Kaggle: ASHRAE - Great Energy Predictor III Summary

Abstract：

本次比赛的目标是要解决能源消耗预测问题（预测给定建筑物，在指定时间下，指定能源类型的能源消耗量）。我们使用基于决策树算法的分布式梯度提升框架 LightGBM，从过去一段时间内的建筑物相关信息（建筑物面积、建筑年代、建筑物用途等）、建筑物所处位置的天气信息（温度、大气压强、降水量等）、能源消耗读数（标签）以及特征工程加入的一些特征学习模型。我们通过学习得到的 LightGBM 模型，预测测试集上的最终结果。采用不同的策略训练得到多个 LightGBM 模型，最终通过模型融合的方法来获得在测试集上更好的效果。

Overview：

本次比赛的目标是要解决能源消耗预测问题（预测给定建筑物，在指定时间下，指定能源类型的能源消耗量）。1448 个建筑物的详细信息存储在 ***building_meta*** 数据中，这些建筑物分布在 15 个地方。这 15 个地方的天气信息存储在 ***weather_train/test*** 数据中。
weather_train 数据集存储的是训练数据中的时间戳对应时间下当地的天气信息，***test*** 对应的是测试集。训练集中会给出最终的能源消耗读数（***meter_reading***），测试集中则需要预测给定情况下的能源消耗读数（***meter_reading***）。
最终采用的评估方法为 RMSLE。

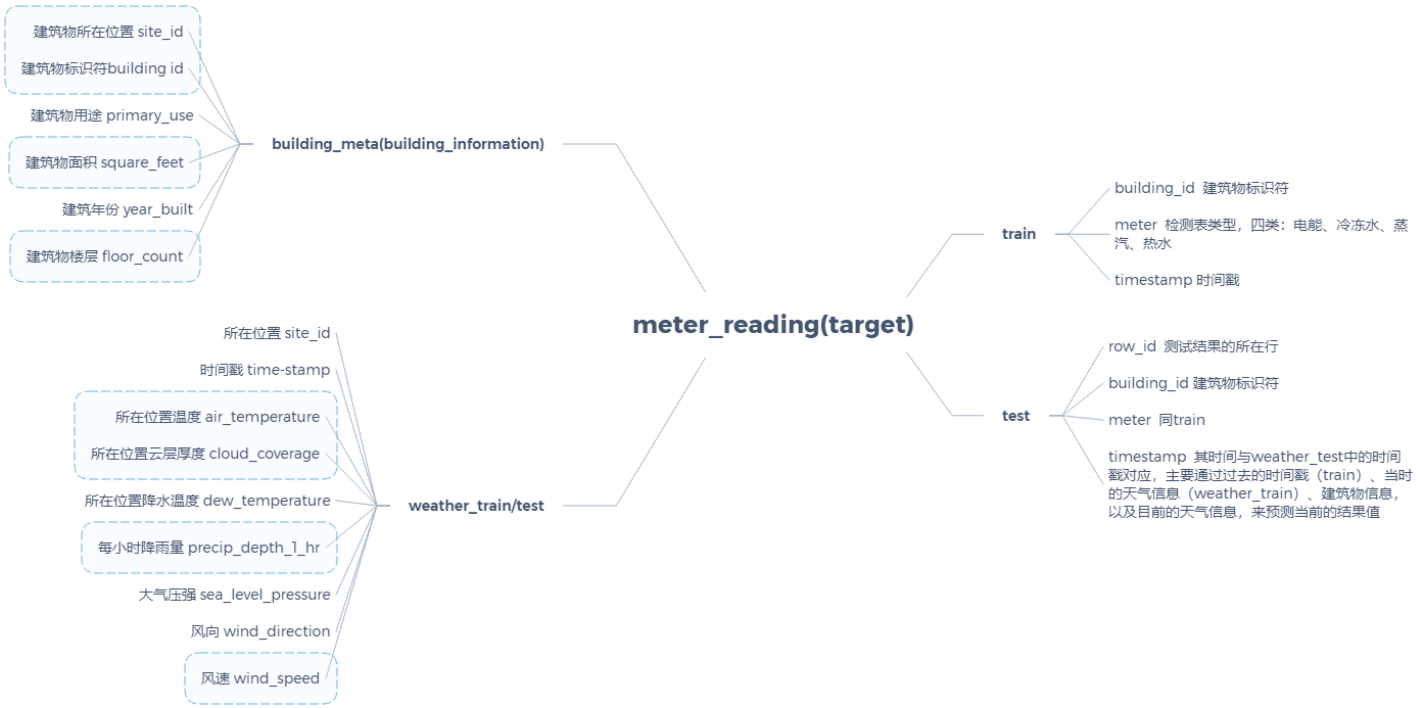
The RMSLE is calculated as

$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}$$

Where:

ϵ is the RMSLE value (score)
n is the total number of observations in the (public/private) data set,
p_i is your prediction of target, and
a_i is the actual target for i.
$\log(x)$ is the natural logarithm of x

原始数据集:
building_meta、weather_train、weather_test、sample_submission、train、test 六个文件。



整体思路：

将训练集 train 和 building_meta、以及 weather_train 数据拼接在一起（采用 building_id，（site_id, timestamp）为外键）。最终得到的训练集的格式为
（building_id,
site_id,
timestamp,
building_id 对应的 building_meta 的相关信息,
（site_id, timestamp）对应的 weather_train 的相关信息,
meter,
meter_reading）。
其中要预测的标签即为 meter_reading。测试数据格式中没有 meter_reading，其余相同。

训练模型使用的是 LightGBM，以及模型融合。

LightGBM 使用的是基于 Histogram 的决策树算法，并使用带深度限制的 Leaf_wise 的叶子生长策略，具有轻量化、速度快的特点，并直接支持类别特征。。LightGBM 训练使用的是 python 的 lightgbm 包。将训练数据切分成两部分，一部分是 LightGBM 的训练集，另一部分是 LightGBM 的验证集。先将训练数据转化为 LightGBM 中包装好的 Dataset 格式，其中分类 features 需要特别指定（在 categorical_feature 参数中）。然后使用 lightgbm 包中的 train 函数训练返回模型。train 函数中的训练集和验证集需要特别指定，以及其他一些相关的训练参数。最终通过模型的 predict 函数来预测最终的结果。

```
params = {
```

```
    "objective": "regression",
    "boosting": "gbdt",
    "num_leaves": 40,
    "learning_rate": 0.05,
    "feature_fraction": 0.85,
    "reg_lambda": 2,
    "metric": "rmse"
}

print("Building model with first half and validating on second half:")
model_half_1 = lgb.train(params, train_set=d_half_1, num_boost_round=1000, valid_sets=watchlist_1, verbose_eval=200, early_stopping_rounds=200)

print("Building model with second half and validating on first half:")
model_half_2 = lgb.train(params, train_set=d_half_2, num_boost_round=1000, valid_sets=watchlist_2, verbose_eval=200, early_stopping_rounds=200)
```

Kernel 1：ASHRAE: Half and Half （RMSLE 1.1）
https://www.kaggle.com/rohanrao/ashrae-half-and-half
特征工程：给了一个 holidays 列表，加入一个 is_holiday 的列，若该天在 holidays 中，则 is_holiday 的值为 1，否则为 0。

训练方式：将原训练集切割（原训练集按照时间排序，非随机切割）成大小相同的两部分，训练出两个 LightGBM 模型。最终用这两个模型预测最终的 test 集结果做平均。

Kernel 2：ASHRAE: KFold LightGBM - without leak （RMSLE 1.08）
https://www.kaggle.com/aitude/ashrae-kfold-lightgbm-without-leak-1-08
特征工程：

（1） 训练集存在一些异常的数据，meter_reading 一直为 0，因此先将满足 `(building_id <= 104 & meter == 0 & timestamp <= "2016-05-20")` 这部分数据去掉。

（2） 对于每个缺失（NaN）的数据，按照一定的分组，使用分组的平均值来填充。相比起直接粗暴地将整列的平均值直接填入，可能效果会更好些。

（3） 删去了"timestamp","sea_level_pressure", "wind_direction", “wind_speed","year_built","floor_count"一些缺失数据较多的列。

（4） 由于不同建筑物的 square_feet 相差较大，为了使数据不太离散，做了取对数处理。

训练方式：将原训练集切割成大小相同的三部分，训练出三个 LightGBM 模型。最终用这三个模型预测最终的 test 集结果做平均。

Kernel 3：ASHRAE: Highway Kernel Route2（1.03）
https://www.kaggle.com/yamsam/ashrae-highway-kernel-route2
特征工程：

（1） 加入一些 lag 特征，来解决时间序列问题。

```
def add_lag_feature(weather_df, window=3):
    group_df = weather_df.groupby('site_id')
    cols = ['air_temperature', 'cloud_coverage', 'dew_temperature', 'pre
cip_depth_1_hr', 'sea_level_pressure', 'wind_direction', 'wind_speed']
    rolled = group_df[cols].rolling(window=window, min_periods=0)
    lag_mean = rolled.mean().reset_index().astype(np.float16)
    lag_max = rolled.max().reset_index().astype(np.float16)
    lag_min = rolled.min().reset_index().astype(np.float16)
    lag_std = rolled.std().reset_index().astype(np.float16)
    for col in cols:
        weather_df[f'{col}_mean_lag{window}'] = lag_mean[col]
        weather_df[f'{col}_max_lag{window}'] = lag_max[col]
        weather_df[f'{col}_min_lag{window}'] = lag_min[col]
        weather_df[f'{col}_std_lag{window}'] = lag_std[col]
```

训练方式：将原训练集切割成大小相同的三部分，训练出三个 LightGBM 模型。最终用这三个模型预测最终的 test 集结果做平均。

模型融合：
使用几个模型的结果，分别取一定的比重加权求和得到最终结果，以泄露的数据作为标签，计算评测指标 RMSLE。取 RMSLE 最小的一组作为结果。

最终结果：

在 88%的公开数据上的最好 RMSLE = 0.957。

本次比赛尝试过的但是效果不好的方法：

（1） 根据标签值 meter_reading 对所有建筑物进行聚类分组，加入分组标签。效果反而下降了一些。

（2） 直接使用三层全连接神经网络对多个模型的输出拟合最终结果，效果不好。

未来得及尝试的一些方法：

（1） 使用全连接网络对 LightGBM 的叶子输出做拟合，得到最终的输出结果。

（2） 使用 LSTM 网络对时序信息处理。

训练 trick：

（1） 根据每列数据中的最大最小值，将其转化为适合的数据类型，可以减少一部分的内存占用空间：

```
def reduce_mem_usage(df, use_float16=False):
    """
    Iterate through all the columns of a dataframe and modify the data ty
pe 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:
        if is_datetime(df[col]) or is_categorical_dtype(df[col]):
            continue
        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.i
nt8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(n
p.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(n
p.int32).max:
                    df[col] = df[col].astype(np.int32)
```

```
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(n
p.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if use_float16 and 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:
            df[col] = df[col].astype("category")

    end_mem = df.memory_usage().sum() / 1024**2
    print("Memory usage after optimization is: {:.2f} MB".format(end_me
m))
    print("Decreased by {:.1f}%".format(100 * (start_mem - end_mem) / sta
rt_mem))

    return df
```

（2） 将原数据文件 csv 转化为 feather 文件，读取速度会大大提升（10 倍以上）。
https://www.kaggle.com/corochann/ashrae-feather-format-for-fast-loading

```
# Read data...
root = '../input/ashrae-energy-prediction'

train_df = pd.read_csv(os.path.join(root, 'train.csv'))
weather_train_df = pd.read_csv(os.path.join(root, 'weather_train.csv'))
test_df = pd.read_csv(os.path.join(root, 'test.csv'))
weather_test_df = pd.read_csv(os.path.join(root, 'weather_test.csv'))
building_meta_df = pd.read_csv(os.path.join(root, 'building_metadata.csv
'))
sample_submission = pd.read_csv(os.path.join(root, 'sample_submission.cs
v'))
```

```
train_df['timestamp'] = pd.to_datetime(train_df['timestamp'])
test_df['timestamp'] = pd.to_datetime(test_df['timestamp'])
weather_train_df['timestamp'] = pd.to_datetime(weather_train_df['timesta
mp'])
weather_test_df['timestamp'] = pd.to_datetime(weather_test_df['timestamp
'])
```

```
train_df.to_feather('train.feather')
test_df.to_feather('test.feather')
weather_train_df.to_feather('weather_train.feather')
weather_test_df.to_feather('weather_test.feather')
building_meta_df.to_feather('building_metadata.feather')
sample_submission.to_feather('sample_submission.feather')
```

```
train_df = pd.read_feather('train.feather')
weather_train_df = pd.read_feather('weather_train.feather')
test_df = pd.read_feather('test.feather')
weather_test_df = pd.read_feather('weather_test.feather')
building_meta_df = pd.read_feather('building_metadata.feather')
sample_submission = pd.read_feather('sample_submission.feather')
```