wrangling_all_1202

December 8, 2024

[28]:
```python
import sys
from pathlib import Path
import os
import gc
import datetime
from glob import glob
import numpy as np
import pandas as pd
import polars as pl

import matplotlib.pyplot as plt
import joblib

from sklearn.model_selection import StratifiedGroupKFold
from sklearn.metrics import roc_auc_score

import warnings
warnings.filterwarnings('ignore')

ROOT = '/Users/wuqianran/Desktop/bigdata_finalproject/final'

from sklearn.model_selection import TimeSeriesSplit, GroupKFold,␣
↪StratifiedGroupKFold
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.metrics import roc_auc_score
import lightgbm as lgb

# from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import OrdinalEncoder
from sklearn.impute import KNNImputer
```

[29]:
```python
class Pipeline:

    def set_table_dtypes(df):
        for col in df.columns:
            if col in ["case_id", "WEEK_NUM", "num_group1", "num_group2"]:
                df = df.with_columns(pl.col(col).cast(pl.Int64))
```

```
            elif col in ["date_decision"]:
                df = df.with_columns(pl.col(col).cast(pl.Date))
            elif col[-1] in ("P", "A"):
                df = df.with_columns(pl.col(col).cast(pl.Float64))
            elif col[-1] in ("M",):
                df = df.with_columns(pl.col(col).cast(pl.String))
            elif col[-1] in ("D",):
                df = df.with_columns(pl.col(col).cast(pl.Date))
        return df

    def handle_dates(df):
        for col in df.columns:
            if col[-1] in ("D",):
                df = df.with_columns(pl.col(col) - pl.col("date_decision"))  #!!
↪?
                df = df.with_columns(pl.col(col).dt.total_days()) # t - t-1
        df = df.drop("date_decision", "MONTH")
        return df

    def filter_cols(df):
        for col in df.columns:
            if col not in ["target", "case_id", "WEEK_NUM"]:
                isnull = df[col].is_null().mean()
                if isnull > 0.95:
                    df = df.drop(col)

        for col in df.columns:
            if (col not in ["target", "case_id", "WEEK_NUM"]) & (df[col].dtype␣
↪== pl.String):
                freq = df[col].n_unique()
                if (freq == 1) | (freq > 200):
                    df = df.drop(col)

        return df



class Aggregator:
    # Please add or subtract features yourself, be aware that too many features␣
↪will take up too much space.
    def num_expr(df):
        cols = [col for col in df.columns if col[-1] in ("P", "A")]
        expr_max = [pl.max(col).alias(f"max_{col}") for col in cols]

        expr_last = [pl.last(col).alias(f"last_{col}") for col in cols]
        # expr_first = [pl.first(col).alias(f"first_{col}") for col in cols]
        expr_mean = [pl.mean(col).alias(f"mean_{col}") for col in cols]
```

```
        expr_median = [pl.median(col).alias(f"median_{col}") for col in cols]
        expr_var = [pl.var(col).alias(f"var_{col}") for col in cols]

        return expr_max  + expr_mean

    def date_expr(df):
        cols = [col for col in df.columns if col[-1] in ("D")]
        expr_max = [pl.max(col).alias(f"max_{col}") for col in cols]
        # expr_min = [pl.min(col).alias(f"min_{col}") for col in cols]
        expr_last = [pl.last(col).alias(f"last_{col}") for col in cols]
        # expr_first = [pl.first(col).alias(f"first_{col}") for col in cols]
        expr_mean = [pl.mean(col).alias(f"mean_{col}") for col in cols]
        expr_median = [pl.median(col).alias(f"median_{col}") for col in cols]

        return expr_max  + expr_mean

    def str_expr(df):
        cols = [col for col in df.columns if col[-1] in ("M",)]
        expr_max = [pl.max(col).alias(f"max_{col}") for col in cols]
        # expr_min = [pl.min(col).alias(f"min_{col}") for col in cols]
        expr_last = [pl.last(col).alias(f"last_{col}") for col in cols]
        # expr_first = [pl.first(col).alias(f"first_{col}") for col in cols]
        # expr_count = [pl.count(col).alias(f"count_{col}") for col in cols]
        expr_mean = [pl.mean(col).alias(f"mean_{col}") for col in cols]
        return expr_max  + expr_mean

    def other_expr(df):
        cols = [col for col in df.columns if col[-1] in ("T", "L")]
        expr_max = [pl.max(col).alias(f"max_{col}") for col in cols]
        # expr_min = [pl.min(col).alias(f"min_{col}") for col in cols]
        expr_last = [pl.last(col).alias(f"last_{col}") for col in cols]
        # expr_first = [pl.first(col).alias(f"first_{col}") for col in cols]
        expr_mean = [pl.mean(col).alias(f"mean_{col}") for col in cols]
        return expr_max  + expr_mean

    def count_expr(df):
        cols = [col for col in df.columns if "num_group" in col]
        expr_max = [pl.max(col).alias(f"max_{col}") for col in cols]
        # expr_min = [pl.min(col).alias(f"min_{col}") for col in cols]
        expr_last = [pl.last(col).alias(f"last_{col}") for col in cols]
        # expr_first = [pl.first(col).alias(f"first_{col}") for col in cols]
        expr_mean = [pl.mean(col).alias(f"mean_{col}") for col in cols]
        return expr_max + expr_mean

    def get_exprs(df):
        exprs = Aggregator.num_expr(df) + \
                Aggregator.date_expr(df) + \
```

```
                Aggregator.str_expr(df) + \
                Aggregator.other_expr(df) + \
                Aggregator.count_expr(df)

        return exprs
```

[30]:
```
def read_file(path, depth=None):
    df = pl.read_parquet(path)
    df = df.pipe(Pipeline.set_table_dtypes)
    if depth in [1,2]:
        df = df.group_by("case_id").agg(Aggregator.get_exprs(df))
    return df


def read_files(regex_path, depth=None):
    chunks = []

    for path in glob(str(regex_path)):
        df = pl.read_parquet(path)
        df = df.pipe(Pipeline.set_table_dtypes)
        if depth in [1, 2]:
            df = df.group_by("case_id").agg(Aggregator.get_exprs(df))
        chunks.append(df)

    df = pl.concat(chunks, how="vertical_relaxed")
    df = df.unique(subset=["case_id"])
    return df
```

[31]:
```
def feature_eng(df_base, depth_0, depth_1, depth_2):
    df_base = (
        df_base
        .with_columns(
            month_decision = pl.col("date_decision").dt.month(),
            weekday_decision = pl.col("date_decision").dt.weekday(),
        )
    )
    for i, df in enumerate(depth_0 + depth_1 + depth_2):
        df_base = df_base.join(df, how="left", on="case_id", suffix=f"_{i}")
    df_base = df_base.pipe(Pipeline.handle_dates)
    return df_base


def to_pandas(df_data, cat_cols=None):
    df_data = df_data.to_pandas()
    if cat_cols is None:
        cat_cols = list(df_data.select_dtypes("object").columns)
    df_data[cat_cols] = df_data[cat_cols].astype("category")
```

```
    return df_data, cat_cols
```

[32]:
```
def reduce_mem_usage(df):
    """ iterate through all the columns of a dataframe and modify the data type
        to reduce memory usage.
    """
    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 str(col_type)=="category":
            continue

        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.float16).min and c_max < np.finfo(np.
 ↪float16).max:
                    df[col] = df[col].astype(np.float16)
                elif 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:
            continue
    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
```

[33]:
```
%%time

ROOT             = Path(ROOT)

TRAIN_DIR        = ROOT / "parquet_files" / "train"
TEST_DIR         = ROOT / "parquet_files" / "test"

data_store = {
    "df_base": read_file(TRAIN_DIR / "train_base.parquet"),
    "depth_0": [
        read_file(TRAIN_DIR / "train_static_cb_0.parquet"),
        read_files(TRAIN_DIR / "train_static_0_*.parquet"),
    ],
    "depth_1": [
        read_files(TRAIN_DIR / "train_applprev_1_*.parquet", 1),
        read_file(TRAIN_DIR / "train_tax_registry_a_1.parquet", 1),
        read_file(TRAIN_DIR / "train_tax_registry_b_1.parquet", 1),
        read_file(TRAIN_DIR / "train_tax_registry_c_1.parquet", 1),
        read_files(TRAIN_DIR / "train_credit_bureau_a_1_*.parquet", 1),
        read_file(TRAIN_DIR / "train_credit_bureau_b_1.parquet", 1),
        read_file(TRAIN_DIR / "train_other_1.parquet", 1),
        read_file(TRAIN_DIR / "train_person_1.parquet", 1),
        read_file(TRAIN_DIR / "train_deposit_1.parquet", 1),
        read_file(TRAIN_DIR / "train_debitcard_1.parquet", 1),
    ],
    "depth_2": [
        read_file(TRAIN_DIR / "train_credit_bureau_b_2.parquet", 2),
        read_files(TRAIN_DIR / "train_credit_bureau_a_2_*.parquet", 2),
    ]
}
```

```
CPU times: user 2min 5s, sys: 1min 59s, total: 4min 5s
Wall time: 1min 38s
```

[34]:
```
%%time

df_train = feature_eng(**data_store)
print("train data shape:\t", df_train.shape)
del data_store
df_train = df_train.pipe(Pipeline.filter_cols)
gc.collect()
```

```
train data shape:	 (1526659, 720)
CPU times: user 14.2 s, sys: 8.55 s, total: 22.7 s
Wall time: 15.1 s
```

[34]: 3869

[35]:
```
!pip install --upgrade polars
```

```
Requirement already satisfied: polars in ./new_venv/lib/python3.9/site-packages
(1.16.0)

[notice] A new release of pip is
available: 23.2.1 -> 24.3.1
[notice] To update, run:
pip install --upgrade pip
```

[36]:
```
cnt_encoding_cols = df_train.select(pl.selectors.by_dtype([pl.String, pl.
 ↪Boolean, pl.Categorical])).columns

mappings = {}
for col in cnt_encoding_cols:
    mappings[col] = df_train.group_by(col).len()

df_train_lazy = df_train.select(mappings.keys()).lazy()
# df_train_lazy = pl.LazyFrame(df_train.select('case_id'))

for col, mapping in mappings.items():
    remapping = {category: count for category, count in mapping.rows()}
    remapping[None] = -2
    expr = pl.col(col).replace(
                remapping,
                default=-1,
            )
    df_train_lazy = df_train_lazy.with_columns(expr.alias(col + '_cnt'))
    del col, mapping, remapping
    gc.collect()

del mappings
transformed_train = df_train_lazy.collect()

df_train = pl.concat([df_train, transformed_train.select("^*cnt$")],␣
 ↪how='horizontal')
del transformed_train, cnt_encoding_cols

gc.collect()
```

[36]: 0

[37]:
```
df_train, cat_cols = to_pandas(df_train)
df_train = reduce_mem_usage(df_train)
print("train data shape:\t", df_train.shape)
```

```python
nums=df_train.select_dtypes(exclude='category').columns
from itertools import combinations, permutations
#df_train=df_train[nums]
nans_df = df_train[nums].isna()
nans_groups={}
for col in nums:
    cur_group = nans_df[col].sum()
    try:
        nans_groups[cur_group].append(col)
    except:
        nans_groups[cur_group]=[col]
del nans_df; x=gc.collect()

def reduce_group(grps):
    use = []
    for g in grps:
        mx = 0; vx = g[0]
        for gg in g:
            n = df_train[gg].nunique()
            if n>mx:
                mx = n
                vx = gg
            #print(str(gg)+'-'+str(n),',  ',end='')
        use.append(vx)
        #print()
    print('Use these',use)
    return use

def group_columns_by_correlation(matrix, threshold=0.8):
    #
    correlation_matrix = matrix.corr()

    #
    groups = []
    remaining_cols = list(matrix.columns)
    while remaining_cols:
        col = remaining_cols.pop(0)
        group = [col]
        correlated_cols = [col]
        for c in remaining_cols:
            if correlation_matrix.loc[col, c] >= threshold:
                group.append(c)
                correlated_cols.append(c)
        groups.append(group)
        remaining_cols = [c for c in remaining_cols if c not in correlated_cols]

    return groups
```

```
uses=[]
for k,v in nans_groups.items():
    if len(v)>1:
            Vs = nans_groups[k]
            #cross_features=list(combinations(Vs, 2))
            #make_corr(Vs)
            grps= group_columns_by_correlation(df_train[Vs], threshold=0.8)
            use=reduce_group(grps)
            uses=uses+use
            #make_corr(use)
    else:
        uses=uses+v
    print('####### NAN count =',k)
print(uses)
print(len(uses))
uses=uses+list(df_train.select_dtypes(include='category').columns)
print(len(uses))
df_train=df_train[uses]
# df_train.drop(['requesttype_4525192L_cnt','max_empl_employedtotal_800L_cnt',␣
 ↪'max_empl_industry_691L_cnt'], axis=1, inplace=True)
```

```
Memory usage of dataframe is 5809.23 MB
Memory usage after optimization is: 2137.36 MB
Decreased by 63.2%
train data shape:        (1526659, 564)
Use these ['case_id', 'WEEK_NUM', 'target', 'month_decision',
'weekday_decision', 'credamount_770A', 'applicationcnt_361L',
'applications30d_658L', 'applicationscnt_1086L', 'applicationscnt_464L',
'applicationscnt_867L', 'clientscnt_1022L', 'clientscnt_100L',
'clientscnt_1071L', 'clientscnt_1130L', 'clientscnt_157L', 'clientscnt_257L',
'clientscnt_304L', 'clientscnt_360L', 'clientscnt_493L', 'clientscnt_533L',
'clientscnt_887L', 'clientscnt_946L', 'deferredmnthsnum_166L',
'disbursedcredamount_1113A', 'downpmt_116A', 'homephncnt_628L',
'isbidproduct_1095L', 'mobilephncnt_593L', 'numactivecreds_622L',
'numactivecredschannel_414L', 'numactiverelcontr_750L', 'numcontrs3months_479L',
'numnotactivated_1143L', 'numpmtchanneldd_318L', 'numrejects9m_859L',
'sellerplacecnt_915L', 'max_mainoccupationinc_384A', 'max_birth_259D',
'mean_persontype_1072L', 'description_5085714M_cnt', 'education_1103M_cnt',
'maritalst_893M_cnt', 'maritalst_385M_cnt', 'requesttype_4525192L_cnt',
'bankacctype_710L_cnt', 'cardtype_51L_cnt', 'credtype_322L_cnt',
'disbursementtype_67L_cnt', 'equalitydataagreement_891L_cnt',
'isbidproduct_1095L_cnt', 'lastapprcommoditycat_1041M_cnt',
'lastcancelreason_561M_cnt', 'lastrejectcommoditycat_161M_cnt',
'lastrejectcommodtypec_5251769M_cnt', 'lastrejectreason_759M_cnt',
'lastst_736L_cnt', 'paytype1st_925L_cnt', 'twobodfilling_608L_cnt',
'typesuite_864L_cnt', 'max_cancelreason_3545846M_cnt',
```

```
'max_education_1138M_cnt', 'max_postype_4733339M_cnt',
'max_credacc_status_367L_cnt', 'max_credtype_587L_cnt',
'max_familystate_726L_cnt', 'max_isbidproduct_390L_cnt',
'max_isdebitcard_527L_cnt', 'max_status_219L_cnt',
'max_collaterals_typeofguarante_669M_cnt', 'max_classificationofcontr_400M_cnt',
'max_contractst_545M_cnt', 'max_contractst_964M_cnt',
'max_financialinstitution_382M_cnt', 'max_financialinstitution_591M_cnt',
'max_purposeofcred_874M_cnt', 'max_subjectrole_93M_cnt',
'max_education_927M_cnt', 'max_empladdr_district_926M_cnt',
'max_language1_981M_cnt', 'max_contaddr_matchlist_1032L_cnt',
'max_contaddr_smempladdr_334L_cnt', 'max_empl_employedtotal_800L_cnt',
'max_empl_industry_691L_cnt', 'max_familystate_447L_cnt',
'max_housetype_905L_cnt', 'max_incometype_1044T_cnt', 'max_role_1084L_cnt',
'max_safeguarantyflag_411L_cnt', 'max_sex_738L_cnt', 'max_type_25L_cnt',
'max_collaterals_typeofguarante_359M_cnt']
####### NAN count = 0
####### NAN count = 1389663
Use these ['assignmentdate_4527235D', 'pmtaverage_4527227A',
'pmtcount_4527229L']
####### NAN count = 1411681
####### NAN count = 918788
Use these ['mean_contractsum_5085717L']
####### NAN count = 1369330
Use these ['dateofbirth_337D', 'days180_256L', 'days30_165L', 'days360_512L',
'firstquarter_103L', 'fourthquarter_440L', 'secondquarter_766L',
'thirdquarter_1082L', 'max_debtoutstand_525A', 'max_debtoverdue_47A',
'max_refreshdate_3813885D', 'mean_refreshdate_3813885D']
####### NAN count = 140968
####### NAN count = 1383070
####### NAN count = 1380253
Use these ['pmtscount_423L', 'pmtssum_45A']
####### NAN count = 954021
####### NAN count = 806659
####### NAN count = 866332
####### NAN count = 1301747
####### NAN count = 418178
Use these ['amtinstpaidbefduel24m_4187115A', 'numinstlswithdpd5_4187116L']
####### NAN count = 561124
Use these ['annuitynextmonth_57A', 'currdebt_22A', 'currdebtcredtyperange_828A',
'numinstls_657L', 'totalsettled_863A']
####### NAN count = 4
Use these ['mindbddpdlast24m_3658935P']
####### NAN count = 613202
####### NAN count = 948244
Use these ['mindbdtollast24m_4525191P']
####### NAN count = 972827
####### NAN count = 467175
Use these ['avginstallast24m_3658937A', 'maxinstallast24m_3658928A']
```

```
####### NAN count = 624875
####### NAN count = 1364150
####### NAN count = 757006
####### NAN count = 841181
####### NAN count = 1026987
####### NAN count = 455190
####### NAN count = 460822
Use these ['commnoinclast6m_3546845L', 'maxdpdfrom6mto36m_3546853P']
####### NAN count = 343375
####### NAN count = 833735
####### NAN count = 1392841
####### NAN count = 887659
Use these ['daysoverduetolerancedd_3976961L', 'numinsttopaygr_769L']
####### NAN count = 452594
####### NAN count = 977119
Use these ['eir_270L']
####### NAN count = 190833
####### NAN count = 859214
####### NAN count = 482103
####### NAN count = 1334357
####### NAN count = 453587
Use these ['lastapplicationdate_877D', 'mean_creationdate_885D',
'mean_isbidproduct_390L', 'max_num_group1']
####### NAN count = 305137
Use these ['lastapprcredamount_781A', 'lastapprdate_640D']
####### NAN count = 442041
####### NAN count = 977975
Use these ['lastrejectcredamount_222A', 'lastrejectdate_50D']
####### NAN count = 769046
####### NAN count = 511255
Use these ['mastercontrelectronic_519L', 'mastercontrexist_109L',
'maxannuity_159A', 'maxdebt4_972A', 'maxdpdlast24m_143P', 'maxdpdlast3m_392P',
'maxdpdtolerance_374P']
####### NAN count = 306019
####### NAN count = 960953
####### NAN count = 705504
####### NAN count = 876276
####### NAN count = 826000
####### NAN count = 829402
####### NAN count = 1032856
####### NAN count = 766958
####### NAN count = 1129330
Use these ['numinstpaidearly_338L', 'numinstpaidearly5d_1087L',
'numinstpaidlate1d_3546852L']
####### NAN count = 452593
####### NAN count = 455081
Use these ['numinstlsallpaid_934L']
####### NAN count = 445669
```

```
Use these ['numinstlswithdpd10_728L', 'numinstlswithoutdpd_562L']
####### NAN count = 456495
Use these ['numinstpaid_4499208L']
####### NAN count = 847191
####### NAN count = 446983
Use these ['numinstregularpaidest_4493210L', 'numinstpaidearly5dest_4493211L',
'sumoutstandtotalest_4493215A']
####### NAN count = 840646
####### NAN count = 669186
####### NAN count = 455612
Use these ['pctinstlsallpaidearl3d_427L', 'pctinstlsallpaidlate1d_3546856L']
####### NAN count = 458738
####### NAN count = 461362
####### NAN count = 459827
####### NAN count = 460079
####### NAN count = 44954
####### NAN count = 78526
####### NAN count = 131888
####### NAN count = 181122
####### NAN count = 223240
####### NAN count = 445320
####### NAN count = 3
####### NAN count = 1174211
####### NAN count = 1374886
Use these ['mean_actualdpd_943P']
####### NAN count = 305154
Use these ['max_annuity_853A', 'mean_annuity_853A']
####### NAN count = 308739
Use these ['mean_credacc_actualbalance_314A', 'mean_credacc_maxhisbal_375A',
'mean_credacc_minhisbal_90A', 'mean_credacc_transactions_402L']
####### NAN count = 1273086
Use these ['max_credacc_credlmt_575A', 'max_credamount_590A',
'max_downpmt_134A', 'mean_credacc_credlmt_575A', 'mean_credamount_590A',
'mean_downpmt_134A']
####### NAN count = 307441
Use these ['max_currdebt_94A', 'mean_currdebt_94A']
####### NAN count = 419006
Use these ['max_mainoccupationinc_437A', 'mean_mainoccupationinc_437A']
####### NAN count = 306361
Use these ['mean_maxdpdtolerance_577P']
####### NAN count = 450969
Use these ['max_outstandingdebt_522A', 'mean_outstandingdebt_522A']
####### NAN count = 420383
Use these ['mean_revolvingaccount_394A']
####### NAN count = 1273082
Use these ['max_approvaldate_319D', 'mean_approvaldate_319D']
####### NAN count = 442999
Use these ['max_dateactivated_425D', 'mean_dateactivated_425D']
```

```
####### NAN count = 454678
Use these ['max_dtlastpmt_581D', 'mean_dtlastpmt_581D']
####### NAN count = 703840
Use these ['max_dtlastpmtallstes_3545839D', 'mean_dtlastpmtallstes_3545839D']
####### NAN count = 548987
Use these ['max_employedfrom_700D']
####### NAN count = 559169
Use these ['max_firstnonzeroinstldate_307D', 'mean_firstnonzeroinstldate_307D']
####### NAN count = 334873
Use these ['mean_byoccupationinc_3656910L']
####### NAN count = 961606
Use these ['mean_childnum_21L']
####### NAN count = 552766
Use these ['max_pmtnum_8L', 'mean_pmtnum_8L']
####### NAN count = 321446
####### NAN count = 1182972
Use these ['max_amount_4527230A', 'max_recorddate_4527225D', 'max_num_group1_3']
####### NAN count = 1068725
Use these ['mean_amount_4917619A', 'max_deductiondate_4917603D',
'mean_deductiondate_4917603D', 'max_num_group1_4']
####### NAN count = 1375927
Use these ['max_pmtamount_36A', 'max_processingdate_168D',
'mean_processingdate_168D', 'max_num_group1_5']
####### NAN count = 1044394
Use these ['mean_credlmt_230A']
####### NAN count = 1036944
Use these ['mean_credlmt_935A']
####### NAN count = 603001
Use these ['mean_pmts_dpd_1073P', 'mean_dpdmaxdatemonth_89T',
'mean_dpdmaxdateyear_596T']
####### NAN count = 263166
Use these ['max_pmts_dpd_303P', 'mean_dpdmax_757P', 'max_dpdmaxdatemonth_442T',
'max_dpdmaxdateyear_896T', 'mean_dpdmaxdatemonth_442T',
'mean_dpdmaxdateyear_896T', 'mean_pmts_dpd_303P']
####### NAN count = 514070
Use these ['mean_instlamount_768A']
####### NAN count = 606920
Use these ['mean_instlamount_852A']
####### NAN count = 1136162
Use these ['mean_monthlyinstlamount_332A']
####### NAN count = 263233
Use these ['max_monthlyinstlamount_674A', 'mean_monthlyinstlamount_674A']
####### NAN count = 517511
Use these ['mean_outstandingamount_354A']
####### NAN count = 545885
Use these ['mean_outstandingamount_362A']
####### NAN count = 636453
Use these ['mean_overdueamount_31A']
```

```
####### NAN count = 512650
Use these ['mean_overdueamount_659A', 'mean_numberofoverdueinstls_725L']
####### NAN count = 263171
Use these ['mean_overdueamountmax2_14A', 'mean_totaloutstanddebtvalue_39A',
'mean_dateofcredend_289D', 'mean_dateofcredstart_739D', 'max_lastupdate_1112D',
'mean_lastupdate_1112D', 'mean_numberofcontrsvalue_258L',
'mean_numberofoverdueinstlmax_1039L', 'mean_overdueamountmaxdatemonth_365T',
'mean_overdueamountmaxdateyear_2T', 'mean_pmts_overdue_1140A',
'max_pmts_month_158T', 'max_pmts_year_1139T', 'mean_pmts_month_158T',
'mean_pmts_year_1139T']
####### NAN count = 262653
Use these ['mean_overdueamountmax2_398A', 'max_dateofcredend_353D',
'max_dateofcredstart_181D', 'mean_dateofcredend_353D',
'max_numberofoverdueinstlmax_1151L', 'mean_numberofoverdueinstlmax_1151L']
####### NAN count = 512590
Use these ['mean_overdueamountmax_35A', 'max_overdueamountmaxdatemonth_284T',
'max_overdueamountmaxdateyear_994T', 'mean_overdueamountmaxdatemonth_284T',
'mean_overdueamountmaxdateyear_994T', 'mean_pmts_overdue_1152A']
####### NAN count = 513987
Use these ['max_residualamount_488A']
####### NAN count = 1039597
Use these ['mean_residualamount_856A']
####### NAN count = 606900
Use these ['max_totalamount_6A', 'mean_totalamount_6A']
####### NAN count = 545855
Use these ['mean_totalamount_996A']
####### NAN count = 636448
Use these ['mean_totaldebtoverduevalue_718A',
'mean_totaloutstanddebtvalue_668A', 'mean_numberofcontrsvalue_358L']
####### NAN count = 297072
Use these ['max_dateofrealrepmt_138D', 'mean_dateofrealrepmt_138D']
####### NAN count = 512961
Use these ['max_lastupdate_388D', 'mean_lastupdate_388D']
####### NAN count = 512591
Use these ['max_numberofoverdueinstlmaxdat_148D']
####### NAN count = 802351
Use these ['mean_numberofoverdueinstlmaxdat_641D']
####### NAN count = 1012361
Use these ['mean_overdueamountmax2date_1002D']
####### NAN count = 806653
Use these ['max_overdueamountmax2date_1142D']
####### NAN count = 1007594
Use these ['mean_annualeffectiverate_199L']
####### NAN count = 1237917
Use these ['mean_annualeffectiverate_63L']
####### NAN count = 1270160
Use these ['mean_nominalrate_281L']
####### NAN count = 822517
```

```
Use these ['max_nominalrate_498L', 'mean_nominalrate_498L']
####### NAN count = 745109
Use these ['max_numberofinstls_229L', 'mean_numberofinstls_229L']
####### NAN count = 545898
Use these ['mean_numberofinstls_320L']
####### NAN count = 636545
Use these ['mean_numberofoutstandinstls_520L']
####### NAN count = 545895
Use these ['mean_numberofoutstandinstls_59L']
####### NAN count = 636544
Use these ['max_numberofoverdueinstls_834L', 'mean_numberofoverdueinstls_834L']
####### NAN count = 512657
Use these ['max_periodicityofpmts_1102L', 'mean_periodicityofpmts_1102L']
####### NAN count = 561307
Use these ['mean_periodicityofpmts_837L']
####### NAN count = 649082
Use these ['mean_prolongationcount_1120L']
####### NAN count = 1436524
Use these ['mean_num_group1_6']
####### NAN count = 140386
Use these ['max_empl_employedfrom_271D']
####### NAN count = 959958
Use these ['mean_contaddr_matchlist_1032L', 'mean_contaddr_smempladdr_334L']
####### NAN count = 441
####### NAN count = 935626
####### NAN count = 2
Use these ['mean_amount_416A', 'mean_openingdate_313D', 'max_num_group1_10']
####### NAN count = 1421548
Use these ['mean_openingdate_857D']
####### NAN count = 1421572
Use these ['max_num_group1_11']
####### NAN count = 1414887
Use these ['mean_collater_valueofguarantee_1124L']
####### NAN count = 262659
Use these ['mean_collater_valueofguarantee_876L']
####### NAN count = 512884
Use these ['max_pmts_month_706T', 'max_pmts_year_507T', 'mean_pmts_month_706T',
'mean_pmts_year_507T']
####### NAN count = 512598
Use these ['mean_num_group1_13', 'max_num_group2_13', 'mean_num_group2_13']
####### NAN count = 141371
['case_id', 'WEEK_NUM', 'target', 'month_decision', 'weekday_decision',
'credamount_770A', 'applicationcnt_361L', 'applications30d_658L',
'applicationscnt_1086L', 'applicationscnt_464L', 'applicationscnt_867L',
'clientscnt_1022L', 'clientscnt_100L', 'clientscnt_1071L', 'clientscnt_1130L',
'clientscnt_157L', 'clientscnt_257L', 'clientscnt_304L', 'clientscnt_360L',
'clientscnt_493L', 'clientscnt_533L', 'clientscnt_887L', 'clientscnt_946L',
'deferredmnthsnum_166L', 'disbursedcredamount_1113A', 'downpmt_116A',
```

'homephncnt_628L', 'isbidproduct_1095L', 'mobilephncnt_593L',
'numactivecreds_622L', 'numactivecredschannel_414L', 'numactiverelcontr_750L',
'numcontrs3months_479L', 'numnotactivated_1143L', 'numpmtchanneldd_318L',
'numrejects9m_859L', 'sellerplacecnt_915L', 'max_mainoccupationinc_384A',
'max_birth_259D', 'mean_persontype_1072L', 'description_5085714M_cnt',
'education_1103M_cnt', 'maritalst_893M_cnt', 'maritalst_385M_cnt',
'requesttype_4525192L_cnt', 'bankacctype_710L_cnt', 'cardtype_51L_cnt',
'credtype_322L_cnt', 'disbursementtype_67L_cnt',
'equalitydataagreement_891L_cnt', 'isbidproduct_1095L_cnt',
'lastapprcommoditycat_1041M_cnt', 'lastcancelreason_561M_cnt',
'lastrejectcommoditycat_161M_cnt', 'lastrejectcommodtypec_5251769M_cnt',
'lastrejectreason_759M_cnt', 'lastst_736L_cnt', 'paytype1st_925L_cnt',
'twobodfilling_608L_cnt', 'typesuite_864L_cnt', 'max_cancelreason_3545846M_cnt',
'max_education_1138M_cnt', 'max_postype_4733339M_cnt',
'max_credacc_status_367L_cnt', 'max_credtype_587L_cnt',
'max_familystate_726L_cnt', 'max_isbidproduct_390L_cnt',
'max_isdebitcard_527L_cnt', 'max_status_219L_cnt',
'max_collaterals_typeofguarante_669M_cnt', 'max_classificationofcontr_400M_cnt',
'max_contractst_545M_cnt', 'max_contractst_964M_cnt',
'max_financialinstitution_382M_cnt', 'max_financialinstitution_591M_cnt',
'max_purposeofcred_874M_cnt', 'max_subjectrole_93M_cnt',
'max_education_927M_cnt', 'max_empladdr_district_926M_cnt',
'max_language1_981M_cnt', 'max_contaddr_matchlist_1032L_cnt',
'max_contaddr_smempladdr_334L_cnt', 'max_empl_employedtotal_800L_cnt',
'max_empl_industry_691L_cnt', 'max_familystate_447L_cnt',
'max_housetype_905L_cnt', 'max_incometype_1044T_cnt', 'max_role_1084L_cnt',
'max_safeguarantyflag_411L_cnt', 'max_sex_738L_cnt', 'max_type_25L_cnt',
'max_collaterals_typeofguarante_359M_cnt', 'assignmentdate_238D',
'assignmentdate_4527235D', 'pmtaverage_4527227A', 'pmtcount_4527229L',
'birthdate_574D', 'mean_contractsum_5085717L', 'dateofbirth_337D',
'days180_256L', 'days30_165L', 'days360_512L', 'firstquarter_103L',
'fourthquarter_440L', 'secondquarter_766L', 'thirdquarter_1082L',
'max_debtoutstand_525A', 'max_debtoverdue_47A', 'max_refreshdate_3813885D',
'mean_refreshdate_3813885D', 'pmtaverage_3A', 'pmtcount_693L', 'pmtscount_423L',
'pmtssum_45A', 'responsedate_1012D', 'responsedate_4527233D',
'responsedate_4917613D', 'actualdpdtolerance_344P',
'amtinstpaidbefduel24m_4187115A', 'numinstlswithdpd5_4187116L',
'annuitynextmonth_57A', 'currdebt_22A', 'currdebtcredtyperange_828A',
'numinstls_657L', 'totalsettled_863A', 'mindbddpdlast24m_3658935P',
'avgdbddpdlast3m_4187120P', 'mindbdtollast24m_4525191P',
'avgdpdtolclosure24_3658938P', 'avginstallast24m_3658937A',
'maxinstallast24m_3658928A', 'avglnamtstart24m_4525187A',
'avgmaxdpdlast9m_3716943P', 'avgoutstandbalancel6m_4187114A',
'avgpmtlast12m_4525200A', 'cntincpaycont9m_3716944L', 'cntpmts24_3658933L',
'commnoinclast6m_3546845L', 'maxdpdfrom6mto36m_3546853P',
'datefirstoffer_1144D', 'datelastinstal40dpd_247D', 'datelastunpaid_3546854D',
'daysoverduetolerancedd_3976961L', 'numinsttopaygr_769L',
'dtlastpmtallstes_4499206D', 'eir_270L', 'firstclxcampaign_1125D',

```
'firstdatedue_489D', 'inittransactionamount_650A', 'lastactivateddate_801D',
'lastapplicationdate_877D', 'mean_creationdate_885D', 'mean_isbidproduct_390L',
'max_num_group1', 'lastapprcredamount_781A', 'lastapprdate_640D',
'lastdelinqdate_224D', 'lastrejectcredamount_222A', 'lastrejectdate_50D',
'maininc_215A', 'mastercontrelectronic_519L', 'mastercontrexist_109L',
'maxannuity_159A', 'maxdebt4_972A', 'maxdpdlast24m_143P', 'maxdpdlast3m_392P',
'maxdpdtolerance_374P', 'maxdbddpdlast1m_3658939P',
'maxdbddpdtollast12m_3658940P', 'maxdbddpdtollast6m_4187119P',
'maxdpdinstldate_3546855D', 'maxdpdinstlnum_3546846P',
'maxlnamtstart6m_4525199A', 'maxoutstandbalancel12m_4187113A',
'maxpmtlast3m_4525190A', 'numinstpaidearly_338L', 'numinstpaidearly5d_1087L',
'numinstpaidlate1d_3546852L', 'numincomingpmts_3546848L',
'numinstlsallpaid_934L', 'numinstlswithdpd10_728L', 'numinstlswithoutdpd_562L',
'numinstpaid_4499208L', 'numinstpaidearly3d_3546850L',
'numinstregularpaidest_4493210L', 'numinstpaidearly5dest_4493211L',
'sumoutstandtotalest_4493215A', 'numinstpaidlastcontr_4325080L',
'numinstregularpaid_973L', 'pctinstlsallpaidearl3d_427L',
'pctinstlsallpaidlate1d_3546856L', 'pctinstlsallpaidlat10d_839L',
'pctinstlsallpaidlate4d_3546849L', 'pctinstlsallpaidlate6d_3546844L',
'pmtnum_254L', 'posfpd10lastmonth_333P', 'posfpd30lastmonth_3976960P',
'posfstqpd30lastmonth_3976962P', 'price_1097A', 'sumoutstandtotal_3546847A',
'totaldebt_9A', 'totinstallast1m_4525188A', 'validfrom_1069D',
'mean_actualdpd_943P', 'max_annuity_853A', 'mean_annuity_853A',
'mean_credacc_actualbalance_314A', 'mean_credacc_maxhisbal_375A',
'mean_credacc_minhisbal_90A', 'mean_credacc_transactions_402L',
'max_credacc_credlmt_575A', 'max_credamount_590A', 'max_downpmt_134A',
'mean_credacc_credlmt_575A', 'mean_credamount_590A', 'mean_downpmt_134A',
'max_currdebt_94A', 'mean_currdebt_94A', 'max_mainoccupationinc_437A',
'mean_mainoccupationinc_437A', 'mean_maxdpdtolerance_577P',
'max_outstandingdebt_522A', 'mean_outstandingdebt_522A',
'mean_revolvingaccount_394A', 'max_approvaldate_319D', 'mean_approvaldate_319D',
'max_dateactivated_425D', 'mean_dateactivated_425D', 'max_dtlastpmt_581D',
'mean_dtlastpmt_581D', 'max_dtlastpmtallstes_3545839D',
'mean_dtlastpmtallstes_3545839D', 'max_employedfrom_700D',
'max_firstnonzeroinstldate_307D', 'mean_firstnonzeroinstldate_307D',
'mean_byoccupationinc_3656910L', 'mean_childnum_21L', 'max_pmtnum_8L',
'mean_pmtnum_8L', 'mean_isdebitcard_527L', 'max_amount_4527230A',
'max_recorddate_4527225D', 'max_num_group1_3', 'mean_amount_4917619A',
'max_deductiondate_4917603D', 'mean_deductiondate_4917603D', 'max_num_group1_4',
'max_pmtamount_36A', 'max_processingdate_168D', 'mean_processingdate_168D',
'max_num_group1_5', 'mean_credlmt_230A', 'mean_credlmt_935A',
'mean_pmts_dpd_1073P', 'mean_dpdmaxdatemonth_89T', 'mean_dpdmaxdateyear_596T',
'max_pmts_dpd_303P', 'mean_dpdmax_757P', 'max_dpdmaxdatemonth_442T',
'max_dpdmaxdateyear_896T', 'mean_dpdmaxdatemonth_442T',
'mean_dpdmaxdateyear_896T', 'mean_pmts_dpd_303P', 'mean_instlamount_768A',
'mean_instlamount_852A', 'mean_monthlyinstlamount_332A',
'max_monthlyinstlamount_674A', 'mean_monthlyinstlamount_674A',
'mean_outstandingamount_354A', 'mean_outstandingamount_362A',
```

```
'mean_overdueamount_31A', 'mean_overdueamount_659A',
'mean_numberofoverdueinstls_725L', 'mean_overdueamountmax2_14A',
'mean_totaloutstanddebtvalue_39A', 'mean_dateofcredend_289D',
'mean_dateofcredstart_739D', 'max_lastupdate_1112D', 'mean_lastupdate_1112D',
'mean_numberofcontrsvalue_258L', 'mean_numberofoverdueinstlmax_1039L',
'mean_overdueamountmaxdatemonth_365T', 'mean_overdueamountmaxdateyear_2T',
'mean_pmts_overdue_1140A', 'max_pmts_month_158T', 'max_pmts_year_1139T',
'mean_pmts_month_158T', 'mean_pmts_year_1139T', 'mean_overdueamountmax2_398A',
'max_dateofcredend_353D', 'max_dateofcredstart_181D', 'mean_dateofcredend_353D',
'max_numberofoverdueinstlmax_1151L', 'mean_numberofoverdueinstlmax_1151L',
'mean_overdueamountmax_35A', 'max_overdueamountmaxdatemonth_284T',
'max_overdueamountmaxdateyear_994T', 'mean_overdueamountmaxdatemonth_284T',
'mean_overdueamountmaxdateyear_994T', 'mean_pmts_overdue_1152A',
'max_residualamount_488A', 'mean_residualamount_856A', 'max_totalamount_6A',
'mean_totalamount_6A', 'mean_totalamount_996A',
'mean_totaldebtoverduevalue_718A', 'mean_totaloutstanddebtvalue_668A',
'mean_numberofcontrsvalue_358L', 'max_dateofrealrepmt_138D',
'mean_dateofrealrepmt_138D', 'max_lastupdate_388D', 'mean_lastupdate_388D',
'max_numberofoverdueinstlmaxdat_148D', 'mean_numberofoverdueinstlmaxdat_641D',
'mean_overdueamountmax2date_1002D', 'max_overdueamountmax2date_1142D',
'mean_annualeffectiverate_199L', 'mean_annualeffectiverate_63L',
'mean_nominalrate_281L', 'max_nominalrate_498L', 'mean_nominalrate_498L',
'max_numberofinstls_229L', 'mean_numberofinstls_229L',
'mean_numberofinstls_320L', 'mean_numberofoutstandinstls_520L',
'mean_numberofoutstandinstls_59L', 'max_numberofoverdueinstls_834L',
'mean_numberofoverdueinstls_834L', 'max_periodicityofpmts_1102L',
'mean_periodicityofpmts_1102L', 'mean_periodicityofpmts_837L',
'mean_prolongationcount_1120L', 'mean_num_group1_6',
'max_empl_employedfrom_271D', 'mean_contaddr_matchlist_1032L',
'mean_contaddr_smempladdr_334L', 'mean_remitter_829L',
'mean_safeguarantyflag_411L', 'mean_amount_416A', 'mean_openingdate_313D',
'max_num_group1_10', 'mean_openingdate_857D', 'max_num_group1_11',
'mean_collater_valueofguarantee_1124L', 'mean_collater_valueofguarantee_876L',
'max_pmts_month_706T', 'max_pmts_year_507T', 'mean_pmts_month_706T',
'mean_pmts_year_507T', 'mean_num_group1_13', 'max_num_group2_13',
'mean_num_group2_13']
352
424
```

[38]:
```python
y = df_train["target"]
weeks = df_train["WEEK_NUM"]
df_train= df_train.drop(columns=["target", "case_id", "WEEK_NUM"])
n_splits=5
cv = StratifiedGroupKFold(n_splits=n_splits, shuffle=False)
```

[ ]:
```python
#
categorical_features = []
```

```
for col in df_train.columns:
    if pd.api.types.is_categorical_dtype(df_train[col]) or df_train[col].dtype␣
↪== 'object':
        categorical_features.append(col)

#
joblib.dump(categorical_features, '/Users/wuqianran/Desktop/
↪bigdata_finalproject/final/categorical_features.pkl')

#
joblib.dump(df_train.dtypes, '/Users/wuqianran/Desktop/bigdata_finalproject/
↪final/column_dtypes.pkl')
```

[40]:
```
# params = {
#     "boosting_type": "gbdt",

#     "objective": "binary",
#     "metric": "auc",
#     "max_depth": 8,
#     "learning_rate": 0.01,
#     "n_estimators": 10000,
#     "colsample_bytree": 0.8,
#     "colsample_bynode": 0.8,
#     "verbose": -1,
#     "random_state": 42,
#     "reg_alpha": 0.3,
#     "reg_lambda": 8,
#     "extra_trees":True,
#     'num_leaves':32,
#     "sample_weight":'balanced',
#     # "device": "cpu",
#     "device": "gpu",
#     "verbose": -1,
# }
#
df_train.dtypes.to_frame('dtype').to_csv('/Users/wuqianran/Desktop/
↪bigdata_finalproject/final/column_dtypes.csv')

#
df_train.to_csv('/Users/wuqianran/Desktop/bigdata_finalproject/final/
↪processed_data.csv', index=False)

params = {
    "boosting_type": "gbdt",
    "objective": "binary",
    "metric": "auc",
    "max_depth": 4,  #
```

```
    "learning_rate": 0.05,  #
    "n_estimators": 100,  #
    "colsample_bytree": 0.6,
    "colsample_bynode": 0.6,
    "verbose": -1,
    "random_state": 42,
    "reg_alpha": 0.1,  #   L1
    "reg_lambda": 1,  #   L2
    "extra_trees": True,
    'num_leaves': 8,  #
    "min_data_in_leaf": 50,  #
    "device": "cpu",
    # "device": "gpu",
    "verbose": -1,
}
fitted_models = []
cv_scores = []
best_auc = 0
best_model = None

for idx_train, idx_valid in cv.split(df_train, y, groups=weeks):#   Because it␣
 ↪takes a long time to divide the data set,
    X_train, y_train = df_train.iloc[idx_train], y.iloc[idx_train]# each time␣
 ↪the data set is divided, two models are trained to each other twice, which␣
 ↪saves time.
    X_valid, y_valid = df_train.iloc[idx_valid], y.iloc[idx_valid]
    model = lgb.LGBMClassifier(**params)
    model.fit(
        X_train, y_train,
        eval_set = [(X_valid, y_valid)],
        callbacks = [lgb.log_evaluation(200), lgb.early_stopping(100)] )
    fitted_models.append(model)
    y_pred_valid = model.predict_proba(X_valid)[:,1]
    auc_score = roc_auc_score(y_valid, y_pred_valid)
    cv_scores.append(auc_score)

    #      AUC
    if auc_score > best_auc:
        best_auc = auc_score
        best_model = model

if best_model is not None:
    joblib.dump(best_model, '/Users/wuqianran/Desktop/bigdata_finalproject/
 ↪final/lgbm_best_model.pkl')

lgb_cv_results = pd.DataFrame({
    'fold': range(1, n_splits + 1),
```

```
    'auc_score': cv_scores
})
lgb_cv_results.to_csv('/Users/wuqianran/Desktop/bigdata_finalproject/final/
↪lgbm_results.csv', index=False)


print("CV AUC scores: ", cv_scores)
print("Maximum CV AUC score: ", max(cv_scores))
```

```
Training until validation scores don't improve for 100 rounds
Did not meet early stopping. Best iteration is:
[100]   valid_0's auc: 0.818329
Training until validation scores don't improve for 100 rounds
Did not meet early stopping. Best iteration is:
[100]   valid_0's auc: 0.817141
Training until validation scores don't improve for 100 rounds
Did not meet early stopping. Best iteration is:
[100]   valid_0's auc: 0.823298
Training until validation scores don't improve for 100 rounds
Did not meet early stopping. Best iteration is:
[100]   valid_0's auc: 0.823715
Training until validation scores don't improve for 100 rounds
Did not meet early stopping. Best iteration is:
[100]   valid_0's auc: 0.816754
CV AUC scores:  [0.8183287976164907, 0.8171405047673326, 0.8232983625757633,
0.8237148323046233, 0.8167539058241644]
Maximum CV AUC score:  0.8237148323046233
```

[ ]:
```
# %%
#
categorical_features = []
for col in df_train.columns:
    if pd.api.types.is_categorical_dtype(df_train[col]) or df_train[col].dtype␣
↪== 'object':
        categorical_features.append(col)

#
joblib.dump(categorical_features, '/Users/wuqianran/Desktop/
↪bigdata_finalproject/final/categorical_features.pkl')

#
joblib.dump(df_train.dtypes, '/Users/wuqianran/Desktop/bigdata_finalproject/
↪final/column_dtypes.pkl')
```

[ ]:
```
['/Users/wuqianran/Desktop/bigdata_finalproject/final/column_dtypes.pkl']
```

[ ]:
```python
y = df_train["target"]
weeks = df_train["WEEK_NUM"]
df_train= df_train.drop(columns=["target", "case_id", "WEEK_NUM"])
df_train[cat_cols] = df_train[cat_cols].astype(str)
```

[ ]:
```python
from catboost import CatBoostClassifier, Pool

# params = {
#     "eval_metric": "AUC",
#     # "depth": 10,
#     "learning_rate": 0.03,
#     "iterations": 6000,  # 4000
#     # "random_seed": 3107,
#     # "l2_leaf_reg": 10,
#     # "border_count": 254,
#     "verbose": 500,
#     "task_type": "GPU",
#     "early_stopping_rounds": 100  #
# }

params = {
    "eval_metric": "AUC",
    "depth": 6,  #
    "learning_rate": 0.01,  #
    "iterations": 3000,  #
    "l2_leaf_reg": 5,  #   L2
    "verbose": 500,
    "task_type": "CPU",
    "early_stopping_rounds": 100  #
}

fitted_models = []
cv_scores = []
best_auc = 0
best_model = None

cv = StratifiedGroupKFold(n_splits=n_splits, shuffle=False)

step = 0
for idx_train, idx_valid in cv.split(df_train, y, groups=weeks):#   Because it␣
↪takes a long time to divide the data set,
    step += 1
    print(f'current step: {step}')

    X_train, y_train = df_train.iloc[idx_train], y.iloc[idx_train]# each time␣
↪the data set is divided, two models are trained to each other twice, which␣
↪saves time.
```

```
    X_valid, y_valid = df_train.iloc[idx_valid], y.iloc[idx_valid]

    train_pool = Pool(X_train, y_train,cat_features=cat_cols)
    val_pool = Pool(X_valid, y_valid,cat_features=cat_cols)

    model = CatBoostClassifier(**params)
    model.fit(train_pool, eval_set=val_pool, verbose=100, early_stopping_rounds=50)


    fitted_models.append(model)
    y_pred_valid = model.predict_proba(X_valid)[:,1]
    auc_score = roc_auc_score(y_valid, y_pred_valid)
    cv_scores.append(auc_score)

    #      AUC
    if auc_score > best_auc:
        best_auc = auc_score
        best_model = model

#
if best_model is not None:
    joblib.dump(best_model, '/Users/wuqianran/Desktop/bigdata_finalproject/final/catboost_best_model.pkl')

#
cv_results = pd.DataFrame({
    'fold': range(1, n_splits + 1),
    'auc_score': cv_scores
})
cv_results.to_csv('/Users/wuqianran/Desktop/bigdata_finalproject/final/catboost_results.csv', index=False)


print("CV AUC scores: ", cv_scores)
print("AVG CV AUC score: ", np.mean(cv_scores))
print("Maximum CV AUC score: ", max(cv_scores))
```