Рекомендация тарифов

В вашем распоряжении данные о поведении клиентов, которые уже перешли на эти тарифы. Нужно построить модель для задачи классификации, которая выберет подходящий тариф. Данные предобработаны.

Постройте модель с максимально большим значением *accuracy*. Чтобы сдать проект успешно, нужно довести долю правильных ответов по крайней мере до 0.75. Проверьте *accuracy* на тестовой выборке самостоятельно.

План

1. Получение данных и изучение общей информации о них

B [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as st
import os
import urllib
import random

from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, precision_score, recall_score

pd.options.display.float_format = '{:.3f}'.format
```

B [3]:
```
name, url = 'datasets/users_behavior.csv', 'https://code.s3.yandex.net/datasets/u
if not os.path.exists(name.split('/')[0]):
    os.makedirs(name.split('/')[0])
if not os.path.exists(name):
    _ = urllib.request.urlretrieve(url, name)
```

B [4]:
```
data = pd.read_csv('datasets/users_behavior.csv')
data.head()
```

Out[4]:

	calls	minutes	messages	mb_used	is_ultra
0	40.000	311.900	83.000	19915.420	0
1	85.000	516.750	56.000	22696.960	0
2	77.000	467.660	86.000	21060.450	0
3	106.000	745.530	81.000	8437.390	1
4	66.000	418.740	1.000	14502.750	0

B [5]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3214 entries, 0 to 3213
Data columns (total 5 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   calls     3214 non-null   float64
 1   minutes   3214 non-null   float64
 2   messages  3214 non-null   float64
 3   mb_used   3214 non-null   float64
 4   is_ultra  3214 non-null   int64
dtypes: float64(4), int64(1)
memory usage: 125.7 KB
```

B [6]:
```
data.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
calls	3214.000	63.039	33.236	0.000	40.000	62.000	82.000	244.000
minutes	3214.000	438.209	234.570	0.000	274.575	430.600	571.927	1632.060
messages	3214.000	38.281	36.148	0.000	9.000	30.000	57.000	224.000
mb_used	3214.000	17207.674	7570.968	0.000	12491.903	16943.235	21424.700	49745.730
is_ultra	3214.000	0.306	0.461	0.000	0.000	0.000	1.000	1.000

B [7]:
```
data.calls = data.calls.astype(np.uint16)
data.minutes = data.minutes.astype(np.uint16)
data.messages = data.messages.astype(np.uint16)
```

B [8]:
```
print(f'{data.duplicated().sum()}')
```

```
0
```

Вывод:

Данные не содержат пропусков и дубликатов. Были изменены типы данных с целью рационального хранения информации

Каждый объект в наборе данных — это информация о поведении одного пользователя за месяц. Известно:

- *calls* — количество звонков,
- *minutes* — суммарная длительность звонков в минутах,
- *messages* — количество sms-сообщений,
- *mb_used* — израсходованный интернет-трафик в Мб,
- *is_ultra* — каким тарифом пользовался в течение месяца («Ультра» — 1, «Смарт» — 0).

2. Разделение исходных данных на выборки

B [9]:
```
data_features = data.drop(['is_ultra'], axis=1)
data_target = data.is_ultra
```

B [10]:
```
features_train, features_valid, target_train, target_valid =\
    train_test_split(data_features, data_target, test_size=.4, random_state=1)
features_valid, features_test, target_valid, target_test =\
    train_test_split(features_valid, target_valid, test_size=.5, random_state=1)
```

B [11]:
```
for part in ((features_train, features_valid, features_test), (target_train, targ
    print(' - '.join([str(round(cur.shape[0] / data.shape[0], 1)) for cur in part
```

```
0.6 - 0.2 - 0.2
0.6 - 0.2 - 0.2
```

Вывод:

Разделили данные на выборки для обучения, валидации и тестирования модели соответственно.
Проверили, что полученные выборки фич (признаков) и таргета (целевого признака) относятся как 3:1:1.

3. Исследование моделей классификации

B [12]:
```
models = {
    'tree': ['DecisionTreeClassifier', 0, 0, ''],
    'forest': ['RandomForestClassifier', 0, 0, ''],
    'regression': ['LogisticRegression', 0, 0, '']
}
```

B [13]:
```
for crit in "gini", "entropy":
    for featur in "auto", "sqrt", "log2":
        for depth in range(1, 10):
            model = DecisionTreeClassifier(random_state=1, criterion=crit, max_de
            model.fit(features_train, target_train)
            pred_train = model.predict(features_train)
            acc_train = accuracy_score(target_train, pred_train)
            pred = model.predict(features_valid)
            acc = accuracy_score(target_valid, pred)
            if models['tree'][1] < acc:
                models['tree'][1] = acc
                models['tree'][2] = acc_train
                models['tree'][3] = f'criterion {crit} | M_depth {depth} | M_feat
```

B [ ]:
```
for depth in range(1, 10):
    for estim in range(10, 100):
        for bootstr in True, False:
            model = RandomForestClassifier(random_state=1, max_depth=depth, n_est
            model.fit(features_train, target_train)
            pred_train = model.predict(features_train)
            acc_train = accuracy_score(target_train, pred_train)
            pred = model.predict(features_valid)
            acc = accuracy_score(target_valid, pred)
            if models['forest'][1] < acc:
                models['forest'][1] = acc
                models['forest'][2] = acc_train
                models['forest'][3] = f'M_depth {depth} | N_estims {estim} | Boot
```

B [ ]:
```
model = LogisticRegression(random_state=1)
model.fit(features_train, target_train)
pred_train = model.predict(features_train)
acc_train = accuracy_score(target_train, pred_train)
pred = model.predict(features_valid)
acc = accuracy_score(target_valid, pred)
models['regression'][1] = acc
models['regression'][2] = acc_train
models['regression'][3] = '---'
```

B [ ]:
```
pd.DataFrame(models, index=['model', 'acc', 'acc_train', 'hyparam']).T
```

Вывод:

Произвели исследование (обучение на тренировочной выборке и предсказание на валидационной) моделей классификации,
а именно `DecisionTreeClassifier`, `RandomForestClassifier`, `LogisticRegression`.
Налучший результат метрики `accuracy` показала модель `RandomForestClassifier` со следующими значениями гиперпараметров:

- `max_depth` = 8
- `n_estimators` = 25
- `bootstrap` = `True`

4. Тестирование (точность обученой модели)

В [ ]:
```
model = RandomForestClassifier(random_state=1, max_depth=8, n_estimators=25, boot
model.fit(features_train, target_train)
pred = model.predict(features_test)
res_metrics = (
                accuracy_score(target_test, pred),
                precision_score(target_test, pred),
                recall_score(target_test, pred)
)
```

В [ ]:
```
result = pd.DataFrame({
        'result': res_metrics,
}, index=['Accuracy', 'Precision', 'Recall'])
result.T
```

Вывод:

Произвели исследование моделей классификации, а именно `DecisionTreeClassifier`, `RandomForestClassifier`, `LogisticRegression`
Изменяя гиперпараметры моделей классификаторов, нашли налучший результат метрики `accuracy` модели `RandomForestClassifier`:

- `max_depth = 8`
- `n_estimators = 25`
- `bootstrap = True`

5. Проверка модели на адекватность

В [ ]:
```
pred_random = pd.Series([random.randint(0, 1) for i in range(len(features_test))]

random_metrics = (accuracy_score(target_test, pred_random),
                  precision_score(target_test, pred_random),
                      recall_score(target_test, pred_random)
)
```

В [ ]:
```
ok, n_ok = '\u2705', '\u274C'
result['random'] = random_metrics
result['sanity_check'] = [ok if res > ran else n_ok for res, ran in zip(res_metri
result.T
```

Вывод:

В ходе проверки `sanity check` (сравнение метрик предсказаний обученной модели и случайной величины) выяснили:

Метрика	Описание	Результат
`accuracy/ аккуратность`	Доля верно предсказанных величин целевого признака к размеру всей тестовой выборки	+
`precision/ точность`	Доля помеченных моделью величин действительно имеют такое значение в выборке	+
`recall/полнота`	Доля помеченных моделью величин из всех присутствующих в выборке	-