Содержание

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

Подготовка данных

Проверка данных

In [1]:
```
# Импортируем необходимые библиотеки для выполнения проекта
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.utils import shuffle
from sklearn.model_selection import GridSearchCV
```

In [2]:
```
data= pd.read_csv('/datasets/Churn.csv')
```

In [3]:
```
#Общая информация

print('Общая информация:')

display(data.info())

print(30*' =')

#Вывод первых пяти строк

print('Первые пять строк датафрейма:')

display(data.head())

print(30*' =')

print('Название столбцов датафрейма:')

print(f'Название столбцов: {list(data.columns)}')

print(30*' =')

#Пропуски

print('Информация о пропусках:')

display(data.isnull().sum())

print(30*' =')

#Полные дубликаты

print(f'Полных дубликатов: {data.duplicated().sum()}')

print(30*' =')
```

```
Общая информация:
<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
None
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
Первые пять строк датафрейма:
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2.0	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1.0	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8.0	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1.0	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2.0	125510.82	1	1	1	79084.10	0

```
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
Название столбцов датафрейма:
Название столбцов: ['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary', 'Exited']
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
Информация о пропусках:
RowNumber            0
CustomerId           0
Surname              0
CreditScore          0
Geography            0
Gender               0
Age                  0
Tenure             909
Balance              0
NumOfProducts        0
HasCrCard            0
IsActiveMember       0
EstimatedSalary      0
Exited               0
dtype: int64
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
Полных дубликатов: 0
 = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
```

- Приводим название столбцов к нижнему регистру, также преобразуем названия столбцов :

In [4]:
```
data = data.rename(columns = {'RowNumber': 'row_number', 'CustomerId':'custom_id', 'Surname':'surname', 'CreditScore':'credit_score', 'NumOfProducts':'num_of_products',
                              'EstimatedSalary':'estimated_salary' })

data.columns = map(str.lower, data.columns)

# заполняем пропуски нулевым значение в значениях Tenure (сколько лет человек является клиентом банка)
data['tenure'] = data['tenure'].fillna(0)

display(data.head())
display(data.info())
```

	row_number	custom_id	surname	credit_score	geography	gender	age	tenure	balance	num_of_products	has_cr_card	is_active_member	estimated_salary	exited
0	1	15634602	Hargrave	619	France	Female	42	2.0	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1.0	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8.0	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1.0	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2.0	125510.82	1	1	1	79084.10	0

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   row_number        10000 non-null  int64
 1   custom_id         10000 non-null  int64
 2   surname           10000 non-null  object
 3   credit_score      10000 non-null  int64
 4   geography         10000 non-null  object
 5   gender            10000 non-null  object
 6   age               10000 non-null  int64
 7   tenure            10000 non-null  float64
 8   balance           10000 non-null  float64
 9   num_of_products   10000 non-null  int64
 10  has_cr_card       10000 non-null  int64
 11  is_active_member  10000 non-null  int64
 12  estimated_salary  10000 non-null  float64
 13  exited            10000 non-null  int64
dtypes: float64(3), int64(8), object(3)
memory usage: 1.1+ MB
None
```

Данные кооректны, пропусков и явных дубликатов нет

Подготовка данных для машинного обучения

- Устианавливаем целевой признак - 'exited' - факт ухода Клиента из Банка
- Удаляем столбцы данные которых не понадобятся в исследовании

In [5]:
```
data_new = data.drop(['custom_id','surname', 'row_number'], axis=1)


display(data_new.head())
```

	credit_score	geography	gender	age	tenure	balance	num_of_products	has_cr_card	is_active_member	estimated_salary	exited
0	619	France	Female	42	2.0	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1.0	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8.0	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1.0	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2.0	125510.82	1	1	1	79084.10	0

- Преобразуем категориальные признаки в количественные для gender, geography и age используя метод OHE:

In [6]:
```
data_new = pd.get_dummies(data_new, drop_first=True)

display(data_new.head())

data_new.shape
```

	credit_score	age	tenure	balance	num_of_products	has_cr_card	is_active_member	estimated_salary	exited	geography_Germany	geography_Spain	gender_Male
0	619	42	2.0	0.00	1	1	1	101348.88	1	0	0	0
1	608	41	1.0	83807.86	1	0	1	112542.58	0	0	1	0
2	502	42	8.0	159660.80	3	1	0	113931.57	1	0	0	0
3	699	39	1.0	0.00	2	0	0	93826.63	0	0	0	0
4	850	43	2.0	125510.82	1	1	1	79084.10	0	0	1	0

Out[6]:
```
(10000, 12)
```

Создаем учебную, валидационную и тестовые выборки

In [7]:
```
# разобьем данные на признаки и целевой признак
features = data_new.drop(['exited'], axis=1)
target = data_new['exited']

display(features.head())
display(target.head())
```

	credit_score	age	tenure	balance	num_of_products	has_cr_card	is_active_member	estimated_salary	geography_Germany	geography_Spain	gender_Male
0	619	42	2.0	0.00	1	1	1	101348.88	0	0	0
1	608	41	1.0	83807.86	1	0	1	112542.58	0	1	0
2	502	42	8.0	159660.80	3	1	0	113931.57	0	0	0
3	699	39	1.0	0.00	2	0	0	93826.63	0	0	0
4	850	43	2.0	125510.82	1	1	1	79084.10	0	1	0

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

In [8]:
```
# валидационная выборка (в пропорции 60:40 от общих данных)
features_train, features_valid_test, target_train, target_valid_test,  = train_test_split(features, target,
                                                                                          train_size=0.60, random_state=12345, stratify=target)
print('Признаки обучающей выборки:', features_train.shape)
print('Целевой признак обучающей выборки:', target_train.shape)
print('Признаки валидационной выборки:', features_valid_test.shape)
print('Целевой признаки валидационной выборки:', target_valid_test.shape)
```

```
Признаки обучающей выборки: (6000, 11)
Целевой признак обучающей выборки: (6000,)
Признаки валидационной выборки: (4000, 11)
Целевой признаки валидационной выборки: (4000,)
```

In [9]:
```
# тестовая выборка (в пропорции 50:50 от валидационной)
features_valid, features_test, target_valid, target_test = train_test_split(features_valid_test,
                                                                            target_valid_test,
                                                                            train_size=0.5,
                                                                            random_state=12345, stratify=target_valid_test)


print('Признаки и целевые признаки по валидационной и целевой выборкам:', features_valid.shape, target_valid.shape, features_test.shape, target_test.shape)
```

```
Признаки и целевые признаки по валидационной и целевой выборкам: (2000, 11) (2000,) (2000, 11) (2000,)
```

Масштабирование

- Поскольку у формат колличественных данных значительно отличается друг от друга необходимо применить мастшатбирование, чтобы машина не сделела ошибочных предположении о важности данных чей удельный вес выше.

In [10]:
```
# выделим количественные признаки
numeric = ['credit_score', 'age', 'tenure', 'balance', 'num_of_products', 'estimated_salary']
```

In [11]:
```
# масштабируем на обучающей выборке
scaler = StandardScaler()
scaler.fit(features_train[numeric])
```

Out[11]:
```
StandardScaler()
```

In [12]:
```
#Масштабируем количественные признаки обучающей выборки
features_train[numeric] = scaler.transform(features_train[numeric])
features_train.head()
```

```
/tmp/ipykernel_79/2218298652.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  features_train[numeric] = scaler.transform(features_train[numeric])
/opt/conda/lib/python3.9/site-packages/pandas/core/indexing.py:1738: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(loc, value[:, i].tolist(), pi)
```

Out[12]:

	credit_score	age	tenure	balance	num_of_products	has_cr_card	is_active_member	estimated_salary	geography_Germany	geography_Spain	gender_Male
2837	-1.040434	0.953312	0.467449	0.774657	-0.914708	0	1	-0.119110	1	0	0
9925	0.454006	-0.095244	-1.461501	1.910540	-0.914708	1	1	-0.258658	0	0	0
8746	0.103585	-0.476537	1.110432	0.481608	0.820981	0	1	1.422836	0	0	1
660	-0.184996	0.190726	-1.461501	0.088439	-0.914708	1	1	-1.160427	1	0	0
3610	-0.720933	1.620574	-1.140009	0.879129	-0.914708	1	0	0.113236	0	0	0

In [13]:
```
#Масштабируем количественные признаки валидационной выборки
features_valid[numeric] = scaler.transform(features_valid[numeric])
features_valid.head()
```

```
/tmp/ipykernel_79/2605056509.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  features_valid[numeric] = scaler.transform(features_valid[numeric])
/opt/conda/lib/python3.9/site-packages/pandas/core/indexing.py:1738: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(loc, value[:, i].tolist(), pi)
```

Out[13]:

	credit_score	age	tenure	balance	num_of_products	has_cr_card	is_active_member	estimated_salary	geography_Germany	geography_Spain	gender_Male
6612	-1.524838	2.383160	-0.497026	1.011655	-0.914708	0	1	0.010275	0	0	0
519	0.587990	1.811221	0.467449	0.213192	-0.914708	0	1	0.453089	0	0	1
1609	-0.318980	-0.381213	1.110432	0.893896	-0.914708	1	0	-0.603508	0	0	0
3475	1.226991	-0.762506	-0.175534	-0.240363	0.820981	1	1	-1.621460	0	0	1
2707	-0.133464	-0.095244	-0.497026	0.447305	-0.914708	0	0	0.338574	0	0	0

In [14]:
```
#Масштабируем количественные признаки тестовой выборки
features_test[numeric] = scaler.transform(features_test[numeric])
features_test.head()
```

```
/tmp/ipykernel_79/626509267.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  features_test[numeric] = scaler.transform(features_test[numeric])
/opt/conda/lib/python3.9/site-packages/pandas/core/indexing.py:1738: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._setitem_single_column(loc, value[:, i].tolist(), pi)
```

Out[14]:

	credit_score	age	tenure	balance	num_of_products	has_cr_card	is_active_member	estimated_salary	geography_Germany	geography_Spain	gender_Male
657	-0.504497	1.429928	-1.140009	0.873883	0.820981	1	1	1.725255	1	0	0
2083	-1.195031	-1.429769	-1.140009	-1.219673	-0.914708	1	1	1.203665	0	0	1
3234	-1.267176	0.762665	-1.140009	-1.219673	-0.914708	1	0	-1.374743	0	0	1
1551	0.825039	1.239281	0.788940	-1.219673	0.820981	0	1	0.382306	0	1	0
2344	0.660135	0.953312	-0.175534	0.267087	-0.914708	1	0	-1.613578	1	0	0

Масштабированные данные для обучающей, валидационной и тестовой выборок получены

Исследование задачи

Обучаем модели на обучающей выборке и находим точность каждой из них на валидационной

In [15]:
```
model_tree = DecisionTreeClassifier(random_state=123)
tree_score = model_tree.fit(features_train, target_train).score(features_valid, target_valid)

model_forest = RandomForestClassifier(random_state=12345, n_estimators = 100)
forest_score = model_forest.fit(features_train, target_train).score(features_valid, target_valid)

model_regression = LogisticRegression(solver = 'liblinear')
regression_score = model_regression.fit(features_train, target_train).score(features_valid, target_valid)


print("Точность:" "Дерево решений", tree_score, "Случайный лес ", forest_score, "Логистческая регрессия", regression_score)
```

```
Точность:Дерево решений 0.795 Случайный лес  0.8685 Логистческая регрессия 0.811
```

- проверяем балланс классов

In [16]:
```
target_train.value_counts(normalize = 1)
```

Out[16]:
```
0    0.796333
1    0.203667
Name: exited, dtype: float64
```

In [17]:
```
target_valid.value_counts(normalize = 1)
```

Out[17]:
```
0    0.796
1    0.204
Name: exited, dtype: float64
```

Вывод: у классов наблюдается дисбалланс, поэтому очевидно что предсказания моделей будет склоняться в пользу варианта ответа - 0

Проверка на адекватность моделей

In [18]:
```
# модель решающее дерево
model_tree = DecisionTreeClassifier(random_state=1234)
model_tree.fit(features_train, target_train)
model_tree_class_frequency = pd.Series(model_tree.predict(features_valid)).value_counts(normalize = 1)

print(model_tree_class_frequency)
```

```
0    0.782
1    0.218
dtype: float64
```

In [19]:
```
# модель случайный лес
model_forest = RandomForestClassifier(random_state=12345, n_estimators = 100)
model_forest.fit(features_train, target_train)
model_forest_class_frequency = pd.Series(model_forest.predict(features_valid)).value_counts(normalize = 1)
print(model_forest_class_frequency)
```

```
0    0.8665
1    0.1335
dtype: float64
```

In [20]:
```
# модель логистической регрессии
model_regression = LogisticRegression(solver = 'liblinear')
model_regression.fit(features_train, target_train)
model_regression_class_frequency = pd.Series(model_regression.predict(features_valid)).value_counts(normalize = 1)
print(model_regression_class_frequency)
```

```
0    0.931
1    0.069
dtype: float64
```

Самая высокая точность у логистической регресии

- Сравним качество предсказаний моделей с точностью константной модели

In [21]:
```
target_predict_constant = pd.Series([0]*len(target_valid))
accuracy_score_constant = accuracy_score(target_valid, target_predict_constant)

print(accuracy_score_constant)
```

```
0.796
```

Точность константной модели является првктичеки такой же как и точность других моделей, что указывает на неадекаватность и на возможный дисбаланс классов в данных моделях

In [22]:
```
# создаем матрицу ошибок для дерева решений изучаем полноту, точность и F1 меру

model_tree = DecisionTreeClassifier(random_state=123)
model_tree.fit(features_train, target_train)
model_tree_prediction = model_tree.predict(features_valid)
confusion_matrix(target_valid, model_tree_prediction)
```

Out[22]:
```
array([[1374,  218],
       [ 192,  216]])
```

In [23]:
```
print("Полнота" , recall_score(target_valid, model_tree_prediction))
print("Точность", precision_score(target_valid, model_tree_prediction))
print("F1-мера", f1_score(target_valid, model_tree_prediction))
print("AUC-ROC", roc_auc_score(target_valid, model_tree_prediction))
```

```
Полнота 0.5294117647058824
Точность 0.4976958525345622
F1-мера 0.5130641330166271
AUC-ROC 0.696238545669524
```

In [24]:
```
# создаем матрицу ошибок для случайного леса изучаем полноту, точность и F1 меру

model_forest = RandomForestClassifier(random_state=1234, n_estimators = 100)
model_forest.fit(features_train, target_train)
model_forest_prediction = model_forest.predict(features_valid)
confusion_matrix(target_valid, model_forest_prediction)
```

Out[24]:
```
array([[1542,   50],
       [ 209,  199]])
```

In [25]:
```
print("Полнота" , recall_score(target_valid, model_forest_prediction))
print("Точность", precision_score(target_valid, model_forest_prediction))
print("F1-мера", f1_score(target_valid, model_forest_prediction))
print("AUC-ROC", roc_auc_score(target_valid, model_forest_prediction))
```

```
Полнота 0.4877450980392157
Точность 0.7991967871485943
F1-мера 0.6057838660578386
AUC-ROC 0.7281690314316682
```

In [26]:
```
# создаем матрицу ошибок для логистической регрессии

model_regression  = LogisticRegression(solver = 'liblinear')
model_regression.fit(features_train, target_train)
model_regression_prediction = model_regression.predict(features_valid)
confusion_matrix(target_valid, model_regression_prediction)
```

Out[26]:
```
array([[1538,   54],
       [ 324,   84]])
```

In [27]:
```
print("Полнота" , recall_score(target_valid, model_regression_prediction))
print("Точность", precision_score(target_valid, model_regression_prediction))
print("F1-мера", f1_score(target_valid, model_regression_prediction))
print("AUC-ROC", roc_auc_score(target_valid, model_regression_prediction))
```

```
Полнота 0.20588235294117646
Точность 0.6086956521739131
F1-мера 0.3076923076923077
AUC-ROC 0.5859813774756133
```

Значение агрегирующей метрики F1 выше всего у модели Случайный лес, тем не менее данного значения не достаточно, необходимо поработать над точностью, чтобы увеличить значение общего показателя, определяющего предсказания модели

Борьба с дисбалансом

Для увеличения точности моделей необходимо прийти к баллансу классов данных, дающих положительный и отрицательные ответы в целевом признаке

In [28]:
```
def upsample(features, target, repeat, upsampled_class):
    features_zeros = features[target == 0]
    features_ones = features[target == 1]
    target_zeros = target[target == 0]
    target_ones = target[target == 1]

    if upsampled_class == 0:
        features_upsampled = pd.concat([features_zeros]* repeat + [features_ones] )
        target_upsampled = pd.concat([target_zeros]* repeat + [target_ones] )
        features_upsampled, target_upsampled = shuffle(
        features_upsampled, target_upsampled, random_state=12345)

    elif upsampled_class == 1:
        features_upsampled = pd.concat([features_zeros] + [features_ones] * repeat)
        target_upsampled = pd.concat([target_zeros] + [target_ones] * repeat)
        features_upsampled, target_upsampled = shuffle(
        features_upsampled, target_upsampled, random_state=12345)
    else:
        features_upsampled = 0
        target_upsampled = 0
    return features_upsampled, target_upsampled


features_train_upsampled, target_train_upsampled = upsample(features_train, target_train, 4, 1)
print(target_train_upsampled.value_counts(normalize = 1))
print(target_train_upsampled.shape)
```

```
1    0.50569
0    0.49431
Name: exited, dtype: float64
(9666,)
```

Сбалансированное количество классов получено

Обучение моделей на сбалансированой выборке

In [29]:
```
# Обучаем модели на  upsample выборке и проверяем точность предсказаний на валидационной
model_tree = DecisionTreeClassifier(random_state=123)
tree_score = model_tree.fit(features_train_upsampled, target_train_upsampled).score(features_valid, target_valid)

model_forest = RandomForestClassifier(random_state=123, n_estimators = 100)
forest_score = model_forest.fit(features_train_upsampled, target_train_upsampled).score(features_valid, target_valid)

model_regression = LogisticRegression(solver = 'liblinear')
regression_score = model_regression.fit(features_train_upsampled, target_train_upsampled).score(features_valid, target_valid)


print("Точность:" "Дерево решений", tree_score, "Случайный лес ", forest_score, "Логистческая регрессия", regression_score)
```

```
Точность:Дерево решений 0.8065 Случайный лес  0.861 Логистческая регрессия 0.7255
```

Точность моделей немного увеличилась. Посмторим изменилось ли качество предсказаний

In [30]:
```
# Обучаем модель решающее дерево на сбалансированной выборке изучаем полноту, точность и F1
model_tree_upsampled = DecisionTreeClassifier(random_state=123)
model_tree_upsampled.fit(features_train_upsampled, target_train_upsampled)
model_tree_upsampled_prediction = model_tree_upsampled.predict(features_valid)

print("Полнота" , recall_score(target_valid, model_tree_upsampled_prediction))
print("Точность", precision_score(target_valid, model_tree_upsampled_prediction))
print("F1-мера", f1_score(target_valid, model_tree_upsampled_prediction))
print("AUC-ROC", roc_auc_score(target_valid, model_tree_upsampled_prediction))
```

```
Полнота 0.5147058823529411
Точность 0.5263157894736842
F1-мера 0.5204460966542751
AUC-ROC 0.6979936446940584
```

In [31]:
```
# Обучаем модель случайный лес на сбалансированной выборке изучаем полноту, точность и F1

model_forest_upsampled = RandomForestClassifier(random_state=1234, n_estimators = 100)
model_forest_upsampled.fit(features_train_upsampled, target_train_upsampled)
model_forest_upsampled_prediction = model_forest_upsampled.predict(features_valid)

print("Полнота" , recall_score(target_valid, model_forest_upsampled_prediction))
print("Точность", precision_score(target_valid, model_forest_upsampled_prediction))
print("F1-мера", f1_score(target_valid, model_forest_upsampled_prediction))
print("AUC-ROC", roc_auc_score(target_valid, model_forest_upsampled_prediction))
```

```
Полнота 0.5735294117647058
Точность 0.6763005780346821
F1-мера 0.6206896551724138
AUC-ROC 0.751588826485368
```

In [32]:
```
# Обучаем модель Логистическая регрессия на сбалансированной выборке изучаем полноту, точность и F1

model_regression_upsampled  = LogisticRegression(solver = 'liblinear')
model_regression_upsampled.fit(features_train_upsampled, target_train_upsampled)
model_regression_upsampled_prediction = model_regression_upsampled.predict(features_valid)

print("Полнота" , recall_score(target_valid, model_regression_upsampled_prediction))
print("Точность", precision_score(target_valid, model_regression_upsampled_prediction))
print("F1-мера", f1_score(target_valid, model_regression_upsampled_prediction))
print("AUC-ROC", roc_auc_score(target_valid, model_regression_upsampled_prediction))
```

```
Полнота 0.7426470588235294
Точность 0.40562248995983935
F1-мера 0.5246753246753246
AUC-ROC 0.7318762932308602
```

Показатели моделей стали лучше. Самое высокое значение метрики F1 - 0.62 (а также других метрик) у модели Случайный лес, что выше целевого значение на которое мы ориентируемся при улучшении моделей. Также значение AUC-ROC у модели случанйый лес выше чем у остальных моделей, что указывает на то что данная модель ялется оптимальной для решения поставленной задачи

In [1]:
```
#используем инструмент GridSearchCv для поиска лучших параметров модели "Случайный лес"
X_train = features_train_upsampled
y_train = target_train_upsampled

clf = RandomForestClassifier()

parametrs = { 'n_estimators': range (0, 10, 1),
              #'max_depth': range (1,13, 2),
              #'min_samples_leaf': range (1,8),
              #'min_samples_split': range (2,10,2) }

grid = GridSearchCV(clf, parametrs, cv=5)
grid.fit(X_train, y_train)

grid.best_params_
```

- Обучим финальную модель

In [34]:
```
model_final = RandomForestClassifier(
     class_weight = 'balanced', max_depth= 9,  n_estimators = 9, random_state=1234)
model_final.fit(features_train_upsampled, target_train_upsampled)

model_final_predict = model_final.predict(features_valid)

print("Полнота" , recall_score(target_valid, model_final_predict))
print("Точность", precision_score(target_valid, model_final_predict))
print("F1-мера", f1_score(target_valid, model_final_predict))
print("AUC-ROC", roc_auc_score(target_valid, model_final_predict))
```

```
Полнота 0.7352941176470589
Точность 0.5671077504725898
F1-мера 0.6403415154749199
AUC-ROC 0.7957249482707657
```

- Проверяем финальную модель на адекватность

In [35]:
```
target_predict_const = pd.Series([0]*len(target_valid))
target_predict_const.value_counts()
```

Out[35]:
```
0    2000
dtype: int64
```

In [36]:
```
#Сравним показатель точности (accuracy_score) константной модели и финальной
print('accuracy_score константой модели:', accuracy_score(target_valid, target_predict_const))
print('accuracy_score финальной модели:', accuracy_score(target_valid, model_final_predict))

#Сравним AUC-ROC  - константная модель содержит только негативные ответы, поэтому важно сравнить показатель с финальной моделью
print('AUC-ROC константой модели:', roc_auc_score(target_valid, target_predict_const))
print('AUC-ROC финальной модели:', roc_auc_score(target_valid, model_final_predict))
```

```
accuracy_score константой модели: 0.796
accuracy_score финальной модели: 0.8315
AUC-ROC константой модели: 0.5
AUC-ROC финальной модели: 0.7957249482707657
```

Финальная модель показывает результаты лучше, чем константная модель — модель можно считать адекватной.

Общий вывод: В полученных данных были обнаружены существенные расхождения в количественных показателях, поэтому было применено масштабирование. В данных наблюдался значительный дисбаланс классов по целевому признаку в примерном отношении 20:80, поэтому обученные модели не проходили проверку на адекватность. Мы устранили дисбаланс методом upsampling, увеличив количество позитивного класса в 4 раза. После обучения на новых данных все модели показали лучшие показатели точности, полноты, F1, параметр ROC-AUR так же стал выше, что указывает на рост True Positive вариантов ответов. Абсолютно лучшей оказалась модель - Случайный лес со следующими метриками:

Полнота 0.5735294117647058 Точность 0.6763005780346821 F1-мера 0.6206896551724138 AUC-ROC 0.751588826485368

Далее используя инструмент GridSearchCv были подобраны оптимальные набор параметров для данной модели:

Максимальная глубина дерева - max_depth': 11, Минимальное количество листьев - 'min_samples_leaf': 1, Минимальное число образцов для сплита 'min_samples_split': 2, Число деревьев в лесу - 'n_estimators': 9

Финальная модель прошла проверку на адекватность в сравнении с константной моделью:

accuracy_score константой модели: 0.796 accuracy_score финальной модели: 0.8285 AUC-ROC константой модели: 0.5 AUC-ROC финальной модели: 0.7692321903635826

Тестирование модели

In [37]:
```
# Тестируем финальную модель на тестовой выборке
model_final_predict = model_final.predict(features_test)

print("Полнота" , recall_score(target_test, model_final_predict))
print("Точность", precision_score(target_test, model_final_predict))
print("F1-мера", f1_score(target_test, model_final_predict))
print("AUC-ROC", roc_auc_score(target_test, model_final_predict))
```

```
Полнота 0.6781326781326781
Точность 0.5277246653919694
F1-мера 0.5935483870967743
AUC-ROC 0.7615396598447446
```

Финальная модель прошла тестирование, достигла заданой метрики F1>0.59 и по другим показателям также является адекватной. Метрика AUC-ROC увеличивалась в ходе улучшения модели, что указывает на рост правильных предсказаний. Модель характеризуется высоким показателем полноты = 0.67 (min = 0, max = 1), поэтому она с высокой вероятностью предскажит уход клиента из банка. Показатель точности не высокий = 0.527 (min = 0, max = 1) — модель верно предсказывает только половину ухода клиентов. Тем не менее, полученная модель поможет лучше определять килентов, которые могут уйти из Банка в ближайшее время.

In [ ]: