Power laws - summary of the solution

Tomasz Waleń

2018–11–08

Problem description

Given limited cold start data about previous energy consumption predict future consumption in specified prediction window:

- 24 hourly predictions, or
- 7 daily predictions, or
- 2 weekly predictions.

Train set consisted of 758 different series of data, each with exactly 28 days of consumption data (with hourly consumption). Cold start data (and test) consisted of 625 different series of data, each with varying amount of lagging data (from 1 to 14 days).

The metric used for evaluation of submissions was very similar to *mean absolute percentage error* (but additional weights were used to make each series equally important).

More details about the contest could be found in:

https://drivendata.org/competitions/55/schneider-cold-start/.

General approach

First I've divided the problem into three separate sub-problems depending on the prediction window (hourly, daily, weekly). I assumed that each prediction window would require different approach. Since LSTM approach was too complicated and required a lot more computational power, I've decided to use simpler neural networks. But in this case it was necessary to handle different length of inputs (different number of cold start days). The simplest solution was to divide the problem even further by the number of cold start days (to simplify the problem I was not using more than 7 days of cold start data) I've ended up with 21 different models (3 prediction windows * 7 cold start days). For each subproblem I've tried to select best neural network architecture that would solve it, and the final solution was a mixture of simple FF networks (with 1 or 2 layers) and custom made neural network (used for hourly predictions).

The custom made neural network was created in such a way, that it analyzed the whole input (all lagging consumption and additional features) and then it computed how to multiply each lagging day to best match target day. At the end all of such partial predictions were averaged.

Since our team was created in the very last days, we did not have chance for more in-depth cooperation, our final solution was a simple average of our best solutions. It gave surprisingly good result probably due to the fact that our approaches were complementary.

Business understanding

First I've browsed Google Scholar looking for scholar papers about modelling energy consumption. Unfortunately the contest assumptions were slightly different than the models from the papers. In real-life solutions the schedule of is_day_on/off (including holidays and special events) is known in advance, in the contest this information was very limited. In real-life models the important factor was the temperature, and in the contest this information was non-reliable. But this review gave me basic understanding how the energy consumption in buildings is measured and controlled.

Data preprocessing

I've used combination of command line tools and jupyter notebooks for data preprocessing. Eventually all preprocessing code was moved to python command line tools (located in `src/data` directory).

EDA

At the beginning of the contest I've did some preliminary EDA, the sample visualizations are included in notebook `1-walen-eda.ipynb`.

First I've tried to look for some selected series. Some are very regular and some are so irregular that correct prediction is virtually impossible (see fig. 1).

As we can see, there is very much variety between series, some contains very low consumption buildings, and there are series with huge consumption (see fig. 2). Also we can see that large portion of data is from 2017 and there is smaller number of samples from 2013 and 2014.

Although the temperature data was not available for all series, we can clearly see, that there is much variety in the geographical distribution of series. Probably most of the data come from northern hemisphere, but there is lot of outliers like high temperatures during December or January (see fig. 3).

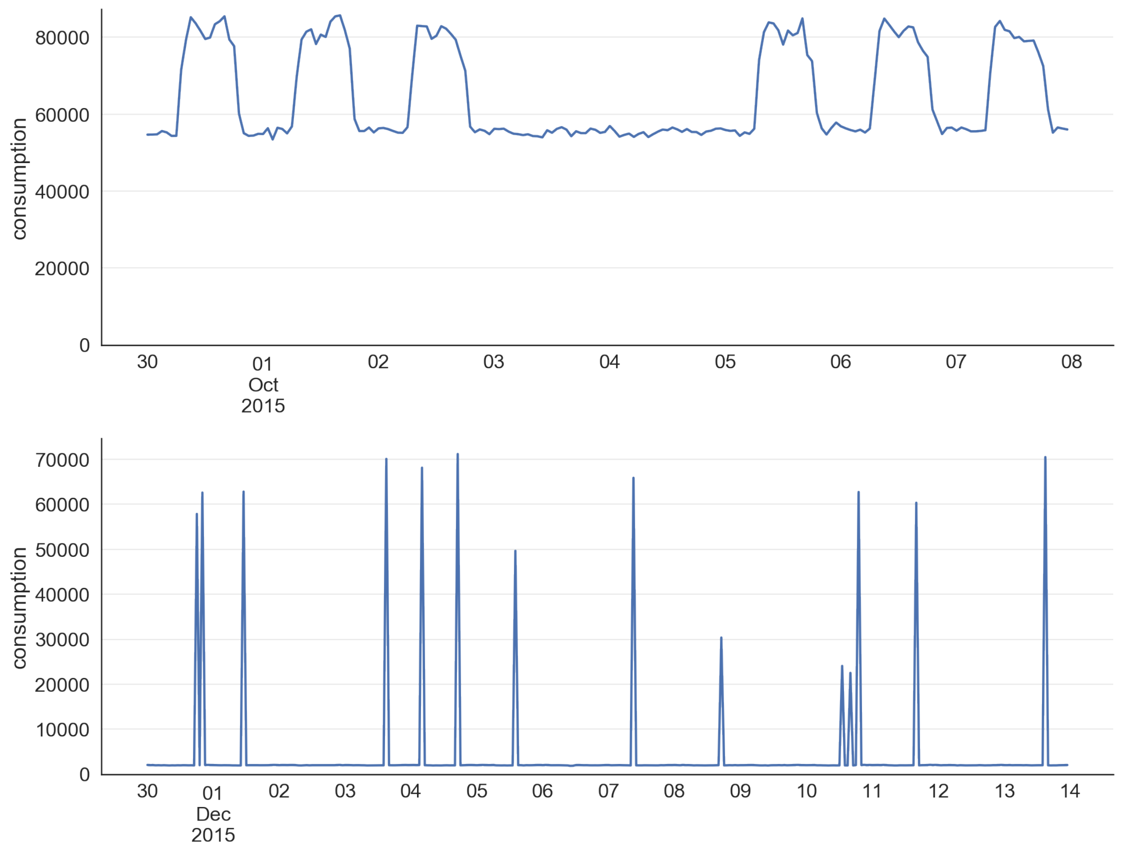


Figure 1: Example of regular and random consumption series

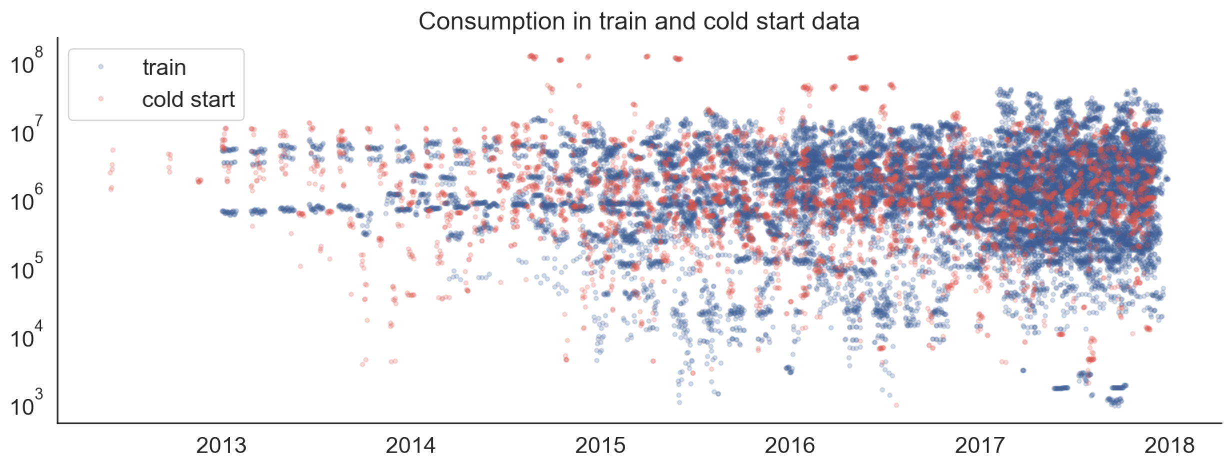


Figure 2: Consumption data over time

Cleaning

- some series (from both train and cold_start) contained wrong wrong data about hourly consumption (ie. constant hourly consumption over few days) - this was probably caused by the incorrect readings for some particular time period or by averaging some time frames - I've removed those entries from

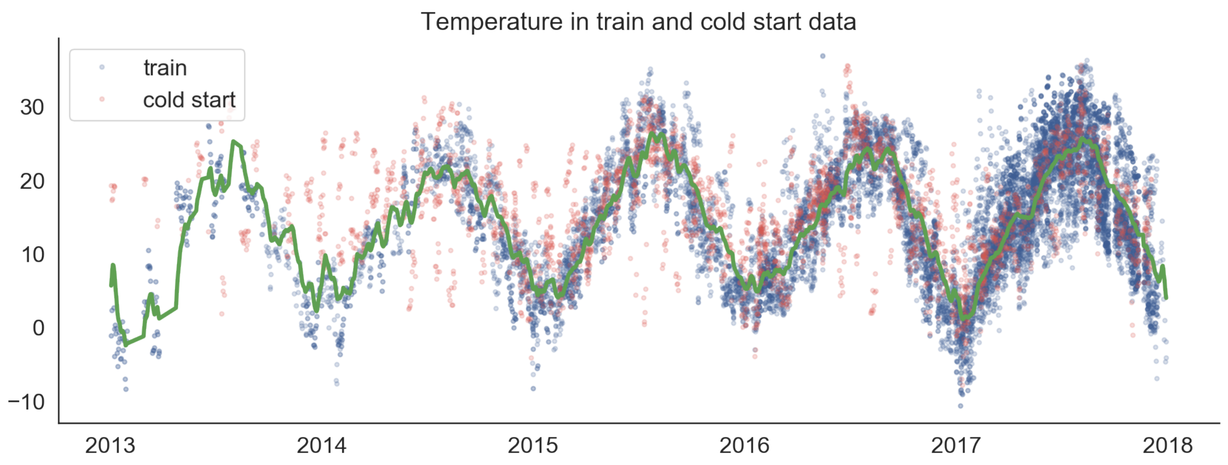


Figure 3: Temperature data over time

training
- some data series contained 0 consumption, since there were very few of such cases I've ignored this problem
- there were few series (in training set) with 5 or 7 day off days (in meta data), I assumed that this was caused due to a mistake and those entries were inverted (to 2 or 0 days off)

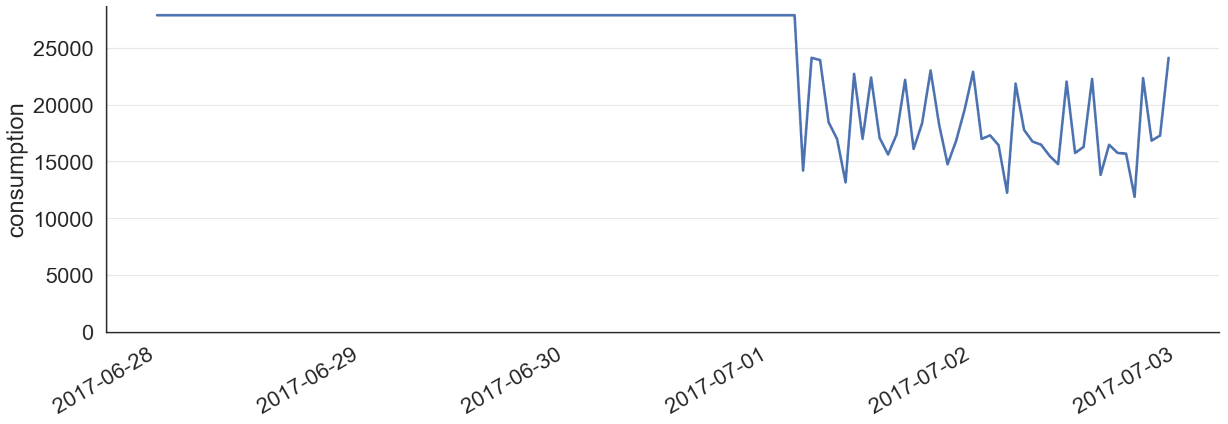


Figure 4: Example of series with invalid (averaged) consumption data

Features

- lagging hourly consumption
- lagging daily consumption (for daily / weekly predictions)
- mean consumption for all lagged days / is_day_on / is_day_off / min / max / days
- approximation of mean weekly consumption based on mean values (`consumption_h_mean_*` features):

- due to a mistake initially I was considering last 7 days, but this caused a data leakage in training set - unfortunately I've discovered this to late and some models still use those features, to mark it I've named those features with `leaking_` prefix,
- after discovering this issue, I've created multiple copies of this features depending on the number of allowed days in the history (i.e. `consumption_h_mean_w_last_4d` is computed based on only 4 days of historical data)
- number of working days in a series
- is_day_off (for future and lags)
- is_holiday (for future and lags) - I was using working calendars (from `holidays` library) for US, France, Australia, I was also experimenting with some combined holidays list (for given day is a holidays in any of selected countries – features `is_holiday_custom`)
- I was experimenting with other features like (day of week, temperature, surface_id, detecting sudden shutdown of given facility) but without great success
- all features were rescaled to $[0, \ldots, 1]$ range using min-max scaler (but both minimal and maximal values were adjusted by constant factor 0.9 or 4.0)
- I was also computing the sample weights, such that loss calculated within Keras would be the same as in the contest rules

Trivial predictions

During work on the project I've noticed that (due to NMAE properties) some predictions could incur very high cost. I've identified 5 series that:

- had dramatically different predictions depending on the choose model/features in my experiments
- had very little information or inconsistent information in the cold start data

I was afraid (and I think that this also happened in reality) that my predictions on those series could mask the general behavior of the models on the whole test set and public LB score.

So I gave up upon those 5 series and decided to choose some fixed predictions (based on the cold start data) that would reduce fluctuations in LB score.

The trivial predictions are either copy of some previous day for a series or copies of some selected hours repeated to pad 24 hours, so it resembles trivial model that returns some fixed lagged hours.

For 3 series it was educated guess based on the previous submissions, for 2 series it was pure gambling.

It should be noted that such predictions could affect in predictable way only public LB score.

Models

I was using Keras neural networks models. For each subproblem the most efficient architecture was used:

- for hourly predictions: 1- or 2- layers NN, and customized NN for 6+ cold start days,

- for daily predictions: 1-layer NN,
- for weekly predictions: 2-layers NN.

In the very last day of the competition, selected 3 models (that covered more than 50% of test data) were trained with 20 different random seeds and I have used averaged results from those models (refer script `gen-20seeds-pred.sh` for details).

Customized NN for hourly predictions

If the number of cold start days was large enough it was very hard to train simple NN to predict next 24-hours consumption (too many parameters, too little data). So I tried to reduce the number of parameters in NN, but on the other hand be able to use all historical data.

I've ended up with the following architecture:

```
def original_gen_hourly_pred_model(features, cold_start_days=None):
    input_size = len(features)
    output_size = 24

    inputs = keras.layers.Input(shape=(input_size, ), name='input')
    x = inputs
    for layer_num in range(2):
        x = keras.layers.Dense(128, activation='relu')(x)
        x = keras.layers.Dropout(0.2, seed=layer_num)(x)

    out = []
    for i in range(cold_start_days):
        j = features.index(f'consumption_lag_h_{24*(i+1):03d}')
        assert len(features[j: j+24]) == 24
        inp_i = keras.layers.Lambda(lambda x: x[:, j:j+24])(inputs)
        x_i = keras.layers.Dense(1, activation='relu')(x)
        m_i = keras.layers.multiply([inp_i, x_i])
        out.append(m_i)
    avg = keras.layers.average(out)
    avg = keras.layers.Dense(output_size, activation='relu')(avg)
    outputs = avg

    model = keras.models.Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer='adam', loss='mean_absolute_error')
    return model
```

Simply this network tries to found out optimal parameters, how to multiply the consumption from each previous day to best resemble the target day, and averages all the predictions generated using this method.

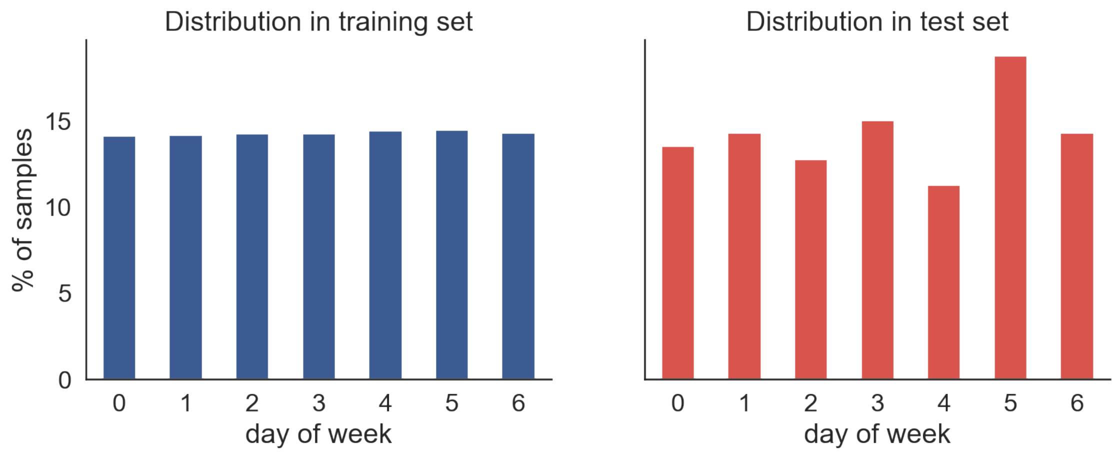
Training

I was using all available input data for training (from both `train` and `cold_start` sets). Each series of data could be a source for multiple training samples, I've been using sliding window approach with an increment of 1 day. Of course models with lower number of cold start days had large number of training samples then the models with higher number of cold start days.

Validation

The train/validate split was done using `series_id` (so all data from given series were either in train or validate set).

I've noticed that the distribution of weekdays in dates from the train set and the test set were different.



To handle that issue I've been also experimenting with:

- using samples from validate set that started with the dates from the test set
- increasing sample weights for training/validation samples with the dates from the test set

What could I have done better

The project itself became quite big (many models, features, etc), I've lost lot of time due to simple programming errors. Using even simple automated tests could have saved me a lot of time. Definitely in my next data science project I would use Test Driven Development.

Also I have learnt that the time management is very important in such contests - you simply need to reserve a significant amount of time, especially around the end of the contest. I made an error to work on the project mostly in early mornings or late evenings hours, and it turned out that that this is not enough to be very productive.

I also lost a lot of time due to change of the framework and validation methods during the contest. At the beginning of the contest I was using Jupyter notebooks to calculate models, but after reaching some threshold of complexity, I was not able to work efficiently and I had to rewrite most of the code. Also this caused a lot of problems when I was preparing this repository, since the final model is a composition of different models (some created very early during the competition) I had to rollback some of changes (even some bugfixes) to be able to reproduce final solution.