RESIDENTIAL POWER FORECASTING

DEFINITION

Domain Background

Residential energy consumption accounts for 37% of total electric energy in the USA[1]. Therefore, increasing energy efficiency is key both to reduce environmental impact and improve economic metrics. In order to increase efficiency, especially during times of high demand, regulators and organizations in charge of delivering energy to customers utilize price-based signals to reduce residential consumption through Demand Response programs[2]. These type of programs typically penalize consumption (energy is more expensive) during high-demand hours. This type of programs has led to emergence of residential batteries as those sold by Tesla[3]. In order to know when it is cost effective to buy electricity from the grid or take it from the battery, it is important to optimally schedule the usage of these types of battery systems. However, for optimal schedule it is important to know 24 ahead what will be the likely consumption pattern of a household. Therefore, this capstone project addresses the problem of forecasting residential electricity consumption.

In order to do so, the following datasets are leveraged in this project:

- Historical power consumption.
    - Source: Pecan Street[4]
    - Data:
        - Power Consumption (in Watts) indexed by timestamp
        - Data resolution: 1 minute
- Weather:
    - Source: Weather Underground[5]
    - Data:
        - Temperature (ºC)
        - Humidity (%)
        - Data resolution: 1 hour

---

[1] Elma, O., & Selamoğullar, U. S. (2017). A Survey of a Residential Load Profile for Demand Side Management Systems. In *2017 IEEE International Conference on Smart Energy Grid Engineering (SEGE), Oshawa, ON* (pp. 85-89).

[2] Mathieu, J. L., Haring, T., Ledyard, J. O., & Andersson, G. (2013, May). Residential demand response program design: Engineering and Economic perspectives. In *European Energy Market (EEM), 2013 10th International Conference on the* (pp. 1-8). IEEE.

[3] https://www.tesla.com/powerwall?redirect=no

[4] http://www.pecanstreet.org/

[5] https://www.wunderground.com/

Problem Statement

The objective of this project is to create an electrical power consumption model to forecast how much energy a household is going to consume in the next 24 hours. The model will take information from historical power consumption, calendar information, and local weather. The output will be a T prediction of how many kW the household is going to consume in the next 24 hours. In this case, the 24 hours will be divided into 24 timesteps, one for each hour of the day. Similar work has been carried out before in scientific research publications. Zheng, H. *et al.* [6] propose the use of Long-Short Term Memory Deep Nets as well as Boosted Decision Trees to solve this problem. Fu, Y. *et al.*[7] propose using Support Vector Machine regressor to predict building energy consumption. Suganthi, L. *et al*.[8] review papers that have taken a similar challenge in building energy forecast models.

In order to solve this problem, we follow the following steps:

1. Load the complete dataset and divide it into training and testing. Due to the sequential nature of the data, the dataset is divided into training and testing set as follows:
    a. Training:
        i. Data from 01/01/2016 to 12/31/2016 for a household.
    b. Test:
        i. Data from 01/01/2017 to 06/30/2017 for a household.
2. Perform preliminary analysis on the dataset to better understand the problem and check what variables might be important.
3. Calcule a benchmark model based on traditional ARIMA model.
4. Define the LSTM architecture using a validation (a subset of the training set described above).
5. Compare both models in the testing set.

To sum up, the result of the project should provide a model that is ready to perform 24 hours ahead forecast of a household's electricity consumption with a resolution of 1 hour.

Metrics

The forecasting of a series of real numbers can be considered as a regression problem. Therefore, in order to measure the performance of the forecasting models developed below, Mean Square Error (MSE) and Mean Absolute Error (MAE) of the models' output will be calculated. Both MSE and MAE are typical error metrics in time series forecasting problems. While MSE penalizes large deviations from the actual value (by squaring them), MAE considers

[6] Zheng, H., Yuan, J., & Chen, L. (2017). Short-term load forecasting using EMD-LSTM neural networks with a Xgboost algorithm for feature importance evaluation. *Energies*, *10*(8), 1168.
[7] Fu, Y., Li, Z., Zhang, H., & Xu, P. (2015). Using support vector machine to predict next day electricity load of public buildings with sub-metering devices. *Procedia Engineering*, *121*, 1016-1022.
[8] Suganthi, L., & Samuel, A. A. (2012). Energy models for demand forecasting—A review. *Renewable and sustainable energy reviews*, *16*(2), 1223-1240.

all the errors equally important. It is worth mentioning that in order to select the best LSTM architecture, we will use the MSE, for its better mathematical properties (it is a better loss function for the Neural Net).

$$\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \tilde{y}_i)^2$$

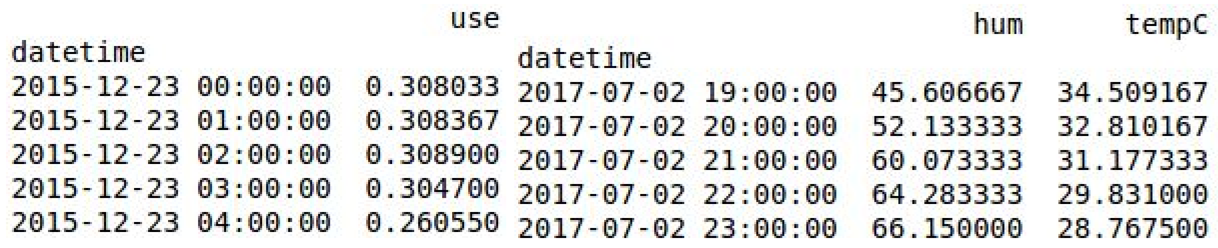
$$\mathrm{MAE} = \frac{1}{n} \sum_{j=1}^{n} |y_j - y_j|$$

Fig. 1: MSE and MAE definitions.

ANALYSIS

Data Exploration and Visualization

In section an exploratory analysis of the data is provided. It is worth mentioning that the raw data did not have any missing values nor abnormal values were detected. The data analyzed here has been already preprocessed to an appropriate format. That is energy data has been averaged across hours to adjust data resolution from 1 minute to 1 hour as the output requires. On the other hand the weather data has been preprocessed as well to obtain 1 hour resolution data. These steps are documented in following sections. The data analyzed in this section looks like the series in Fig. 2.

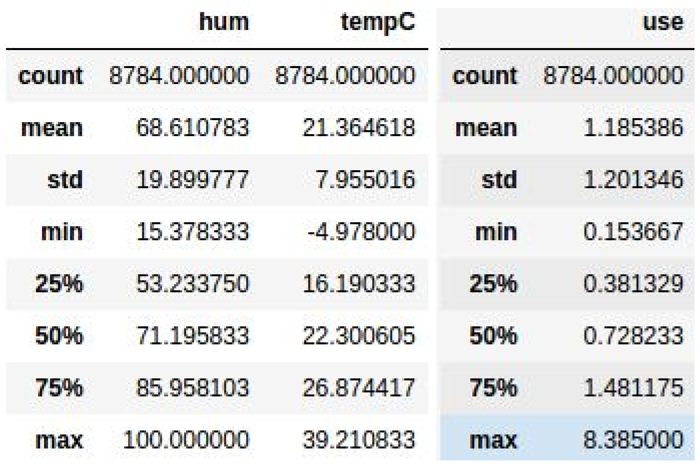


```
                          use
datetime
2015-12-23 00:00:00  0.308033
2015-12-23 01:00:00  0.308367
2015-12-23 02:00:00  0.308900
2015-12-23 03:00:00  0.304700
2015-12-23 04:00:00  0.260550
```

```
                           hum      tempC
datetime
2017-07-02 19:00:00  45.606667  34.509167
2017-07-02 20:00:00  52.133333  32.810167
2017-07-02 21:00:00  60.073333  31.177333
2017-07-02 22:00:00  64.283333  29.831000
2017-07-02 23:00:00  66.150000  28.767500
```

Fig. 2: Original Dataframes.

The summary statistics for both the energy and the weather data in the training set (year 2016) are presented in Fig. 3. In that year we have that the mean household consumption is 1.18 kW with average weather conditions of 68 % humidity and temperature of 21.36 ºC.

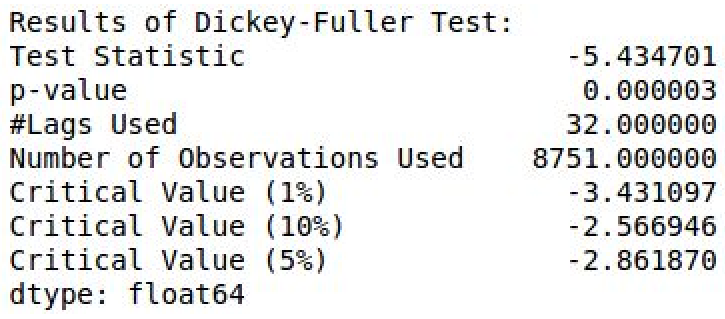


	hum	tempC
count	8784.000000	8784.000000
mean	68.610783	21.364618
std	19.899777	7.955016
min	15.378333	-4.978000
25%	53.233750	16.190333
50%	71.195833	22.300605
75%	85.958103	26.874417
max	100.000000	39.210833

	use
count	8784.000000
mean	1.185386
std	1.201346
min	0.153667
25%	0.381329
50%	0.728233
75%	1.481175
max	8.385000

Fig. 3: Summary Statistics.

Since we are dealing with timeseries, is interesting to find out whether these timeseries are stationary or not. Stationary timeseries are typically easier to make forecast for and there is many approaches to deal with those (such ARIMA). In order to find out, we run a Dickey-Fuller Test[9] where the null hypothesis is that the timeseries is non-stationary. If the test statistic is lower (considering sign) than some critical values for different confidence intervals, the null hypothesis can be rejected. The results of the test applied to the household power consumption for year 2016 are in Fig. 4.



```
Results of Dickey-Fuller Test:
Test Statistic                    -5.434701
p-value                            0.000003
#Lags Used                        32.000000
Number of Observations Used     8751.000000
Critical Value (1%)               -3.431097
Critical Value (10%)              -2.566946
Critical Value (5%)               -2.861870
dtype: float64
```

Fig. 4: Results of Dickey-Fuller Test.

Since the Test Statistic is lower than any of the Critical Values (for different levels of confidence), the can reject the null hypothesis (non-stationarity) with 99 % of confidence. Therefore, the household power consumption are stationary timeseries.

Now the analysis is focused on how the different features (such as historical consumption and weather) affect the variable we are trying to predict. In order to measure the relationship between the target variable and the features, the Pearson Correlation Coefficient is used[10]. This test measures the linear correlation between two vectors of length 24 (since our target is a prediction of 24 floating point numbers). The test performed measure the Pearson coefficient for the 366 days in 2016. Then, the average of this coefficient as well as a box plot is presented. The following test were performed:

[9] Dickey, D. G. (2011). Dickey-Fuller Tests. In *International Encyclopedia of Statistical Science* (pp. 385-388). Springer Berlin Heidelberg.

[10] https://onlinecourses.science.psu.edu/stat501/node/256

- Correlation between Power Consumption at day ‘i’ and day ‘i-1’: 0.348

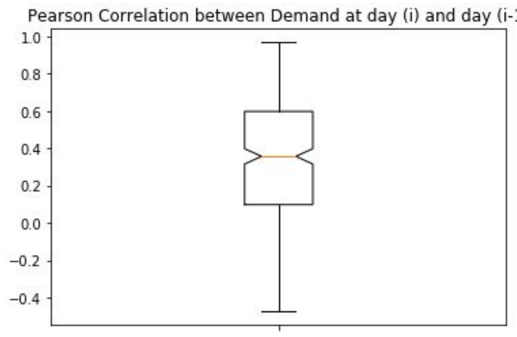


Fig. 5: Pearson Correlation between Demand at day (i) and day (i-1).

- Correlation between Power Consumption at day ‘i’ and day ‘i-2’: 0.296

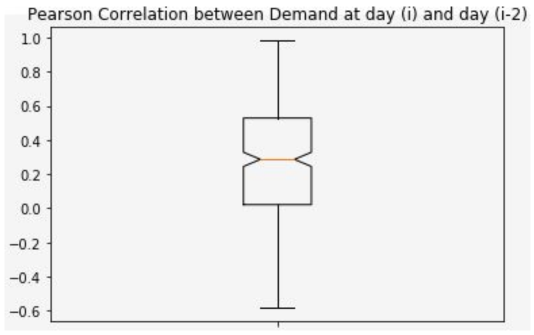


Fig. 6: Pearson Correlation between Demand at day (i) and day (i-2).

- Correlation between Power Consumption at day ‘i’ and day ‘i-7’: 0.347

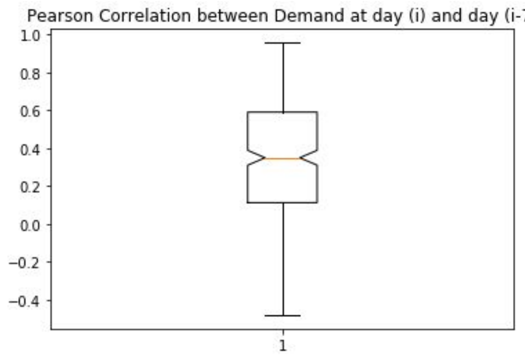


Fig. 7: Pearson Correlation between Demand at day (i) and day (i-7).

- Correlation between Power Consumption at day ‘i’ and Temperature at day ‘i’: 0.272

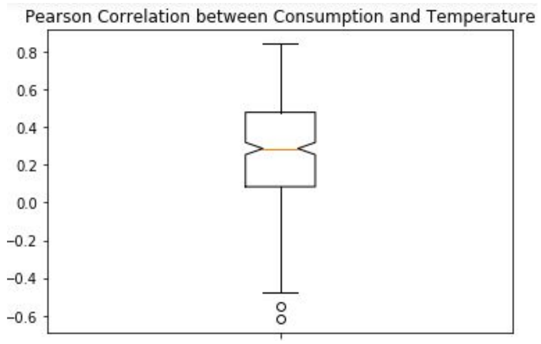


Fig. 8: Pearson Correlation between Demand and Temperature.

- Correlation between Power Consumption at day ‘i’ and Humidity at day ‘i’: -0.272

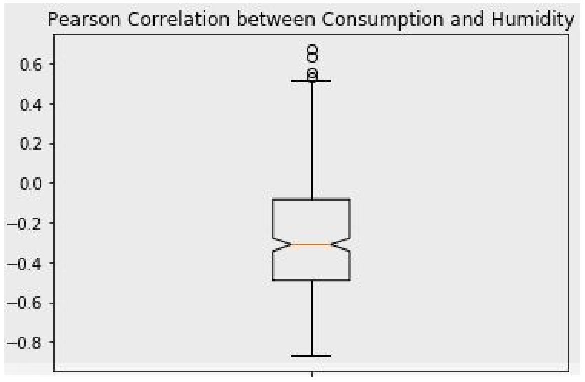


Fig. 9: Pearson Correlation between Demand and Humidity.

Based on this study we could say that the variables affecting more the Power Consumption of this household are the consumption the day before and one week before. On the other hand, temperature and humidity are also important with positive and negative effects, respectively. The effect of the weather can also be shown studying the annual behaviour of the variables at hand. The plot in Fig. 10 shows the daily average of electricity consumption, temperature, and humidity (properly scaled so they can be overlap, making comparison easier) across the year 2016. In this plot, we can observe a clear correlation between the trend in temperature and electricity consumption. During the hottest seasons, there is a higher energy consumption on average per day. This probably due to the HVAC system cooling down the household. On the other hand, the relationship between electricity consumption and humidity is less clear. The humidity mean is highly volatile except during the summer.

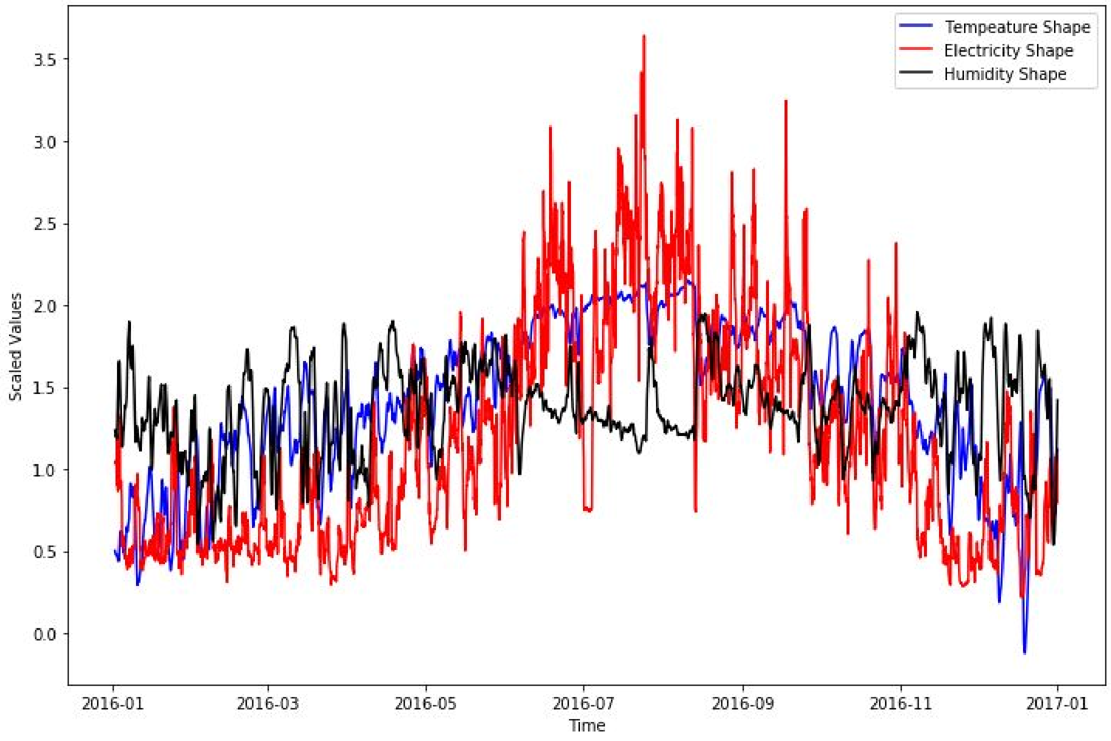
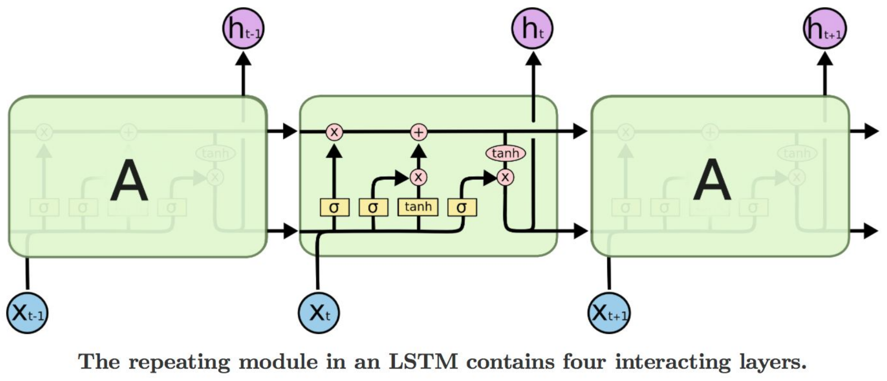


Fig. 10: Scaled Daily Average of Electricity Consumption, Temperature and Humidity.

Algorithms and Techniques

Deep Learning techniques are applied to solve this problem. Due to the sequential component of the data (timeseries), a Long Short Term Memory (LSTM) Recurrent Neural Network[11] is used. The input to the network is historical power consumption as well as the local weather. In order for this model to work, the input data has to be scaled to values between -1.0 and 1.0. On the other hand, the output will be T values corresponding to the consumption (in kW) for the next T timesteps where the same type of scaling has to be applied. Fig. 11 shows how a single LSTM cell works. For each timestep the output is determined by the state of the cell, that is information from previous timesteps, as well as from the features at the current timestep.



h_{t-1}
h_t
h_{t+1}
A
A
tanh
σ
σ
tanh
σ
x_{t-1}
x_t
x_{t+1}
The repeating module in an LSTM contains four interacting layers.

Fig. 11: LSTM Cell Temporal Diagram.

To add more complexity to the network and be able to represent a higher amount of information, more LSTM cells can be stacked together. Following the example in the image above, a new LSTM cell could be placed so h(t) for the first cells is x(t) for the second cell, and so on.

Benchmark

As benchmark Auto-Regressive Integrated Moving Average (ARIMA) model is used. ARIMA[12] has been extensively used in the literature to provide forecast in univariate timeseries. In the same way as with RNN, the output will be the 24 values corresponding to the power consumption (in kW) for the next 24 hours. In this case, the input is reduced only to the

[11] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.

[12] Marinescu, A., Harris, C., Dusparic, I., Clarke, S., & Cahill, V. (2013, May). Residential electrical demand forecasting in very small scale: An evaluation of forecasting methods. In *Software Engineering Challenges for the Smart Grid (SE4SG), 2013 2nd International Workshop on* (pp. 25-32). IEEE.

historical power consumption of the household. This is because ARIMA only uses values extracted from the timeseries[13]. ARIMA has three parameters (p, d, q) that need to be discovered in order to fit an appropriate model. This parameters correspond to the following components of ARIMA:

- Auto-Regression (p): this means that the forecast value x(t) is linear combination of x(t-1), …, x(t-p). Then the parameter p is related to how many previous values the model is using. Since we are trying to predict 24 timesteps ahead, we set p as 24.
- Integration (d): this parameter refers to how many differentiations are made to the original data. Usually this is done when the timeseries are non-stationary. Since we have proven above that this is, indeed, an stationary timeseries, we set d as 0.
- Moving-Average (q): this means that the forecast value x(t) is a linear combination of past errors e(t-1), …, e(t-q). Typically the value q is set to the “lag” for which the Auto-Correlation Function (ACF) is 0 (given a confidence interval). Fig. 12 shows the ACF for the series. Since this value is never 0 in our timeseries, for simplicity we set q to 0.

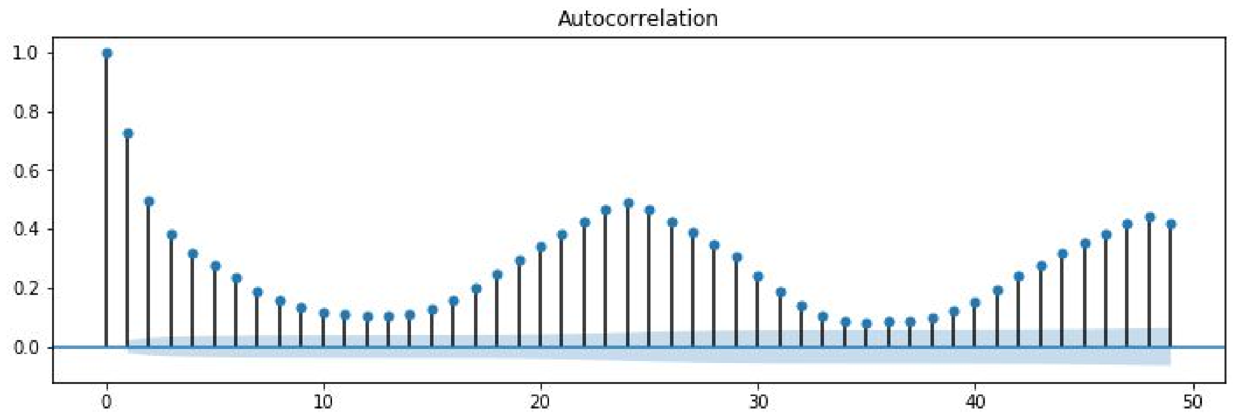


Fig. 12: Auto-Correlation Function.

Therefore, we have a final model that is basically and Auto-Regresive model of order 24. The results yielded by the ARIMA(24, 0, 0) model are:

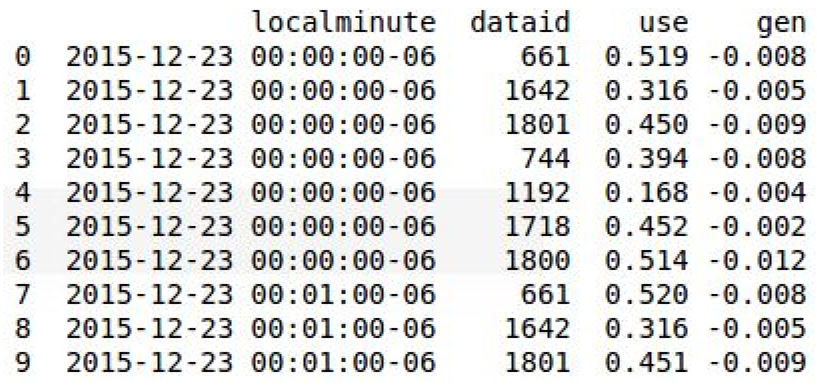
- Mean Squared Error: 2.47
- Mean Absolute Error: 1.04

[13] https://onlinecourses.science.psu.edu/stat510/node/48

METHODOLOGY

Data Preprocessing

As indicated above, the original data, came from heterogeneous data sources and in different formats. Energy data was given for several households with 1 minute resolution as in Fig. 13. In order to get a usable dataset, we randomly select a household that we use in the whole project and select the data for that single “dataid”. Furthermore, since we are only interested in power consumption and not generation we can delete the column “gen” that is part of the dataset. Then in order to get hourly data, the function “resample(60T).mean()” is applied to the dataframe.

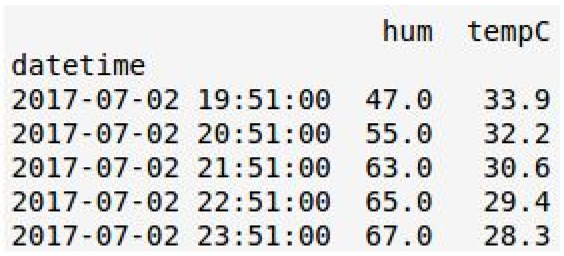


```
                localminute  dataid    use    gen
0  2015-12-23 00:00:00-06     661  0.519 -0.008
1  2015-12-23 00:00:00-06    1642  0.316 -0.005
2  2015-12-23 00:00:00-06    1801  0.450 -0.009
3  2015-12-23 00:00:00-06     744  0.394 -0.008
4  2015-12-23 00:00:00-06    1192  0.168 -0.004
5  2015-12-23 00:00:00-06    1718  0.452 -0.002
6  2015-12-23 00:00:00-06    1800  0.514 -0.012
7  2015-12-23 00:01:00-06     661  0.520 -0.008
8  2015-12-23 00:01:00-06    1642  0.316 -0.005
9  2015-12-23 00:01:00-06    1801  0.451 -0.009
```

Fig. 13: Original Energy Dataframe.

On the other hand, weather data was given with 1 hour resolution but with arbitrary timestamps (e.g. data was collected at minute 51 every hour) as in Fig. 14. In order to obtain hourly data centered at minute 0 and second 0, the following operations are performed:

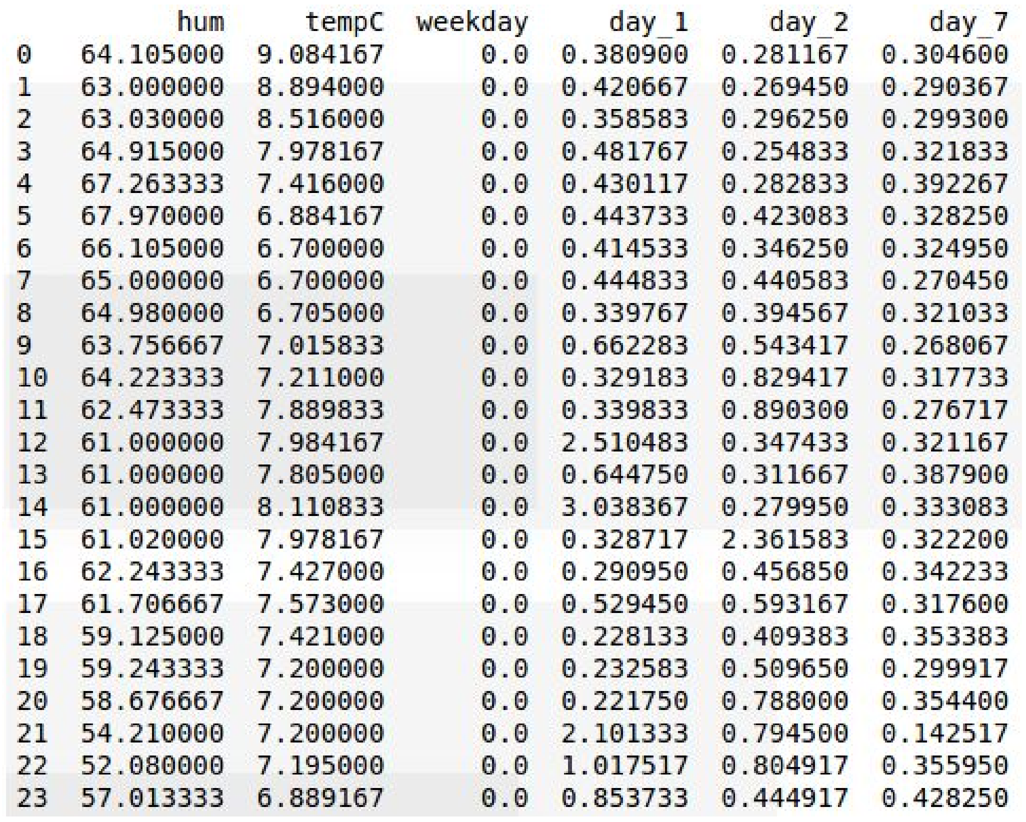
- Data is interpolated to obtain 1 minute data using the “interpolate()” function.
- Data is resampled to obtain hourly data centered at the same minutes and seconds as the energy data.



```
                      hum  tempC
datetime
2017-07-02 19:51:00  47.0   33.9
2017-07-02 20:51:00  55.0   32.2
2017-07-02 21:51:00  63.0   30.6
2017-07-02 22:51:00  65.0   29.4
2017-07-02 23:51:00  67.0   28.3
```

Fig. 14: Original Weather Dataframe.

Another type of preprocessing consist in separating the training and testing datasets. This is done by applying the “loc()” function to the dataframes. Finally, while the ARIMA model does not need data preprocessing as we can feed the data directly to the model, there is a preprocessing step in order to build the input for the Deep Learning model. The LSTM-based model takes input with shape “number of timesteps” x “number of features”. Therefore, the dataset has to be processed to meet this constraint. The “number of timesteps” is always 24. However, two different set of features were tried out on the model, one with 5 features and one with 6. Furthermore, in order to come up with an optimized Deep Neural Network architecture, a validation set is constructed based on the original training set (year 2016). Due to seasonal variations of the output variable through the year, the validation set is constructed with first 7 days of each month. Finally, the data has is scaled to values between 0 and 1 using the “MinMaxScaler” from Scikit learn package. An example of the input to the neural net without the final scaling is shown in Fig. 15.

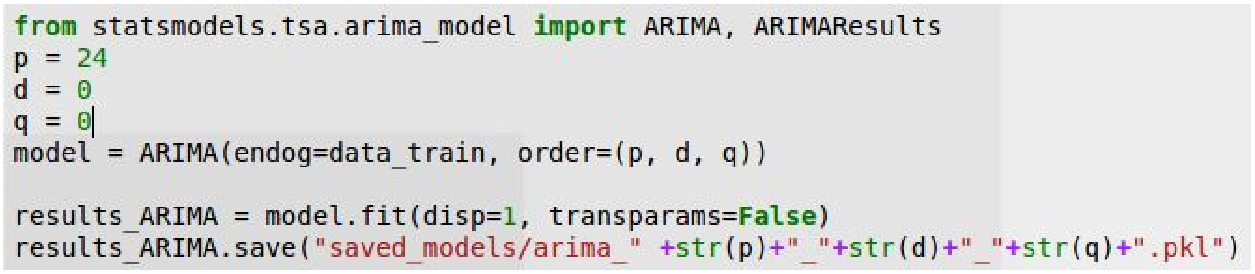


```
          hum     tempC  weekday     day_1     day_2     day_7
0   64.105000  9.084167      0.0  0.380900  0.281167  0.304600
1   63.000000  8.894000      0.0  0.420667  0.269450  0.290367
2   63.030000  8.516000      0.0  0.358583  0.296250  0.299300
3   64.915000  7.978167      0.0  0.481767  0.254833  0.321833
4   67.263333  7.416000      0.0  0.430117  0.282833  0.392267
5   67.970000  6.884167      0.0  0.443733  0.423083  0.328250
6   66.105000  6.700000      0.0  0.414533  0.346250  0.324950
7   65.000000  6.700000      0.0  0.444833  0.440583  0.270450
8   64.980000  6.705000      0.0  0.339767  0.394567  0.321033
9   63.756667  7.015833      0.0  0.662283  0.543417  0.268067
10  64.223333  7.211000      0.0  0.329183  0.829417  0.317733
11  62.473333  7.889833      0.0  0.339833  0.890300  0.276717
12  61.000000  7.984167      0.0  2.510483  0.347433  0.321167
13  61.000000  7.805000      0.0  0.644750  0.311667  0.387900
14  61.000000  8.110833      0.0  3.038367  0.279950  0.333083
15  61.020000  7.978167      0.0  0.328717  2.361583  0.322200
16  62.243333  7.427000      0.0  0.290950  0.456850  0.342233
17  61.706667  7.573000      0.0  0.529450  0.593167  0.317600
18  59.125000  7.421000      0.0  0.228133  0.409383  0.353383
19  59.243333  7.200000      0.0  0.232583  0.509650  0.299917
20  58.676667  7.200000      0.0  0.221750  0.788000  0.354400
21  54.210000  7.200000      0.0  2.101333  0.794500  0.142517
22  52.080000  7.195000      0.0  1.017517  0.804917  0.355950
23  57.013333  6.889167      0.0  0.853733  0.444917  0.428250
```

Fig. 15: Feature Matrix input to Deep Neural Network.

Implementation

The ARIMA[14] model was taken from the package "statsmodels.tsa.arima_model". The model was fit with the entire training set (year 2016) and persisted in memory using the "pickle" package.

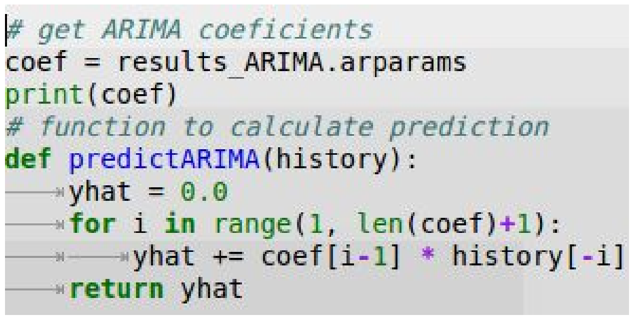


```
from statsmodels.tsa.arima_model import ARIMA, ARIMAResults
p = 24
d = 0
q = 0
model = ARIMA(endog=data_train, order=(p, d, q))

results_ARIMA = model.fit(disp=1, transparams=False)
results_ARIMA.save("saved_models/arima_" +str(p)+"_"+str(d)+"_"+str(q)+".pkl")
```

Fig. 16: Implementation of ARIMA model.

Making predictions with the ARIMA model was a little bit more complicated since this Python package does not offer a good solution to perform predictions with data that is not included in the model. For instance, if we want to predict the February 1st of 2017, we should have trained the model with data until January 31st of 2017. Following this approach would require a lot of refitting and extra computational cost. In order to solve this issue. I implemented my own function that by taking the coefficients of the ARIMA model, builds a prediction. This function is shown in Fig. 17.



```
# get ARIMA coeficients
coef = results_ARIMA.arparams
print(coef)
# function to calculate prediction
def predictARIMA(history):
    yhat = 0.0
    for i in range(1, len(coef)+1):
        yhat += coef[i-1] * history[-i]
    return yhat
```

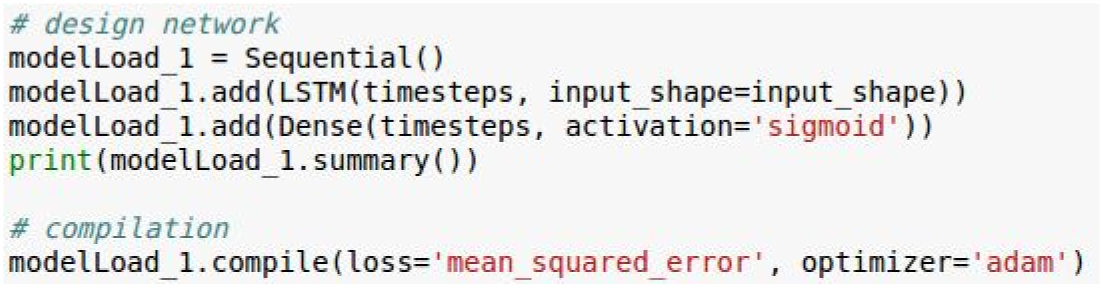
Fig. 17: Function to make predictions using trained ARIMA.

On the other hand, the LSTM models are constructed using the Keras[15] wrapper with TensorFlow[16] as back-end. For all of them the loss function to optimize is the Mean Squared Error and the training algorithm selected is ADAM method. It is worth mentioning that in order to stack LSTM cells the attribute "return sequence" has to be set to "True" to ensure that

[14] https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/
[15] https://keras.io/
[16] https://www.tensorflow.org/

information is forwarded appropriately between layers. The networks tried in this project are constructed as follows[17].

- 1 LSTM cell with fully connected layer:

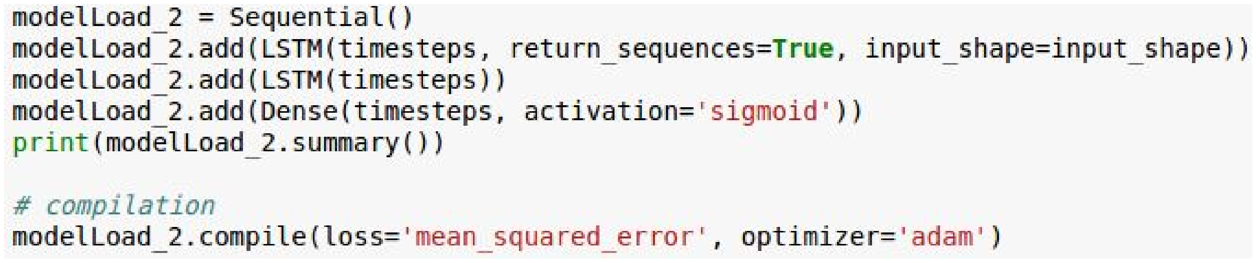


```
# design network
modelLoad_1 = Sequential()
modelLoad_1.add(LSTM(timesteps, input_shape=input_shape))
modelLoad_1.add(Dense(timesteps, activation='sigmoid'))
print(modelLoad_1.summary())

# compilation
modelLoad_1.compile(loss='mean_squared_error', optimizer='adam')
```

Fig. 18: Network Model with 1 LSTM cell and fully connected layer.

- 2 Stacked LSTM cells with fully connected layer:

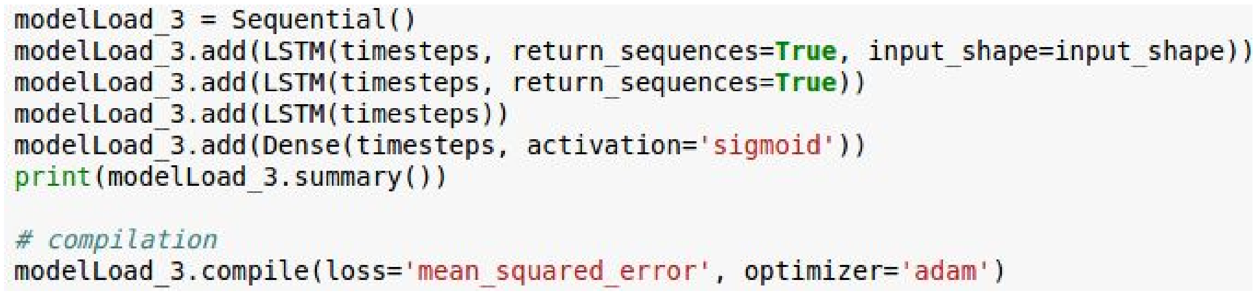


```
modelLoad_2 = Sequential()
modelLoad_2.add(LSTM(timesteps, return_sequences=True, input_shape=input_shape))
modelLoad_2.add(LSTM(timesteps))
modelLoad_2.add(Dense(timesteps, activation='sigmoid'))
print(modelLoad_2.summary())

# compilation
modelLoad_2.compile(loss='mean_squared_error', optimizer='adam')
```

Fig. 19: Network Model with 2 LSTM cells and fully connected layer.

- 3 Stacked LSTM cells with fully connected layer:

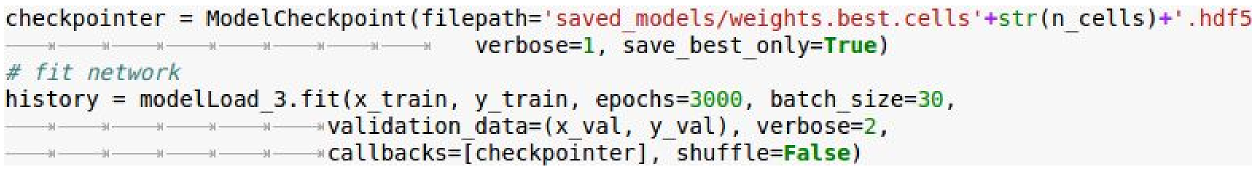


```
modelLoad_3 = Sequential()
modelLoad_3.add(LSTM(timesteps, return_sequences=True, input_shape=input_shape))
modelLoad_3.add(LSTM(timesteps, return_sequences=True))
modelLoad_3.add(LSTM(timesteps))
modelLoad_3.add(Dense(timesteps, activation='sigmoid'))
print(modelLoad_3.summary())

# compilation
modelLoad_3.compile(loss='mean_squared_error', optimizer='adam')
```

Fig. 20: Network Model with 3 LSTM cells and fully connected layer.

The code used for training and saving the best models on the validation set is shown in Fig. 21. To train, a batch size of 30 is selected for 3000 epochs.



```
checkpointer = ModelCheckpoint(filepath='saved_models/weights.best.cells'+str(n_cells)+'.hdf5
                                  verbose=1, save_best_only=True)
# fit network
history = modelLoad_3.fit(x_train, y_train, epochs=3000, batch_size=30,
                        validation_data=(x_val, y_val), verbose=2,
                        callbacks=[checkpointer], shuffle=False)
```

Fig. 21: Definition of training process.

17 https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/

Refinement

Initially, a 1 LSTM Cell model with 5 features (3 related to consumption in previous days and 2 weather features) was calculated. This model was improved adding a new categorical feature that adds information about human behavior (which highly influences power consumption). The addition of this feature led to an improvement in the validation set. Then, trying to improve the previous models several LSTM cells were stacked together. However, this new model did not performed better. Summing up, the best model was the second one with 6 features an a single LSTM Cell.

RESULTS

Model Evaluation and Validation

In this section, a summary with the main results are provided. The process followed to derive in the final model is explained in the previous section. The Mean Squared Error of the models are used to select the final model. The MSE of the models in the validation set with scaled features are the following:

- 1 LSTM Cell w/ 5 features. Scaled MSE: 0.00334
- 1 LSTM Cell w/ 6 features. Scaled MSE: 0.00261
- 2 LSTM Cell w/ 6 features. Scaled MSE: 0.00424
- 3 LSTM Cell w/ 6 features. Scaled MSE: 0.00453

Therefore, the model selected as final is the one using 6 features (5 numerical and 1 categorical) with a single LSTM cell. Then, this model is tried on the test set yielding the following results below. Furthermore, in order to further understand the quality of the model, the error distribution for each day in the testing set is also calculated and presented in the boxplot in Fig. 22.

- Scaled MSE: 0.00414
- MSE: 1.85
- MAE: 0.85

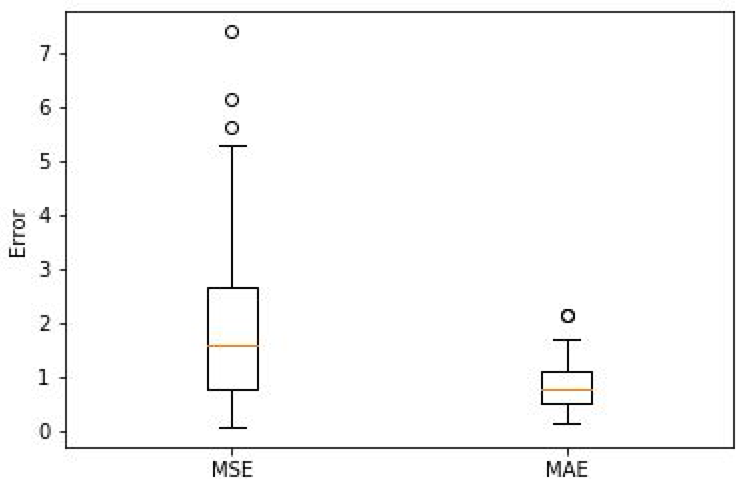


Fig. 22: Error Distribution across different days for Deep Learning model.

On the other hand, the ARIMA model's results on testing set are below together with the distribution of the errors for each day in the testing set.

- MSE: 2.47
- MAE: 1.04

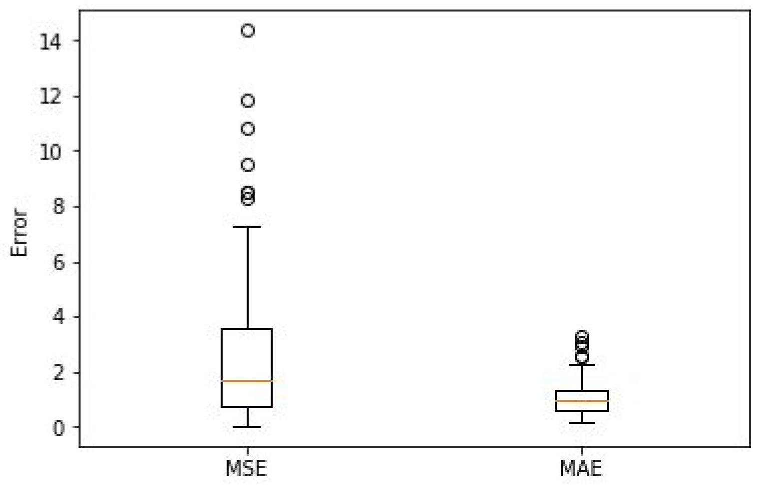


Fig. 23: Error Distribution across different days for ARIMA model.

As we can see, our model has improved the benchmark provided by ARIMA. Furthermore, it is worth mentioning that the scaled MSE calculated on the validation set and testing set are in the same order of magnitude, showing good generalization properties. On the other hand, we could say that the model performs good predictions in most days on the testing set (as shown in the in Fig. 23).

CONCLUSION

The project provides and end-to-end description on how to perform Residential Load Forecasting a day ahead. In this project we have provided a benchmark model constructed with ARIMA. Then, Deep Learning and specifically LSTM Networks have been used to provide an improved solution. The figure below shows the actual power consumption during a week for the household being studied. Here we can see how the Deep Learning approach (blue) clearly outperforms the ARIMA benchmark (red) at predicting the actual load (black).

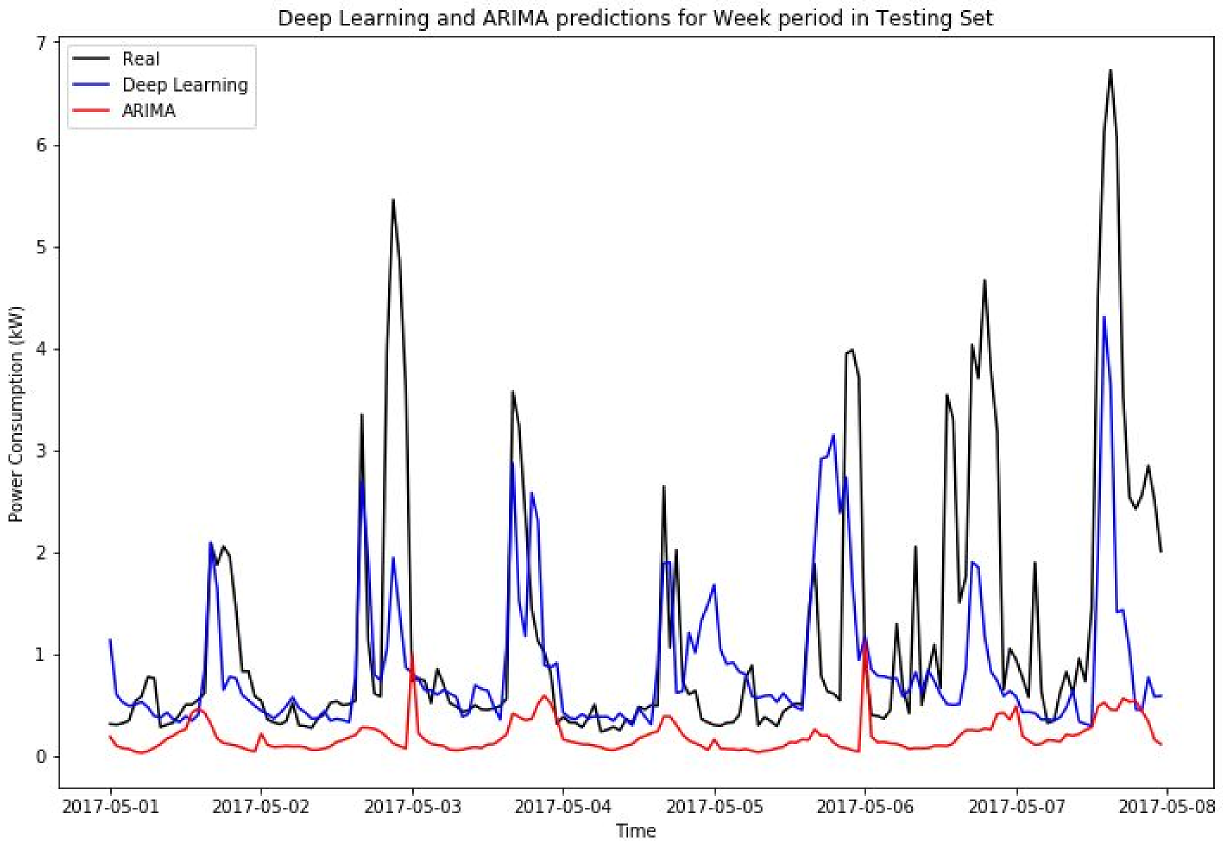


Fig. 24: LSTM and ARIMA Power Predictions (in kW) in comparion with Real Power.

Predicting the power consumption in a household a day ahead is a challenging task as this project shows. Human behavior highly influences consumption by being or not at home, feeling colder or warmer, etc. and it is a great source of stochasticity in the power consumption, that depends on many factors. In this project, we have incorporated some of the factors that affect consumption such as historical consumption information, weather data, and calendar data. However, more information could be added to get a more accurate prediction. For instance, it would be interesting to know how many people is on the household and at what time, when people like to do laundry, etc. In the future, systems such as Nest[18] from Google, could help improve these predictions by leverage all the user information that Google collect from users' smartphones (such as knowing when people is going to arrive at home).

[18] https://nest.com/es/