Рекомендации по предложению тарифов

К нам обратилась телекоммуникационная компания с просьбой предложить рекомендации для пользователей по переходу на актуальные тарифы. В компании есть два новых тарифа - smart и ultra. Ряд клиентов уже перешли на них. Компания предоставила сведения о поведении клиентов, которым понравились эти тарифы.

Большая часть клиентов еще пользуются устаревшими тарифами. Компании будет выгодно, если они перейдут на новые. Перед нами поставлена задача обучить модель, которая будет по поведению пользователя определять, какой из новых тарифов ему лучше подойдет и тогда компания сможет сделать более эффективную и таргетную рекламу.

1 Описание данных

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 Загружаем необходимые библиотеки

Ввод [1]:
```
import pandas as pd
```

Ввод [2]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.dummy import DummyClassifier
from sklearn.preprocessing import StandardScaler
```

3 Изучаем данные

Ввод [3]:
```
df = pd.read_csv('users_behavior.csv')

df.shape
```

Out[3]:
```
(3214, 5)
```

У нас данные о 3214 абонентах, которые пользовались интересующими нас тарифами

Посмотрим на состав данных, обращая внимание на их полноту

Ввод [4]:
```
df.head()
```

Out[4]:

	calls	minutes	messages	mb_used	is_ultra
0	40.0	311.90	83.0	19915.42	0
1	85.0	516.75	56.0	22696.96	0
2	77.0	467.66	86.0	21060.45	0
3	106.0	745.53	81.0	8437.39	1
4	66.0	418.74	1.0	14502.75	0

Ввод [5]:
```
df.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
```

Ввод [6]:
```
df.describe()
```

Out[6]:

	calls	minutes	messages	mb_used	is_ultra
count	3214.000000	3214.000000	3214.000000	3214.000000	3214.000000
mean	63.038892	438.208787	38.281269	17207.673836	0.306472
std	33.236368	234.569872	36.148326	7570.968246	0.461100
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	40.000000	274.575000	9.000000	12491.902500	0.000000
50%	62.000000	430.600000	30.000000	16943.235000	0.000000
75%	82.000000	571.927500	57.000000	21424.700000	1.000000
max	244.000000	1632.060000	224.000000	49745.730000	1.000000

Пропущенных данных нет.

Так как у нас is_ultra mean=0.3, то для класса ultra у нас 30% данных, а для класса smart 70%.

При проведении обучения модели нам необходимо учесть несбалансированность классов.

Кроме того, мы видим, что абсолютные значения в переменных сильно отличаются, поэтому их необходимо стандартизировать.

4 Разделение данных на выборки

Выделим целевой признак

Ввод [7]:
```
features = df.drop(['is_ultra'], axis=1)
target = df['is_ultra']
```

Делим данные на исходные данные для моделирования и тестовую выборку, указывая параметр stratify, так как данные несбалансированны по классам

Ввод [8]:
```
features_base, features_test, target_base, target_test = train_test_split(
    features, target, test_size=0.2, random_state=12345, stratify=target)
```

А теперь базовую разделим на обучающую и валидационную часть, на которой будем проверять результаты обучения в процессе

Ввод [9]:
```
features_train, features_valid, target_train, target_valid = train_test_split(
    features_base, target_base, test_size=0.25, random_state=12345, stratify=target_base)
```

Ввод [10]:
```
target_train.describe()
```

Out[10]:
```
count    1928.000000
mean        0.306535
std         0.461174
min         0.000000
25%         0.000000
50%         0.000000
75%         1.000000
max         1.000000
Name: is_ultra, dtype: float64
```

В результате на валидационную и на тестовую части приходится по 20% исходных данных, а на 60% мы будем обучать модель. Так как мы разбивали выборки с параметром stratify, доля пользователей с тарифом ultra сохранилась стабильной - ~30%

5 Стандартизация

Ввод [11]:
```
scaler = StandardScaler()
scaler.fit(features_train)
features_train_st = scaler.transform(features_train)
features_valid_st = scaler.transform(features_valid)
features_test_st = scaler.transform(features_test)
```

6 Обучение моделей

6.1 Обучим модель Случайный лес с подбором глубины и количества деревьев

Ввод [12]:
```
best_model = None
best_accuracy = 0
best_est = 0
best_depth = 1
```

В качестве основной универсальной метрики мы будем использовать accuracy, так как это метрика, которая характеризует качество модели, агрегированное по всем классам. В нашем случае это полезно, так как классы для нас имеют одинаковое значение.

Ввод [13]:
```
for est in range(1, 11):
    for depth in range(1,30):
        model = RandomForestClassifier(random_state=12345,
                                       n_estimators=est,
                                       max_depth=depth,
                                       class_weight='balanced')

        model.fit(features_train_st, target_train)

        predictions_valid = model.predict(features_valid_st)

        accuracy = accuracy_score(target_valid, predictions_valid)

        if accuracy > best_accuracy:
            best_model = model
            best_accuracy = accuracy
            best_est = est
            best_depth = depth

predictions_valid = best_model.predict(features_valid_st)

# Для класса 1 - ultra
f1_u = f1_score(target_valid, predictions_valid)
precision_u = precision_score(target_valid, predictions_valid)
recall_u =  recall_score(target_valid, predictions_valid)

# для класса 0 - smart
f1_s = f1_score(target_valid, predictions_valid, pos_label=0)
precision_s = precision_score(target_valid, predictions_valid, pos_label=0)
recall_s =  recall_score(target_valid, predictions_valid, pos_label=0)
```

Ввод [14]:
```
print("Accuracy наилучшей модели на валидационной выборке:", best_accuracy)
print('Лучший max_depth:', best_depth)
print('Лучший n_estimators:', best_est)
print()
print('Метрики для класса 1 ultra')
print("Precision наилучшей модели на валидационной выборке:", precision_u)
print("Recall наилучшей модели на валидационной выборке:", recall_u)
print("F1 наилучшей модели на валидационной выборке:", f1_u)
print()
print('Метрики для класса 0 smart')
print("Precision наилучшей модели на валидационной выборке:", precision_s)
print("Recall наилучшей модели на валидационной выборке:", recall_s)
print("F1 наилучшей модели на валидационной выборке:", f1_s)
```

```
Accuracy наилучшей модели на валидационной выборке: 0.8180404354587869
Лучший max_depth: 7
Лучший n_estimators: 7

Метрики для класса 1 ultra
Precision наилучшей модели на валидационной выборке: 0.7380952380952381
Recall наилучшей модели на валидационной выборке: 0.6294416243654822
F1 наилучшей модели на валидационной выборке: 0.6794520547945205

Метрики для класса 0 smart
Precision наилучшей модели на валидационной выборке: 0.8463157894736842
Recall наилучшей модели на валидационной выборке: 0.9013452914798207
F1 наилучшей модели на валидационной выборке: 0.8729641693811075
```

Проверим на всякий случай более простую модель - есть вероятность, что она может датьрезультат лучше

6.2 Обучим модель алгоритмом логистической регрессии

Ввод [15]:
```
model = LogisticRegression(random_state=12345, class_weight='balanced')
model.fit(features_train_st, target_train)
predictions_valid = model.predict(features_valid_st)
result = accuracy_score(target_valid, predictions_valid)

print("Accuracy модели логистической регрессии на валидационной выборке:", result)
```

```
Accuracy модели логистической регрессии на валидационной выборке: 0.6547433903576982
```

Результат обучения логистической регрессии хуже, чем у случайного леса

7 Проверим модель на тестовой выборке

Случайный лес

Ввод [16]:
```
predictions_test = best_model.predict(features_test_st)
result_test = accuracy_score(target_test, predictions_test)
print('Accuracy наилучшей модели случайного леса на тестовой выборке:', result_test)
```

```
Accuracy наилучшей модели случайного леса на тестовой выборке: 0.7962674961119751
```

Ввод [17]:
```
test_accuracy = accuracy_score(target_test, predictions_test)

# Для класса 1 - ultra
f1_u = f1_score(target_test, predictions_test)
precision_u = precision_score(target_test, predictions_test)
recall_u =  recall_score(target_test, predictions_test)

# для класса 0 - smart
f1_s = f1_score(target_test, predictions_test, pos_label=0)
precision_s = precision_score(target_test, predictions_test, pos_label=0)
recall_s =  recall_score(target_test, predictions_test, pos_label=0)
```

Ввод [18]:
```
print("Accuracy наилучшей модели на тестовой выборке:", test_accuracy)
print('Лучший max_depth:', best_depth)
print('Лучший n_estimators:', best_est)
print()
print('Метрики для класса 1 ultra')
print("Precision наилучшей модели на тестовой выборке:", precision_u)
print("Recall наилучшей модели на тестовой выборке:", recall_u)
print("F1 наилучшей модели на тестовой выборке:", f1_u)
print()
print('Метрики для класса 0 smart')
print("Precision наилучшей модели на тестовой выборке:", precision_s)
print("Recall наилучшей модели на тестовой выборке:", recall_s)
print("F1 наилучшей модели на тестовой выборке:", f1_s)
```

```
Accuracy наилучшей модели на тестовой выборке: 0.7962674961119751
Лучший max_depth: 7
Лучший n_estimators: 7

Метрики для класса 1 ultra
Precision наилучшей модели на тестовой выборке: 0.6987951807228916
Recall наилучшей модели на тестовой выборке: 0.5888324873096447
F1 наилучшей модели на тестовой выборке: 0.6391184573002755

Метрики для класса 0 smart
Precision наилучшей модели на тестовой выборке: 0.8301886792452831
Recall наилучшей модели на тестовой выборке: 0.8878923766816144
F1 наилучшей модели на тестовой выборке: 0.85807150595883
```

7.1 Вывод

По результатам моделирования и тестирования лучшим вариантом модели оказался Случайный лес с accuracy 0.79.

Проверка по качеству предсказания по классам показала, что наша рекомендация будет гораздо лучше работать для людей, которым по их потребностям лучше подходит тариф smart (F1=0.86. Для тарифа ultra F1=0.64). Это может означать, что тариф ultra в реальности объединяет несколько различных групп людей. Рекомендуем проработать еще несколько вариантов тарифов, сделав сегментацию пользователей тарифа ultra.

8 Проверим модель на адекватность

Для оценки нашей модели применим самый простой классификатор, который выдаёт на каждом объекте константное предсказание – самый часто встречающийся класс.

Ввод [19]:
```
dummy_clf = DummyClassifier(strategy="most_frequent", random_state=0)
dummy_clf.fit(features_train, target_train)
dummy_clf.score(features_train, target_train)
```

Out[19]:
```
0.6934647302904564
```

Наша модель случайного леса дает заметно лучший результат, чем простой выбор наиболее частого класса. А вот логистическая регрессия дала даже хуже результат чем dummy-классификатор.

9 Выводы

По результатам тестирования лучшим вариантом модели оказался случайный лес с параметрами max_depth=7, n_estimators=7. Accuracy=0.80.

Выбранная модель лучше, чем простейшая модель (а также модель логистической регрессии). Проверка на адекватность пройдена.

Проверка по качеству предсказания по классам показала, что наша рекомендация будет гораздо лучше работать для людей, которым по их потребностям лучше подходит тариф smart (F1=0.86. Для тарифа ultra F1=0.64). Это может означать, что тариф ultra в реальности объединяет несколько различных групп людей. Рекомендуем компании подобрать еще несколько вариантов тарифа, отталкиваясь от тарифа ultra.

Ввод [ ]:
```

```