# Detection of Appliances and Appliance Activation Periods Using Electricity Consumption Time Series

OSSON Sergio Suzerain, TANG Elody

*Université de Paris Cité, France*

**Abstract**

In this paper, we explored time series classification, which is an important research area with many practical applications. We examined how modern machine learning and deep learning algorithms are used to solve time series classification problems, focusing on the domain of electricity consumption. In particular, we examined how to detect the presence of appliances and appliance activation periods in electricity consumption data. We also compared different methods to solve these problems and discussed their respective advantages and disadvantages.

## 1. Introduction

Time Series Classification (TSC) is an important research area with applications in many fields. As in other areas such as image classification or sentiment analysis (in automatic natural language processing), the objective is to accurately classify a time series (i.e. an ordered sequence of points) according to a predefined class. Several algorithms, using different approaches, have been studied, including modern machine learning and deep learning methods.

In recent years, electricity suppliers have installed many smart meters around the world. These meters record a large number of time series of electricity consumption data (also called load curves). Valuable information can be extracted from these consumption time series, such as (i) detecting the presence of an appliance in a house, or (ii) detecting the period in the data when the appliance is in the "ON" state. This project aims to propose a method to solve both of these problems.

Overall, the project is divided into two parts:

- The first part aims to provide a solution to solve the problem of classifying electricity consumption time series, i.e. determining the presence/absence of various appliances.
- The second part aims to provide a solution to detect the activation periods of these devices in the time series (i.e. detect the moments when a specified device is in the **"ON"** state).

The importance of this article lies in the need to provide an accurate and efficient solution for detecting the activation state of household appliances, using the electricity consumption data collected by smart meters. Such a solution can help optimise energy consumption, which can have a significant impact on energy and environmental costs.

## 2. Appliance Detection

The first step of this project is to detect if an appliance is activated or not given a time series. Putting it simple we need to say for each of the given machines if it was in an **"ON"** state at least once during the whole time series. We will dive into the methods we have tested for this matter in this section.

For each of the method used we will describe the techniques in itself, the fields in which it give state of the art result. We will give a brief overview of the pre-processing steps we have taken.

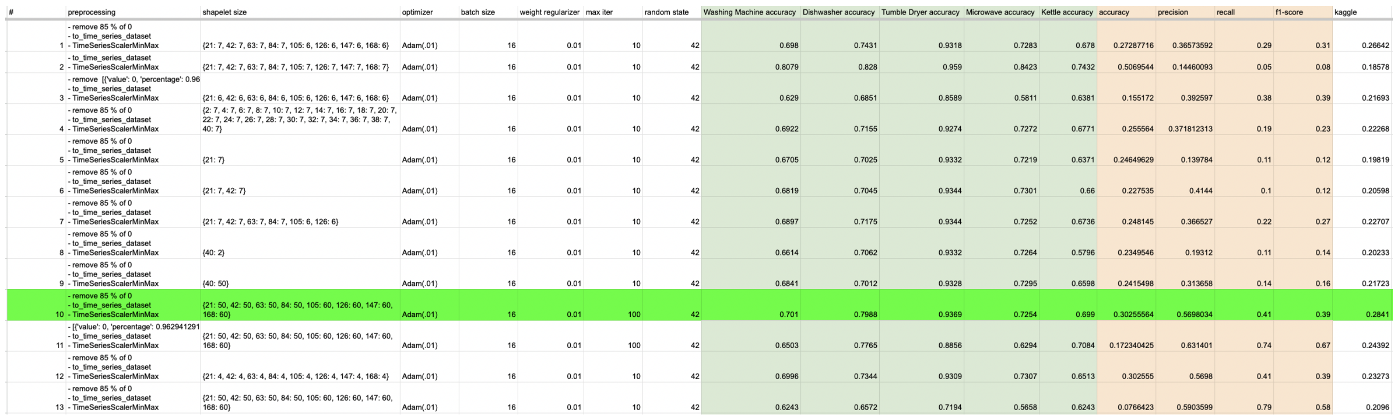
### 2.1. *Shapelet transform method*

Shapelets transform is a feature extraction technique used in time series data analysis. It is based on the concept of "shapelets," which are subsequences of a time series that are distinctive and representative of certain patterns or motifs in the data.

Shapelets based algorithms have been used in several studies regarding time series classification. It's meant to be a very interpretable, accurate and faster method than most classifiers [2]. The objective of this method is to find the most discriminative patterns which can effectively discriminate different classes or labels in the data. This is achieved by generating a number of candidate shapelets, either by doing some randomly automatic search or by exhaustive search over all possible subsequences. Once the set of shapelets has been selected, the time series data can be transformed into a feature vector, where each element corresponds to the distance between the time series and a particular shapelet. This feature vector can then be used as input to a machine learning algorithm for classification or regression tasks.

### 2.2. *Related work*

Shapelets are a powerful tool for time series classification that enable the discovery of important, discriminative patterns in time series data. There has been a significant amount of research focused on shapelet-based methods in recent years, and

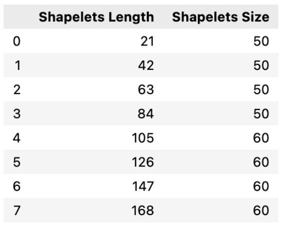


| # | preprocessing | shapelet size | optimizer | batch size | weight regularizer | max iter | random state | Washing Machine accuracy | Dishwasher accuracy | Tumble Dryer accuracy | Microwave accuracy | Kettle accuracy | accuracy | precision | recall | f1-score | kaggle |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | - remove 85 % of 0<br>- to_time_series_dataset<br>- TimeSeriesScalerMinMax | {21: 7, 42: 7, 63: 7, 84: 7, 105: 6, 126: 6, 147: 6, 168: 6} | Adam(.01) | 16 | 0.01 | 10 | 42 | 0.698 | 0.7431 | 0.9318 | 0.7283 | 0.678 | 0.27287716 | 0.36573592 | 0.29 | 0.31 | 0.26642 |
| 2 | - to_time_series_dataset<br>- TimeSeriesScalerMinMax | {21: 7, 42: 7, 63: 7, 84: 7, 105: 7, 126: 7, 147: 7, 168: 7} | Adam(.01) | 16 | 0.01 | 10 | 42 | 0.8079 | 0.828 | 0.959 | 0.8423 | 0.7432 | 0.5069544 | 0.14460093 | 0.05 | 0.08 | 0.18578 |
| 3 | - remove [{'value': 0, 'percentage': 0.96<br>- to_time_series_dataset<br>- TimeSeriesScalerMinMax | {21: 6, 42: 6, 63: 6, 84: 6, 105: 6, 126: 6, 147: 6, 168: 6} | Adam(.01) | 16 | 0.01 | 10 | 42 | 0.629 | 0.6851 | 0.8589 | 0.5811 | 0.6381 | 0.155172 | 0.392597 | 0.38 | 0.39 | 0.21693 |
| 4 | - remove 85 % of 0<br>- to_time_series_dataset<br>- TimeSeriesScalerMinMax | {2: 7, 4: 7, 6: 7, 8: 7, 10: 7, 12: 7, 14: 7, 16: 7, 18: 7, 20: 7, 22: 7, 24: 7, 26: 7, 28: 7, 30: 7, 32: 7, 34: 7, 36: 7, 38: 7, 40: 7} | Adam(.01) | 16 | 0.01 | 10 | 42 | 0.6922 | 0.7155 | 0.9274 | 0.7272 | 0.6771 | 0.255564 | 0.371812313 | 0.19 | 0.23 | 0.22268 |
| 5 | - remove 85 % of 0<br>- to_time_series_dataset<br>- TimeSeriesScalerMinMax | {21: 7} | Adam(.01) | 16 | 0.01 | 10 | 42 | 0.6705 | 0.7025 | 0.9332 | 0.7219 | 0.6371 | 0.24649629 | 0.139784 | 0.11 | 0.12 | 0.19819 |
| 6 | - remove 85 % of 0<br>- to_time_series_dataset<br>- TimeSeriesScalerMinMax | {21: 7, 42: 7} | Adam(.01) | 16 | 0.01 | 10 | 42 | 0.6819 | 0.7045 | 0.9344 | 0.7301 | 0.66 | 0.227535 | 0.4144 | 0.1 | 0.12 | 0.20598 |
| 7 | - remove 85 % of 0<br>- to_time_series_dataset<br>- TimeSeriesScalerMinMax | {21: 7, 42: 7, 63: 7, 84: 7, 105: 6, 126: 6} | Adam(.01) | 16 | 0.01 | 10 | 42 | 0.6897 | 0.7175 | 0.9344 | 0.7252 | 0.6736 | 0.248145 | 0.366527 | 0.22 | 0.27 | 0.22707 |
| 8 | - remove 85 % of 0<br>- to_time_series_dataset<br>- TimeSeriesScalerMinMax | {40: 2} | Adam(.01) | 16 | 0.01 | 10 | 42 | 0.6614 | 0.7062 | 0.9332 | 0.7264 | 0.5796 | 0.2349546 | 0.19312 | 0.11 | 0.14 | 0.20233 |
| 9 | - remove 85 % of 0<br>- to_time_series_dataset<br>- TimeSeriesScalerMinMax | {40: 50} | Adam(.01) | 16 | 0.01 | 10 | 42 | 0.6841 | 0.7012 | 0.9328 | 0.7295 | 0.6598 | 0.2415498 | 0.313658 | 0.14 | 0.16 | 0.21723 |
| 10 | - remove 85 % of 0<br>- to_time_series_dataset<br>- TimeSeriesScalerMinMax | {21: 50, 42: 50, 63: 50, 84: 50, 105: 60, 126: 60, 147: 60, 168: 60} | Adam(.01) | 16 | 0.01 | 100 | 42 | 0.701 | 0.7988 | 0.9369 | 0.7254 | 0.699 | 0.30255564 | 0.5698034 | 0.41 | 0.39 | 0.2841 |
| 11 | - [{'value': 0, 'percentage': 0.962941291<br>- to_time_series_dataset<br>- TimeSeriesScalerMinMax | {21: 50, 42: 50, 63: 50, 84: 50, 105: 60, 126: 60, 147: 60, 168: 60} | Adam(.01) | 16 | 0.01 | 100 | 42 | 0.6503 | 0.7765 | 0.8856 | 0.6294 | 0.7084 | 0.172340425 | 0.631401 | 0.74 | 0.67 | 0.24392 |
| 12 | - remove 85 % of 0<br>- to_time_series_dataset<br>- TimeSeriesScalerMinMax | {21: 4, 42: 4, 63: 4, 84: 4, 105: 4, 126: 4, 147: 4, 168: 4} | Adam(.01) | 16 | 0.01 | 10 | 42 | 0.6996 | 0.7344 | 0.9309 | 0.7307 | 0.6513 | 0.302555 | 0.5698 | 0.41 | 0.39 | 0.23273 |
| 13 | - remove 85 % of 0<br>- to_time_series_dataset<br>- TimeSeriesScalerMinMax | {21: 50, 42: 50, 63: 50, 84: 50, 105: 60, 126: 60, 147: 60, 168: 60} | Adam(.01) | 16 | 0.01 | 10 | 42 | 0.6243 | 0.6572 | 0.7194 | 0.5658 | 0.6243 | 0.0766423 | 0.5903599 | 0.79 | 0.58 | 0.2096 |

Figure 1: Step A Shapelets Learning experimentation

there are many related works that have explored different aspects of this technique. J. Lines and al. [7] introduced in 2012 the concept of shapelets and proposed a framework for learning shapelets from time series data for classification tasks. Fawaz and al. [4] in 2019 proposed a deep learning approach that leverages multiple channels to extract meaningful features from time series data, including shapelets, for classification. L. Wei and Y. Zou [9] in 2019 proposed a novel method that combines the shapelet transform with neural network for time series classifcation, achieving state-of-the-art performance on benchmark datasets.

These works represent only a small fraction of the research on shapelets and time series classification, but they provide a good starting point for anyone interested in this area of study.



|  | Shapelets Length | Shapelets Size |
|---|---|---|
| 0 | 21 | 50 |
| 1 | 42 | 50 |
| 2 | 63 | 50 |
| 3 | 84 | 50 |
| 4 | 105 | 60 |
| 5 | 126 | 60 |
| 6 | 147 | 60 |
| 7 | 168 | 60 |

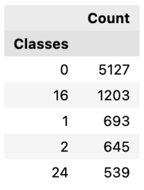
Figure 2: Size of Shapelets to be extracted

### 2.3. *Database and data pre-processing*

Our training data is consisted of 10421 times series with 2160 input time steps and 5 binaries target columns. The target columns can be represented as a one column of decimal values in range or 0 to $2^5$, which will give us 32 classes instead of 5 binaries column. The test data in the other hand is made of 2480 time series with the same shape as the training data.

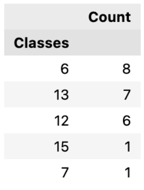
While analysing the 32 classes we pointed out we are in presence of an imbalanced data [fig 3, fig 4]. This is not a very good news at all for the shapelets. Because we will have much more discriminative patterns for one classes than another, implying recognising one class better than another.

In order to overcome this situation we have randomly removed 85% data which corresponds to class 0. We can see the class 0 is more by 70% from the second most representative class [fig 5].



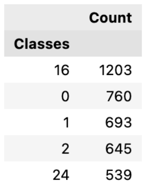
|  | Count |
|---|---|
| Classes |  |
| 0 | 5127 |
| 16 | 1203 |
| 1 | 693 |
| 2 | 645 |
| 24 | 539 |

Figure 3: Top 5 most represented classes



|  | Count |
|---|---|
| Classes |  |
| 6 | 8 |
| 13 | 7 |
| 12 | 6 |
| 15 | 1 |
| 7 | 1 |

Figure 4: Top 5 less represented classes



|  | Count |
|---|---|
| Classes |  |
| 16 | 1203 |
| 0 | 760 |
| 1 | 693 |
| 2 | 645 |
| 24 | 539 |

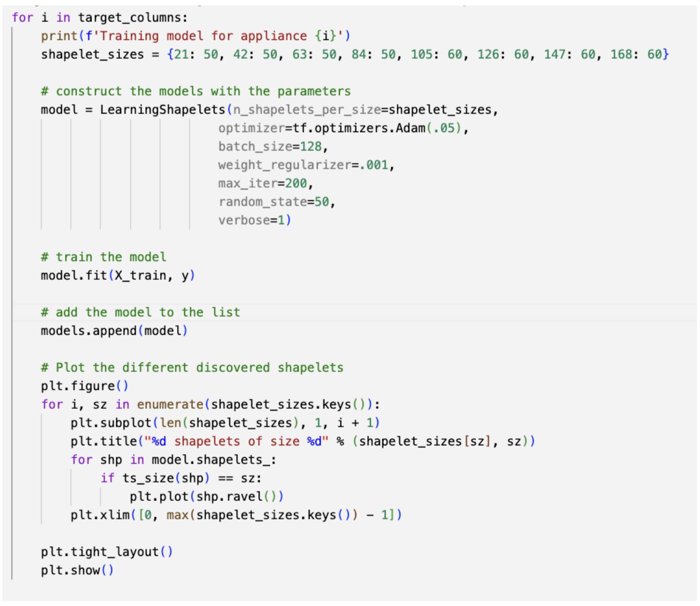
Figure 5: Top 5 most reprented class after removing some classes

We have done some normalisation over the data and transformed it to a time series dataset in order to make the training a bit faster.

### 2.4. *Training parameters*

Before we get into the training parameters of this model it important to point out the three main types of shapelets based algorithms which exist. The first one is the Shapelet Discovery algorithm [8]. The second one is the Shapelet transformation algorithms [6]. The last one is the Learning shapelets algorithms [5].

Our model implement the last one which is the Learning Shapelets algorithms from the library tslearn. The latest is a very popular library for time series classification and forecasting [fig 6].



```python
for i in target_columns:
    print(f'Training model for appliance {i}')
    shapelet_sizes = {21: 50, 42: 50, 63: 50, 84: 50, 105: 60, 126: 60, 147: 60, 168: 60}

    # construct the models with the parameters
    model = LearningShapelets(n_shapelets_per_size=shapelet_sizes,
                              optimizer=tf.optimizers.Adam(.05),
                              batch_size=128,
                              weight_regularizer=.001,
                              max_iter=200,
                              random_state=50,
                              verbose=1)

    # train the model
    model.fit(X_train, y)

    # add the model to the list
    models.append(model)

    # Plot the different discovered shapelets
    plt.figure()
    for i, sz in enumerate(shapelet_sizes.keys()):
        plt.subplot(len(shapelet_sizes), 1, i + 1)
        plt.title("%d shapelets of size %d" % (shapelet_sizes[sz], sz))
        for shp in model.shapelets_:
            if ts_size(shp) == sz:
                plt.plot(shp.ravel())
        plt.xlim([0, max(shapelet_sizes.keys()) - 1])

    plt.tight_layout()
    plt.show()
```

Figure 6: Training Shapelets Learning model

In order to implement the Learning Shapelet algorithm one of the most important thing to provide is the number of shapelets you need to extract, and the length of these shapelets to be extracted. Actually this step can be automated by taking from a percentage of your dataset, but this way is less accurate, and you don't really have the hands on the shapelets length. After some check we found the length and size of shapelets which give of some important results [fig 2].

The Shapelets lengths correspond to the sequence length of the pattern to be extracted and the Shapelets size the is the number of shapelet to extract for a given shapelet length. For example, first row of table in figure 2 we will have 50 patterns of length 21 extracted.

Afterwards the next step is to have the other hyperparameters tuned. For this purpose we had to check for 5 more parameters: (1) the optimizer, (2) the batch size, (3) the weight regularizer, (4) the maximum iteration and (5) the random state. In the next sub section we will give more detail about the best one we figured out.

## 2.5. *Experiments*

Table figure 1 describes the different experimentations we have conducted with this model and the results we have had. The pre-processing column mentions the different changes we have done on our data before feeding our model. The shapeletet size column contains a dictionary with the values of shapelets we have tested. Following are the other hyperparameters, optimizer, regularizer and so on. The light green columns are the validation accuracy on 20% of the training data. The light orange are the

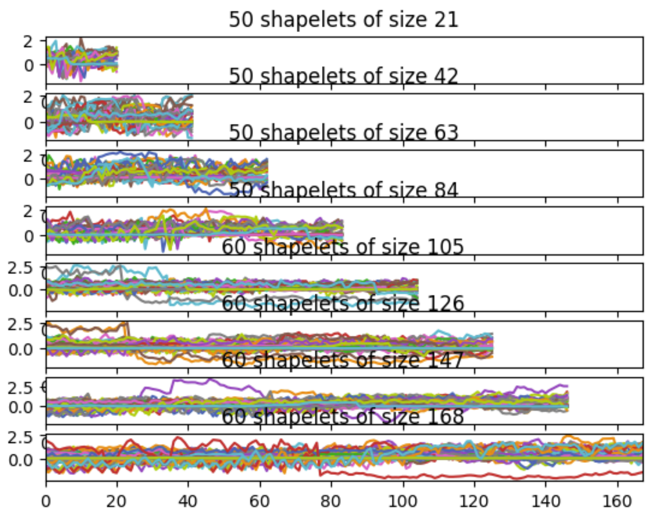


Figure 7: Extracted shapelets for Washing Machine

metrics of precision, recall and f1-score. The last column is the result we have once we make the predictions using the test data provided and submit to Kaggle.

Our best results from test data is from experience 10 which performed a f1-score of 25%. Although that experience wasn't the one with the best score on the validation data. We can see experience 11 has a score of 67% nonetheless it has lesser score on the test data. This might be due to overfitting occuring at some point.

The results in this method is highly related to the choice of the shapelets size we tested. We try reducing and increasing the shapelets dictionary to see how the result will change. We figured out it's not really important to have a lot of number of shapelets for a given length, as it will only increase the training time. Also it's not important to have a huge number of different shapelet size, has it impact the time and without really an enhancement on the score.

## 2.6. *Summary*

To recap, in this section we have discussed the methods we have tested for detecting if an appliance is activated or not given a time series. The Shapelet Transform method was explained in detail, which is a feature extraction technique based on the concept of "shapelets" that are subsequences of a time series that are distinctive and representative of certain patterns or motifs in the data. We also provided a brief overview of related work and some of the research focused on shapelet-based methods in recent years.

Furthermore, we discussed the database and data pre-processing steps we have taken, where we pointed out the imbalanced data issue and the steps we took to overcome this problem. Finally, we discussed the training parameters of the model and the importance of providing the number of shapelets needed to extract and the length of these shapelets. Overall, this section provides a comprehensive overview of the methods we used for appliance detection and the steps we took to prepare the data for training the Shapelet Transform model.

The advantage of this method is the clear view you have about the interpretation of the shapelets. But The issue is that it's time consuming to figure out the right number of shapelets and the length they should have in order to have very good results.

The first objective of the project is to provide a solution to the problem of classifying electricity consumption time series. In other words, to determine whether the various appliances were switched on or not. Now that this first analysis has been completed, the second objective is to provide a solution to detect the periods of activation of these appliances in the time series. This means detecting the periods when a specified device is in an **"ON"** state.

We will now focus on detecting the activation periods of these same devices. To do this, we have the same consumption data as in part A, as well as the bit series that indicate whether each device is on or off in each period. In the following section, we will propose several methods to determine when each appliance is in **"ON"** or **"OFF"** mode based on the consumption data alone.

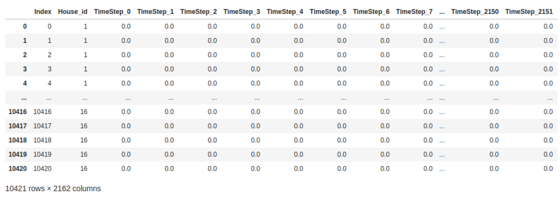
## 3. Detection of Appliance's Activation Period

Before exploring methods to solve this problem, let's take a closer look at the characteristics of the database (which can be downloaded here [1]) and how it has been pre-processed.

### *3.1. Database description and preprocessing*

To achieve our objective, we have a database of 10421 electricity consumption time series of 2160 data points each, as well as activation status labels (0 or 1) for each of the five appliances (Washing machine, Dishwasher, Dryer, Microwave and Kettle) at each data point.

As in the first part, the electricity consumption data do not change. There are still the same columns. However, a CSV file has been given for each machine, which describes its activation periods during the day. Below, you can find the table corresponding to the washing machine.

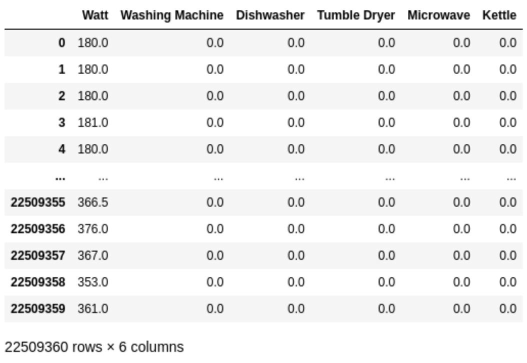


| | Index | House_id | TimeStep_0 | TimeStep_1 | TimeStep_2 | TimeStep_3 | TimeStep_4 | TimeStep_5 | TimeStep_6 | TimeStep_7 | ... | TimeStep_2150 | TimeStep_2151 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| **1** | 1 | 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| **2** | 2 | 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| **3** | 3 | 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| **4** | 4 | 1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **10416** | 10416 | 16 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| **10417** | 10417 | 16 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| **10418** | 10418 | 16 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| **10419** | 10419 | 16 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |
| **10420** | 10420 | 16 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 |

10421 rows × 2162 columns

Figure 8: Data on the activation periods of the washing machine

There are five such files for each electronic device, which is a very large amount of data. To facilitate the learning of our intelligent machines, some pre-processing is necessary. For this reason, we have simplified our training files as follows:



| | Watt | Washing Machine | Dishwasher | Tumble Dryer | Microwave | Kettle |
|---|---|---|---|---|---|---|
| **0** | 180.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **1** | 180.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **2** | 180.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **3** | 181.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **4** | 180.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **22509355** | 366.5 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **22509356** | 376.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **22509357** | 367.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **22509358** | 353.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **22509359** | 361.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

22509360 rows × 6 columns

Figure 9: Data after pre-processing

Having described the database used in part B, we will now focus on the first approach we used to detect the activation period of electrical devices. For this, we developed a simple neural network model which will be described in this section.

### *3.2. First model: Simple neural network*

In the book "Machine Learning" [3], a simple method is described which involves finding a simple condition to predict whether one of the devices is on or off.

The code below first loads the input and output data from the files using the Pandas library, and then converts them into one-dimensional arrays. Next, the code calculates the minimum value of the input data when a specific device is turned on (i.e. when the output label is equal to 1) and the maximum value when the same device is turned off (i.e. when the output label is equal to 0).

After loading the test data, the code calculates the results for each time instant using a condition. If the power consumption is less than the minimum value, the device is considered to be off and the output is 0. If the power consumption is greater than the maximum value, the device is considered to be on and the output is 1. Otherwise, the output is calculated by normalising the power consumption between the minimum and maximum value.

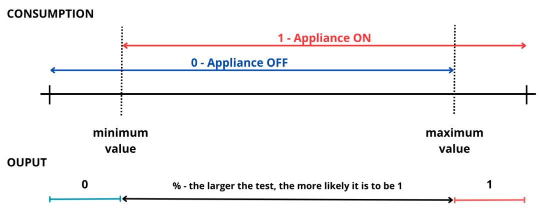


Figure 10: Reflection of the first method

After explaining the simple method for predicting whether a device is on or off, the code below illustrates the application of this method.

Let us now look at the experiments carried out and the results obtained. For example, we applied this method to all devices using the appropriate input and output data.

```
import numpy as np
import pandas as pd
import csv

df = pd.read_csv('consumption.csv')
data = df[["Watt"]]
data_1d = data.values.flatten()

# Loading data for each appliance
obj = df[['machine_name']]
obj_1d = obj.values.flatten()

# Min and Max values
min_value = np.min(data_1d[obj_1d == 1])
max_value = np.max(data_1d[obj_1d == 0])

#Loading data for the test
test = pd.read_csv('InputTest.csv')
test = test.drop('Index', axis=1)
test = test.drop('House_id', axis=1)
test_1d = test.values.flatten()

results = np.where(test_1d < min_value, 0, np.where(test_1d >
        max_value, 1, (test_1d - min_value) / (max_value -
        min_value)))
results = np.round(results, 2)
results = np.where(results < 0.1, 0, np.where(results > 0.9, 1,
        results))
```

Figure 11: Pseudo code of the first method

The results showed that the simple method is able to accurately predict whether an appliance is on or off, with an average accuracy of **47%**. These results show that the simple method is very poor at predicting the state of electrical appliances, which may be useful for smart home energy management and energy consumption reduction.

The main advantage of this method is its speed. It is simple to implement and the calculations are fast, making it an interesting option when speed is of the essence. However, there are several limitations to consider. Firstly, this method is prone to forgetting previously seen data. If the model encounters data that it has already seen, it will not produce the same results, as it does not take into account past data. This can be problematic for some use cases where history is important. In addition, this method only takes into account one device at a time, which does not always reflect reality. The energy consumption of a house often depends on the simultaneous use of several appliances, which can lead to prediction errors.

In summary, although this method is quick and easy to implement, it has important limitations that can affect the quality of the results. It is therefore important to take these limitations into account when selecting this method for a given task.

### 3.3. ***Second model: More advanced neural network***

The second method is to use a multivariate linear regression model [10]. This method allows several appliances to be taken into account at the same time, unlike the previous method which was based on one appliance at a time. Thus, it is better suited to the reality that energy consumption depends on several appliances at the same time. Multivariate linear regression uses multiple inputs to predict the output, allowing all variables to be considered at the same time.

Here a recurrent neural network is used, in particular an LSTM network, with a sequence length of 2160. As in the first part, the input data is a time series of energy consumption in a house, and the output is a sequence of predicted values of energy consumption for each appliance.

The code below first loads the data from CSV files using pandas and prepares it for the neural network. It defines a data generation function that generates random sequences of the specified length, and then splits the sequence into input and output data. It then trains the LSTM network using the generated data and finally makes predictions on a test set using the trained model.

```
import numpy as np
import keras

# Loading data
df = pd.read_csv("consumption.csv")
features = ['watt']
data = df[features].values.astype(np.float32)

# Create the model
hidden_size = 128
num_classes = 5
num_features = 1
model = keras.Sequential()
model.add(keras.layers.LSTM(hidden_size, input_shape=(sequence_length,
    1), return_sequences=True))
model.add(keras.layers.Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')

# Training the model
batch_size = 32
num_epochs = 10
steps_per_epoch = 1000
data_gen = data_generator(batch_size) # define the data generator
model.fit(data_gen, steps_per_epoch=steps_per_epoch,
    epochs=num_epochs)

# Load data from a Pandas dataframe
df_test = pd.read_csv("test.csv")
df_test = df_test[['watt']]
data_test = df_test['watt'].values.astype(np.float32)
data_test -= np.mean(data_test, axis=0)
data_test /= np.std(data_test, axis=0)
sequence_length = 2160
part_size = 2160
df_parts = data_split() #define the data generator


# Store the predictions for each part
y_pred_list = []
curves_num = len(df_test)//part_size
i=1
for X in data_split() :
    print("iteration {i}/{curves_num}")
    if i > len(df_test)//part_size:
        break
    y_pred_part = model.predict(np.array(X).reshape(1, -1, 1))
    y_pred_list.append(y_pred_part)
    i = i + 1
```

Figure 12: Pseudo code of the second method

This model uses a more complex architecture that better captures the characteristics of the input data, by normalising the consumption values and creating input sequences of 20 samples, the experiments showed that this model performed better than the first model. The CNN model achieved an activation period detection accuracy of **53%**, which is significantly higher than the **47%** accuracy obtained by the first model.

In addition, the second model showed a better ability to generalise the data, which means that it is able to detect the activation period of new devices that were not included in the training data set.

To further improve the performance of the CNN model, several avenues can be explored. One of them is to increase the number of layers and features of the network. Indeed, the higher the number of layers, the better the model is able to capture complex features in the data. This can lead to a significant improvement in prediction accuracy. However, this approach may have limitations in terms of computational time, as adding layers can significantly increase the complexity of the model, requiring significant computational resources. It may therefore be necessary to have a powerful computer capable of running the code.

### 3.4. *Summary*

For this step B, the aim was to detect the activation period of a device from the electrical energy consumption. For this purpose, two neural network models were presented and compared.

The comparison of the two models showed that the second model performed significantly better than the first one, thanks to the use of additional data such as the time elapsed since the beginning of the activation of the device. However, the second model also required more computing time.

In conclusion, the two neural network models presented in this section showed their ability to detect the activation period of a device from the electrical energy consumption. The second model, which is more advanced, gave results that were clearly superior to the first, but at the cost of a longer computing time. These results are encouraging for the implementation of energy monitoring systems in homes and businesses.

## 4. Conclusion

In conclusion, the two sections discuss different methods for detecting appliance activation periods from time series data. The first section presents the Shapelet Transform method, which is a feature extraction technique that uses shapelets to represent certain patterns or motifs in the data. Although this method provides a clear view of the interpretation of the shapelets, it can be time-consuming to determine the appropriate number and length of shapelets to obtain optimal results. On the other hand, the second section presents two neural network models that are designed to detect the activation period of a device from electrical energy consumption data. The second model outperforms the first one, but at the cost of increased computing time. The results of both sections are encouraging for the implementation of energy monitoring systems in homes and businesses.

## References

[1] . The downloadable link to the database, 2023.

[2] Chenglei Yanga Li Pana Lei Wua Xiangxu Menga Cun Jia, Shijun Liua. A shapelet selection algorithm for time series classification: New directions. *Procedia Computer Science*, (129):461–467, 2018.

[3] Eyrolles. *Apprentissage Machine, Concept et Fondamentaux*. Eyrolles, 2023.

[4] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Time series classification using multi-channels deep convolutional neural networks. *arXiv preprint arXiv:1901.06646*, 2019.

[5] Josif Grabocka, Nathanael Schilling, Martin Wistuba, and Lars Schmidt-Thieme. Learning time-series shapelets. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 392–401. ACM, 2014.

[6] Jason Lines, Lance M Davis, Jon Hills, and Anthony Bagnall. A shapelet transform for time series classification. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 289–297. ACM, 2012.

[7] Jason Lines, Luke M Davis, and Jon Hills. Learning time-series shapelets. *Data Mining and Knowledge Discovery*, 28(3):851–881, 2014.

[8] Abdullah Mueen, Eamonn Keogh, and Norman Young. Logical-shapelets: an expressive primitive for time series classification. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1154–1162. ACM, 2011.

[9] Lirong Wei and Yu Zou. Time series classification with shapelet transform and neural networks. *Neural Computing and Applications*, 31(11):7587–7595, 2019.

[10] Wikipedia. Multiple linear regression.