B [714]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pyth
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all file

# import os
# for dirname, _, filenames in os.walk('/kaggle/input'):
#     for filename in filenames:
#         print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserv
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside o
```

B [715]:
```
import time
import numpy as np
import pandas as pd


# Модель
import xgboost as xgb
import catboost as cb
import lightgbm as lgb

from sklearn.metrics import roc_auc_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

pd.set_option("display.max_columns", 30)
```

Usefull Functions

В [716]:
```python
def catboost_cross_validation(params, X, y, cv, categorical = None):
    # Кросс-валидация модели catboost
    estimartors, folds_scores = [], []
    oof_preds = np.zeros(X.shape[0])

    print(f"{time.ctime()}, Cross-validation, {X.shape[0]} rows, {X.shape[1]} cols")
    X[categorical] = X[categorical].astype(str)
    #X[categorical] = X[categorical].astype('S32')

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

        model = cb.CatBoostClassifier(**params)
        model.fit(
            x_train, y_train, categorical,
            eval_set=[(x_train, y_train), (x_valid, y_valid)]
        )
        oof_preds[valid_idx] = model.predict_proba(x_valid)[:, 1]
        score = roc_auc_score(y_valid, oof_preds[valid_idx])
        print(f"Fold {fold+1}, Valid score = {round(score, 5)}")
        folds_scores.append(round(score, 5))
        estimartors.append(model)

    print(f"Score by each fold: {folds_scores}")
    print("="*65)

    return estimartors, oof_preds
```

В [717]:
```python
def lightgbm_cross_validation_1(params, X, y, cv, categorical = None):
    # Кросс-валидация модели lightgbm
    estimartors, folds_scores = [], []
    oof_preds = np.zeros(X.shape[0])

    print(f"{time.ctime()}, Cross-validation, {X.shape[0]} rows, {X.shape[1]} cols")

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

        model = lgb.LGBMClassifier(**params)
        model.fit(
            X=x_train[numerical + categorical],
            y=y_train,
            eval_set=[(x_train[numerical + categorical], y_train), (x_valid[numerical +
            categorical_feature = categorical,
            early_stopping_rounds=50,
            eval_metric="auc",
            verbose=50
        )

        oof_preds[valid_idx] = model.predict_proba(x_valid)[:, 1]
        score = roc_auc_score(y_valid, oof_preds[valid_idx])
        print(f"Fold {fold+1}, Valid score = {round(score, 5)}")
        folds_scores.append(round(score, 5))
        estimartors.append(model)

    print(f"Score by each fold: {folds_scores}")
    print("="*65)

    return estimartors, oof_preds
```

В [718]:
```python
def lightgbm_cross_validation(params, X, y, cv, categorical = None):
    # Кросс-валидация модели lightgbm
    estimartors, folds_scores = [], []
    oof_preds = np.zeros(X.shape[0])

    print(f"{time.ctime()}, Cross-validation, {X.shape[0]} rows, {X.shape[1]} cols")
    if isinstance(categorial, list):
        X[categorial] = X[categorial].astype(pd.category)
        categorical = categorical
    else:
        catigorical = "auto"


    for fold, (train_idx, valid_idx) in enumerate(cv.split(X, y)):

        x_train, x_valid = X.loc[train_idx], X.loc[valid_idx]
        y_train, y_valid = y[train_idx], y[valid_idx]

        model = lgb.LGBMClassifier(**params)
        model.fit(
            X=x_train,
            y=y_train,
            categorical_feature = categorical,
            eval_set=[(x_train, y_train), (x_valid, y_valid)],
            eval_names=["dtrain", "dvalid"],
            early_stopping_rounds=100,
            eval_metric="auc",
            verbose=25
        )

        oof_preds[valid_idx] = model.predict_proba(x_valid)[:, 1]
        score = roc_auc_score(y_valid, oof_preds[valid_idx])
        print(f"Fold {fold+1}, Valid score = {round(score, 5)}")
        folds_scores.append(round(score, 5))
        estimartors.append(model)

    print(f"Score by each fold: {folds_scores}")
    print("="*65)

    return estimartors, oof_preds
```

В [719]:

```python
def create_client_profile_features(X: pd.DataFrame, copy: bool = True) -> pd.DataFrame:
    # Создание признака на основе профиля клиентов.

    # AMOUNT_CREDIT - сумма кредита
    # AMOUNT_ANNUITY - сумма платежа
    if copy:
        X = X.copy()

    X["DAYS_ON_LAST_JOB"] = X["DAYS_ON_LAST_JOB"].replace(365243, np.nan)
    bki_flags = [flag for flag in X.columns if "AMT_REQ_CREDIT_BUREAU" in flag]
    X["bki_requests_count"] = X[bki_flags].sum(axis=1)
    X["bki_kurtosis"] = X[bki_flags].kurtosis(axis=1)

    X["external_scoring_prod"] = X["EXTERNAL_SCORING_RATING_1"] * X["EXTERNAL_SCORING_RA
    X["external_scoring_weighted"] = X.EXTERNAL_SCORING_RATING_1 * 2 + X.EXTERNAL_SCORIN

    for function_name in ["min", "max", "mean", "nanmedian", "var"]:
        feature_name = "external_scoring_rating_{}".format(function_name)
        X[feature_name] = eval("np.{}".format(function_name))(
            X[["EXTERNAL_SCORING_RATING_1", "EXTERNAL_SCORING_RATING_2", "EXTERNAL_SCORI
        )

    # Отношение между основными фин. показателями
    X["ratio_credit_to_annuity"] = X["AMOUNT_CREDIT"] / X["AMOUNT_ANNUITY"]
    X["ratio_annuity_to_salary"] = X["AMOUNT_ANNUITY"] / X["TOTAL_SALARY"]
    X["ratio_credit_to_salary"] = X["AMOUNT_CREDIT"] / X["TOTAL_SALARY"]

    # Отношение фин. показателей к возрасту и временным фичам
    X["ratio_annuity_to_age"] = X["AMOUNT_ANNUITY"] / X["AGE"]
    X["ratio_credit_to_age"] = X["AMOUNT_CREDIT"] / X["AGE"]
    X["ratio_salary_to_age"] = X["TOTAL_SALARY"] / X["AGE"]
    X["ratio_salary_to_experience"] = X["TOTAL_SALARY"] / X["DAYS_ON_LAST_JOB"]
    X["ratio_credit_to_experience"] = X["AMOUNT_CREDIT"] / X["DAYS_ON_LAST_JOB"]
    X["ratio_annuity_to_experience"] = X["AMOUNT_ANNUITY"] / X["DAYS_ON_LAST_JOB"]

    # Отношения временных признаков
    X["ratio_age_to_experience"] = X["AGE"] / X["DAYS_ON_LAST_JOB"]
    X["ratio_salary_to_region_population"] = X["TOTAL_SALARY"] / X["REGION_POPULATION"]
    X["ratio_car_to_experience"] = X["OWN_CAR_AGE"] / X["DAYS_ON_LAST_JOB"]
    X["ratio_car_to_age"] = X["OWN_CAR_AGE"] / X["AGE"]

    # Произведение фин. показателей кредита на вероятность дефолта
    # Такая штука называется математическим ожиданием дефолта или ожиданиемыми потерями
    X["expected_total_loss_1"] = X["EXTERNAL_SCORING_RATING_1"] * X["AMOUNT_CREDIT"]
    X["expected_total_loss_2"] = X["EXTERNAL_SCORING_RATING_2"] * X["AMOUNT_CREDIT"]
    X["expected_total_loss_3"] = X["EXTERNAL_SCORING_RATING_3"] * X["AMOUNT_CREDIT"]
    X["expected_monthly_loss_1"] = X["EXTERNAL_SCORING_RATING_1"] * X["AMOUNT_ANNUITY"]
    X["expected_monthly_loss_2"] = X["EXTERNAL_SCORING_RATING_2"] * X["AMOUNT_ANNUITY"]
    X["expected_monthly_loss_3"] = X["EXTERNAL_SCORING_RATING_3"] * X["AMOUNT_ANNUITY"]

    return X
```

В [720]:
```python
def new_catigirical_features(X: pd.DataFrame, copy: bool = True) -> pd.DataFrame:
    if copy:
        df = X.copy()

        # NAME_CONTRACT_TYPE
        cat_colname = 'NAME_CONTRACT_TYPE'
        # Years in current job
        df[cat_colname] = df[cat_colname].replace(to_replace = np.nan, value = 'неизвест

        df.loc[df[cat_colname] == 'Cash', cat_colname] = 0
        df.loc[df[cat_colname] == 'Credit Card', cat_colname] = 1
        df.loc[df[cat_colname] == 'неизвестно', cat_colname] = 2

        # GENDER
        cat_colname = 'GENDER'
        df[cat_colname] = df[cat_colname].replace(to_replace = np.nan, value = 'неизвест

        df.loc[df[cat_colname] == 'F', cat_colname] = 0
        df.loc[df[cat_colname] == 'M', cat_colname] = 1
        df.loc[df[cat_colname] == 'XNA', cat_colname] = 2
        df.loc[df[cat_colname] == 'неизвестно', cat_colname] = 3

        # EDUCATION_LEVEL
        cat_colname = 'EDUCATION_LEVEL'
        df[cat_colname] = df[cat_colname].replace(to_replace = np.nan, value = 'неизвест

        df.loc[df[cat_colname] == 'Secondary / secondary special', cat_colname] = 0
        df.loc[df[cat_colname] == 'Higher education', cat_colname] = 1
        df.loc[df[cat_colname] == 'Incomplete higher', cat_colname] = 2
        df.loc[df[cat_colname] == 'Lower secondary', cat_colname] = 3
        df.loc[df[cat_colname] == 'Academic degree', cat_colname] = 4
        df.loc[df[cat_colname] == 'неизвестно', cat_colname] = 5

        # FAMILY_STATUS
        cat_colname = 'FAMILY_STATUS'
        df[cat_colname] = df[cat_colname].replace(to_replace = np.nan, value = 'неизвест

        df.loc[df[cat_colname] == 'Married', cat_colname] = 0
        df.loc[df[cat_colname] == 'Single / not married', cat_colname] = 1
        df.loc[df[cat_colname] == 'Civil marriage', cat_colname] = 2
        df.loc[df[cat_colname] == 'Separated', cat_colname] = 3
        df.loc[df[cat_colname] == 'Widow', cat_colname] = 4
        df.loc[df[cat_colname] == 'Separated', cat_colname] = 5
        df.loc[df[cat_colname] == 'Unknown', cat_colname] = 6
        df.loc[df[cat_colname] == 'неизвестно', cat_colname] = 7

        # CREDIT_DEBT
        cat_colname = 'CREDIT_DEBT'
        df[cat_colname] = df[cat_colname].replace(to_replace = np.nan, value = 0)

        # Обработка категорий
        for colname in ['NAME_CONTRACT_TYPE', 'GENDER', 'EDUCATION_LEVEL', 'FAMILY_STATU
            df[colname] = df[colname].astype('int8')

    return df
```

Data Description

Для построения модели в данном соревновании, сначала нужно будет собрать выборку для обучения модели. Формат соревнования очень похож на то, как в промышленности Data Scinetist'ы строят алгоритмы: сначала нужно провести анализ данных, собрать выборку и после этого строить модели. В соревновании представлены 4 типы источника данных, которые могут быть интерпретированы как

таблицы в базе данных. Некоторые источники данных уже готовы для моделирования, представлены в агрерированном виде. Другие источники данных требуется представить в удобном для модели виде.

Описание источников данных:

- train.csv - пары "заявка - целевая переменная", для этой выборки нужно собрать признаки и обучить модель;
- test.csv - пары "заявки - прогнозное значение", для этой выборки нужно собрать признаки и построить прогнозы;
- bki.csv - данные БКИ о предыдущих кредитах клиента;
- client_profile.csv - клиентский профиль, некоторые знания, которые есть у компании о клиенте;
- payments.csv - история платежей клиента;
- applications_history.csv - история предыдущих заявок клиента.

B [721]:
```
base_path = "/kaggle/input/geekbrains-competitive-data-analysis/"

TRAIN_DATASET_PATH = base_path + 'train.csv'
TEST_DATASET_PATH = base_path + 'test.csv'
bki_DATASET_PATH = base_path + 'bki.csv'
applications_history_DATASET_PATH = base_path + 'applications_history.csv'
client_profile_DATASET_PATH = base_path + 'client_profile.csv'
payments_DATASET_PATH = base_path + 'payments.csv'
sample_submit_DATASET_PATH = base_path + 'sample_submit.csv'

ID_COLUMN = "APPLICATION_NUMBER"
ID_COLUMN_PR = "PREV_APPLICATION_NUMBER"
TARGET = "TARGET"
```

Загрузка данных

Base Tables

B [722]:
```
# 1. train.csv - пары "заявка - целевая переменная", для этой выборки нужно собрать приз
train = pd.read_csv(TRAIN_DATASET_PATH)
print(train.shape)
train.head(2)
```

```
(110093, 3)
```

Out[722]:

	APPLICATION_NUMBER	TARGET	NAME_CONTRACT_TYPE
0	123687442	0	Cash
1	123597908	1	Cash

B [723]:
```
# 2. test.csv - пары "заявки - прогнозное значение", для этой выборки нужно собрать приз
test = pd.read_csv(TEST_DATASET_PATH)
print(test.shape)
test.head(2)
```

```
(165141, 2)
```

Out[723]:

	APPLICATION_NUMBER	NAME_CONTRACT_TYPE
0	123724268	Cash
1	123456549	Cash

B [724]:
```
data = pd.concat([train, test], axis = 0)
data.reset_index(drop=True)
print(data.shape)
data.head(n=2)
```

```
(275234, 3)
```

Out[724]:

	APPLICATION_NUMBER	TARGET	NAME_CONTRACT_TYPE
0	123687442	0.0	Cash
1	123597908	1.0	Cash

client_profile

B [725]:
```
# 4. client_profile.csv - клиентский профиль
client_profile = pd.read_csv(client_profile_DATASET_PATH)
df_client_profile = create_client_profile_features(client_profile)
```

```
/opt/conda/lib/python3.7/site-packages/numpy/lib/nanfunctions.py:1114: RuntimeWarning:
All-NaN slice encountered
  overwrite_input=overwrite_input)
```

B [726]:
```
data = data.merge(
    df_client_profile, how="left", on="APPLICATION_NUMBER"
)
data.head(2)
```

Out[726]:

	APPLICATION_NUMBER	TARGET	NAME_CONTRACT_TYPE	GENDER	CHILDRENS	TOTAL_SALARY	AMC
0	123687442	0.0	Cash	M	1.0	157500.0	
1	123597908	1.0	Cash	NaN	NaN	NaN	

2 rows × 54 columns

bki

B [727]:
```
# 3. bki.csv - данные БКИ о предыдущих кредитах клиента;
df_bki = pd.read_csv(bki_DATASET_PATH)
print(df_bki.shape)
df_bki.head(2)
```

```
(945234, 17)
```

Out[727]:

	APPLICATION_NUMBER	BUREAU_ID	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_
0	123538884	5223613	Active	currency 1	718.0	
1	123436670	6207544	Closed	currency 1	696.0	

B [728]:
```
#df_c = df_bki[["APPLICATION_NUMBER", "AMT_CREDIT_MAX_OVERDUE"]]
df_c = df_bki[["APPLICATION_NUMBER", "AMT_CREDIT_MAX_OVERDUE", "AMT_CREDIT_SUM_LIMIT", "
df = df_bki.groupby('APPLICATION_NUMBER').sum()
df['CREDIT_DEBT'] = 1
print(df_bki.shape)
#print(df.info())
```

```
(945234, 17)
```

B [729]:
```
df.tail()
```

Out[729]:

APPLICATION_NUMBER	BUREAU_ID	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DAY
123779588	59775214	18372.0	0	14640.0	
123779589	6913730	1002.0	0	272.0	
123779592	11413155	1632.0	0	1500.0	
123779593	6723687	1104.0	0	859.0	
123779594	31130714	6286.0	0	5768.0	

B [730]:
```
# df_bki_columns = df_bki[["APPLICATION_NUMBER", "AMT_CREDIT_MAX_OVERDUE", "AMT_CREDIT_S
# df = df_bki.groupby('APPLICATION_NUMBER').sum()
# df['CREDIT_DEBT'] = 1
# df.head(2)
# df.info()
```

B [731]:
```
#df.drop('AMT_CREDIT_MAX_OVERDUE', axis=1, inplace=True)
#df.head(2)
```

B [732]:
```
print(data.shape)
data = data.merge(
    df, how="left", on="APPLICATION_NUMBER"
)
# print(data.shape)
# print(data.info())
```

```
(275234, 54)
```

baseline

B [733]:
```
# mask = data["TARGET"].isnull()
# features_to_drop = ["APPLICATION_NUMBER", "TARGET"]

# train, test = data.loc[~mask], data.loc[mask]

# target, test_id = train["TARGET"], test["APPLICATION_NUMBER"]
# train = train.drop(features_to_drop, axis=1)
# test = test.drop(features_to_drop, axis=1)

# # categorial = train.dtypes[train.dtypes == "category"].index
# categorial = train.dtypes[train.dtypes == "object"].index

# # categorical = list(set(categorial + categorial_1))
# numerical = list(set(train.columns) - set(categorial))

# ###
# train = new_catigirical_features(train)
# test = new_catigirical_features(test)
# ###

# train = train.replace(np.inf, np.nan)
# train = train.replace(-np.inf, np.nan)

# print(categorial)
```

KFold

CatBoost

B [734]:
```
cb_params = {
    "n_estimators": 2000,
    "learning_rate": 0.01,
    "loss_function": "Logloss",
    "eval_metric": "AUC",
    "task_type": "CPU",
    "max_bin": 20,
    "l2_leaf_reg": 10,
    "verbose": 20,
    "max_depth": 6,
    "early_stopping_rounds": 50,
    "thread_count": 6,
    "random_seed": 42
}
```

B [735]:
```
# cv = KFold(n_splits=7, random_state=1234123, shuffle=True)
# estimators, oof_preds = catboost_cross_validation(
#     params=cb_params, X=train, y=target, cv=cv, categorical=categorial
# )
```

LightGBM

В [736]:
```
mask = data["TARGET"].isnull()
features_to_drop = ["APPLICATION_NUMBER", "TARGET"]
categorial = data.dtypes[data.dtypes == "object"].index
numerical = list(set(train.columns) - set(categorial))

for feature in categorial:
    encoder = LabelEncoder()
    data[feature] = encoder.fit_transform(data[feature].astype("str").fillna("NA"))

train, test = data.loc[~mask], data.loc[mask]
target, train_id, test_id = train["TARGET"], train["APPLICATION_NUMBER"], test["APPLICAT
train = train.drop(features_to_drop, axis=1)
test = test.drop(features_to_drop, axis=1)


###
train = new_catigirical_features(train)
test = new_catigirical_features(test)
###

train = train.replace(np.inf, np.nan)
train = train.replace(-np.inf, np.nan)
```

В [737]:
```
#print(test.info())
#test.head(2)
# print(train.info())
# train.head(2)
```

В [738]:
```
lgb_params = {
    "boosting_type": "gbdt",
    "n_estimators": 10000,  # число деревьев
    "learning_rate": 0.05134,
    "num_leaves": 54,
    "max_depth": 10,  # глубина дерева
    "subsample_for_bin": 240000,
    "reg_alpha": 0.436193,
    "reg_lambda": 0.479169,  # регуляризация (то что используется при F2-штрафе (1:15:10
    "colsample_bytree": 0.508716,
    "min_split_gain": 0.024766,
    "subsample": 0.7,
    "is_unbalance": False,
    "random_state": 27,
    "silent": -1,
    "verbose": 1,
}
```

В [739]:
```
categorical=list(categorial)
categorical.append('CREDIT_DEBT')
categorical
```

Out[739]:
```
['NAME_CONTRACT_TYPE',
 'GENDER',
 'EDUCATION_LEVEL',
 'FAMILY_STATUS',
 'CREDIT_DEBT']
```

B [740]:
```
cv = KFold(n_splits=15, random_state=1234123, shuffle=True)
estimators, oof_preds = lightgbm_cross_validation(
    params=lgb_params, X=train, y=target, cv=cv, categorical=categorical
)
```

```
[50]    dtrain's auc: 0.773506  dtrain's binary_logloss: 0.242464       dvalid's au
c: 0.735362     dvalid's binary_logloss: 0.25435
[75]    dtrain's auc: 0.794243  dtrain's binary_logloss: 0.235646       dvalid's au
c: 0.739678     dvalid's binary_logloss: 0.252923
[100]   dtrain's auc: 0.811365  dtrain's binary_logloss: 0.230081       dvalid's au
c: 0.741358     dvalid's binary_logloss: 0.252394
[125]   dtrain's auc: 0.827166  dtrain's binary_logloss: 0.225331       dvalid's au
c: 0.740812     dvalid's binary_logloss: 0.252483
[150]   dtrain's auc: 0.839793  dtrain's binary_logloss: 0.221181       dvalid's au
c: 0.73984      dvalid's binary_logloss: 0.252662
[175]   dtrain's auc: 0.851351  dtrain's binary_logloss: 0.217128       dvalid's au
c: 0.739186     dvalid's binary_logloss: 0.252651
Early stopping, best iteration is:
[90]    dtrain's auc: 0.804559  dtrain's binary_logloss: 0.232248       dvalid's au
c: 0.74153      dvalid's binary_logloss: 0.252507
Fold 15, Valid score = 0.74153
Score by each fold: [0.71123, 0.73201, 0.72912, 0.73251, 0.7239, 0.73004, 0.73276,
0.73986, 0.73075, 0.73995, 0.71993, 0.71651, 0.71078, 0.73748, 0.74153]
================================================================
```

B [741]:
```
print(data.shape)
data.loc[data['CREDIT_DEBT'] == 1].shape
```

```
(275234, 68)
```

Out[741]:
```
(210977, 68)
```

B [742]:
```
categorial
```

Out[742]:
```
Index(['NAME_CONTRACT_TYPE', 'GENDER', 'EDUCATION_LEVEL', 'FAMILY_STATUS'], dtype='obje
ct')
```

LightGBM Sklearn-API

B [743]:
```
params = {
    "boosting_type": "gbdt",
    "objective": "binary",
    "metric": "auc",
    "learning_rate": 0.01,
    "n_estimators": 10000,  # число деревьев
    "reg_lambda": 100,  # регуляризация (то что используется при F2-штрафе (1:15:10))
    #"max_depth": 4,  # глубина дерева
    "n_jobs": 6,
    "seed": 27
}
```

B [744]:
```
categorial = list(categorial)
```

B [745]:
```
# cv = KFold(n_splits=5, random_state=1234123, shuffle=True)
# estimators, oof_preds = lightgbm_cross_validation(
#     params=params, X=train, y=target, cv=cv, categorical=categorial
# )
```

B [746]:
```
oof_score = roc_auc_score(
    target, oof_preds
)
print(f"OOF-score = {round(oof_score,5)}")
# Score by each fold: [0.71258, 0.73625, 0.72642, 0.73016, 0.731, 0.73805, 0.73708, 0.71
# OOF-score = 0.72733
# Score by each fold: [0.71352, 0.73414, 0.72961, 0.73237, 0.72938, 0.73352, 0.73783, 0.
# OOF-score = 0.72727
# Score by each fold: [0.71123, 0.73201, 0.72912, 0.73251, 0.7239, 0.73004, 0.73276, 0.7
# OOF-score = 0.72785
```

```
OOF-score = 0.72785
```

Подготовка прогноза

B [747]:
```
y_pred = np.zeros(test.shape[0])
# test[numerical] = test[numerical].astype(float)
# test[categorial] = test[categorial].astype(int)

for estimator in estimators:
    y_pred += estimator.predict_proba(test)[:, 1] / len(estimators)
```

B [748]:
```
submission = pd.DataFrame({
    "APPLICATION_NUMBER": test_id,
    "TARGET": y_pred / cv.n_splits
})
```

B [749]:
```
# print(test_id.shape)
# test_id
# estimators
```

B [750]:
```
submission.head(10)
```

Out[750]:

	APPLICATION_NUMBER	TARGET
110093	123724268	0.004655
110094	123456549	0.017125
110095	123428178	0.010461
110096	123619984	0.005617
110097	123671104	0.001215
110098	123632747	0.001666
110099	123728867	0.004418
110100	123459592	0.007612
110101	123595888	0.000651
110102	123480224	0.001838

B [751]:
```
submission.to_csv("ILSokovnin_predictions.csv", index=False)
```

B [ ]: