Time Series Prediction - Energy Consumption Implementation

Cohen Leonardo
lkoen@uth.gr

Peteinarelis Charalambos
pcharalampos@.uth.gr

Panagiotidis Ioannis
panioannis@uth.gr

Abstract

The purpose of this report is to establish a theoretical framework around the subject of time series forecasting and its applications, by reviewing popular Machine Learning approaches to the solution of this problem. Also, the application of Energy Consumption is examined in a practical way that features the implementation of Machine (Gradient Boosting, SVMs) and Deep (LSTM, Bi-directional LSTM, CNN-LSTM architectures) Learning methods, the display of the prediction results on a specific hourly energy consumption dataset and their comparison with respect to the overall Mean Squared Error.

1. Introduction

1.1. Definition

Time series forecasting is the process of analyzing time series data using statistics and modeling to make predictions and inform strategic decision-making. Time series forecasting occurs when one makes scientific predictions based on data where there is an ordered relationship between observations, and involves building models through historical analysis and using them to make observations and drive future strategic decision-making. It is not always an exact prediction, and the likelihood of forecasts can vary wildly—especially when dealing with the commonly fluctuating variables in time series data as well as factors outside our control.

However, forecasting gives us insight about which outcomes are more likely—or less likely—to occur than other potential outcomes. While forecasting and "prediction" generally mean the same thing, there is a notable distinction. In some industries, forecasting might refer to data at a specific future point in time, while prediction refers to future data in general. Time series forecasting is often used in conjunction with time series analysis. [1]

1.2. Problem Taxonomy

In order to better understand the problem at hand, we define a framework that can classify it to a certain category, taking into consideration several key points. As for our input variables, we have to determine whether they are endogenous or exogenous, i.e. whether they are - or are not, accordingly - influenced by other variables in the system and on which the output variable depends. Of the same importance is to detect any time-dependent patterns such as trends (monotonous increases or decreases), seasonality (periodical changes), cycles (long-term patterns that have a waveform and recurring nature similar to seasonal patterns but with variable length, but don't have a fixed time period.) or irregularities (Irregular components appear due to unexpected events, like cataclysms, or are simply representative of noise in the data) in the variable of interest, which, if present, would label our data as structured (versus unstructured).

As for the prediction, we have to determine whether it is a regression or classification task, whether it is univariate or multivariate (depending on how many variables we focus our prediction on), whether we perform a single-step or multi-step forecast (depending on the number of the future time steps we predict), and, finally, whether our model is static or dynamic, that is, if it is fit once and used to make predictions, or fit on newly available data prior to each prediction.

1.3. Common Time Series Forecasting Implementation

Demand forecasting for retail, procurement, and dynamic pricing: Predicting customer demand is a cornerstone task for businesses that manage supplies and procurement. Another common application is predicting prices and rates for products and services that dynamically adjust prices depending on demand and revenue targets.

Price prediction for customer-facing apps and better user experience: Price prediction in time series forecasting also produces great opportunities for improving and personalizing the user experience.

Forecasting pandemic spread, diagnosis, and medication planning in healthcare: Time series analysis and forecasting became the key technique applied in healthcare to predict the spread of Covid-19. It was used for transmission predictions, mortality ratios, the spread of the epidemic, and more.

Anomaly detection for fraud detection, cyber security, and predictive maintenance: Anomaly detection is one of the common machine learning tasks that looks for outliers in the way data points are normally distributed. While it doesn't necessarily has to be time series data, anomaly detection often goes hand in hand with it. Detecting anomalies in time series entails finding irregular spikes or valleys that significantly deviate from the way seasons and trends look.

Fraud detection is a business-critical activity for any industry dealing with payments and other financial operations. PayPal — an ML powerhouse when it comes to fraud detection — applies time series analysis to track the normal frequency of operations of each account to find irregular spikes in the number of transactions. These findings are also checked against other suspicious activities, like recent changes in the shipping address or massive funds withdrawal, to highlight a given transaction as likely being fraudulent.

Weather forecasting: In the past, the human forecaster was responsible for generating the entire weather forecast based upon available observations.Today, human input is generally confined to choosing a model based on various parameters, such as model biases and performance. Using a consensus of forecast models, as well as ensemble members of the various models, can help reduce forecast error. However, regardless how small the average error becomes with any individual system, large errors within any particular piece of guidance are still possible on any given model run.[2]

Astronomy: There is a need to examine how light is distributed in terms of position on the sky (images), wavelength or photon energy (spectroscopy), polarization, and time. Early examples of astronomical time series include observations of the positions of the stars and planets, the phases of the Moon and planets, and sunspot numbers. [8]

There are many more usages of time series forecasting in the fields of engineering and mathematics, in which they have to deal with random signals.

2. Machine learning Methods: Simple Forecasting, Autoregressive and Exponential Smoothing Methods

There is a vast variety of techniques that we can apply to a time series forecasting problem. Surprisingly enough, in many occasions the best option is to apply a relatively simplistic one, but even if that is not the case, simple forecasting methods are always useful as baseline indicators for the performance of the model we actually employ. A very popular simple forecasting method paradigm is the naive forecast. A naive forecast involves using the previous observation directly as the forecast without any change. It is often called the persistence forecast as the prior observation is persisted. This simple approach can be adjusted slightly for seasonal data. In this case, the observation at the same time in the previous cycle may be persisted instead [3]. One step above the naive forecast is the strategy of averaging prior values. All prior observations are collected and averaged, either using the mean or the median, with no other treatment to the data. In some cases, we may want to shorten the history used in the average calculation to the last few observations. We can generalize this to the case of testing each possible set of n-prior observations to be included into the average calculation [3].

More sophisticated methods include autoregression and exponential smoothing. Autoregressive models use a number of lag features (lagging will be explained later in this report, for now let us just treat them as past observations) and discover their relationship with the most recent observation. Popular autoregressive methods are ARIMA (AutoRegressive Integrated Moving Average) and its variation that can work with data that present seasonality, SARIMA (Seasonal Autoregressive Integrated Moving Average). ARIMA and SARIMA share some configurable hyperparameters, mentioned in literature as "trend elements" : (p: Trend autoregression order, d: Trend difference order, q: Trend moving average order).

Note that we can set any of these parameters to zero, which will make the model ignore the corresponding element and leave us with an (S)IMA, (S)ARMA, (S)ARI model. Now, the configuration of SARIMA introduces an additional set of seasonal elements: (P: Seasonal autoregressive order, D: Seasonal difference order, Q: Seasonal moving average order, m: the number of time steps for a single seasonal period), which are used in a similar way. Exponential smoothing functions in a similar way for univariate problems, by using lag features, but in this case the model explicitly uses an exponentially decreasing weight for past observations, thus making them weighted with a geometrically decreasing ratio. This weight, alpha, is called smoothing factor or smoothing coefficient, and controls the rate at which the influence of past observations decays, or, intuitively, how quickly the model "forgets" them.

3. Deep Learning Methods

3.1. Data Transformation: Unsupervised to Supervised

A crucial step that should not be overlooked when it comes to solving a time series forecasting problem is the transformation of this problem from unsupervised to supervised. This re-framing of the data is usually achieved through the sliding window technique. The unsupervised-to-supervised transition involves the use of prior time steps to predict the next time step and the formation of the data to samples, that is input-target pairs that consist of the variable

of interest at time steps from t_{n-w}, to tn and the variable at time step t_{n+1} accordingly. The main concept is to use the aforementioned previous time steps to predict the next one, and then compare our prediction with the actual value of the variable at time step t_{n+1} to find the error and train our model (hence supervised learning). This concept can be generalized to multivariate forecasting, but we refer to univariate forecasting for ease of comprehension. The values of the variable (or variables) at previous time steps are called lag features. All these values create a "window", a snapshot of the time series of length w which is moving, "sliding" one time step at a time as we make a prediction for the next time step and correct accordingly.

4. Our Problem: Energy Prediction

Considering that electricity consumption is a time-dependent attribute [6], the task of its prediction falls into the category of time series forecasting problems. While mid-term and long-term forecasting are proved to be particularly challenging areas, most machine learning methods can give satisfying results in short-term forecasting (hourly up to weekly predictions); but even when it comes to short-term forecasting, we need to acknowledge that electricity consumption is affected by many variables and therefore we need models that can live up to this complexity, a statement that makes ANNs seem as an appealing choice.

In the work of Nugaliyadde, Somaratne and Wong [6], it is referred that in an experimental comparison between ANN, Multiple Regression (MR), Genetic Programming (GP), DNN and SVM, the results showed higher performance for ANN. More specifically, the paper suggests and examines the use of RNNs (Recurrent Neural Networks) and LSTM (Long Short-Term Memory) networks, as the prevalent ANNs in sequential data processing that can detect co-relationships between neighboring data instances. The RNN recursively adds the past output from the previous time step to the current input, this way letting past observations affect future predictions. The LSTM network has a slightly more complex, hardware-like structure; it is an assembly of memory modules, cells, which in turn include three gates, input, forget, and output gate that give the network the ability to explicitly control which past observations will affect future predictions and which ones will be "forgotten".

5. Dataset

The dataset we used [5] is a time series dataset that contains the hourly energy consumption in MW in Ohio/Kentucky from 2012-12-31 01:00:00 to 2018-01-02 00:00:00 containing a total of 57739 entries. The measurements are provided from Duke Energy, an American electric power and natural gas holding company.

	Datetime	DEOK_MW
0	2012-12-31 01:00:00	2945.0
1	2012-12-31 02:00:00	2868.0
2	2012-12-31 03:00:00	2812.0
3	2012-12-31 04:00:00	2812.0
4	2012-12-31 05:00:00	2860.0
...	...	...
57734	2018-01-01 20:00:00	4426.0
57735	2018-01-01 21:00:00	4419.0
57736	2018-01-01 22:00:00	4355.0
57737	2018-01-01 23:00:00	4224.0
57738	2018-01-02 00:00:00	4100.0

57739 rows × 2 columns

Figure 1. Dataframe

6. Preparation

After minimal preprocessing, which included data normalization with the use of MinMaxScaler of scikit-learn and the extraction of year, month, day and hour features from the Datetime column, we moved on to applying and comparing models. Note that our goal is to forecast the energy consumption of the next time step, thus to perform a single-step prediction.

6.1. Lags

As mentioned in 3.1 we created Lags. Lags are very useful in time series analysis because of a phenomenon called autocorrelation, which is a tendency for the values within a time series to be correlated with previous copies of itself. We created 23 1-hour difference lags. Each lag has 1-hour difference from its previous lag. We named the first lag diff. The final training of the models performed with lags as features.

6.2. Split the Dataset and Reshape

After creating lag features and removed rows with null values the dataset had 24 columns and 57716 rows. We splitted the dataset for training and testing purposes without shuffling because time is crucial for generalisation. For train data we took 80% of the dataset and for test data the remaining 20%.

diff	lag_1	lag_2	...	lag_13	lag_14	lag_15	lag_16	lag_17	lag_18	lag_19	lag_20	lag_21	lag_22
-0.022918	-0.031732	-0.026003	...	0.009255	0.016307	0.002204	0.024240	0.025342	0.021375	0.010577	0.000000	-0.012340	-0.016968
0.016747	-0.022918	-0.031732	...	0.001763	0.009255	0.016307	0.002204	0.024240	0.025342	0.021375	0.010577	0.000000	-0.012340
-0.027325	0.016747	-0.022918	...	-0.007933	0.001763	0.009255	0.016307	0.002204	0.024240	0.025342	0.021375	0.010577	0.000000
-0.008594	-0.027325	0.016747	...	-0.008153	-0.007933	0.001763	0.009255	0.016307	0.002204	0.024240	0.025342	0.021375	0.010577
-0.006170	-0.008594	-0.027325	...	-0.001983	-0.008153	-0.007933	0.001763	0.009255	0.016307	0.002204	0.024240	0.025342	0.021375
...	...	...	...	...	...	...	...	...	...	...	...	...	...
0.007272	0.042530	0.069414	...	0.015866	0.011900	0.008594	-0.000881	-0.005509	-0.003526	-0.071397	-0.034817	-0.035699	-0.018070
-0.001543	0.007272	0.042530	...	0.017188	0.015866	0.011900	0.008594	-0.000881	-0.005509	-0.003526	-0.071397	-0.034817	-0.035699
-0.014103	-0.001543	0.007272	...	-0.003526	0.017188	0.015866	0.011900	0.008594	-0.000881	-0.005509	-0.003526	-0.071397	-0.034817
-0.028867	-0.014103	-0.001543	...	-0.006170	-0.003526	0.017188	0.015866	0.011900	0.008594	-0.000881	-0.005509	-0.003526	-0.071397
-0.027325	-0.028867	-0.014103	...	-0.005729	-0.006170	-0.003526	0.017188	0.015866	0.011900	0.008594	-0.000881	-0.005509	-0.003526

Figure 2. Dataframe with new lag columns

Then we reshaped our data in an input format acceptable by tensorflow. (batch_size, timesteps, features). The features field is the number of depended variables, in our case DEOK_MW column of the dataset. We want our models to look back 23 timesteps so the timesteps dimension is 23 and the batch_size dimension is the number of the rows of the dataset (57716)

7. Testing Methodology and Comparisons

We used MSE, using the inverse normalization of the data, to better indicate the Error. We tuned the hyperparameters of our deep neural nets and machine learning methods and used the best architectures to predict the last 20% of our dataset.

8. Extreme Gradient Boosting

We take up a weak learner and at each step, we add another weak learner to increase the performance and build a strong learner. This reduces the loss of the loss function. We iteratively add each model and compute the loss. The loss represents the error residuals(the difference between actual value and predicted value) and using this loss value the predictions are updated to minimize the residuals. In XGBoost the trees can have a varying number of terminal nodes and left weights of the trees that are calculated with less evidence are shrunk more heavily. Newton Boosting uses Newton-Raphson method of approximations which provides a direct route to the minima than gradient descent. The extra randomisation parameter can be used to reduce the correlation between the trees, as seen in the previous article, the lesser the correlation among classifiers, the better our ensemble of classifiers will turn out[4]. We used decision trees as our weak regressor with max depth 5 and 1000 estimators and eta 0.1 negative mean square error as our loss function. We evaluated the method on our test sample and it produced MSE = 57417.17825.

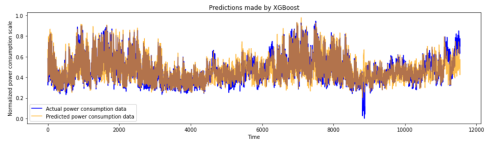

Figure 3. XGBoost Results on unknown data

9. SVM - Regressor

Support Vector Regression is a supervised learning algorithm that is used to predict discrete values. Support Vector Regression uses the same principle as the SVMs. The basic idea behind SVR is to find the best fit line. In SVR, the best fit line is the hyperplane that has the maximum number of points. Unlike other Regression models that try to minimize the error between the real and predicted value, the SVR tries to fit the best line within a threshold value. The threshold value is the distance between the hyperplane and boundary line.[7] The MSE that our SVR model produced is 87769.11935.

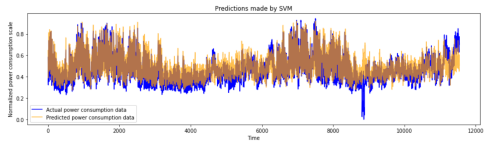

Figure 4. Svm Results

10. Deep Learning Architectures

We created three different models, a LSTM model, a CNN-LSTM model and a 2 bidirectional LSTM model. We used keras tuner to find the best hyperparameters for our models.

10.1. Keras Tuner

KerasTuner is an easy-to-use, scalable hyperparameter optimization framework that solves the pain points of hyperparameter search. You can easily configure your search space with a define-by-run syntax, then leverage one of the available search algorithms to find the best hyperparameter values for your models.

10.2. LSTM:

Long Short Term Memory Network is an advanced RNN(Recurrent Neural Network), a sequential network, that allows information to persist(Memory). It is capable of handling the vanishing gradient problem faced by RNN.

MSE: 81469.03227

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 256)               264192

 dense (Dense)               (None, 1)                 257

=================================================================
Total params: 264,449
Trainable params: 264,449
Non-trainable params: 0
_________________________________________________________________
```

Figure 5. Best LSTM Architecture

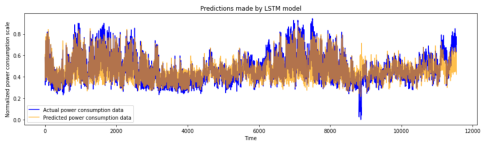

Figure 6. LSTM Results

10.3. CNN - LSTM:

Convolutional Neural Network (CNN) is an neural network which extracts or identifies a feature in a particular input. With 1D Convolutional Network we can extract features and then feed them to a LSTM to utilize the memory property.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 22, 32)            96

 max_pooling1d (MaxPooling1D  (None, 11, 32)           0
 )

 lstm (LSTM)                 (None, 256)               295936

 dense (Dense)               (None, 4)                 1028

 dense_1 (Dense)             (None, 1)                 5

=================================================================
Total params: 297,065
Trainable params: 297,065
Non-trainable params: 0
_________________________________________________________________
```

Figure 7. Best CNN - LSTM Architecture

MSE: 71961.3314

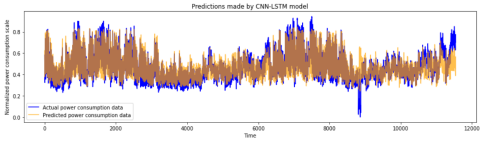

Figure 8. CNN - LSTM Results

10.4. Bidirectional LSTM:

Bidirectional LSTM are useful when we are dealing with long sequences of data and the model is required to learn relationship between future and past.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirectiona  (None, 23, 512)          528384
 l)

 bidirectional_1 (Bidirectio  (None, 512)              1574912
 nal)

 dense (Dense)               (None, 1)                 513

=================================================================
Total params: 2,103,809
Trainable params: 2,103,809
Non-trainable params: 0
_________________________________________________________________
```

Figure 9. Best Bi LSTM Architecture

MSE: 71862.7826

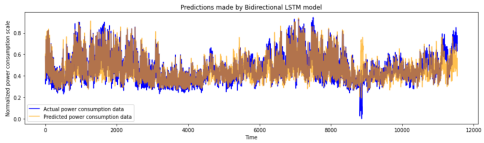

Figure 10. Bi LSTM Results

10.5. AutoML

model_id	rank	ensemble_weight	type	cost	duration
6	1	0.88	gradient_boosting	0.140368	11.861833
4	2	0.12	ard_regression	0.237577	26.405528

Figure 11. AutoMl Best Model Architecture

MSE: 60505.1557

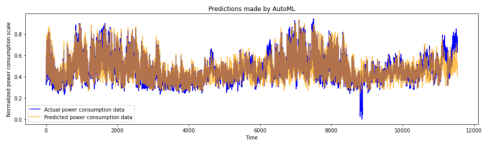

Figure 12. Auto Ml Results

11. Results

The best model was XGBoost with 1000 decision trees of max length 5, followed by the AutoML model. As we can see from the Figure 11 the worst model was SVM which MSE is 1.52 times larger than the MSE from the XGBoost model. Deep Learning models are in the middle of the barplot with MSE from 1.25 to 1.41 larger than the XGBoost's MSE. The AutoSklearn model consisted of an Gradient Boosting regressor with a weight of 0.88 and an ARD regressor with a weight of 0.12. We can see that even the automated AutoSklearn model converged to a gradient boosting model.

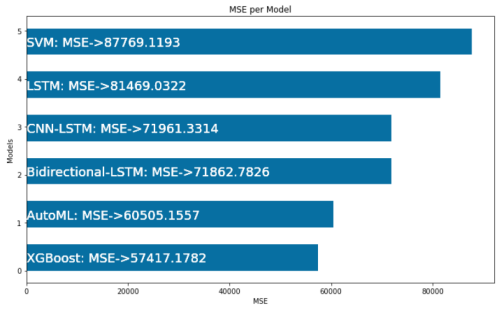

Figure 13. Comparison Results

12. Conclusions

When one is assigned to solve a time series forecasting problem, they are equipped with a particular yet diverse arsenal of Machine Learning methods. Despite being specific, this set of methods contains genuinely different approaches to the solution. Thus, it is of great importance to test and tune a vast variety of methods and compare the results to understand which solution fits better. As mentioned before, in such problems no method is a panacea, however at times there might be an obviously prevalent solution; even then, it is very useful to try other methods too, in order to have a frame of reference in terms of error and evaluation metrics.

This philosophy holds for all kinds of methods in forecasting, even in data preparation/preprocessing techniques: we ourselves noticed a great improvement in the total loss of the models when we produced and used lag features. This means that although correlating neighboring values might seem like a task that a neural architecture with memory would automatically do, the proper use of windowing (with window length equal to 23 hours in our case) really adds useful information to the prediction process.

References

[1] Time series forecasting: Definition, applications, and examples. *Tableau*.

[2] Weather forecasting. *Wikipedia*, Jun 2022.

[3] J. Brownlee. *Machine Learning Mastery With Python: Understand Your Data, Create Accurate Models, and Work Projects End-to-End*. Machine Learning Mastery, 2016.

[4] R. Gandhi. Gradient boosting and xgboost. *Medium*, May 2019.

[5] Msripooja. Hourly energy consumption time series rnn, lstm. *Kaggle*, May 2019.

[6] A. Nugaliyadde, U. Somaratne, and K. W. Wong. Predicting electricity consumption using deep recurrent neural networks. *arXiv preprint arXiv:1909.08182*, 2019.

[7] A. Raj. Unlocking the true power of support vector regression. *Medium*, Oct 2020.

[8] S. Vaughan. Random time series in astronomy. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, Feb 2013.