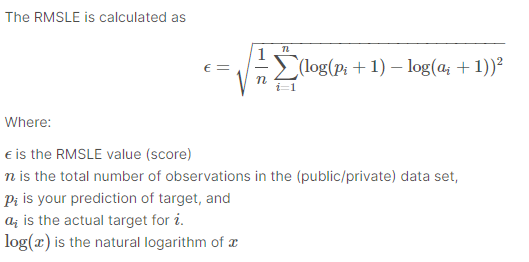
Kaggle: ASHRAE - Great Energy Predictor III Summary

**Abstract：**

本次比赛的目标是要解决[能源消耗预测问题](https://www.kaggle.com/c/ashrae-energy-prediction/overview)（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。

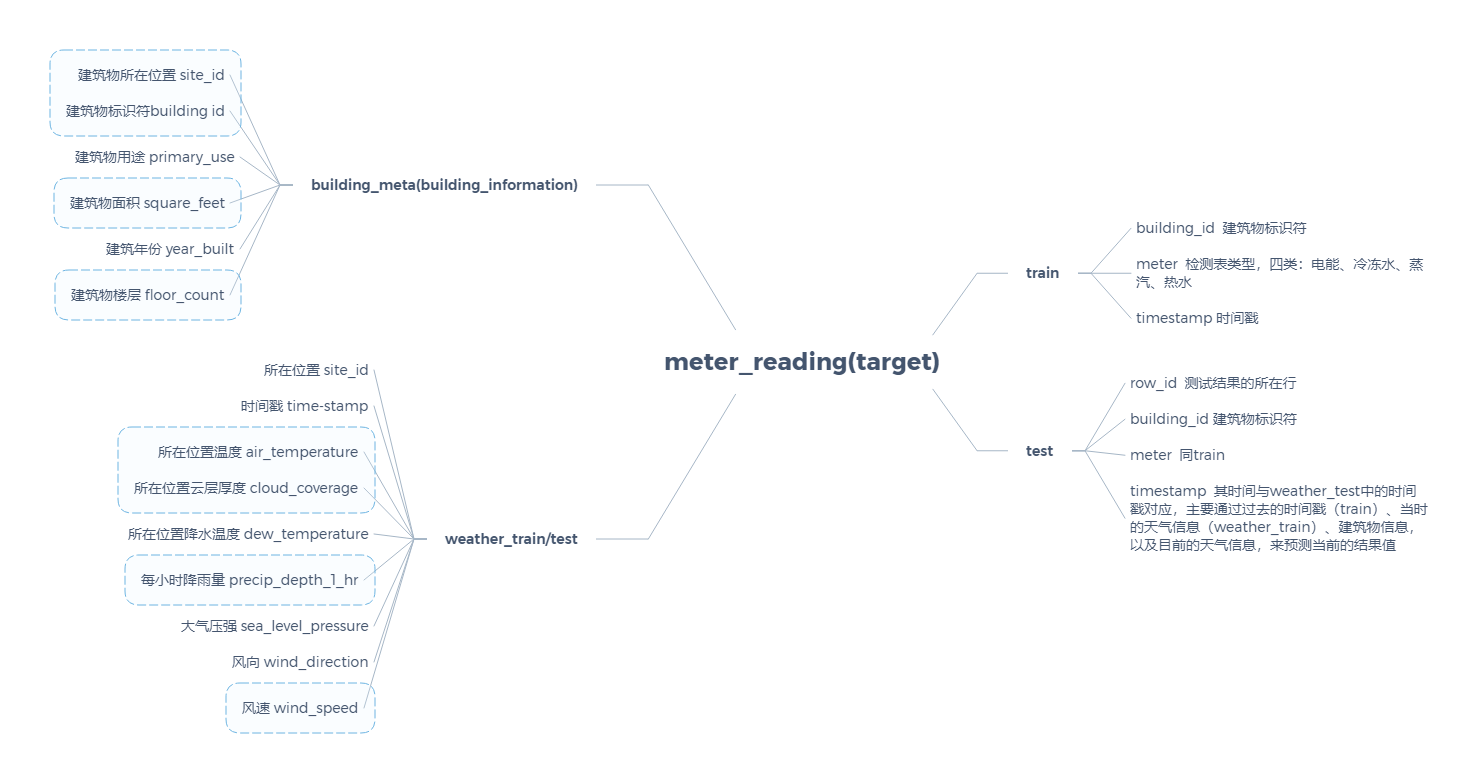
**Overview：**

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。

[原始数据集](https://www.kaggle.com/c/ashrae-energy-prediction/data)：

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。

1. 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。
2. 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。
3. 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', 'precip\_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 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:

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.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 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\_mem))

print("Decreased by **{:.1f}**%".format(100 \* (start\_mem - end\_mem) / start\_mem))

return df

1. 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.csv'))

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['timestamp'])

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')