Отток клиентов

Из «Бета-Банка» стали уходить клиенты. Каждый месяц. Немного, но заметно. Банковские маркетологи посчитали: сохранять текущих клиентов дешевле, чем привлекать новых.

Нужно спрогнозировать, уйдёт клиент из банка в ближайшее время или нет. Вам предоставлены исторические данные о поведении клиентов и расторжении договоров с банком.

Постройте модель с предельно большим значением *F1*-меры. Чтобы сдать проект успешно, нужно довести метрику до 0.59. Проверьте *F1*-меру на тестовой выборке самостоятельно.

Дополнительно измеряйте *AUC-ROC*, сравнивайте её значение с *F1*-мерой.

Источник данных: https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling (https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling)

План

B [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objs as go
from plotly.subplots import make_subplots
import plotly.express as px

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.utils import shuffle
from sklearn.metrics import roc_auc_score, roc_curve

import os
import urllib

pd.options.plotting.backend = "plotly"
```

1. Подготовка данных

B [2]:
```
name, url = 'datasets/Churn.csv', 'https://code.s3.yandex.net/datasets/Churn.csv'
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 [3]:
```
data = pd.read_csv(name)
data.sample(5)
```

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balanc
5484	5485	15595842	Paramor	748	Germany	Male	45	2.0	119852.0
9818	9819	15619699	Yeh	558	France	Male	31	7.0	0.0
3252	3253	15699619	Rivas	641	France	Male	31	10.0	155978.1
6570	6571	15790958	Sanders	685	Spain	Male	38	4.0	0.0
5988	5989	15809227	Chukwudi	850	France	Male	35	2.0	0.0

In [4]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           9091 non-null   float64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(3), int64(8), object(3)
memory usage: 1.1+ MB
```

Описание данных

Признаки

- `RowNumber` — индекс строки в данных
- `CustomerId` — уникальный идентификатор клиента
- `Surname` — фамилия
- `CreditScore` — кредитный рейтинг
- `Geography` — страна проживания
- `Gender` — пол
- `Age` — возраст
- `Tenure` — количество недвижимости у клиента
- `Balance` — баланс на счёте
- `NumOfProducts` — количество продуктов банка, используемых клиентом
- `HasCrCard` — наличие кредитной карты
- `IsActiveMember` — активность клиента
- `EstimatedSalary` — предполагаемая зарплата

Целевой признак

- `Exited` — факт ухода клиента

In [5]:
```
data = data.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1)
```

B [6]:
```
def CamelCase_to_norm_case(s):
    return ''.join('_' + c.lower() if c.isupper() and i else c.lower() for i, c i

data.columns = [CamelCase_to_norm_case(col) for col in data.columns]
```

B [7]:
```
data.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%
credit_score	10000.0	650.528800	96.653299	350.00	584.00	652.000	718.0000
age	10000.0	38.921800	10.487806	18.00	32.00	37.000	44.0000
tenure	9091.0	4.997690	2.894723	0.00	2.00	5.000	7.0000
balance	10000.0	76485.889288	62397.405202	0.00	0.00	97198.540	127644.2400
num_of_products	10000.0	1.530200	0.581654	1.00	1.00	1.000	2.0000
has_cr_card	10000.0	0.705500	0.455840	0.00	0.00	1.000	1.0000
is_active_member	10000.0	0.515100	0.499797	0.00	0.00	1.000	1.0000
estimated_salary	10000.0	100090.239881	57510.492818	11.58	51002.11	100193.915	149388.2475
exited	10000.0	0.203700	0.402769	0.00	0.00	0.000	0.0000

B [8]:
```
data_original = data.copy()

data.tenure = data.tenure.fillna(
    data.credit_score.map(
        data.groupby('credit_score').tenure.median()
    )
)

print(data.tenure.isna().sum())
```

```
2
```

B [9]:
```
query_str = 'geography == "France" and 35 < age < 45 and 96_000 < balance < 130_0
data.tenure = data.tenure.fillna(data.query(query_str).tenure.median())
```

B [10]:
```
print(data.shape[0], data.isna().sum().sum())
```

```
10000 0
```

Вывод

Изменили названия столбцов. Удалили ненужные для исследования признаки. В данных есть дубликатов нет. Имели место пропуски в столбце `tenure`, но сгуппировав по признаку `credit_score` нашли медианы, которыми и заполнили пропуски.

2. Исследование задачи

B [11]:
```
categorical_features = ['gender', 'geography']
numeric_features = list(set(data.columns) - set(categorical_features + ['exited'])
```

B [12]:
```
encoder = OrdinalEncoder()
data.loc[:, categorical_features] =\
    pd.DataFrame(encoder.fit_transform(data[categorical_features]), columns=categ
```

B [13]:
```
features = data.drop(['exited'], axis=1)
target = data.exited
```

Исследован баланс классов

В [14]:
```
try:
    fig = go.Figure()
    fig.add_trace(go.Histogram(x=data['exited'], histnorm='percent', marker_color
    fig.update_xaxes(range=[0, 1])
    fig.update_layout(title_text='Гистограмма целевого признака (баланс классов)'
                      title_x=.5,
                      xaxis_title='Целевой признака',
                      yaxis_title='Процент от общего числа',
                     )
    fig.show()
except:
    target.hist()
    plt.show()
```

Изучены модели без учёта дисбаланса

B [15]:
```
main_out = {'model': [], 'descr':[], 'estim': [], 'depth': [], 'train': [], 'vali


def add_data(main_out, model, description, new_one):
    main_out['model'] += [model]
    main_out['descr'] += [description]
    for k in new_one:
        main_out[k] += [new_one[k]]
    return main_out
```

B [16]:
```
def classification(features, target, model_name, score, scale=True, class_weight=
    def sample(ftrs, trgt, ratio, down=False):
        features_z = ftrs[trgt == 0]
        target_z = trgt[trgt == 0]
        features_o = ftrs[trgt == 1]
        target_o = trgt[trgt == 1]

        features_new = pd.concat([features_z.sample(frac=ratio)] + [features_o])
            else pd.concat([features_z] + [features_o] * ratio)
        target_new = pd.concat([target_z.sample(frac=ratio)] + [target_o]) if dow
            else pd.concat([target_z] + [target_o] * ratio)

        features_new, target_new = shuffle(features_new, target_new, random_state
        return features_new, target_new

    def fit_predict_score(estim, depth, test=False):
        balance_classes = 'balanced' if class_weight else None
        if 'forest' in model_name.lower():
            mdl = model(random_state=1, n_estimators=estim, max_depth=depth, clas
        else:
            mdl = model(random_state=1, max_depth=depth, class_weight=balance_cla
        mdl.fit(features_train, target_train)

        pred_train = mdl.predict(features_train)
        score_train = score(target_train, pred_train)
        pred_valid = mdl.predict(features_valid)
        score_valid = score(target_valid, pred_valid)

        if test:
            probs_ones_valid = mdl.predict_proba(features_valid)[:, 1]
            roc_valid = roc_auc_score(target_valid, probs_ones_valid)

            pred_test = mdl.predict(features_test)
            score_test = score(target_test, pred_test)

            probs_ones_test = mdl.predict_proba(features_test)[:, 1]
            roc_test = roc_auc_score(target_test, probs_ones_test)
            return {'train':score_train, 'valid':score_valid, 'test':score_test,
        else:
            return {'train':score_train, 'valid':score_valid, 'test':'-', 'auc_va

    features_train, features_valid, target_train, target_valid =\
        train_test_split(features, target, random_state=1, test_size=.4)
    features_valid, features_test, target_valid, target_test =\
        train_test_split(features_valid, target_valid, random_state=1, test_size=

    if upsampling:
        features_train, target_train = sample(features_train, target_train, ratio
    elif downsampling:
        features_train, target_train = sample(features_train, target_train, ratio

    if scale:
        scaler = StandardScaler()
        scaler.fit(features_train)
        features_train = scaler.transform(features_train)
        features_valid = scaler.transform(features_valid)
```

```
        features_test = scaler.transform(features_test)

    out = {'estim': 0, 'depth': 0, 'train': 0, 'valid': 0, 'test': 0, 'auc_valid'
    if 'forest' in model_name.lower():
        model = RandomForestClassifier
        for dep in range(1, 10):
            for est in range(5, 30):
                cur = fit_predict_score(est, dep)
                if out['valid'] < cur['valid']:
                    out['estim'], out['depth'] = est, dep
                    out['train'], out['valid'] = cur['train'], cur['valid']
    else:
        model = DecisionTreeClassifier
        for dep in range(1, 10):
            cur = fit_predict_score('-', dep)
            if out['valid'] < cur['valid']:
                out['estim'], out['depth'] = '-', dep
                out['train'], out['valid'] = cur['train'], cur['valid']
    res = fit_predict_score(out['estim'], out['depth'], test=True)
    out['test'], out['auc_valid'], out['auc_test'] = res['test'], res['auc_valid'
    return out
```

B [17]:
```
%%time
main_out = add_data(main_out, 'RandomForest', 'Not being scaled',\
                    classification(features, target, 'forest', f1_score, scale=Fa
```

```
Wall time: 19 s
```

B [18]:
```
%%time
main_out = add_data(main_out, 'DecisionTree', 'Not being scaled',\
                    classification(features, target, 'tree', f1_score, scale=Fals
```

```
Wall time: 235 ms
```

B [19]:
```
%%time
main_out = add_data(main_out, 'RandomForest', 'Not being balanced, but scaled',\
                    classification(features, target, 'forest', f1_score))
```

```
Wall time: 19.2 s
```

B [20]:
```
%%time
main_out = add_data(main_out, 'DecisionTree', 'Not being balanced, but scaled',\
                    classification(features, target, 'tree', f1_score))
```

```
Wall time: 170 ms
```

Вывод:

Имеет место дисбаланс классов у целевого признака. Для дальнейших сравнений обучили модели на имеюшихся даных (без баланса классов)
и получили значение соответствующих метрик

3. Учёт дисбаланс

Применено несколько способов борьбы с дисбалансом

class_weight

B [21]:
```
%%time
main_out = add_data(main_out, 'RandomForest', 'Being balanced by \'class_weight\'
                    classification(features, target, 'forest', f1_score, class_we
```

```
Wall time: 19.1 s
```

B [22]:
```
%%time
main_out = add_data(main_out, 'DecisionTree', 'Being balanced by \'class_weight\'
                    classification(features, target, 'tree', f1_score, class_weig
```

```
Wall time: 185 ms
```

Upsampling

B [23]:
```
%%time
main_out = add_data(main_out, 'RandomForest', 'Being balanced by Upsampling',\
                    classification(features, target, 'forest', f1_score, upsampli
```

```
Wall time: 25.5 s
```

B [24]:
```
%%time
main_out = add_data(main_out, 'DecisionTree', 'Being balanced by Upsampling',\
                    classification(features, target, 'tree', f1_score, upsampling
```

```
Wall time: 248 ms
```

Downsampling

B [25]:
```
%%time
main_out = add_data(main_out, 'RandomForest', 'Being balanced by Downsampling',\
                    classification(features, target, 'forest', f1_score, downsamp
```

```
Wall time: 11.7 s
```

B [26]:
```
%%time
main_out = add_data(main_out, 'DecisionTree', 'Being balanced by Downsampling',\
                    classification(features, target, 'tree', f1_score, downsampli
```

```
Wall time: 105 ms
```

Результаты обучения и предсказаний (чтобы не ждать)

B [27]:
```
result = pd.DataFrame(main_out)
estim = result[result.test == result.test.max()].estim.iloc[0]
depth = result[result.test == result.test.max()].depth.iloc[0]
pd.DataFrame(result.groupby(['model', 'descr']).first())
```

Out[27]:

model	descr	estim	depth	train	valid	test	auc_valid	auc_test
DecisionTree	**Being balanced by 'class_weight'**	-	5	0.596067	0.597484	0.589147	0.819756	0.827758
	Being balanced by Downsampling	-	6	0.777184	0.585413	0.568569	0.830222	0.821145
	Being balanced by Upsampling	-	5	0.733364	0.597484	0.589147	0.819756	0.827758
	Not being balanced, but scaled	-	9	0.678218	0.561737	0.554286	0.792718	0.778615
	Not being scaled	-	9	0.678218	0.561737	0.554286	0.792718	0.778615
RandomForest	**Being balanced by 'class_weight'**	24	9	0.743383	0.628261	0.607889	0.854641	0.854368
	Being balanced by Downsampling	25	6	0.795435	0.604167	0.598000	0.863433	0.860325
	Being balanced by Upsampling	22	9	0.868538	0.622544	0.614894	0.855436	0.853543
	Not being balanced, but scaled	8	8	0.615630	0.583717	0.540123	0.846514	0.852024
	Not being scaled	8	8	0.616366	0.582435	0.533333	0.846829	0.852378

Вывод

На примере модели `DecisionTreeClassifier` и `RandomForestClassifier` нашли оптимальные гиперпараметры для каждого случая, а именно:

- без масштабирования и баланса классов

- без баланса классов, но с масштабированием
- с балансом (с помощью параметра `class_weight` , `upsampling` , `downsampling`)

Результаты смотри в таблице выше. Итого `DecisionTreeClassifier` с наилучшими показателями:

B [28]:
```
result[result.valid == result[result.model == "DecisionTree"].valid.max()]\
    .drop(['estim'], axis=1).reset_index(drop=True)
```

Out[28]:

	model	descr	depth	train	valid	test	auc_valid	auc_test
0	DecisionTree	Being balanced by 'class_weight'	5	0.596067	0.597484	0.589147	0.819756	0.827758
1	DecisionTree	Being balanced by Upsampling	5	0.733364	0.597484	0.589147	0.819756	0.827758

`RandomForestClassifier` с наилучшими показателями:

B [29]:
```
result[result.valid == result[result.model == "RandomForest"].valid.max()].reset_
```

Out[29]:

	model	descr	estim	depth	train	valid	test	auc_valid	auc_test
0	RandomForest	Being balanced by 'class_weight'	24	9	0.743383	0.628261	0.607889	0.854641	0.854368

`train` , `valid` , `test` - столбцы обозначают значения метрики `F1` для соответствующих частей датасета

На основании значений метрики `f1 score` на валидационной выборке нашли модели с лучшими показателями.

`RandomForestClassifier`

- `esitm = 24`
- `depth = 9`
- `class_weight = 'balanced'`

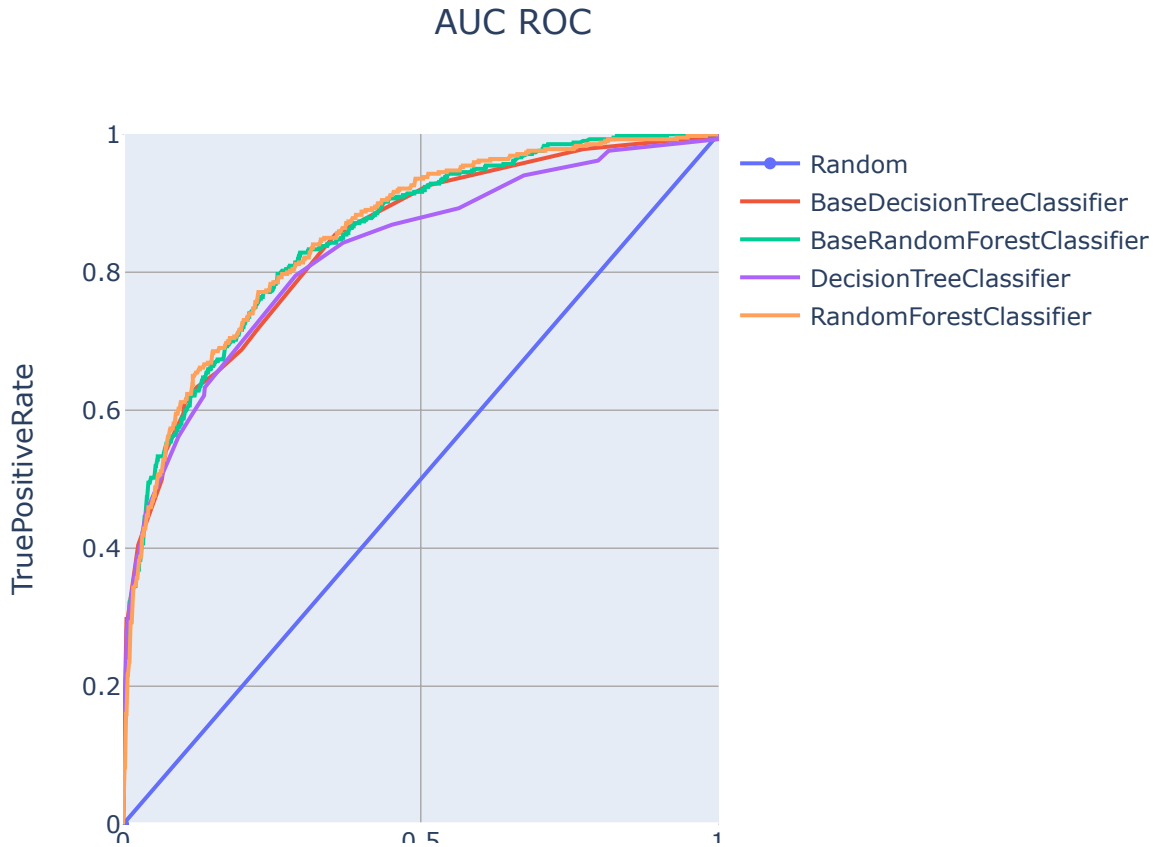
`DecisionTreeClassifier`

- `depth = 5`
- `class_weight = 'balanced'`

4. Тестирование модели

B [30]:
```
features_train, features_valid, target_train, target_valid =\
    train_test_split(features, target, random_state=1, test_size=.4)
features_valid, features_test, target_valid, target_test =\
    train_test_split(features_valid, target_valid, random_state=1, test_size=.5)

model_tree = DecisionTreeClassifier(random_state=1, max_depth=5)
model_tree.fit(features_train, target_train)
probs_test_tree = model_tree.predict_proba(features_test)[:, 1]
fpr_tree_base, tpr_tree_base, threshold_tree_base= roc_curve(target_test, probs_t

model_forest = RandomForestClassifier(random_state=1, n_estimators=8, max_depth=8
model_forest.fit(features_train, target_train)
probs_test_forest = model_forest.predict_proba(features_test)[:, 1]
fpr_forest_base, tpr_forest_base, threshold_forest_base = roc_curve(target_test,

scaler = StandardScaler()
scaler.fit(features_train)
features_train = scaler.transform(features_train)
features_valid = scaler.transform(features_valid)
features_test = scaler.transform(features_test)

model_tree = DecisionTreeClassifier(random_state=1, max_depth=5, class_weight='ba
model_tree.fit(features_train, target_train)
probs_test_tree = model_tree.predict_proba(features_test)[:, 1]
fpr_tree, tpr_tree, threshold_tree = roc_curve(target_test, probs_test_tree)

model_forest = RandomForestClassifier(random_state=1, n_estimators=24, max_depth=
model_forest.fit(features_train, target_train)
probs_test_forest = model_forest.predict_proba(features_test)[:, 1]
fpr_forest, tpr_forest, threshold_forest = roc_curve(target_test, probs_test_fore

fpr_random, tpr_random, threshold_random = roc_curve(target_test, pd.Series(0.5,
```

B [31]:
```
try:
    fig = go.Figure()

    fig.add_trace(go.Scatter(x=fpr_random, y=tpr_random, name='Random'))
    fig.add_trace(go.Scatter(x=fpr_tree_base, y=tpr_tree_base, name='BaseDecisior
    fig.add_trace(go.Scatter(x=fpr_forest_base, y=tpr_forest_base, name='BaseRanc
    fig.add_trace(go.Scatter(x=fpr_tree, y=tpr_tree, name='DecisionTreeClassifier
    fig.add_trace(go.Scatter(x=fpr_forest, y=tpr_forest, name='RandomForestClassi

    fig.update_xaxes(range=[0, 1])
    fig.update_yaxes(range=[0, 1])

    fig.update_layout(title_text='AUC ROC',
                      title_x=.45,
                      xaxis_title='FalsePositiveRate',
                      yaxis_title='TruePositiveRate',
     )
    fig.show()
except:
    plt.figure()
    plt.plot(fpr_tree, tpr_tree)
    plt.plot(fpr_forest, tpr_forest)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.0])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC-curve')
    plt.show()
```



Вывод:

Посмотрели на `ROC-кривую` (Receiver Operating Characteristic) для 4-ёх модей со значениями гиперпараметров, которые дают лучшие значения метрик.

- `Random` (модель далет предсказания случайно, следовательно, вероятность 0.5)
- `BaseDecisionTreeClassifier` (модель без баланса классов и масшитабирования)
- `BaseRandomForestClassifier` (модель без баланса классов и масшитабирования)
- `DecisionTreeClassifier`
- `RandomForestClassifier`

Качественно подтверидили, что **лучшая** модель для классификации (целевой признак) - **`RandomForestClassifier`**

- `esitm = 24`
- `depth = 9`
- `class_weight = 'balanced'`

Также на графике видно, что баланс классов и масштабирование вносит небольшой вклад в увеличении качества модели **`RandomForestClassifier`**