



Open in app

Get started



Published in Towards Data Science

You have **2** free member-only stories left this month.[Sign up for Medium and get an extra one](#)

Robert Gramillano

Follow

Apr 1, 2019 · 7 min read · ✨ · 🎧 Listen



Save





Predicting Electricity Demand in LA — outperforming the government

Introduction

This is a small blog post describing my first capstone project for [Springboard's data science career track](#). I compared a handful of machine learning models that predict electricity demand from weather features. I found that the majority of the models predicted electricity demand ~5% more accurately than government forecasts. Check out my [GitHub repository](#) for this project's code, plots, and data sets.

Data Collection and Analysis

To train and test my model, I used ~27,000 hourly measures of electricity demand in Los Angeles from July 2015 to September 2018. This data was recorded by the Los Angeles Department of Water and Power, and made available through the [Energy Information Administration's public API](#).

For weather data, I retrieved hourly, local climatological data (LCD) [from the National Oceanic and Atmospheric Administration \(NOAA\) website](#). In the LCD data set we have familiar measures such as pressure and temperature and less directly relevant quantities like wind direction. One additional feature was added: hourly cooling and heating degrees. Heat.  30 |  See days are the number of



[Open in app](#)[Get started](#)

months. **Figure 1** shows the electricity demand time stream for LA from July 2015 to September 2018. Notice the spike in demand during the summer.

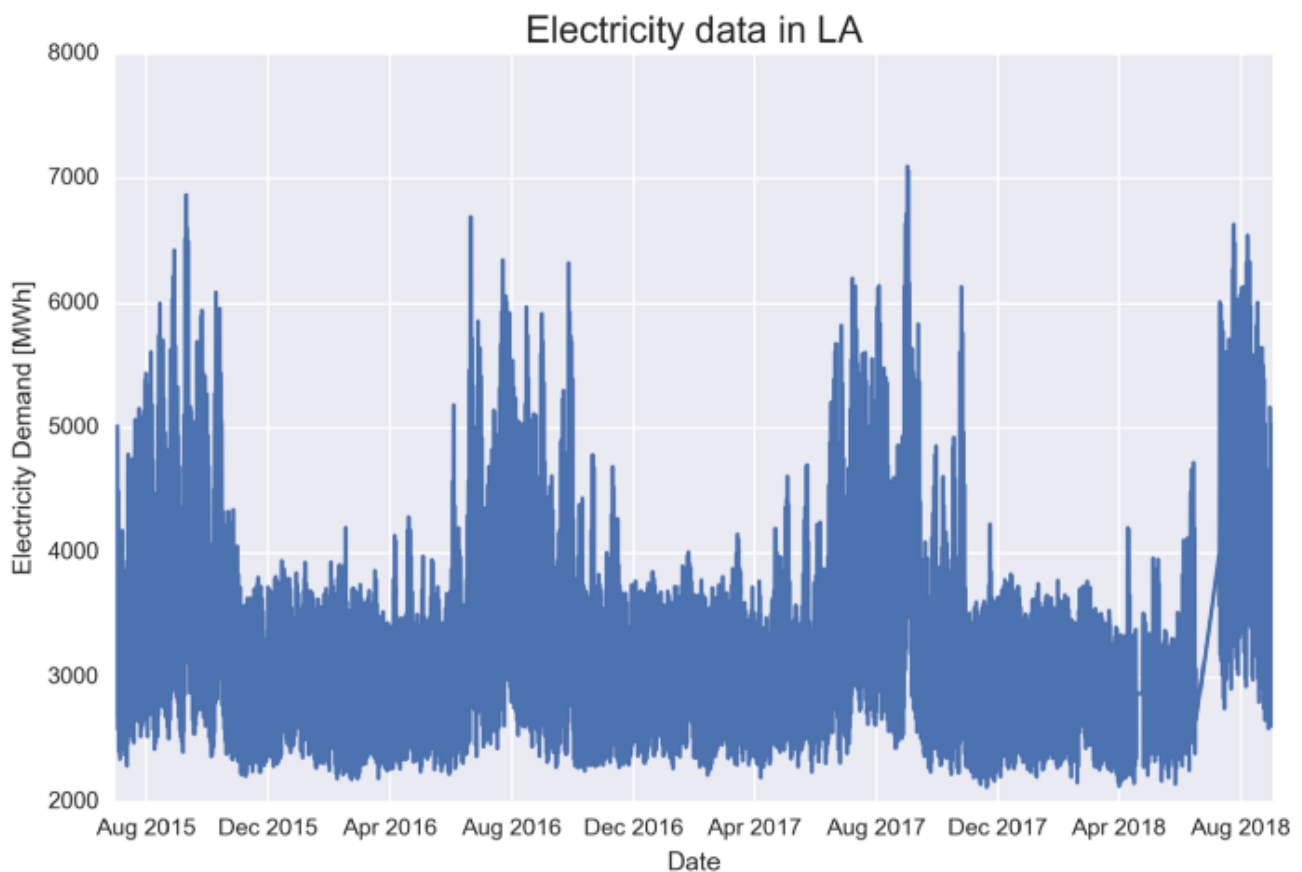


Figure 1. Electricity demand versus time in LA. Note the sinusoidal pattern, with the summer season experiencing more demand because of air conditioning.

After the initial cleansing and preparation of the day I performed some exploratory data analysis to look at initial relationships between the independent (weather) and the dependent (electricity) variables. **Figure 2** shows the Pearson correlation coefficients and **Figure 3** the r^2 values between every feature and electricity demand. We see that daily cooling degree days have a strong positive correlation with electricity demand — the vast majority of electricity usage is during the summer months of LA for cooling.





Open in app

Get started

DEMAND CORRELATIONS (PEARSON) FOR LA	
dailycoolingdegreedays	0.572275
hourlydewpointtempf	0.383226
hourlyrelativehumidity	0.365116
hourlywetbulbtempf	0.302208
hourlycoolingdegrees	0.192912
hourlydrybulbtempf	0.044278
hourlyskyconditions_CLR	0.034075
hourlyvisibility	0.009670
hourlyskyconditions_FEW	-0.011283
hourlyskyconditions_BKN	-0.013311
hourlyskyconditions_OVC	-0.014977
hourlyprecip	-0.022645
hourlyskyconditions_SCT	-0.029785
dailyheatingdegreedays	-0.221475
hourlywindspeed	-0.232047
hourlystationpressure	-0.265702
hourlysealevelpressure	-0.266134
hourlyaltimetersetting	-0.267439
hourlyheatingdegrees	-0.290998

Figure 2. Pearson correlation coefficients for all weather features with electricity demand. Cooling degrees is a strong predictor of electricity demand.

```
# Code to produce figure 3
import scipy
import pandas as pd

# get r^2 values per column and print
demand_r = {}
for col in df.columns:
    if col != 'demand':
        slope, intercept, r_value, p_value, std_err =
scipy.stats.stats.linregress(df['demand'], df[col])
        demand_r[col] = r_value**2
```





Open in app

Get started

DEMAND CORRELATIONS (r^2) FOR LA		
	col	r^2
13	dailycoolingdegreedays	0.327499
11	hourlydewpointtempf	0.146862
18	hourlyrelativehumidity	0.133310
8	hourlywetbulbtempf	0.091330
2	hourlyheatingdegrees	0.084680
0	hourlyaltimetersetting	0.071524
5	hourlysealevelpressure	0.070827
17	hourlystationpressure	0.070597
15	hourlywindspeed	0.053846
14	dailyheatingdegreedays	0.049051
10	hourlycoolingdegrees	0.037215
6	hourlydrybulbtempf	0.001961
16	hourlyskyconditions_CLR	0.001161
4	hourlyskyconditions_SCT	0.000887
12	hourlyprecip	0.000513
9	hourlyskyconditions_OVC	0.000224
1	hourlyskyconditions_BKN	0.000177
7	hourlyskyconditions_FEW	0.000127
3	hourlyvisibility	0.000094

Figure 3. r^2 values for all weather features with electricity demand.

Some of our weather features have collinearities; there are three features for pressure (station, sea level, altimeter) and additionally three for temperature (wet bulb, dry bulb, dew point). Collinearities make predictive modeling more difficult, so we drop the two temperature and pressure features with the lowest r^2 values.

Since people behave differently during the day versus the night regardless of how hot or cold it is that day, I added an 'hourlytimeofday' column, representing day or night (zero between 6AM and 6PM, one otherwise). The temperature data is important for identifying heating and cooling peaks, but the regression can start





Open in app

Get started

Before using other machine learning techniques, we perform a multiple ordinary least squares (OLS) regression on all the features versus electricity demand to isolate insignificant features. **Figure 4** shows the results. High p-value features confirm our intuitions — that heating is not important for LA. Sky condition, i.e., the cloud coverage, and wind speed are intuitively not important features for predicting electricity demand. These less-useful features are indicated by high p-values and any feature with a p-value greater than 0.1 was dropped from the data set.

-----LOS ANGELES-----						
OLS Regression Results						
=====						
Dep. Variable:	demand	R-squared:	0.701			
Model:	OLS	Adj. R-squared:	0.701			
Method:	Least Squares	F-statistic:	3970.			
Date:	Thu, 15 Nov 2018	Prob (F-statistic):	0.00			
Time:	12:14:49	Log-Likelihood:	-2.0240e+05			
No. Observations:	27055	AIC:	4.048e+05			
Df Residuals:	27038	BIC:	4.050e+05			
Df Model:	16					
Covariance Type:	nonrobust					
=====						
	coef	std err	t	P> t	[0.025	0.975]

const	4205.7364	914.386	4.600	0.000	2413.493	5997.980
hourlyvisibility	29.4799	1.741	16.935	0.000	26.068	32.892
hourlydewpointtempf	-22.0628	1.148	-19.217	0.000	-24.313	-19.812
hourlyrelativehumidity	20.3704	0.421	48.439	0.000	19.546	21.195
hourlywindspeed	-1.8849	1.196	-1.576	0.115	-4.228	0.459
hourlystationpressure	-73.6075	30.451	-2.417	0.016	-133.293	-13.922
hourlyprecip	-1034.1409	367.083	-2.817	0.005	-1753.643	-314.638
dailyheatingdegreedays	5.4324	1.098	4.949	0.000	3.281	7.584
dailycoolingdegreedays	113.6195	0.857	132.630	0.000	111.940	115.299
hourlycoolingdegrees	-32.0139	2.668	-12.001	0.000	-37.243	-26.785
hourlyheatingdegrees	-0.1662	1.452	-0.114	0.909	-3.013	2.680
hourlytimeofday	514.3887	7.579	67.872	0.000	499.534	529.244
hourlyskyconditions_BKN	73.7468	62.313	1.183	0.237	-48.389	195.883
hourlyskyconditions_CLR	127.3497	61.472	2.072	0.038	6.861	247.838
hourlyskyconditions_FEW	85.4083	63.257	1.350	0.177	-38.579	209.396
hourlyskyconditions_OVC	82.5619	61.228	1.348	0.178	-37.448	202.572
hourlyskyconditions_SCT	56.9317	62.880	0.905	0.365	-66.316	180.179

Figure 4. Multivariate OLS regression for all the features + constant. Shown are a list of coefficients with errors, t-statistics, confidence intervals.

Machine Learning Modeling

Now we prepare our dataset for machine learning modeling. Our first step is to transform our time series data into a form suitable for supervised learning. Drawing heavily [from this blog post](#), I transformed our dataset such that I used all of the





Open in app

Get started

split our data into a training and testing set, then begin evaluating our machine learning models. To make the analysis consistent with our later use of recurrent neural networks and long short-term memory, we designate the first year of data as the training set and the rest of the data as the testing set (~32% training and ~68% testing). This split avoids overfitting and a more traditional 80/20 split on training and testing yields virtually the same model performance. The general process for modeling is: 1.) use a [min-max scaler](#) on all the features to make them more comparable, 2.) fit to the training set, and 3.) compute the relevant metrics on the testing set and compare all of the models at the end. In our analysis we perform no hyperparameter tuning for the models (default settings of sklearn) and still achieve quite remarkable results.

I trained and tested a plethora of standard machine-learning models as described above using the sklearn module. The specific procedure and results of the modeling can be found in [this IPython notebook](#). With straight, out-of-the-box models, I achieved impressive accuracy, with r^2 values over 0.95 for over half the models I tried. In addition, the training and testing r^2 values are within a few percent of each other, indicating that I'm not overfitting.

Besides out-of-the-box models, we also used recurrent neural networks (RNN) and long short-term memory (LSTM). A detailed walkthrough on the basics of RNNs and LSTM along with the modeling procedure can be found [in this IPython notebook](#). Using this model, I achieved comparable performance to the best machine-learning models I tested previously. **Figure 5** shows the loss (in mean-absolute error) versus epoch for training and testing sets using LSTM modeling. The testing loss remains a bit higher than the training loss, indicating that the model is not overfitting.

```
# Code to produce figure 5
from keras.models import Sequential
from keras.layers import Dense, LSTM
import matplotlib.pyplot as plt

# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1],
train_X.shape[2])))
model.add(Dense(1))
```



[Open in app](#)[Get started](#)

```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.xlabel('Epoch')
plt.ylabel('Loss (MAE)')
plt.legend()
plt.show()
```

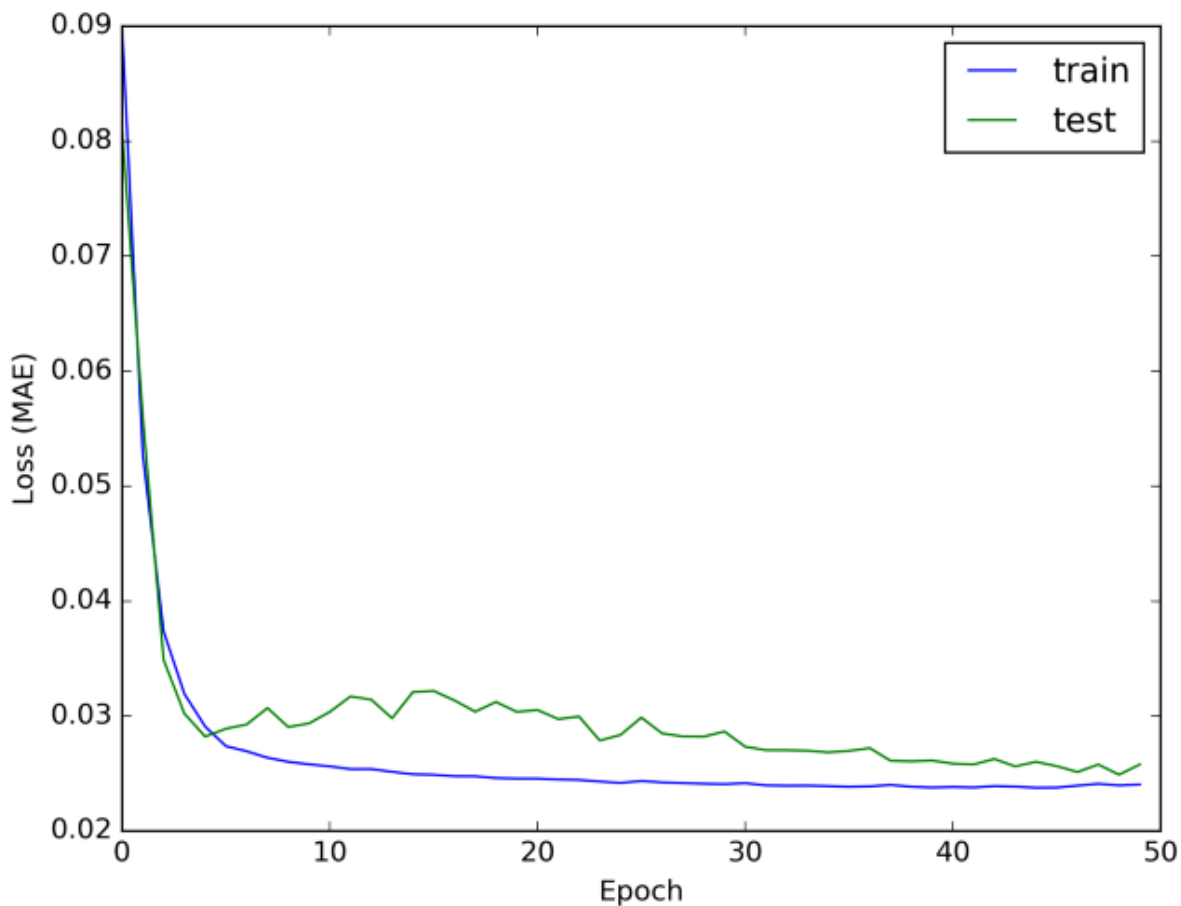


Figure 5. Loss measured in mean-absolute error (MAE) versus epoch for both training and testing of our LSTM network. Testing error remains slightly above training, indicating that we are not overfitting. The LSTM network was optimized using [the Adam implementation of stochastic gradient descent](#) with MAE loss.

EIA also provides a day-ahead electricity demand forecast that we would like to compare to our models. EIA uses various methods to [forecast demand](#). The forecast depends heavily on weather forecasts and the [day of the week](#). Using the same testing set we find that EIA's day-ahead demand forecast achieves worse accuracy than all but one of our models, indicating the power of simple, out-of-the-box machine learning models on regression problems.





performs very well on its own. The training and testing errors for all models remain within the percent level of each other, indicating we are not substantially overfitting the data. Since cross-validation methods become trickier when dealing with correlated, time stream data, we leave out cross-validation in our analysis to keep all of our models comparable and consistent. We find that the gradient boosting model has the best performance, although the top five models are all similar in performance (differences of ~1% in performance between them).

Model Name	Root Mean Squared Error (RMSE) [MWh]	r^2
Gradient Boosting	151.9	0.9634
Linear Regression	158.0	0.9604
Bagging	160.0	0.9594
Random Forest	160.5	0.9592
RNNs w/ LSTM	162.7	0.9581
AdaBoost	178.6	0.9495
Decision Tree	214.4	0.9272
EIA's day-ahead forecast	227.9	0.9174
k-NN	242.7	0.9067

Table 1. Table of results of machine learning models. The best performing model (gradient boosting) is highlighted.

With our best-performing model chosen (gradient boosting), we now want to see which features are most important in predicting electricity demand. This is shown in **Figure 6**. “Feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of samples that reach the node, divided by the total number of samples.” Demand during the previous hour is the most important feature for predicting demand at the current hour, but other proxies like cooling degree days are also a strong predictor as we expected.





Open in app

Get started

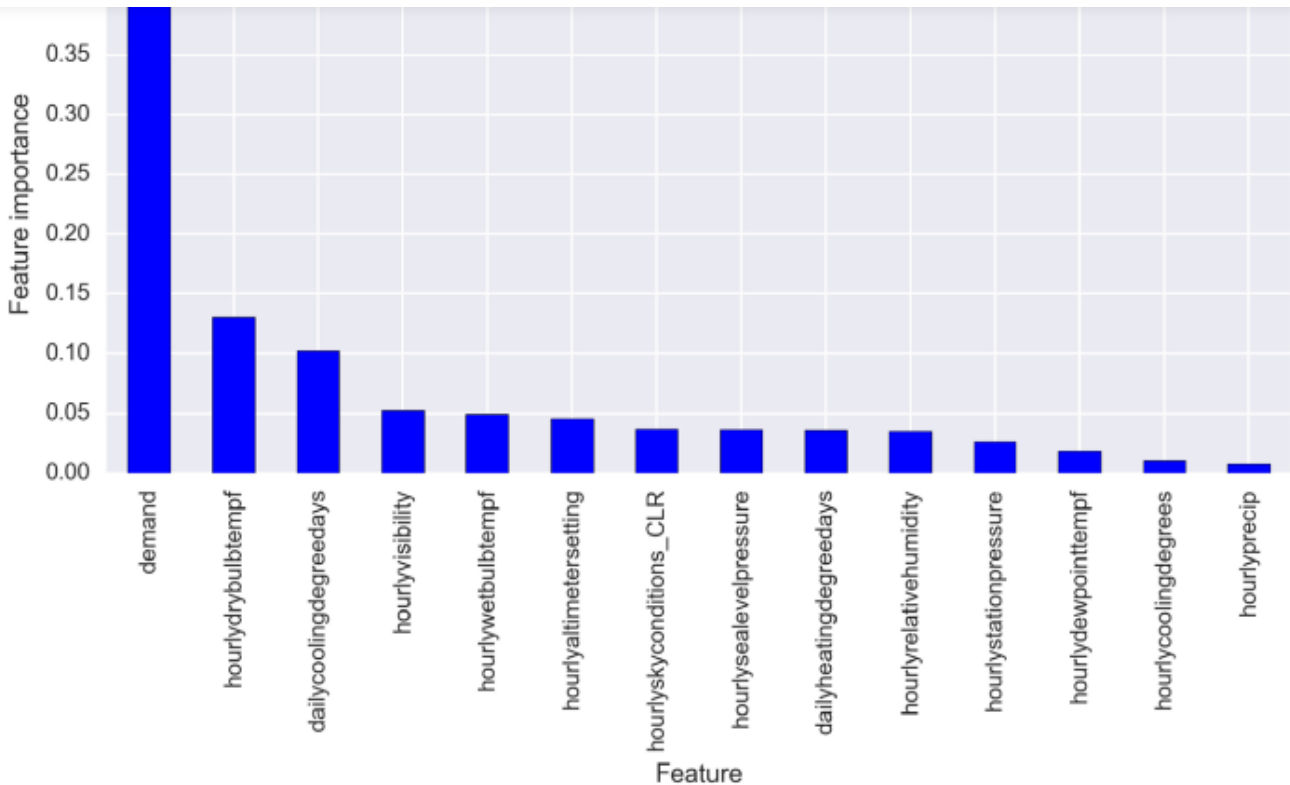


Figure 6. Feature importance for each feature in the gradient boosting model. Demand at the previous time step is four times more important than cooling degree days. Reframing our supervised learning problem in this way greatly improved the predictive power of the models.

Conclusion

We show that outperforming EIA's day-ahead electricity demand forecast is actually quite easy with machine learning. Remarkably, all but one of our models (k-NN) performance better than the EIA's model. Relatively simple data scraping, cleaning, and machine-learning modeling from public data sources results in much more accurate modeling than traditional methods. We report an overall ~5% increase in performance over EIA's model; energy companies could use these models to better brace for high-demand spikes in electricity and ultimately increase profits. Additional hyperparameter tuning and cross-validation techniques could be performed to increase performance now that the best model has been identified, although the increase would most likely be marginal at best and since our models already outperform the EIA's, we conclude the report without tuning. [Please check out my GitHub for this project](#) for a more detailed report, code, plots, and documentation.



[Open in app](#)[Get started](#)

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.



Get this newsletter

[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app

