

Загрузка и первичный анализ данных

Для выполнения задания был выдан датасет с данными о характеристиках моделей мобильных телефонов и их рейтингом цен.

<https://www.kaggle.com/datasets/abhishekofficial/mobile-price-classification?select=train.csv> (<https://www.kaggle.com/datasets/abhishekofficial/mobile-price-classification?select=train.csv>)

```
from operator import itemgetter

from sklearn.model_selection import train_test_split, KFold, GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

import numpy as np
import pandas as pd
import seaborn as sns

import math

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

%matplotlib inline
sns.set(style="ticks")

data = pd.read_csv('Mobile_price_classification/data.csv', sep=",")
data.dtypes
```

```
battery_power      int64
blue                int64
clock_speed         float64
dual_sim            int64
fc                  int64
four_g              int64
int_memory           int64
m_dep               float64
mobile_wt           int64
n_cores             int64
pc                  int64
px_height           int64
px_width            int64
ram                 int64
sc_h                int64
sc_w                int64
talk_time           int64
three_g             int64
touch_screen        int64
wifi                int64
price_range         int64
dtype: object
```

```
data.shape
```

```
(2000, 21)
```

```
np.where(pd.isnull(data))
```

```
(array([], dtype=int64), array([], dtype=int64))
```

В массиве нет пропусков.

```
data
```

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ra
--	---------------	------	-------------	----------	----	--------	------------	-------	-----------	---------	-----	-----------	----------	----

	battery_power	blue	clock_speed	dual_sim	fc	four_g	int_memory	m_dep	mobile_wt	n_cores	...	px_height	px_width	ra
0	842	0	2.2	0	1	0	7	0.6	188	2	...	20	756	25
1	1021	1	0.5	1	0	1	53	0.7	136	3	...	905	1988	26
2	563	1	0.5	1	2	1	41	0.9	145	5	...	1263	1716	26
3	615	1	2.5	0	0	0	10	0.8	131	6	...	1216	1786	27
4	1821	1	1.2	0	13	1	44	0.6	141	2	...	1208	1212	14
...
1995	794	1	0.5	1	0	1	2	0.8	106	6	...	1222	1890	66
1996	1965	1	2.6	1	0	0	39	0.2	187	4	...	915	1965	20
1997	1911	0	0.9	1	1	1	36	0.7	108	8	...	868	1632	30
1998	1512	0	0.9	0	4	1	46	0.1	145	5	...	336	670	86
1999	510	1	2.0	1	5	1	45	0.9	168	6	...	483	754	39

2000 rows × 21 columns

Будем решать задачу классификации. Для решения задачи планируется использование метрики ассигасу. Проверим, можно ли применить ее.

```
np.unique(data['price_range'], return_counts=True)
```

```
(array([0, 1, 2, 3], dtype=int64), array([500, 500, 500, 500], dtype=int64))
```

Классы сбалансированы. Можно использовать метрику ассигасу.

```
y=np.array(data['price_range'])
X=np.array(data.drop(['price_range'], axis=1))
X, y
```

```
(array([[8.420e+02, 0.000e+00, 2.200e+00, ..., 0.000e+00, 0.000e+00,
        1.000e+00],
       [1.021e+03, 1.000e+00, 5.000e-01, ..., 1.000e+00, 1.000e+00,
        0.000e+00],
       [5.630e+02, 1.000e+00, 5.000e-01, ..., 1.000e+00, 1.000e+00,
        0.000e+00],
       ...,
       [1.911e+03, 0.000e+00, 9.000e-01, ..., 1.000e+00, 1.000e+00,
        0.000e+00],
       [1.512e+03, 0.000e+00, 9.000e-01, ..., 1.000e+00, 1.000e+00,
        1.000e+00],
       [5.100e+02, 1.000e+00, 2.000e+00, ..., 1.000e+00, 1.000e+00,
        1.000e+00]]),
 array([1, 2, 2, ..., 3, 0, 3], dtype=int64))
```

Разделение выборки на обучающую и тестовую

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, shuffle=True)
```

Выборка разделена на обучающую X_train, y_train и тестовую X_test, y_test

Начинаем обучение

```
KNN_Clf= KNeighborsClassifier(n_neighbors=2)
KNN_Clf
```

```
KNeighborsClassifier(n_neighbors=2)
```

```
KNN_Clf.fit(X_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=2)
```

```
y_pred = KNN_Clf.predict(X_test)
```

```
accur = accuracy_score(y_test, y_pred)  
accur
```

```
0.8966666666666666
```

Мы получили точность совпадения предсказанных результатов с истинными 90%

Подбор гиперпараметра K с использованием GridSearchCV

```
parameters = { 'n_neighbors': range (1, 30)}  
clf = KNeighborsClassifier()
```

```
grid3 = GridSearchCV(clf, parameters, cv=3)  
grid3.fit(X, y)
```

```
GridSearchCV(cv=3, estimator=KNeighborsClassifier(),  
             param_grid={'n_neighbors': range(1, 30)})
```

```
grid3.best_params_
```

```
{'n_neighbors': 7}
```

```
grid5 = GridSearchCV(clf, parameters, cv=5)  
grid5.fit(X, y)
```

```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),  
             param_grid={'n_neighbors': range(1, 30)})
```

```
grid5.best_params_
```

```
{'n_neighbors': 11}
```

```
grid7 = GridSearchCV(clf, parameters, cv=7)  
grid7.fit(X, y)
```

```
GridSearchCV(cv=7, estimator=KNeighborsClassifier(),  
             param_grid={'n_neighbors': range(1, 30)})
```

```
grid7.best_params_
```

```
{'n_neighbors': 12}
```

```
grid10 = GridSearchCV(clf, parameters, cv=10)  
grid10.fit(X, y)
```

```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),  
             param_grid={'n_neighbors': range(1, 30)})
```

```
grid10.best_params_
```

```
{'n_neighbors': 12}
```

Оптимальное значение гиперпараметра K = 12

Создадим класс с оптимальным значением гиперпараметра

```
KNN_Clф_Opt= KNeighborsClassifier(n_neighbors=12)  
KNN_Clф_Opt
```

```
KNeighborsClassifier(n_neighbors=12)
```

```
KNN_Clф_Opt.fit(X_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=12)
```

```
y_pred_opt = KNN_Clф.predict(X_test)
```

```
accur_opt = accuracy_score(y_test, y_pred_opt)  
accur_opt
```

```
0.92
```

Качество метрики оптимальной модели 92% выше, чем качество метрики исходной.