

COMP9417 Group Project

Building Energy Consumption Prediction

1. Introduction

Due to the increasing climate change and global warming, energy consumption and emission of buildings, which are remarkably associated with the impact of environment, has been gradually considered as one of the most significant aspects of buildings construction as well as cost and site. Lots of elements related to temperature, weather and topography are all under the lower energy consuming consideration of designers. Information can be utilized by investors to predict annual energy consumption priorly for promotion of energy usage.

In this project, we will implement three models, random forests, ridge and xgboost, which are trained by tens of millions of data, to predict the annual energy consumption of specific buildings. This project and associated data come from Kaggle: [ASHRAE - Great Energy Predictor III](#).

This project has been completed about two years ago, but it is attractive enough to enroll lately since it is close to reality, has interesting topic and most important, moderate difficulty to handle.

Data, especially features parts, contains very different data types and many missing values represented as “NaN”, a relatively complicate preprocess algorithm is produced to process and reconstruct dataset. Next, three models with 5-fold cross-validation are implemented to train, then tuning, and finally predict the target value corresponding to annual energy consumption of every building in dataset. What’s more, the final predicted result will be submitted to Kaggle for evaluation and ranking.

2. Data processing and method determination

In this section, we will firstly talk about dataset preprocess, including features removal, data types change and data value reconstruction. Following to this, three models are implemented for training, tuning and prediction. preprocessed dataset without sampling subset is used for training to promise whole information gain and low error. Mean squared error (MSE) and 5-fold cross-validation is used for parameter tuning measurement, and through comparison of every model’s prediction, a “best” model would be selected to predict once again and submit result to Kaggle for evaluation and ranking.

2.1 Preprocessing dataset

The original dataset contains five individual datasets, respective information shows below.

building_metadata.csv	site_id, building_id, primary_use, square_feet, year_built, floor_count
train.csv	building_id, meter timestamp, timestamp, meter_reading
weather_train.csv	site_id, timestamp, air_temperature, cloud_coverage, dew_temperature, precip_depth_1_hr, sea_level_pressure, wind_direction, wind_speed
test.csv	row_id, building_id, meter, timestamp
weather_test.csv	site_id, timestamp, air_temperature, cloud_coverage, dew_temperature, precip_depth_1_hr, sea_level_pressure, wind_direction, wind_speed

Table 1. datasets and corresponding features

Firstly, for train and test set, we combine them respectively with corresponding weather and building information sets, produces an initial training and test sets with complete 15 features. Since datasets contains tens of millions of samples, which may cause severe problem of memory leakage if machine has not enough memory allocation, we implement an API called `skmem.py`⁽¹⁾ written by a competitor on Kaggle that remarkably decrease the memory usage of data file, its source and citation can be found in README file. Then, we notice that feature “primary_use” is valued by string likes Education and Office, and we are not in the circumstance of natural language processing, so it is better to convert them to number for easier fit and convergence. “Categorical” type is selected because it can automatically categorize different elements and assign same value to the same elements. After this process, feature “primary_use” is converted to number 1, 2, 3...

Secondly, there are lots of missing value in datasets, which is represented as “NaN”, we count number in every feature, as shown below, and find that more than half samples in feature “year_built” and “floor_count” are missing, so we drop these two features preventing to affect performance of model badly. As for other features with missing value, we fill these samples with mean value of this feature.

```

building_id      0.000000
meter            0.000000
timestamp        0.000000
site_id          0.000000
primary_use      0.000000
square_feet      0.000000
year_built       59.148082
floor_count      82.724073
air_temperature  0.055115
cloud_coverage   46.611607
dew_temperature  0.148848
precip_depth_1_hr 18.319990
sea_level_pressure 5.585245
wind_direction   6.698144
wind_speed       0.248346
dtype: float64

```

Figure 1. the number of missing values in every feature

Thirdly, feature “timestamp” has different value type compared to other features. For instance, 2016-01-01 00:00:00, which is not suitable for fitting and reaching convergence, hence we separate this feature to three parts, “hour”, “day”, and “week”. we ignore “month” and “year” because “month” change slightly and “year” is constant (i.e., 2016), they may lead to higher correlation between two sub-datasets and overfitting. Furthermore, feature “timestamp” has been dropped.

Finally, we use the natural logarithm of one plus ($\text{numpy.log1p}()$ ⁽²⁾) the target value (“meter_reading” column), to make the data closer to Gaussian distribution, and compress in a relatively small interval, similar with normalization, since there are many value of targets have huge difference like 812.3919 and 0.1055. 5 samples of the final preprocessed dataset show below. The whole preprocessed dataset contains 20216100 samples, 15 features and 1 target.

	building_id	meter	meter_reading	site_id	primary_use	square_feet	air_temperature	cloud_coverage
10000	665	0	1.160021	5	1	8773	4.0	1.900424
10001	666	0	1.029619	5	1	12045	4.0	1.900424
10002	667	0	1.064711	5	1	15715	4.0	1.900424
10003	668	0	3.414443	5	1	47275	4.0	1.900424
10004	669	0	2.104134	5	1	4768	4.0	1.900424

dew_temperature	precip_depth_1_hr	sea_level_pressure	wind_direction	wind_speed	hour	day	week
3.0	0.796416	1016.084656	110.0	4.1	4	1	53
3.0	0.796416	1016.084656	110.0	4.1	4	1	53
3.0	0.796416	1016.084656	110.0	4.1	4	1	53
3.0	0.796416	1016.084656	110.0	4.1	4	1	53
3.0	0.796416	1016.084656	110.0	4.1	4	1	53

Figure 2. 5 samples of preprocessed dataset

2.2 5-fold cross-validation and root mean squared error

k-fold cross-validation is implemented to evaluate the predict performance of models as well as handle overfitting. Also, mean squared error (MSE) is used to assess the fitting of the model, and according to this metric, we tune the parameters. Its function is:

$$MSE = \frac{1}{m} \sum_{i=0}^m (y_i - f(x_i))^2$$

where m is the size of training set, y is target, x is samples of features and f(x) is the predict function.

To be specific, due to the tremendous data level, we select 5-fold cross-validation instead of 10-fold cross-validation to decrease the cost of computation⁽³⁾. Besides, for easier representation and expression of predict result, we use radical to compress the value to be the same level compared to the target value. Therefore, we select root mean squared error (RMSE) to evaluate the predict performance of model and tune parameters.

3. Implementation

In this section, we will firstly utilize three different models including random forests, ridge and xgboost to perform training and validation with the same dataset and tune each parameter according to RMSE. Consequently, comparing their predict performance respectively and finally finding out the “best” model to solve this energy consumption prediction problem.

3.1 Random Forests

Based on the theory of ensemble learning, random forests are constructed by a certain number of decision tree created through bootstrap sampling, every decision tree is trained with random and put back sample dataset, which promises each tree’s dataset has intersection and then is unbiased. Every individual decision tree gives prediction, and random forests, based on the thought of bagging, use average of individual predictions as new prediction result. It performs better compared with support vector regression (SVR), neural network, and some other models in some case, also is good at handle overfitting⁽⁴⁾. This algorithm can be performed as following figure.

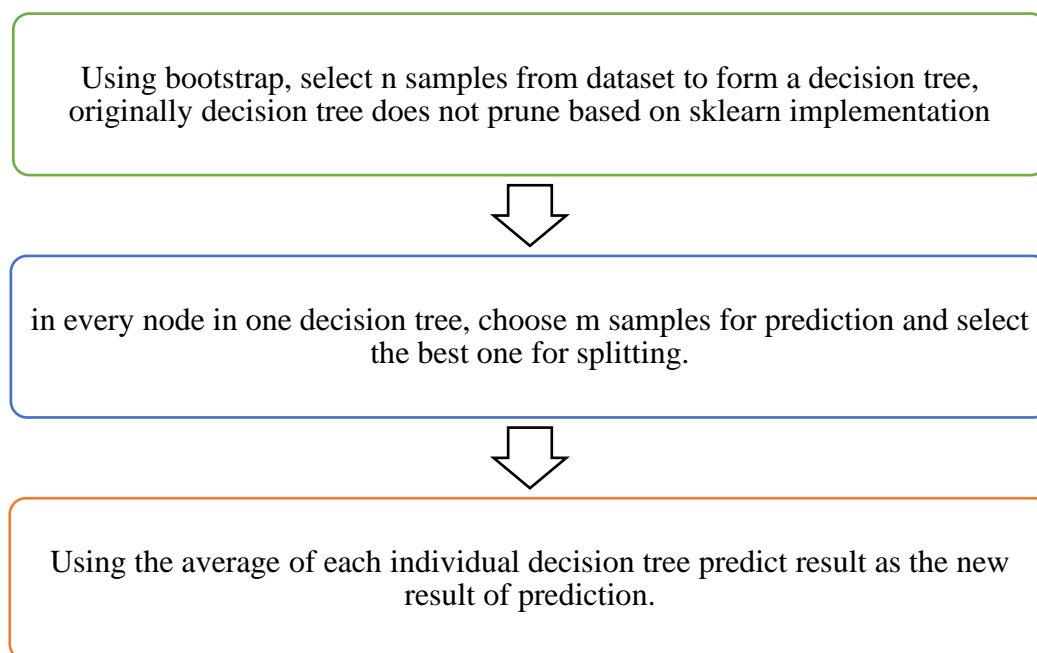


Figure 3. flow chart of random forests algorithm process

We first implement random forests to solve this prediction problem with some pre-set parameters, because the default random forests regressor in sklearn would cost too long in every iteration and do harm to memory management when it comes to 20216100 samples in training set.

We set **max_sample** to **300000** since bootstrap is enabled, **max_feature** to **0.5**, **min_sample_leaf** to **3** and **n_jobs** to **-1** to make every core in CPU work. The initial random forests regressor can be implemented like:

```
RandomForestRegressor(n_jobs=-1, max_samples=300000, max_features=0.5, min_samples_leaf=3)
```

Then using 5-fold cross-validation and root mean squared error (RMSE) to evaluate random forests model after each training and thus tuning parameters.

(1) Firstly, we try to find out the best $n_{\text{estimators}}$ (i.e., the number of trees in model) relying on the average of RMSE after training complete. The range of the parameter value is 10 to 150 and change every 10. The relation between the number of trees and average of RMSE shows in figure 4.

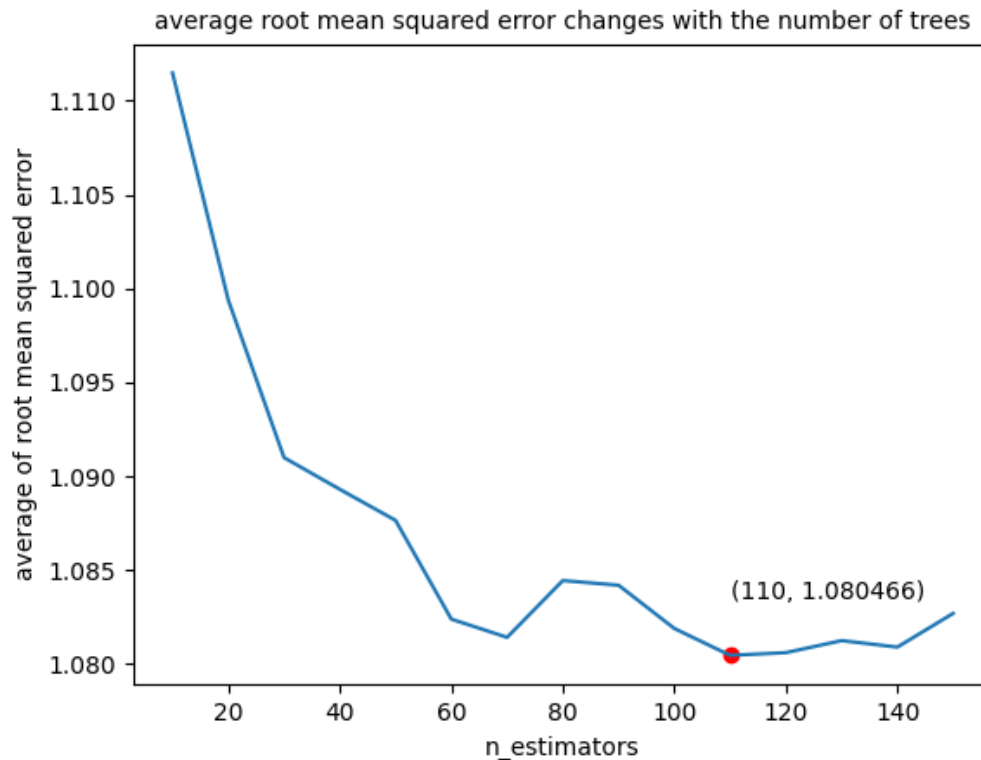


Figure 4

When the number of trees reach to 110, model has the lowest root mean squared error, compared to the default setting in sklearn, raise by 10 from 100 trees. Therefore, random forests regressor changes to:

```
RandomForestRegressor(n_jobs=-1, n_estimators=110, max_samples=300000, max_features=0.5,  
min_samples_leaf=3)
```

(2) Secondly, we turn to evaluate the maximum depth of trees. Since the dataset is enormous, the depth should be restricted to cut down the cost time and set to slightly higher to avoid underfitting, but we start at a relatively small depth to avoid missing the global minimum. The range is 3 to 50 and change with 2 plus no restriction on depth of trees. The relation between the maximum depth of trees and average of RMSE shows in figure 5.

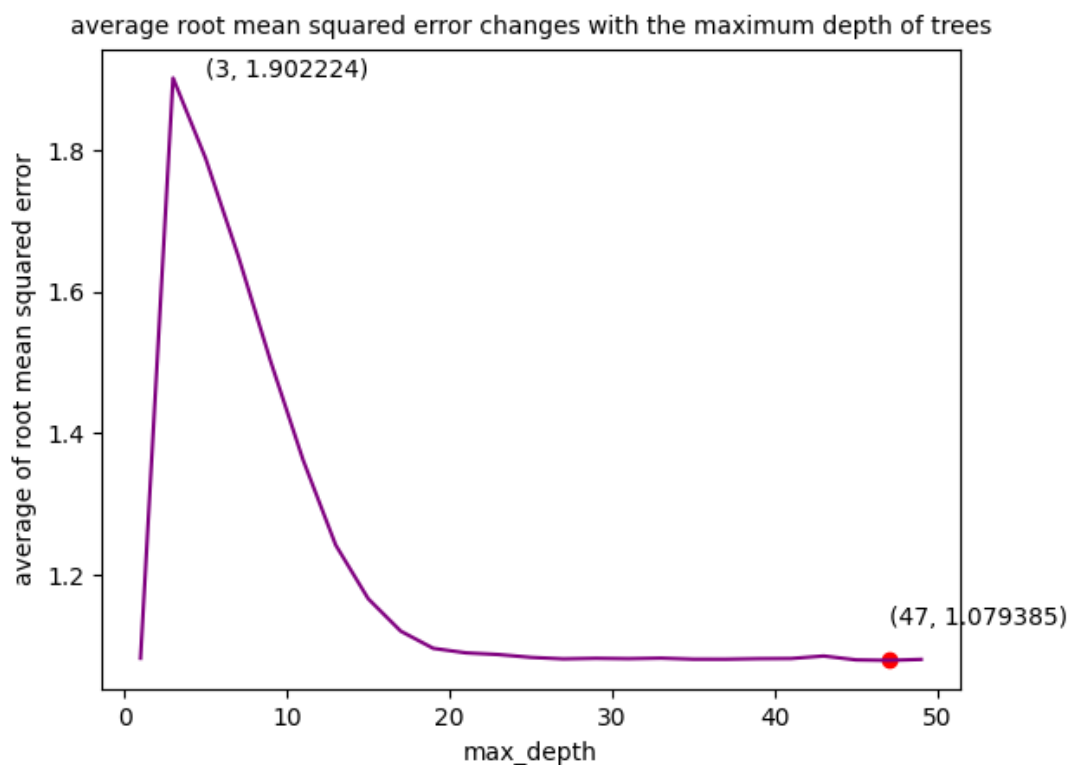


Figure 5

From the graph, if the maximum depth has no restriction, model could remain in a low prediction error level, however, the small depth of trees would lead to high prediction error, and the error meets approximate linear decline with the depth grow until about 15, then starts to be fluctuation ending at depth of 47. Therefore, we find out the “best” maximum depth of tree for good fit and acceptable time cost. The random forest regressor now changes to:

```
RandomForestRegressor(n_jobs=-1, n_estimators=110, max_depth=47, max_samples=300000,  
max_features=0.5, min_samples_leaf=3)
```

(3) Thirdly, we move to evaluate the minimum samples in a leaf node.

Since the decision tree model in sklearn does not implement prune, to avoid overfitting and decrease time cost, this parameter should increase from default 1 aiming to smoothing model. The parameter ranges from 1 to 10, changes with 1. The relation between the minimum samples in a leaf node and average of RMSE shows in figure 6.

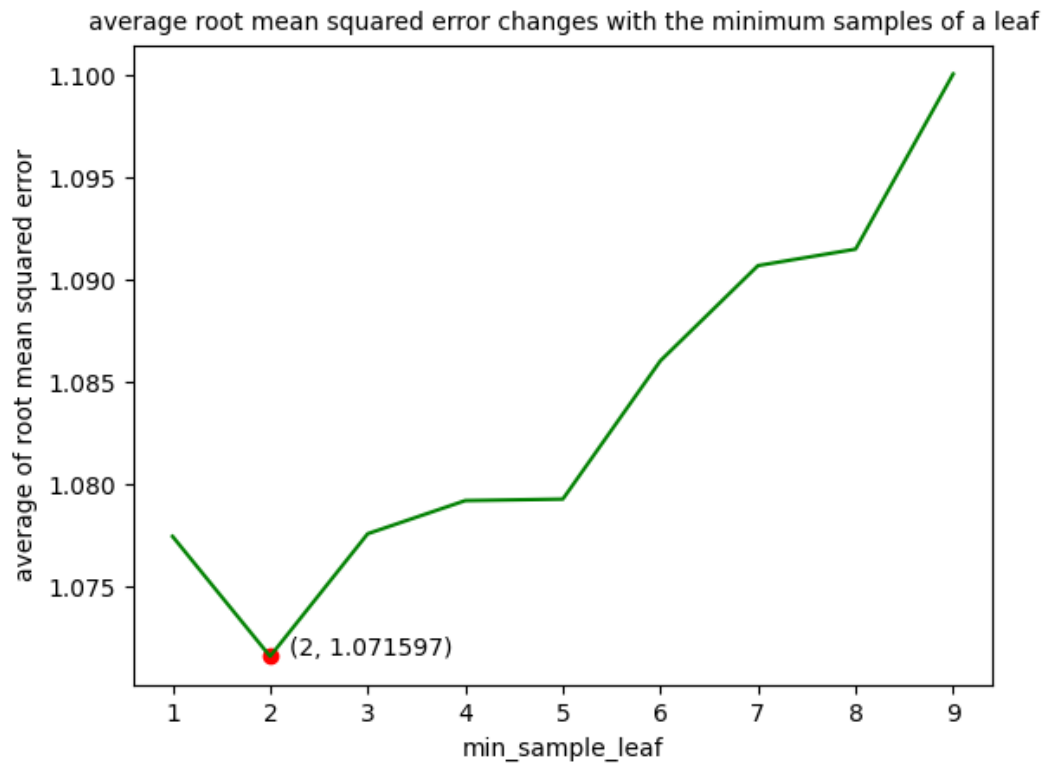


Figure 6

The graph indicates that average of RMSE meets fluctuation all the time but drops to the minimum when the minimum number of samples in a leaf reaches to 2 and then keeps rising. Hence, the “best” number of samples in a leaf can be determined as 2, which means that it makes tree deeper and contains lowest error but would take more time due to higher model complexity. The random forests regressor now changes to:

```
RandomForestRegressor(n_jobs=-1, n_estimators=110, max_depth=47, max_samples=300000,  
max_features=0.5, min_samples_leaf=2)
```

(4) Next, we will evaluate the maximum number of features when trees figuring out the best split. In sklearn's implementation, it defaulted using every feature to gain all information, but would do harm to time cost when the dataset is enormous. To decrease the complexity of model and also keep a promised low error, we set the parameter `max_features` range from 0.1 to 0.9, changes with 0.1 (fraction of whole number of features) plus keywords "auto", "sqrt" and "log2" (all features, root of all features and logarithm of all features respectively). The relation between the maximum number of features and average of RMSE shows in figure 7.

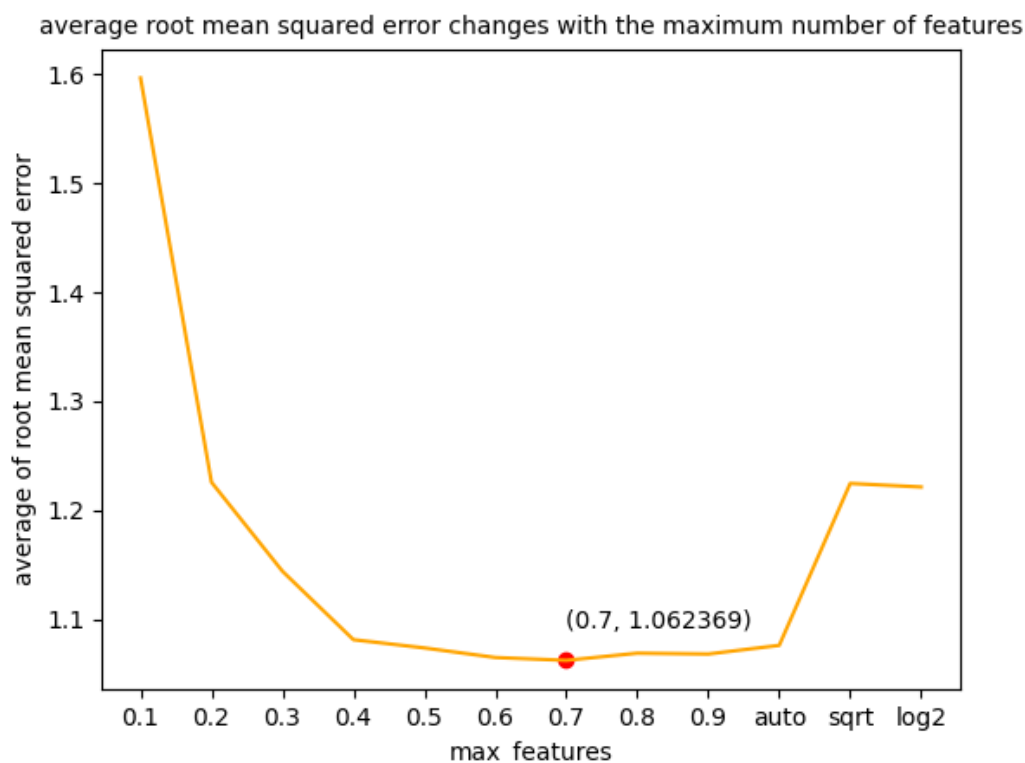


Figure 7

The figure shows that average of root mean squared error decrease consistently until 0.7 proportion of all features are used for splitting. The proportion continue to increase and error also increase slightly but still remain in a low level until all features are used. However, root and logarithm of all features would considerably affect the error and worsen the performance of model. Therefore, the "best" proportion of number of features is 0.7, and the random forests regressor now changes to:

```
RandomForestRegressor(n_jobs=-1, n_estimators=110, max_depth=47, max_samples=300000,  
max_features=0.7, min_samples_leaf=2)
```


(5) Finally, the number of samples trained in each tree is evaluated. More samples are used in each tree would promote the performance of prediction but also give burden on computation cost. We try to find one “best” parameter `max_samples` to balance these two elements. This parameter is set to range from 50000 to 750000, changes with 50000, we cannot set more higher value because of hardware performance restriction. The relation between average RMSE and the number of samples shows in figure 8.

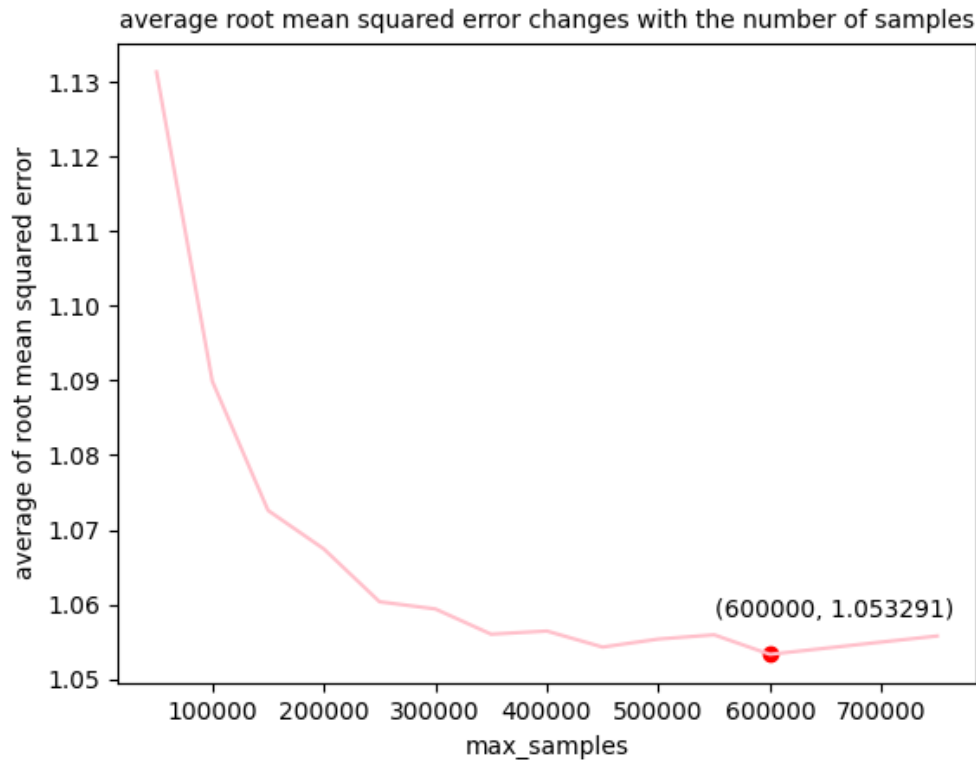


Figure 8

The figure indicates that average of RMSE remains decline continuously with some fluctuation until the number of samples becomes 600000 in every tree for training. Following to this, the error begins to raise slightly, hence 600000 is the “best” parameter value making model reach to lowest error. Although the computation time is a little bit long, it is acceptable and promised in good predict performance.

To summarize, after five parts tuning, we finally get the “best” random forests regressor with following form:

```
RandomForestRegressor(n_jobs=-1, n_estimators=110, max_depth=47, max_samples=600000,  
max_features=0.7, min_samples_leaf=2)
```

We implement this final model perform prediction with 5-fold cross-validation once again, and get the final RMSE, **1.0560270116488915**, which is a promised low error indicating good performance of prediction.

Also, we can explore the importance of each features using the final model. each importance of features is normalized and sum up to 1, the higher the value, the more important the feature is. Details presents in figure 9.

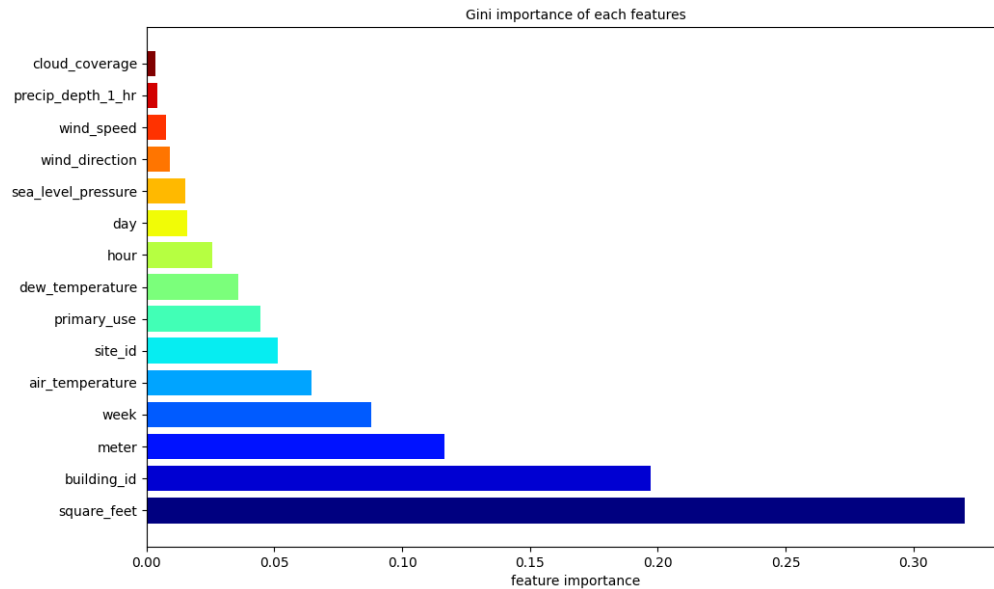


Figure 9

From the graph, we can observe square_feet is the most important feature contributing to beyond one third information and cloud_coverage is the least important with little information utilized to model. The similar unimportant features are precip_depth_1_hr, wind_speed and wind_direction. With this conclusion, we can further optimize the preprocess of dataset like dropping these unimportant features, thus reduce the dimension of dataset, complexity of model and also computation cost. This is a direction of future improvement but is not including in this report.

3.2 Ridge

We choose another simple regression model to test this dataset is a complex challenge. It is the ridge regression model. Among the parameter of ridge regression model in sklearn, the only one could affect the error of the output weights is Alpha or we can say regularization strength. We first need to find the Alpha which could lead to the lowest error prediction. With flitting, there should be 15 features in the dataset, which is not a small number. As we observed in homework1, with the Alpha increasing, the weights of all features will approach to 0. With such many features, the weights should be all very low. And also, our target values have been changed into $\log(Y)$. Lower target values will lead to lower weights as well. Therefore, we could predict that the Alpha should be a very large number. Usually, we will get a best alpha after many iterations. However, this database is a very large one (over 20 million samples). With our hardware, every time we got a output with a certain Alpha costs 50-60 seconds. It is impossible to iterate this step for a dramatic amount to find the best Alpha, which will cost several days. Therefore, we try to find a way to confirm the approximate value of alpha. At first, we set alphas in the `alpha_list` with a very large step.

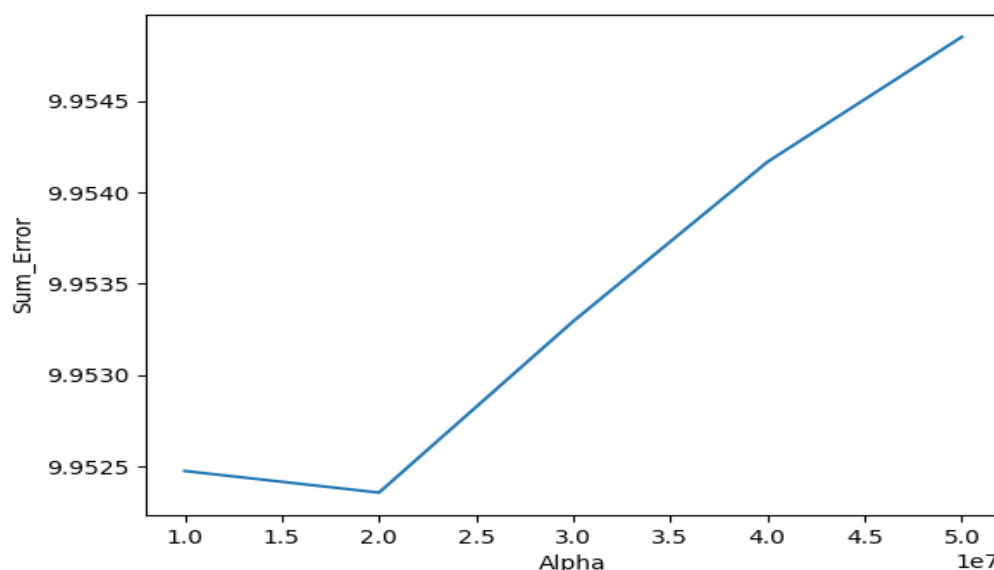


Figure 10. Error-Alpha when Alpha between 10000000 and 50000000

As Figure, we set the step as 10000000, and could find the alpha which will lead to the minimal error is between 10000000 and 30000000.

Then, we set the two neighbour Alpha value as the range of next iteration's range of Alpha, and the new step is equal to the difference of the two Alpha divide a certain number, which is equal to the number of points in new plot minus 1 and it should bigger than 3. In this case, we choose 5. This will make the minimal point in the next iteration are still between the new Alpha range but not the two magrine, which means we could still pick two neighbors of the minimal point and iterate the same steps again and again.

When the size of step is very small (< 0.5 , the distance of two neighbors < 1), we could stop the iteration, change the step of Alpha to 0.1 and try to find the final best

Alpha between the last range, which have been very small.

And with the limitation of the hardware, continuous processing will lead to out of memory, we split these iterations into 12(= $\log_5 10000000$) single process to keep the usage of memory and get the final range.

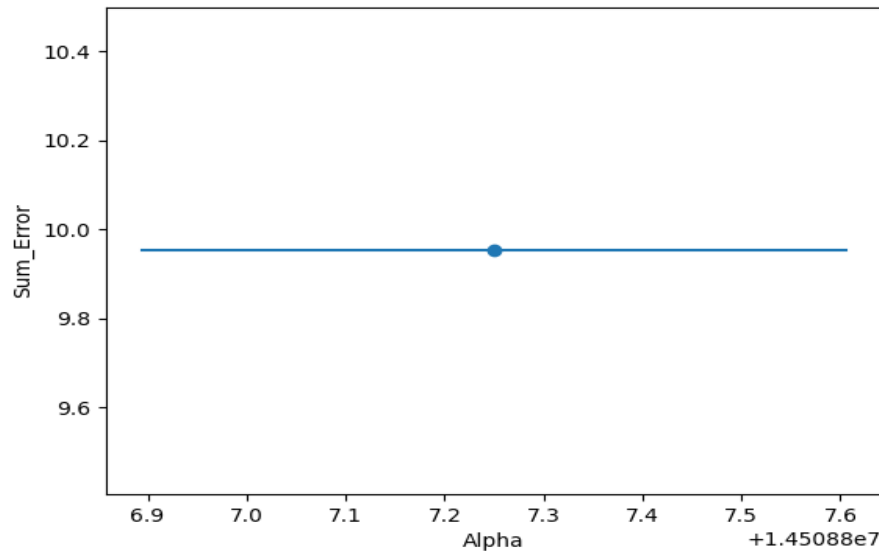


Figure 11. Error-Alpha when Alpha between 14508806.89375 and 14508807.60625

This process saved us lots of time. We finally find when Alpha=14508807.3, the error of ridge model on test set will be lowest.

From Figure 12, we could find the ridge model set the site_id as the most important feature and some other features that we think should be important such as square_feet, temperature is very low. As an ID feature, it is actually the foreign key in the data set. Therefore, maybe we should delete some of them in the preprocess. On the other hand, we think the low weight of some features and the high error of the ridge model with best Alpha shows it should not be a good choice here as a training model.

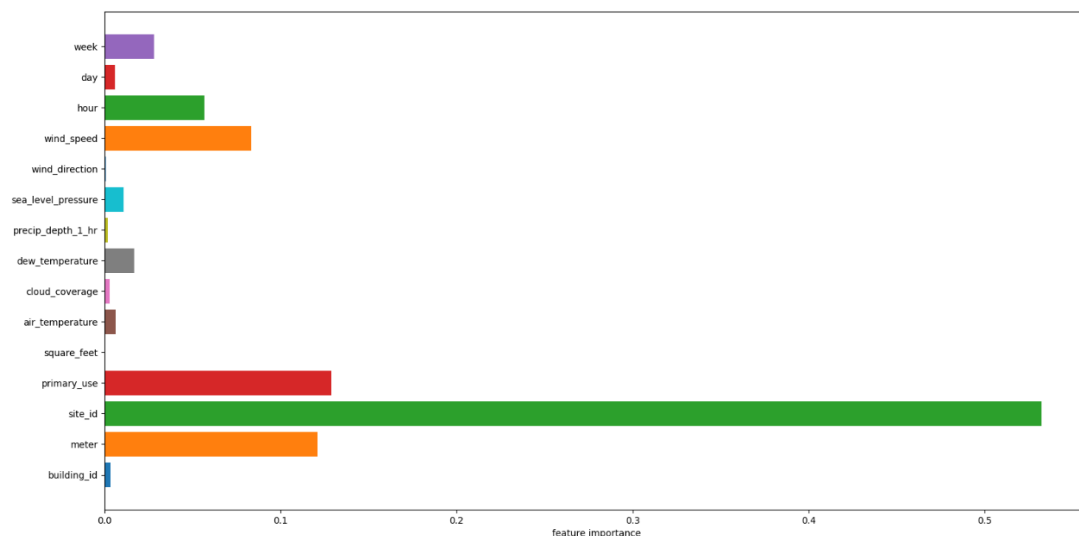


Figure 12. The importance of every feature from the best Ridge model

3.3 XGBoost

Xgboost is a popular model in machine learning area. XgBoost is basing on GBDT algorithm. Compare to GBDT, xgboost are improved in algorithm to save more memory and parallel computing is possible in Xgboost, which means faster processing.

3.3.1 Classification and Regression Tree (CART)

CART is a kind of binary tree which could implement on continuous data. The aim of CART is to split X into two parts which the two parts' Y could get the minimal square error. After several, the leaf will only cover a small range of features' value and could directly get the approximate value of Y.

3.3.2 Gradient Boosting Decision Tree (GBDT)

GDBT model is a kind of boosting model. It is a little like Random Forests, GDBT is also a model that consist of several decision trees. After GBDT fit the first CART, it will compute the loss between predicted value and true target value. Then, GDBT will form another CART. In the new tree, the loss of previous tree is set as the new target value of the new tree, so the aim of the new tree is to minimize the loss from previous tree. The number of trees is basing on the parameters

There are several important parameters in Xgboost, we should optimize them generally.

(1) First, we need to adjust parameters of the CART. The most important is the number of CART we need to generate, which should influent the final output dramatically. We should first set all the parameters are default, and keep increasing `n_estimators` (step should be 10) to find the best number.

(2) And then is the performance of every single tree should be also important. We will try to adjust `max_depth`, which is the most important parameter to form a tree. And also, we could find that `min_child_weight` is a very important parameter for the output in the document of xgboost, and it is also related to the depth of tree. Therefore, we will use gridsearch to find the best match of the two parameters.

(3) And then, we should will adjust the gamma in xgboost, it is similar to gamma in other DL model, which means the regularization strength. Actually, to the dataset with more features, gamma will be larger, so we should set the range and the step very huge at the beginning and narrow it generally.

(4) The learning rate is the last parameter we should adjust because the best learning rate will be changed by the other parameters dramatic. After these optimize step, we should get relatively best xgboost model to the dataset.

```
iteration 1: train error: 1.1153669357299805    test error: 1.2367873191833496
Running time: 738.6269903182983
iteration 2: train error: 1.1116058826446533    test error: 1.1853299140930176
Running time: 767.5133588314056
iteration 3: train error: 1.1250616312026978    test error: 1.2368340492248535
Running time: 819.8666784763336
iteration 4: train error: 1.0945299863815308    test error: 1.2825068235397339
Running time: 900.0113728046417
iteration 5: train error: 1.1182143688201904    test error: 1.2651740312576294
Running time: 891.2807993888855
Running time: 4122.919212579727
```

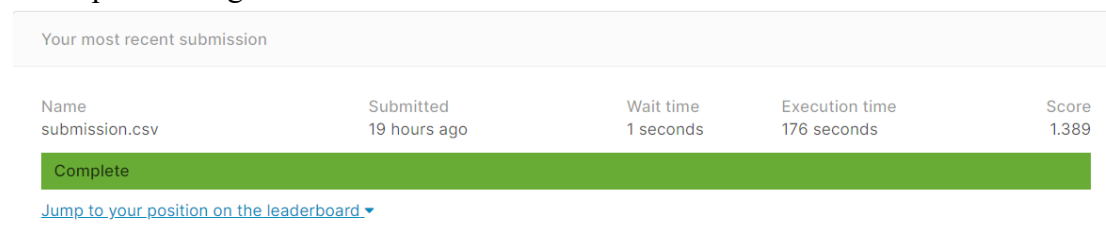
Figure 13. The output and running time of *xgboost* with *n_estimator*=50

However, when we train to optimize the *n_estimator*, we find it is not an easy work here. The training time exponential increasing with the higher *n_estimator* though the error is also getting smaller. It is easy to understand this situation. Every node of CARTs will try to find the point that could split the data into two parts whose square errors of are most close, which means it must compute the square errors of all possible split point in *X*. Therefore, it will take a long time to set a CART for a lot of samples and *n_estimator*'s increasing will repeat this step for more times.

As Figure 13 shows, although the error is much lower than Ridge and could decrease more, we will have to spend much time on it. Finally, we stopped optimize but we still want to write it on the report in terms of its performance.

4. Conclusion

We implement Ridge, Random Forest and xgboost model on the Great Energy Predictor III project. Due to the computation resource limitation, we could not get the best output with xgboost, hence we choose **Random Forest** with much lower error than Ridge as our best model. After submitting the final prediction result and testing on Kaggle, **the score is 1.389 and ranking about 2100 (whole 3614 teams) on the leader board**. The submission status of Kaggle shows below. We guess more pre-processing could produce higher score.



Your most recent submission				
Name	Submitted	Wait time	Execution time	Score
submission.csv	19 hours ago	1 seconds	176 seconds	1.389
Complete				
Jump to your position on the leaderboard ▼				

Figure 14. Kaggle submission status

This dataset is a very large one, with over 20 million samples and 16 features. At the beginning of our project, we even have to spend a long time to read and pre-processing our data until we try to save our pre-processing output and directly read it. Then we tried several different ML model to fit the dataset but most of them have to spend nearly an hour on every CV iteration. Therefore, we realize the importance of the models' training speed and parallel computing.

From the output of our models, we could find that the Random Forest could give a series features weights which seems reasonable basing on our common knowledge. The square of the building should certainly be the most important factor. On the other hand, the ID features' weight is relatively high in the output of both models, shows that the importance of pre-processing. As the future work, we should delete some of them especially which are seem as the foreign key.

And finally, we may spend more time on xgboost on the dataset. Its performance increased obviously though we just optimize one parameter partly. And the speed is also much faster than other model with complex structure. It seems a widely used ML model in various area.

5. Reference

- [1] Kaggle, johnM, skmem Python script
available in: <https://www.kaggle.com/jpmiller/skmem>
- [2] Numpy, numpy.log1p
available in:
<https://numpy.org/doc/stable/reference/generated/numpy.log1p.html?highlight=log1p#numpy.log1p>
- [3] Yadav, S. and Shukla, S., 2016, February. Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. In 2016 IEEE 6th International conference on advanced computing (IACC) (pp. 78-83). IEEE.
available in: <https://ieeexplore.ieee.org/abstract/document/7544814>
- [4] Liaw, A. and Wiener, M., 2002. Classification and regression by randomForest. R news, 2(3), pp.18-22.
available in: <https://cogns.northwestern.edu/cbmgl/LiawAndWiener2002.pdf>
- [5] XGBoost Documentation
available in: <https://xgboost.readthedocs.io/en/latest>
- [6] [1] Chen T , Guestrin C . XGBoost: A Scalable Tree Boosting System[C]// ACM. ACM, 2016.
- [7] A Gentle Introduction to XGBoost for Applied Machine Learning
available in: <https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/>