Predicting energy consumption using ARIMA time-series models

Author: Barbu Paul – Gheorghe

Teacher: Árpád GELLÉRT

2017-2018

Table of Contents

Introduction

The aim of this project is to predict energy consumption over a period of time, given the previous several days of the load profile of the network.
The prediction will be done using time-series prediction models, namely variations of the seasonal (and non-seasonal) ARIMA model.

Data sets

The data sets were provided by the university, they represent energy consumption over three months, captured every 5 minutes.

The data sets are called *PV1*, *PV2*, *Ph1*, *Ph2*, *Ph3* and look like this*:*

timestamp	PV1	PV2	Ph1	Ph2	Ph3
1420066800	0	0	0	352.8	401.2
1420067100	0	0	0	354.8	484.6
1420067400	0	0	0	356.8	423.6
1420067700	0	0	0	388.2	418.6
1420068000	0	0	0	353	187.2
1420068300	0	0	0	352.4	247
1420068600	0	0	64.2	377.2	203.4
1420068900	0	0	0	209.6	217
1420069200	0	0	0	234.4	603.4
1420069500	0	0	46.8	232	948.2
1420069800	0	0	70.4	110.6	148
1420070100	0	0	0	0	109.2
1420070400	0	0	0	0	149.4
1420070700	0	0	0	0	128.8
1420071000	0	0	0	0	47
1420071300	0	0	0	0	0
1420071600	0	0	0	0	23.6
1420071900	0	0	0	0	146.4
1420072200	0	0	0	0	139.4
1420072500	0	0	0	0	109.8

The timestamp column represents the time the capture was made at, it could be a UNIX timestamp or just a simple five minute counter, it doesn’t matter.
Missing or invalid energy values (ie. negative values) were replaced by the last valid value.

Implementation

General

In order to predict the future energy consumption we have to discover the ARIMA parameters that best suit the data patterns we have observed during the capture.
Like all machine learning and prediction methods, we're going to use a variation of the 80/20 rule. 80% of the data is used for training the model, and the rest of 20% is used to test the trained model.

Evaluation

In order to evaluate the accuracy of our model's predictions we're going to use the Mean Absolute Error (MAE), which has the following formula:

$$E = \frac{\sum_{i=1}^{N} |R_i - P_i|}{N}$$

Ri – the real value (observed)
Pi – the predicted value
N – The number of predictions the model made

Running the prediction algorithm

Once the model is fitted and ready to make predictions, we have two ways of running it.

1. **Incrementally –** this means that we make one prediction into the future, then use the real value in order to re-train the algorithm and only then make the second prediction. This method has the disadvantage of being more time and computation intensive since the model has to be re-trained for every prediction.
2. **All at once –** this method just asks the ARIMA implementation to provide all the predictions for the entire period, all at once. This method has the disadvantage of using just the initial training data for every prediction that we make, but it is more economical time and computation wise.

Languages & Technologies

The following languages have been used to study, evaluate and implement this project:

- **Python** – used Python to study how the data looks and try to manually estimate the ARIMA parameters. The study has been done in a Jupyter IPython Notebook (http://jupyter.org/) in order to be able to interactively see the data plots.
- **R** – this statistics focused language provides a library (https://www.jstatsoft.org/article/view/v027i03) that can automatically estimate the ARIMA parameters (both non-seasonal and seasonal parameters)
- **Java** – the final implementation had to be done in Java and it uses the **timeseries-forecast** library from Workday (https://github.com/Workday/timeseries-forecast/).

Usage instructions

Python

The installation steps are as follows:

1. Install Python 3.x
2. Install the Jupyter IPython Notebook: http://jupyter.org/
3. Using a command line a the **pip** utility, run the following command:

   $ pip install pandas numpy matplotlib statsmodels

4. Run the Jupyter notebook in a command line by running the following command:

   $ jupyter notebook

5. A new borwser window should've opened, navigate to the **Statistical Study of The Data for Arima Application.ipynb** file (in the **statistical-study** directory) and you may run it in order to see the plots and the results of a model, this can be used interactively to study how the model changes when modifying certain parameters.

R

In order to be able to run the R code, which just reads the data and tries to automatically estimate the parameters for the ARIMA algorithm we have to:

1. Install R: https://www.r-project.org/
2. (Optionally) install Rstudio: https://www.rstudio.com/
3. On the command line start the **R** interpreter OR start **Rstudio** and inside run:

   install.packages("forecast")

4. Then you'll be able to run the R code in the **statistical-study** directory.

Java

The Java implementation uses Maven to build & run the project:

1. Download and install any Java version, I have used:

   $ java -version
   openjdk version "1.8.0_141"
   OpenJDK Runtime Environment (build 1.8.0_141-8u141-b15-3-b15)
   OpenJDK 64-Bit Server VM (build 25.141-b15, mixed mode)

2. Download Maven: https://maven.apache.org/
3. Install Maven: https://www.mkyong.com/maven/how-to-install-maven-in-windows/
4. Download the **timeseries-forecast** library: https://github.com/Workday/timeseries-forecast/ (or use the version provided in this repo) and in a command line in that directory run:

   $ mvn install

5. Then run the following in the **energyarima** directory:

   $ mvn package
   $./run.sh

6. The Java code should run and provide the Mean Absolute Error for the data set you have configured, please see below on how to configure the predicted periods, the benchmarks and the running method (iterative or all at once) of the model.

Code

The Java code is based on the **timeseries-forecast** library.

The code is in the package com.paul.energyarima (in the **energyarima** directory), the class's name is "App".

First of all, the configurable part of the code:

```
1  private static final int PREDICTION_DAYS = 3;
2  private static final int OBSERVED_DAYS = 10;
3  private static final String FILE_NAME = "benchmarks/2hrs-ph3.txt";
4  private static final int TIME_SPAN = 120;
```

Here we can specify:

- How many days we should predict: PREDICTION_DAYS.
- How many days we should read from the observed data: OBSERVED_DAYS. Please note, this constant and the one above, should follow the 80/20 rule as is the case for any learning algorithm.
- The FILE_NAME constant specified the file that contains the energy observations.
- The TIME_SPAN constant contains the time in minutes, between two observations, this is directly dependent on the FILE_NAME constant since different files may have different capture intervals.

Further down in the code, in the "main" method we have:

```
1  int p = 0;
2  int d = 1;
3  int q = 1;
4
5  int P = 0;
6  int D = 0;
7  int Q = 2;
8
9  int m = 12;
10
11 ArimaParams params = new ArimaParams(p, d, q, P, D, Q, m);
12
13 oneShot(params);
```

These are the ARIMA parameters that will be used to fit the model over our data and later, make the predictions using one of the two methods, the iterative, or the "all at once" method.

I will present here just the second method of running the algoritm, the “all at once” method since the code is shorter. The method’s name is “oneShot”.

```
1  FileReader fr = new FileReader(FILE_NAME);
2  BufferedReader bfr = new BufferedReader(fr);
3
4  double[] trainData = readBenchmarks(NUM_OBSERVATIONS, bfr);
5  double[] testData = readBenchmarks(NUM_PREDICTIONS, bfr);
6
7  ForecastResult forecastResult = Arima.forecast_arima(trainData, NUM_PREDICTIONS, params);
8
9  double[] forecastData = forecastResult.getForecast();
```

On lines 1 and 2 we initialize the file & buffers which we’ll use to read the observed energy consumption up to this point.

On lines 4 and 5 we do the actual reading operation, by using the **readBenchmarks** function.

On line 7 we fit the model using the “params” we got as functions parameters, these are the ARIMA specific **`(p, d, q) (P, D, Q) m`** parameters. Here we also specify how many predictions we want to make when we call “getForecast” on the 9th line.

Results

Unfortunately ARIMA is not very well suited for running on very frequent data set with a high seasonality.

The initial data set, the one captured every 5 minutes, had a seasonality of 288 (every 288 entries we expect the data to repeat itself). This number comes from the fact that we expect the energy consumption to be similar in two 24 hours periods, hence 288 * 5 minutes = 24 hours.

This seasonality number is too high, in order to reduce it we can simulate a one hour capture period by using one the 12th observation from the original data set. We do this in order to generate a 2 hour data set as well.

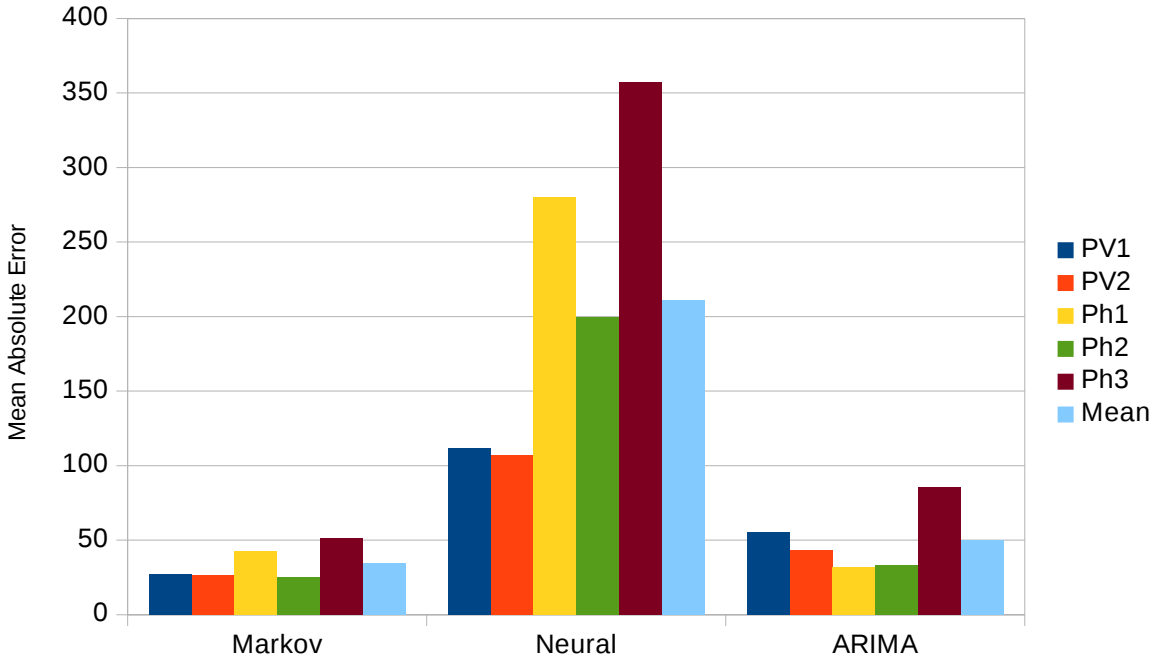
As such we have three data sets in the end:

- a 5 minute one (seasonality of 288)
- a one hour one (seasonality of 24)
- and a two hour one (seasonality of 12)

The ARIMA parameters used are a result of manual Design Space Exploration using Python and some Automatic Design Space Exploration using the R code.

I used both seasonal and nonseasonal ARIMA models as can be seen in the **best-arima.xlsx** document in the **energyarima** directory.

The final comparison is done against other methods of prediction (Markov and neural-network based) and the results are mixed. The ARIMA method is better than the neural-network, but worse than the Markov predictions as can be seen below.



It should be noted that the results used here all used an iterative method of prediction as described in the previous sections. For all tests, the observed days constant was set to 10 days and we computed the equivalent of a 3 day prediction. The ARIMA results are the ones best achieved on every data set as follows:

Data set	Capture interval	MAE	p	d	q	P	D	Q	m
PV1	2 hours	55.38	5	1	0	20	0	0	12
PV2	2 hours	43	5	1	0	20	0	0	12
Ph1	1 hours	31.96	4	1	1	2	0	0	24
Ph2	1 hours	32.92	4	1	1	2	0	0	24
Ph3	5 mins	85.28	1	1	2	2	0	0	288

It’s worth noting that in some cases the “all at once” prediction method would have yielded better results (lower Mean Absolute Error).

Bibliography

- How to Difference a Time Series Dataset with Python: https://machinelearningmastery.com/difference-time-series-dataset-python/
- ARIMA models: https://www.otexts.org/fpp/8
- Rules for identifying ARIMA models: http://people.duke.edu/~rnau/arimrule.htm
- Seasonal ARIMA with Python: http://www.seanabu.com/2016/03/22/time-series-seasonal-ARIMA-model-in-python/
- How to Create an ARIMA Model for Time Series Forecasting with Python: https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/
- Partial Autocorrelation Plot: http://www.itl.nist.gov/div898/handbook/pmc/section4/pmc4463.htm
- ARIMA parameters for a very frequent data set https://stats.stackexchange.com/questions/315493/arima-parameters-for-a-very-frequent-data-set