Выполнил: Соковнин Игорь

Спортивный анализ данных. Платформа Kaggle

Урок 3. Построение надежных схем валидации решения, оптимизация целевых метрик

Домашнее задание:

Основное задание:

Даны выборки для обучения и для тестирования. Задание заключается в том, чтобы попробовать разные способы валидации, проанализировать плюсы / минусы каждой и сделать выводы о том, какой способ валидации наиболее устойчивый в данной задаче.

Метрика качества для оценки прогнозов - **ROC-AUC**, название целевой переменной - **IsFraud**.

Рекомендуется использовать модели градиетного бустинга, реализация любая / гипепараметры любые.

```
Внимание!
Выборка assignment_2_test.csv - наш аналог лидерборда. Будем моделировать ситуацию отправки решения на лидерборд
и сравнить значение метрики на лидерборде и на локальной валидации. Для других целей использовать выборку запрещено!.
```

Терминалогия, используемая в задании:

- ***обучающая выборка*** - выборка, которая передается в метод fit/train;
- ***валидационная выборка*** - выборка, которая получается при Hold-Out на 2 выборки (train, valid);
- ***тестовая выборка*** - выборка, которая получается при Hold-Out на 3 выборки (train, valid, test);
- ***ЛБ*** - лидерборд, выборка assignment_2_test.csv.

Ссылка на данные - https://drive.google.com/file/d/1gMEVl47pIoV1-AseB9doQ6DZNJrY3NkW/view?usp=sharing (https://drive.google.com/file/d/1gMEVl47pIoV1-AseB9doQ6DZNJrY3NkW/view?usp=sharing)

Задание 1: сделать Hold-Out валидацию с разбиением, размер которого будет адекватным, по вашему мнению. Разбиение проводить по id-транзакции (TransactionID), обучать модель градиетного бустинга любой реализации с подбором числа деревьев по early_stopping критерию до достижения сходимости. Оценить качество модели на валидационной выборке, оценить расхождение по сравнению с качеством на обучающей выборке и валидационной выборке. Оценить качество на ЛБ, сравнить с качеством на обучении и валидации. Сделать выводы.

Задание 2: сделать Hold-Out валидацию с разбиением на 3 выборки, разбиение проводить по id-транзакции (TransactionID), размер каждой выборки подобрать самостоятельно. Повторить процедуру из п.1. для каждой выборки.

Задание 3: построить доверительный интервал на данных из п.2 на основе бутстреп выборок, оценить качество модели на ЛБ относительно полученного доверительного интервала. Сделать выводы.

Задание 4: выполнить Adversarial Validation, подобрать объекты из обучающей выборки, которые сильно похожи на объекты из assignment_2_test.csv, и использовать их в качестве валидационного набора. Оценить качество модели на ЛБ, сделать выводы о полученных результатах.

Задание 5: сделать KFold / StratifiedKFold валидацию (на ваше усмотрение), оценить получаемые качество и разброс по метрике качества. Сделать выводы об устойчивости кросс-валидации, сходимости оценки на кросс-валидации и отложенном наборе данных. Оценить качество на ЛБ, сделать выводы.

Подключение библиотек и скриптов

В [1]:

```
import warnings
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

# Модель
import catboost as cb
# Метрика
from sklearn.metrics import roc_auc_score, auc
from sklearn.model_selection import KFold, StratifiedKFold, train_test_split, cross_val_sco

warnings.simplefilter("ignore")
%matplotlib inline
```

В [2]:

```
# разварачиваем выходной дисплей, чтобы увидеть больше столбцов и строк а pandas DataFrame
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

В [3]:

```
def reduce_mem_usage(df):
    '''Сокращение размера датафрейма за счёт изменения типа данных'''

    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('category')

    end_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

    return df
```

В [4]:

```
# !dir
```

В [5]:

```
# input
TRAIN_DATASET_PATH = '../../data/assignment_2_train.csv'
TEST_DATASET_PATH = '../../data/assignment_2_test.csv'
```

Загрузка данных

В [6]:

```
# Тренировочные данные
# train = pd.read_csv(TRAIN_DATASET_PATH, header = none)  # если надо скрыть названия столб
train = pd.read_csv(TRAIN_DATASET_PATH)
df_train =reduce_mem_usage(train)  # Уменьшаем размер данныхМ
df_train.head(2)
```

```
Memory usage of dataframe is 541.08 MB
Memory usage after optimization is: 262.48 MB
Decreased by 51.5%
```

Out[6]:

	TransactionID	isFraud	TransactionDT	TransactionAmt	ProductCD	card1	card2	card3	
0	2987000	0	86400	68.5	W	13926	NaN	150.0	
1	2987001	0	86401	29.0	W	2755	404.0	150.0	ma

В [7]:

```
# Тестовые данные
leaderboard = pd.read_csv(TEST_DATASET_PATH)
df_leaderboard =reduce_mem_usage(leaderboard)  # Уменьшаем размер данных

df_leaderboard.head(2)
```

```
Memory usage of dataframe is 300.60 MB
Memory usage after optimization is: 145.83 MB
Decreased by 51.5%
```

Out[7]:

	TransactionID	isFraud	TransactionDT	TransactionAmt	ProductCD	card1	card2	card3	ca
0	3287000	1	7415038	226.0	W	12473	555.0	150.0	\
1	3287001	0	7415054	3072.0	W	15651	417.0	150.0	\

Числовых признаки

В [118]:

```
numerical_features = df_train.select_dtypes(exclude=["category"])
numerical_features = numerical_features.columns.tolist()
```

Обрабатка категориальные признаков

В [9]:

```
catigorical_features = [
'ProductCD',  # 180000 non-null  category
'card4',  # 179992 non-null  category
'card6',  # 179993 non-null  category
'P_emaildomain',  # 151560 non-null  category
'R_emaildomain',  # 60300 non-null   category
'M1',  # 61749 non-null   category
'M2',  # 61749 non-null   category
'M3',  # 61749 non-null   category
'M4',  # 83276 non-null   category
'M5',  # 61703 non-null   category
'M6',  # 105652 non-null  category
'M7',  # 31652 non-null   category
'M8',  # 31652 non-null   category
'M9'  # 31652 non-null   category
]
```

Подготовка тренировочных данных

В [11]:

```
data = []
data = df_train[numerical_features + catigorical_features]

# заполняем пропуски в категориалиных признаках
for feature in catigorical_features:
    data[feature] = data[feature].cat.add_categories('Unknown')
    data[feature].fillna('Unknown', inplace =True)

# Каждой категории сопоставляет целое число (номер категории) - https://dyakonov.org/2016/0
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
for cat_colname in data[catigorical_features].columns:
    le.fit(data[cat_colname])
    data[cat_colname+'_le'] = le.transform(data[cat_colname])

# target = df_train["isFraud"]
```

В [12]:

```
df_train_new = data
df_train_new = df_train_new.drop(catigorical_features, axis=1)
# df_train_new.columns
```

В [13]:

```
# df_train_new.head(2)
```

Подготовка ЛБ (лидерборд) данных

В [14]:

```
data = []
data = df_leaderboard[numerical_features + catigorical_features]

# заполняем пропуски в категориалиных признаках
for feature in catigorical_features:
    data[feature] = data[feature].cat.add_categories('Unknown')
    data[feature].fillna('Unknown', inplace =True)

# Каждой категории сопоставляет целое число (номер категории) - https://dyakonov.org/2016/0
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
for cat_colname in data[catigorical_features].columns:
    le.fit(data[cat_colname])
    data[cat_colname+'_le'] = le.transform(data[cat_colname])

y_leaderboard = df_leaderboard["isFraud"]
```

В [15]:

```
x_leaderboard = data
x_leaderboard = x_leaderboard.drop(catigorical_features, axis=1)

x_leaderboard = x_leaderboard.drop(["TransactionID", "isFraud"], axis=1)
```

В [16]:

```
catigorical_features_name = ['ProductCD_le', 'card4_le', 'card6_le', 'R_emaildomain_le',
                      'M1_le', 'M2_le', 'M3_le', 'M4_le', 'M5_le', 'M6_le', 'M7_le', 'M8_le
```

Задание 1:

Сделать **Hold-Out** валидацию с разбиением, размер которого будет адекватным, по вашему мнению. Разбиение проводить по **id**-транзакции (**TransactionID**), обучать модель градиетного бустинга любой реализации с подбором числа деревьев по **early_stopping** критерию до достижения сходимости. Оценить качество модели на валидационной выборке, оценить расхождение по сравнению с качеством на обучающей выборке и валидационной выборке. Оценить качество на ЛБ, сравнить с качеством на обучении и валидации. Сделать выводы.

Hold-Out разбиение

В [17]:

```
x_train, x_test = train_test_split(
    df_train_new.drop(["TransactionID", "isFraud"], axis=1), train_size=0.70, shuffle=True,
)

y_train, y_test = train_test_split(
    df_train_new["isFraud"], train_size=0.70, shuffle=True, random_state=1,
)
```

B [18]:

```
x_train.head(2)
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 125999 entries, 59549 to 128037
Columns: 392 entries, TransactionDT to M9_le
dtypes: float32(376), int16(1), int32(15)
memory usage: 189.1 MB
```

B [19]:

```
x_test.head(2)
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 125999 entries, 59549 to 128037
Columns: 392 entries, TransactionDT to M9_le
dtypes: float32(376), int16(1), int32(15)
memory usage: 189.1 MB
```

B [20]:

```
print("x_train.shape = {} rows, {} cols".format(*x_train.shape))
print("x_test.shape = {} rows, {} cols".format(*x_test.shape))
```

```
x_train.shape = 125999 rows, 392 cols
x_test.shape = 54001 rows, 392 cols
```

B [21]:

```
print(y_train.shape)
print(y_test.shape)
```

```
(125999,)
(54001,)
```

XGBoost

B [22]:

```
import xgboost as xgb
from sklearn.metrics import r2_score

#model = xgb.XGBRegressor(random_state=1)
#model.fit(x_train, y_train)
```

B [23]:

```
#train_score = r2_score(y_train, model.predict(x_train))
#test_score = r2_score(y_test, model.predict(x_test))

#print(f"Train-score: {round(train_score, 3)}, Test-score: {round(test_score, 3)}")
```

LightGBM Sklearn-API

В [24]:

```
import lightgbm as lgb

#from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
```

В [25]:

```
"""params = {
    "boosting_type": "gbdt",  # gradient boosting tree decision tree (бустинг над решающими
    "objective": "binary",
    #"metric": "auc", # метрика качества - ROC AUC
    "metric":"None",
    "learning_rate": 0.01,  # скорсть обучения
    "n_estimators": 10000,  # число деревьев
    # регуляризация
    "n_jobs": 6,
    "seed": 27
}"""
```

Out[25]:

```
'params = {\n    "boosting_type": "gbdt",  # gradient boosting tree decision
tree (бустинг над решающими деревьями)\n    "objective": "binary",\n    #"me
tric": "auc", # метрика качества - ROC AUC\n    "metric":"None",\n    "learn
ing_rate": 0.01,  # скорсть обучения\n    "n_estimators": 10000,  # число де
ревьев\n    # регуляризация\n    "n_jobs": 6,\n    "seed": 27\n}'
```

В [57]:

```
# Задача бинарной классификации
params = {
    "boosting_type": "gbdt",  # gradient boosting tree decision tree (бустинг над решающими
    "objective": "binary",
    "metric": "auc", # метрика качества - ROC AUC
    "learning_rate": 0.01,  # скорсть обучения
    "n_estimators": 20000,  # число деревьев
     # регуляризация
    "reg_lambda": 100,  # регуляризация (то что используется при F2-штрафе (1:15:10))
    "max_depth": 4,  # глубина дерева
    #"gamma": 10,  # min-е улучшение функции потерь при котором мы будем делать разбиени (1
    #"nthread": 6, # число ядер
    "n_jobs": 6,
    "seed": 27
}
```

В [27]:

```
# Оценить качество модели на валидационной выборке, оценить расхождение
# по сравнению с качеством на обучающей выборке и валидационной выборке.
model = lgb.LGBMClassifier(**params)
model.fit(
    X=x_train,
    y=y_train,
    eval_set=[(x_train, y_train), (x_test, y_test)],
    categorical_feature = catigorical_features_name,
    early_stopping_rounds=25,
    eval_metric="auc",
    verbose=500
)
```

```
[LightGBM] [Warning] Unknown parameter: gamma
Training until validation scores don't improve for 25 rounds
[500]   training's auc: 0.90193 valid_1's auc: 0.892426
[1000]  training's auc: 0.915824        valid_1's auc: 0.902881
[1500]  training's auc: 0.923122        valid_1's auc: 0.908277
[2000]  training's auc: 0.928314        valid_1's auc: 0.912221
[2500]  training's auc: 0.93248 valid_1's auc: 0.915232
[3000]  training's auc: 0.936058        valid_1's auc: 0.917712
[3500]  training's auc: 0.938812        valid_1's auc: 0.919346
[4000]  training's auc: 0.941278        valid_1's auc: 0.920767
[4500]  training's auc: 0.943461        valid_1's auc: 0.922273
[5000]  training's auc: 0.945636        valid_1's auc: 0.923682
[5500]  training's auc: 0.947323        valid_1's auc: 0.924723
[6000]  training's auc: 0.949253        valid_1's auc: 0.925867
[6500]  training's auc: 0.950772        valid_1's auc: 0.926758
[7000]  training's auc: 0.952319        valid_1's auc: 0.927659
[7500]  training's auc: 0.954209        valid_1's auc: 0.928775
[8000]  training's auc: 0.955837        valid_1's auc: 0.929767
[8500]  training's auc: 0.957305        valid_1's auc: 0.930679
[9000]  training's auc: 0.958314        valid_1's auc: 0.931195
[9500]  training's auc: 0.959449        valid_1's auc: 0.931993
[10000] training's auc: 0.960552        valid_1's auc: 0.932614
[10500] training's auc: 0.961406        valid_1's auc: 0.933022
Early stopping, best iteration is:
[10713] training's auc: 0.96171 valid_1's auc: 0.933166
```

Out[27]:

```
LGBMClassifier(gamma=10, learning_rate=0.01, max_depth=4, metric='auc',
               n_estimators=20000, n_jobs=6, objective='binary', reg_lambda=
100,
               seed=27)
```

В [28]:

```
#x_train.head(2)
```

В [29]:

```
#y_train
```

Оценка качества модели

B [30]:

```
train_score = roc_auc_score(y_train, model.predict(x_train))
test_score = roc_auc_score(y_test, model.predict(x_test))
leaderboard_score = roc_auc_score(y_leaderboard, model.predict(x_leaderboard))

print(f'train_score={train_score}')
print(f'test_score={test_score}')
print(f'leaderboard_score={leaderboard_score}')
```

```
train_score=0.7688092681395586
test_score=0.737555057193122
leaderboard_score=0.6156232360519123
```

Результат:

```
train_size = 0.70 (разбиение данных на 70% (train) и 30% (test))

train_score=0.7688092681395586 - Качество на обучении
test_score=0.737555057193122 - Качество на тесте
leaderboard_score=0.6156232360519123 - Качество на ЛБ
```

Вывод:

Разница Δ между обучением (train_score) и валидационной выборкой (test_score):

- $\Delta_{0.7}$ **= train_score - test_score = 0.7688092681395586 - 0.737555057193122 = 0.031254210946436656 ~ 0.03**

Разница Δ между обучением (train_score) и ЛБ (leaderboard_score):

- $\Delta_{0.7}$ **= train_score - leaderboard_score = 0.737555057193122 - 0.6156232360519123 = 0.1219318211412097 ~ 0.12**

Δ между обучением и валидационной выборкой, $\Delta_{0.7}$ = 0.031254210946436656 ~ 3% (< 5%) от значения ROC AUC тренировочной выборки.

Δ между обучением и ЛБ, $\Delta_{0.7}$ = 0.1219318211412097 ~ 12% (> 10%) от значения ROC AUC тренировочной выборки.

Следовательно:

- *распределения на обучающей выборке и валидационной выборкой близки друг другу - модель обучена хорошо*;
- *распределения на обучающей выборке и ЛБ значительно отличаются между собой*.

B [31]:

```
0.7688092681395586 - 0.737555057193122
0.737555057193122 - 0.6156232360519123
```

Out[31]:

```
0.1219318211412097
```

Задание 2:

Сделать Hold-Out валидацию с разбиением на 3 выборки, разбиение проводить по id-транзакции (TransactionID), размер каждой выборки подобрать самостоятельно. Повторить процедуру из п.1. для каждой выборки.

B [32]:

```
x_train, x_valid = train_test_split(
    df_train_new.drop(["TransactionID", "isFraud"], axis=1), train_size=0.70, shuffle=True,
)
y_train, y_valid = train_test_split(
    df_train_new["isFraud"], train_size=0.70, shuffle=True, random_state=1,
)

x_valid, x_test = train_test_split(
    x_valid, train_size=0.7, shuffle=True, random_state=27
)
y_valid, y_test = train_test_split(
    y_valid, train_size=0.7, shuffle=True, random_state=27
)

print("x_train.shape = {} rows, {} cols".format(*x_train.shape))
print("x_valid.shape = {} rows, {} cols".format(*x_valid.shape))
print("x_test.shape = {} rows, {} cols".format(*x_test.shape))
```

```
x_train.shape = 125999 rows, 392 cols
x_valid.shape = 37800 rows, 392 cols
x_test.shape = 16201 rows, 392 cols
```

B [33]:

```
# Оценить качество модели на валидационной выборке, оценить расхождение
# по сравнению с качеством на обучающей выборке и валидационной выборке.
model_1 = lgb.LGBMClassifier(**params)
model_1.fit(
    X=x_train,  # используем обработанный датафрэйм из предыдущего занятия
    y=y_train,
    eval_set=[(x_train, y_train), (x_valid, y_valid)],
    categorical_feature = catigorical_features_name,
    early_stopping_rounds=25,
    eval_metric="auc",
    verbose=500
)
```

```
[LightGBM] [Warning] Unknown parameter: gamma
Training until validation scores don't improve for 25 rounds
[500]   training's auc: 0.90193 valid_1's auc: 0.896502
[1000]  training's auc: 0.915824        valid_1's auc: 0.906843
[1500]  training's auc: 0.923122        valid_1's auc: 0.911853
[2000]  training's auc: 0.928314        valid_1's auc: 0.915473
[2500]  training's auc: 0.93248 valid_1's auc: 0.918239
[3000]  training's auc: 0.936058        valid_1's auc: 0.920355
[3500]  training's auc: 0.938812        valid_1's auc: 0.921745
[4000]  training's auc: 0.941278        valid_1's auc: 0.923011
[4500]  training's auc: 0.943461        valid_1's auc: 0.924294
[5000]  training's auc: 0.945636        valid_1's auc: 0.925551
[5500]  training's auc: 0.947323        valid_1's auc: 0.926388
[6000]  training's auc: 0.949253        valid_1's auc: 0.927516
[6500]  training's auc: 0.950772        valid_1's auc: 0.928302
[7000]  training's auc: 0.952319        valid_1's auc: 0.929226
[7500]  training's auc: 0.954209        valid_1's auc: 0.930246
[8000]  training's auc: 0.955837        valid_1's auc: 0.931039
[8500]  training's auc: 0.957305        valid_1's auc: 0.931808
Early stopping, best iteration is:
[8660]  training's auc: 0.957642        valid_1's auc: 0.931965
```

Out[33]:

```
LGBMClassifier(gamma=10, learning_rate=0.01, max_depth=4, metric='auc',
               n_estimators=20000, n_jobs=6, objective='binary', reg_lambda=
100,
               seed=27)
```

Оценка качества модели

In [34]:

```
train_score = roc_auc_score(y_train, model_1.predict(x_train))
valid_score = roc_auc_score(y_valid, model_1.predict(x_valid))
test_score = roc_auc_score(y_test, model_1.predict(x_test))
leaderboard_score = roc_auc_score(y_leaderboard, model_1.predict(x_leaderboard))

print(f'train_score={train_score}')
print(f'valid_score={valid_score}')
print(f'test_score={test_score}')
print(f'leaderboard_score={leaderboard_score}')
```

```
train_score=0.759251168407984
valid_score=0.7330467583957665
test_score=0.7316095612084162
leaderboard_score=0.6161438007400407
```

Результат:

```
train_size = 0.70 (разбиение данных на 70% (train), 20% (valid) и 10% (test))

train_score=0.759251168407984 - Качество на обучении
valid_score=0.7330467583957665 - Качество на валидации
test_score=0.7316095612084162 - Качество на тесте
leaderboard_score=0.6161438007400407 - Качество на ЛБ
```

Вывод:

Разница Δ между обучением (train_score) и валидационной выборкой (test_score):

- $\Delta_{0.7}$ **= train_score - test_score** = 0.759251168407984 - 0.7330467583957665 = 0.026204410012217516 ~ 0.03

Разница Δ между обучением (train_score) и тестом (test_score):

- $\Delta_{0.7}$ **= train_score - leaderboard_score** = 0.759251168407984 - 0.7316095612084162 = 0.027641607199567764 ~ 0.03

Разница Δ между обучением (train_score) и ЛБ (leaderboard_score):

- $\Delta_{0.7}$ **= train_score - leaderboard_score** = 0.759251168407984 - 0.6161438007400407 = 0.1431073676679433 ~ 0.143

Δ между train и valid: $\Delta_{0.7}$ = 0.026204410012217516, что ~ 3% (< 10%) от значения ROC AUC тренировочной выборки.

Δ между train и test: $\Delta_{0.7}$ = 0.027641607199567764, что ~ 3% (< 10%) от значения ROC AUC тренировочной выборки.

Δ между train и ЛБ: $\Delta_{0.7}$ = 0.1431073676679433, что ~ 14.3% (> 10%) от значения ROC AUC тренировочной выборки.

B [35]:

```
0.759251168407984 - 0.7330467583957665
0.759251168407984 - 0.7316095612084162
0.759251168407984 - 0.6161438007400407
```

Out[35]:

```
0.1431073676679433
```

Задание 3:

Построить доверительный интервал на данных из п.2 на основе бутстреп выборок, оценить качество модели на ЛБ относительно полученного доверительного интервала. Сделать выводы.

Bootstrap

B [36]:

```
from tqdm import tqdm
from typing import List, Tuple

#from sklearn.model_selection import KFold, StratifiedKFold, train_test_split, cross_val_sc
```

В [37]:

```
def create_bootstrap_samples(data: np.array, n_samples: int = 10000) -> np.array:
    """
    Создание бутстреп-выборок.

    Parameters
    ----------
    data: np.array
        Исходная выборка, которая будет использоваться для
        создания бутстреп выборок.

    n_samples: int, optional, default = 10000
        Количество создаваемых бутстреп выборок.
        Опциональный параметр, по умолчанию, равен 10000.

    Returns
    -------
    bootstrap_idx: np.array
        Матрица индексов, для создания бутстреп выборок.

    """
    bootstrap_idx = np.random.randint(
        low=0, high=len(data), size=(n_samples, len(data))
    )
    return bootstrap_idx


def create_bootstrap_metrics(y_true: np.array,
                             y_pred: np.array,
                             metric: callable,
                             n_samlpes: int = 10000) -> List[float]:
    """
    Вычисление бутстреп оценок.

    Parameters
    ----------
    y_true: np.array
        Вектор целевой переменной.

    y_pred: np.array
        Вектор прогнозов.

    metric: callable
        Функция для вычисления метрики.
        Функция должна принимать 2 аргумента: y_true, y_pred.

    n_samples: int, optional, default = 10000
        Количество создаваемых бутстреп выборок.
        Опциональный параметр, по умолчанию, равен 10000.

    Returns
    -------
    bootstrap_metrics: List[float]
        Список со значениями метрики качества на каждой бустреп выборке.
        (оценки метрика качества на каждой бутстреп выборке)

    """
    scores = []

    if isinstance(y_true, pd.Series):
```

```
        y_true = y_true.values

    bootstrap_idx = create_bootstrap_samples(y_true)  # генерим ндексы
    for idx in bootstrap_idx:
        y_true_bootstrap = y_true[idx]
        y_pred_bootstrap = y_pred[idx]

        score = metric(y_true_bootstrap, y_pred_bootstrap)  # считаем метрику
        scores.append(score)

    return scores


def calculate_confidence_interval(scores: list, conf_interval: float = 0.95) -> Tuple[float
    """
    Вычисление доверительного интервала.

    Parameters
    ----------
    scores: List[float / int]
        Список с оценками изучаемой величины.

    conf_interval: float, optional, default = 0.95
        Уровень доверия для построения интервала.
        Опциональный параметр, по умолчанию, равен 0.95.

    Returns
    -------
    conf_interval: Tuple[float]
        Кортеж с границами доверительного интервала.

    """
    left_bound = np.percentile(
        scores, ((1 - conf_interval) / 2) * 100
    )
    right_bound = np.percentile(
        scores, (conf_interval + ((1 - conf_interval) / 2)) * 100
    )

    return left_bound, right_bound


def plot_validation(scores: list):
    fig, axes = plt.subplots(1, 2, figsize=(15, 5))
    plt.suptitle("Bootstrap for evaluating validation stability", size=15)
    axes[1].scatter(range(len(scores)), scores, alpha=0.25, color="blue")
    axes[1].set_xlabel("sampel number", size=15)
    axes[1].set_ylabel("ROC AUC score", size=15)

    sns.distplot(scores, ax=axes[0], color="green", bins=20)
    axes[0].set_xlabel("sampel number", size=15)
```

B [38]:

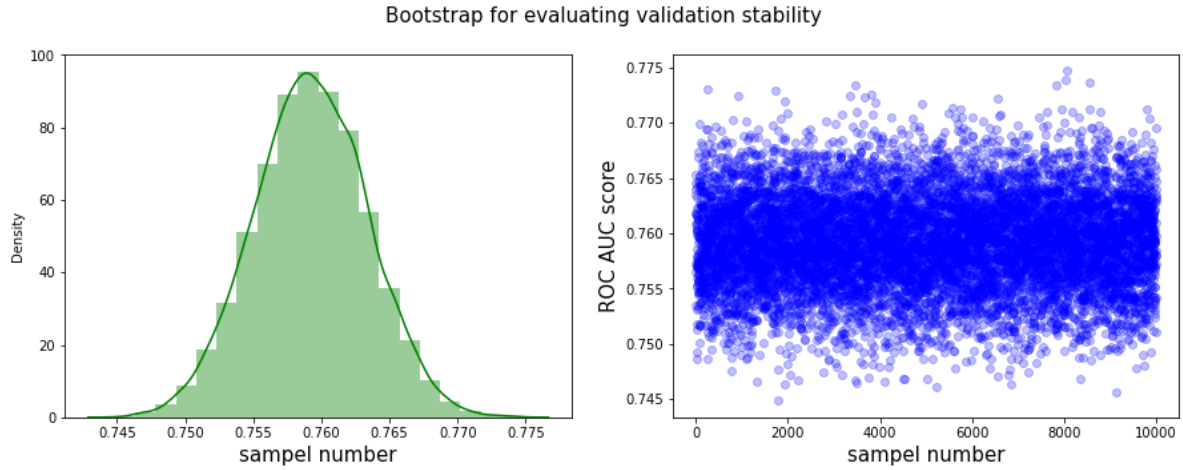
```
from sklearn.metrics import roc_auc_score

#x_train,x_valid, x_test, x_leaderboard
np.random.seed(27)
scores = create_bootstrap_metrics(y_train, model_1.predict(x_train), roc_auc_score)

calculate_confidence_interval(scores)
```

Out[38]:

```
(0.7510943481267962, 0.7673024788466506)
```

B [39]:

```
plot_validation(scores)
```



B [40]:

```
#scores = create_bootstrap_metrics(y_valid, model_1.predict(x_valid), roc_auc_score)
#calculate_confidence_interval(scores)
#plot_validation(scores)
```

B [41]:

```
#scores = create_bootstrap_metrics(y_test, model_1.predict(x_test), roc_auc_score)
#calculate_confidence_interval(scores)
#plot_validation(scores)
```

B [42]:

```
#scores = create_bootstrap_metrics(y_leaderboard, model_1.predict(x_leaderboard), roc_auc_s
#calculate_confidence_interval(scores)
#plot_validation(scores)
```

Вывод:

Разбиение данных на 70% (train), 20% (valid) и 10% (test):

- train_size = 0.70

Доверительный интервал для train, на основе бутстрэп выборок:

- (0.7510943481267962, 0.7673024788466506)

Качество на ЛБ:

- leaderboard_score=0.6161438007400407

leaderboard_score лежит за пределами доверительного интервала.

Задание 4:

Выполнить Adversarial Validation (Состязательная проверка), подобрать объекты из обучающей выборки, которые сильно похожи на объекты из assignment_2_test.csv, и использовать их в качестве валидационного набора. Оценить качество модели на ЛБ, сделать выводы о полученных результатах.

B [166]:

```
from sklearn.metrics import roc_auc_score
```

B [167]:

```
df_train_new.head(2)
df_train_adv = df_train_new.drop(["TransactionID", "isFraud"], axis=1)
```

B [168]:

```
print(df_train_adv.shape)
df_train_adv.head(2)
```

```
(180000, 392)
```

Out[168]:

	TransactionDT	TransactionAmt	card1	card2	card3	card5	addr1	addr2	dist1	dist2	C1
0	86400	68.5	13926	NaN	150.0	142.0	315.0	87.0	19.0	NaN	1.0
1	86401	29.0	2755	404.0	150.0	102.0	325.0	87.0	NaN	NaN	1.0

B [169]:

```
print(x_leaderboard.shape)
x_leaderboard.head(2)
```

```
(100001, 392)
```

Out[169]:

	TransactionDT	TransactionAmt	card1	card2	card3	card5	addr1	addr2	dist1	dist2	C1
0	7415038	226.0	12473	555.0	150.0	226.0	299.0	87.0	116.0	NaN	2.0
1	7415054	3072.0	15651	417.0	150.0	226.0	330.0	87.0	NaN	NaN	1.0

B [195]:

```
# Объединяем train (assignment_2_train.csv) и leaderboard (assignment_2_test.csv) выборки
x_adv = pd.concat([
    df_train_adv, x_leaderboard], axis=0
)
# Объявляем target = 0, если объект из leaderboard-а и target=1 если объект из train
y_adv = np.hstack((np.zeros(df_train_adv.shape[0]), np.ones(x_leaderboard.shape[0])))
assert x_adv.shape[0] == y_adv.shape[0]  # проверять предположение о значениях данных
print(x_adv.shape[0])
print(y_adv.shape[0])
```

```
280001
280001
```

B [196]:

```
y_adv
```

Out[196]:

```
array([0., 0., 0., ..., 1., 1., 1.])
```

B [197]:

```
print(x_adv.shape)
x_adv.head(2)
```

```
(280001, 392)
```

Out[197]:

	TransactionDT	TransactionAmt	card1	card2	card3	card5	addr1	addr2	dist1	dist2	C1
0	86400	68.5	13926	NaN	150.0	142.0	315.0	87.0	19.0	NaN	1.0
1	86401	29.0	2755	404.0	150.0	102.0	325.0	87.0	NaN	NaN	1.0

В [199]:

```
if 'TransactionID' in numerical_features:
    numerical_features.remove('TransactionID')
if 'isFraud' in numerical_features:
    numerical_features.remove('isFraud')
# Обучаем любой классификатор, обязательно с метрикой roc_auc
model_xgb = xgb.XGBClassifier(n_estimators=25)
model_xgb.fit(x_adv[numerical_features], y_adv)
```

```
[18:23:30] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_
1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evaluation
metric used with the objective 'binary:logistic' was changed from 'error' to
'logloss'. Explicitly set eval_metric if you'd like to restore the old behav
ior.
```

Out[199]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=25, n_jobs=8, num_parallel_tree=1, random_state=
0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

В [200]:

```
# Обучаем любой классификатор, обязательно с метрикой roc_auc
y_pred_adv = model_xgb.predict_proba(x_adv[numerical_features])
score = roc_auc_score(y_adv, y_pred_adv[:, 1])
print(round(score, 4))
```

```
1.0
```

Модель идеально разделяет данные на две выборки, т. е. данные test и train (leaderbord) не похожи др. на друга?

Записей похожих на объекты из assignment_2_test.csv нет

В [202]:

```
# Мы можем сделать прогнозы на обучающую часть на исходную выборку x_train
# и от этой выборки отобрать наблюдения с большой вероятностью
y_pred = model_xgb.predict_proba(df_train_adv[numerical_features])
y_pred
```

Out[202]:

```
array([[9.997488e-01, 2.511662e-04],
       [9.997488e-01, 2.511662e-04],
       [9.997488e-01, 2.511662e-04],
       ...,
       [9.997488e-01, 2.511662e-04],
       [9.997488e-01, 2.511662e-04],
       [9.997488e-01, 2.511662e-04]], dtype=float32)
```

В [205]:

```
# Отбираем наблюдения с большой вероятностью
# Из общей выборки выбираем наблюдения которые очень сильно похожи на тестовую выборку
# и их использования для теста, то что осталось для крос-валидации
# тогда у нас осталась бы честная оценка.
# С одной стороны у нас крос-валидация на объектах train-а, с другой стороны у нас Hold-Out
# еа тех объектах которые очень сильно похожи на test.
pd.cut(
    y_pred[:, 1], bins=np.arange(0, 1.01, 0.1)
).value_counts().sort_index()
```

Out[205]:

```
(0.0, 0.1]    180000
(0.1, 0.2]         0
(0.2, 0.3]         0
(0.3, 0.4]         0
(0.4, 0.5]         0
(0.5, 0.6]         0
(0.6, 0.7]         0
(0.7, 0.8]         0
(0.8, 0.9]         0
(0.9, 1.0]         0
dtype: int64
```

В [ ]:

```

```

В [228]:

```
if 'TransactionID' in numerical_features:
    numerical_features.remove('TransactionID')
if 'isFraud' in numerical_features:
    numerical_features.remove('isFraud')

x_train_adv, x_test_adv = train_test_split(
    x_adv[numerical_features], train_size=0.70, shuffle=True, random_state=1,
)

y_train_adv, y_test_adv = train_test_split(
    y_adv, train_size=0.70, shuffle=True, random_state=1,
)
```

В [229]:

```
# Обучаем любой классификатор, обязательно с метрикой roc_auc
model_lgb = lgb.LGBMClassifier(**params)
model_lgb.fit(
    X=x_train_adv[numerical_features],
    y=y_train_adv,
    eval_set=[(x_train_adv[numerical_features], y_train_adv), (x_test_adv[numerical_feature
    #categorical_feature = catigorical_features_name,
    early_stopping_rounds=25,
    eval_metric="auc",
    verbose=500
)
```

```
Training until validation scores don't improve for 25 rounds
Early stopping, best iteration is:
[1]     valid_0's auc: 0.999813 valid_1's auc: 0.999787
```

Out[229]:

```
LGBMClassifier(learning_rate=0.01, max_depth=4, metric='auc',
               n_estimators=20000, n_jobs=6, objective='binary', reg_lambda=
100,
               seed=27)
```

В [230]:

```
# Обучаем любой классификатор, обязательно с метрикой roc_auc
y_pred_adv = model_lgb.predict_proba(x_adv[numerical_features])
score = roc_auc_score(y_adv, y_pred_adv[:, 1])
print(round(score, 4))
```

```
0.9998
```

В [231]:

```
# Мы можем сделать прогнозы на обучающую часть на исходную выборку x_train
# и от этой выборки отобрать наблюдения с большой вероятностью
y_pred = model_lgb.predict_proba(df_train_new[numerical_features])
y_pred
```

Out[231]:

```
array([[0.64581735, 0.35418265],
       [0.64581735, 0.35418265],
       [0.64581735, 0.35418265],
       ...,
       [0.63585899, 0.36414101],
       [0.63585899, 0.36414101],
       [0.63585899, 0.36414101]])
```

In [232]:

```
# Отбираем наблюдения с большой вероятностью
# Из общей выборки выбираем наблюдения которые очень сильно похожи на тестовую выборку
# и их использования для теста, то что осталось для крос-валидации
# тогда у нас осталась бы честная оценка.
# С одной стороны у нас крос-валидация на объектах train-а, с другой стороны у нас Hold-Out
# еа тех объектах которые очень сильно похожи на test.
pd.cut(
    y_pred[:, 1], bins=np.arange(0, 1.01, 0.1)
).value_counts().sort_index()
```

Out[232]:

```
(0.0, 0.1]         0
(0.1, 0.2]         0
(0.2, 0.3]         0
(0.3, 0.4]    180000
(0.4, 0.5]         0
(0.5, 0.6]         0
(0.6, 0.7]         0
(0.7, 0.8]         0
(0.8, 0.9]         0
(0.9, 1.0]         0
dtype: int64
```

Задание 5:

Сделать KFold / StratifiedKFold валидацию (на ваше усмотрение), оценить получаемые качество и разброс по метрике качества. Сделать выводы об устойчивости кросс-валидации, сходимости оценки на кросс-валидации и отложенном наборе данных. Оценить качество на ЛБ, сделать выводы.

In [234]:

```
df_train_new.head(2)
```

Out[234]:

	TransactionID	isFraud	TransactionDT	TransactionAmt	card1	card2	card3	card5	addr1
0	2987000	0	86400	68.5	13926	NaN	150.0	142.0	315.0
1	2987001	0	86401	29.0	2755	404.0	150.0	102.0	325.0

В [235]:

```
data = []
data = df_train_new
x_test=df_train_new.drop(["TransactionID", "isFraud"], axis=1)
y_test=df_train_new["isFraud"]

cv = cross_val_score(
    estimator=model,
    #X=data[numerical_features],
    X=x_test,
    y=y_test,
    scoring="roc_auc",
    cv=5  # число фолдов
)

print(f"CV-results: {round(np.mean(cv), 4)} +/- {round(np.std(cv), 3)}")
```

```
CV-results: 0.8447 +/- 0.082
```

В [236]:

```
def make_cross_validation(X: pd.DataFrame,
                          y: pd.Series,
                          estimator: object,
                          metric: callable,
                          cv_strategy):
    """
    Кросс-валидация.

    Parameters
    ----------
    X: pd.DataFrame
        Матрица признаков.

    y: pd.Series
        Вектор целевой переменной.

    estimator: callable
        Объект модели для обучения.

    metric: callable
        Метрика для оценки качества решения.
        Ожидается, что на вход будет передана функция,
        которая принимает 2 аргумента: y_true, y_pred.

    cv_strategy: cross-validation generator
        Объект для описания стратегии кросс-валидации.
        Ожидается, что на вход будет передан объект типа
        KFold или StratifiedKFold.

    Returns
    -------
    oof_score: float
        Значение метрики качества на OOF-прогнозах (outo fold прогнозы).

    fold_train_scores: List[float]
        Значение метрики качества на каждом обучающем датасете кросс-валидации.

    fold_valid_scores: List[float]
        Значение метрики качества на каждом валидационном датасете кросс-валидации.

    oof_predictions: np.array
        Прогнозы на OOF.

    """
    estimators, fold_train_scores, fold_valid_scores = [], [], []
    oof_predictions = np.zeros(X.shape[0])

    for fold_number, (train_idx, valid_idx) in enumerate(cv_strategy.split(X, y)):
        x_train, x_valid = X.loc[train_idx], X.loc[valid_idx]
        y_train, y_valid = y.loc[train_idx], y.loc[valid_idx]

        estimator.fit(x_train, y_train)
        y_train_pred = estimator.predict(x_train)
        y_valid_pred = estimator.predict(x_valid)

        fold_train_scores.append(metric(y_train, y_train_pred))
        fold_valid_scores.append(metric(y_valid, y_valid_pred))
        oof_predictions[valid_idx] = y_valid_pred
```

```
        msg = (
            f"Fold: {fold_number+1}, train-observations = {len(train_idx)}, "
            f"valid-observations = {len(valid_idx)}\n"
            f"train-score = {round(fold_train_scores[fold_number], 4)}, "
            f"valid-score = {round(fold_valid_scores[fold_number], 4)}"
        )
        print(msg)
        print("="*69)
        estimators.append(estimator)

    oof_score = metric(y, oof_predictions)
    print(f"CV-results train: {round(np.mean(fold_train_scores), 4)} +/- {round(np.std(fold
    print(f"CV-results valid: {round(np.mean(fold_valid_scores), 4)} +/- {round(np.std(fold
    print(f"OOF-score = {round(oof_score, 4)}")

    return estimators, oof_score, fold_train_scores, fold_valid_scores, oof_predictions
```

B [242]:

```
from sklearn.model_selection import KFold
```

B [254]:

```
#cv_strategy = KFold(n_splits=5, random_state=1)
cv_strategy = KFold(n_splits=5, random_state=42, shuffle=True)
```

B [247]:

```
#estimators, oof_score, fold_train_scores, fold_valid_scores, oof_predictions = make_cross_
#    data[numerical_features], data["y"], model, metric=r2_score, cv_strategy=cv_strategy)

estimators, oof_score, fold_train_scores, fold_valid_scores, oof_predictions = make_cross_v
    x_test, y_test, model, metric=roc_auc_score, cv_strategy=cv_strategy
)
```

```
Fold: 1, train-observations = 144000, valid-observations = 36000
train-score = 0.7998, valid-score = 0.7446
=====================================================================
Fold: 2, train-observations = 144000, valid-observations = 36000
train-score = 0.799, valid-score = 0.7618
=====================================================================
Fold: 3, train-observations = 144000, valid-observations = 36000
train-score = 0.7985, valid-score = 0.753
=====================================================================
Fold: 4, train-observations = 144000, valid-observations = 36000
train-score = 0.8005, valid-score = 0.7517
=====================================================================
Fold: 5, train-observations = 144000, valid-observations = 36000
train-score = 0.7994, valid-score = 0.7565
=====================================================================
CV-results train: 0.7994 +/- 0.001
CV-results valid: 0.7535 +/- 0.006
OOF-score = 0.7536
```

В [248]:

```
estimators
```

Out[248]:

```
[LGBMClassifier(learning_rate=0.01, max_depth=4, metric='auc',
                n_estimators=20000, n_jobs=6, objective='binary', reg_lambda
=100,
                seed=27),
 LGBMClassifier(learning_rate=0.01, max_depth=4, metric='auc',
                n_estimators=20000, n_jobs=6, objective='binary', reg_lambda
=100,
                seed=27),
 LGBMClassifier(learning_rate=0.01, max_depth=4, metric='auc',
                n_estimators=20000, n_jobs=6, objective='binary', reg_lambda
=100,
                seed=27),
 LGBMClassifier(learning_rate=0.01, max_depth=4, metric='auc',
                n_estimators=20000, n_jobs=6, objective='binary', reg_lambda
=100,
                seed=27),
 LGBMClassifier(learning_rate=0.01, max_depth=4, metric='auc',
                n_estimators=20000, n_jobs=6, objective='binary', reg_lambda
=100,
                seed=27)]
```

В [249]:

```
oof_score
```

Out[249]:

```
0.7536473672795632
```

В [250]:

```
fold_train_scores
```

Out[250]:

```
[0.7998046639171603,
 0.7990066316532937,
 0.7984615016507928,
 0.8004525389375149,
 0.7994139279258937]
```

В [251]:

```
fold_valid_scores
```

Out[251]:

```
[0.7445990296803654,
 0.761833065681327,
 0.7530236150637785,
 0.751668088369442,
 0.7564774518252644]
```

B [252]:

```
oof_predictions
```

Out[252]:

```
array([0., 0., 0., ..., 0., 0., 0.])
```

B [ ]:

```

```