Спортивный анализ данных. Платформа Kaggle

Урок 7. Тюнинг гиперпараметров, построение ансамблей алгоритмов.

Домашнее задание:

Чтобы было больше времени на выполнение курсовой работы, задание выполнить на **наборе данных для соревнования**:

Тестовая выборка - это выборка для применения модели и загрузки на ЛБ.

Задание 1: Обучить алгоритмы LightGBM и XGBoost, получить OOF прогнозы, оценить корреляцию прогнозов на обучающей выборке. Применить модели на тестовую выборку и оценить корреляцию.

Задание 2: Усреднить прогнозы с помощью арифмитического среднего, геометрического среднего и усреднить ранги, сделать выводы о качестве отдельных моделей и о качестве комбинации.

Задание 3: Обучить CatBoost, получить OOF прогнозы и выполнить задание 1 для трех моделей. Выполнить задание 2 для трех моделей.

Задание 4: (опция) Объединить OOF-прогнозы для трех моделей и обучить алгоритм Логистической регрессии (и любой другой, на ваше усмотрение). Сделать выводы о достигаемом качестве, сравнить достигаемое качество с качеством отдельных моделей и моделей, полученных в п.2 и п.4.

Задание 5: (опция) Обучить алгоритмRandomForest (желательно подтюнить параметры) и добавить к построенным ранее моделям. Выполнить задание 5.

В [1]:
```
import pandas as pd
import numpy as np
# Модель
import xgboost as xgb

from sklearn.model_selection import KFold, StratifiedKFold, train_test_split, cross_val_score
```

В [2]:
```
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 [3]:
```
TRAIN_DATASET_PATH = '../../data/train.csv'
TEST_DATASET_PATH = '../../data/test.csv'
BKI_DATASET_PATH = '../../data/bki.csv'
applications_history_DATASET_PATH = '../../data/applications_history.csv'
client_profile_DATASET_PATH = '../../data/client_profile.csv'
payments_DATASET_PATH = '../../data/payments.csv'
sample_submit_DATASET_PATH = '../../data/sample_submit.csv'

ID_COLUMN = 'APPLICATION_NUMBER'
ID_COLUMN_PR = 'PREV_APPLICATION_NUMBER'
TARGET = 'TARGET'
```

В [4]:
```
train = pd.read_csv(TRAIN_DATASET_PATH)
df_train =reduce_mem_usage(train)  # Уменьшаем размер данных
#df_train.info()


test = pd.read_csv(TEST_DATASET_PATH)
df_test =reduce_mem_usage(test)  # Уменьшаем размер данных

bki = pd.read_csv(BKI_DATASET_PATH)
df_bki =reduce_mem_usage(bki)  # Уменьшаем размер данных
#df_bki.info()

client_profile = pd.read_csv(client_profile_DATASET_PATH)
df_client_profile =reduce_mem_usage(client_profile)  # Уменьшаем размер данных
#df_client_profile.info()

payments = pd.read_csv(payments_DATASET_PATH)
df_payments =reduce_mem_usage(payments)  # Уменьшаем размер данных
#df_payments.info()

applications_history = pd.read_csv(applications_history_DATASET_PATH)
df_applications_history =reduce_mem_usage(applications_history)  # Уменьшаем размер данных
#df_applications_history.info()
```

```
Memory usage of dataframe is 2.52 MB
Memory usage after optimization is: 0.63 MB
Decreased by 75.0%
Memory usage of dataframe is 2.52 MB
Memory usage after optimization is: 0.79 MB
Decreased by 68.7%
Memory usage of dataframe is 122.60 MB
Memory usage after optimization is: 48.68 MB
Decreased by 60.3%
Memory usage of dataframe is 45.78 MB
Memory usage after optimization is: 18.12 MB
Decreased by 60.4%
Memory usage of dataframe is 62.50 MB
Memory usage after optimization is: 29.30 MB
Decreased by 53.1%
Memory usage of dataframe is 331.31 MB
Memory usage after optimization is: 114.69 MB
Decreased by 65.4%
```

В [5]:
```
df_train['TARGET'].value_counts()  # Количество различных значений признака 'TARGET'
```

Out[5]:
```
0    101196
1      8897
Name: TARGET, dtype: int64
```

В [6]:
```
train_df = df_train.merge(bki, on=ID_COLUMN, how='left')
train_df = train_df.merge(client_profile, on=ID_COLUMN, how='left')
train_df = train_df.merge(bki, on=ID_COLUMN, how='left')
```

В [7]:
```
test_df = df_test.merge(bki, on=ID_COLUMN, how='left')
test_df = test_df.merge(client_profile, on=ID_COLUMN, how='left')
test_df = test_df.merge(bki, on=ID_COLUMN, how='left')
#test_df.info()
```

В [8]:
```
train_df.set_index('APPLICATION_NUMBER', inplace=True)
```

В [9]:
```
X = train_df.drop('TARGET', axis=1)#.fillna(-1)
y = train_df['TARGET']
```

В [10]:
```
numerical_features = train_df.select_dtypes(exclude=["category"])
numerical_features = numerical_features.columns.tolist()
#numerical_features.remove('APPLICATION_NUMBER')
numerical_features.remove('TARGET')
```

В [11]:
```
ids = test_df['APPLICATION_NUMBER'].values
test_df.set_index('APPLICATION_NUMBER', inplace=True)
X_test = test_df[numerical_features]
```

В [12]:
```
x_train, x_test = train_test_split(
    X, train_size=0.75, random_state=27
)
y_train, y_test = train_test_split(
    y, train_size=0.75, random_state=27
)
print("x_train.shape = {} rows, {} cols".format(*x_train.shape))
print("x_test.shape = {} rows, {} cols".format(*x_test.shape))
```

```
x_train.shape = 1214461 rows, 56 cols
x_test.shape = 404821 rows, 56 cols
```

Задание 1:

Обучить алгоритмы LightGBM и XGBoost, получить OOF прогнозы, оценить корреляцию прогнозов на обучающей выборке. Применить модели на тестовую выборку и оценить корреляцию.

OOF — out of folds, техника получения предсказаний модели для тренировочной части датасета используя кросс-валидацию. Незаменима для дальнейшей сборки нескольких решений в ансамбль.

XGBoost

В [13]:
```
# Модель
import xgboost as xgb
# Метрика
from sklearn.metrics import roc_auc_score, auc
from sklearn.model_selection import KFold, StratifiedKFold, train_test_split, cross_val_score
```

В [14]:
```
xgb_params = {
    "booster": "gbtree",
    "objective": "binary:logistic",
    "eval_metric": "auc",
    "n_estimators": 2000,  # количество деревьев
    #"n_estimators": 250,
    "learning_rate": 0.1,
    "reg_lambda": 10,
    "max_depth": 4,
    "gamma": 10,
    "nthread": 6,
    "seed": 27
}
```

В [15]:
```
eval_sets= [
    (x_train[numerical_features], y_train),
    (x_test[numerical_features], y_test)
]
```

В [16]:
```
xgb_model = xgb.XGBClassifier(**xgb_params)

xgb_model.fit(
    y=y_train,
    X=x_train[numerical_features],
    early_stopping_rounds=50,
    eval_set=eval_sets,
    eval_metric="auc",
    verbose=10
)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\xgboost\sklearn.py:888: UserWarning: The use of label encoder in XGBClassifi
er is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_
label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0,
i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[0]	validation_0-auc:0.68473	validation_1-auc:0.68488
[10]	validation_0-auc:0.72499	validation_1-auc:0.72449
[20]	validation_0-auc:0.73940	validation_1-auc:0.73864
[30]	validation_0-auc:0.75032	validation_1-auc:0.74948
[40]	validation_0-auc:0.76294	validation_1-auc:0.76161
[50]	validation_0-auc:0.77358	validation_1-auc:0.77216
[60]	validation_0-auc:0.78203	validation_1-auc:0.78092
[70]	validation_0-auc:0.78772	validation_1-auc:0.78683
[80]	validation_0-auc:0.79132	validation_1-auc:0.79030
[90]	validation_0-auc:0.79683	validation_1-auc:0.79575
[100]	validation_0-auc:0.80106	validation_1-auc:0.79998
[110]	validation_0-auc:0.80515	validation_1-auc:0.80405
[120]	validation_0-auc:0.80891	validation_1-auc:0.80754
[130]	validation_0-auc:0.81325	validation_1-auc:0.81174
[140]	validation_0-auc:0.81732	validation_1-auc:0.81571
[150]	validation_0-auc:0.82085	validation_1-auc:0.81921
[160]	validation_0-auc:0.82336	validation_1-auc:0.82169
[170]	validation_0-auc:0.82557	validation_1-auc:0.82375
[180]	validation_0-auc:0.82832	validation_1-auc:0.82646
[190]	validation_0-auc:0.83143	validation_1-auc:0.82925
[200]	validation_0-auc:0.83322	validation_1-auc:0.83095
[210]	validation_0-auc:0.83509	validation_1-auc:0.83277
[220]	validation_0-auc:0.83692	validation_1-auc:0.83431
[230]	validation_0-auc:0.83837	validation_1-auc:0.83571
[240]	validation_0-auc:0.84115	validation_1-auc:0.83831
[250]	validation_0-auc:0.84302	validation_1-auc:0.83995
[260]	validation_0-auc:0.84461	validation_1-auc:0.84147
[270]	validation_0-auc:0.84780	validation_1-auc:0.84480
[280]	validation_0-auc:0.84926	validation_1-auc:0.84642
[290]	validation_0-auc:0.85124	validation_1-auc:0.84829
[300]	validation_0-auc:0.85327	validation_1-auc:0.85051
[310]	validation_0-auc:0.85481	validation_1-auc:0.85210
[320]	validation_0-auc:0.85719	validation_1-auc:0.85424
[330]	validation_0-auc:0.85854	validation_1-auc:0.85555
[340]	validation_0-auc:0.85969	validation_1-auc:0.85660
[350]	validation_0-auc:0.86162	validation_1-auc:0.85850
[360]	validation_0-auc:0.86256	validation_1-auc:0.85941
[370]	validation_0-auc:0.86428	validation_1-auc:0.86104
[380]	validation_0-auc:0.86602	validation_1-auc:0.86264
[390]	validation_0-auc:0.86750	validation_1-auc:0.86402
[400]	validation_0-auc:0.86897	validation_1-auc:0.86532
[410]	validation_0-auc:0.87022	validation_1-auc:0.86662
[420]	validation_0-auc:0.87165	validation_1-auc:0.86812
[430]	validation_0-auc:0.87292	validation_1-auc:0.86939
[440]	validation_0-auc:0.87487	validation_1-auc:0.87135
[450]	validation_0-auc:0.87602	validation_1-auc:0.87238
[460]	validation_0-auc:0.87757	validation_1-auc:0.87383
[470]	validation_0-auc:0.87870	validation_1-auc:0.87489
[480]	validation_0-auc:0.87997	validation_1-auc:0.87614
[490]	validation_0-auc:0.88104	validation_1-auc:0.87711
[500]	validation_0-auc:0.88182	validation_1-auc:0.87787
[510]	validation_0-auc:0.88292	validation_1-auc:0.87891
[520]	validation_0-auc:0.88422	validation_1-auc:0.88003
[530]	validation_0-auc:0.88514	validation_1-auc:0.88096
[540]	validation_0-auc:0.88612	validation_1-auc:0.88189
[550]	validation_0-auc:0.88670	validation_1-auc:0.88243
[560]	validation_0-auc:0.88703	validation_1-auc:0.88269
[570]	validation_0-auc:0.88810	validation_1-auc:0.88369
[580]	validation_0-auc:0.88874	validation_1-auc:0.88427
[590]	validation_0-auc:0.88935	validation_1-auc:0.88476
[600]	validation_0-auc:0.89028	validation_1-auc:0.88565
[610]	validation_0-auc:0.89081	validation_1-auc:0.88616
[620]	validation_0-auc:0.89163	validation_1-auc:0.88703
[630]	validation_0-auc:0.89247	validation_1-auc:0.88777
[640]	validation_0-auc:0.89310	validation_1-auc:0.88837
[650]	validation_0-auc:0.89419	validation_1-auc:0.88937
[660]	validation_0-auc:0.89542	validation_1-auc:0.89049
[670]	validation_0-auc:0.89685	validation_1-auc:0.89194
[680]	validation_0-auc:0.89769	validation_1-auc:0.89279
[690]	validation_0-auc:0.89848	validation_1-auc:0.89362
[700]	validation_0-auc:0.89943	validation_1-auc:0.89457
```

```
[710]	validation_0-auc:0.89990	validation_1-auc:0.89502
[720]	validation_0-auc:0.90026	validation_1-auc:0.89527
[730]	validation_0-auc:0.90089	validation_1-auc:0.89590
[740]	validation_0-auc:0.90202	validation_1-auc:0.89700
[750]	validation_0-auc:0.90315	validation_1-auc:0.89809
[760]	validation_0-auc:0.90396	validation_1-auc:0.89891
[770]	validation_0-auc:0.90476	validation_1-auc:0.89965
[780]	validation_0-auc:0.90526	validation_1-auc:0.90012
[790]	validation_0-auc:0.90598	validation_1-auc:0.90081
[800]	validation_0-auc:0.90660	validation_1-auc:0.90140
[810]	validation_0-auc:0.90726	validation_1-auc:0.90201
[820]	validation_0-auc:0.90805	validation_1-auc:0.90278
[830]	validation_0-auc:0.90879	validation_1-auc:0.90345
[840]	validation_0-auc:0.90996	validation_1-auc:0.90460
[850]	validation_0-auc:0.91073	validation_1-auc:0.90534
[860]	validation_0-auc:0.91126	validation_1-auc:0.90578
[870]	validation_0-auc:0.91183	validation_1-auc:0.90632
[880]	validation_0-auc:0.91252	validation_1-auc:0.90697
[890]	validation_0-auc:0.91322	validation_1-auc:0.90766
[900]	validation_0-auc:0.91359	validation_1-auc:0.90802
[910]	validation_0-auc:0.91418	validation_1-auc:0.90858
[920]	validation_0-auc:0.91448	validation_1-auc:0.90886
[930]	validation_0-auc:0.91448	validation_1-auc:0.90886
[940]	validation_0-auc:0.91448	validation_1-auc:0.90886
[950]	validation_0-auc:0.91448	validation_1-auc:0.90886
[960]	validation_0-auc:0.91448	validation_1-auc:0.90886
[967]	validation_0-auc:0.91448	validation_1-auc:0.90886
```

Out[16]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='auc',
              gamma=10, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.1, max_delta_step=0,
              max_depth=4, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=2000, n_jobs=6, nthread=6,
              num_parallel_tree=1, random_state=27, reg_alpha=0, reg_lambda=10,
              scale_pos_weight=1, seed=27, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

```
[920]	validation_0-auc:0.91448	validation_1-auc:0.90886
```

В [ ]:

```
y_pred_train_xgb = xgb_model.predict(train_df[numerical_features])
```

В [19]:

```
# make predictions for test data (Тестовая выборка)
# x = X_test.values
y_pred_xgb = xgb_model.predict(X_test)
output_xgb = pd.DataFrame({'APPLICATION_NUMBER': ids, 'TARGET': y_pred_xgb})
```

В [20]:

```
output_xgb.tail(10)
```

Out[20]:

	APPLICATION_NUMBER	TARGET
2436793	123433260	0
2436794	123433260	0
2436795	123433260	0
2436796	123433260	0
2436797	123433260	0
2436798	123433260	0
2436799	123433260	0
2436800	123433260	0
2436801	123433260	0
2436802	123433260	0

В [21]:

```
output_xgb['TARGET'].value_counts()  # Количество различных значений признака 'TARGET'
```

Out[21]:

```
0    2421344
1      15459
Name: TARGET, dtype: int64
```

Оценка качества модели

В [23]:
```python
train_score = roc_auc_score(y_train, xgb_model.predict(x_train[numerical_features]))
test_score = roc_auc_score(y_test, xgb_model.predict(x_test[numerical_features]))

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

```
train_score=0.6694041054483558
test_score=0.6671894975645648
```

LightGBM Sklearn-API

В [24]:
```python
import lightgbm as lgb
```

В [28]:
```python
# Задача бинарной классификации
lgb_params = {
    "boosting_type": "gbdt",  # gradient boosting tree decision tree (бустинг над решающими деревьями)
    "objective": "binary",
    "metric": "auc", # метрика качества - ROC AUC
    #"learning_rate": 0.01,  # скорость обучения
    "learning_rate": 0.1,  # скорость обучения
    # "n_estimators": 20000,  # число деревьев
    "n_estimators": 10000,  # число деревьев
    #"n_estimators": 250,
     # 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 (1:15:40)
    #"nthread": 6, # число ядер
    "n_jobs": 6,
    "seed": 27
}
```

В [29]:
```python
# Оценить качество модели на валидационной выборке, оценить расхождение
# по сравнению с качеством на обучающей выборке и валидационной выборке.
model_lgb = lgb.LGBMClassifier(**lgb_params)
model_lgb.fit(
    X=x_train[numerical_features],
    y=y_train,
    eval_set=[(x_train[numerical_features], y_train), (x_test[numerical_features], y_test)],
    #categorical_feature = catigorical_features_name,
    early_stopping_rounds=25,
    eval_metric="auc",
    verbose=500
)
```

```
Training until validation scores don't improve for 25 rounds
[500]	valid_0's auc: 0.869603	valid_1's auc: 0.865265
[1000]	valid_0's auc: 0.903491	valid_1's auc: 0.897719
[1500]	valid_0's auc: 0.925507	valid_1's auc: 0.919065
[2000]	valid_0's auc: 0.938472	valid_1's auc: 0.931507
[2500]	valid_0's auc: 0.948264	valid_1's auc: 0.941072
[3000]	valid_0's auc: 0.955083	valid_1's auc: 0.947619
[3500]	valid_0's auc: 0.960691	valid_1's auc: 0.953151
[4000]	valid_0's auc: 0.965539	valid_1's auc: 0.95776
[4500]	valid_0's auc: 0.969229	valid_1's auc: 0.961325
[5000]	valid_0's auc: 0.972071	valid_1's auc: 0.964034
[5500]	valid_0's auc: 0.974708	valid_1's auc: 0.966659
[6000]	valid_0's auc: 0.977071	valid_1's auc: 0.96911
[6500]	valid_0's auc: 0.979196	valid_1's auc: 0.971297
[7000]	valid_0's auc: 0.9808	valid_1's auc: 0.972898
[7500]	valid_0's auc: 0.982253	valid_1's auc: 0.974398
[8000]	valid_0's auc: 0.983597	valid_1's auc: 0.975818
[8500]	valid_0's auc: 0.98494	valid_1's auc: 0.977109
[9000]	valid_0's auc: 0.986057	valid_1's auc: 0.978247
[9500]	valid_0's auc: 0.987085	valid_1's auc: 0.979238
[10000]	valid_0's auc: 0.987944	valid_1's auc: 0.980112
Did not meet early stopping. Best iteration is:
[10000]	valid_0's auc: 0.987944	valid_1's auc: 0.980112
```

Out[29]:
```
LGBMClassifier(max_depth=4, metric='auc', n_estimators=10000, n_jobs=6,
               objective='binary', reg_lambda=100, seed=27)
```

```
[10000]	valid_0's auc: 0.987944	valid_1's auc: 0.980112
```

В [ ]:
```python
y_pred_train_lgb = model_lgb.predict(train_df[numerical_features])
```

В [31]:
```python
# make predictions for test data  (Тестовая выборка)
#ids = test_df['APPLICATION_NUMBER'].values
x = X_test[numerical_features].values
y_pred_lgb = model_lgb.predict(x)
output_lgb = pd.DataFrame({'APPLICATION_NUMBER': ids, 'TARGET': y_pred_lgb})
```

B [32]:
```
output_lgb.head(10)
```

Out[32]:

	APPLICATION_NUMBER	TARGET
0	123724268	0
1	123456549	1
2	123456549	1
3	123456549	1
4	123456549	1
5	123428178	0
6	123428178	0
7	123428178	0
8	123428178	0
9	123428178	0

B [33]:
```
output_lgb['TARGET'].value_counts()  # Количество различных значений признака 'TARGET'
```

Out[33]:
```
0    2401692
1      35111
Name: TARGET, dtype: int64
```

Оценка качества модели

B [35]:
```
train_score = roc_auc_score(y_train, model_lgb.predict(x_train[numerical_features]))
test_score = roc_auc_score(y_test, model_lgb.predict(x_test[numerical_features]))

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

```
train_score=0.8688907741973214
test_score=0.8552374766749109
```

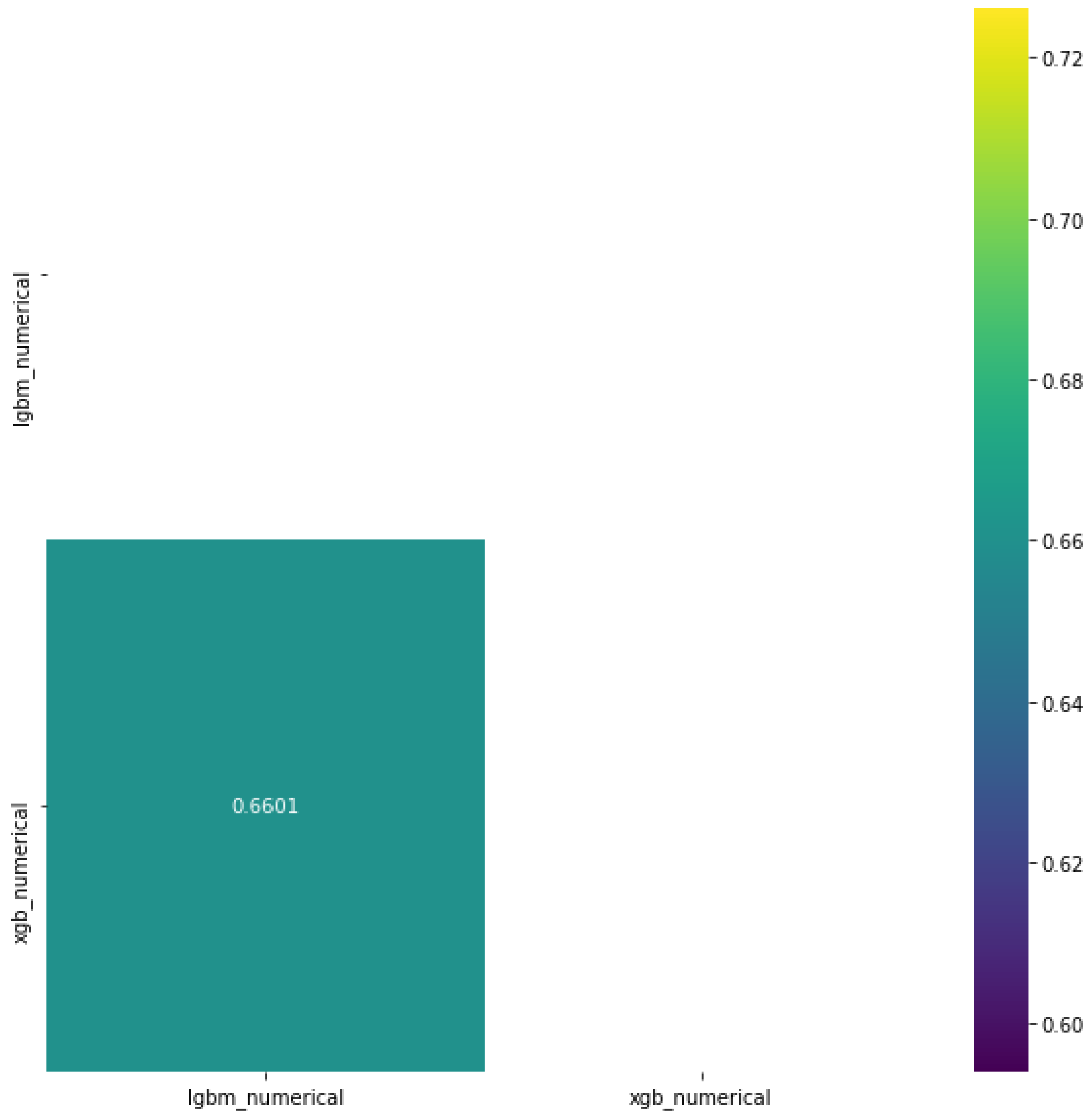
Prediction Correlation

Корреляция на тренировочной выборке

B [ ]:
```
# 2. Визуализация
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

# scores = pd.DataFrame({
#     'lgbm_numerical': model_lgb.predict(x_train[numerical_features]),
#     'xgb_numerical': xgb_model.predict(x_train[numerical_features])
# })


# y_pred_train_lgb = model_lgb.predict(train_df[numerical_features]
# y_pred_train_xgb = xgb_model.predict(train_df[numerical_features])
# scores = pd.DataFrame({
#     'lgbm_numerical': model_lgb.predict(train_df[numerical_features]),
#     'xgb_numerical': xgb_model.predict(train_df[numerical_features])
# })

scores = pd.DataFrame({
    'lgbm_numerical': y_pred_train_lgb,
    'xgb_numerical': y_pred_train_xgb
})

corr = scores.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
```

В [39]:
```
fig, axes = plt.subplots(1, 1, figsize=(10, 10))
sns.heatmap(corr, mask=mask, annot=True, fmt=".4f", cmap="viridis", ax=axes)
```

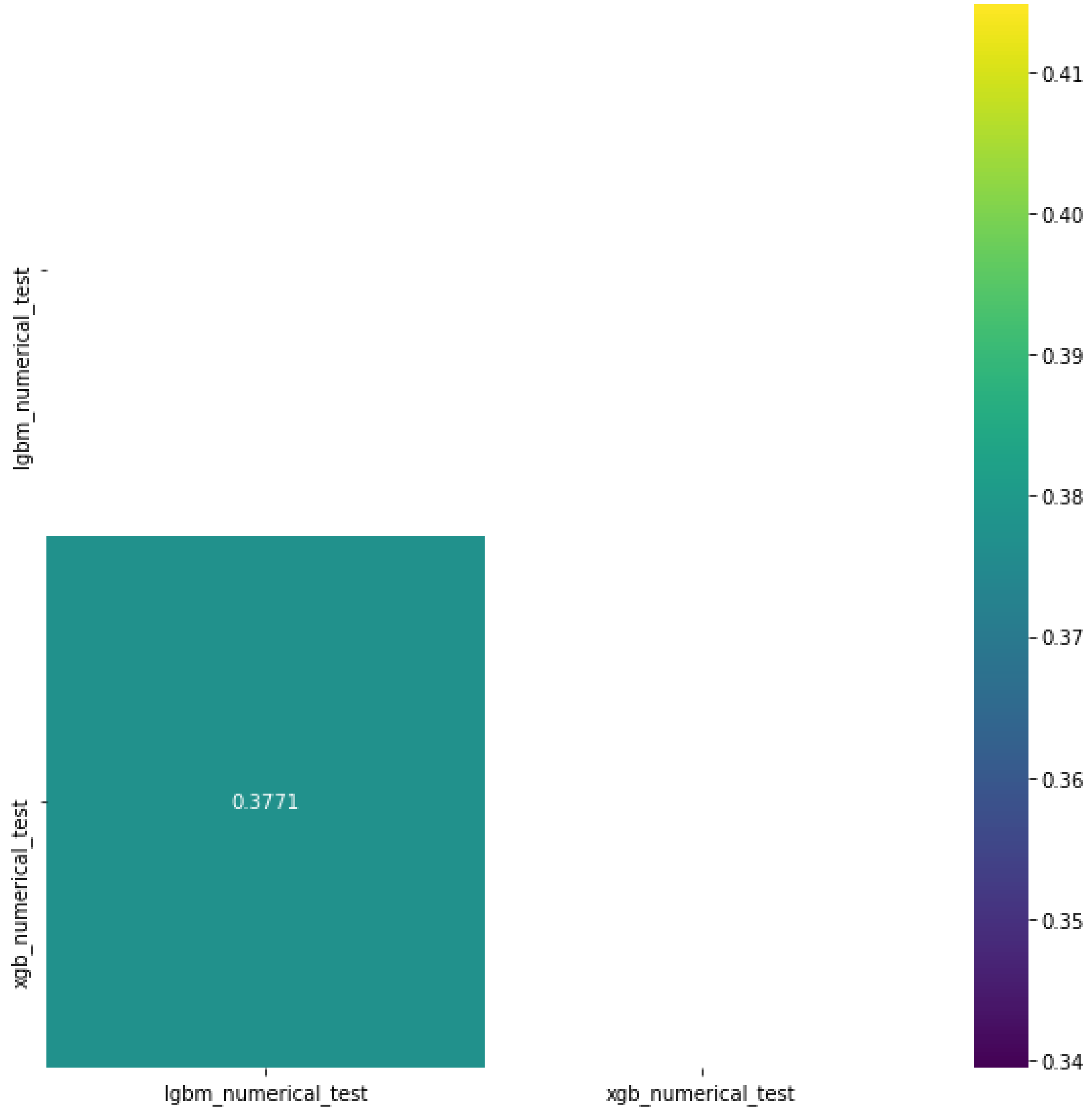
Out[39]: <AxesSubplot:>



Корреляция на тестовой выборке

В [40]:
```
scores = pd.DataFrame({
    'lgbm_numerical_test': y_pred_lgb,
    'xgb_numerical_test': y_pred_xgb
})

corr = scores.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

fig, axes = plt.subplots(1, 1, figsize=(10, 10))
sns.heatmap(corr, mask=mask, annot=True, fmt=".4f", cmap="viridis", ax=axes)
```

Out[40]: <AxesSubplot:>



Задание 2:

Усреднить прогнозы с помощью арифмитического среднего, геометрического среднего и усреднить ранги, сделать выводы о качестве отдельных моделей и о качестве комбинации.

В [42]:
```
# scores = pd.DataFrame({
#     'lgbm_numerical': model_lgb.predict(x_train[numerical_features]),
#     'xgb_numerical': xgb_model.predict(x_train[numerical_features])
# })

#y_pred_train_lgb = model_lgb.predict(train_df[numerical_features])
#y_pred_train_xgb = xgb_model.predict(train_df[numerical_features])

# scores = pd.DataFrame({
#     'lgbm_numerical': model_lgb.predict(train_df[numerical_features]),
#     'xgb_numerical': xgb_model.predict(train_df[numerical_features])
# })

scores = pd.DataFrame({
    'lgbm_numerical': y_pred_train_lgb,
    'xgb_numerical': y_pred_train_xgb
})
```

AMean

В [45]:
```
# y = train_df['TARGET']
scores_mean = scores.mean(axis=1)
# score = roc_auc_score(y_train, scores_mean)
score = roc_auc_score(y, scores_mean)
print(f"Score = {round(score, 4)}")
```

```
Score = 0.8657
```

GMean

В [52]:
```
from scipy.stats import gmean, rankdata

scores_mean = gmean(scores, axis=1)
# score = roc_auc_score(target, scores_mean)
#score = roc_auc_score(y_train, scores_mean)
score = roc_auc_score(y, scores_mean)
print(f"Score = {round(score, 4)}")
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:402: RuntimeWarning: divide by zero encountered in log
  log_a = np.log(np.array(a, dtype=dtype))

Score = 0.6688
```

Rankdata

В [53]:
```
scores_mean = scores.rank().mean(axis=1)
#score = roc_auc_score(y_train, scores_mean)
score = roc_auc_score(y, scores_mean)
print(f"Score = {round(score, 4)}")
```

```
Score = 0.8657
```

В [55]:
```
scores_mean = gmean(scores.rank(), axis=1)
#score = roc_auc_score(y_train, scores_mean)
score = roc_auc_score(y, scores_mean)
#score_meean = scores.rank().gmean(axis=1)
print(f"Score = {round(score, 4)}")
```

```
Score = 0.8658
```

Задание 3:

Обучить CatBoost, получить OOF прогнозы и выполнить задание 1 для трех моделей. Выполнить задание 2 для трех моделей.

[https://catboost.ai/docs/concepts/python-usages-examples.html (https://catboost.ai/docs/concepts/python-usages-examples.html)](https://catboost.ai/docs/concepts/python-usages-examples.html)

В [56]:
```
import catboost as cb
```

В [61]:
```
cb_params = {
    "n_estimators": 10000,
    "loss_function": "Logloss",
    "eval_metric": "AUC",
    "task_type": "CPU",
    #"max_bin": 20,
    "verbose": 10,
    "max_depth": 6,
    "l2_leaf_reg": 100,
    "early_stopping_rounds": 50,
    "thread_count": 6,
    "random_seed": 42
}
```

В [62]:
```
cb_model = cb.CatBoostClassifier(**cb_params)
```

В [63]:
```
# eval_sets= [
#     (x_train[numerical_features], y_train),
#     (x_test[numerical_features], y_test)
# ]
```

В [64]:
```
cb_model.fit(
    x_train[numerical_features],
    y_train,
    # cat_features = new_categorical_features,
    eval_set=eval_sets)
```

```
9870:   test: 0.9602069 test1: 0.9536735        best: 0.9536735 (9870)  total: 1h 19m 36s       remaining: 1m 2s
9880:   test: 0.9602345 test1: 0.9537001        best: 0.9537001 (9880)  total: 1h 19m 40s       remaining: 57.6s
9890:   test: 0.9602796 test1: 0.9537483        best: 0.9537483 (9890)  total: 1h 19m 45s       remaining: 52.7s
9900:   test: 0.9603023 test1: 0.9537708        best: 0.9537713 (9899)  total: 1h 19m 50s       remaining: 47.9s
9910:   test: 0.9603148 test1: 0.9537798        best: 0.9537809 (9907)  total: 1h 19m 54s       remaining: 43.1s
9920:   test: 0.9603330 test1: 0.9537947        best: 0.9537955 (9917)  total: 1h 19m 58s       remaining: 38.2s
9930:   test: 0.9603464 test1: 0.9538075        best: 0.9538075 (9930)  total: 1h 20m 2s        remaining: 33.4s
9940:   test: 0.9603868 test1: 0.9538476        best: 0.9538476 (9940)  total: 1h 20m 6s        remaining: 28.5s
9950:   test: 0.9604075 test1: 0.9538700        best: 0.9538700 (9950)  total: 1h 20m 10s       remaining: 23.7s
9960:   test: 0.9604168 test1: 0.9538764        best: 0.9538764 (9960)  total: 1h 20m 14s       remaining: 18.9s
9970:   test: 0.9604482 test1: 0.9539101        best: 0.9539101 (9970)  total: 1h 20m 18s       remaining: 14s
9980:   test: 0.9604717 test1: 0.9539284        best: 0.9539284 (9980)  total: 1h 20m 22s       remaining: 9.18s
9990:   test: 0.9605020 test1: 0.9539554        best: 0.9539554 (9990)  total: 1h 20m 26s       remaining: 4.35s
9999:   test: 0.9605143 test1: 0.9539686        best: 0.9539686 (9999)  total: 1h 20m 30s       remaining: 0us

bestTest = 0.9539685918
bestIteration = 9999
```

Out[64]:
```
<catboost.core.CatBoostClassifier at 0x20406bd400>
```

bestTest = 0.8436279973   bestIteration = 999

В [65]:
```
y_pred_train_cb = cb_model.predict(train_df[numerical_features])
```

В [66]:
```
# make predictions for test data   (Тестовая выборка)
x = X_test.values
y_pred_cb = cb_model.predict(x)
output_cb = pd.DataFrame({'APPLICATION_NUMBER': ids, 'TARGET': y_pred_cb})
```

В [67]:
```
output_cb.head(10)
```

Out[67]:

	APPLICATION_NUMBER	TARGET
0	123724268	0
1	123456549	0
2	123456549	0
3	123456549	0
4	123456549	0
5	123428178	0
6	123428178	0
7	123428178	0
8	123428178	0
9	123428178	0

В [68]:
```
output_cb['TARGET'].value_counts()  # Количество различных значений признака 'TARGET'
```

Out[68]:
```
0    2420411
1      16392
Name: TARGET, dtype: int64
```

Оценка качества модели

В [70]:
```
train_score = roc_auc_score(y_train, cb_model.predict(x_train[numerical_features]))
test_score = roc_auc_score(y_test, cb_model.predict(x_test[numerical_features]))

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

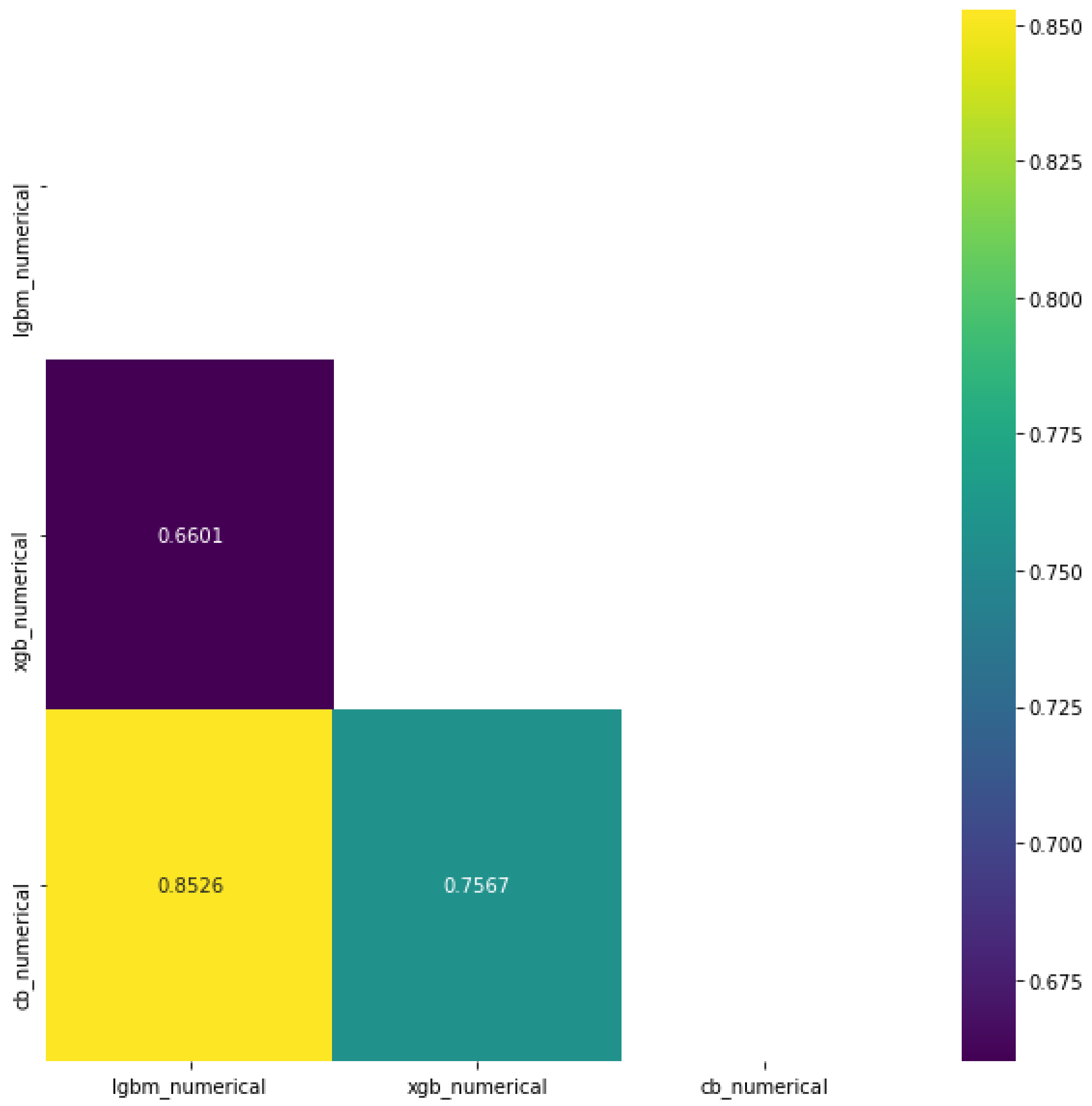
```
train_score=0.7767423095417059
test_score=0.7695052642551211
```

Prediction Correlation

Корреляция на тренировочной выборке

В [73]:
```
import seaborn as sns

# scores = pd.DataFrame({
#     'lgbm_numerical': model_lgb.predict(x_train[numerical_features]),
#     'xgb_numerical': xgb_model.predict(x_train[numerical_features]),
#     'cb_numerical': cb_model.predict(x_train[numerical_features])
# })

# scores = pd.DataFrame({
#     'lgbm_numerical': model_lgb.predict(train_df[numerical_features]),
#     'xgb_numerical': xgb_model.predict(train_df[numerical_features]),
#     'cb_numerical': cb_model.predict(train_df[numerical_features])
# })

scores = pd.DataFrame({
    'lgbm_numerical': y_pred_train_lgb,
    'xgb_numerical': y_pred_train_xgb,
    'cb_numerical': y_pred_train_cb
})

corr = scores.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

fig, axes = plt.subplots(1, 1, figsize=(10, 10))
sns.heatmap(corr, mask=mask, annot=True, fmt=".4f", cmap="viridis", ax=axes)
```

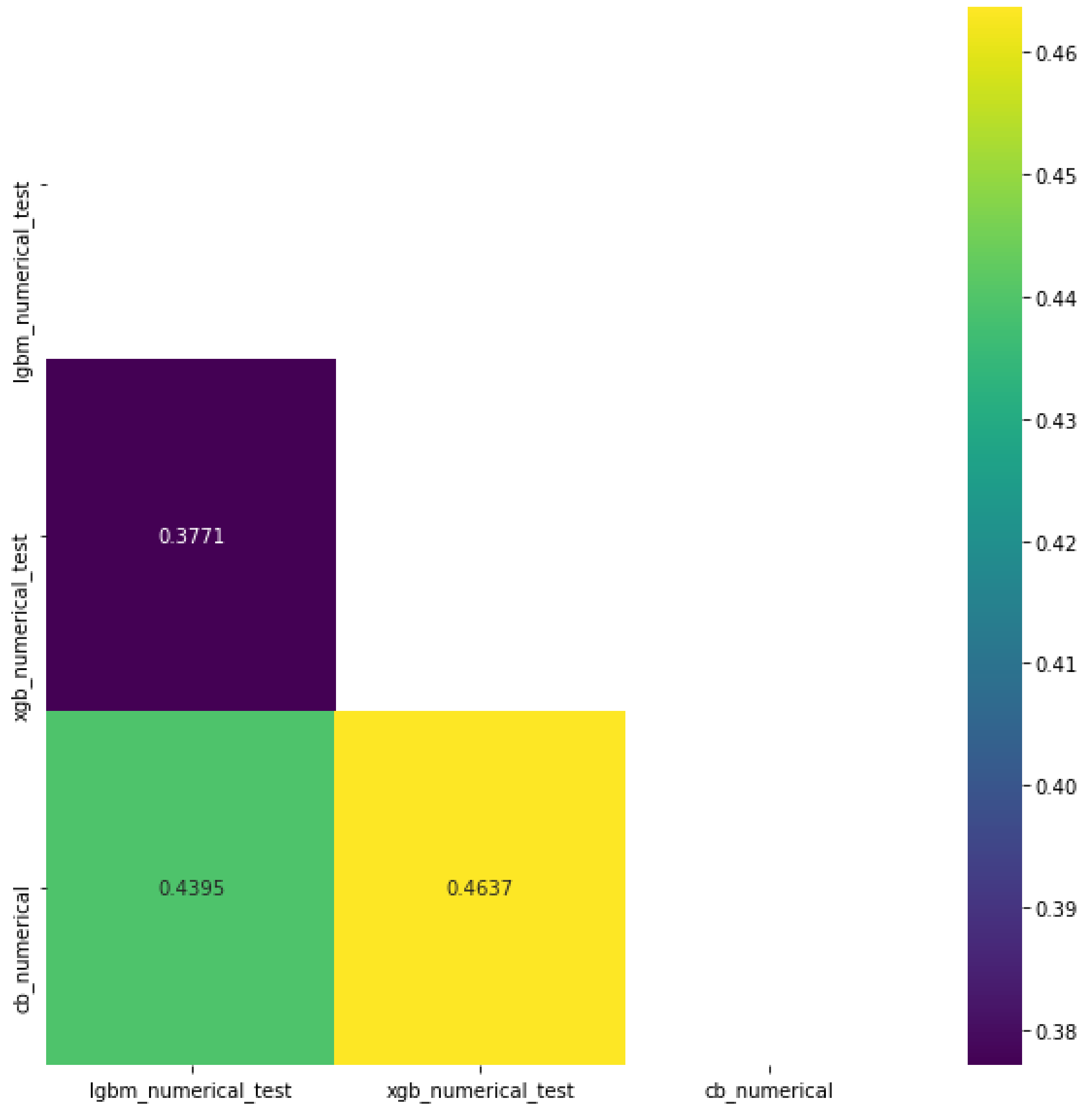
Out[73]: <AxesSubplot:>



Корреляция на тестовой выборке

B [74]:
```
scores = pd.DataFrame({
    'lgbm_numerical_test': y_pred_lgb,
    'xgb_numerical_test': y_pred_xgb,
    'cb_numerical': y_pred_cb
})

corr = scores.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

fig, axes = plt.subplots(1, 1, figsize=(10, 10))
sns.heatmap(corr, mask=mask, annot=True, fmt=".4f", cmap="viridis", ax=axes)
```

Out[74]: <AxesSubplot:>



B [75]:
```
scores = pd.DataFrame({
    'lgbm_numerical': y_pred_train_lgb,
    'xgb_numerical': y_pred_train_xgb,
    'cb_numerical': y_pred_train_cb
})
```

AMean

B [76]:
```
# y = train_df['TARGET']
scores_mean = scores.mean(axis=1)
# score = roc_auc_score(y_train, scores_mean)
score = roc_auc_score(y, scores_mean)
print(f"Score = {round(score, 4)}")
```

```
Score = 0.8665
```

GMean

B [78]:
```
#scores_meean = scores.gmean(axis=1)
scores_mean = gmean(scores, axis=1)
# score = roc_auc_score(target, scores_mean)
#score = roc_auc_score(y_train, scores_mean)
score = roc_auc_score(y, scores_mean)
print(f"Score = {round(score, 4)}")
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\stats\stats.py:402: RuntimeWarning: divide by zero encountered in log
  log_a = np.log(np.array(a, dtype=dtype))

Score = 0.6663
```

Rankdata

В [79]:
```
scores_mean = scores.rank().mean(axis=1)
#score = roc_auc_score(y_train, scores_mean)
score = roc_auc_score(y, scores_mean)
print(f"Score = {round(score, 4)}")
```

```
Score = 0.8665
```

В [80]:
```
scores_mean = gmean(scores.rank(), axis=1)
#score = roc_auc_score(y_train, scores_mean)
score = roc_auc_score(y, scores_mean)
#score_meean = scores.rank().gmean(axis=1)
print(f"Score = {round(score, 4)}")
```

```
Score = 0.8665
```

Задание 4:

(опция) Объединить OOF-прогнозы для трех моделей и обучить алгоритм Логистической регрессии (и любой другой, на ваше усмотрение). Сделать выводы о достигаемом качестве, сравнить достигаемое качество с качеством отдельных моделей и моделей, полученных в п.2 и п.4.

Задание 5:

(опция) Обучить алгоритмRandomForest (желательно подтюнить параметры) и добавить к построенным ранее моделям. Выполнить задание 5.

В [ ]:
```

```