导入包

In [11]:
```
import os
import re
import logging
from datetime import datetime, date
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold, KFold
from sklearn.metrics import mean_squared_error
import xgboost as xgb
from xgboost import DMatrix
import lightgbm as lgb
from lightgbm import Dataset
import matplotlib.pyplot as plt
from astral import LocationInfo
from astral.sun import sunrise, sunset, dawn, noon, dusk


logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=lc
```

读取数据

比赛数据

In [12]:
```
x_train = pd.read_csv("../data/A榜-训练集_分布式光伏发电预测_气象变量数据.csv", enc
y_train = pd.read_csv("../data/A榜-训练集_分布式光伏发电预测_实际功率数据.csv", enc
info_train = pd.read_csv("../data/A榜-训练集_分布式光伏发电预测_基本信息.csv", enco
x_test = pd.read_csv("../data/A榜-测试集_分布式光伏发电预测_气象变量数据.csv", enco
y_test = pd.read_csv("../data/submit_example.csv", encoding="utf-8")
info_test = pd.read_csv("../data/A榜-测试集_分布式光伏发电预测_基本信息.csv", encod
```

In [13]:
```
x_train = pd.merge(x_train, info_train[["光伏用户编号", "装机容量(kW)", "经度", "纟
x_train["时间"] = pd.to_datetime(x_train["时间"], utc=False).dt.tz_localize('Asia
x_test = pd.merge(x_test, info_test[["光伏用户编号", "装机容量(kW)", "经度", "纬度
x_test["时间"] = pd.to_datetime(x_test["时间"], utc=False).dt.tz_localize('Asia/S
```

In [14]:
```
y_train = y_train.set_index(["光伏用户编号", "综合倍率", "时间"]).stack().reset_in
y_train["level_3"] = y_train["level_3"].apply(lambda x: int(x[1:]))
y_train["时间"] = pd.to_datetime(y_train["时间"], utc=False).dt.tz_localize('Asia
y_train["时间"] = y_train["时间"] + (y_train["level_3"] - 1) * 15 * pd.Timedelta(
y_train = y_train.drop(columns=["level_3"])

y_test = y_test.set_index(["光伏用户编号", "综合倍率", "时间"]).stack().reset_inde
y_test["level_3"] = y_test["level_3"].apply(lambda x: int(x[1:]))
y_test["时间"] = pd.to_datetime(y_test["时间"], utc=False).dt.tz_localize('Asia/S
y_test["时间"] = y_test["时间"] + (y_test["level_3"] - 1) * 15 * pd.Timedelta(1,
y_test = y_test.drop(columns=["level_3"])
```

In [15]:
```
df_train = pd.merge(x_train, y_train, on=["光伏用户编号", "时间"], how="left")
df_test = pd.merge(x_test, y_test, on=["光伏用户编号", "时间"], how="left")
```

```
df = pd.concat([df_train, df_test], axis=0)
logging.info(df.columns)
```

```
2024-04-04 17:02:35,766 : INFO : Index(['光伏用户编号', '时间', '气压(Pa) ', '相对
湿度 (%) ', '云量', '10米风速 (10m/s) ', '10米风向 (°)',
       '温度 (K) ', '辐照强度 (J/m2) ', '降水 (m) ', '100m风速 (100m/s) ', '100m风向
(°)',
       '装机容量(kW)', '经度', '纬度', '综合倍率', 'target'],
      dtype='object')
```

外部数据

In [16]:
```
outer_files = os.listdir("../data/")
outer_files = [x for x in outer_files if re.match(r"^open-meteo-f\d\.csv$", x)]
outer_dfs = list()
for outer_file in outer_files:
    outer_df = pd.read_csv(os.path.join("../data", outer_file), skiprows=3)
    outer_df["time"] = pd.to_datetime(outer_df["time"], utc=False).dt.tz_localiz
    outer_df = outer_df.set_index("time", drop=True).asfreq("15T")
    outer_df = outer_df.interpolate(method="time", axis=0)

    outer_df = outer_df.reset_index(drop=False).rename(columns={"time": "时间"})
    outer_df["光伏用户编号"] = re.match(r"^open-meteo-(f\d)\.csv$", outer_file).g
    outer_dfs.append(outer_df)
outer_data = pd.concat(outer_dfs, axis=0)
logging.info(outer_data.columns)
```

```
2024-04-04 17:03:04,705 : INFO : Index(['时间', 'temperature_2m_archive_best_matc
h (°C)',
       'relative_humidity_2m_archive_best_match (%)',
       'dew_point_2m_archive_best_match (°C)',
       'apparent_temperature_archive_best_match (°C)',
       'precipitation_archive_best_match (mm)', 'rain_archive_best_match (mm)',
       'snowfall_archive_best_match (cm)', 'snow_depth_archive_best_match (m)',
       'weather_code_archive_best_match (wmo code)',
       ...
       'direct_normal_irradiance_era5_land (W/m²)',
       'global_tilted_irradiance_era5_land (W/m²)',
       'terrestrial_radiation_era5_land (W/m²)',
       'shortwave_radiation_instant_era5_land (W/m²)',
       'direct_radiation_instant_era5_land (W/m²)',
       'diffuse_radiation_instant_era5_land (W/m²)',
       'direct_normal_irradiance_instant_era5_land (W/m²)',
       'global_tilted_irradiance_instant_era5_land (W/m²)',
       'terrestrial_radiation_instant_era5_land (W/m²)', '光伏用户编号'],
      dtype='object', length=222)
```

In [17]:
```
outer_day_files = os.listdir("../data/")
outer_day_files = [x for x in outer_day_files if re.match(r"^open-meteo-day-f\d\
outer_day_dfs = list()
for outer_day_file in outer_day_files:
    outer_day_df = pd.read_csv(os.path.join("../data", outer_day_file), skiprows
    outer_day_df["time"] = pd.to_datetime(outer_day_df["time"], utc=False).dt.tz
    outer_day_df = outer_day_df.rename(columns={"time": "时间"})
    outer_day_df["光伏用户编号"] = re.match(r"^open-meteo-day-(f\d)\.csv$", outer
    outer_day_dfs.append(outer_day_df)
outer_day_data = pd.concat(outer_day_dfs, axis=0)
logging.info(outer_day_data.columns)
```

```
2024-04-04 17:03:04,756 : INFO : Index(['时间', 'weather_code (wmo code)', 'tempe
rature_2m_max (°C)',
       'temperature_2m_min (°C)', 'temperature_2m_mean (°C)',
       'apparent_temperature_max (°C)', 'apparent_temperature_min (°C)',
       'apparent_temperature_mean (°C)', 'sunrise (iso8601)',
       'sunset (iso8601)', 'daylight_duration (s)', 'sunshine_duration (s)',
       'precipitation_sum (mm)', 'rain_sum (mm)', 'snowfall_sum (cm)',
       'precipitation_hours (h)', 'wind_speed_10m_max (km/h)',
       'wind_gusts_10m_max (km/h)', 'wind_direction_10m_dominant (°)',
       'shortwave_radiation_sum (MJ/m²)', 'et0_fao_evapotranspiration (mm)',
       '光伏用户编号'],
      dtype='object')
```

In [ ]:

```
outer_data["日期"] = outer_data["时间"].dt.date
outer_day_data["日期"] = outer_day_data["时间"].dt.date
outer_day_data = outer_day_data.drop(columns=["时间"])
outer_data = pd.merge(outer_data, outer_day_data, on=["光伏用户编号", "日期"], ho
for column in outer_data.columns:
    logging.info(column)
```

整合数据

In [19]:

```
df = pd.merge(df, outer_data[[
    "光伏用户编号",
    "时间",
    "apparent_temperature_archive_best_match (°C)",
    "surface_pressure_archive_best_match (hPa)",
    "is_day_archive_best_match ()",
    "relative_humidity_2m_archive_best_match (%)",
    "cloud_cover_archive_best_match (%)",
    "global_tilted_irradiance_instant_archive_best_match (W/m²)",
    "precipitation_archive_best_match (mm)",
    "wind_speed_10m_archive_best_match (km/h)",
    "wind_speed_100m_archive_best_match (km/h)",
    "wind_direction_10m_archive_best_match (°)",
    "wind_direction_100m_archive_best_match (°)",
    "sunshine_duration_archive_best_match (s)",
    "weather_code_archive_best_match (wmo code)",
    "temperature_2m_max (°C)",
    "temperature_2m_min (°C)",
    "temperature_2m_mean (°C)",
    "precipitation_sum (mm)",
    "rain_sum (mm)",
    "snowfall_sum (cm)",
    "apparent_temperature_max (°C)",
    "apparent_temperature_min (°C)",
    "apparent_temperature_mean (°C)",
    "sunrise (iso8601)",
    "sunset (iso8601)",
    "shortwave_radiation_archive_best_match (W/m²)",
    "direct_radiation_archive_best_match (W/m²)",
    "diffuse_radiation_archive_best_match (W/m²)",
    "direct_normal_irradiance_archive_best_match (W/m²)",
    "global_tilted_irradiance_archive_best_match (W/m²)",
    "terrestrial_radiation_archive_best_match (W/m²)"
]], how="left", on=["光伏用户编号", "时间"])

df["温度 (K) "] = df["apparent_temperature_archive_best_match (°C)"] + 273.15
```

```
df["气压(Pa) "] = df["surface_pressure_archive_best_match (hPa)"] * 100
df["是白天"] = df["is_day_archive_best_match ()"].copy()
df["相对湿度 (%) "] = df["relative_humidity_2m_archive_best_match (%)"].copy()
df["云量"] = df["cloud_cover_archive_best_match (%)"] / 100
df["辐照强度 (J/m2) "] = df["global_tilted_irradiance_instant_archive_best_match
df["降水 (m) "] = df["precipitation_archive_best_match (mm)"] / 1000
df["10米风速 (10m/s) "] = df["wind_speed_10m_archive_best_match (km/h)"] / 3600 *
df["100m风速 (100m/s) "] = df["wind_speed_100m_archive_best_match (km/h)"] / 3600
df["10米风向 (°)"] = df["wind_direction_10m_archive_best_match (°)"].copy()
df["100m风向 (°)"] = df["wind_direction_100m_archive_best_match (°)"].copy()
df["sunrise (iso8601)"] = pd.to_datetime(df["sunrise (iso8601)"], utc=False).dt.
df["sunset (iso8601)"] = pd.to_datetime(df["sunset (iso8601)"], utc=False).dt.tz
df["时间-日出时间"] = ((df["时间"] - df["sunrise (iso8601)"]).dt.days * 24 * 3600
df["日落时间-时间"] = ((df["sunset (iso8601)"] - df["时间"]).dt.days * 24 * 3600

df = df.drop(columns=[
    "apparent_temperature_archive_best_match (°C)",
    "surface_pressure_archive_best_match (hPa)",
    "is_day_archive_best_match ()",
    "relative_humidity_2m_archive_best_match (%)",
    "cloud_cover_archive_best_match (%)",
    "global_tilted_irradiance_instant_archive_best_match (W/m²)",
    "precipitation_archive_best_match (mm)",
    "wind_speed_10m_archive_best_match (km/h)",
    "wind_speed_100m_archive_best_match (km/h)",
    "wind_direction_10m_archive_best_match (°)",
    "wind_direction_100m_archive_best_match (°)",
    "sunrise (iso8601)",
    "sunset (iso8601)"
])
```

特征工程

时间特征

In [20]:

```
df["年"] = df["时间"].dt.year
df["季节"] = df["时间"].dt.quarter
df["月"] = df["时间"].dt.month
df["日"] = df["时间"].dt.day
df["周"] = df["时间"].dt.week
df["分"] = df["时间"].dt.minute // 15 + df["时间"].dt.hour * 4
```

```
c:\program files\python37\lib\site-packages\ipykernel_launcher.py:5: FutureWarnin
g: Series.dt.weekofyear and Series.dt.week have been deprecated.  Please use Seri
es.dt.isocalendar().week instead.
  """
```

In [21]:

```
# df["分_"] = df["分"].copy()
# df = pd.get_dummies(df, columns=["分_"], prefix_sep="")
df["分"] = df["分"].astype("category")
```

根据日出时间日落时间计算时间段

In [22]:

```
def solar_time(current_time, dawn_time, sunrise_time, noon_time, sunset_time, du
    """
    根据太阳判断当前时间段\n
    """
    if dawn_time < current_time < sunrise_time:
        return 1
    elif sunrise_time <= current_time < noon_time:
        return 2
    elif noon_time <= current_time < sunset_time:
        return 3
    elif sunset_time <= current_time < dusk_time:
        return 4
    else:
        return 0
```

In [23]:
```
# df["地点"] = df.apply(lambda x: LocationInfo(name=x["光伏用户编号"], region="Ch
# df["黎明时刻"] = df.apply(lambda x: dawn(x["地点"].observer, date=x["时间"], tz
# df["日出时刻"] = df.apply(lambda x: sunrise(x["地点"].observer, date=x["时间"],
# df["正午时刻"] = df.apply(lambda x: noon(x["地点"].observer, date=x["时间"], tz
# df["日落时刻"] = df.apply(lambda x: sunset(x["地点"].observer, date=x["时间"],
# df["黄昏时刻"] = df.apply(lambda x: dusk(x["地点"].observer, date=x["时间"], tz
# df["时间段"] = df.apply(lambda x: solar_time(x["时间"], x["黎明时刻"], x["日出时
# df = df.drop(columns=["地点", "黎明时刻", "日出时刻", "正午时刻", "日落时刻", "黄
```

光伏用户编号

In [24]:
```
df["光伏用户编号_"] = df["光伏用户编号"].copy()
df = pd.get_dummies(df, columns=["光伏用户编号_"], prefix_sep="")
```

气象特征

In [25]:
```
df['100m风速 (100m/s) '] = df['100m风速 (100m/s) '] * np.sin(np.pi * df['100m风向
# df['cos_100m风速 (100m/s) '] = df['100m风速 (100m/s) '] * np.cos(np.pi * df['10
df['10米风速 (10m/s) '] = df['10米风速 (10m/s) '] * np.sin(np.pi * df['10米风向 (°
# df['cos_10米风速 (10m/s) '] = df['10米风速 (10m/s) '] * np.cos(np.pi * df['10米
```

In [26]:
```
df["光照/温度"] = df["辐照强度 (J/m2) "] / df["温度 (K) "]
```

历史值特征

In [27]:
```
dfs = []
for site, df_site in df.groupby("光伏用户编号"):
    df_site = df_site.sort_values("时间")
    df_site["辐照强度 (J/m2)  - 1"] = df_site["辐照强度 (J/m2) "].shift(1) - df_si
    df_site["辐照强度 (J/m2)  - 8"] = df_site["辐照强度 (J/m2) "].shift(8) - df_si
#     df_site["辐照强度 (J/m2)  - 2"] = df_site["辐照强度 (J/m2) "].shift(2) - df_
    dfs.append(df_site)
df = pd.concat(dfs, axis=0)
```

处理异常值

In [28]:
```
print(df_train["target"].nsmallest(3))
df_train[df_train["target"] < -8]
df.loc[207628:207633, ["光伏用户编号", "时间", "target"]]
```

```
207630   -8.8900
37316    -0.0085
39139    -0.0084
Name: target, dtype: float64
```

Out[28]:

	光伏用户编号	时间	target
207628	f6	2022-08-15 20:30:00+08:00	-0.002
207629	f6	2022-08-15 20:45:00+08:00	-0.002
207630	f6	2022-08-15 21:00:00+08:00	-8.890
207631	f6	2022-08-15 21:15:00+08:00	-0.002
207632	f6	2022-08-15 21:30:00+08:00	NaN
207633	f6	2022-08-15 21:45:00+08:00	NaN

In [29]:
```
df.loc[207630, "target"] = -0.002
```

光照与当天最强光照的比值

In [30]:
```
df["日期"] = df["时间"].dt.date
day_max_values = df[["光伏用户编号", "日期", "辐照强度 (J/m2) "]].groupby(by=["光伏
day_max_values = day_max_values.rename(columns={x: x + "_max" for x in day_max_v
df = pd.merge(df, day_max_values, on=["光伏用户编号", "日期"], how="left").drop(c
df["辐照强度 (J/m2) _max"] = df["辐照强度 (J/m2) "] / df["辐照强度 (J/m2) _max"]
```

当天的平均光照

In [31]:
```
df["日期"] = df["时间"].dt.date
day_mean_values = df[["光伏用户编号", "日期", "是白天", "辐照强度 (J/m2) "]].groupb
day_mean_values = day_mean_values.rename(columns={x: x + "_mean" for x in day_me
df = pd.merge(df, day_mean_values, on=["光伏用户编号", "日期", "是白天"], how="lef
```

温度与当天最高温最低度的差值

In [32]:
```
df["日期"] = df["时间"].dt.date
day_max_values = df[["光伏用户编号", "日期", "温度 (K) "]].groupby(by=["光伏用户编号
day_min_values = df[["光伏用户编号", "日期", "温度 (K) "]].groupby(by=["光伏用户编号
day_max_values = day_max_values.rename(columns={x: x + "_max" for x in day_max_v
day_min_values = day_min_values.rename(columns={x: x + "_min" for x in day_min_v
df = pd.merge(df, day_max_values, on=["光伏用户编号", "日期"], how="left")
df = pd.merge(df, day_min_values, on=["光伏用户编号", "日期"], how="left").drop(c
df["温度 (K) _max"] = df["温度 (K) _max"] - df["温度 (K) "]
df["温度 (K) _min"] = df["温度 (K) "] - df["温度 (K) _min"]
df = df.rename(columns={
    "辐照强度 (J/m2) _max": "光照/当天最强光照",
    "温度 (K) _max": "与当天最高温度之差",
    "温度 (K) _min": "与当天最低温度之差"
})
```

划分测试集

In [33]:
```
df_train = df[df["时间"] <= df_train["时间"].max()]
df_test = df[df["时间"] >= df_test["时间"].min()]
```

训练模型

评测指标

In [34]:
```
def score(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    return 1 / (1 + rmse)
```

lightgbm模型

In [35]:
```
params_lgb = {
    "num_boost_round": 1000,
    'learning_rate': 0.02,
    'boosting_type': 'gbdt',
    'objective': 'mse',
    'metric': 'rmse',
    'num_leaves': 127,
    'verbose': -1,
    'seed': 42,
    'n_jobs': -1,
    'feature_fraction': 0.8,
    'bagging_fraction': 0.9,
    'bagging_freq': 4,
    "early_stopping_round": 100
}
model_lgb = []
```

xgboost模型

In [36]:
```
params_xgb = {
    "num_boost_round": 500,
    "learning_rate": 0.02,
    "booster": "gbtree",
    "objective": "reg:squarederror",
    "eval_metric": "rmse",
    "max_leaves": 127,
    "verbosity": 1,
    "seed": 42,
    "nthread": -1,
    "colsample_bytree": 0.6,
    "subsample": 0.7,
    "early_stopping_rounds": 100
}
model_xgb = []
```

交叉验证

In [ ]:
```
kfold = KFold(n_splits=5, random_state=42, shuffle=True)
```

```
x = df_train.drop(columns=["光伏用户编号", "时间"]).dropna().astype(np.float32)
y = x.pop("target")
mse = 0
for fold, (train_index, val_index) in enumerate(kfold.split(x, y)):
    logging.info(f'############ fold: {fold} ###########')
    x_train, x_val, y_train, y_val = x.iloc[train_index], x.iloc[val_index], y.i

    trainset = Dataset(x_train, y_train)
    valset = Dataset(x_val, y_val)
    model = lgb.train(params_lgb, trainset, valid_sets=[trainset, valset], categ
    model.save_model("../models/lgb_%d.txt" % fold)
    model_lgb.append(model)
    lgb_pred = Series(model.predict(x_val, num_iteration=model.best_iteration),

    trainset = DMatrix(x_train, y_train, enable_categorical=True, nthread=-1)
    valset = DMatrix(x_val, y_val, enable_categorical=True, nthread=-1)
    model = xgb.train(params_xgb, trainset, evals=[(trainset, 'train'),(valset,
    model.save_model("../models/xgb_%d.json" % fold)
    model_xgb.append(model)
    xgb_pred = Series(model.predict(valset, iteration_range=(0, model.best_ntree

    val_pred = (lgb_pred + xgb_pred) / 2
    mse += mean_squared_error(y_val.fillna(0), val_pred)
rmse = np.sqrt(mse / kfold.n_splits)
score = 1 / (1 + rmse)
logging.info(f"-------------本地分数 {score}-------------")
```

In [ ]:
```
importance = DataFrame()
importance["特征"] = model_lgb[0].feature_name()
importance["重要性"] = 0
for model in model_lgb:
    importance["重要性"] = importance["重要性"] + model.feature_importance()
importance["重要性"] = importance["重要性"] / kfold.n_splits
importance.sort_values("重要性", ascending=False)[0:50]
```

预测

In [39]:
```
x_test = df_test.drop(columns=["光伏用户编号", "时间"]).astype(np.float32)
y_test = x_test.pop("target")
y_pred = np.zeros((df_test.shape[0], ))
for i in range(0, kfold.n_splits):
    y_pred += model_lgb[i].predict(x_test, num_iteration=model_lgb[i].best_itera
    y_pred += model_xgb[i].predict(DMatrix(x_test, enable_categorical=True, nthr
y_pred = y_pred / 2 / kfold.n_splits
df_test["target"] = y_pred
```

```
c:\program files\python37\lib\site-packages\ipykernel_launcher.py:8: SettingWithC
opyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [40]:
```
df_test = df_test[["光伏用户编号", "综合倍率", "年", "月", "日", "分", "target"]]
df_test["时间"] = df_test["年"].astype(str) + "-" + df_test["月"].astype(str) + '
```

```
df_test["分"] = "p" + (df_test["分"].astype(int) + 1).astype(str)
df_test = df_test.drop(columns=["年", "月", "日"])
```

In [41]:

```
result = pd.pivot(df_test, index=["光伏用户编号", "综合倍率", "时间"], columns="分
result = result[result["综合倍率"].notnull()]
result["综合倍率"] = result["综合倍率"].astype(int)
```

In [42]:

```
result.to_csv("../data/%s.csv" % datetime.now().strftime("%Y%m%d_%H%M%S"), encod
```

In [ ]: