0.5         1   0       1           2
     1996           1965     1          2.6         1   0       0          39
     1997           1911     0          0.9         1   1       1          36
     1998           1512     0          0.9         0   4       1          46
     1999            510     1          2.0         1   5       1          45

           m_dep  mobile_wt  n_cores  …  px_height  px_width   ram  sc_h  sc_w  \
     0       0.6        188        2  …         20       756  2549     9     7
     1       0.7        136        3  …        905      1988  2631    17     3
     2       0.9        145        5  …       1263      1716  2603    11     2
     3       0.8        131        6  …       1216      1786  2769    16     8
     4       0.6        141        2  …       1208      1212  1411     8     2
     …     …        …      …  …        …       …   …   …
     1995    0.8        106        6  …       1222      1890   668    13     4
     1996    0.2        187        4  …        915      1965  2032    11    10
     1997    0.7        108        8  …        868      1632  3057     9     1
     1998    0.1        145        5  …        336       670   869    18    10
     1999    0.9        168        6  …        483       754  3919    19     4

           talk_time  three_g  touch_screen  wifi  price_range
```

```
0             19       0             0     1            1
1              7       1             1     0            2
2              9       1             1     0            2
3             11       1             0     0            2
4             15       1             1     0            1
…         …      …           …   …           …
1995          19       1             1     0            0
1996          16       1             1     1            2
1997           5       1             1     0            3
1998          19       1             1     1            0
1999           2       1             1     1            3

[2000 rows x 21 columns]
```

[3]:

```
df.shape
```

[3]:

```
(2000, 21)
```

[4]:

```
df.isnull().sum()
```

[4]:

```
battery_power    0
blue             0
clock_speed      0
dual_sim         0
fc               0
four_g           0
int_memory       0
m_dep            0
mobile_wt        0
n_cores          0
pc               0
px_height        0
px_width         0
ram              0
sc_h             0
sc_w             0
talk_time        0
three_g          0
touch_screen     0
wifi             0
price_range      0
dtype: int64
```

[5]:

```
df.isnull().sum().sum()
```

[5]:

```
0
```

[6]:

```
x = df[['battery_power','px_height','px_width','ram']]
y = df['price_range']
print(type(x))
print(type(y))
```

```
<class 'pandas.core.frame.DataFrame'>
<class 'pandas.core.series.Series'>
```

[7]:

```
x.head()
```

[7]:

```
   battery_power  px_height  px_width   ram
0            842         20       756  2549
1           1021        905      1988  2631
2            563       1263      1716  2603
3            615       1216      1786  2769
4           1821       1208      1212  1411
```

[8]:

```
y.head()
```

[8]:

```
0    1
1    2
2    2
3    2
4    1
Name: price_range, dtype: int64
```

[9]:

```
print(x.shape)
```

```
(2000, 4)
```

[10]:

```
print(y.shape)
```

```
(2000,)
```

[11]:

```
from sklearn.model_selection import train_test_split
```

[12]:

```
# Split data into training and test data.

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.25)
```

[13]:

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(1500, 4)
(500, 4)
(1500,)
(500,)
```

Building Logistic Regression

[14]:

```
from sklearn.linear_model import LogisticRegression
```

[15]:

```
m1 = LogisticRegression()
m1.fit(x_train,y_train)
```

```
C:\Users\anant\Anaconda1\lib\site-
packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[15]:

```
LogisticRegression()
```

[16]:

```
ypred_m1 = m1.predict(x_test)
print(ypred_m1)
```

```
[3 2 3 0 3 1 1 1 3 3 1 0 3 1 0 2 0 3 0 1 0 2 3 1 2 1 1 2 1 1 1 3 2 2 0 2 0
 1 1 2 3 3 0 2 1 2 1 1 2 1 3 2 0 1 3 1 3 2 2 1 3 1 2 2 2 2 1 0 3 1 3 0 3 0
 0 3 1 0 0 2 3 2 0 0 1 2 1 3 3 0 2 2 3 2 1 3 3 3 3 2 1 0 2 1 0 3 1 0 1 1 2
 2 1 0 1 0 2 1 3 2 3 2 1 1 1 1 1 3 2 0 3 3 1 1 1 3 0 3 1 1 0 2 2 3 0 3 1 1
 1 3 3 0 2 2 2 3 0 3 1 3 1 2 0 0 1 3 0 3 1 3 1 2 3 3 2 2 1 0 0 3 3 2 0 3 2
 0 1 2 3 2 1 2 0 2 3 1 0 3 0 1 1 0 0 2 2 1 1 3 1 3 0 0 2 0 1 1 3 0 1 2 1 2
 3 3 1 1 3 0 1 2 0 0 3 1 0 1 2 2 0 0 0 0 2 2 1 0 3 2 1 1 2 2 2 2 3 1 1 1 2
 0 3 3 2 3 3 0 0 3 2 3 2 3 0 1 3 0 2 2 2 1 1 2 3 0 2 3 0 3 3 3 2 2 0 1 0 0
 1 1 3 2 3 3 0 1 3 2 2 1 0 0 1 2 1 1 0 2 1 2 3 0 1 1 2 0 2 0 0 0 0 2 1 1 1
 0 0 0 3 0 0 0 2 3 3 1 0 0 0 1 2 2 3 2 1 3 1 2 1 0 2 1 3 1 3 1 2 1 2 3 2 1
 0 2 2 3 1 1 3 1 3 2 0 2 2 1 0 2 3 2 1 1 1 3 3 1 1 0 2 1 1 1 3 0 3 1 0 0 2
 0 0 1 2 2 3 1 2 1 1 2 0 0 0 0 1 3 1 1 0 3 0 2 3 3 3 2 3 0 2 1 0 0 1 0 3 2
 1 2 2 2 1 3 2 0 2 1 2 1 2 0 2 1 0 1 2 3 0 2 3 2 2 0 0 2 0 2 2 2 1 2 2 3 2
 1 3 1 2 2 3 1 3 0 2 2 2 2 0 2 1 3 0 0]
```

[17]:

```
#Accuracy.
print('Training Score', m1.score(x_train , y_train))
print('Testing Score', m1.score(x_test,y_test))
```

```
Training Score 0.9606666666666667
Testing Score 0.962
```

[18]:

```
from sklearn.metrics import confusion_matrix, classification_report
```

```
[19]: cm = confusion_matrix(y_test, ypred_m1)
      print(cm)
      print(classification_report(y_test,ypred_m1))
```

```
[[115   5   0   0]
 [  0 132   7   0]
 [  0   1 124   2]
 [  0   0   4 110]]
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       120
           1       0.96      0.95      0.95       139
           2       0.92      0.98      0.95       127
           3       0.98      0.96      0.97       114

    accuracy                           0.96       500
   macro avg       0.96      0.96      0.96       500
weighted avg       0.96      0.96      0.96       500
```

```
[20]: test1 = pd.DataFrame()
```

```
[21]: test1['price_range'] = y_test
```

```
[22]: test1['logistic_pred'] = y_test
```

```
[23]: test1
```

```
[23]:       price_range  logistic_pred
      1431            3              3
      1548            2              2
      1655            3              3
      463             0              0
      1767            3              3
      …             …              …
      132             2              2
      1089            1              1
      1973            3              3
      901             0              0
      1859            0              0

      [500 rows x 2 columns]
```

KNN (K - nearest neighbors)

```
[24]: from sklearn.neighbors import KNeighborsClassifier
```

[25]:
```
m2 = KNeighborsClassifier (n_neighbors = 21)
m2.fit(x_train, y_train)
```

[25]:
```
KNeighborsClassifier(n_neighbors=21)
```

[26]:
```
ypredkn_m2 = m2.predict(x_test)
print('Training Score', m2.score(x_train, y_train))
print('Testing Score', m2.score(x_test ,y_test))
```

```
Training Score 0.9426666666666667
Testing Score 0.924
```

[27]:
```
from sklearn.metrics import confusion_matrix , classification_report
cm = confusion_matrix(y_test, ypredkn_m2)
print(cm)
print(classification_report(y_test, ypredkn_m2))
```

```
[[115   5   0   0]
 [  7 122  10   0]
 [  0   3 119   5]
 [  0   0   8 106]]
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       120
           1       0.94      0.88      0.91       139
           2       0.87      0.94      0.90       127
           3       0.95      0.93      0.94       114

    accuracy                           0.92       500
   macro avg       0.93      0.93      0.93       500
weighted avg       0.93      0.92      0.92       500
```

[29]:
```
test1['kn_pred'] = ypredkn_m2
test1
```

[29]:
```
      price_range  logistic_pred  kn_pred
1431            3              3        3
1548            2              2        2
1655            3              3        3
463             0              0        0
1767            3              3        3
…            …              …        …
132             2              2        2
1089            1              1        1
1973            3              3        3
901             0              0        0
1859            0              0        0
```

```
[500 rows x 3 columns]
```

SVM (Support Vector Machine)

[30]:
```
from sklearn.svm import SVC
```

[31]:
```
s1 = SVC(kernel='linear',C=1)
s1.fit(x_train,y_train)
```

[31]:
```
SVC(C=1, kernel='linear')
```

[32]:
```
ypredsvm_s1 = s1.predict(x_test)
print('Training Score',s1.score(x_train, y_train))
print('Testing Score', s1.score(x_test, y_test))
```

```
Training Score 0.96
Testing Score 0.958
```

[33]:
```
cm = confusion_matrix(y_test, ypredsvm_s1)
print(cm)
print(classification_report(y_test, ypredsvm_s1))
```

```
[[115   5   0   0]
 [  0 131   8   0]
 [  0   1 124   2]
 [  0   0   5 109]]
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       120
           1       0.96      0.94      0.95       139
           2       0.91      0.98      0.94       127
           3       0.98      0.96      0.97       114

    accuracy                           0.96       500
   macro avg       0.96      0.96      0.96       500
weighted avg       0.96      0.96      0.96       500
```

[34]:
```
test1['svm_pred'] = ypredsvm_s1
test1
```

[34]:
```
      price_range  logistic_pred  kn_pred  svm_pred
1431            3              3        3         3
1548            2              2        2         2
1655            3              3        3         3
463             0              0        0         0
1767            3              3        3         3
…            …             …      …        …
```

```
132             2              2        2         2
1089            1              1        1         1
1973            3              3        3         3
901             0              0        0         0
1859            0              0        0         0

[500 rows x 4 columns]
```

RBF Kernel

[35]:
```python
s2 = SVC(kernel = 'rbf', C=10, gamma=0.00001)
s2.fit(x_train, y_train)
```

[35]:
```
SVC(C=10, gamma=1e-05)
```

[36]:
```python
ypredrbf_s2 = s2.predict(x_test)
print('Training Score', s2.score(x_train, y_train))
print('Testing Score' , s2.score(x_test, y_test))
```

```
Training Score 0.9973333333333333
Testing Score 0.92
```

[37]:
```python
cm = confusion_matrix(y_test, ypredrbf_s2)
print(cm)
print(classification_report(y_test, ypredrbf_s2))
```

```
[[115   5   0   0]
 [  9 120  10   0]
 [  0   4 119   4]
 [  0   0   8 106]]
              precision    recall  f1-score   support

           0       0.93      0.96      0.94       120
           1       0.93      0.86      0.90       139
           2       0.87      0.94      0.90       127
           3       0.96      0.93      0.95       114

    accuracy                           0.92       500
   macro avg       0.92      0.92      0.92       500
weighted avg       0.92      0.92      0.92       500
```

[38]:
```python
test1['rbf_pred'] = ypredrbf_s2
test1
```

[38]:
```
      price_range  logistic_pred  kn_pred  svm_pred  rbf_pred
1431            3              3        3         3         3
1548            2              2        2         2         2
1655            3              3        3         3         3
```

```
463             0              0         0         0         0
1767            3              3         3         3         3
…           …            …      …       …       …
132             2              2         2         2         2
1089            1              1         1         1         1
1973            3              3         3         3         3
901             0              0         0         0         0
1859            0              0         0         0         0

[500 rows x 5 columns]
```

Conclusion: Model with best Accuracy.

1) Logistic Regression score(in percentage) : 95.8%
2) KNN score(in percentage) : 95.8%
3) SVM score(in percentage) : 96.26%
4) SVM is the most accurate model among the classification model I have used in this project with the accuracy score 96.26%.

```
[ ]:
```