Скрипт для подготовки датафрейма

Из-за загруженности было не очень много времени на обработку данных, поэтому взял только данные соответствующие по 'id', 'buy_time', т.е признаки. Считаю это вполне корректным при поставленной задаче, т.к. не учитывается только параметр двойного предложения услуги и датасет меньше. Но эот позволило работать быстрее за счет меньшей длительности обработок.

In [ ]:
```
import dask.dataframe as dd
feature_df = dd.read_csv(r'D:\features.csv', sep='\t')
test_df = dd.read_csv('data_test.csv')
train_df = dd.read_csv('data_train.csv')
mg = test_df.merge(feature_df, left_on=['id', 'buy_time'],right_on=['id', 'buy_time'])
mgtr = train_df.merge(feature_df, left_on=['id', 'buy_time'],right_on=['id', 'buy_time'])
df = mg.compute()
df_tr = mg.compute()
df.to_csv('small_test.csv')
df_tr.to_csv('small_test.csv')
```

Данные поместил в новые файлы, чтобы можно было их быстрее читать.

In [2]:
```
import pandas as pd
test_df = pd.read_csv('small_test.csv')
train_df = pd.read_csv('small_train.csv')
```

In [3]:
```
train_df.head()
```

Out[3]:

	Unnamed: 0	Unnamed: 0_x	id	vas_id	buy_time	target	Unnamed: 0_y	0	1	2	...	243	244	245	
0	0	4477	3769599	5.0	1540760400	0.0	7368	199.810029	-106.489112	185.869214	...	-977.373846	-613.770792	-25.996269	-37.
1	1	8184	3726958	2.0	1532293200	1.0	7187	106.510029	58.440888	179.839214	...	-733.373846	-463.770792	-25.996269	-37.
2	2	21122	4109995	2.0	1544994000	0.0	864	-96.799971	66.400888	-110.740786	...	1378.626154	484.229208	-22.996269	-36.
3	3	24987	944353	2.0	1544994000	0.0	2207	-73.909971	187.450888	-74.720786	...	-909.373846	-606.770792	-0.996269	-33.
4	4	33338	294437	1.0	1533502800	0.0	10884	-96.799971	208.350888	-110.740786	...	-934.373846	-613.770792	-25.996269	163.

5 rows × 260 columns

Подготовка датасета

In [4]:
```
train_df = train_df.drop(columns = ['Unnamed: 0', 'Unnamed: 0_x', 'Unnamed: 0_y'])
train_df.drop_duplicates(subset=['id'], inplace=True)
train_df.set_index('id', inplace=True)
```

In [5]:
```
data_prelim = train_df.copy()

X = data_prelim.drop('target', axis=1)
y = data_prelim['target']
```

Проверка id на уникальность и дубли, а также на null значения

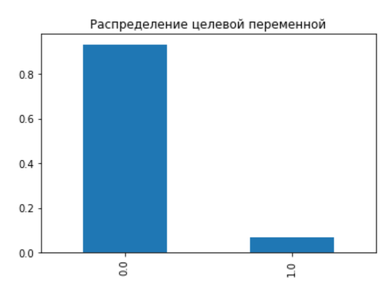
In [49]:
```
print("ID уникален? ", X.index.is_unique)
print("Есть ли дубли в строках?", X.index.duplicated().sum())
print("Сколько процент признаков могут принимать null-значениями? %d%%" % float((X.isnull().sum() > 0).sum()/X.shape
[1]*100))
```

```
ID уникален?  True
Есть ли дубли в строках? 0
Сколько процент признаков могут принимать null-значениями? 0%
```

Смотрим распределение классов, видим, что имеется дисбаланс, из-за проблем со временем не удалось над этим поработать

In [7]:
```
(y.value_counts()/y.shape[0]).plot(kind='bar', title='Распределение целевой переменной');
y.value_counts()/y.shape
```

Out[7]:
```
0.0    0.932352
1.0    0.067648
Name: target, dtype: float64
```



Импортируем первую модель, делим выборку на тренировочную и валидационную, создаем шаг для pipline

In [8]:
```
from sklearn.linear_model import LogisticRegression
RANDOM_STATE = 888
from sklearn.model_selection import train_test_split

RANDOM_STATE = 888

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=RANDOM_STATE)
step_log_reg = ('log_reg', LogisticRegression(random_state=RANDOM_STATE, n_jobs=-1))
```

In [9]:
```
from sklearn.pipeline import Pipeline

bl_estimator = Pipeline([
    step_log_reg
])
```

Функция для кросс-валидации и вывода результатов

In [10]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_validate
RANDOM_STATE = 888
kfold_cv = KFold(n_splits=3, shuffle=True, random_state=RANDOM_STATE)

def run_cv(estimator, cv, X, y, scoring='roc_auc', model_name=""):
    cv_res = cross_validate(estimator, X, y, cv=cv, scoring=scoring, n_jobs=-1)

    print("%s: %s = %0.2f (+/- %0.2f)" % (model_name,
                                          scoring,
                                          cv_res['test_score'].mean(),
                                          cv_res['test_score'].std() * 2))
```

Результаты baseline модели

In [11]:
```
run_cv(bl_estimator, kfold_cv, X_train, y_train, model_name="Baseline");
```

```
Baseline: roc_auc = 0.45 (+/- 0.02)
```

In [12]:
```
bl_estimator.fit(X_train, y_train)

bl_y_pred = bl_estimator.predict_proba(X_test)[:,1]
```

Feature Engineering

Определиние типов признаков из одного из уроков, сработало хорошо, проверил также бинарные признаки(их не оказалось) и поработал над признаком с типом дата.

In [13]:
```
from matplotlib import pyplot as plt
X_nunique = X.apply(lambda x: x.nunique(dropna=False))
X_nunique.shape
```

Out[13]: (255,)

In [14]:
```
plt.title("Распределение уникальных значений признаков");
X_nunique.hist(bins=100, figsize=(10, 5));
```

In [ ]:
```
f_all = set(X_nunique.index.tolist())
len(f_all)
```

In [16]:
```
f_const = set(X_nunique[X_nunique == 1].index.tolist())
len(f_const)
f_const
```

Out[16]: {'139', '15', '154', '203', '216', '75', '81', '85', '95'}

In [17]:
```
f_other = f_all - f_const
len(f_other)
```

Out[17]: 246

In [18]:
```
f_numeric = (X.loc[:, f_other].astype(int).sum() - X.loc[:, f_other].sum()).abs()
f_numeric = set(f_numeric[f_numeric > 0].index.tolist())
len(f_numeric)
```

Out[18]: 243

In [19]:
```
f_other = f_all - (f_numeric | f_const)
len(f_other)
```

Out[19]: 3

In [20]:
```
f_categorical = set(X_nunique.loc[f_other][X_nunique.loc[f_other] <= 10].index.tolist())
```

In [21]:
```
f_other = f_other - f_categorical
len(f_other)
```

Out[21]: 1

Написал кастомный трансформер для поля buy_time, не очень хорошо явно использовать поле, но он написан только для пайплайна итолько для этой задачи, поэтому думаю проблем быть не должно.

In [22]:
```
from sklearn.preprocessing import FunctionTransformer
def data_transform(X):
    X['buy_time'] = pd.to_datetime(X['buy_time'], unit='s')
    X['Month'] = pd.DatetimeIndex(X['buy_time']).month
    return X
from sklearn.preprocessing import FunctionTransformer
date_transformer = FunctionTransformer(data_transform)
```

In [23]:
```
f_ok = list(f_categorical | f_numeric)
f_categorical, f_numeric = list(f_categorical), list(f_numeric)
```

In [24]:
```
from sklearn.model_selection import train_test_split

RANDOM_STATE = 888

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=RANDOM_STATE)
```

In [25]:
```
from sklearn.base import BaseEstimator, TransformerMixin
```

Еще один пользовательски трансорматор для выбора колонок

In [26]:
```
from sklearn.base import BaseEstimator, TransformerMixin
import numpy as np
class ColumnSelector(BaseEstimator, TransformerMixin):
    def __init__(self, columns):
        self.columns = columns

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        assert isinstance(X, pd.DataFrame)

        try:
            return X[self.columns]
        except KeyError:
            cols_error = list(set(self.columns) - set(X.columns))
            raise KeyError("DataFrame не содердит следующие колонки: %s" % cols_error)
```

Собираем пайплайн предобрабоки

In [27]:
```
from sklearn.pipeline import FeatureUnion, make_pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
f_prep_pipeline = make_pipeline(
    date_transformer,
    ColumnSelector(columns=f_ok),
    FeatureUnion(transformer_list=[
        ("numeric_features", make_pipeline(
            ColumnSelector(f_numeric),
            StandardScaler()
        )),
        ("categorical_features", make_pipeline(
            ColumnSelector(f_categorical),
            OneHotEncoder(handle_unknown='ignore')
        )),

    ])
)
```

In [28]:
```
f_prep_pipeline.fit(X_train)
```

Out[28]:
```
Pipeline(steps=[('functiontransformer',
                 FunctionTransformer(func=<function data_transform at 0x0000024114D96678>)),
                ('columnselector',
                 ColumnSelector(columns=['210', '111', '51', '108', '220',
                                         '143', '123', '164', '13', '195',
                                         '176', '54', '189', '27', '102', '69',
                                         '162', '202', '38', '245', '135', '31',
                                         '115', '178', '67', '63', '119', '204',
                                         '43', '129', ...])),
                ('featureunion',
                 Feat...
                                                                                 '143',
                                                                                 '123',
                                                                                 '164',
                                                                                 '13',
                                                                                 '195',
                                                                                 '176',
                                                                                 '54',
                                                                                 '189',
                                                                                 '27',
                                                                                 '102',
                                                                                 '69',
                                                                                 '162',
                                                                                 '202',
                                                                                 '38',
                                                                                 '245',
                                                                                 '135',
                                                                                 '31',
                                                                                 '115',
                                                                                 '178',
                                                                                 '67',
                                                                                 '63',
                                                                                 '119',
                                                                                 '204',
                                                                                 '43',
                                                                                 '129', ...])),
                                                              ('standardscaler',
                                                               StandardScaler())])),
                                                 ('categorical_features',
                                                  Pipeline(steps=[('columnselector',
                                                                   ColumnSelector(columns=['252',
                                                                                           'vas_id'])),
                                                                  ('onehotencoder',
                                                                   OneHotEncoder(handle_unknown='ignore'))]))]))])
```

In [29]:
```
n_features = f_prep_pipeline.transform(X_train).shape[1]
n_features
```

Out[29]: 254

Эксперементируем с разными моделями

In [30]:
```
from sklearn.linear_model import LogisticRegression
lg_pipe = make_pipeline(
    f_prep_pipeline,
    LogisticRegression(random_state=888, max_iter=1000)
)
```

In [31]:
```
from sklearn.model_selection import GridSearchCV
```

Функция для подбора парметров

In [32]:
```
def run_grid_search(estimator, X, y, params_grid, cv, scoring='roc_auc'):
    gsc = GridSearchCV(estimator, params_grid, scoring=scoring, cv=cv, n_jobs=-1)

    gsc.fit(X, y)
    print("Best %s score: %.2f" % (scoring, gsc.best_score_))
    print()
    print("Best parameters set found on development set:")
    print()
    print(gsc.best_params_)
    print()
    print("Grid scores on development set:")
    print()

    for i, params in enumerate(gsc.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"
              % (gsc.cv_results_['mean_test_score'][i], gsc.cv_results_['std_test_score'][i] * 2, params))

    print()

    return gsc
```

In [33]:
```
param_grid = {
    "logisticregression__penalty": ['l1', 'l2'],
    "logisticregression__C": [0.01, 0.1, 5.0]
}

lg_gsc = run_grid_search(lg_pipe, X_train, y_train, param_grid, kfold_cv)
```

```
C:\Users\Gar\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:372: FitFailedWarning:
9 fits failed out of a total of 18.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
9 fits failed with the following error:
Traceback (most recent call last):
  File "C:\Users\Gar\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py", line 680, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "C:\Users\Gar\anaconda3\lib\site-packages\sklearn\pipeline.py", line 394, in fit
    self._final_estimator.fit(Xt, y, **fit_params_last_step)
  File "C:\Users\Gar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 1461, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "C:\Users\Gar\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py", line 449, in _check_solver
    % (solver, penalty)
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
C:\Users\Gar\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:972: UserWarning: One or more of the test
scores are non-finite: [       nan 0.87750332        nan 0.87815014        nan 0.87487731]
  category=UserWarning,
```

```
Best roc_auc score: 0.88

Best parameters set found on development set:

{'logisticregression__C': 0.1, 'logisticregression__penalty': 'l2'}

Grid scores on development set:

nan (+/-nan) for {'logisticregression__C': 0.01, 'logisticregression__penalty': 'l1'}
0.878 (+/-0.009) for {'logisticregression__C': 0.01, 'logisticregression__penalty': 'l2'}
nan (+/-nan) for {'logisticregression__C': 0.1, 'logisticregression__penalty': 'l1'}
0.878 (+/-0.010) for {'logisticregression__C': 0.1, 'logisticregression__penalty': 'l2'}
nan (+/-nan) for {'logisticregression__C': 5.0, 'logisticregression__penalty': 'l1'}
0.875 (+/-0.010) for {'logisticregression__C': 5.0, 'logisticregression__penalty': 'l2'}
```

In [34]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

Еще одна модель

In [35]:
```
gb_pipe = make_pipeline(
    f_prep_pipeline,
    GradientBoostingClassifier()
)
```

In [36]:
```
param_grid = {
    "gradientboostingclassifier__max_depth": [1, 5],
    "gradientboostingclassifier__n_estimators": [10, 100]
}

gb_fs_gsc = run_grid_search(gb_pipe, X, y, param_grid, kfold_cv)
```

```
Best roc_auc score: 0.90

Best parameters set found on development set:

{'gradientboostingclassifier__max_depth': 5, 'gradientboostingclassifier__n_estimators': 100}

Grid scores on development set:

0.875 (+/-0.004) for {'gradientboostingclassifier__max_depth': 1, 'gradientboostingclassifier__n_estimators': 10}
0.886 (+/-0.007) for {'gradientboostingclassifier__max_depth': 1, 'gradientboostingclassifier__n_estimators': 100}
0.899 (+/-0.002) for {'gradientboostingclassifier__max_depth': 5, 'gradientboostingclassifier__n_estimators': 10}
0.901 (+/-0.000) for {'gradientboostingclassifier__max_depth': 5, 'gradientboostingclassifier__n_estimators': 100}
```

In [37]:
```
lg_pipe_final = lg_gsc.best_estimator_
lg_pipe_final.fit(X_train, y_train)

lg_pred = lg_pipe_final.predict_proba(X_test)[:,1]
```

In [38]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, lg_pred > 0.5))
```

```
              precision    recall  f1-score   support

         0.0       0.93      0.99      0.96      9060
         1.0       0.39      0.05      0.09       667

    accuracy                           0.93      9727
   macro avg       0.66      0.52      0.53      9727
weighted avg       0.90      0.93      0.90      9727
```

In [39]:
```
gb_pipe_final = gb_fs_gsc.best_estimator_

gb_pipe_final.fit(X_train, y_train)

gb_pred = gb_pipe_final.predict_proba(X_test)[:,1]
```

In [40]:
```
print(classification_report(y_test, gb_pred > 0.5))
```

```
              precision    recall  f1-score   support

         0.0       0.94      0.99      0.96      9060
         1.0       0.44      0.15      0.23       667

    accuracy                           0.93      9727
   macro avg       0.69      0.57      0.59      9727
weighted avg       0.91      0.93      0.91      9727
```

Провел те же операции на тестовом наборе, чтобы сделать предсказание.

In [41]:
```
test_df = test_df.drop(columns = ['Unnamed: 0', 'Unnamed: 0_x', 'Unnamed: 0_y'])
test_df.drop_duplicates(subset=['id'], inplace=True)
test_df.set_index('id', inplace=True)
X_final = test_df.copy()
```

In [42]:
```
final_pred = gb_pipe_final.predict_proba(X_final)[:,1]
```

Экспорт модели в формате pkl

Так как лучшие результаты получились у модели градиентного бустинга экспортируем именно ее

In [43]:
```
import joblib
from sklearn.preprocessing import MinMaxScaler
```

In [44]:
```
joblib.dump(gb_pipe_final, 'pipeline.pkl')
```

Out[44]: ['pipeline.pkl']