# Planete Oui: Non-Intrusive Load Monitoring (NILM)

Final Report

Honghao Yu, Qiwen Zhao

(Username on leaderboard: alexiszhao & honghao.yu)

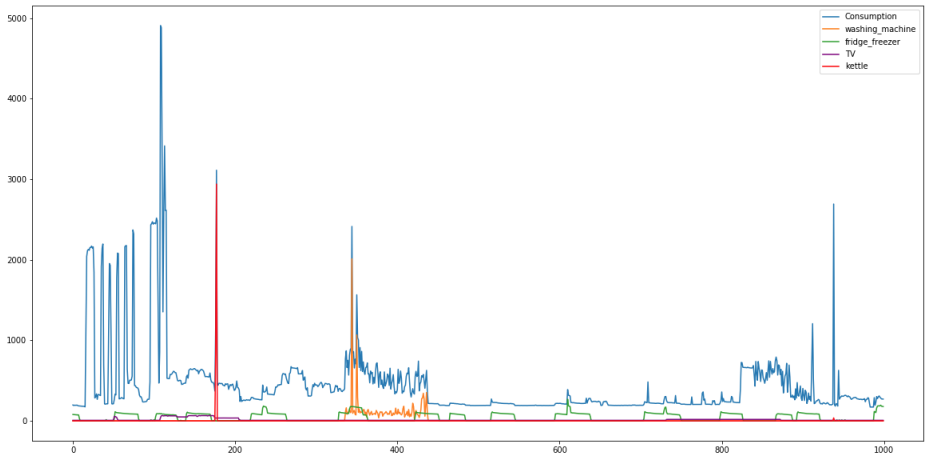
## I. Introduction

Energy consumption and saving have always been a global topic of interest. Within the context of modern energy management, real-time load monitoring is considered as a prerequisite to the following steps of optimization. To achieve this, two approaches have been proposed: Intrusive Load Monitoring and Non-intrusive Load Monitoring (NILM), with the latter getting more and more attention as it's less costly and easier to manage. NILM requires the usage of sophisticated methods to decompose the total energy consumption into respective consumption of individual appliances. It is desirable to increase the accuracy of decomposition using state-of-art knowledge from domains like IoT and Machine Learning. In this report, we mainly focus on the Non-intrusive Load Monitoring challenge provided by BCM Energy. We would like to propose a Machine Learning method to extract the individual electricity consumption of four appliances (washing machine, fridge freezer, TV, kettle) from the total consumption of a given household at any moment as accurately (measured by the given metric) as possible. Specifically, we utilized XGBoost and LightGBM with 106 features in total and achieved a weighted square error of 36.55.

## II. Initial Exploration and Visualization

The given X_train data contains the aggregate electricity consumption recorded at the end of each minute and 7 meteorological metrics (visibility, temperature, humidity, humidex, wind chill, wind, pressure) recorded at the beginning of each hour, covering the period from March 17, 2013, to Dec 31, 2013, totalling 417,599 records, while the y_train data consists of the corresponding consumption data of four appliances (washing machine, fridge freezer, TV, kettle) recorded at the same time as the aggregate consumption. Due to technical flaws or other reasons, there are 10,231 (2.44%) missing values in the consumption data, and it should be noted that the missing in aggregate data and

**Graph 1**: Consumption Patterns over time



individual data appear simultaneously (in the same minutes). The X_test data covered the period from January 1, 2014 to June 7, 2014 totalling 226,081 records with 24,719 (10.9%) missing entries in the 'consumption' column. Since the meteorological data were only recorded once per hour, we have around 410,000 (98.2%) missing values for each of the seven variables. As each record is associated with a timestamp, we thought it would be useful to visualize the electricity consumption pattern over time. See **Graph 1**.

Note that we only visualized a small segment (1000 mins / 16.6h) of the full data because otherwise it would be too chaotic. We could easily identify different patterns for different appliances and associate them with the pattern of aggregate consumption. In general, all except for fridge freezer were turned off most of the time. It took around 100 mins per time for the washing machine to perform its task, and it always started with a high peak at around 2000W for 3~5 mins and then dropped to around 150W for the rest of the operations. We suspect that the machine was equipped with a heating function and the initial high consumption was caused by the heating. The kettle had the most drastic changes in its consumption, working at close to 3000W for fewer than 3 mins and then soon dying down. The TV operated at comparatively low power below 100W for a few hours (not fixed but usually 2~3h). The fridge freezer exhibited a more regular pattern with constant shifts. It was usually on for 40~50 mins and then off for a slightly longer period.

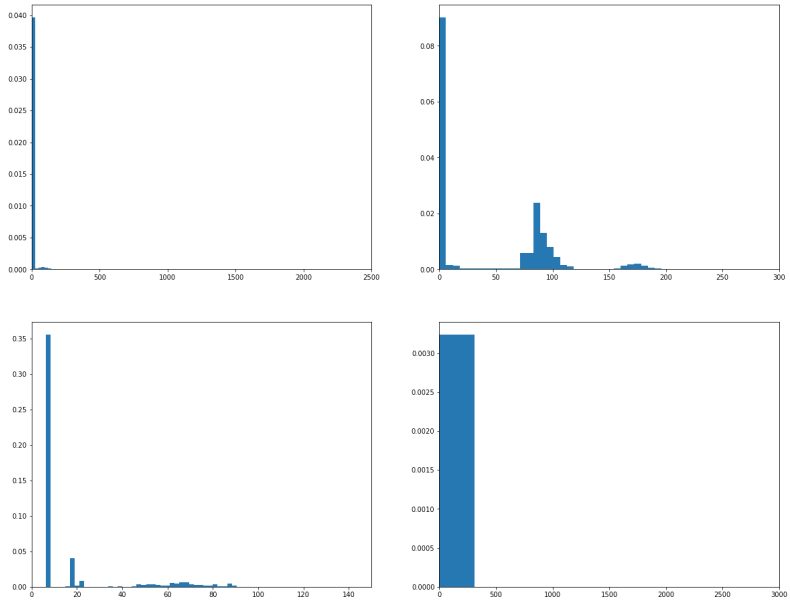
When observed on a larger scale, the other three appliances also showed some periodical patterns. These concerned the usual time and frequency for which they were turned on. For example, both washing machine and TV were regularly turned on in certain hours of a day or on certain days of a week. The observed patterns indicated that time features (functions of day, hour, minute) could possess strong predictive power. We could even first predict the possibility of being switched on or off based on time and other derived features and then plug in the calculated consumption pattern in order to have an accurate estimate of the decomposition (we didn't, however, implement this approach in this challenge as it doesn't closely align with our workflow).

A problematic issue with the decomposition was that the aggregate consumption data was quite noisy. We can see that, apart from the four appliances which contributed to the fluctuations of the aggregate consumption, there were many unpredictable factors (e.g. other appliances unaccounted for) that complicated the task. A quick look at the initial records would tell us that the aggregate consumption exhibited a pattern similar to that when there was a kettle working while in fact, only the fridge freezer was functioning. The noises require us to develop a robust method to identify the key indicators of the states of these appliances.

To quantify the pattern of each appliance, we also drew the histograms of electricity consumption as below (**Graph 2**):

**Graph 2**: Histograms of Consumption by Appliance

(Upper left: washing machine; upper right: fridge freezer; lower left: TV; lower right: kettle)



Obviously, the records were highly concentrated around 0 for all the appliances as they were switched off most of the time. For washing machine and kettle, values greater than 0 accounted for only a negligible proportion of all the records but the range was much larger with the highest exceeding 2500W. There were three clusters of values (centred at 0, 80, 170 respectively) for the fridge freezer. The variation for TV was large with many non-zero values scattered within the range 0 to 140, although there was a small cluster centred at 20W and most of the non-zero values were within 40 to 100.

Given that the individual patterns were so different from each other, it would be reasonable to consider different feature sets and hyperparameters or even different models for each appliance. This was reflected in our approaches to this challenge.

## III. Data Preprocessing

Our data preprocessing mainly focuses on the imputation of missing values. For the 7 weather metrics, we utilized forward linear imputation (pandas.interpolate) to fill the void between two records because it's reasonable to suppose that temperature, humidity, pressure, etc. change continuously at a linear rate in a short period of time. In this way, each record serves as a smooth transition to the following estimation. For the non-imputed initial records, we used backward linear imputation.

Regarding the missing values in the consumption data (both aggregate and individuals), we consulted several papers and decided on using forward nearest imputation with a stride (number of records to be imputed) of 11 mins (95% percentile) based on the distribution of gap durations (see **Table 1**).

**Table 1**: Percentiles of Gap Durations

| **Percentile** | 50% | 75% | 80% | 85% | 90% | 95% | 99% |
|---|---|---|---|---|---|---|---|
| **Gap** (mins) | 1 | 2 | 4.8 | 10 | **11** | **11** | 30.79 |

Due to technical flaws, the appliances may have functioned normally but were not recorded. Different authors have suggested different stride sizes, and here we chose 11 mins. If the stride was too large, then circumstances like power cut would be falsely imputed. We have used the nearest imputation because of the activity continuity of the appliances. If there is a 11-min void in the records of the washing machine, it's reasonable to assume that it is still working during the void if right before it's working, though this may not hold true for kettle as it was only turned on for fewer than 5 mins each time. We therefore decided not to fill in the missing values for kettle using this method. Instead, we just replaced them with zero. For the rest of the missing entries, we replaced them all with zeros.

Abnormal values (i.e. outliers) could greatly impact the estimations in regression problems, but luckily we didn't observe any of them in our data.

Machine learning techniques usually require scaling the data before feeding them to the algorithms, However, with some quick experimentation, we found that centring and scaling would significantly impair the model performance (after standardization, the results were both worse and less robust to the learning parameters), hence we chose to keep the original data.

## IV. Feature Engineering

In this challenge, we were given only 9 features (timestamp, aggregate consumption and 7 meteorological metrics) and were unable to find relevant external data, so we had to extract as much useful information as possible from the existing ones.

### 4.1 Time features

Our first step was to deal with the timestamps. We extracted the date, month, day of week (Monday to Sunday), hour (0~23) and minute for each record, and did one-hot encoding for hour and day of week. We also created a dummy variable to indicate whether it's weekday or weekend. Considering the time patterns, we added the triangular transformations of hour (with a period of 24) and minute (with a period of 60) as well as the absolute time within a day (with a period of 720, half-day).

### 4.2 Weather features

Logically, meteorological metrics would change simultaneously to reflect the change of weather, so it's highly possible that they are linearly correlated to each other. Thus we performed a Principal Component Analysis (PCA) to obtain the necessary information while getting rid of noises. The first

three principal components accounted for 95.2% (75.8%, 13.4%, 6.0% respectively) of the total variation and are thus included in our feature set.

### 4.3 Consumption derived features

We spent the most effort in experimenting with aggregate consumption. Several transformations have been done. Firstly, we included the consumption of the last/next one, two, three, four, and five minutes to capture the continuity. Moving averages of the last/next three and five minutes were calculated as we mainly focused on boosting methods instead of linear regressions (therefore multicollinearity would not be a problem). Besides, the maximums, minimums, standard deviations and ranges of the last/next five minutes were also taken into consideration.

For the same purpose of accounting for continuity, we also computed hourly and half-hourly aggregations (maximums, minimums, means, standard deviations and ranges). Starting from the first entry, we sliced the data by every 60 minutes and every 30 minutes and performed the in-group aggregate computations. The five common descriptive statistics were used to describe the essential characteristics of each segment.

### 4.4 Windows with margins method

The next idea was based on two academic papers (Salerno & Rabbeni, 2018; Zhuang et al., 2018). According to the authors, an essential part of NILM is accurate event detecting, and a practical approach is to use Moving Windows with Margins (MWM). The basic idea of MWM is to set a fixed-size window with fixed-size margins at the two sides and to scan the full data sequentially. During the scanning, the average consumptions within the two margins were calculated and then compared. If the difference exceeds certain thresholds, a change of state or an activity is said to have been detected. In the original work, a secondary MWM was introduced to reduce false labelling, but here we only adopted the primary window. After taking into consideration the intrinsic properties of consumptions of each appliance, we decide to perform MWM on the kettle, washing machine and freezer and produced a series of the change of states for each. We observed from **Graph 1** that washing machine and kettle can significantly shift the aggregate consumption within a short period while the freezer was turned on periodically with an almost constant low consumption, hence could be derived more easily and accurately. Accordingly, we set different windows with different margins for the 3 appliances as below (**Table 2**):

**Table 2**: Original MWMs

| Appliance | Window Size | Margin Size | Lower Threshold |
|---|---|---|---|
| Washing Machine | 100 | 5 | 1800 |
| Washing Machine | 100 | 5 | 2000 |
| Washing Machine | 100 | 10 | 2000 |

| | | | |
|---|---|---|---|
| Kettle | 3 | 1 | 2500 |
| Kettle | 3 | 1 | 2700 |
| Kettle | 5 | 1 | 2700 |

Initially, we only set a lower threshold for each window. However, after observing that at many times the aggregate consumption exceeded the lower bounds while the corresponding appliances were not working, we decided to add an upper threshold and created three new windows (see **Table 3**).

**Table 3**: MWMs with Upper Thresholds

| Appliance | Window Size | Margin Size | Lower Threshold | Upper Threshold |
|---|---|---|---|---|
| Washing Machine | 100 | 5 | 1800 | 2300 |
| Kettle | 3 | 1 | 2600 | 3000 |
| Freezer | 25 | 3 | 50 | 200 |

Admittedly, we didn't replicate the methods discussed in the papers but merely borrowed their basic ideas. In fact, what we had produced were just rough indicators of state changes specific to each appliance. They served as the stepping stones for obtaining more accurate windows.

#### 4.5 On-Off state prediction

To better utilize the information obtained from our MWMs, we did an On-Off state prediction for each record which indicates whether the appliance was turned on or off at each minute. This was accomplished by comparing the adjacent entries in the state series generated by MWMs. When there was sufficient evidence that a state pattern similar to that of an appliance was detected, we replaced the whole following period of the window size with a single on/off label. This forced replacement ensured that the continuity of activities was taken into consideration, otherwise, the states would be relatively independent of each other. Hence, the window size was crucial as it determines whether the records were split correctly into two parts: on and off.

#### 4.6 Conclusion

After the feature engineering, we obtained a pool of **106** features in total. Please refer to the **Appendix** for the details.

## V. Model Selection

We iterated back and forth between choosing the optimal feature set and trying different models with different hyperparameters. We mainly experimented with three models: **ELM**, **XGBoost**, and **LightGBM**. As mentioned before, since each appliance exhibited a distinguishable pattern of

consumption, we decided to independently train a different model (different feature set and hyperparameters) for different appliances (we have also experimented with a unified model which will be discussed later).

### 5.1 ELM

ELM stands for Extreme Learning Machine, a machine learning method proposed by Guangbin Huang in the 2000s. It utilizes a feedforward neural network architecture to perform classification, regression, clustering, etc. with a single layer or multiple layers (partially or fully connected) of hidden nodes whose parameters can be randomly assigned and need not be tuned ("Extreme learning machine", n.d.). The final output layer can be modified to fit different purposes (e.g. Generalized Linear Regressor for regression problems). In a basic ELM model, only three steps need to be done to obtain the output weights: 1) assign randomly hidden node parameters, 2) calculate the hidden layer output matrix, and 3) calculate the output weights. Since it avoids iterative tuning, its learning speed can be extremely fast (Huang, n.d.).

We first considered ELM since it was used in the paper (Salerno & Rabbeni, 2018) and achieved impressive results. In this paper, both the basic ELM and a hierarchical ELM (H-ELM) were implemented and compared. Essentially, training a basic ELM only requires to set the random state, the number of hidden nodes, and the activation function. We performed several experiments on the four appliances and obtained the best result for each as follows (**Table 4**):

**Table 4**: Experimentation Results with ELM

| Appliance | Feature Set | Hyperparameters | RMSE |
|---|---|---|---|
| Washing Machine | X_vars_base + X_vars_day + X_vars_hour1 + X_vars_time + X_vars_hourly | random_state=24, n_hidden=256, activation_func='multiquadric', alpha=0.7 | 45.95 |
| Fridge Freezer | Same as above | Same as above | 44.51 |
| TV | Same as above | Same as above | 15.33 |
| Kettle | Same as above | Same as above | 67.52 |

This brought the weighted validation RMSE down to 40.46. However, when we submitted the predicted results onto the leaderboard, we only got a score slightly better than the benchmark. The paper recommended setting n_hidden at 4,096 which took approximately three hours on their machine, but we've already run out of memory and crashed our runtime when we raised it to 2,056. On the other hand, setting a large number of hidden nodes didn't help in our case as we observed an increase instead of a decrease in the RMSE when the number exceeded 512, probably due to overfitting. In the meanwhile, the results were highly sensitive to the random state (which determined

the randomly assigned weights), making the ELM model even less desirable. Due to these reasons, we decided to continue with other methods.

### 5.2 XGBoost

XGBoost is an innovative variant of the boosting methods, part of the so-called ensemble methods, where multiple weak learners integrate into a much stronger learner. In Gradient Boosting, the aim of each base learner (a decision tree) is to fit the residual of the preceding base learner and move one more step further towards the negative gradient of the loss function. The construction of trees is in level-wise sequential order. XGBoost model is an extension of Gradient Boosting in the sense that it utilizes information from both the first-order derivative (gradient) and the second-order derivative (Hessian) of the newly-added tree. Generally speaking, XGBoost is faster compared to other implementations of gradient boosting (excluding LightGBM which will be discussed later) and has dominantly better performances on classification and regression predictive modelling problems.

Unlike ELM, XGBoost models are controlled by a large number of hyperparameters which makes it more flexible but also much harder to train. There are mainly 4 categories of impactful hyperparameters: 1) model complexity: max_depth, min_child_weight, n_estimators, max_leaves, max_bin, etc.; 2) subsampling: subsample, colsample_bytree, colsample_bylevel, colsample_bynode, etc.; 3) regularization: eta, gamma, lambda, alpha, etc.; 4) training: learning_rate, etc. We would only fine-tune a subset of these in the following phase while keeping the default values for all other parameters.

We also tried some vanilla models with manually chosen hyperparameters. RFECV was used to obtain the respective optimal feature set given these parameters. The best results for each appliance we obtained (using an early-stopping round of 50) are as follows (**Table 5**):

**Table 5**: Experimentation Results with XGBoost

| Appliance | Hyperparameters | Best Rounds | Train RMSE | Test RMSE |
|---|---|---|---|---|
| Washing Machine | max_depth=6, n_estimators=5000, min_child_weight=5 | 5000 | 2.32 | 24.35 |
| Fridge Freezer | max_depth=5, n_estimators=6000, min_child_weight=5 | 6000 | 15.00 | 19.50 |
| TV | max_depth=5, n_estimators=4000, min_child_weight=5 | 4000 | 2.32 | 3.34 |
| Kettle | max_depth=6, n_estimators=6000, min_child_weight=5 | 898 | 8.42 | 55.24 |

As shown in the table, XGBoost not only allowed much greater room for tuning but also performed significantly better than ELM. Therefore, we opted for XGBoost as our next model.

### 5.3 LightGBM

The complex sequential construction of trees in XGBoost and its great many hyperparameters to be tuned made the training and refining process extremely time-consuming. When it came to feature selection with cross-validation, the process for every single appliance can take up to 7~8 hours, thus it's simply impossible to cope with the frequent changes in our approach. Under such constraint, we also experimented with another variant of the Gradient Boosting - LightGBM - and achieved similar RMSEs with significantly less time.

LightGBM was designed to be distributed and efficient with advantages including: 1) faster training speed and higher efficiency; 2) lower memory usage; 3) better accuracy; 4) support of parallel and GPU learning. It splits the tree leaf-wise instead of depth-wise or level-wise (XGBoost) as in other boosting methods. In this fashion, LightGBM reduces more loss when growing on the same leaf and is very fast, which was proved by our own experiments.

Similar to those of XGBoost, LightGBM also has a large set of tunable hyperparameters: 1) model complexity: num_iterations, num_leaves, max_depth, min_data_in_leaf, min_sum_hessian_in_leaf, max_bin, etc.; 2) subsampling: bagging_fraction, bagging_freq, feature_fraction, etc.; 3) regularization: lambda_l1, lambda_l2, etc.; 4) learning: learning_rate, etc. We would also tune only a subset of these parameters during the training phase.

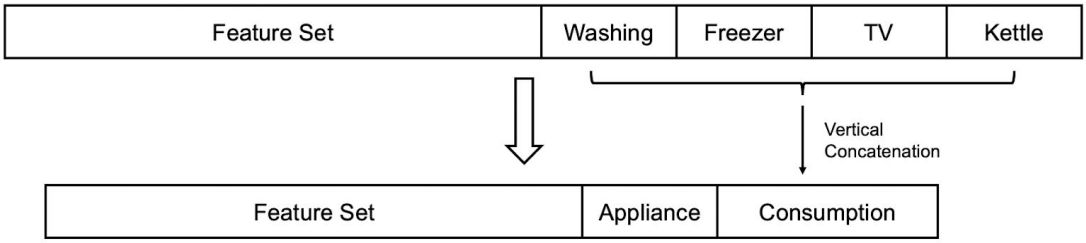
Since XGBoost and LightGBM share many common hyperparameters and have similar APIs, we kind of used them parallelly in the later phase of our studies. We first used LightGBM to filter out the optimal feature sets and fed them to both XGBoost and LightGBM with a few manually assigned parameters respectively. When the results produced by XGBoost were only slightly better or even worse, we would opt for LightGBM and fine-tune its hyperparameters; otherwise, we would fine-tune the XGBoost model.

## VI. Feature Selection & Hyperparameter Tuning

We would like to first mention the unified model though it was actually carried out in the halfway. The idea was inspired by the fact that in real-world applications, it can be rather costly to train different models for different appliances as there are apparently far more than 4 and there are always new ones coming in. An individual approach also requires sufficient training data for each single appliance which is simply too costly and may cause a significant lag in the product delivery (several months alone to collect the data). Considering that future implementation may further require the algorithms to adapt to newly-coming streaming data (continuously updating the weights based on newly-obtained real-time data), the individual approach can become exponentially more complex and unrealistic in practical uses.

Within our unified framework, all these four appliances shared the same feature set and hyperparameters. We transformed our dataset in a way as below (**Graph 3**):

**Graph 3**: Dataset Transformation within the Unified Framework




After such transformation, our dataset had four times as many rows as before and an extra feature that labelled each appliance. The four columns of individual consumptions were concatenated into a single one. We then utilized RFECV to select the optimal feature set (71 features) with the default hyperparameter values. Since the feature selection had already taken a really long time to finish, we didn't perform parameter tuning with either grid search or random search. The best results we obtained with XGBoost are shown in **Table 6**:

**Table 6**: Experimentation Results with Unified XGBoost

| **Hyperparameters** | **Appliance** | **5-fold Test RMSE** |
|---|---|---|
| max_depth=6,<br>n_estimators=2000,<br>min_child_weight=5 | Washing machine | 32.10 |
| | Fridge freezer | 33.47 |
| | TV | 14.16 |
| | Kettle | 52.20 |

Though it had the benefits of unified feature set and hyperparameters, the unified model gave far less accurate predictions than XGBoost, whilst took longer time to train. Therefore, we had to abandon this framework at the present.

We then did several rounds of iterations between feature selection and parameter fine-tuning. Prior to the formal phase, we performed fitting on the full-scale training data with the full feature set using XGBoost with the default hyperparameters except for max_depth, n_estimators and min_child_weight. We then reported the feature importances with a self-defined function and filtered out those with negligible significance. In this way, we knew better which features might have the strongest predictive power and could derive more candidate features accordingly. In each iteration, we firstly started with similar operations, using RFECV (step=1, cv=4) to select the metric-based optimal feature set, and then manually deleted those without which the RMSE wouldn't change much (at the magnitude of 0.1), all with the default hyperparameter values except for max_depth,

n_estimators and min_child_weight. During the selection, we set the n_estimators at 100 to speed up the process. Secondly, we used a random search with 5-fold cross-validation to choose the optimal combination of hyperparameters. Since we already had some priori knowledge about how the results would be impacted by these parameters, we set a relatively narrow range for each to avoid nonsense experimentation. The focus was largely on those controlling model complexity such as max_depth, min_child_weight, min_samples_leaf, regularization terms such as reg_alpha, reg_lambda and learning_rate. We left out n_estimators to be determined later on. In the final step, we used the xgb.cv module to determine the best boosting rounds with 5-fold cross-validation and an early-stopping round of 50.

Pretty much the same things were done for LightGBM except that we changed the parameter set to be tuned. Due the the fast speed of LightGBM, we were able to perform far more rounds and fine-tune a much larger set of parameters.

One major issue we encountered during the whole process was the large discrepancy between the local cross-validation RMSE and the leaderboard RMSE. We could achieve very low RMSE in our local training and validation but got an unexpectedly high value when we submitted the predictions onto the leaderboard. At first we thought it might be due to over-fitting, but we didn't observe the turning point where the cross-validation RMSE began to go up while the training RMSE kept going down as supposed to be in over-fitting. Instead, for most of the time, these two values were pretty close to each other. After reducing model complexity and reinforcing regularization, the situation remained unimproved. We therefore proposed that it was not over-fitting, but rather due to the different distributions of consumptions in the training data and the test data. Given that the two datasets covered two non-overlapping time periods, it's highly possible that the user consumption pattern had changed significantly to the extent that some features might lose their predictive powers. This requires us to find out the features that have notably different importance when used in fitting the two different datasets. Unfortunately we didn't have enough time to carry out this analysis. To obtain an accurate estimation of the real test RMSE, we split the X_train data into training set and validation set without shuffling so that the two sets also covered two non-overlapping time periods. We found that the resulted validation RMSEs were indeed much closer to the leaderboard RMSE, hence we were convinced that there was an intrinsic change in the consumption across different time periods and a strategy was needed to address this heterogeneity (e.g. using features that are more robust over time). We admitted that there was a major pitfall in our feature engineering that caused the lack of generalization ability.

The final results we obtained using XGBoost and LightGBM individually for each appliance with selected feature sets and fine-tuned parameters are as follows (**Table 7**):

**Table 7**: Final Results with XGBoost and LightGBM

| Model | Appliance | # feature | Hyperparameters | Validation RMSE |
|---|---|---|---|---|
| XGB | Washing Machine | 36 | 'max_depth':6,'n_estimators':1000,'n_jobs':4, 'min_child_weight':8, ‘min_samples_leaf’:4, ‘reg_lambda’:0.05, ‘reg_alpha’:0.01, ‘learning_rate’:0.1 | 54.85 |
| | Fridge Freezer | 42 | 'max_depth':6, 'n_estimators':1000, 'n_jobs':4, 'min_child_weight':8, ‘min_samples_leaf’:4, ‘reg_lambda’:0.05, ‘reg_alpha’:0.01, ‘learning_rate’:0.1 | 42.52 |
| | TV | 38 | max_depth':6, 'n_estimators':1000, 'n_jobs':4, 'min_child_weight':8, ‘min_samples_leaf’:4, ‘reg_lambda’:0.05, ‘reg_alpha’:0.01, ‘learning_rate’:0.1 | 17.04 |
| | Kettle | 37 | 'max_depth':6, 'n_estimators':1000, 'n_jobs':4, 'min_child_weight':8, ‘min_samples_leaf’:4, ‘reg_lambda’:0.05, ‘reg_alpha’:0.01, ‘learning_rate’:0.1 | 88.00 |
| LGB | Washing Machine | 22 | num_leaves=200, min_sum_hessian_in_leaf = 15, min_data_in_leaf = 25, max_depth = 30, max_bin = 250, learning_rate=0.01, lambda_l1 = 0.1, lambda_l2 = 0.1, feature_fraction = 0.8, bagging_freq = 5, bagging_fraction = 0.8, n_estimators=15000 | 47.03 |
| | Fridge Freezer | 25 | num_leaves=250, min_sum_hessian_in_leaf = 10, min_data_in_leaf = 25, max_depth = 30, max_bin = 200, learning_rate=0.01, lambda_l1 = 0.5, lambda_l2 = 0.1, feature_fraction = 0.8, bagging_freq = 5, bagging_fraction = 0.8, n_estimators=15000 | 39.69 |
| | TV | 35 | num_leaves=200, min_sum_hessian_in_leaf = 10, min_data_in_leaf = 25, max_depth = 25, max_bin = 250, learning_rate=0.005, lambda_l1 = 0.1, lambda_l2 = 0.1, feature_fraction = 0.8, bagging_freq = 5, bagging_fraction = 0.8, n_estimators=15000 | 15.20 |
| | Kettle | 37 | num_leaves=250, min_sum_hessian_in_leaf = 15, min_data_in_leaf = 25, max_depth = 30, max_bin = 255, learning_rate=0.01, lambda_l1 = 0.1, lambda_l2 = 0.1, feature_fraction = 0.8, bagging_freq = 5, bagging_fraction = 0.8, n_estimators=15000 | 89.24 |

Conclusion:

Based on our comparison of the two model, we obtained that Light GBM is faster in training and yields a more accurate result than XGBoost.

## VII. Results

We shall briefly discuss the feature importances and prediction accuracy in this section. We are most interested in which features can best help us disaggregate the total electricity consumption into component consumptions of individual appliances and which appliances are comparatively more difficult to deal with.

For washing machine, the most powerful features are the aggregate consumption, consumption_last4min, hourly_avg, consumption_last5min, next_5_min_avg (ranked by feature importance). It seemed no features other than the nearest consumptions and hourly averages could be good indicators of the states of washing machine, making it highly reliant on how we extracted information from time and aggregate consumption.

For fridge freezer, the most powerful features are the aggregate consumption, visibility, temperature and pressure which makes sense because the refrigerator may automatically adjust its state according to the real-time meteorological conditions.

For TV, the most powerful features are visibility, hourly_min, temperature, and pressure. This was probably due to the fact that these feature were highly correlated with the time within a day and people tend to watch TV at night when these features exhibited different characteristics as compared to the daytime.

For kettle, the most powerful features are the aggregate consumption, consumption_next1min, and consumption_last1min. Similar to washing machine, disaggregating the consumption for kettle is largely dependent on deriving useful features from aggregate consumption.

It's easy to figure out that the most unpredictable appliances were those with high variations in consumption (kettle, washing machine). For one, they had extra high peaks which only lasted for several minutes and were switched off most of the time; for another, their functioning time had been quite random without regular patterns. For these reasons, the predictions were greatly disturbed by background noises, causing high RMSEs. On the contrary, the RMSEs for fridge freezer and kettle were notably lower as they exhibited quite regular patterns in terms of both consumption variation and functioning periods.

## VIII. Discussion

Due to the time limit and the COVID-19 outbreak, we have left many flaws hence much room to improve in our studies. We've consulted several academic papers for useful features and models but didn't have the time to try them all. There were quite a few papers talking about Non-intrusive Load Monitoring with machine learning methods and provided many enlightening insights, but we merely borrowed the idea of Moving Windows with Margin (MWM). Even for the MWM, we were not really sure that we had the right implementation. Future work needs to focus on using a scientific approach

to determine the window size and margin size instead of basing them on mere observations as we had done here. We would need to dive deeper into feature engineering to extract more useful information from the aggregate consumption data.

As mentioned earlier, we would also need to deal with the heterogeneity across different time periods. We didn't notice vital flaws in our feature selection and hyperparameter tuning techniques, yet the problem in finding robust features was unsolved. Unlike in other machine learning problems where the observations are relatively independent from each other, here the observations are highly dependent on the preceding ones, which made the training more complex. If not split in a proper way, the training and validation can yield misleading results. Future work should deal with keeping the balance between randomness in cross-validation and result reliability.

## Appendix

| Feature | Description |
|---|---|
| month | -- |
| hour | -- |
| hour0~23 | One-hot encoding of hour |
| minute | -- |
| dayofweek | 0~6; 0 for Monday and 6 for Sunday |
| day0~6 | One-hot encoding of day |
| isweekend | Dummy for whether it's on weekend |
| hour_sin/cos | Triangular transformations of hour |
| min_sin/cost | Triangular transformations of minute |
| time | Combination of hour and minute in the unit of minute |
| time_sin/cos | Triangular transformations of time |
| hourly_avg/max/min/std/diff | Hourly aggregate statistics |
| half_hourly_avg/max/min/std/diff | Half-hourly aggregate statistics |
| WeatherPCA1/2/3 | Principal components of the weather metrics after performing PCA |
| is_kettle/washing/_XXX | The results obtained with MWMs, XXX stands for thresholds |
| kettle/washing/freezer_current | The on/off predictions based on MWM results |
| consumption_last/next 1/2/3/4/5min | The consumption of the last/next 1/2/3/4/5 minute. |
| last/next_3/5min_avg | The average of the consumptions of the last/next 3 and 5 minutes |
| consumption_last/next 5min_min/max/std/diff | 5-minute aggregate statistics |

## References

*Extreme learning machine*. En.wikipedia.org. Retrieved 6 April 2020, from https://en.wikipedia.org/wiki/Extreme_learning_machine.

Huang, G. *Extreme Learning Machines: Random Neurons, Random Features, Kernels*. Ntu.edu.sg. Retrieved 6 April 2020, from https://www.ntu.edu.sg/home/egbhuang/.

Mocanu, D., Mocanu, E., Nguyen, P., Gibescu, M., & Liotta, A. (2016). Big IoT data mining for real-time energy disaggregation in buildings. In *2016 IEEE International Conference on Systems, Man, and Cybernetics*. Budapest, Hungary; IEEE. Retrieved 6 April 2020, from https://ieeexplore.ieee.org/document/7844820.

Pattem, S. (2012). Unsupervised Disaggregation for Non-intrusive Load Monitoring. In *2012 11th International Conference on Machine Learning and Applications* (pp. 515-520). Santa Clara, CA, USA; IEEE. Retrieved 6 April 2020, from https://ieeexplore.ieee.org/document/6406788.

Sadeghianpourhamami, N., Ruyssinck, J., Deschrijver, D., Dhaene, T., & Develder, C. (2017). Comprehensive feature selection for appliance classification in NILM. *Energy And Buildings*, *151*, 98-106. https://doi.org/10.1016/j.enbuild.2017.06.042

Salerno, V., & Rabbeni, G. (2018). An Extreme Learning Machine Approach to Effective Energy Disaggregation. *Electronics*, *7*(10), 235. https://doi.org/10.3390/electronics7100235

Welikala, S., Thelasingha, N., Akram, M., Ekanayake, P., Godaliyadda, R., & Ekanayake, J. (2019). Implementation of a robust real-time non-intrusive load monitoring solution. *Applied Energy*, *238*, 1519-1529. https://doi.org/10.1016/j.apenergy.2019.01.167

Zhuang, M., Shahidehpour, M., & Li, Z. (2018). An Overview of Non-Intrusive Load Monitoring: Approaches, Business Applications, and Challenges. In *IEEE* (pp. 4291-4299). Guangzhou, China; POWERCON. Retrieved 6 April 2020, from https://ieeexplore.ieee.org/abstract/document/8601534/versions.