Machine Learning and Deep Learning Approaches to Day-Ahead Load Forecasting in the East Bay

Cjache Kang, Raguvir Kunani, Skye Zhang

Abstract

We evaluate the efficacy of machine learning and deep learning methods in forecasting day-ahead load in the East Bay. Specifically, we study autoregression, random forest, gradient boosted trees, and long short-term memory models. We find that the random forest model best forecasts day-ahead load, but observe that long short-term memory could outperform random forest with further work.

1 Introduction

Community choice aggregation (CCA) is a program in which local governments can purchase power on behalf of their residents, businesses, and municipal accounts from an alternative supplier while still receiving transmission and distribution service from their existing utility provider. The main benefits of CCA include faster adoption of greener power sources and lower consumer prices for energy.[1] Since energy is regulated on a state-by-state basis, CCA takes on different forms in each region where it is implemented.

In the East Bay, CCA is implemented by East Bay Community Energy (EBCE), a public agency which supplies energy to its customers by procuring energy daily in a day-ahead energy market. In order to know how much energy to procure each day, EBCE must forecast the load demand of its customers. It is critical for the forecast to be as accurate as possible in order for community choice aggregators like EBCE to maintain rates competitive with those of existing utility providers.

This paper develops prediction models to evaluate the efficacy of machine learning and deep learning approaches to forecasting day-ahead load in the East Bay.

2 Data

2.1 Data Description

Hourly load data segmented by "jurisdiction" was provided by EBCE. At the time of writing, EBCE's service area comprises 12 jurisdictions: Albany, Berkeley, Dublin, Emeryville, Fremont, Hayward, Livermore, Oakland, Piedmont, San Leandro, Unincorporated Alameda, and Union City. Figures 1 and 2 (known as load profiles) show how load varies by jurisdiction.

[1] https://www.epa.gov/greenpower/community-choice-aggregation

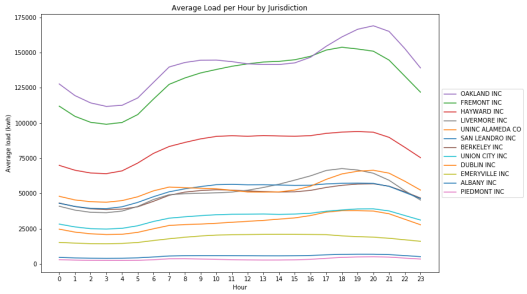


Figure 1: Hourly Load Profiles

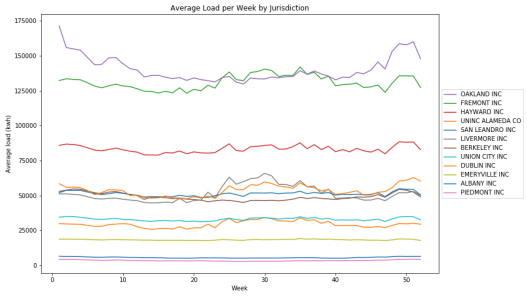


Figure 2: Weekly Load Profiles

From the load profiles, it is clear that each jurisdiction has different load patterns. (The load profiles may appear to indicate that jurisdictions like Piedmont have near zero load. This is not true; it only appears that way because relative to jurisdictions like Oakland, Piedmont's load is insignificant.)

In addition to hourly load data, EBCE provided hourly weather data from 3 weather stations: Oakland, Livermore, Hayward. We map jurisdictions to their closest weather station to obtain weather data for all the jurisdictions. (e.g. Berkeley's weather is approximated data from the Oakland weather station).

After some data exploration, we bring in 2 more datasets: daily sunrise/sunset times and a list of important dates as defined by EBCE.

All the datasets span 2015-2019, but due to issues with 2015 load data we ignore all data from 2015.

2.2 Data Exploration

A natural question is whether load differs by day of the week and by month. Figure 3 reveals a clear distinction between the distribution of weekday load and the distribution weekend load.

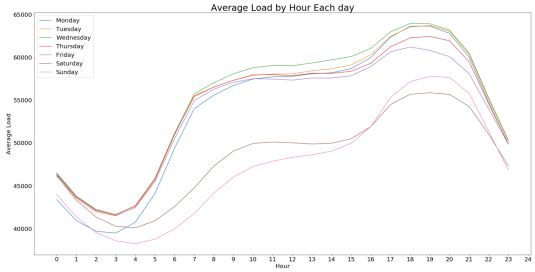


Figure 3: Distribution of Load by Day of Week

Figure 4 shows that the distribution of load in a day is bimodal in non-summer months, but the morning peak is less pronounced – or even non-existent – in the summer months (May, June, July).

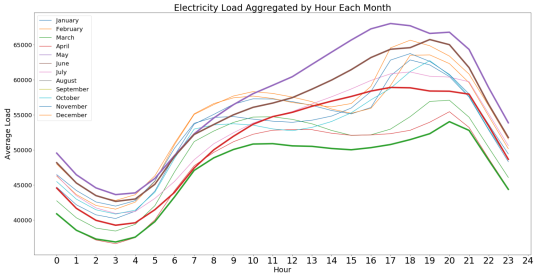


Figure 4: Distribution of Load by Month

Another natural question is how load varies with temperature. Figure 5 illustrates how temperature is related to load in Livermore. The relationship between temperature and load is different in each jurisdiction, but for the sake of brevity we only include Livermore in this paper. Plots for all jurisdictions can be found here.

Figure 5 indicates potential seasonality in load, with higher load in colder and warmer temperatures (winter and summer). We try to capture seasonality by assigning each month a season using different strategies (Figure 6), but ultimately the season feature was not useful in the final forecasting models.

In the last phase of data exploration, we visualize the relationship between load and previous values of load. To do this, we compute autocorrelation of load with lagged values of load. Autocorrelation is defined as

$$A_k = \text{Corr}(X_t, X_{t-k})$$

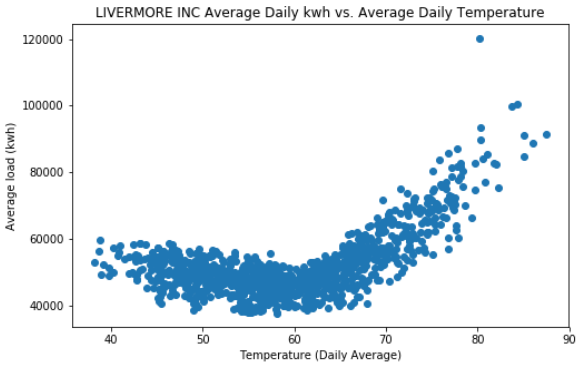
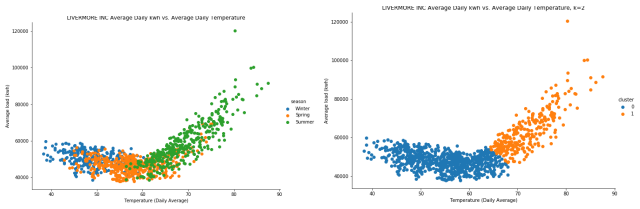


Figure 5: Scatterplot of Average Load vs. Average Temperature. Each dot represents a day between 2016 and 2018.



(a) Manual assignment **(b)** k-means assignment

Figure 6: Each month is assigned to a season based on the average temperature in that month (left) and using k-means clustering with $k = 2$ (right).

where X_t represents the original load time series, X_{t-k} represents the load time series lagged (delayed) by k hours, and Corr represents the Pearson correlation coefficient. Figure 7 reveals that load at any given time is most correlated with load in the previous 2 hours and load 22-24 hours prior.

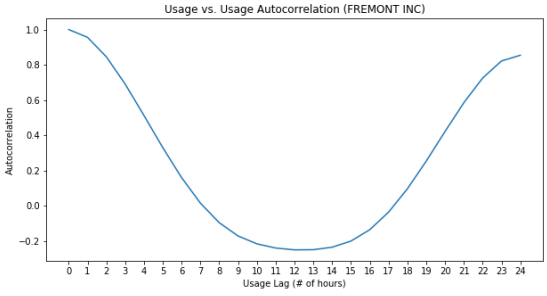


Figure 7: Load autocorrelation for Fremont

3 Models

We use 4 models to forecast day-ahead load: 1 traditional time-series model, 2 ML models, and 1 deep learning model. In all models, we forecast load on an hourly basis – meaning we predict one value for load demand each hour.

Before detailing each model, an explanation of what it means to forecast *day-ahead* load is necessary. Due to the mechanics of the day-ahead energy market, energy to be consumed on any given day must be purchased at the beginning of the previous day. Accordingly, the load forecast for any given day must be available at the beginning of the previous day. For example, the forecast for Thursday's load must be available at the beginning of Wednesday. This means Wednesday's data cannot be used when forecasting Thursday's load.

3.1 Autoregression

The first model we study is an autoregression model. Autoregression is a traditional time-series model that uses a linear combination of previous values of load and other covariates to forecast future values of load. A standard autoregression for load forecasting might look like:

$$\text{Load}_T = \sum_{k=1}^{p} \alpha_k \cdot \text{Load}_{T-k} + \beta_k \cdot \text{Temperature}_{T-k}$$

where T represents a particular hour and p is a parameter that is chosen by validation.

Since our prediction task involves forecasting an entire day's load in a day-ahead fashion, we adapt autoregression to:

$$\text{Load}_{T+24+i} = \sum_{k=1}^{p} \alpha_k \cdot \text{Load}_{T-k} + \beta_k \cdot \text{Temperature}_{T-k}$$

where $i \in \{0, 1, \dots, 23\}$ represents which hour of the day currently being forecasted. This equation is evaluated at every $T = 24z$, $z \in \{0, 1, 2, \dots\}$ (i.e. at midnight every day). This setup essentially decomposes the problem of forecasting an entire day's load into 24 parallel subproblems, each forecasting a particular hour's load.

3.2 Random Forest

The second model we study is a random forest model. A random forest is an ensemble model that bags many (weak) decision trees. Each tree in the forest is independent of the others, and the final prediction for the random forest is the average of all the trees' predictions. Input to the random forest model is specified below:

- **Data from Day being Forecasted:** Hour (0-23), Month (1-12), Weekday/Holiday (0-7), Number of Daylight Hours, Minimum Temperature, Maximum Temperature
- **Data from Previous Days:** Load last week at the hour being forecasted, Load 2 days ago at the hour being forecasted, Morning Peak Load of 2 days ago, Evening Peak Load of 2 days ago

The Weekday/Holiday feature encodes Sunday-Saturday as 0-6, and a holiday (any day in the list of important dates) as 7.

3.3 Gradient Boosted Trees

The third model we study is a gradient boosted trees model, another tree-based ensemble model. The main difference between gradient boosted trees and random forest is that gradient boosted trees employs boosting as opposed to bagging. Specifically, each successive tree in a gradient boosted trees model is trained on the residuals of all the prior trees. This means that the trees are not independent from each other. The final prediction of the model is a weighted sum of the trees' outputs. Input to the gradient boosted trees model is identical to the input to the random forest model, with the addition of a new variable that encodes the week of the year as 1-52.

3.4 Long Short-Term Memory (LSTM)

The fourth and final model we study is an LSTM model. An LSTM model is a neural network that is designed to "remember" previous inputs. The main difference between LSTM and the ML models we study is that the architecture of an LSTM makes the model sensitive to the order in which data is fed in. This allows LSTM to learn not only from the data itself, but the *sequence* in which the model sees the data. This property of LSTM makes it well-suited for temporal data.

Our LSTM model processes data in sequences of one week. Each input data point is identical to the gradient boosted trees input, but without the data from previous days (e.g. there is no "load last week" feature).

3.5 Methodology

For all models, we use 2016-2018 data as training data and 2019 data as test data. When training the random forest and gradient boosted models, we utilize a strategy called dynamic training in which we append each day's data from the test set to the training set when that day is finished. Additionally, we model each jurisdiction separately, as our data exploration in Section 2 reveals each jurisdiction has its own load profile and weather patterns.

4 Results

4.1 Performance Metrics

We track 2 performance metrics, defined below. Recall that we forecast load on an hourly basis, so all of our models produce $365 \times 24 = 8760$ predictions for a year.

To simplify the mathematical explanations of the

performance metrics, we define P as a 365×24 matrix containing the hourly load predictions for a year and A as a 365×24 matrix containing the hourly actual load values for a year.

Metric 1: Average Daily Percent Error
This metric captures, on average, the error we can expect our forecast to have on any given day. To compute average daily percent error:

1. Sum each row of P to obtain a 365×1 vector p. Sum each row of A to obtain a 365×1 vector a. The vectors p and a represent the daily predicted and actual values of load, respectively.
2. Compute $e = \frac{|p-a|}{a} \times 100$. The 365×1 vector e represents the absolute percent error of the daily load predictions.
3. Compute the average of the elements in e. This is the average daily percent error.

Metric 2: Maximum Daily Percent Error
This metric provides an approximate upper bound for the error we can expect our forecast to have on any given day. To compute maximum daily percent error, repeat Steps 1 and 2 above and in Step 3 take the *maximum* of the elements in e instead of the average.

4.2 Model Results

Table 1 reports the performance of the 4 models on test data from 2019. These metrics are computed using the predicted and actual load of all 12 jurisdictions in aggregate.

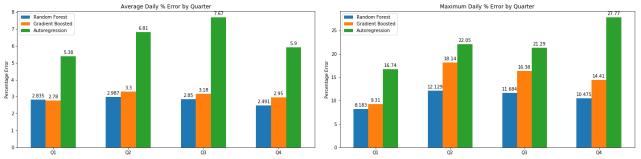
Model	Avg. Daily % Error	Max Daily % Error
Random Forest	2.78%	12.11%
LSTM	2.87%	17.81%
Gradient Boosted Trees	3.06%	18.24%
Autoregression	6.44%	27.77%

Table 1: Performance of all 4 models on 2019 data

Based on these metrics, we conclude that the random forest model is the best-performing model. Although random forest's average daily percent error is only marginally better than that of LSTM and gradient boosted trees, the maximum daily percent error is much better.

4.3 Discussion of Results

A closer look at the performance of the models by quarter reveals that while all models perform worse in the summer months, random forest's performance is the most stable throughout the year (Figure 8). This stability in model performance is likely the reason why random forest is the best model, but we do not yet fully understand why random forest is more stable than the other models.



(a) Average Daily Error **(b)** Maximum Daily Percent Error

Figure 8: Quarterly Performance of Models

While random forest performs the best in our results, we note that LSTM was only able to train for 4 epochs due to compute limitations. With more training time, we think LSTM may outperform random forest since LSTM is better equipped to handle temporal data (discussed in Section 3.4).

There are some limitations of our models that – if addressed – will likely impact the performance of the models:

- Our models use data from 2 days before the day being forecasted, when in reality data may take up to 5 days to be available (due to delays in data being reported).
- Our models use the actual temperature data of the day being forecasted for training, but that data is not available at prediction time (since the day being forecasted has not happened yet). To address this issue, we would need to obtain or generate a dataset of predicted temperatures and use those in place of the actual temperatures.
- We approximate each jurisdiction's weather with the weather from the closest weather station, which in some cases may not be a good approximation (e.g. estimating Albany's weather with Oakland's weather).
- We treat all dates in the list of important dates equally (e.g. Christmas Day and Cinco de Mayo are considered as "equal" important dates).

5 Conclusion

Our findings show that ML and deep learning models are effective in forecasting day-ahead load in the East Bay. Despite the input features being fairly simple, our models are able to achieve quite accurate predictions.

Future work would diagnose the cause of worse model performance in the summer months (Figure 8), address the issues outlined in Section 4.3, and include more complicated features (e.g. cloud cover to capture solar production, macroeconomic indices to understand when people may be unemployed and thus be at home).