Semester Project Autumn 2021

Meta-Learning in Smart Grids

January 6, 2022

Name: Kravtsov Denis, Sciper: 282379

Supervisor: Mahrokh Ghoddousiboroujeni

Prof. Ferrari Trecate Giancarlo, Automatic Control Laboratory

Contents

1 Description

Smart grids bring about automation to electricity networks by establishing a two-way interactive system between consumers and energy providers through fetching users' power consumption in discrete time intervals. These fine-grained measurements can be used for inferring models of households' energy consumption, and for analyzing the effect of various geographical or societal factors on consumption patterns. While there has been a vast amount of work on household electricity prediction, the proposed approaches suffer from requiring long data collection periods before the models can be used and are sensitive to the change of households' habits or technological renovations.

To alleviate the requisition of big training datasets and improve the flexibility of the models to adapt to the changes in consumption patterns, one can decompose the learning algorithm into two layers: a) learning a generic function describing shared consumption patterns among households and b) training personalized models on top to account for personal differences. This scenario is well-described in the context of meta-learning.

This project concentrates on the implementation and assessment of such meta-learning algorithms for short-term energy consumption values predictions.

2 Objectives

This semester project involves a creation of a standard Data Science pipeline. Its objectives are as follows:

1. Familiarizing with energy datasets
2. Choice of a dataset for analysis
3. Visualization and exploratory data analysis
4. Identifying and implementing relevant machine learning algorithms
5. Identifying and implementing relevant meta-learning algorithms
6. Simulation and experimental validations

3 Dataset

For this project, one of the main objectives is the study of efficiency of meta-learning algorithms for short-term horizon energy consumption predictions. Thus, the dataset had to have intervals short enough to satisfy this objective. On top of that, in order to have meaningful predictions, some additional data such as weather reports was needed. A dataset satisfying these conditions was found on Kaggle: **Smart Meters in London dataset**[1]

This dataset has the following data:

1. Half-hourly household energy consumption
2. Hourly weather reports
3. ACORN household details
4. Bank Holidays calendar

This data allowed to get a granularity of one hour. On top of that it had useful information such as weather reports and bank holidays calendar, which could explain potential fluctuations in energy consumption levels. Furthermore, ACORN details (segmentation tool which categorises the UK's population into demographic types) could be potentially useful for a better performance of meta-learning algorithms, as they could explain differences in consumption patterns for different households. However, these details were not put in practice in this project, due to overall complexity of this data.

In the scope of this project, energy consumption during working days is studied, as it represents the major part of the dataset, and as energy consumption pattern during weekend is different from the one during working days. In particular, the day of Wednesday was chosen, as it would be most representative of working days consumption pattern, as it is situated in the middle of working week. Here are some plots, showing distribution of the data in the dataset:

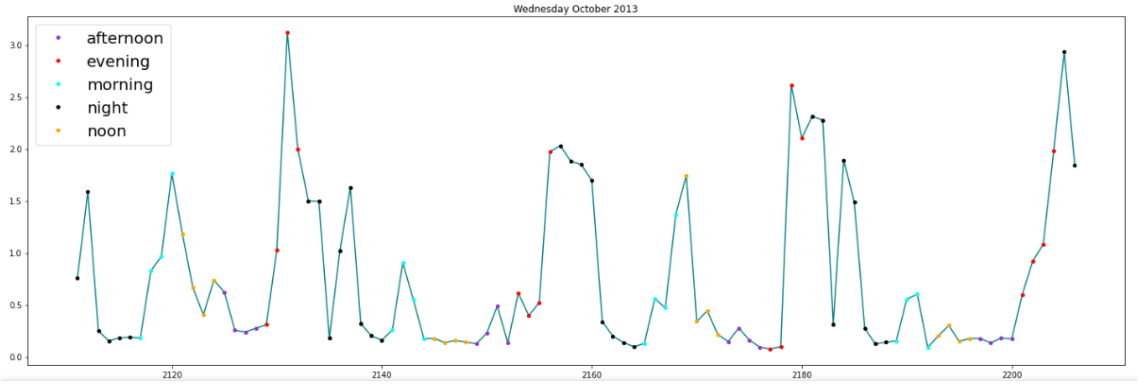

Figure 1: Energy consumption distribution during the day (weekday)

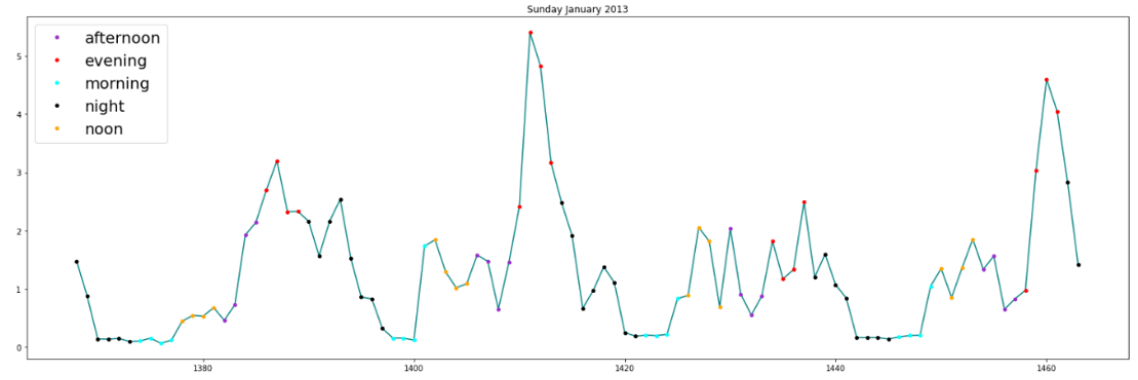

Figure 2: Energy consumption distribution during the day (weekend)

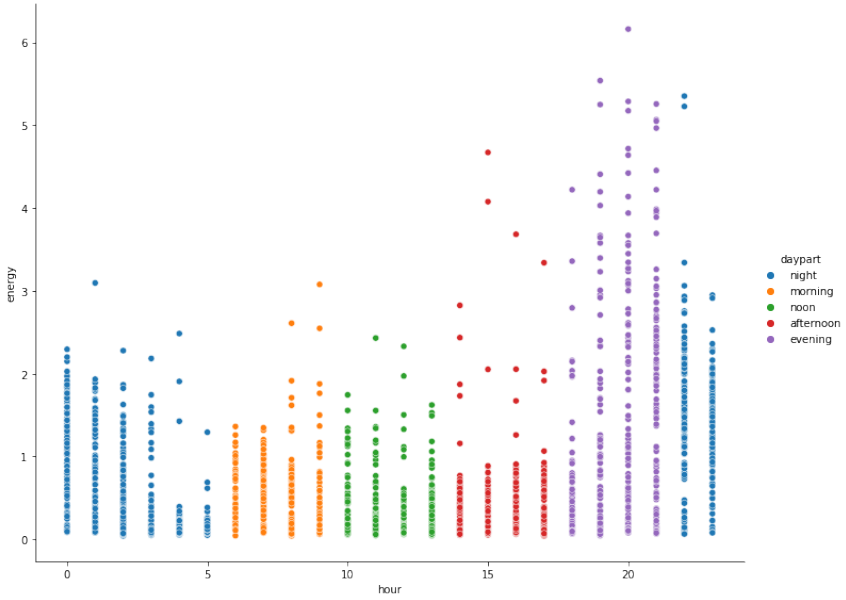

Figure 3: Hourly energy consumption distribution

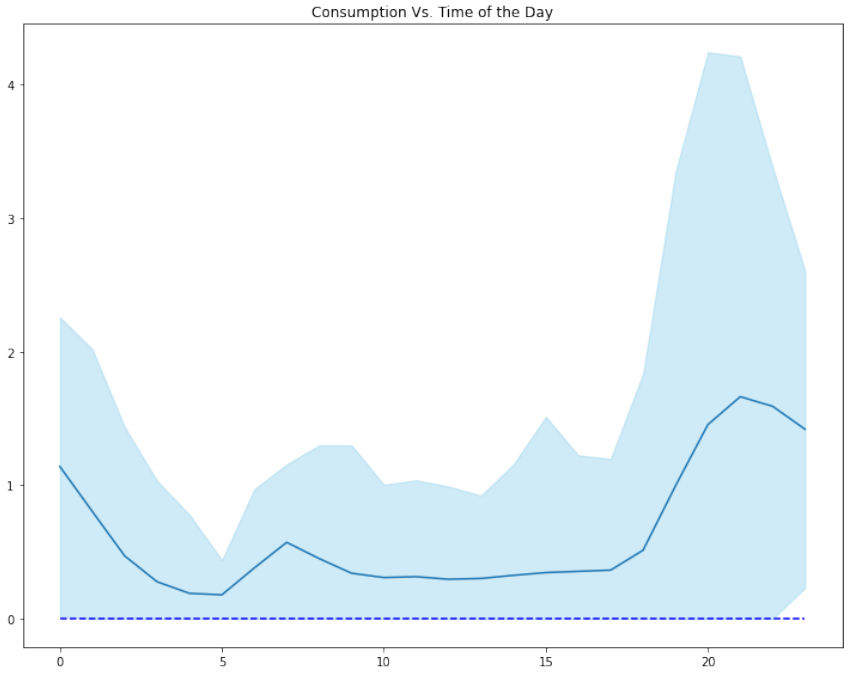

Figure 4: Daytime energy consumption distribution

One can see that main energy consumption is happening during the evenings and in the beginning of the night, when people are coming home from work, during weekdays. One can also observe, that there are peaks of consumption in mornings, before people are leaving for work. During the weekend the situation is different and the energy consumption is more evenly spread out during the day.

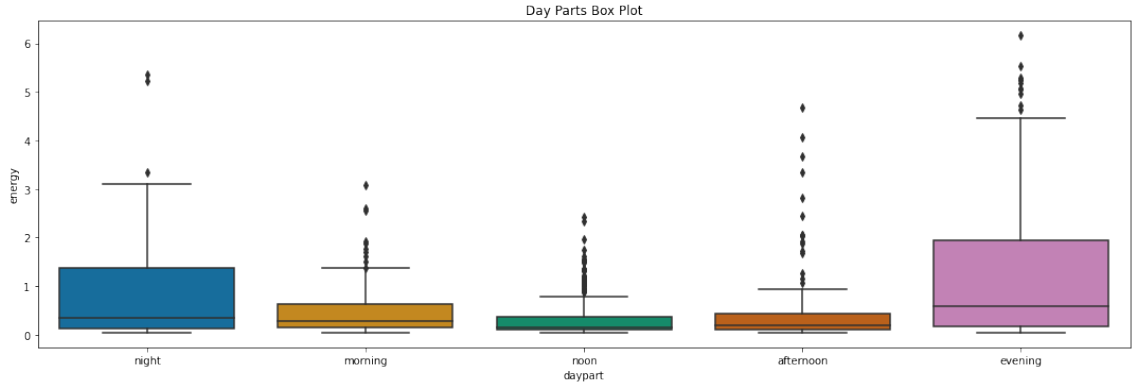

Figure 5: Daily consumption distribution during the day box plot

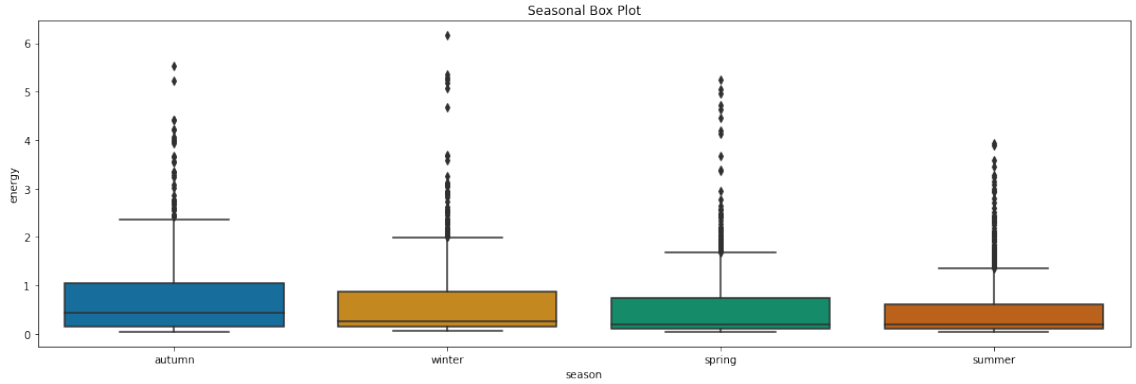

Figure 6: Seasonal energy consumption distribution box plot

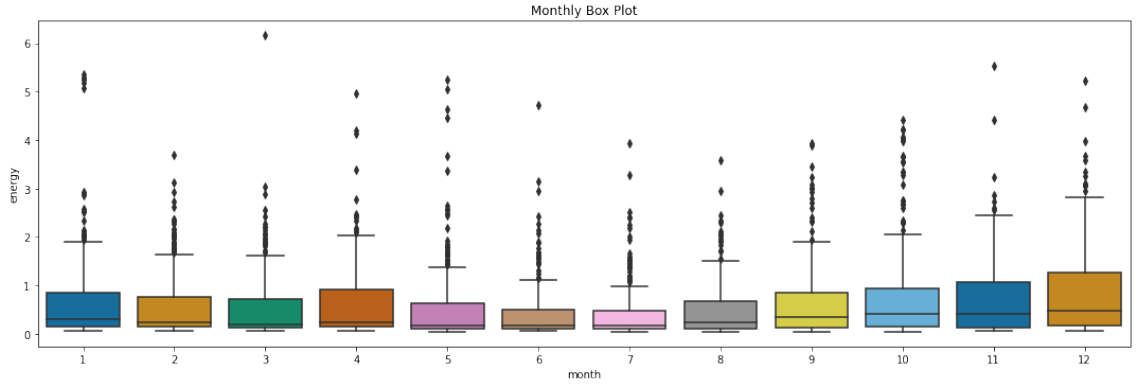

Figure 7: Monthly energy consumption distribution box plot

One can see that, indeed the daily consumption trends follow the trends on the energy consumption graph above. On top of that, from seasonal and monthly box plots, one can see that temperature can have a non negligible effect on energy consumption, as energy consumption levels in autumn and winter are considerably higher than in spring and summer. This is proven by the following graph:

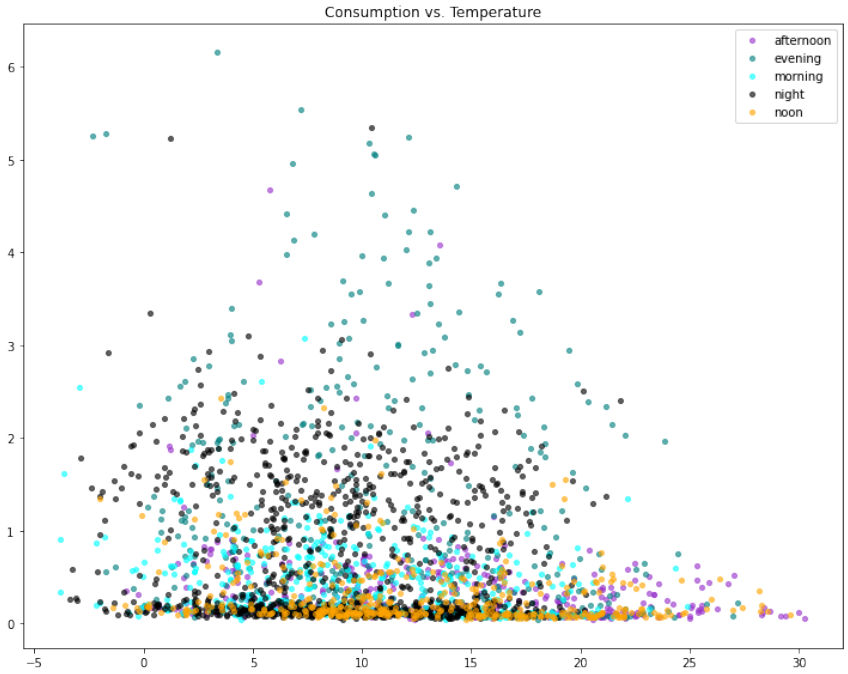

Figure 8: Energy consumption dependancy on temperature

One can see, that indeed with colder temperatures, on average energy consumption increases during all dayparts. However, such effect is observed only until a certain temperature threshold of 10°C, which could be explained by the fact that heating appliances do not consume more energy when it is colder or that this difference is minimal.

On top of that this dataset, has daily weather reports and household energy consumption data, but they were not used in the scope of this project

4 Pre-processing

Prior to the modelling part of the project, the data from dataset had to be pre-processed. This involved the following main steps:

1. Merging of energy consumption data with weather reports and bank holidays calendar
2. Cleaning of the data
3. Split of the data into multiple dataframes corresponding to weekdays
4. Addition of new features such as daypart, hour of the day, day of the month and weekday
5. Addition of lags
6. Cutting of hourly outliers
7. Label encoding of dayparts
8. Transformation of hour of the day into a sinusoidal function
9. Selection of most important features
10. Scaling of the data

4.1 Addition of lags

One of the most important parts of the pre-processing is the addition of lags to the list of the features. Indeed, it allows the model to capture the trends of the energy consumption, which would allow a more precise prediction. In order to understand how many lags would be informative an auto-correlation plot, as well as lag plots were used. On top of the choice of the number of lags from the same day, lags from previous day and week were also chosen, as previous day and previous week may also hold important information for inference of energy consumption levels. Here is the autocorrelation plot:

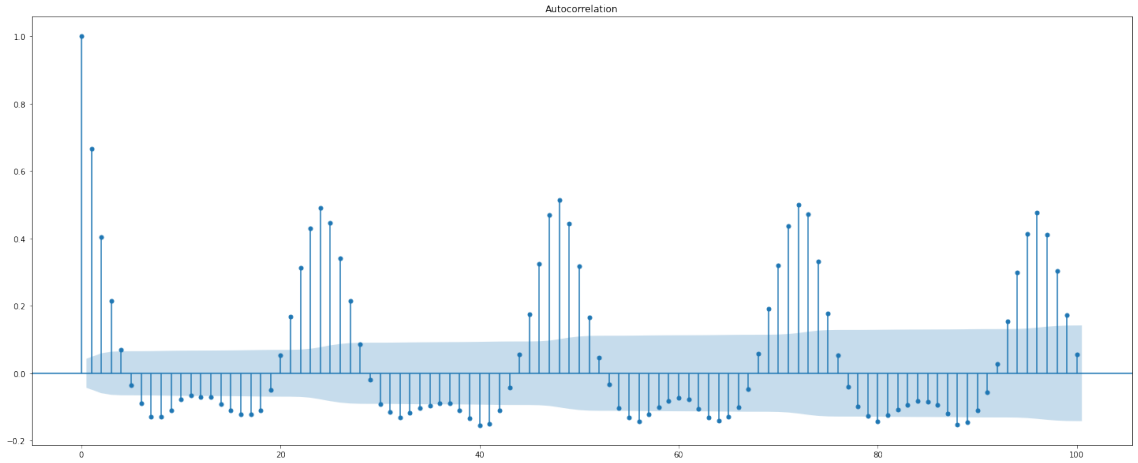

Figure 9: Autocorrelation plot

It can be seen, that after five lags, the autocorrelation decreases, and thus higher order lags would not bring new useful information for the model. On top of that, one can see, that the data has a weekly seasonality, which can be observed from the autocorrelation plot, and its recurrent peaks (as mentioned above, the dataset was split into weekdays, thus every 24 periods correspond to a new week). Thus, first five lags were chosen from the same day, and first three from the previous week, as lags from last week are less autocorrelated. Here are the lag plots obtained for this choice of lags:

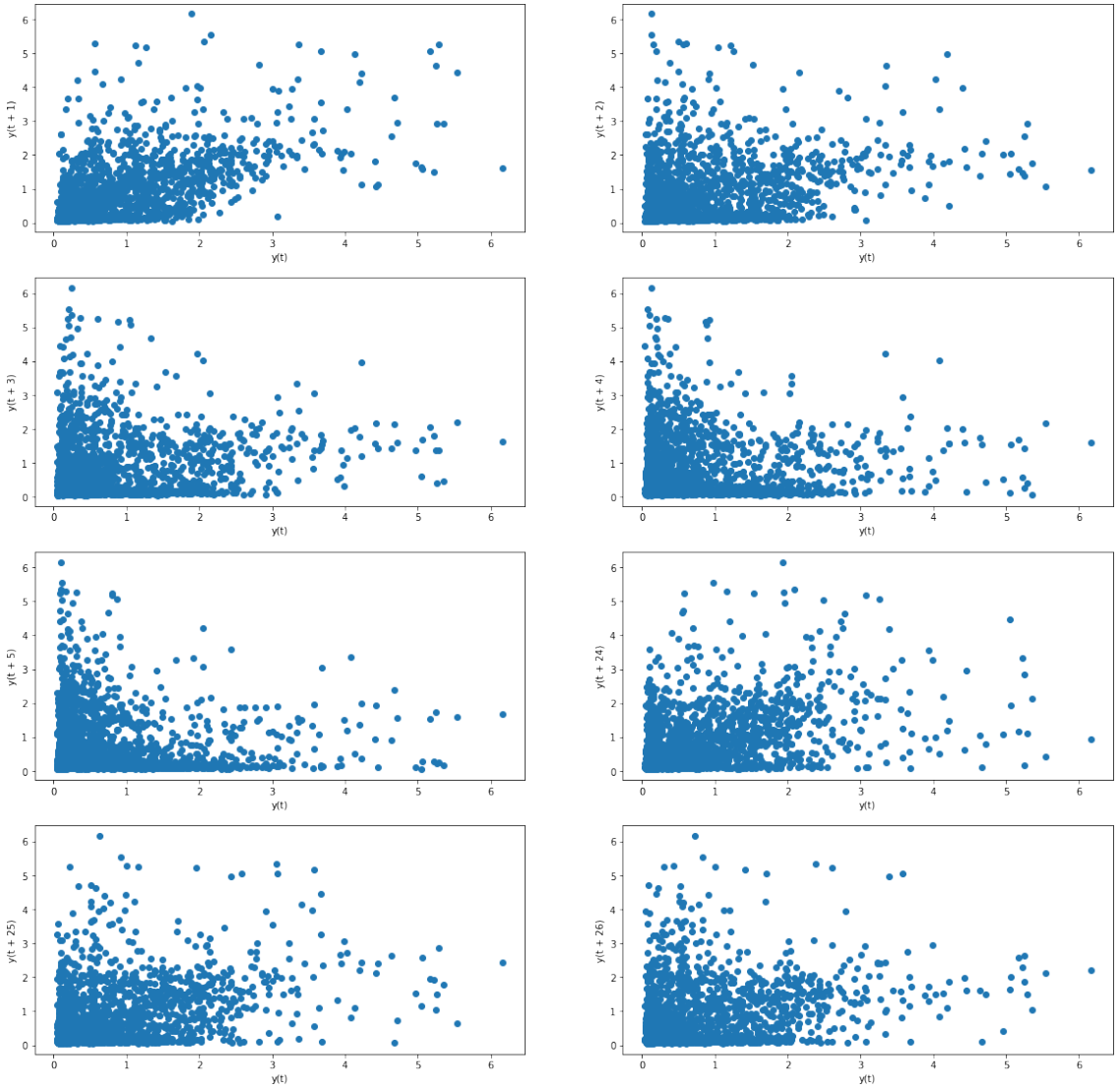

Figure 10: Lag Plots

One can see that, with the higher order of lag, autocorrelation decreases. However, one can also observe, that higher orders of lags still hold some important information, which can be seen from some points on the diagonal of the graph, which means that they are highly autocorrelated with the output. Another interesting observation comes from lags from previous week. Indeed, they break the trend of higher ordered lags being less autocorrelated, which can be seen from the increased linearity of the graph. As for the lags from previous day, as a rule of thumb the same amount of lags was chosen as for the lags coming from previous week.

4.2 Cutting of hourly outliers

The goal of this project being the capacity to model via meta-learning algorithms an average consumption pattern, an option to cut all hourly outliers in the data was introduced, so that the model only concentrates on capturing the general pattern. It produces the following box plots:

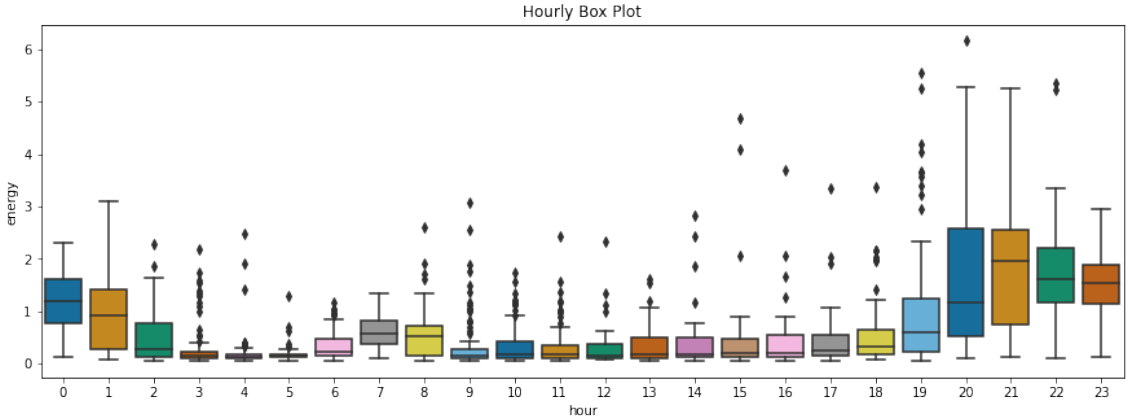

Figure 11: Hourly Energy Consumption Distribution before cutting outliers

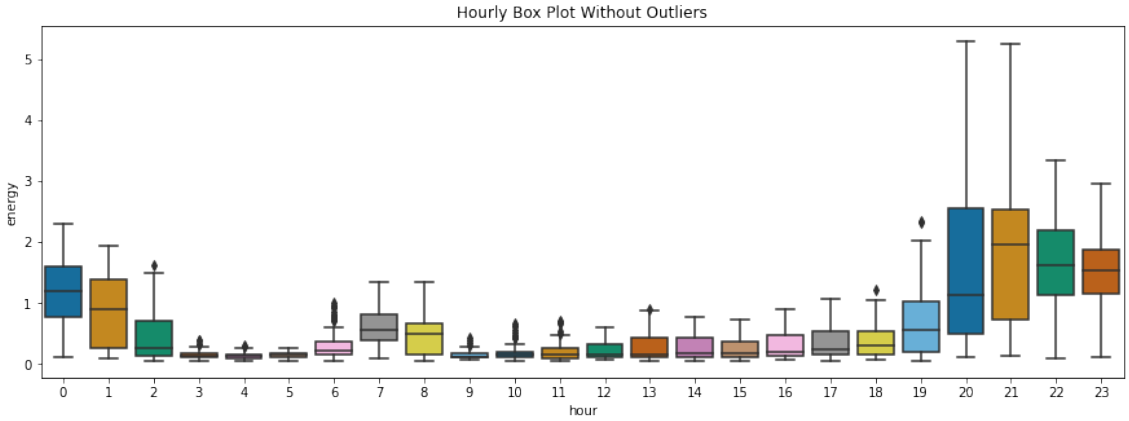

Figure 12: Hourly Energy Consumption Distribution after cutting outliers

Such operation makes the predictions to be a bit further from reality, however it also allows the models to capture general pattern, as mentioned above. From the modelling results, it would be seen that this is the case, however to make the experimentations closer to reality, data with outliers was used for meta-learning.

4.3 Selection of most important features

To get an assessment of most important features and to facilitate their choice, in order not to over-complicate the model, two different methods were used. First of all, Pearson correlation was studied between energy consumption and weather features. The following correlation matrix was obtained:

	energy	temperature	windBearing	dewPoint	windSpeed	pressure	visibility	humidity
energy	1.000000	-0.057386	0.007477	-0.067263	0.058678	-0.034913	0.014215	-0.010951
temperature	-0.057386	1.000000	0.041969	0.879141	0.003484	0.103658	0.291959	-0.458153
windBearing	0.007477	0.041969	1.000000	0.074914	0.108694	-0.065849	0.257208	0.034934
dewPoint	-0.067263	0.879141	0.074914	1.000000	-0.068265	0.028321	0.054177	0.014319
windSpeed	0.058678	0.003484	0.108694	-0.068265	1.000000	-0.457005	0.252178	-0.179804
pressure	-0.034913	0.103658	-0.065849	0.028321	-0.457005	1.000000	-0.003946	-0.143960
visibility	0.014215	0.291959	0.257208	0.054177	0.252178	-0.003946	1.000000	-0.549257
humidity	-0.010951	-0.458153	0.034934	0.014319	-0.179804	-0.143960	-0.549257	1.000000

Figure 13: Correlation matrix

Secondly, KBest algorithm from sklearn library was used. In particular f-regression score was used in order to determine the most important features. It is calculated as follows:

1. The cross correlation between each regressor and the target is computed, that is: ((X[:, i] - mean(X[:, i])) * (y - mean(y))) / (std(X[:, i]) * std(y)).
2. It is then converted to an F-score

The following scores were obtained:

KBest algorithm feature selection	
Feature	F-statistic
Temperature	256.723982
Holiday Indicator	49.182707
Visibility	19.384727
Day of Month	0.576296
Wind Bearing	7.217754
Time	4398.393473
Dew Point	348.806416
Day of Year	9.034129
Humidity	8.721532
Wind Speed	288.811323
Pressure	111.157838
Hour of day (X component of sinusoidal)	3217.412095
Hour of day (Y component of sinusoidal)	47.324949

Combining these two approaches, the following set of features was used in predictions:

- All of the lags mentioned above
- Time
- Hour of day (X component)
- Temperature
- Wind Speed
- Pressure
- Holiday Indicator
- Daypart

For the temperature feature, it was used instead of Dew Point, due to the fact that these features are highly correlated, and intuitively Temperature information can bring more insights, rather than Dew Point data. As for the last two features in the list, following the graphs presented in paragraph on dataset description, these features can bring important insights to the model, as general energy consumption pattern is highly dependent on daypart and on whether the day is the working day or not. In total, 23 features are used.

5 Modelling

After having pre-processed the data, it was split into train and test data in the proportion of 75/25. After that four various models were fit on the data for a single household, that had the most data entries, in order to determine which models would be most fit to try out in meta-learning, by comparing them to the baseline prediction model, which was simple Linear Regression.

5.1 SVR

The first model that was tried out on the prediction problem was the SVR model, in particular its sparse NuSVR iteration.

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

1. Effective in high dimensional spaces
2. Still effective in cases where number of dimensions is greater than the number of samples
3. Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient
4. Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels

The disadvantages of support vector machines include:

1. If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial
2. SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation

Support Vector Machines are powerful tools, but their compute and storage requirements increase rapidly with the number of training vectors. The core of an SVM is a quadratic programming problem (QP), separating support vectors from the rest of the training data. The overall complexity of this algorithm is $\mathcal{O}(\mathrm{n}_{features} \times n^2{}_{samples})$.

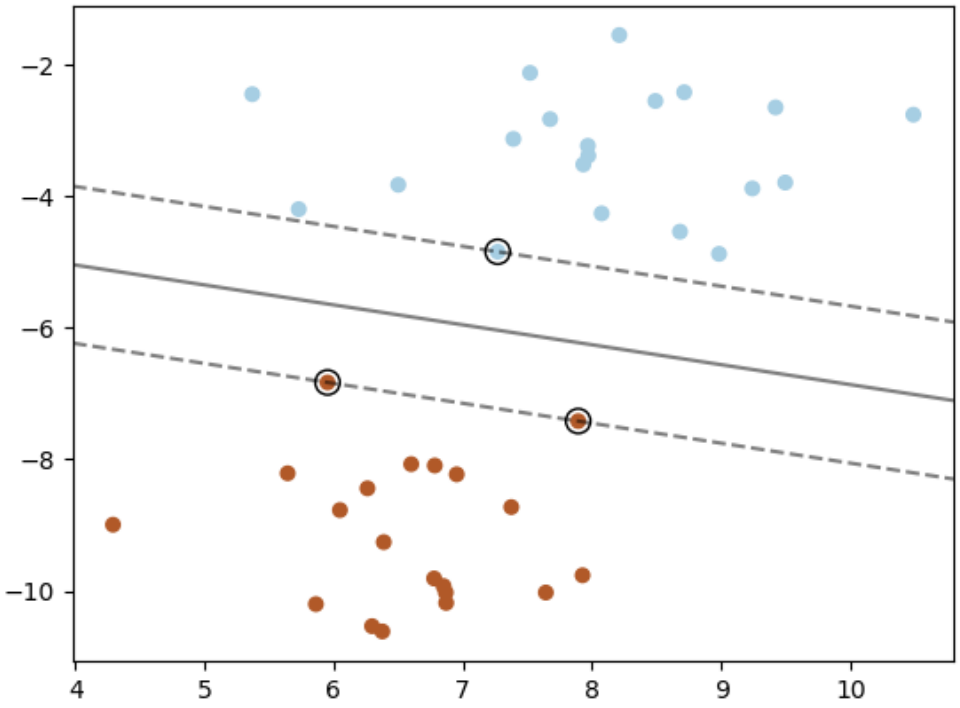

Figure 14: SVM

NuSVR is an extension to standard SVR in a such a way that it uses a parameter nu to control the number of support vectors by replacing the parameter epsilon of epsilon-SVR. Thus the main hyperparameters become:

- Kernel
- ν: An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors
- Degree (for polynomial Kernel) / Gamma (for RBF kernel)
- C: Penalty parameter C of the error term

In this particular application, polynomial kernel showed better results than RBF kernel. The following hyperparameters were chosen:

	With outliers	No outliers in train data	No outliers in all data
C	0.1	0.1	0.1
ν	0.7	0.5	0.7
Degree	1	1	1

Table 1: NuSVR hyperparameters

5.2 GPR

Gaussian Process Regression was the second model tried out for energy consumption prediction. Gaussian Processes (GP) are a generic supervised learning method designed to solve regression and probabilistic classification problems.

The advantages of Gaussian processes are:

1. The prediction interpolates the observations (at least for regular kernels)
2. The prediction is probabilistic (Gaussian) so that one can compute empirical confidence intervals and decide based on those if one should refit (online fitting, adaptive fitting) the prediction in some region of interest
3. Versatile: different kernels can be specified. Common kernels are provided, but it is also possible to specify custom kernels

The disadvantages of Gaussian processes include:

1. They are not sparse, i.e., they use the whole samples/features information to perform the prediction
2. They lose efficiency in high dimensional spaces – namely when the number of features exceeds a few dozens

The optimization of hyperparameters of this model is automatic. The hyperparameters of the kernel are optimized during fitting, based on the maximization of the log-marginal-likelihood (LML). The overall complexity of this algorithm is $\mathcal{O}(\mathrm{n}_{features} \times n^3{}_{samples})$

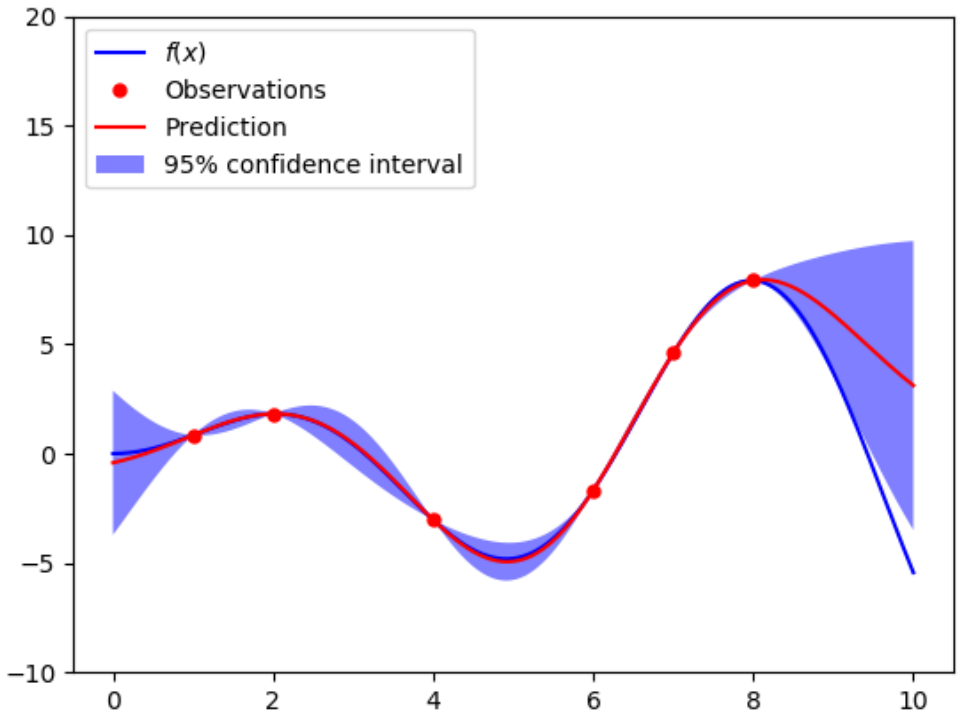

Figure 15: GPR

As mentioned above hyperparameter tuning in GPR is automatic. However, the user has to choose correct kernel. Sklearn library proposes the following kernels:

- Constant Kernel
- White Kernel: explains the noise-component of the signal
- RBF Kernel: parameterized by a length-scale parameter l
- Matérn Kernel: stationary kernel and a generalization of the RBF kernel. It has an additional parameter which controls the smoothness of the resulting function
- Rational quadratic Kernel: can be seen as a scale mixture (an infinite sum) of RBF kernels with different characteristic length-scales
- Exp-Sine-Squared Kernel: allows modeling periodic functions
- Dot-Product Kernel

It should be noted that all of the kernels above can be summed or multiplied between each other, in order to be able to account for more complex signals. In this particular project a simple RBF Kernel multiplied with Dot-Product was used. The multiplication was done in order to take into account the residuals and thus improve the overall fit. On top of that, in order to take into account possible noise in signal, White Kernel was added to the Kernel combination mentioned before.

5.3 XGBoost

Next algorithm that was tested was XGBoost algorithm, which is often competition winning algorithm in Kaggle challenges. It is a gradient boosting algorithm, which has the advantage of being fast and having good performance. Main features of XGBoost include:

- Clever penalization of trees
- A proportional shrinking of leaf nodes
- Newton Boosting
- Extra randomization parameter
- Implementation on single, distributed systems and out-of-core computation
- Automatic Feature selection

Model is fit using any arbitrary differentiable loss function and gradient descent optimization algorithm. In particular XGBoost algorithm is the following:

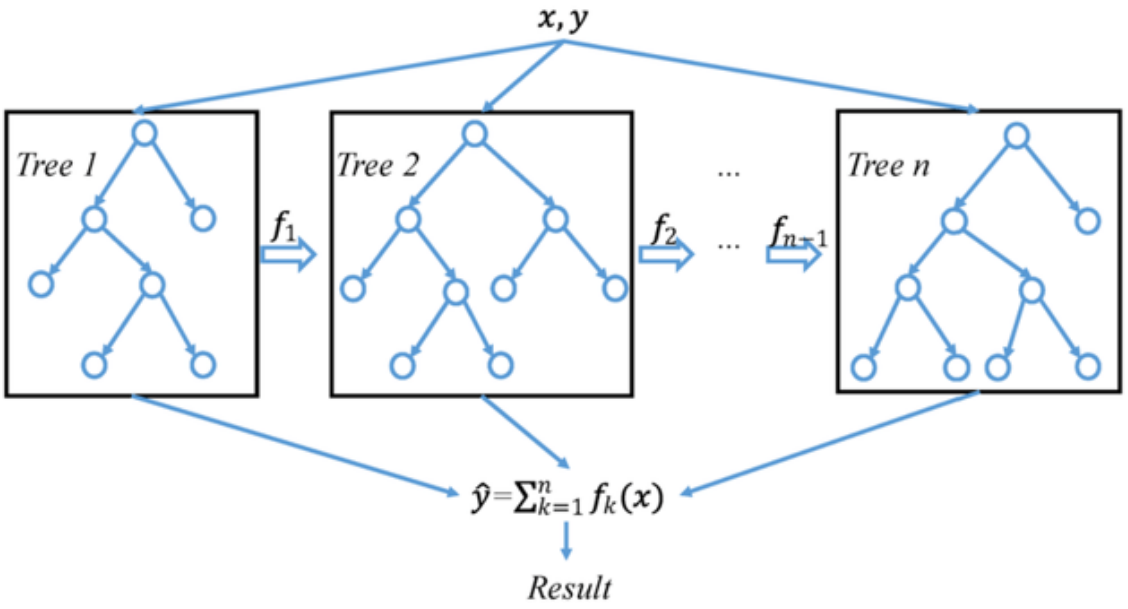

Figure 16: XGBoost

Input: training set $\{(x_i, y_i)\}_{i=1}^{N}$, a differentiable loss function $L(y, F(x))$, a number of weak learners M and a learning rate α.

Algorithm:

1. Initialize model with a constant value:

$$\hat{f}_{(0)}(x) = \arg\min_{\theta} \sum_{i=1}^{N} L(y_i, \theta).$$

2. For m = 1 to M:

   1. Compute the 'gradients' and 'hessians':

$$\hat{g}_m(x_i) = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}_{(m-1)}(x)}.$$

$$\hat{h}_m(x_i) = \left[\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}\right]_{f(x)=\hat{f}_{(m-1)}(x)}.$$

   2. Fit a base learner (or weak learner, e.g. tree) using the training set $\{x_i, -\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}\}_{i=1}^{N}$ by solving the optimization problem below:

$$\hat{\phi}_m = \arg\min_{\phi \in \Phi} \sum_{i=1}^{N} \frac{1}{2} \hat{h}_m(x_i) \left[-\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)} - \phi(x_i)\right]^2.$$

$$\hat{f}_m(x) = \alpha \hat{\phi}_m(x).$$

   3. Update the model:

$$\hat{f}_{(m)}(x) = \hat{f}_{(m-1)}(x) + \hat{f}_m(x).$$

3. Output $\hat{f}(x) = \hat{f}_{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x).$

Figure 17: XGBoost algorithm

Here is the list of hyperparameters, that were tuned in the scope of this project:

- n_estimators
- min_child_weight: Minimum sum of instance weight (hessian) needed in a child.
- max_depth: Maximum depth of a tree
- Learning rate

- gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree

	With outliers	No outliers in train data	No outliers in all data
n_estimators	500	500	500
min_child_weight	0.8	0.4	0.4
max_depth	8	14	14
Learning rate	0.05	0.02	0.02
gamma	0.9	0.8	0.8

Table 2: NuSVR hyperparameters

As for the NuSVR, the tuning was done manually. A better practice would be to use a grid search, however in the scope of this project, due to overall complexity of interpretation of XGBoost algorithm and its parameters' updates, which means that it will not be used for meta-learning, manual tuning was deemed sufficient. However, as it would be seen in the results' section, XGBoost is one of the best algorithms for energy consumption prediction, thus it was interesting to see how competition winning would perform for this task.

5.4 LSTM

Final model that was used for single household modelling was LSTM Deep Learning model. LSTM networks are well-suited to classifying, processing and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series, therefore such network could provide great results for energy consumption prediction problem.

The main advantages of LSTM are:

1. LSTM works great with time series
2. LSTM is able to bridge very long time lags
3. LSTM can handle noise, distributed representations and continuous values
4. There is no need for fine parameter tuning
5. LSTM generalizes well
6. Low update complexity per weight and per time step: $\mathcal{O}(1)$

The disadvantages of LSTM include:

1. LSTM fails to remove completely problem of vanishing gradient
2. Hardware-wise, LSTMs are quite inefficient
3. LSTM get affected by different random weight initialization and hence behave quite similar to that of a feed-forward neural net, but it can be solved by small weight initialization
4. LSTMs are prone to overfitting and it is difficult to apply the dropout algorithm to curb this issue

One of the other advantages of LSTM is that it can be combined with Linear layers of a neural net, to take advantage of LSTM hidden layers, or it can also stack multiple LSTM layers, which can improve the results.

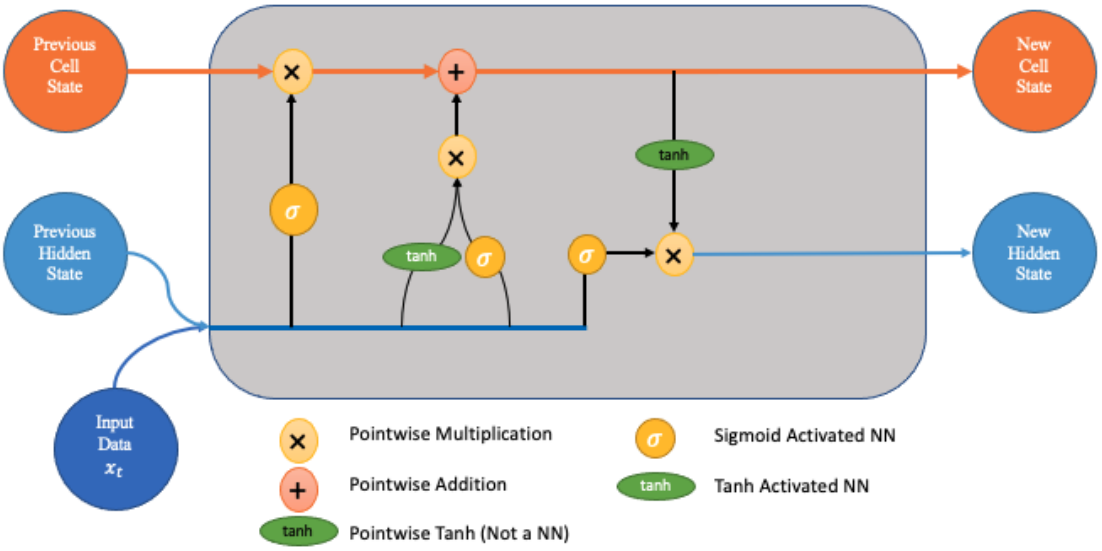

Figure 18: LSTM cell

Parameters that need to be fixed by the user are the following:

Parameter	Value used
Epochs	1000
Learning rate	0.225
Scheduler Gamma	0.8
Number of stacked LSTM layers	4
Number of hidden layers	50
Dropout probability	0.2
Loss function	L1 Loss

Table 3: LSTM parameters

The LSTM model architecture that was used in the scope of this project is the following:

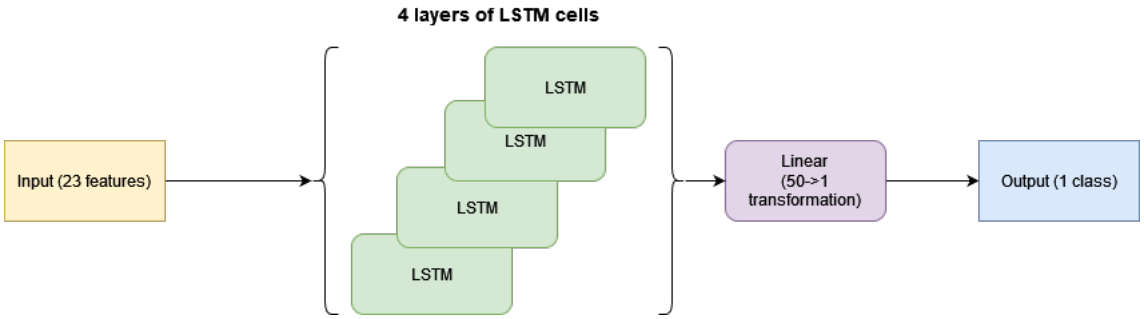

Figure 19: LSTM architecture

6 Results

6.1 Metrics

Before analyzing obtained results by different models, a small overview of metrics used for results' assessment is needed. In total four different metrics were used:

1. Mean Squared Error (MSE)
2. R2 Score
3. Mean Absolute Error (MAE)
4. Mean Directional Accuracy (MDA)

The last metric (MDA), represents the following: it compares the forecast direction (upward or downward) to the actual realized direction. It is calculated as follows:

$$MDA = \frac{1}{N} \sum_{t} \mathbf{1}_{sgn(A_t - A_{t-1}) = sgn(F_t - A_{t-1})}$$

A_t is the actual value at time t and F_t is the forecast value at time t

6.2 Results

For this section, zoomed in prediction plots will presented, alongside metric results for the algorithms that were used for energy consumption prediction. As mentioned in the previous section, all algorithms were compared to the baseline model, which was a simple Linear Regression. Here is the plot, that was obtained for test data prediction:

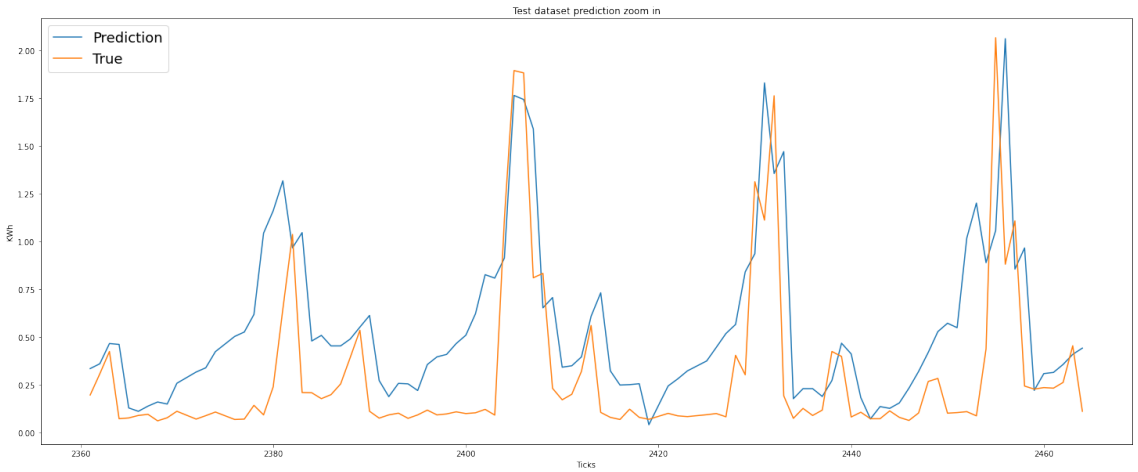

Figure 20: Linear Regression prediction

Its goodness of prediction was assessed with metrics described above and here are the obtained results:

Metric	Value
MSE	0.272
R2	0.369
MDA	0.532
MAE	0.372

Table 4: Linear Regression metric results

Overall, one can see that even a simple Linear Regression is able to model correctly energy consumption peaks. However, it fails to model low level energy consumption, which results in a more linear increase in its consumption, in comparison to the real data. What one could expect from more sophisticated models, is the improvement in the modelling of low-level energy consumption levels, while maintaining the quality of prediction for energy peaks.

As mentioned before, the first algorithm that was compared to Linear Regression, was NuSVR algorithm. Here is the prediction obtained with this algorithm:

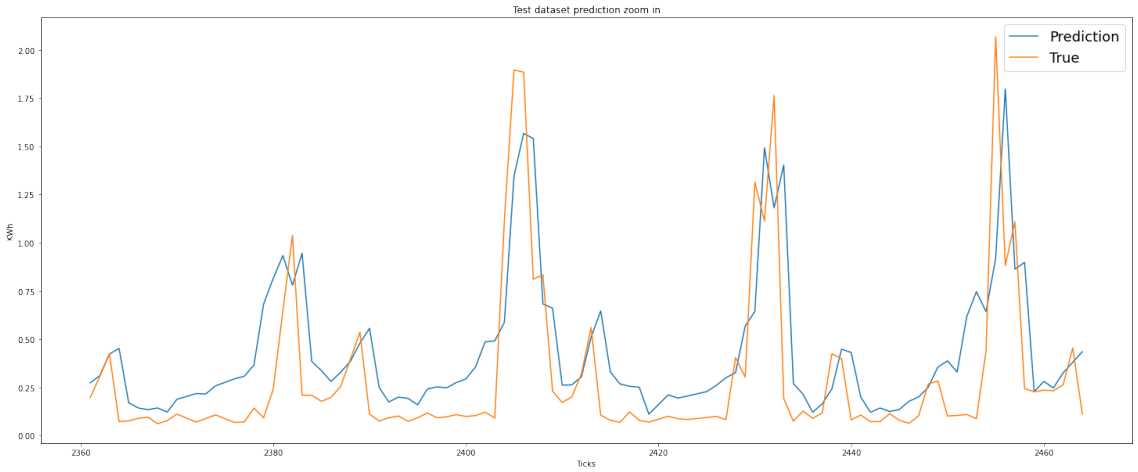

Figure 21: NuSVR prediction

Here are the metric results assessing NuSVR's goodness of fit:

Metric	Value
MSE	0.229
R2	0.468
MDA	0.497
MAE	0.310

Table 5: NuSVR metric results

As expected, after inspecting the plot, thanks to better modelling of low energy levels the overall goodness of fit has greatly increased. However, one of interesting insights, that we get from MDA metric and general plot observation, is the fact that NuSVR algorithm puts a lot of important on lags, and tends to predict a scaled value of the previous timestamp. To sum up, despite a small decrease in fit quality in peaks' modelling, their modelling remains acceptable, which makes the

overall error to go down, in comparison with Linear Regression.

Next algorithm to be inspected is the GPR. It is a particularly interesting algorithm, as it has a possibility to directly output confidence intervals, which is extremely important for forecasting tasks. The prediction obtained with this algorithm is the following:

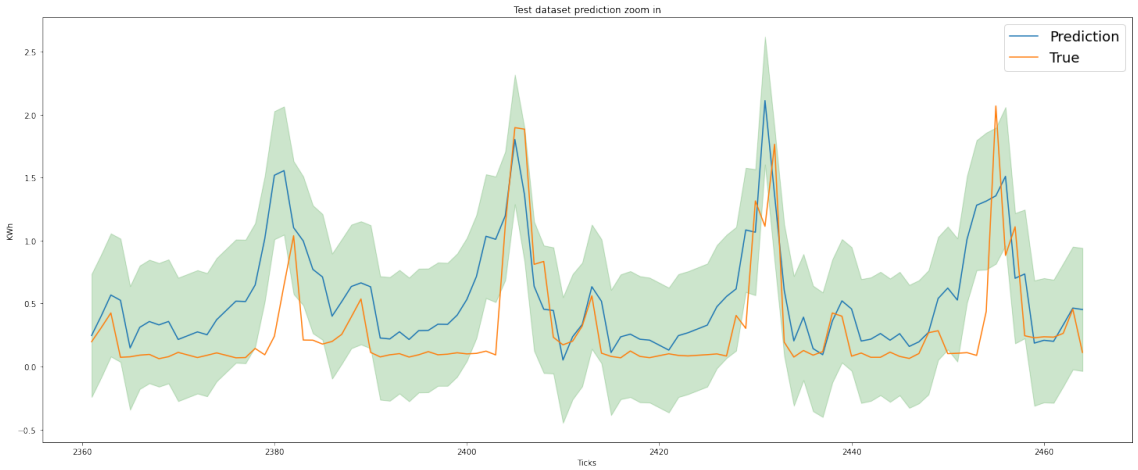

Figure 22: GPR Prediction

Its goodness of fit is as follows:

Metric	Value
MSE	0.255
R2	0.41
MDA	0.595
MAE	0.355

Table 6: GPR metric results

Generally speaking, GPR presents a better alternative to Linear Regression, once again mainly thanks to a better modelling of low energy levels, which is still slightly worse than the of NuSVR. However, energy peaks modelling quality is worse for GPR model, which is the main explanation for the fact of MSE, R2 and MAE metrics being worse than the ones of NuSVR. Nonetheless, GPR tries to model energy consumption prediction independently from the lags, and tries to put more importance on exogenous factors, such as weather conditions, which can be seen from MDA metric, which is higher than for Linear Regression. On top of that, GPR offers confidence bounds prediction, which makes this algorithm particularly attractive for meta-learning.

Last classical machine learning algorithm that was analyzed was XGBoost, an algorithm as mentioned above, that is based on boosting. Thus, in the end it allows to compare algorithms, that all use different methods for regression. The prediction plot obtained for XGBoost is the following:

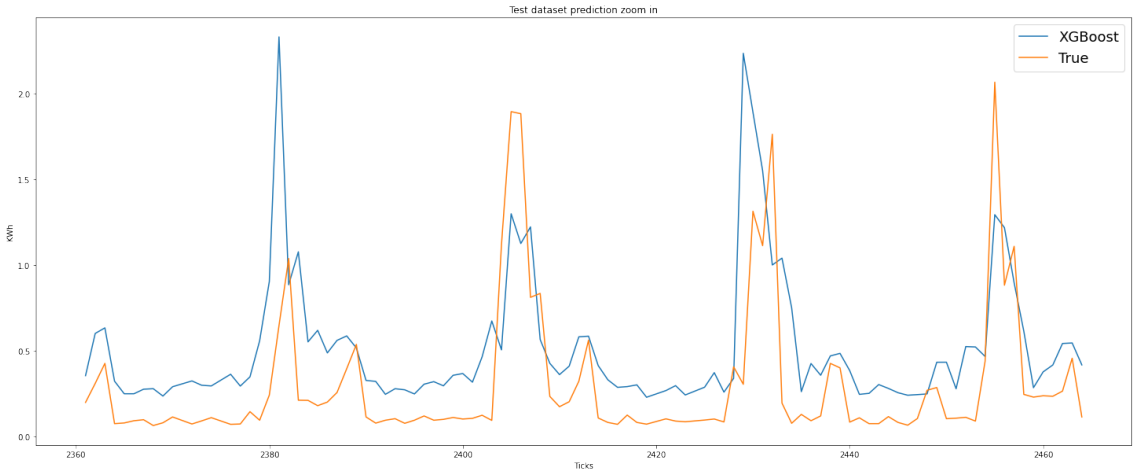

Figure 23: XGBoost prediction

The metric results for this algorithm are the following:

Metric	Value
MSE	0.225
R2	0.479
MDA	0.497
MAE	0.346

Table 7: XGBoost metric results

One of the first observations that can be made from the plot, is the fact that XGBoost is able to nicely capture low-level consumption pattern, however it also has a constant bias. This observation is proven by MAE, as it better takes into account errors for low-level energy consumption, while MSE penalises more for energy peaks modelling errors. As MSE is even lower, than for NuSVR, one can conclude that overall XGBoost is able to correcly model energy peaks. By observing the graph, one can see that XGBoost, like GPR, does not try to predict a scaled value of a previous timestamp, however there are parasitic fluctuations in prediction pattern, which results in a low MDA score.

Final model that was analyzed is the LSTM model, that is considered to be state of art model for forecasting. Here is the plot for this model:

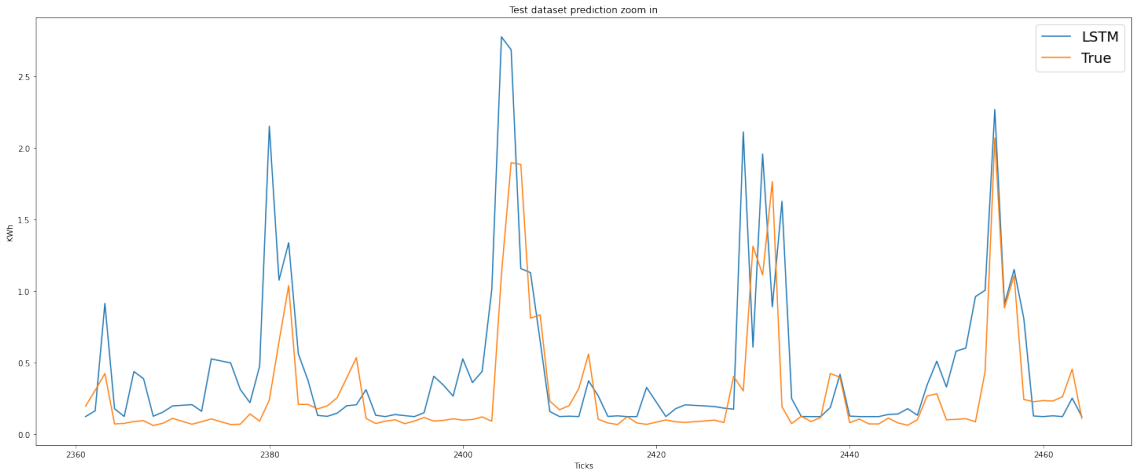

Figure 24: LSTM prediction

The following metric results were obtained:

Metric	Value
MSE	0.268
R2	0.38
MDA	0.562
MAE	0.317

Table 8: LSTM metric results

One of the first observation, like for the XGBoost, concern the quality of low-level energy consumption modelling. In fact, it appears to be well modelled, which is proven by plot and MAE metric. However, one can also observe that LSTM struggles to model correctly energy peaks. One of the main reasons for that, may be the fact that there is not enough data for LSTM to grasp all the ambiguities, which results into high MSE and low R2 score. Indeed, only 2080 samples are available for training, which is generally not enough for deep learning models. In fact, as one would see for the case of meta-learning, doubling the number of samples, would already allow LSTM to bypass XGBoost and NuSVR in terms of performance. On top of that, LSTM is an algorithm that has an architecture that can be easily modified by adding Linear layers or by adding more LSTM layers. Therefore, this algorithm will certainly be tested for meta-learning.

Finally, here is the plot that allows to compare all the algorithms used side by side, and illustrates all the previous analysis:

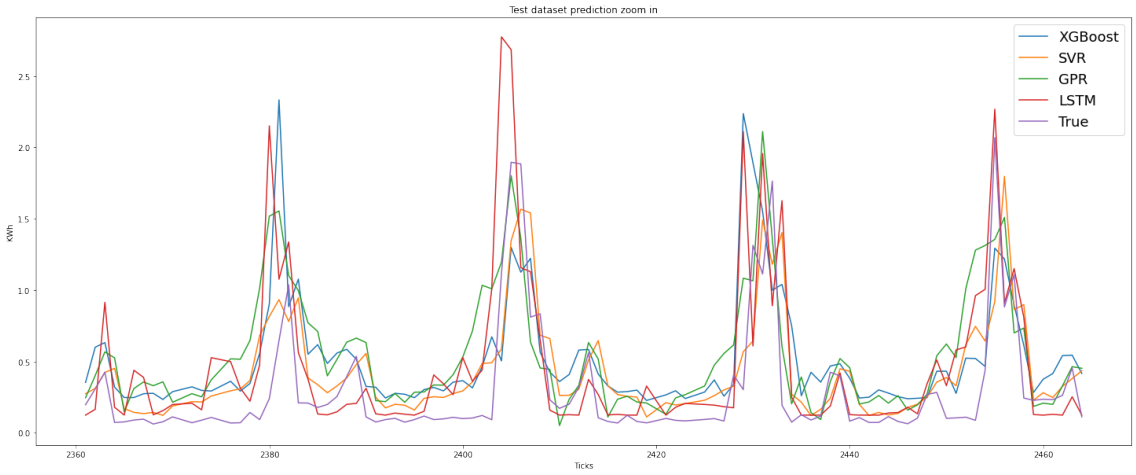

Figure 25: Inter-algorithm comparison

7 What is Meta-Learning

A good meta-learning algorithm aims to solve the problem of solving the problem of fast training with small amount of samples. One expects a good meta-learning model capable of well adapting or generalizing to new tasks and new environments that have never been encountered during training time. The adaptation process, essentially a mini learning session, happens during test but with a limited exposure to the new task configurations.[2] In particular, in the scope of this project 200 samples from each household measurements were taken for training, which accounts for 10-25% of total samples, depending on the household. For model-adjustment purposes to a specific household 100 samples were taken. Eventually these numbers were sufficient for both models (GPR and LSTM) to be able to adapt and learn new tasks.

A good meta-learning model should be trained over a variety of learning tasks and optimized for the best performance on a distribution of tasks, including potentially unseen tasks. Each task is associated with a dataset $\mathcal{D}$, containing both feature vectors and true labels. The optimal model parameters are:

$$\theta^* = \arg\min_\theta \mathbb{E}_{D\sim p(\mathcal{D})}[L_\theta(\mathcal{D})] \qquad \text{with } \theta \text{ being model parameters}$$

It looks very similar to a normal learning task, but one dataset is considered as one data sample. In fact, the dataset $\mathcal{D}$ is often split into two parts, a support set S for learning and a prediction set B for training or testing, $\mathcal{D}$=⟨S,B⟩.

A dataset $\mathcal{D}$ contains pairs of feature vectors and labels, $\mathcal{D}$=(x_i,y_i) and each label belongs to a known label set $\mathcal{L}^{label}$.

8 Meta-Learning Modelling

After having fit the models on a single household, the next step was to implement and assess various meta-learning algorithms. First, of all two main machine learning algorithms to be adapted for meta-learning were chosen. In particular, GPR and LSTM algorithms. The reasons for such choice, were the fact that both of these algorithms use gradient descent for the update of hyper-parameters (in case of GPR, log-marginal-likelihood is used for gradient update), and that they are both easily interpretable in terms of how their parameter weights are updated, which is not the case for NuSVR and especially XGBoost, even though the latter two algorithms provide better

results for single household energy consumption prediction (as mentioned above in case of LSTM, LSTM has a low performance due to low amounts of data).

Three main meta-learning algorithms were chosen to be most prospective, due to the fact that these algorithms are said to be model agnostic, and thus any standard machine learning model can be adapted for usage in these algorithms:

1. Reptile
2. MAML (Model-Agnostic Meta-Learning)
3. LSTM Meta-Learner

8.1 Reptile

Reptile is a remarkably simple meta-learning optimization algorithm. It relies on meta-optimization through gradient descent and is model-agnostic.

The Reptile works by repeatedly:

1. sampling a task
2. training on it by multiple gradient descent steps
3. moving the model weights towards the new parameters

In other words Reptile algorithm is the following: **SGD($\mathcal{L}_{\tau_i}$,θ,k)** performs stochastic gradient update for k steps on the loss $\mathcal{L}_{\tau_i}$ starting with initial parameter θ and returns the final parameter vector. The batch version samples multiple tasks instead of one within each iteration. The reptile gradient is defined as $(\theta - \mathrm{W})/\alpha$, where α is the stepsize used by the SGD operation.

Algorithm 1 Reptile

Initialize θ

for iteration = 1,2,... **do**
  Sample tasks τ_1, τ_2,...,τ_n
  for i = 1,2,...n, **do**
    Compute W_i = SGD($\mathcal{L}_{\tau_i}$,θ,k)
  end for
  $\theta \leftarrow \theta + \beta \frac{1}{n} \sum_{i=1}^{n} (W_i - \theta)$
end for

In case of GPR, SGD is based on the log-marginal-likelihood and its maximization. As for the LSTM based model, standard MSE loss, was changed to mean squared logarithmic error for SGD estimation, as it penalizes less for errors in estimation of energy peaks, and thus allows to better take low energy levels into account. This loss is calculated as follows:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(y_i+1)-\log(out_i+1))^2}$$ with y_i being real output, and out_i being estimated output

Reptile algorithm requires user to set only two parameters:

Parameter	Value used GPR	Value used LSTM
Inner loop learning rate(SGD learning rate)	0.01	0.02
Outer loop learning rate(β)	0.2	0.4
Number of iterations	25	1000

Table 9: Reptile parameters

On top of that a few LSTM specific parameters have to fixed:

Parameter	Value used
Epochs	1000
Number of stacked LSTM layers	1
Number of hidden layers	50
Dropout probability	0.1

Table 10: LSTM Reptile parameters

8.2 MAML

MAML, short for Model-Agnostic Meta-Learning is a fairly general optimization algorithm, compatible with any model that learns through gradient descent and is similar to Reptile.

Given a task τ_i and its associated dataset $({\mathcal{D}_{(i)}}^{train}, {\mathcal{D}_{(i)}}^{test})$, one can update the model parameters by one or more gradient descent steps. MAML algorithm proceeds as follows:

Algorithm 2 MAML

Require: $p(\mathcal{T})$: distribution over tasks

Require: α, β: step size hyperparameters

1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for all** $\mathcal{T}_i$ **do**
5: Evaluate $\Delta_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to K examples
6: Compute adapted parameters with gradient descent: $\theta_i^{'} = \theta - \alpha \Delta_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
7: **end for**
8: $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i^{'}})$
9: **end while**

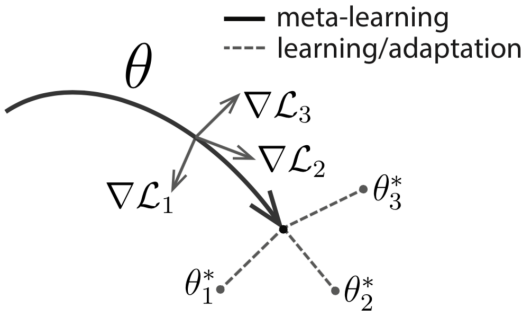

Figure 26: MAML

Overall, MAML and Reptile are quite similar algorithms, which have slighly different gradient update schemes, which set different optimization priorities. Both MAML and Reptile aim to optimize for the same goal, better task performance (guided by A (being average gradient of task loss)) and better generalization (guided by B(being the direction (gradient) that increases the inner product of gradients of two different mini batches for the same task)). This leads to the following dependencies for these algorithms:

$\mathbb{E}_{\tau,1,2}[g_{\text{MAML}}] = A - 2\alpha B + O(\alpha^2)$
$\mathbb{E}_{\tau,1,2}[g_{\text{Reptile}}] = 2A - \alpha B + O(\alpha^2)$
Thus, Reptile would adapt quicker and better to a new task, while MAML would generalize better.

As in the case of Reptile meta-learning algorithm, loss function for LSTM is mean squared logarithmic error, and for GPR the optimization relies on the maximization of log-marginal-likelihood. The parameters, that user has to pass to MAML algorithm are the following:

Parameter	Value used GPR	Value used LSTM
Inner loop learning rate(α)	0.01	0.02
Outer loop learning rate(β)	0.01	0.4
Number of iterations	25	1000

Table 11: MAML parameters

On top of that a few LSTM specific parameters have to fixed:

Parameter	Value used
Epochs	1000
Number of stacked LSTM layers	1
Number of hidden layers	50
Dropout probability	0.1

Table 12: LSTM MAML parameters

8.3 LSTM Meta-Learner

Finally, a more experimental and sophisticated meta-learning algorithm was tried out. It was only tried out with LSTM learner, due to the complexity of its implementation and its adaptation to

standard machine learning techniques.

The meta-learner is modeled as a LSTM, because:

1. There is similarity between the gradient-based update in backpropagation and the cell-state update in LSTM
2. Knowing a history of gradients benefits the gradient update

The update for the learner's parameters at time step t with a learning rate αt is:

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t \tag{1}$$

It has the same form as the cell state update in LSTM, if one sets forget gate f_t=1, input gate i_t= α_t, cell state $c_t = \theta_t$, and new cell state $\tilde{c}_t = -\nabla_{\theta_{t-1}} \mathcal{L}_t$:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t \tag{2}$$

While fixing f_t=1 and i_t= α_t might not be the optimal, both of them can be learnable and adaptable to different datasets.

$f_t = \sigma(\mathbf{W}_f \cdot [\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, f_{t-1}] + \mathbf{b}_f)$; how much to forget the old value of parameters.
$i_t = \sigma(\mathbf{W}_i \cdot [\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, i_{t-1}] + \mathbf{b}_i)$; corresponding to the learning rate at time step t.
$\tilde{\theta}_t = -\nabla_{\theta_{t-1}} \mathcal{L}_t$
$\theta_t = f_t \odot \theta_{t-1} + i_t \odot \tilde{\theta}_t$

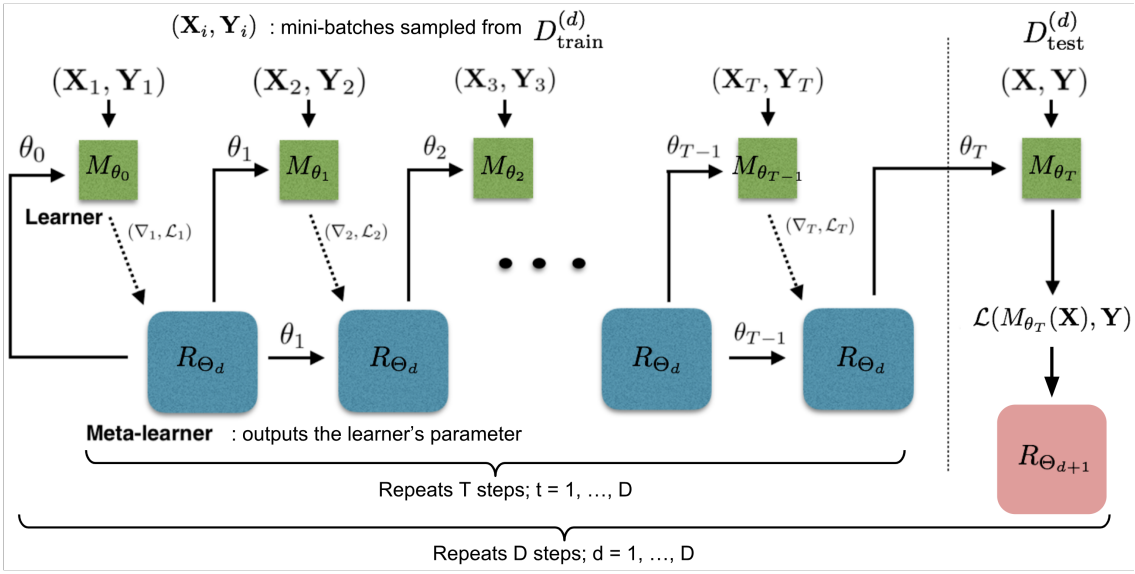

Figure 27: LSTM Meta-Learner

Here is the LSTM Meta-learner algorithm that explains the image above:

Algorithm 3 LSTM Meta-learner

Require: Meta-learning set $\mathcal{D}_{meta-train}$, Learner M with parameters θ, Meta-learner R with parameters Θ

randomly initialize Θ_0
for d = 1,n **do**
  $\mathcal{D}_{train}, \mathcal{D}_{test} \leftarrow$ random dataset from $\mathcal{D}_{meta-train}$
  $\theta_0 \leftarrow c_0$ ▷ Initialize learner parameters
  for t = 1,T **do**
    $X_t, Y_t \leftarrow$ random batch from $\mathcal{D}_{train}$
    $\mathcal{L}_t \leftarrow \mathcal{L}(M(X_t; \theta_{t-1}; Y_t)$ ▷ Get loss of learner on train batch
    $c_t \leftarrow R((\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t); \Theta_{d-1}$ ▷ Get output of meta-learner
    $\theta_t \leftarrow c_t$ ▷ update learner parameters
  end for
  $X, Y \leftarrow \mathcal{D}_{test}$
  $\mathcal{L}_{test} \leftarrow \mathcal{L}(M(X; \theta_T; Y)$ ▷ Get loss of learner on test batch
  Update Θ_d using $\nabla_{\Theta_{d-1}} \mathcal{L}_{test}$ ▷ Update meta-learner parameters
end for

In the scope of this project, LSTM architecture described in previous section was used as a Learner. Meta-learner is as described just above having the following architecture:

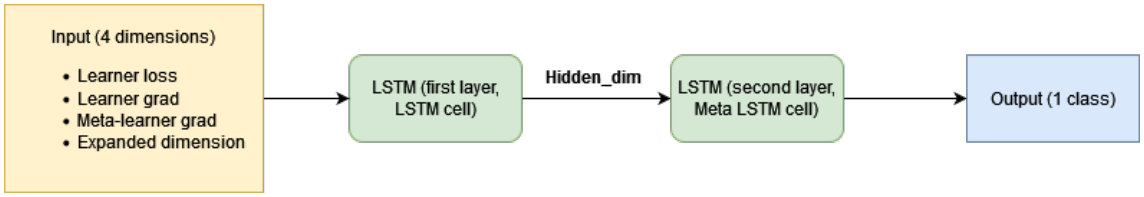

Figure 28: Meta-LSTM architecture

The last expanded dimension, is a prerequisite for LSTM cells in Torch library.

This architecture has the following parameters that need to be set prior to training:

Parameter	Value used
Epochs	500
Learning rate	0.1
Gamma of scheduler	0.9
Number of hidden layers in learner LSTM	50
Number of hidden layers in meta LSTM	3
Dropout probability	0.2

Table 13: LSTM Meta-learner parameters

As it was mentioned in the drawbacks of LSTM cells in general, they are very hardware demanding. Thus, due to hardware limitations, the number of hidden layers is not optimal, and many better-performing learner architectures could not have been tried out.

9 Meta-Learning Results

In this section meta-learning results will be presented. As mentioned before, two algorithms were chosen for adaptation for meta-learning: GPR and LSTM. They were adapted for Reptile and MAML algorithms. The tests were done on three different households, that have different regularity in their consumption habits: MAC000246 (household used for standard machine learning models' evaluation in previous section on results), MAC003463 and MAC003718. Same parameters have been used for each of the households.

9.1 Metrics

For the assessment of goodness of fit, same metrics were used as in the previous section on results:

1. Mean Squared Error (MSE)
2. R2 Score
3. Mean Absolute Error (MAE)
4. Mean Directional Accuracy (MDA)

9.2 Reptile

The first meta-learning algorithm that was tested was the Reptile. First, here are the plots for GPR predictions using Reptile meta-learning algorithm for households mentioned above:

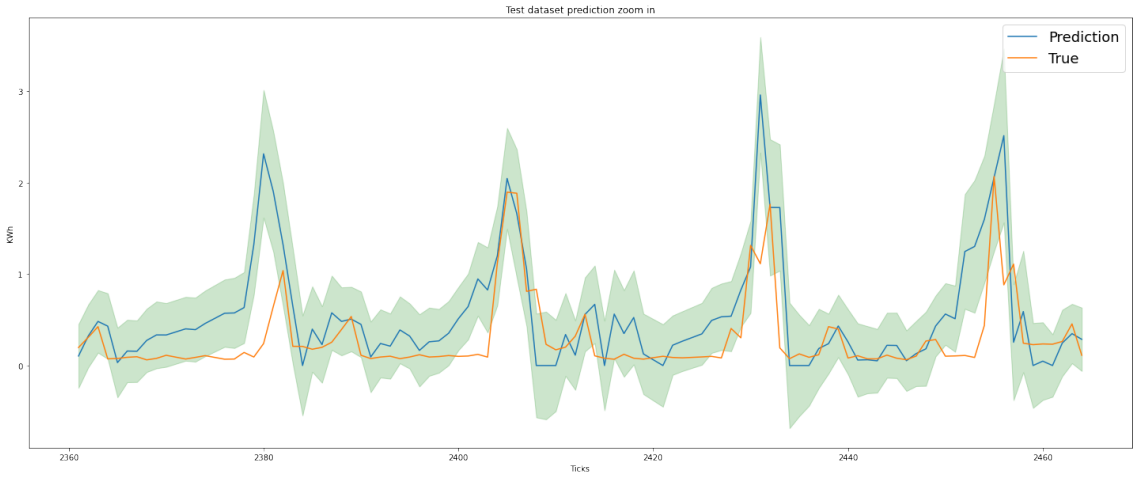

Figure 29: GPR Reptile predictions for MAC000246

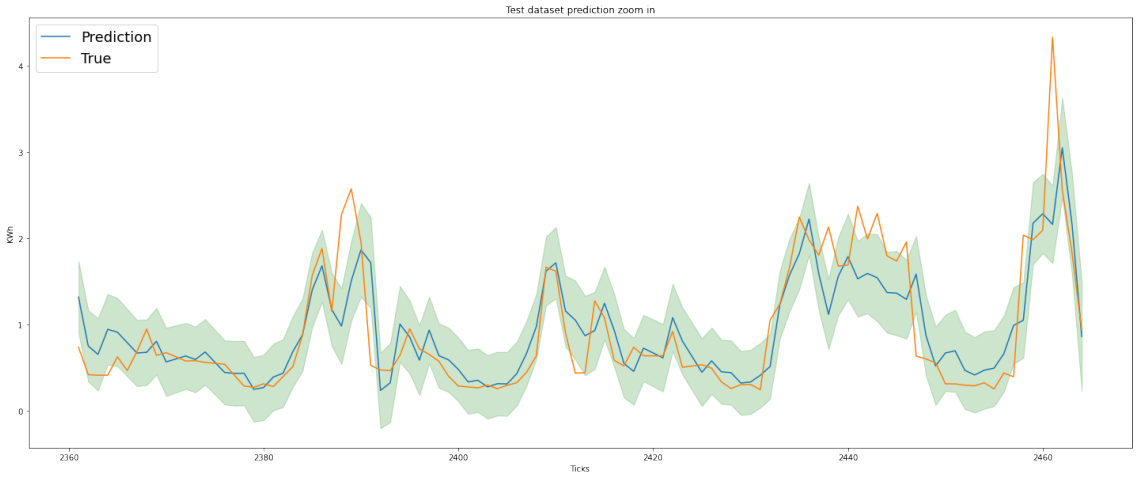

Figure 30: GPR Reptile predictions for MAC003718

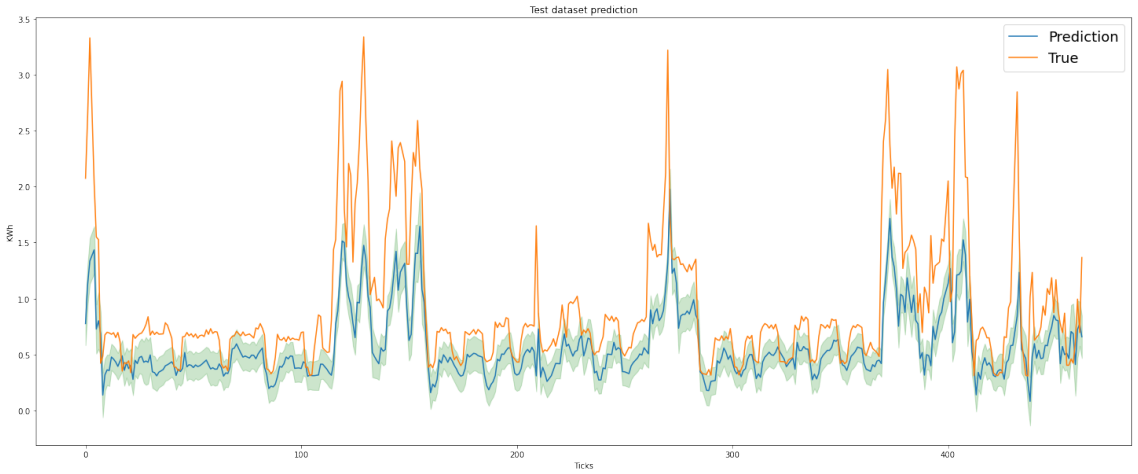

Figure 31: GPR Reptile predictions for MAC003463

Here are the metric results:

Metric	MAC000246	MAC003718	MAC003463
MSE	0.438	0.180	0.289
R2	-0.0148	0.6	0.23
MDA	0.556	0.573	0.524
MAE	0.435	0.293	0.376

Table 14: GPR Reptile metric results

One of the first observation that can be done, is that via the Reptile algorithm, GPR is able to generalize and adapt rather well. However, there are still some households for which GPR adapts worse, such as MAC000246. In fact, GPR Reptile fails to match sudden energy peaks, like in the case of MAC000246, where it tends to overshoot real values, or in case of MAC003463, where it tends to constantly underestimate energy consumption. In addition to that, due to the fact that the learning happens on samples coming from various households a certain bias is present after the end

of meta-learning training phase, which once again can be seen on the example of MAC000246 and MAC003463, where low-level energy consumption predictions over- and undershoot respectively. Thus, one can also deduct that low-level energy consumption predictions are modelled worse, than in the case of standard learning. Overall, one can see, that indeed GPR Reptile meta-learning algorithm is viable, despite it giving worse results, than standard training, which was expected.

After having analyzed GPR Reptile meta-learning algorithm, LSTM fitting was carried out. Here are the results for LSTM:

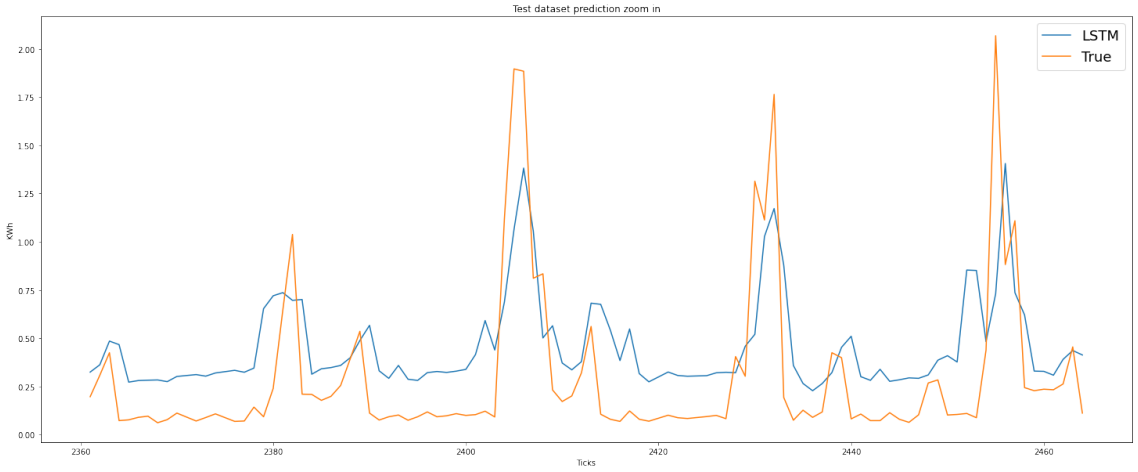

Figure 32: LSTM Reptile predictions for MAC000246

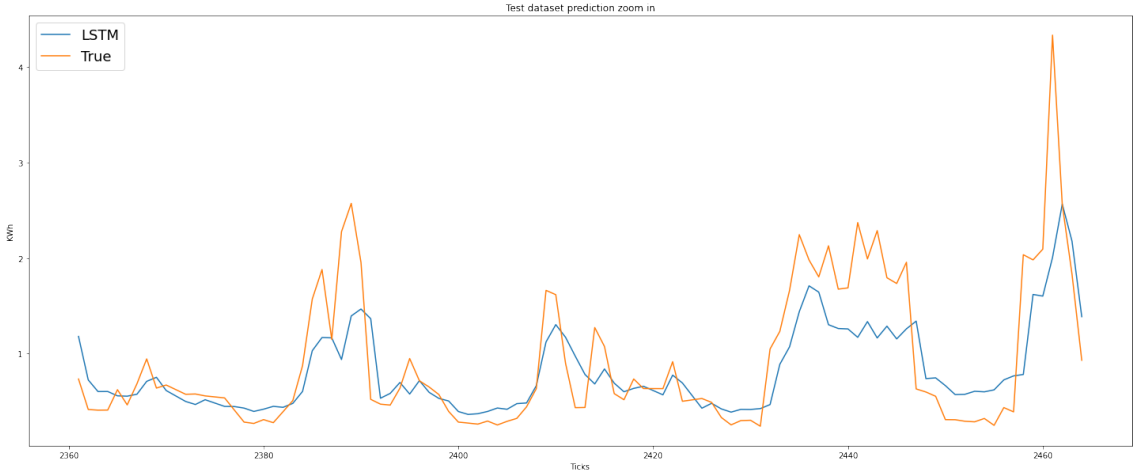

Figure 33: LSTM Reptile predictions for MAC003718

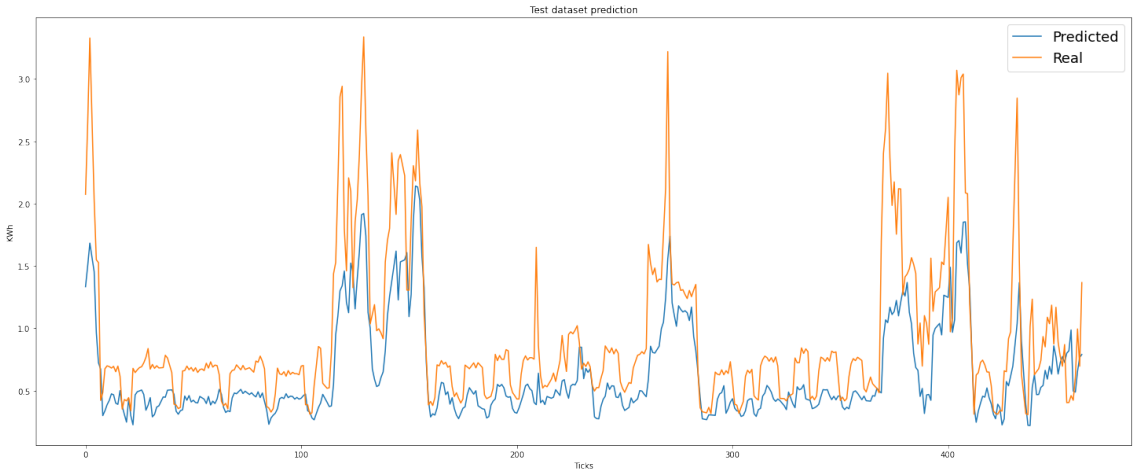

Figure 34: LSTM Reptile predictions for MAC003463

Metric	MAC000246	MAC003718	MAC003463
MSE	0.230	0.182	0.209
R2	0.466	0.596	0.443
MDA	0.522	0.569	0.501
MAE	0.341	0.262	0.325

Table 15: LSTM Reptile metric results

As mentioned in the discussion on LSTM in previous results section, thanks to a higher amount of data, the fit quality has greatly improved, which can be seen from decreased MSE and increased R2 score, despite a decrease in MAE, which due to the average bias in low-level energy consumption estimations. However, as for GPR, LSTM suffers from the same problem coming from the learned average bias, coming from households' energy consumption patterns. On the other hand, it seems that LSTM is able to quicker adapt itself for query points that are needed for adaptation for a particular household. Overall, it seems that LSTM Reptile is a better alternative to GPR Reptile, when it comes to energy consumption prediction, as the error is lower, and on top of that due to the fact that LSTM Reptile needs less query data to adapt itself for new households, which can be deduced from the fact, that it adapts itself quicker than GPR Reptile.

9.3 MAML

The second meta-learning algorithm that was tested was the MAML. The results for MAML GPR are the following:

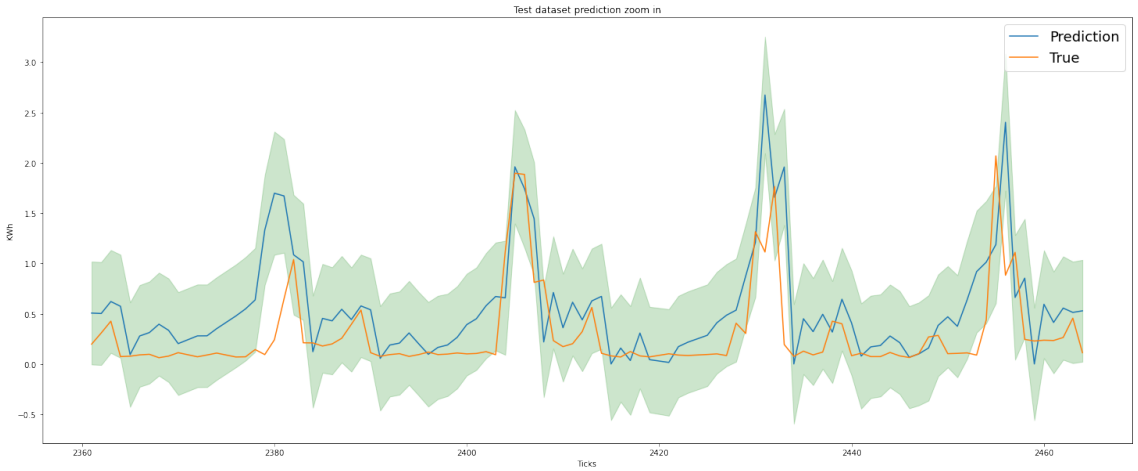

Figure 35: GPR MAML predictions for MAC000246

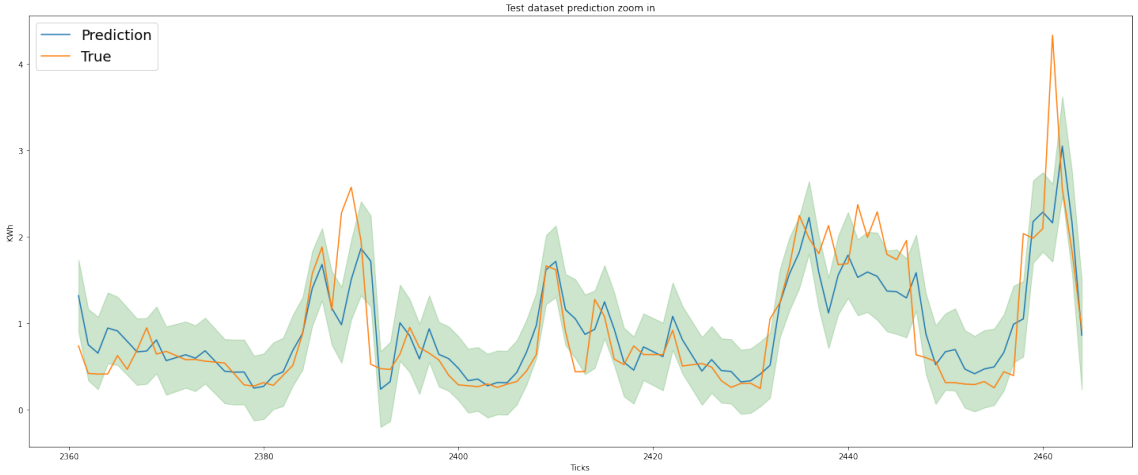

Figure 36: GPR MAML predictions for MAC003718

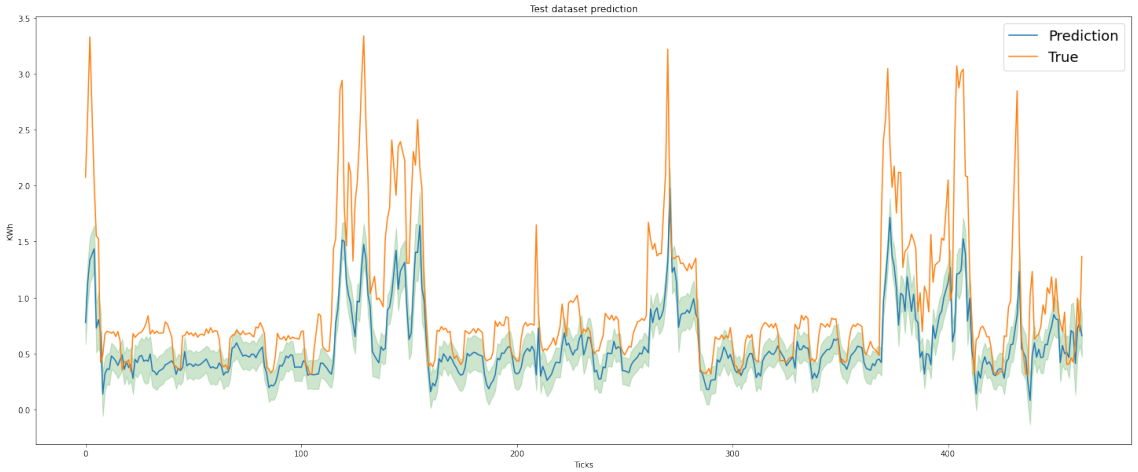

Figure 37: GPR MAML predictions for MAC003463

Here are the metric results for GPR MAML meta-learning algorithm:

Metric	MAC000246	MAC003718	MAC003463
MSE	0.347	0.180	0.289
R2	0.196	0.6	0.23
MDA	0.533	0.573	0.524
MAE	0.390	0.293	0.376

Table 16: GPR Reptile metric results

One can see, that in some cases, MAML performs better than Reptile, which can be due to the fact that GPR optimization can often fall into local minimums, and the fact that in MAML one obtains a better generalization which leads to better results, like in the case of MAC000246 household. However, due to the fact that MAML adapts slower to new tasks, the performance is close to Reptile's one, at least for GPR algorithm.

MAML producing rather same results for GPR, gave completely different results for LSTM. Here are the resulting plots:

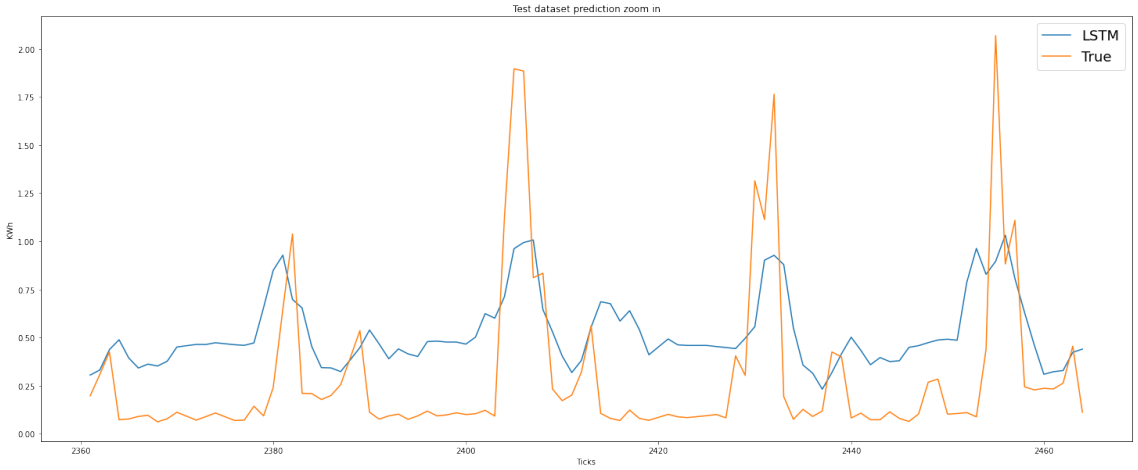

Figure 38: LSTM MAML predictions for MAC000246

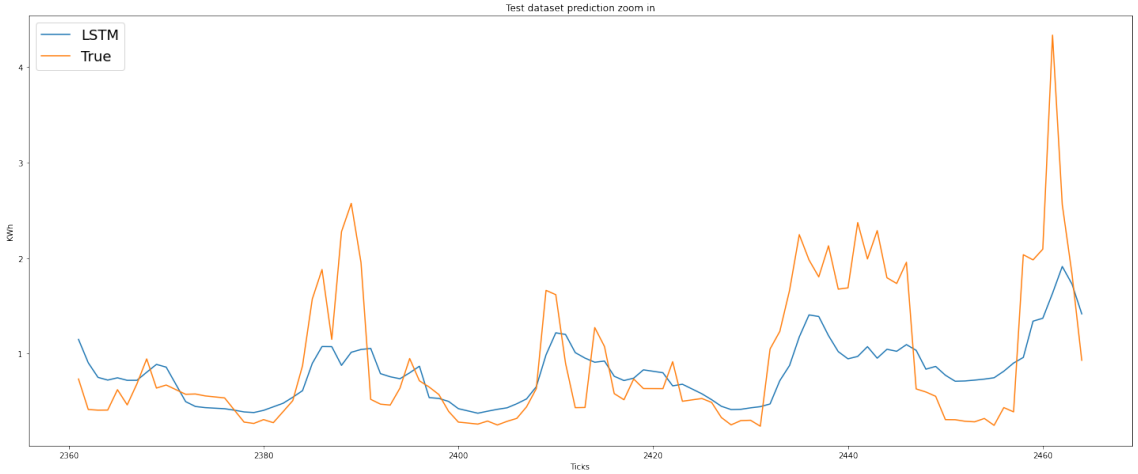

Figure 39: LSTM MAML predictions for MAC003718

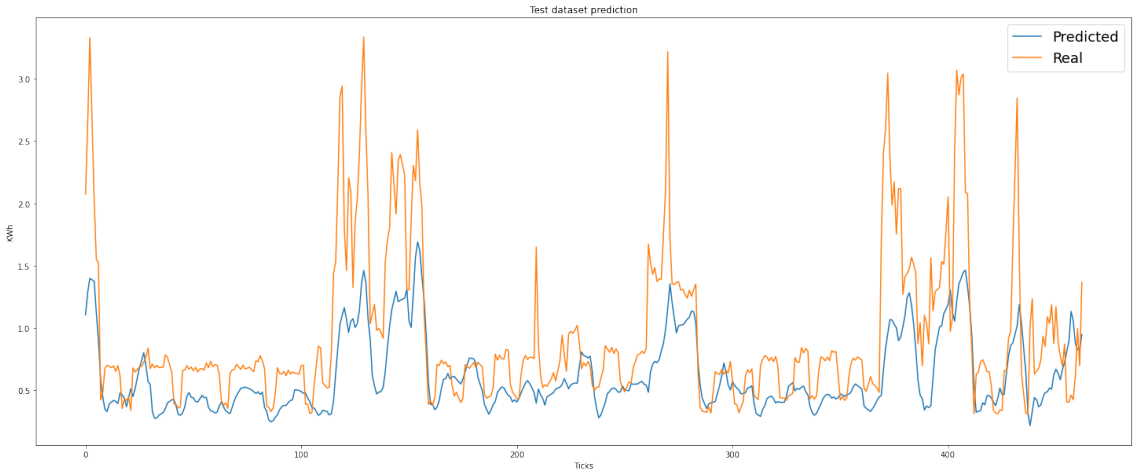

Figure 40: LSTM MAML predictions for MAC003463

Metric	MAC000246	MAC003718	MAC003463
MSE	0.283	0.240	0.281
R2	0.344	0.468	0.251
MDA	0.516	0.615	0.551
MAE	0.409	0.305	0.362

Table 17: LSTM MAML metric results

One can observe that MAML fails to adapt quick enough in LSTM MAML configuration. This was discussed in the description of algorithms, and these results prove the theory. Indeed, one can observe a slightly higher overall average prediction bias, and less spiky peaks in the predictions, which shows that the model, adapts slower for new data. Thus, one can conclude, that in general MAML would be a better choice than Reptile in cases, where it is known that the data would be rather homogeneous with small differences between tasks, unlike in the case of energy consumption prediction.

9.4 LSTM Meta-Learner

As mentioned before, this meta-learning algorithm was not optimized, due to hardware limitations and due to a huge amount of architectures' possibilities. Experiments were tried out with LSTM Learner, that was the base for previous meta-learning algorithms, and the following results were obtained:

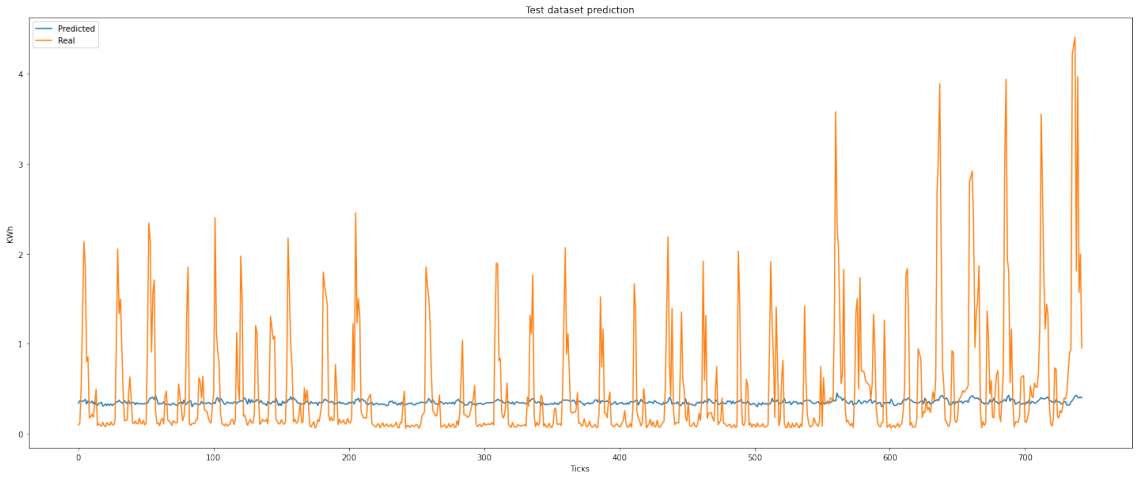

Figure 41: LSTM Meta-learner predictions for MAC000246

Indeed it can be seen, that the results are unsatisfactory. Perhaps, by increasing the number of hidden layers in meta-learner LSTM would improve the results, however it is too computationally heavy in memory and corresponding experiments could not have been carried out.

10 Possible improvements

The models that were used in the scope of this project have shown some good results, and proved that meta-learning can be used for energy consumption prediction. However, there are a few improvements that could have been done to this project and general analysis.

First of all, as it was seen from Section 6.2, NuSVR and XGBoost were in fact two algorithms that demonstrated best results on single household energy consumption prediction. Thus, it would be interesting to adapt meta-learning algorithms described above to be usable with NuSVR and XGBoost.

Secondly, as it was mentioned in the previous section LSTM Meta-learner provides rather poor results, but has potential to outperform other meta-learning algorithms. In fact, the meta-learner that was used had a rather small amount of hidden layers, which could not allow it to grasp all the trends of time series. On top of that, LSTM learner may not be the best learner to be fed into LSTM meta-learner. Not only it becomes incredibly expensive in terms of hardware usage, but also it may results into a too high amount of LSTM layers, which may lead to overfitting.

Finally, last aspect that could have been added into analysis is the performance assessment of standard machine learning algorithms on the whole dataset, which was not split into different weekdays. Moreover, in the scope of this project, ACORN data was not used, due to its overall complexity. By using this data as input and by conducting analysis on whole dataset, one would have obtained a more general analysis of the performance of models, and thus a more realistic assessment of their applicability for the task of energy consumption prediction.

11 Conclusion

As it was stated in introduction, the goal of this project was to implement and assess meta-learning algorithms for short-term energy consumption prediction predictions.

In scope of this project, two meta-learning algorithms were fully implemented and tested: MAML and Reptile. Both of these algorithms proved their viability for household energy consumption

prediction, with Reptile being slighly better for LSTM algorithm. Indeed by adapting LSTM and GPR algorithms, reasonable results were obtained for three different households. The total prediction error was higher, than in the case of fitting on only one household, in case of GPR, however the pattern was still discovered by the algorithm. In case of LSTM, Reptile even allowed to boost the performance of the algorithm, in comparison with the fit on the single household. As for MAML, the general error was higher, however, as in GRP, the consumption pattern was still discovered. In addition to that, fields of application of these two meta-learning algorithms were discussed, with Reptile being more adapted for heterogeneous tasks' data prediction (such as energy consumption prediciton), while MAML was discovered to perform better with homogeneous data (such as SP500 stock price prediction), which as mentioned just above was perfectly visible on LSTM algorithm.

Overall, meta-learning techniques can be applied for household energy consumption prediction.

References

[1] *Smart Meters in London dataset.* URL: `https://www.kaggle.com/jeanmidev/smart-meters-in-london`.

[2] *Meta-Learning Algorithms description.* URL: `https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html`.

12 Appendix

Code used in this project can be found in the following GitHub repository: **Link to GitHub**