EE 660 Type (1) Project

(Kaggle Competition) ASHRAE - Great Energy Predictor III

(Team 70) Number of student authors: 2
Zhengxu Hou (zhengxuh@usc.edu)
Zuoqian Xu (zuoqianx@usc.edu)
Date: 12/07/2019

1. Abstract

Nowadays, Developing energy savings has two key elements: Forecasting future energy usage without improvements, and forecasting energy use after a specific set of improvements have been implemented, like the installation and purchase of investment-grade meters, whose prices continue to fall. One issue preventing more aggressive growth of the energy markets are the lack of cost-effective, accurate, and scalable procedures for forecasting energy use.

In this Project, we'll develop accurate predictions of metered building energy usage in the following areas: chilled water, electric, natural gas, hot water, and steam meters. The data comes from over 1,000 buildings over a three-year timeframe.With better estimates of these energy-saving investments, large scale investors and financial institutions will be more inclined to invest in this area to enable progress in building efficiencies.

The goal of our project is to build accurate models of metered building energy usage in chilled water, electric, hot water and steam meters areas. We will use building information, weather information for featuring dataset and trained our model to predict the final meter reading. Two of the models we will be using in this project have been covered in our EE660 course include Decision Tree and Adaboost, other two models we use here will be more advanced target to our goal which are LightGBM and XGBoost. The evaluation metric for this competition is Root Mean Squared Error, which will be calculated using below equation.

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

$\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n$ are predicted values
$y_1, y_2, \ldots, y_n$ are observed values
n is the number of observations

And for visualization, we will have validation learning curve and tree graph as result.

2. Introduction

2.1. Problem Type, Statement and Goals

We are given three datasets: weather_train.csv, building_metadata.csv and train.csv. Unlike other datasets we practiced before, this datasets have very high dimensionality of feature space and are not merged into single training dataset, thus the first task is to explore the "junctions" of the three datasets and try to find way to merge them into single file that we can train our model for.

The second challenge will be handling the missing data. Our project dataset is collected from indoor and outdoor sensors. And the sensors are from various sites and buildings which may have thousands of them, thus malfunctions of the sensor are inevitable. Weather_train and building_metadata have large amount of missing values, our goal is to use proper way to fill in those missing values before fit them into our machine learning models.

As mentioned above, the data is been collected by time. The feature "timestamp" is the specific date and time the data has been collected. In order to fit our time series model, one task is to expand the timestamp to time features include month, day and hour. Moreover, many features are time dependent and can be binning in specific rules. Thus a good feature engineering can significantly improve our model accuracy due to the high quantity of our dataset.

The goal of our project is to predict the meter_reading which is Energy consumption of each buildings in kWh (or equivalent). The energy consumption value is continuous, thus we need to use regression models to predict. The machine learning methods we are using includes tree based model DecisionTreeRegressor, three boosting model Adaboost, LightGBM and XGBoost. Among those methods, DecisionTreeRegressor and Adaboost are two methods we covered in lectures, whereas Light BGM and Prophet are more challenging for us to investigate.

2.2. Literature Review

2.3 Our Prior and Related Work

Zuoqian did a related work previously on his internship. The work is specifically design a system and use appropriate model to forecast an auto vehicle sales in the next month and year. The forecasting should target on both short term and long term, for example, predicting sales in the future month requires model that accept high variance data caused by short term fluctuation. Whereas the long term prediction required model that accept low variace dataset and yield forecasting result that focus more on long term tendency. The model used in the project are mostly time series forecasting model include: LSTM, ARIMA, also some simple tree and boosting model like RandomForestRegressor, XGBoost etc.

Zhengxu's prior related work.

Zhengxu has done some work related to data science in the previous two years. The most related one is project before. In the competition he design a model which could predict what customers want to buy, based on the features of product information and habit of customers. The model he use is XGBoost and also KNN.

2.4 Overview of Our Approach

Our Project focus on investigating four models. Decision Tree Regressor is the basic machine learning tree model related to our course topic. AdaBoost is the basic boosting model that we also covered in the lecture. From those two models, we will gain insights about basic knowledge of machine learning algorithms, how these two basic models perform on bigger datasets and how the features interact with the parameters we selected.

We also investigate two more advanced models, Light GBM and Prophet. those two models will have more complex parameters we need to consider tuning. It's excited to learn what those parameters do and how they will affect the model preventing from underfitting or overfitting.

3. Implementation

3.1 Dataset

General inforamtion about the dataset:

Size of train_df data (20216100, 4)

Size of weather_train_df data (139773, 9)

Size of weather_test_df data (277243, 9)

Size of building_meta_df data (1449, 6)

Detailed information about the dataset:

train.csv:

- building_id
  - Explaination: Foreign key for the building metadata.
  - Type: Numerical Integer
  - Range: 0 - 1448
- meter
  - Explanation: The meter id code. Read as {0: Electricity, 1: Chilled Water, 2: Steam, Hotwater: 3}. Not every building has all meter types.
  - Type: Categorical Integer
  - Cardinality: 4
  - Range: 0 - 3
- timestamp
  - Explanation: When the measurement was taken
  - Type: Numerical String
  - Range: 2016/1/1 12:00:00 AM - 2016/1/20 12:00:00 AM
- meter_reading
  - Explanation: The target variable. Energy consumption in kWh (or equivalent). Note that this is real data with measurement error, which we expect will impose a baseline level of modeling error.
  - Type: Numercial Real
  - Range: 0 - 8243400

building_meta.csv

- site_id
  - Explaination: Foreign key for the weather_train.csv
  - Type: Catagorical Integer

    - Cardinality: 16
    - Range: 0 - 15
- building_id
    - Explaination Foreign key for training.csv
    - Type: Categorical Integer
    - Cardinality: 1449
    - Range: 0 - 1448
- primary_use
    - Explanation: Indicator of the primary category of activities for the building based on Energy Star property type definitions
    - Type: Categorical String
    - Cardinality: 16
    - Range: 0 - 15 (If Encoded)
- square_feet
    - Explanation: Gross floor area of the building
    - Type: Numerical Integer
    - Range: 283 - 875000
- year_built
    - Explanation: Year building was opened
    - Type: Numberical Integer
    - Range: 1900 - 2017
- floorcount
    - Explanation: Number of floors of the building
    - Type: Categorical Integer
    - Cardinality: 26
    - Range: 1 - 26

weather[train/test].csv

Explaination: Weather data from a meteorological station as close as possible to the site.

- site_id
    - Explanation: Specific geological site for measurment
    - Type: Categorical Integer
    - Cardinality: 16
    - Range: 0 -15
- air_temperature
    - Explanation: Degrees Celsius
    - Type: Numerical Real
    - Range: -28.9 - 47.2
- cloud_coverage

    - Explaination: Portion of the sky covered in clouds, in oktas
    - Type: Categorical Integer
    - Cardinality: 10
    - Range: 0 - 9
- dew_temperature
    - Explaination: Degrees Celsius
    - Type: Numerical Real
    - Range: - 35 - 26.1
- precip_depth_1_hr
    - Explaination: Millimeters
    - Type: Numerical Integer
    - Range: -1 - 343
- sea_level_pressure
    - Explanation: Millibar/hectopascals
    - Type: Numerical Real
    - Range: 968.2 - 1045.5
- wind_direction
    - Explaination: Compass direction (0-360)
    - Type: Categrical Interger
    - Cardinality: 361
    - Range: 0 -360
- wind_speed
    - Explaination: Meters per second
    - Type: Numerical Integer
    - Range: 0 - 19

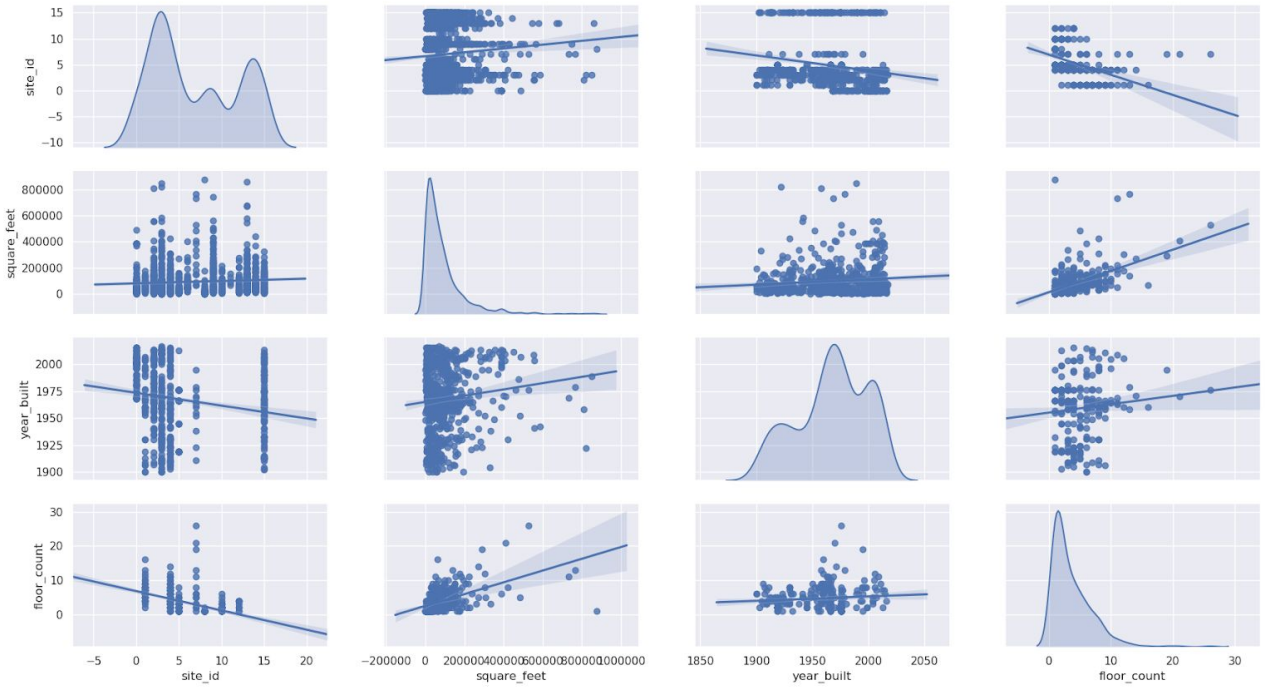
3.2 Preprocessing, Feature Extraction, Dimensionality Adjustment

Feature visualization on builidng dataset

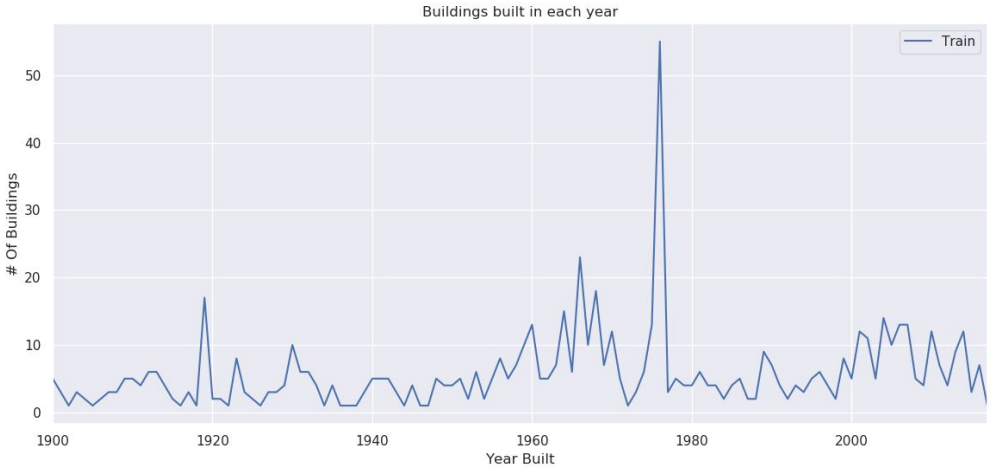
For building dataset, including features: site_id, building_id, primary_use, square_feet, year_built, floor_count. In the pairplot, we could see the primary_use feature is missing, since this feature is not numeric at all, so seaborn cannot plot. And also, building_id is not significant, since it is used to make a concatenation with weather dataset.

So, after I delete the column [building_id], I plot the pairplot of four features. There are two features which are useful.

1. Year_build and floor_count are positive correlation.
2. square_feet and floor_count are also positive correlation, which make sense.



3. The distribution of the year_build is kind of uniform distribution as shown below:



Missing data information for the building dataset:

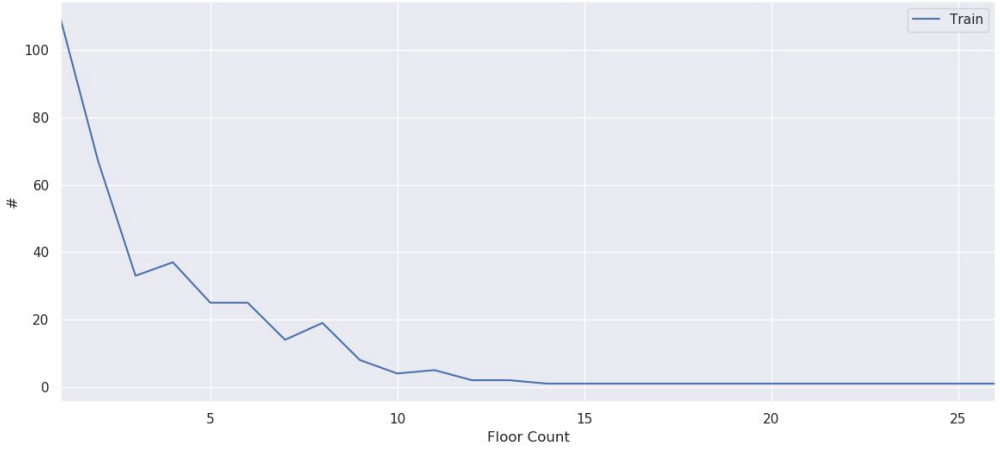
	COLUMN NAME	MISSING VALUES	TOTAL ROWS	% MISSING
0	site_id	0	1449	0.00
1	building_id	0	1449	0.00
2	primary_use	0	1449	0.00
3	square_feet	0	1449	0.00
4	year_built	774	1449	53.42
5	floor_count	1094	1449	75.50

As we could see, the floor_count has a missing value of 1094, which is more than 80%, and for year_built, there are more than 50% data missing.

To streamline this though process it is useful to know the 3 categories in which missing data can be classified into:

- Missing Completely at Random (MCAR)
- Missing at Random (MAR)
- Missing Not at Random (MNAR)

In this time, for the year build data, choose mean to replace NAN is the best choice. SInce for mode, the result would be around year 1970, which would cause a high imbalance data after replacing Nan to mode.



For the floor count part, I could see that the distribution of floor count is kind of decreasing curve. Again, if we use the mode method to fix this data, this curve could be pretty sharp, and will cause high - imbalanced. So, again, I choose mean to fix this data.

Missing data information for the weather dataset:

	COLUMN NAME	MISSING VALUES	TOTAL ROWS	% MISSING
0	site_id	0	139773	0.00
1	timestamp	0	139773	0.00
2	air_temperature	55	139773	0.04
3	cloud_coverage	69173	139773	49.49
4	dew_temperature	113	139773	0.08
5	precip_depth_1_hr	50289	139773	35.98
6	sea_level_pressure	10618	139773	7.60
7	wind_direction	6268	139773	4.48
8	wind_speed	304	139773	0.22

At this point as we can see the missing data is mostly appears in weather_train dataset, and there is not missing data in the training dataset. The sensors that detect and collect the weather features data can be unstable thus cause inconsistent time series missing data.

The method we use here to handle those missing data is by using time series modeling. First we create timestamps which include month, week, day and daytime. The weather dataset is collected hourly, thus we create new timestamp features that group the hour time step into day, week and month. The dataset consists of 16 sites in 2016. So this should have 140,544 records (16 x 24 x 366). But the dataset has 139,773 records so 771 hours of data is missing in total.

The main method is fill with mean by different timestamp. Each feature in weather dataset may affect by different time range. Below is the analysis of missing value handling for different features:

Air Temperature:

Ides is to fill missing air temperature with mean temperature of day of the month. Each month comes in a season and temperature varies lots in a season. So filling with yearly mean value is not a good idea.

Fill Cloud Coverage:

Almost 50% data is missing. And data is missing for most of days and even many consecutive days. So, first, calculate mean cloud coverage of day of the month and then fill rest missing values with last valid observation.

For convenience, here we can summarize that the method of filling the rest of the weather data features are same as above. Filling the missing data of time series features should based on short term time step and take average of the data within these time steps.

With all those missing data filled, the result of missing data detection function is shown below:

```
1 missing_statistics(weather_train)
```

	COLUMN NAME	MISSING VALUES	TOTAL ROWS	% MISSING
0	site_id	0	140544	0.0
1	air_temperature	0	140544	0.0
2	cloud_coverage	0	140544	0.0
3	dew_temperature	0	140544	0.0
4	precip_depth_1_hr	0	140544	0.0
5	sea_level_pressure	0	140544	0.0
6	timestamp	0	140544	0.0
7	wind_direction	0	140544	0.0
8	wind_speed	0	140544	0.0

Feature Engineering:

1. Time stamp data processing

The feature timestamp is a string type feature showing the time when the other features been recorded. We expand this one feature into three more features each as “weekday”, “month” and “hour”. Notice that for date: 2016/1/1, we set the weekday to 4 since January 1st at 2016 is on Thursday. After expand this timestamp feature, we can let the algorithm recognize the time feature in our dataset.

2. Target Data Processing

For the target data meter_reading, take the log(x+1), by using the function np.log1p()

The max of meter reading is 21904700.0, and the minimum is 0.0, and mean is 2117.1210762116857, as we could see, those values are cover a range covering many orders of magnitude, 0 to 1,000,000,000,There are also a lot more data points clustered at the low end of that scale. By using the log of

revenue or budget we spread out those values much more evenly along the log scale. That makes it easier for use to visualise and easier for our models to fit. Of course, at last we need to decode this function to see the final results.

3. **wind speed classification**

For this part, since the wind speed is numeric, if we change this to category, it could be pretty helpful. So at last, we plan to use Beaufort map to classify the data. For example, if the wind speed is 0.2, this is actually in the first category (0, 0.3), so we classify this into 0.

4. **Wind direction**

For the wind direction part, the number of this data is ranging from 0 to 360, so in this part, I also change the numeric value to class value, where I divide the 360 into 16 parts. For example, if the wind direction 45, then it is in the second class.

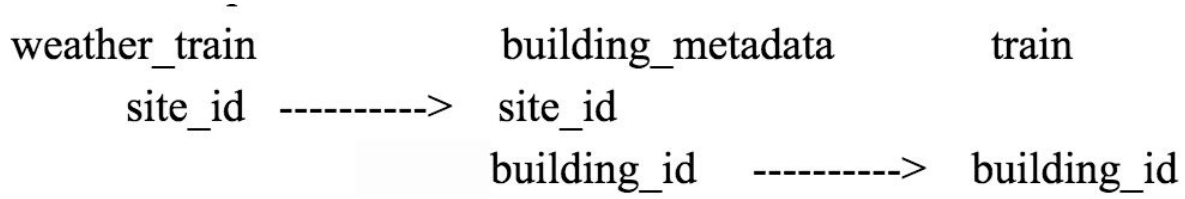
5. **square_feet**

For this data, since it range from 1,000 to 1000,000, so I take log() function in to this dataset, and also the reason that I don't use log1p is that there is no zero in this data, so that log function could work fine.

Merging multiple dataset into one training dataset:

Since we have 3 datasets, one main train dataset that contains the building id, and two sub-datasets that provides information for different sites and buildings in the main training dataset.

The link map is shown below:



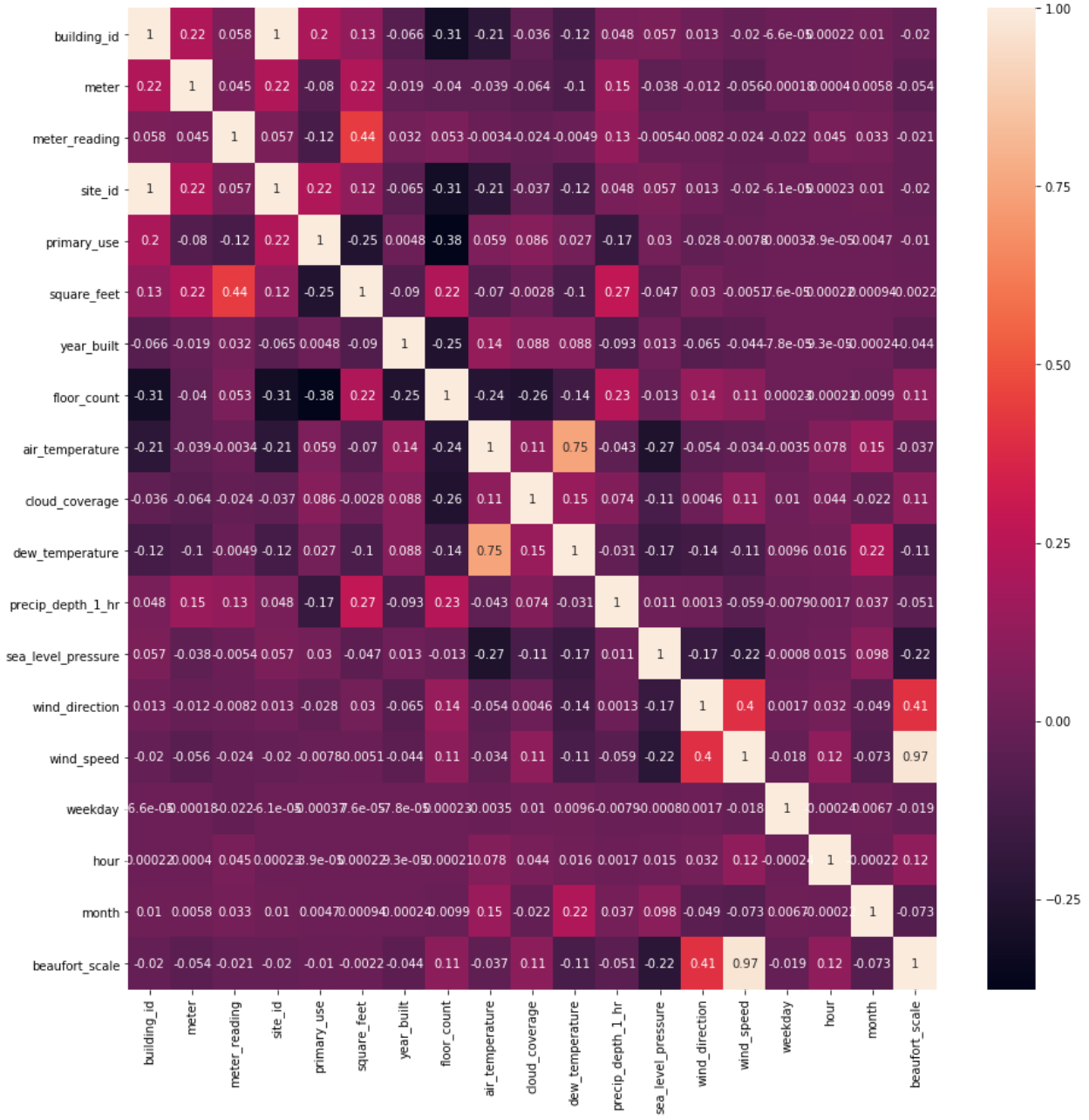
building_metadata contains two intercept features with train and weather train data. Our goal is to concatenate all the features in waether_train into building_meatadata using site_id as the joint. After joining weather_train in to building_metadata, then we need to concatenate all the features in new building_metadata into the main train data using building_id as the joint.

Complete training dataset information:

Size of new_train data (20216100, 20)

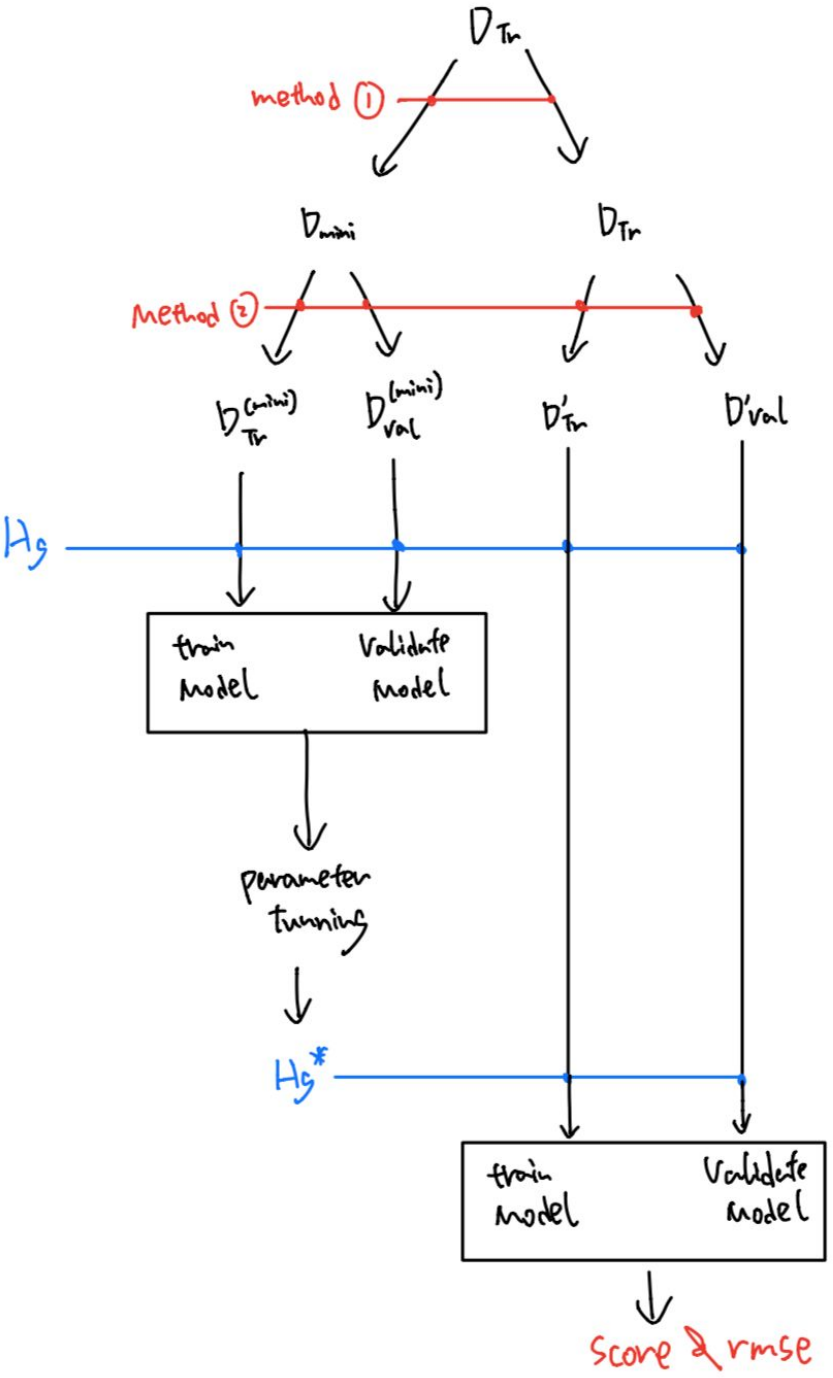
Linear correlation Analysis:

Heat Map:



Above is the heat map showing the linear correlation of all the features in new concatenated training dataset. The correlation metrics used here is “pearson” and from the graph, the top 2 most linearly correlated features are squared_feet and precip_depth_1_hr.

3.3 Dataset Methodology



Explanation:

First we use method(1) to extract a small portion of the data from original train data as mini batch training set. Second, use method(2) to split both the mini-batch set and original set into training and validation sets. Hg represent our hypothesis which is the ML model we are investigating. Implementing the hypothesis sets into training set for mini-batch set and together use validation

set for model parameter tuning. Finally use the best hypothesis yielded from mini-batch set and implement on original training set. With training and validation proceed, we output the visualization results and scores for our final models.

Note:

Method (1) : Subset choosing. Since the whole dataset is too huge that running the model is very time consuming, so we plan to use a small subset for the hypothesis set building, which is 1/100 of the original one. And In the original data, the total points are 20216100, this number contains product of 365(days), 24(hours), 1448(building ID). Then use only a part of this data, that is, a product of 184(days), 6 hours in one day, 235 building ID. With this mini batch of dataset created, although we may lose accuracy on the final complete data implementation of the model, we can save huge amount of run time for the convenience of other complex tasks.

Method (2) : The method we use here target on split the data into training and validation sets. Since in kaggle data provided, final correct prediction of the test set is not given for the sake of competition. Thus in order to validate our model, we need to use sklearn built in function : “train_test_split” in the Python package: sklearn.model_selection. The split percentage of validation set is 33% of the original data. And the data points we are is randomly chosen.

3.4 Training Process

Model 1:

Decision Tree Regressor

Description:

Decision trees are predictive models that use a set of binary rules to calculate a target value.

Each individual tree is a fairly simple model that has branches, nodes and leaves.

Splitting Criterion:

A single number that represents how good a split is which is the weighted average of the mean squared errors of the two groups that create.

A way to find the best split which is to try every variable and to try every possible value of that variable and see which variable and which value gives us a split with the best score.

Halting Condition:

When hit a limit that was requested (for examplemax_depth)

When leaf nodes only have one thing in them (no further split is possible, MSE for the train will be zero but will overfit for any other set -not a useful model)

Why this model?

This model has been covered in EE660 lecture. To jump into application, we chose this model for our task to familiarize deeper.

Advantages:

- Useful in Data exploration: Decision tree is one of the fastest way to identify most significant variables and relation between two or more variables. With the help of decision trees, we can create new variables / features that has better power to predict target variable.
- Less data preparation required: It is not influenced by outliers and missing values to a fair degree.
- Data type is not a constraint: It can handle both numerical and categorical variables.
- Non Parametric Method: Decision tree is considered to be a non-parametric method. This means that decision trees have no assumptions about the space distribution and the classifier structure.
- Can capture Nonlinear relationships

Disadvantages

- Overfitting: This problem gets solved by setting constraints on model parameters and pruning.
- Not fit for continuous variables: While working with continuous numerical variables, decision tree loses information when it categorizes variables in different categories.
- Cannot extrapolate.
- Decision trees can be unstable: Small variations in the data might result in a completely different tree being generated. This is called variance, which needs to be lowered by methods like bagging and boosting.

 No Guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees, where the features and samples are randomly sampled with replacement.

Model Parameters:

min_samples_split: *int, float, optional (default=2)*

It is the minimum samples for a node split that we discuss above.

If int, then consider min_samples_split as the minimum number.

*If float, then min_samples_split is a percentage and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.*

The minimum number of samples required to split an internal node. In this case, we tends to choose 2 or more samples. Normally we choose 2 as the default value, but in this case we want to investigate whether choose more samples for splitting can improve the model performance.

min_samples_leaf: *int, float, optional (default=1)*

It is the minimum number of samples for a terminal node that we discuss above.

If int, then consider min_samples_leaf as the minimum number.

*If float, then min_samples_leaf is a percentage and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.*

The minimum number of samples required to be at a leaf node. A leaf node. The splitting criterion is ‘mse’ as a default value. Choose the number of samples in one leaf node can smooth the regression model.

max_depth: *int or None, optional (default=None)*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

The maximum depth of the tree. There are multiple features in the dataset, the tree model tends to choose the most important features first when

performa regression. Thus, choose the correct depth of the tree can prevent underfitting and overfitting when implement the model

Other parameters in this tree model are selected by default since those parameters won't affect too much on the performance.

Parameter Tuning Procedure:

We use cross validation techniques to choose the best parameters. The method is using sklearn.model_selection.GridSearchCV. This method accept range of the hyperparameters we want to measure, perform cross validation and select the best combination of parameters that gives lowest error.
The parameter results after cross validation (GridSearchCV):
First Run of CV:
Result : *{'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 2}*

At this point we can see that the min_sample_split parameter is the same as the default value which is 2. Thus in the second run of CV, we can remove tuning min_sample_split parameter and set it to default directly when implementing the model.

As for the other two parameters, I would like to change the range of validation for max_depth to : [9,10,11] and min_sample_leaf to : [4,5,6,]
Second Run of CV:
Result : *{'max_depth': 11, 'min_samples_leaf': 5}*

Here we can see that parameter: min_sample_leaf has been tuned, lets fix it to 5 when implement the model

Finally, the max_depth parameter should be higher than the current range.

Thus lets change the range to [17, 18, 19, 20, 21, 22, 23] for the next run.
Third Run of CV:
Result : *{'max_depth': 21}*

The above 3 runs of CV are used for finding the proper range of the parameters in order to reduce the run time. After all the parameter ranges being confirmed, we run the CV again with all the parameters tested within the proper ranges
Final Run of CV:
Parameter Tuning Range:

max_depth: [20, 21, 22]

min_samples_split: [2, 3]
min_samples_leaf: [4, 5, 6]

Model Outputs and visualization:

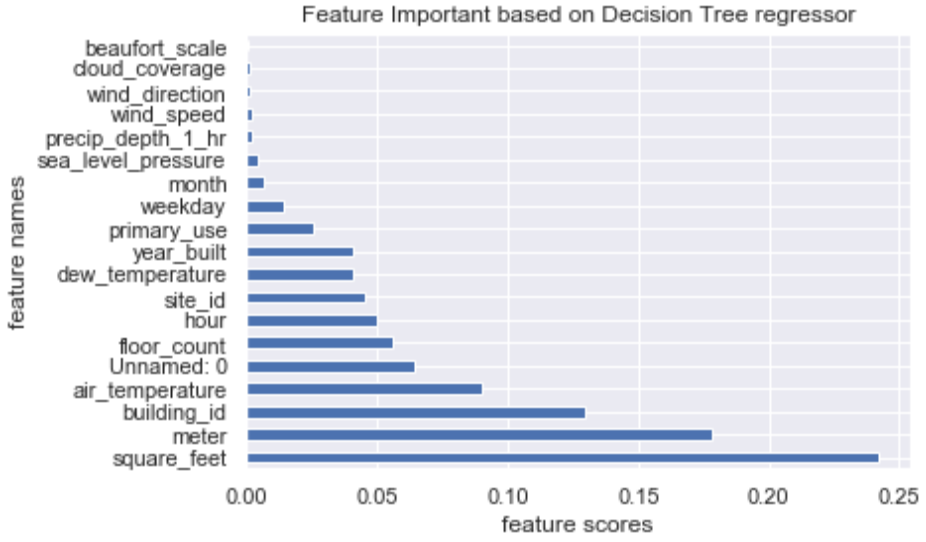
Final Parameter Selection:

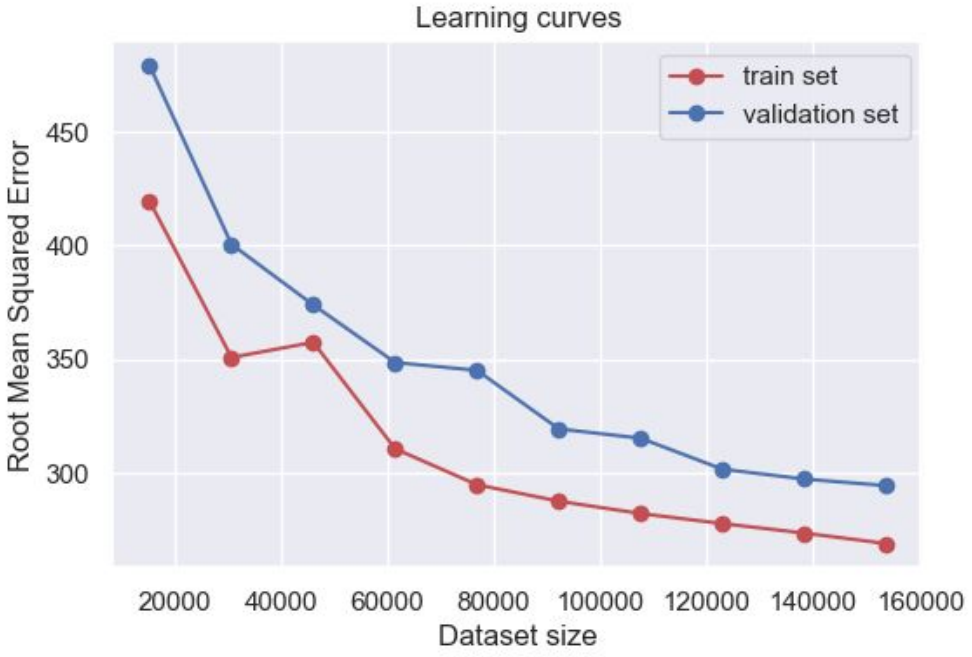
{'max_depth': 20, 'min_samples_leaf': 6, 'min_samples_split': 2}

Model Implementation:

RMSE on mini-batch training dataset: 237.32421458668134

correspnding feature importance plot:

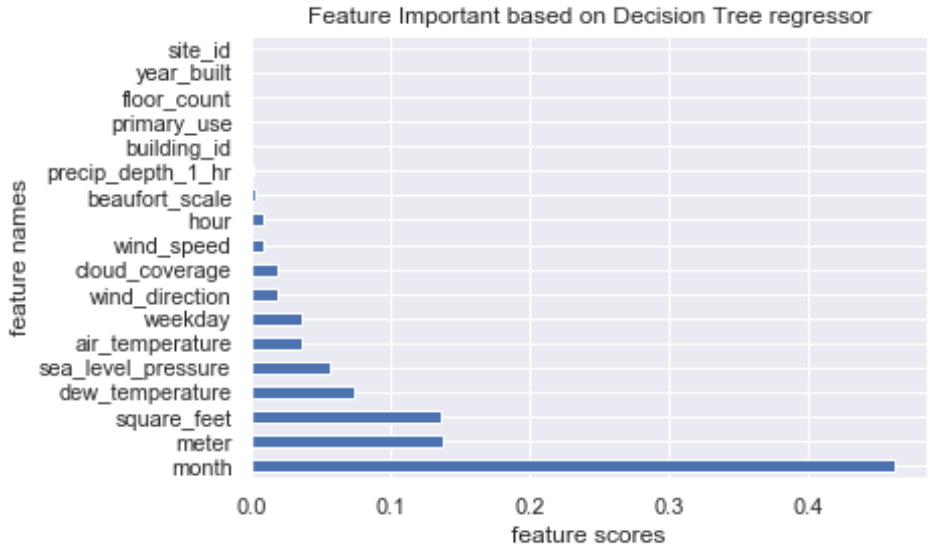




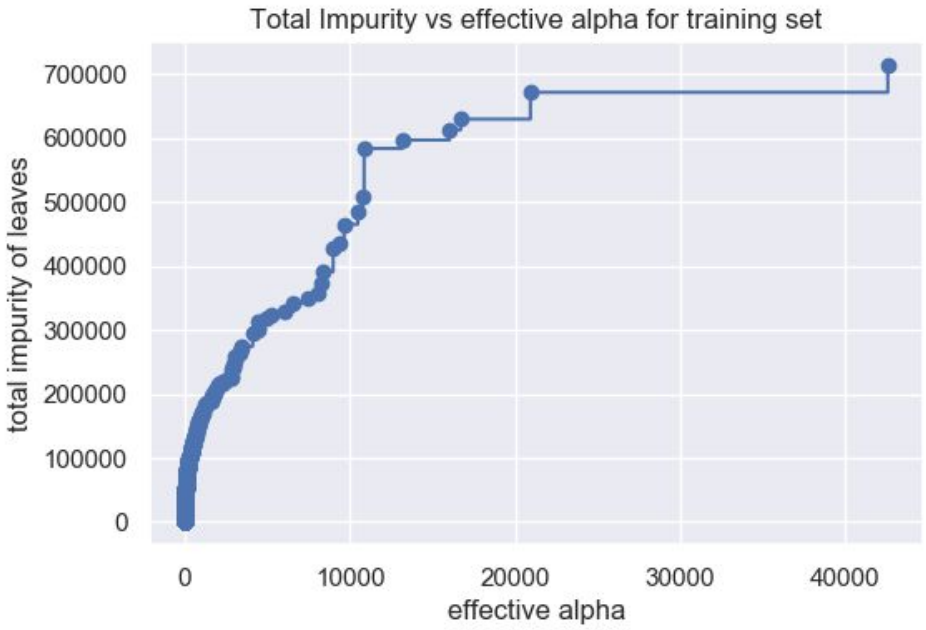
Learning Curve for the mini-batch training process:

RMSE on whole training dataset: 51318.32419808552

Corresponding Feature Importance plot:



Additional Visualization Result:



Model 2:

AdaBoost Regressor

Description:

In machine learning, boosting originated from the question of whether a set of weak classifiers could be converted to a strong classifier. Weak learner or classifier is a learner which is better than random guessing and this will be robust in over-fitting as in a large set of weak classifiers, each weak classifier being better than random. As a weak classifier, a simple threshold on a single feature is generally used. If the feature is above the threshold than predicted, it belongs to positive otherwise belongs to negative.

AdaBoost stands for 'Adaptive Boosting' which transforms weak learners or predictors to strong predictors in order to solve problems of classification.

For classification, the final equation can be put as below:

$$F(x) = sign(\sum_{m=1}^{M} \theta_m f_m(x)),$$

Here fm designates the mth weak classifier and m represents its corresponding weight.

Base Learner:

Decision Tree Regressor (max_depth = 20)

Since our last model is DecisionTreeRegressor, and it performs well on mini-batch training set, thus we chose to use the same model as before for AdaBoost ‘s base learner setting.

Halting Condition:

AdaBoost with a set of weak learners (learners that are only slightly correlated with the true process) are combined to produce a strong learner. Regularization via *early stopping* can provide guarantees of consistency, that is, that the result of the algorithm approaches the true solution as the number of samples goes to infinity.

Why this model?

This model has been covered in EE660 lecture. To jump into application, we chose this model for our task to familiarize deeper.

Advantages:

- Fast, simple and easy to program
- It has the flexibility to be combined with any machine learning algorithm and there is no need to tune the parameters except for T
- It has a high degree of precision
- Can capture Nonlinear relationships
- less susceptible to overfitting problems than other learning algorithms

Disadvantages

- It is from empirical evidence and particularly vulnerable to uniform noise
- Weak learner being too weak can lead to low margins and overfitting.
- Data imbalance leads to a decrease in Regression accuracy
- Training is time consuming, and it is best to cut the point at each reselection of the current classifier;

Model Parameters:

base_estimator: object: optional (default=None)

The base estimator from which the boosted ensemble is built. If None, then the base estimator is DecisionTreeRegressor(max_depth=3).

n_estimators: integer: optional (default=50)

The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

learning_rate: float: optional (default=1.)

Learning rate shrinks the contribution of each regressor by learning_rate. There is a trade-off between learning_rate and n_estimators.

loss: {'linear', 'square', 'exponential'}, optional (default='linear')

The loss function to use when updating the weights after each boosting iteration.

random_state: int, RandomState instance or None, optional (default=None)

If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator;

If None, the random number generator is the RandomState instance used by np.random.

Parameters we want to tune using CrossValidation:

base_estimator: DecisionTreeRegressor(max_depth = 20)

n_estimators

learning_rate

Parameter Tuning Procedure:

We use cross validation techniques to choose the best parameters. The method is using sklearn.model_selection.GridSearchCV. This method accept range of the hyperparameters we want to measure, perform cross validation and select the best combination of parameters that gives lowest error.

Note: Procedures of CV parameter tuning is similar to Decision Tree Regressor previously. We set GridSearch Range first, run the validation, adjust the range of search and finally get the best parameters. Thus here we just show the results for every search run for convenience.

The parameter results after cross validation (GridSearchCV):

First Run of CV:

Range: n_estimator: [10, 20, 30, 40], learning_rate: [0.1, 0.2, 0.3, 0.4, 0.5]

{'n_estimators': 40, 'learning_rate':0.5}

Second Run of CV:

Range: n_estimator: [50, 60, 70], learning_rate: [0.5, 0.6, 1.0]

{'n_estimators': 50, 'learning_rate':1.0}

Third Run of CV:

Range: n_estimator: [45, 50, 55], learning_rate: [1.0, 1.5]

{'n_estimators': 50, 'learning_rate':1.0}

Model Outputs and visualization:

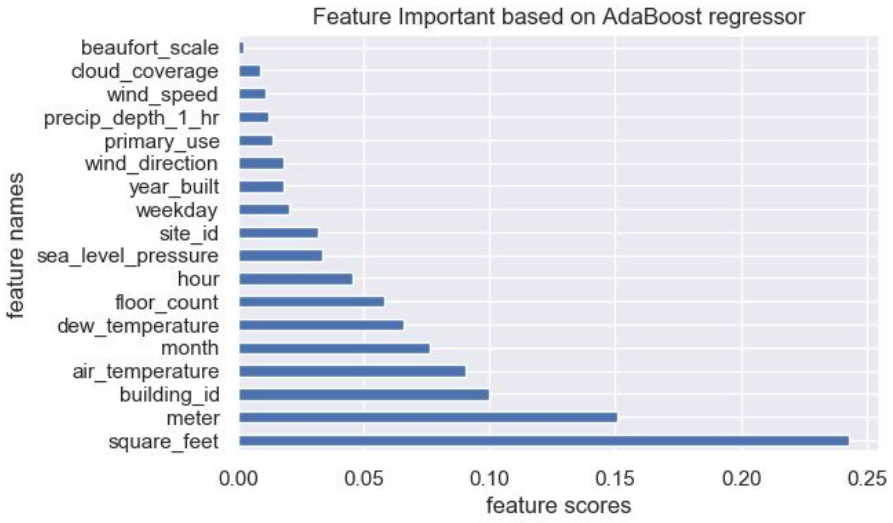
Final Parameter Selection:

{'n_estimators': 50, 'learning_rate':1.0}

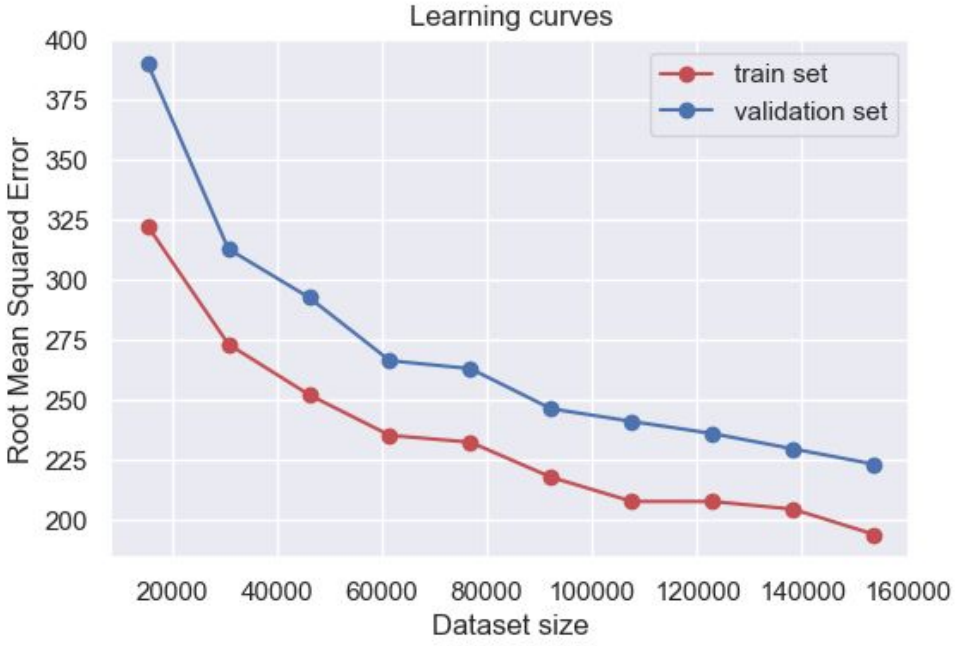
Model Implementation:

RMSE on mini-batch training dataset: 191.09569342066743

correspnding feature importance plot:

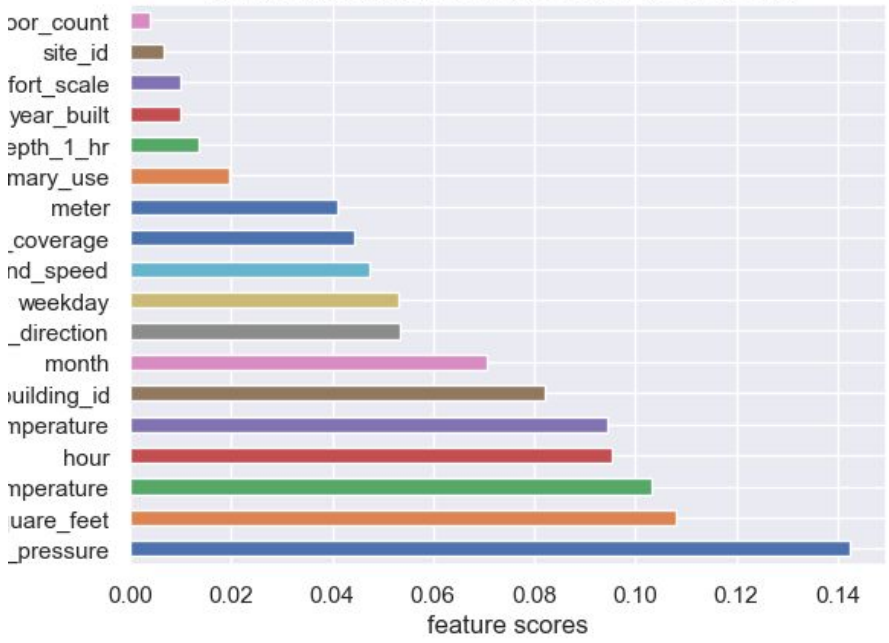


Learning Curve for the mini-batch training process:



RMSE on whole training dataset: 0.6595293065197682

Corresponding Feature Importance plot:

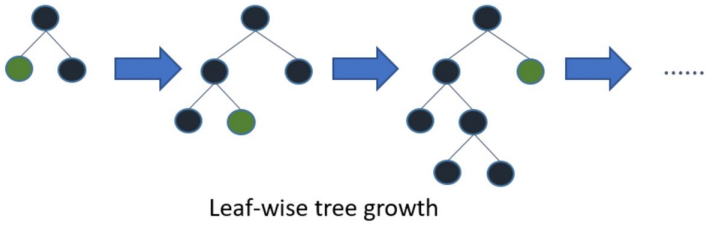


Model 3:

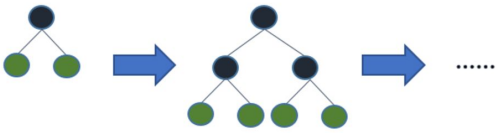
LightGBM

Description:

LightGBM is a gradient boosting framework that uses tree based learning algorithm. Light GBM grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithm grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm. Below diagrams explain the implementation of LightGBM and other boosting algorithms. As we can see, LGBM seems could choose which branch to grow.



Leaf-wise tree growth



Level-wise tree growth

Splitting Criterion:

Based on cost function, if the cost function is pretty low after the splitting, then should choose this one.

Halting Condition:

This is controlled by two parameters, the first one is n_estimators, the second one is early_stopping_rounds. Typically, n_estimator means the number of boost, when the values of boost does not change for early_stopping_rounds, it will converge.

Why this model?

This model has not been covered in EE660 lecture.

Advantages:

- Using less memory and supports GPU and parallel training. Quicker than most of the models.
- It is capable of performing equally good with large datasets with a significant reduction in training time as compared to XGBOOST.
- It produces much more complex trees by following leaf wise split approach rather than a level-wise approach which is the main factor in achieving higher accuracy. However, it can sometimes lead to overfitting which can be avoided by setting the max_depth parameter.

Disadvantages:

- The drawback of LGBM is that it is easily defeated by noisy data, the efficiency of the algorithm is highly affected by outliers as the algorithm.

 Will ignore inner dependencies between one feature. Like time serise feature, wind speed, cloud coverage.

Model Parameters:

Boosting_type*: string, values,optional (default=gbdt)*

This will specify the boosting type in building model, the gbdt means gradient boost.

Max_depth*: int, values, optional (default=-1)*

If the trend that keeps on growing with no real saturation insight, should set this parameter to "linear". If the curve that is showing promise of saturation then should set it to "logistic".

Num_leaves*: int, values, optional (default=20)*

It is the minimum number of the records a leaf may have. The default value is 20, optimum value. It is also used to deal over fitting

Feature_fraction*: float, values, optional (default=0.8)*

Used when boosting is random forest. 0.9 feature fraction means LightGBM will select 90% of parameters randomly in each iteration for building trees.

Bagging_fraction*: float, values, optional (default=0.8)*

Specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting.

Early_stopping_roung*: int, values*

This parameter can help speed up your analysis. Model will stop training if one metric of one validation data doesn't improve in last early_stopping_round rounds. This will reduce excessive iterations.

Lambda*: float, values*

Lambda specifies regularization. Typical value ranges from 0 to 1.

Parameter Tuning Procedure:

We use cross validation techniques to choose the best parameters. The method is using sklearn.model_selection.GridSearchCV. The parameter results after cross validation (GridSearchCV):

First Run of CV:

Result : *{'max_depth': 11, 'num_leaves': 150}*

In this case, first, I set the max_depth range from 5 to 17, each one is larger 3 than previous one, and num_leaves from 50 to 200, each one will larger 50 then previous one.

Second Run of CV:

Result : *{'max_depth': 9, 'num_leaves': 140}*

First Then I set the max_depth range from 8 to 14, and num_leaves from 140 to 160, with a gap of 10.

Third Run of CV:

Result : *{'min_child_samples': 26, 'min_child_weight': 0.001}*

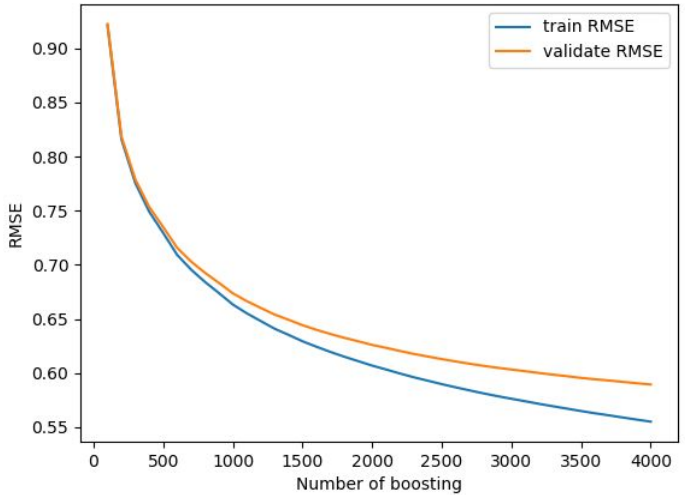
First Then I set the 'min_child_samples' range from 20 to 30, and 'min_child_weight' range from 0.1, to 0.01, to 0.001.

Fine_tuning for n_estimators:

Result : *{'n_estimators': 500}*

This one is a special, since after I use CV to find the optimal value, I find 1500 is the best, however, after I submit this result into Kaggle website, I get a score of 2.98. And when the n_estimator to be 4000, this score is 3.84, which is overfitting. So I plan to draw a learning curve with respect to n_estimators. So, it is easy to find that after the 500,the gap between train RMSE and validation RMSE begin to be larger.

Fine_tuning for features:



Result : *{drop_features:* 'weekday', 'floor_count', and 'primary', 'sea_ level_pressure'*}*

In total, by this LGBM feature importance import(see the graph below), we find there are 6 features which are important to both models (LGBM and decisive tree), that is, meter, square_feet, air_temperature, dew_temperature,

hour, site_id. And we also find something is a little strange, the sea_level_pressure is not so important, may be because some of the 15 site are inland city, so that the weather data of this city has no strong relationship with sea level. And also, I plan to drop ‘weekday’, ‘floor count’, and ‘primary’.

Model Outputs and visualization:

Final Parameter Selection:

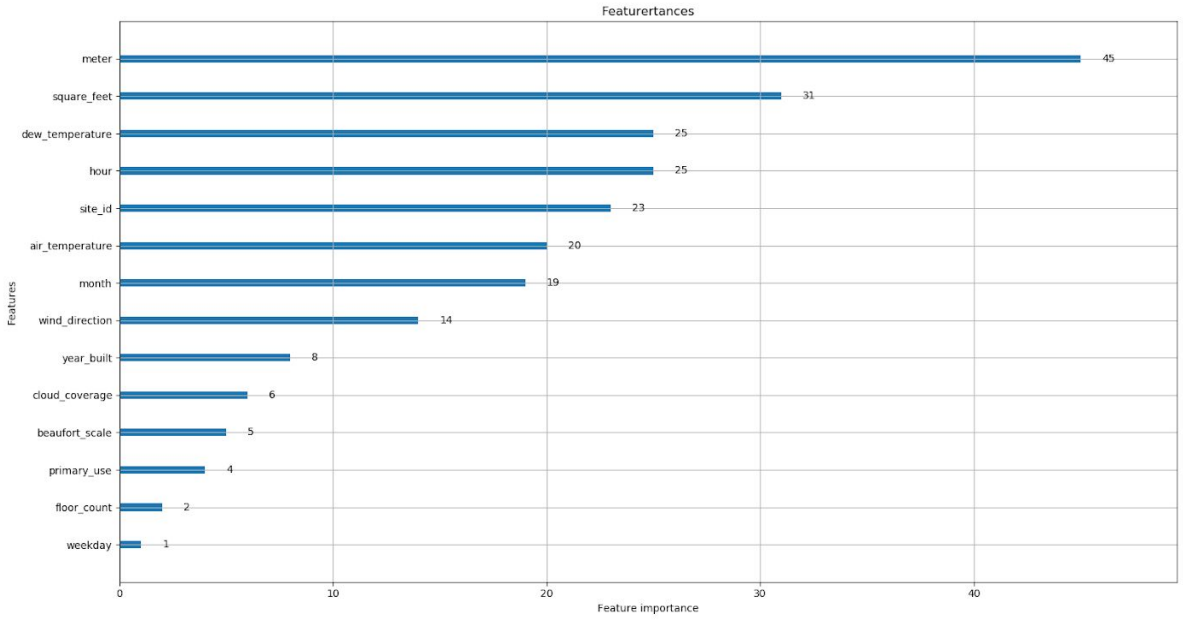
'max_depth': 11, 'num_leaves': 150,'n_estimators': 500,’learning_rate’:0.05

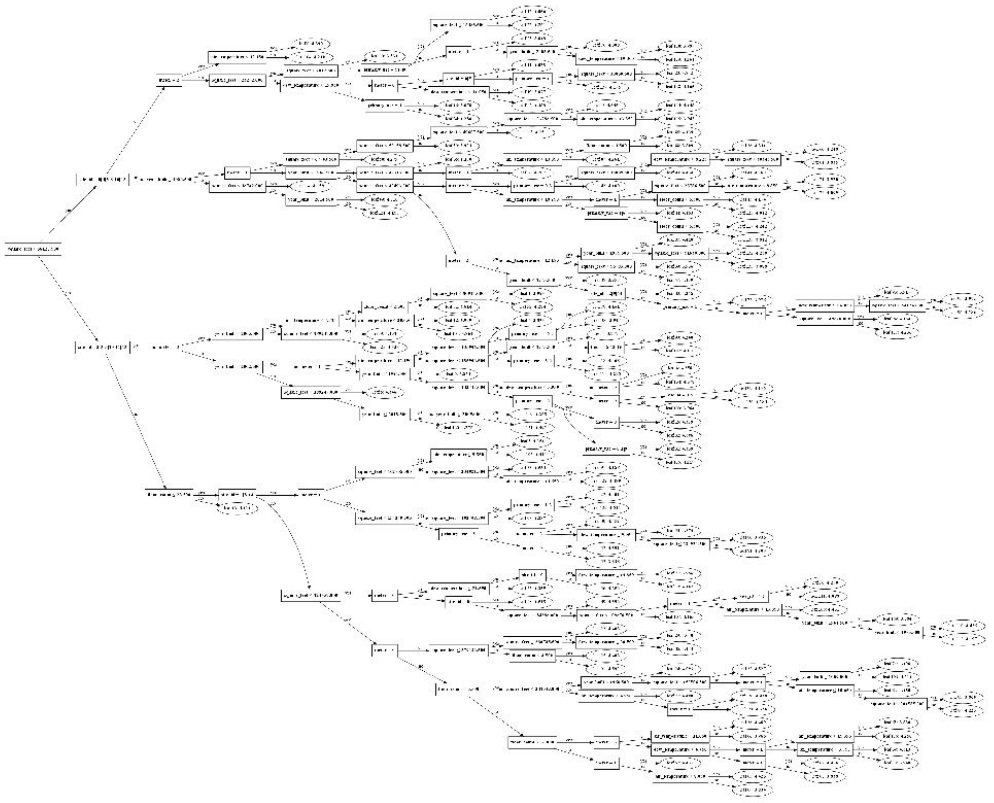
Model Implementation:

RMSE on mini-batch training dataset: 218.987132165465

rmse for whole dataset: 0.623976

correspnding feature importance plot and tree plot:





Model 4:

Prophet Time Series Predictor

Description:

Prophet is an open source library published by Facebook that is based on decomposable (trend+seasonality+holidays) models. It provides us with the ability to make time series predictions with good accuracy using simple intuitive parameters and has support for including impact of custom seasonality and holidays! We use a decomposable time series model with three main model components: trend, seasonality, and holidays. They are combined in the following equation:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon$$

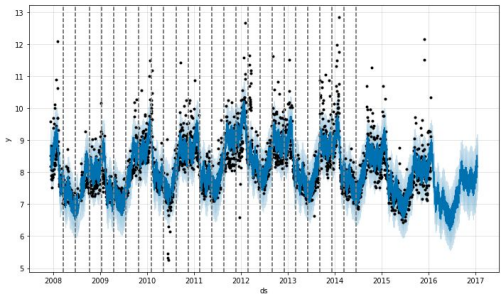
$g(t)$ piecewise linear or logistic growth curve for modelling non-periodic changes in time series

$s(t)$ periodic changes (e.g. weekly/yearly seasonality)

$h(t)$ effects of holidays (user provided) with irregular schedules

ε error term accounts for any unusual changes not accommodated by the model

Trend is modelled by fitting a piecewise linear curve over the trend or the non-periodic part of the time series. The linear fitting exercise ensures that it is least affected by spikes/missing data. And below pictures show the trend of a certain data.



Halting Condition:

Based on Log Joint Probablity, Prophet model tends to maximize this value. And if the improved value is lower than e-09. It will converge. See experiment details below.

Iteration 107. Log joint probability = 4476.98. Improved by 9.05829e-08.
Iteration 108. Log joint probability = 4476.98. Improved by 9.82254e-11.
Prophet model completed for building 753 and meter 0

Why this model?

This model has not been covered in EE660 lecture.

Advantages:

- Based on a time series model, so it is pretty suitable to make time series prediction.
- Hourly, daily, or weekly observations with at least a few months (preferably a year) of history
- Strong multiple “human-scale” seasonalities: day of week and time of year
- Important holidays that occur at irregular intervals that are known in advance (e.g. the Super Bowl)
- A reasonable number of missing observations or large outliers
- Historical trend changes, for instance due to product launches or logging changes

- Trends that are nonlinear growth curves, where a trend hits a natural limit or saturates

Disadvantages:

- Extremely time consuming if there are many features.
- The future prediction highly depends on previous data, not so highly depend on features. The inference part is worse than LGBM.

Model Parameters:

Growth*: ('linear', 'logistic'), string, optional (default=linear)*

If the trend that keeps on growing with no real saturation insight, should set this parameter to "linear". If the curve that is showing promise of saturation then should set it to "logistic".

Holidays: *'holidays', string, optional (default=None)*

Holidays are periods of time were the days have the same sort of effect each year. For example, if want to model the number of subscribers in a city where people migrate over the festive periods, you can put in the dates of the festive period in your model using the holidays parameter.

Changepoints: *'changepoints' ,string, optional (default=None)*

Changepoints are the points in data where there are sudden and abrupt changes in the trend. In a campaign case, if suddenly there are more than 50000 visitors to the websites.

Parameter Tuning Procedure:

For this model, since it is too slow, and actually will take upto more than 5 days to predict the results. In this time, since Prophet needs a more accurate model, so that I plan to drop one half days of one week, and drop one sixth hours in one day, and also drop 9 features, which are not so related to this dataset. After drop these features, I still find pretty time consuming. So I plan to use validation set to see the RMSE error. Due to this is a time series data, so the validation data should be selected from the training data randomly without replacement. And in this case, the validation set comes from the sample training dataset we build in the previous. The feature I drop is 'month', 'hour', 'weekday', 'year_built', 'floor_count', 'primary_use', 'wind_speed', 'cloud_coverage','sea_level_pressure'. And it will take more than 4 hours to see the results.

Model Outputs and visualization:

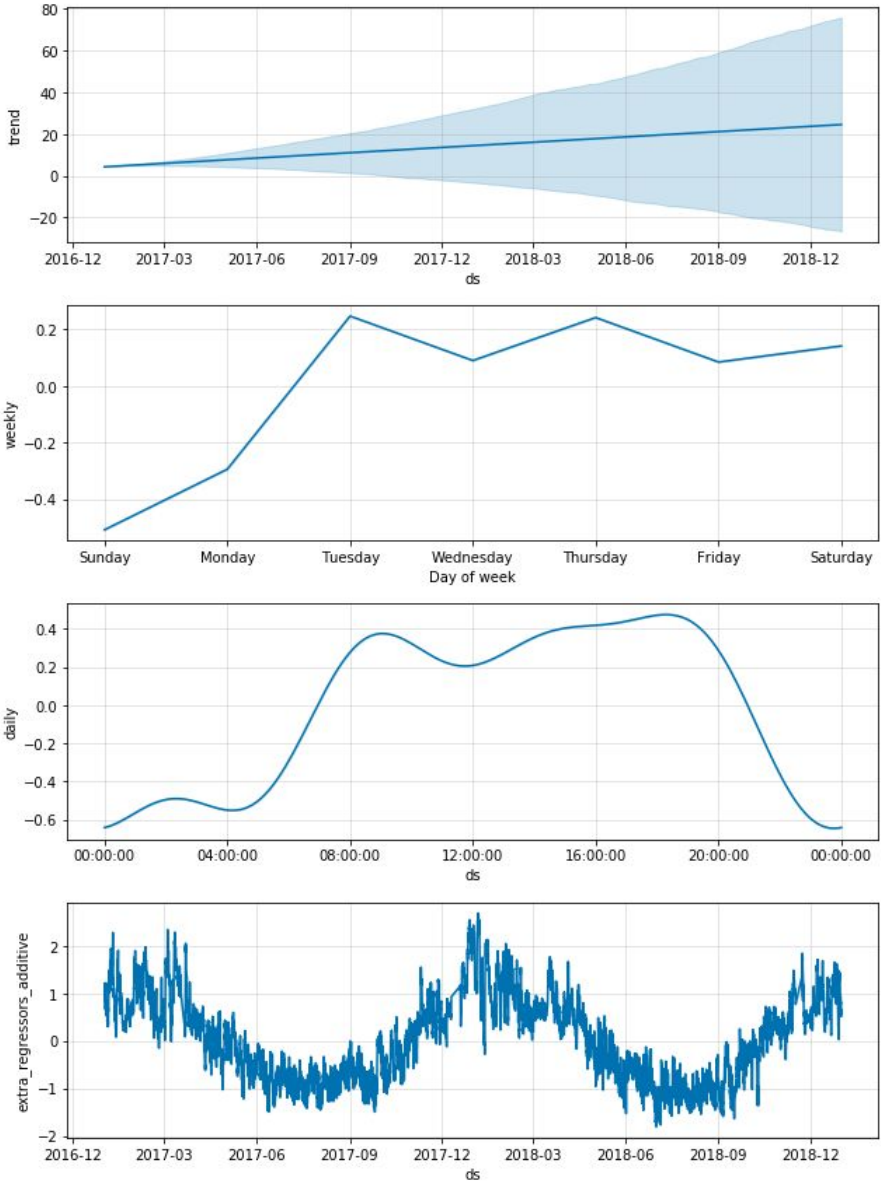
Final Parameter Selection:

{holidays:'holidays', drop features: 'month', 'hour', 'weekday', 'year_built', 'floor_count', 'primary_use', 'wind_speed', 'cloud_coverage', 'sea_level_ pressure'}

Model Implementation:

RMSE on mini-batch training dataset: 1025.3192192130623 (8 hours)

Corresponding prediction plot: The axis are one year, one week, one day, and two years respectively.



Comparison of Results: (Preserve 5 scientific number)

Models	RMSE (mini-batch set)	RMSE (complete set)
DecisionTreeRegressor	237.32421458668134	51318.32419808552
AdaBoost	191.09569342066743	0.659529306519768
LightGBM	0.829423	0.623976
Prophet	1025.3192192130623	**N/A**

From the Result, When using the mini-batch dataset on our models, the best performance is the AdaboostRegressor with the lowest RMSE. We can see that DTR and two Boosting algorithm produce approximately close RMSE when fitting with mini set. But Prophet, as time-series algorithm perform poorly on mini training set. Since we use moderated timestamp for mini set, the sample dataset only covers 6 hours in one day and 3 days in one week. So, the model may lost some information. If running on the whole data, it needs more than 5 days to yield results thus we test on the mini-batch data for this model.

From the table we can clearly recognize that DecisionTreeRegressor perform poorly on complete training set, which yields an RMSE of 51318. The reason is probably that Decision Tree is good at deals with small datasets, when the dataset grows larger, with noise increases concurrently, DTR tends to overfit the data. The candidate solution of this problem would be re-tuning the model parameter when implement the complete training set. But this will cost long running time which may be not capable for us to finish before project due.

4. Final Results and interpretation:

Final Model with best performance: Light GBM

Performance:

RMSE on complete training dataset: 0.623976

Score on Kaggle evaluation: 2.18

note: Kaggle is using different evaluation method

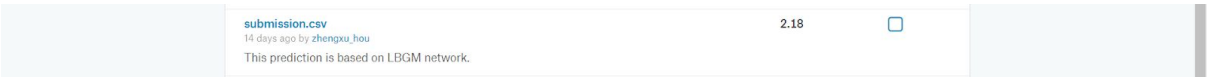
The RMSLE is calculated as

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

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

Kaggle Scoreboard:



Interpretation:

The performances are generally good on boosting methods, since boosting methods Light GBM grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithm grows level-wise. Light GBM is a very popular algorithm now because of it's processing speed and ability to handle big datasets.

Prophet is a time-series algorithm which is pretty new to us. We try to investigate this method on our dataset. However, Prophet needs integrated time dependent features which will require complete dataset. The run time for the model to fit the whole training set is too long, the approximate time will be 5 days. We couldn't finish fitting this model in time limit, but we think that Prophet model will probably get best score among other models since the dataset contains many time dependent features.

5. Contributions of each team member

Data Preprocessing and feature engineer: Zuoqian Xu, Zhengxu Hou
Models:
Decision Tree Regressor: Zuoqian Xu
Prophet: Zuoqian Xu
Light GBM: Zhengxu Hou
AdaBoost Regressor: Zhengxu Hou

6.Summary and Conclusions

With four machine learning algorithms being investigated and implement to real world data. We have gain insights about those models and their characteristics. In the future, when we work as data scientists, we will often need to deal with very large scale data like we have in our project. After this project, we have familiar with tree, boosting, time-series models, their parameters and in what situation and data features we should consider using those models.

7. Reference

[1] “Decision Tree Regressor explained in depth [online] avaliable: ”https://gdcoder.com/decision-tree-regressor-explained-in-depth/

[2] “ sklearn.ensemble.AdaBoostRegressor ” [online] avaliable: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html

[3] “ Welcome to LightGBM’s documentation! ”[online] avaliable: https://lightgbm.readthedocs.io/en/latest/

[4] “Forecasting with Prophet How to make high quality forecasts” [online] available: https://towardsdatascience.com/forecasting-with-prophet-d50bbfe95f91

[5] “ASHRAE -Start Here: A GENTLE Introduction” [online] avaliable: https://www.kaggle.com/caesarlupum/ashrae-start-here-a-gentle-introduction