AMS 559 Project 2

Name: Viraj Kamat
SBU-ID: 112818603

Objective

Given a dataset containing energy usage of different homes and the prediction of their load demand using machine learning algorithms we wish to implement different energy provisioning algorithms and compare their performance through an objective cost function.

The Dataset and predictions

The dataset contains energy demand for 3 different houses at regular time intervals. This dataset was cleaned formatted and resampled to produce energy demand at every 30 minute intervals.

In order to perform our energy provision for each of the houses I first fetched their predicted energy demand using the machine learning models I applied in assignment 1. Based on the models that had the lowest mean absolute error, the energy demand predictions from the following machine learning models were chosen :

1. Linear Regression
2. Long Short Term Memory Neural Model
3. Extreme Gradient Boost

The predictions from each of these models was used to provision energy at 30 minute intervals for each house. As per the assignment requirement only dataset from the 1st two weeks of November was taken giving us 672 time steps was taken.

The objective function

While provisioning load demand for each timestep, there was a need to minimize an objective function given as follows :

$$\sum_{t=1}^{T} p(t)x(t) + a * \max\{0, y(t) - x(t)\} + b|x(t) - x(t-1)|$$

Where
p = powercost/timestep
a/b parameters

The second and third term is the penalty and switching cost respectively. Penalty if the provisioning is less than the demand, and switching cost as we do not want to switch the energy provisioning vastly at each time step.

I aim to minimize this objective function that yields us the total cost of the energy provisioned. I used different algorithms as mentioned below and observed how they performed relative to the cost function.

Implementation of algorithms for energy predictions

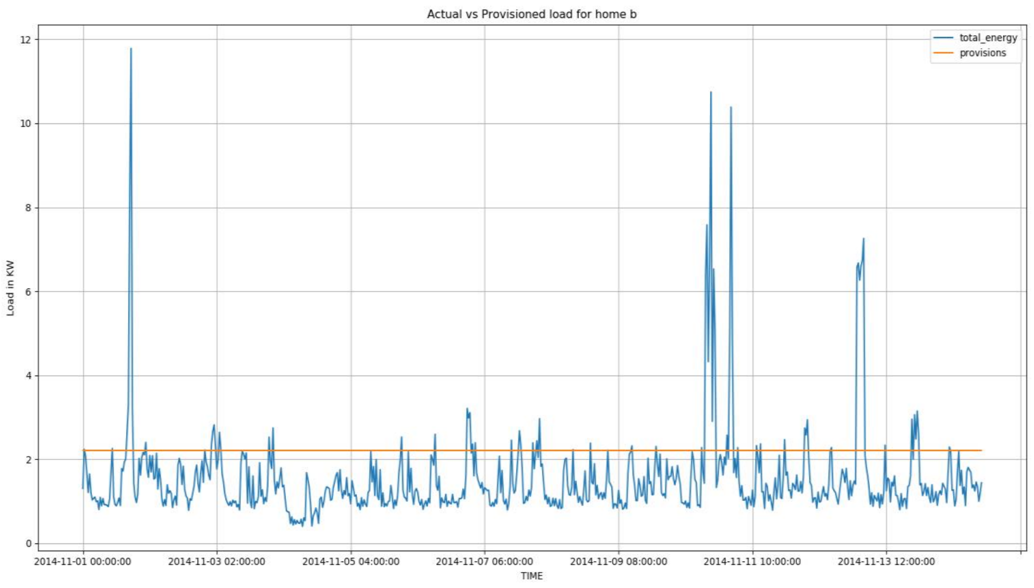
1. Static offline and Dynamic offline solution :-

The static offline optimal solution was straight forward. I used the true value of the load demand at each time step and using the python cvxpy module minimized the objective function that yielded the lowest cost. Note that in the static optimal solution we only provision a single value for each timestep using true load demand. The final cost was then calculated based on the objective function.

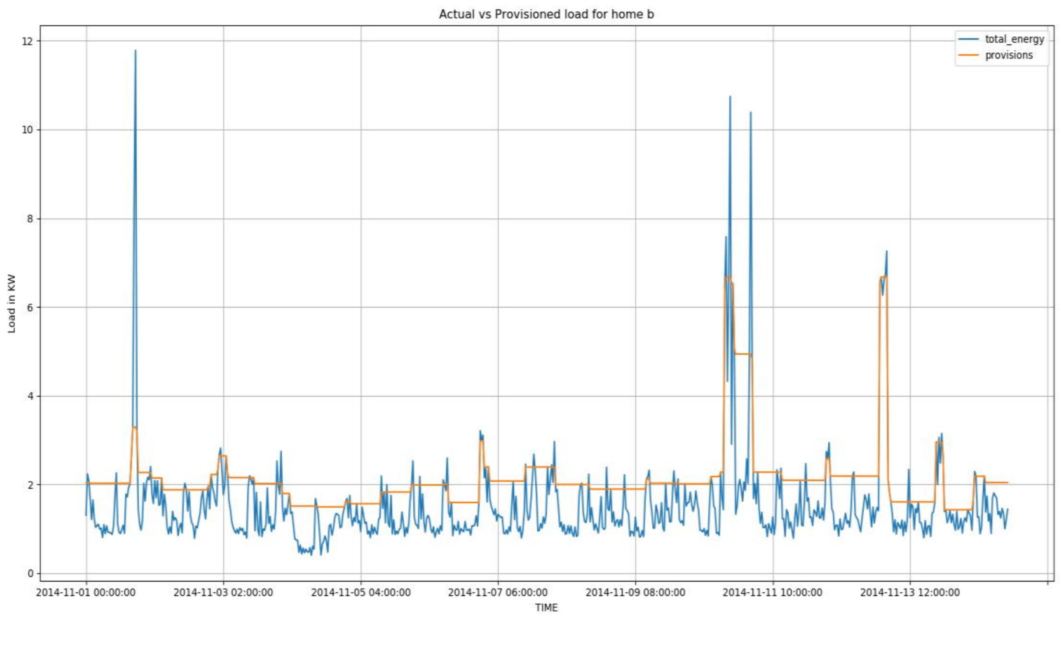
The dynamic offline solution also uses true load demand to perform energy predictions however at each timestep rather than a single value. In this case I also used the cvxpy python module to minimize the objective function.

In each case the cost of electricity was kept at $0.4 and the a/b parameters each at 4 respectively.

The below graphs show the energy provisioned vs the true load demand at each timestep :



Static Optimal Solution for House B using cvxpy



Dynamic Optimal Solution for House B using the cvxpy

The table below demonstrates the cost of each house using the online static and the online dynamic solution

House	Static Offline Cost
B	1018.476328
C	1355.830744
F	183137.4667

House	Dynamic Offline Cost
B	838.9757
C	1133.962
F	147379.4

As we can see above the dynamic offline optimal solution cost is lower and hence better

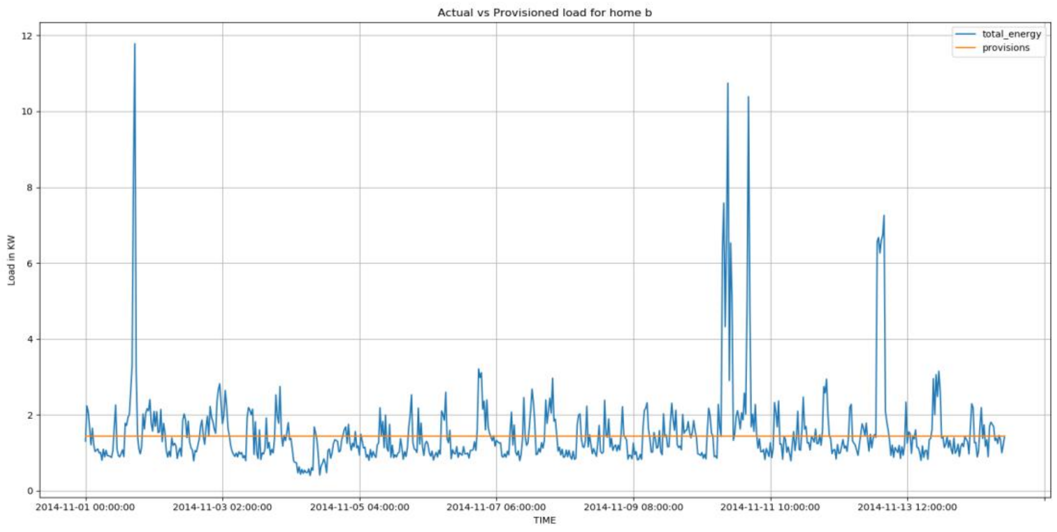
2. Online gradient Descent :-

In this algorithm we aim to mimic the static optimal solution, i,e provide a single provisioning for each timestep. We also use true values of load demand in this case.

The Online gradient descent algorithm works by continually updating the value of the provisioning for each timestep with the help of the derivative of the cost function with respect to the provisioning at a timestep. Note that the following is the case when we use it to update our static optimal solution at each time step .

1. If the current provisioning is less than the true demand :
   a. If the provisioning at the current timestep is less than the previous timestep increase the provisioning by a+b
   b. Else increase the provisioning by b
2. If the current provisioning is greater than the true demand
   a. If the current provisioning is greater than the previous provisioning reduce it by a+b
   b. Else reduce it by b

The plots below show the provisioning done with the help of Online Gradient Descent :



Static optimal solution for house b using online gradient descent

Note that the change in the provisioning at each timestep is governed by the step size parameter which is given as follows :

X(t) = X(t-1) + (step_size * derivative of cost function)

Where x(t) is the provisioning at timestep t and step size is given by 1/square-root(t)

The cost of electricity was kept at $0.4 and the a/b parameters each at 4 respectively.

I varied different step sizes and attempted to find the best cost (lowest) with the help of Online Gradient Descent, the following are the results :

Step-size @ timestep t	House	Cost
1/sqrt(t)	B	1266.235186
1/sqrt(t)	C	1355.886565
1/sqrt(t)	F	474306.5085
2/sqrt(t)	B	1310.875115
2/sqrt(t)	C	1486.262107
2/sqrt(t)	F	448178.0126
4/sqrt(t)	B	1018.756785
4/sqrt(t)	C	1531.693812

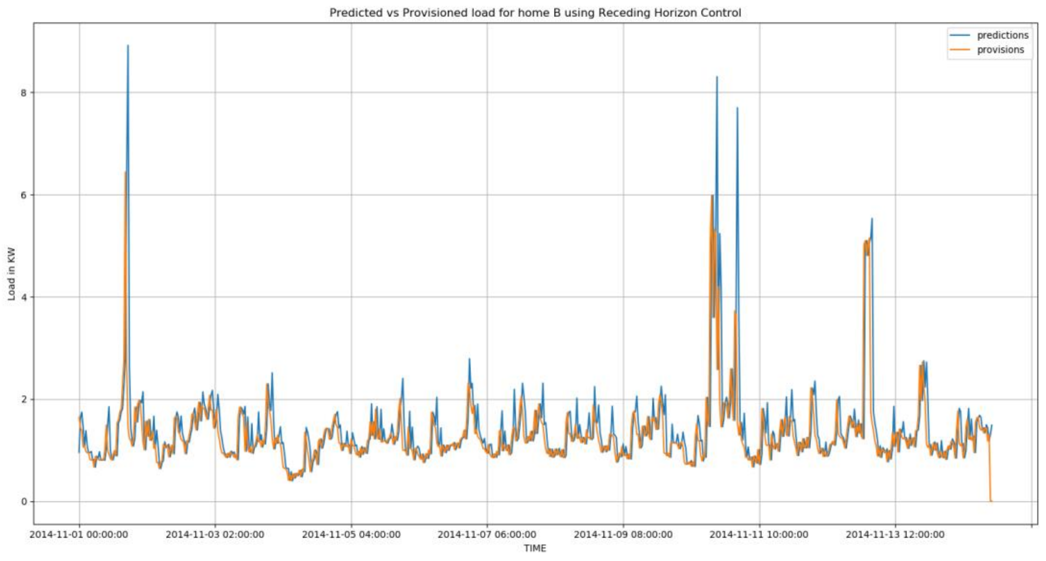
4/sqrt(t)	F	439854.2291
8/sqrt(t)	B	1195.024431
8/sqrt(t)	C	1359.944295
8/sqrt(t)	F	436552.4456
16/sqrt(t)	B	7334.941825
16/sqrt(t)	C	1359.067382
16s/sqrt(t)	F	446761.5991

As can be seen above the step-size 8/square-root(t) provided the lowest cost for the Online Gradient Descent Algorithm.

3. Receding Horizon control :-

The Receding Horizon Control Algorithm aims to perform predictions with the help of a window, i.e perform predictions at each time-step for a window of size w and pick the provisioning for that timestep. Note that the objective function minimized is the same as used in dynamic optimal solution however we use predictions from machine learning models to provide the energy demand at a timestep. We are in-fact mimicking the dynamic online optimal solution.

Below is a prediction with a default window size of w and with predictions from the linear regression model for our load demand. Note that the cost of electricity was kept at $0.4 and the a/b parameters each at 4 respectively.



Prediction for House B using R.H.C and predictions from the linear regression model

As a part of experimentation to find out the optimal value of the window size by keeping all other values constant I provisioned the load for each timestep and varied the window size parameter, below is the chart of those tests. This was also done from the prediction models as stated above (Linear Regression, Xgboost, L.S.T.M)

As usual lower the cost the better :

Model	**House**	**Cost**	**Window-size**
b	XGBoost	1898.108	3
b	XGBoost	1962.862	5
b	XGBoost	1980.918	7
b	XGBoost	1989.242	9
b	XGBoost	1956.179	11
b	XGBoost	1981.549	13
b	XGBoost	1991.498	15
b	L.S.T.M	1840.842	3
b	L.S.T.M	1837.143	5
b	L.S.T.M	1836.114	7
b	L.S.T.M	1835.172	9
b	L.S.T.M	1839.415	11
b	L.S.T.M	1837.579	13
b	L.S.T.M	1837.11	15
b	Linear-Regression	1852.655	3
b	Linear-Regression	1899.262	5
b	Linear-Regression	1911.53	7
b	Linear-Regression	1926.855	9
b	Linear-Regression	1895.575	11
b	Linear-Regression	1894.785	13
b	Linear-Regression	1911.751	15
c	XGBoost	2416.77	3
c	XGBoost	2458.867	5
c	XGBoost	2504.504	7
c	XGBoost	2525.71	9
c	XGBoost	2515.79	11
c	XGBoost	2525.546	13
c	XGBoost	2522.42	15
c	L.S.T.M	2744.706	3
c	L.S.T.M	2731.035	5
c	L.S.T.M	2722.93	7
c	L.S.T.M	2720.644	9
c	L.S.T.M	2703.73	11
c	L.S.T.M	2702.858	13
c	L.S.T.M	2706.174	15
c	Linear-Regression	2617.159	3
c	Linear-Regression	2641.609	5
c	Linear-Regression	2714.796	7
c	Linear-Regression	2735.713	9
c	Linear-Regression	2636.866	11
c	Linear-Regression	2659.269	13
c	Linear-Regression	2676.228	15
f	XGBoost	307711.5	3

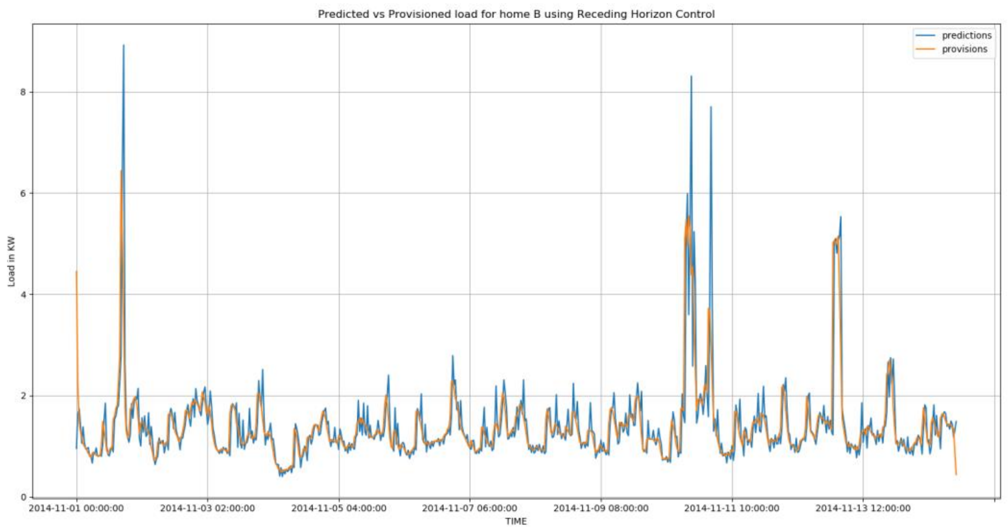
f	XGBoost	313103.2	5
f	XGBoost	314194.1	7
f	XGBoost	314990.7	9
f	XGBoost	327501.1	11

As can be seen above increasing the window size does help in lowering the cost of energy provisioned by Receding Horizon Control. However, its benefits starts to diminish and even worsens if the window size is increased to too large a value, an optimum value is between 5 and 7 .

4. Commitment horizon control :-

The Commitment Horizon Control Algorithm also mimics the dynamic optimal solution at each timestep using predictions from machine learning models as the inputs for the load demand, it also uses window ranges to predict load demand for a certain range of timesteps and we then pick the provisioning from that window for a timestep t, what is different however is the introduction of a new parameter called the commitment level, what this essentially does is decide how many predictions to be used at a certain timestep to gauge the energy to be provisioned at a timestep i.e take an average of the values at a timestep governed by a commitment level.

We also aim to minimize the cost function with the help of cvxpy. Below is the provisioning for house B using CHC with a commitment level of 3 and window size of 5 :



Prediction for House B using c.H.C and predictions from the linear regression model

As a part of testing the Commitment Horizon control algorithm I varied the commitment level while keeping the window size a constant 9. Below are the results of the cost for different commitment levels:

House	**Model**	**Cost**	**Commitement-level**	**Window**
b	Linear-regression	1257.838	7	9
b	XGBoost	1307.676	7	9
b	Linear-regression	1340.546	5	9
b	XGBoost	1389.548	5	9
b	Linear-regression	1513.158	3	9
b	XGBoost	1552.16	3	9
c	Linear-regression	1673.72	7	9
c	XGBoost	1757.416	7	9
c	Linear-regression	1789.406	5	9
b	L.S.T.M	1807.644	7	9
b	L.S.T.M	1810.686	5	9
b	L.S.T.M	1820.676	3	9
c	XGBoost	1847.503	5	9
c	XGBoost	2020.402	3	9
c	Linear-regression	2054.642	3	9
c	L.S.T.M	2611.192	7	9
c	L.S.T.M	2634.158	5	9
c	L.S.T.M	2673.511	3	9
f	Linear-regression	197719.8	7	9
f	Linear-regression	208616.3	5	9
f	XGBoost	215457.4	7	9
f	XGBoost	229406.8	5	9
f	Linear-regression	233732.9	3	9
f	XGBoost	257819.6	3	9
f	L.S.T.M	421671.9	7	9
f	L.S.T.M	422565.4	5	9
f	L.S.T.M	423872	3	9

What can be noted above is that a window size of 9 with a commitment level of 7 provides the lowest cost. Of all the algorithms explored so far C.H.C with window size 9 and commitment level of 7 with linear regression as the input for our predictions of load demand provides the best model for provisioning.

The cost function and comparing our results

Let us now compare the offline static optimal solution and dynamic optimal solution to our best OGD, R.H.C, C.H.C models :

House	Static Optimal Solution Cost	Dynamic Optimal Solution Cost	O.G.D Cost	R.H.C Cost	C.H.C Cost
B	1018.476328	838.9757	1018.756785	1836.114	1257.838
B	1018.476328	838.9757	1195.024431	1837.143	1307.676
C	1355.830744	1133.962	1531.693812	2416.77	1673.72
C	1355.830744	1133.962	1359.944295	2458.867	1757.416
F	183137.4667	147379.4	436552.4456	307711.5	197719.8
F	183137.4667	147379.4	439854.2291	313103.2	208616.3

As can be seen above the offline dynamic optimal solution produces the lowest cost, which would make sense as it works with true load demands rather than the predictions. Online gradient descent also works well but it’s a static optimal solution and not effective at provisioning where the load demand is very high as it gets penalized by minor switching costs.

Commitment Horizon Control with predictions from the Linear Regression model provides good lowering of costs for both high energy and moderate energy load demand houses.

Effect of varying the a & b parameters

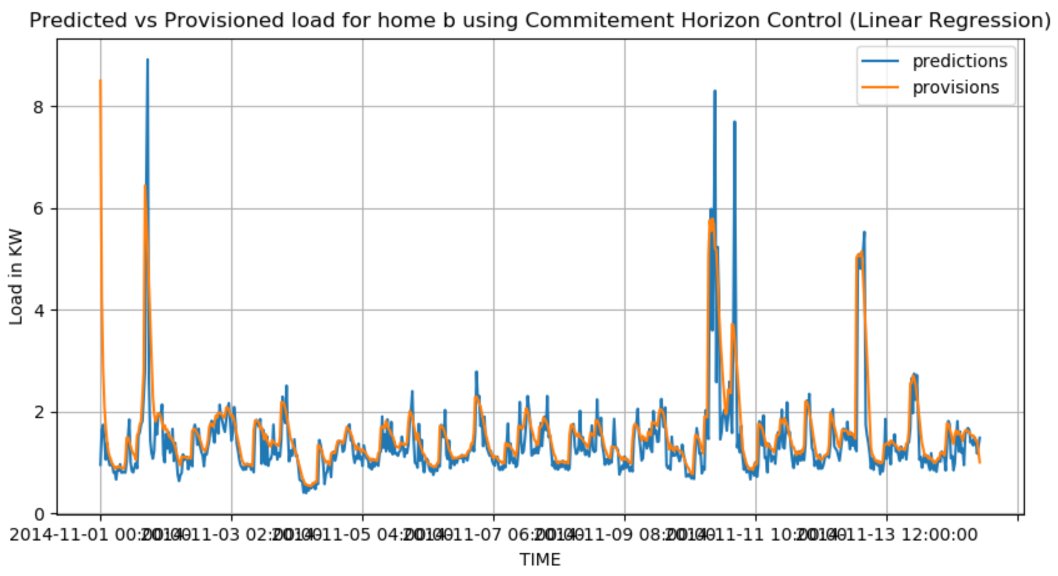
From the above observations I noted that Commitment Horizon Control indeed provides a good provisioning of power based on prediction inputs from the Linear Regression model. To further test the combination of these two algorithms   I varied **a** and **b**; the penalty for lower than required demand and the penalty for switching cost. Below are the observations I made by varying these two parameters :

House	**Model**	**Cost**	**a**	**b**
B	Linear Regression	868.966252	2	2
B	Linear Regression	634.601159	1	1
B	Linear Regression	1121.10514	4	2
B	Linear Regression	1142.62674	4	1
B	Linear Regression	764.30884	1	4
B	Linear Regression	964.827647	2	4
C	Linear Regression	1165.12921	2	2
C	Linear Regression	871.093856	1	1
C	Linear Regression	1469.45576	4	2
C	Linear Regression	1523.18187	4	1

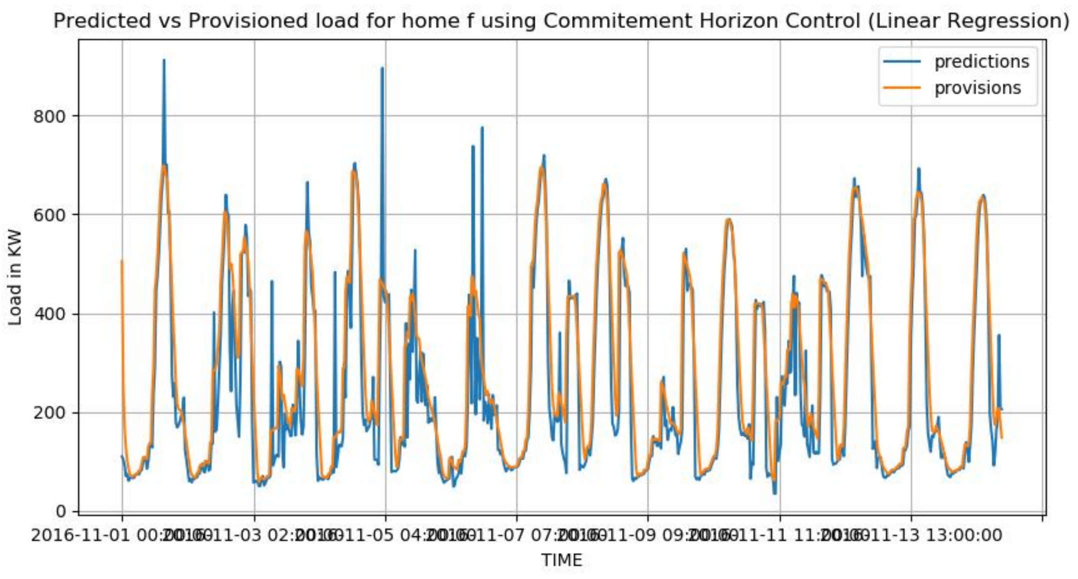
C	Linear Regression	1050.23892	1	4
C	Linear Regression	1279.50486	2	4
F	Linear Regression	142483.286	2	2
F	Linear Regression	106989.833	1	1
F	Linear Regression	171987.259	4	2
F	Linear Regression	171324.635	4	1
F	Linear Regression	156932.843	1	4
F	Linear Regression	176121.892	2	4

What is noted above is that the penalty on switching or not meeting the correct load demand directly affects the outcome of the cost, also any high switching would imply a strong penalty on the cost; too low and we let go with varied fluctuation in provisioning the energy, too high and tightly restrict our provisioning. We can keep the **a** and **b** to a range of 2 and 4 to accurately provision the cost for the models.

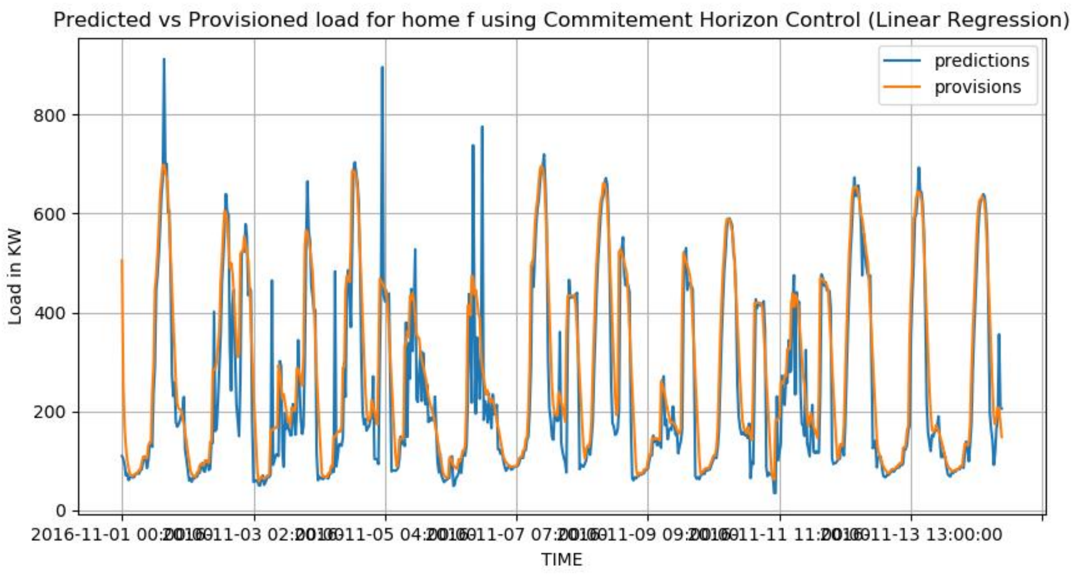
Below are the prediction for each of the houses B, C, F with predictions from the Linear Regression Model with Commitment Horizon control with a and b parameters at 4,4 respectively.



Provisioning for House B with Commitment Horizon Control using Linear Regression



Provisioning for House C with Commitment Horizon Control using Linear Regression



Provisioning for House F with Commitment Horizon Control using Linear Regression

Randomized and Deterministic approach for algorithm selection

In order to begin with the Randomized and Deterministic algorithms I used the weighted majority approach to perform our energy provisioning. The weight for each timestep was computed using a weighted majority cost and then optimized successively for each timestep, in total I computed weights for each of the provisioning algorithms (OGD , RHC, CHC, etc) for each timestep and adjusted their relative values according to the formula below :

$$w_i(t) = \eta \frac{\frac{1}{C_i(t-1)}}{\sum_j \frac{1}{C_j(t-1)}} + (1-\eta) w_i(t-1)$$

Where C(i) is the cost for the provisioning algorithm at timestep t. i is the algorithm for which the weight is being adjusted and j are all the provisioning algorithms.

We use a step-size η is set as 1/square-root(t).

The following is algorithm used to fetch the weights for each provisioning method:

1. Set the default weights for each provisioning algorithm to .25
2. Compute the provision for each timestep t with the provisioning algorithm
3. Based on the formula above adjust the weights of the provisioning algorithm
4. Repeat iteratively until the final timestep. Use the final set of weights for provisioning

Once we have the weights for each algorithm the following is the algorithm to compute the Deterministic Provisioning:

1. Compute the provisioning for each timestep t using our provisioning algorithms
2. Multiply the provisioning for each timestep from each algorithm with its respective weight and sum the provisioning values

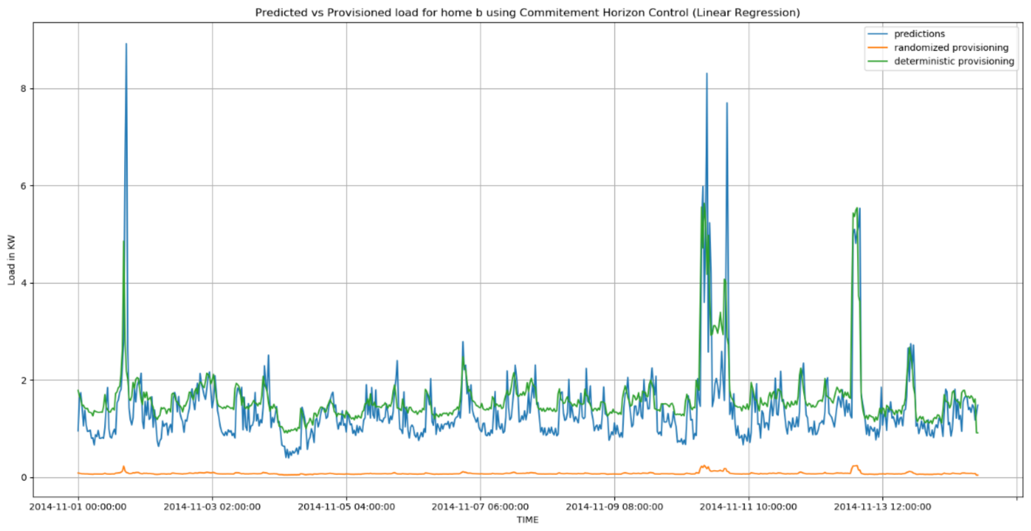
Thus, we are using weighted outputs from each algorithm to provision the energy.

In case of the Randomized Provisioning we multiply the weights for each timestep from output of each of algorithms as well, the difference here being that only the largest value of weight from the provisioning algorithm is chosen to be multiplied with outputs from every provisioning algorithm

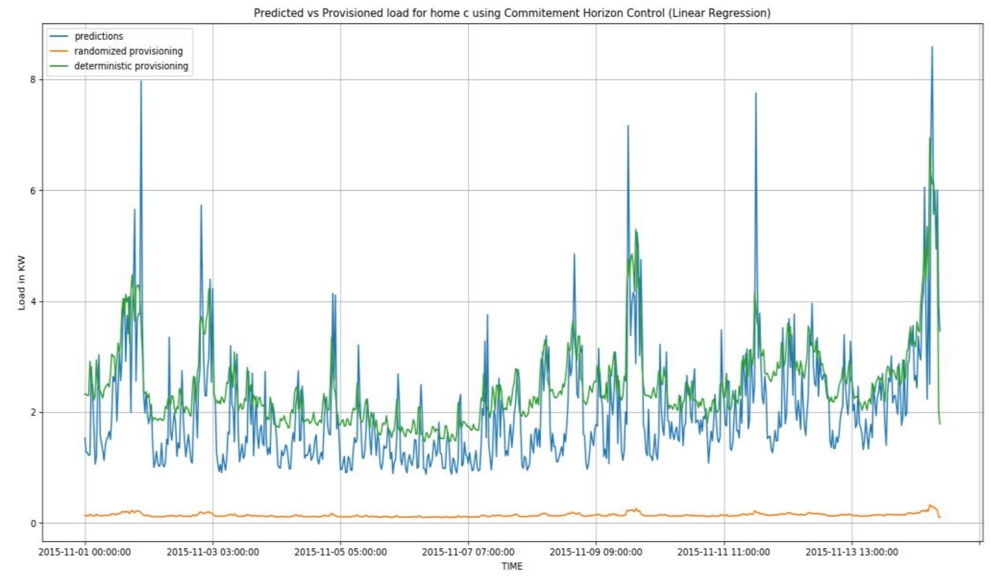
The table below summarized the comparison of provisioning costs using the Randomized and Deterministic provisioning algorithms :

House	**Randomized Provisioning Cost**	**Deterministic Provisioning Cost**
B	232584.5	1256.16
C	5326.77880341738	1702.867
F	604570.5	232584.5

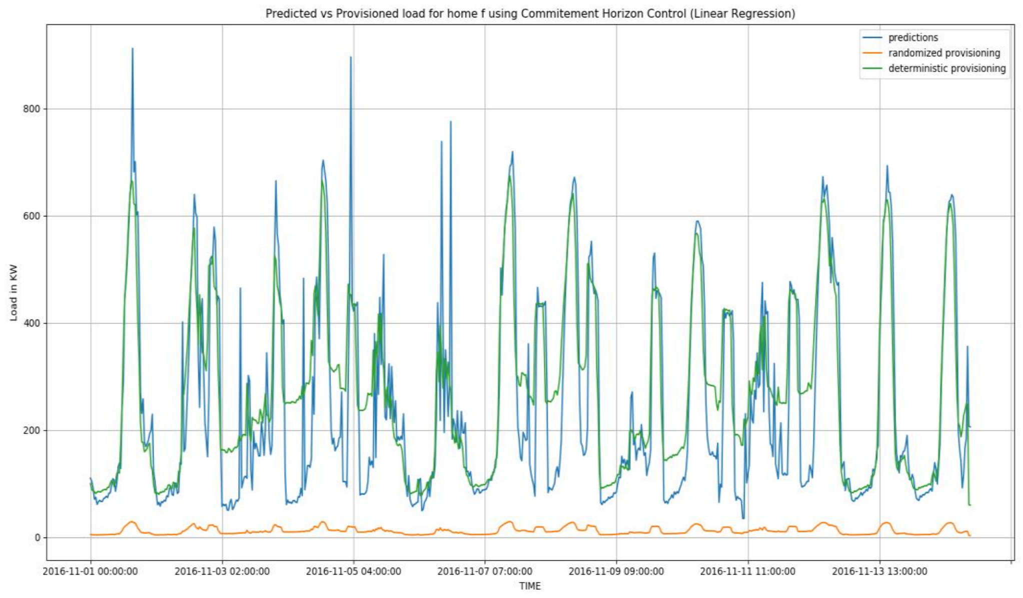
As can be seen the Deterministic Provisioning Algorithm provides a lower cost as compared to the Randomized Provision Algorithm, the plots below display the respective provision:



Randomized vs Deterministic Provisioning for house B (Predictions from Linear Regression)



Randomized vs Deterministic Provisioning for house C (Predictions from Linear Regression)



Randomized vs Deterministic Provisioning for house F (Predictions from Linear Regression)

Online Balanced Descent Algorithm

The Online balanced algorithm draws from two algorithms, online learning algorithms and the Online mirror algorithms. While this was not implemented, I would like to briefly highlight on it.

OBD is another Smooth Convex Optimization algorithm unlike Online gradient descent uses projections onto level sublevel set This sublevel set is used to balance the switching costs and penalty of our gradient descent algorithms.