Using path signatures to generate individual appliance usage plots from whole-house data - project report

Student: Theodor-Mihai Iliant Supervisors : P. J. Moore and Yue Wu

Abstract

The path signature is a means of feature extraction from multivariate time series data that can be used as input in machine learning models. The objective of this study is to use it in order to be able to tell when an individual appliance is turned on or off, the only data provided being the whole-house monitoring voltage and current graphs, sampled 16kHz. In particular, I generate the usage plot of the fridge on a period of 67 hours. I avail myself of machine learning models such as random forest and convolutional neural networks adapted to time series data in order to train the models using past data where each individual appliance is monitored separately. Then, after having trained the model, I make predictions using the unseen whole-house monitoring data - that does not have alongside the individual appliance information - and make them input inside a segmentation algorithm to generate the individual appliance plots. The path signature is doing very well in terms of the accuracy measured as the ratio between the overlapping length of the true versus the predicted plots and the length of the whole interval I make predictions on.

1 Introduction

Nonintrusive appliance load monitoring is the idea of deducing what appliances are used in the house as well as their individual energy consumption, by analysing changes in voltage and current going into the house. The method was invented by George W. Hart, Ed Kern and Fred Schweppe of MIT in the early 1980s. Research into consumer behaviour suggests that consumers are best able to improve their energy efficiency when provided with itemised, appliance-by-appliance consumption information. If the house provided better feedback about which devices used the most energy then users could adjust their behaviour to make more efficient use of appliances [7]. Knowing which appliances consume most could be useful in industrial applications as well because one may see whether and when certain devices consume too much. A successful reduction in the energy consumption could lead to lots of money saved, because large businesses work on a bigger scale than a household. Producing less energy is better for the environment, too.

2 Methods and data

Paths are a fundamental way of describing time series data. There exist various models for time series prediction, such as the autoregressive integrated moving average model (ARIMA) or simple and double exponential smoothing. These models assume a specific distribution of the variables and there almost always exists the problem of the stationarity of the stochastic process. It is even harder to model multivariate time series data, that is, time series ordered by the same timestamps, because having way more parameters will make the models too complicated and prone to overfitting. The path signature is essentially a fixed-length vector which summarizes the path, whether it is one dimensional or not. If for paths of certain type I have got a corresponding output, I can try to learn a function from that vector to the output. Since the signature encodes all time dependant interactions, it can detect non-linear relationships efficiently, without neglecting the linear ones. The path signature was originally introduced by Chen [2] who applied it to piecewise smooth paths, and it was further developed by Lyons and others [1]. It is defined as follows:

Path signature

Let X be a continuous path in $\mathbb{R}^d$ defined as a function of time on $[a, b]$. (As an example, in our use case $d = 3$ and the dimensions are time, voltage and current). To emphasize the dependence on the time parameter, I can write

$$X_t = (X_t^1, X_t^2, ..., X_t^d)$$

Let us define what is means to integrate a path over another path. The Riemann Stieltjes integral of a real-valued function f with respect to another real-to-real function g is denoted by $\int_a^b f(x)\, dg(x)$. Its definition uses sequences of partitions $\{a = x_1 < x_2 < ... < x_n = b\}$. If the limit as the length of the longest subinterval approaches 0 of the following quantity $\sum_{i=0}^{n-1} f(c_i)(g(x_{i+1}) - g(x_i))$ exists, then the integral exists. The best simple existence theorem requires that f is continuous and that the graph of g as traveled along the y-axis, neglecting the contribution of the motion on the x-axis, has finite length, or in other words that g is of bounded variation [3].

We need to understand the idea of **iterated integrals**. Given an index i, the increment of the path X_t at time $a < t$ is $X_t^i - X_a^i = \int_{a<s<t} dX_s^i$, which I denote $S(X)_{a,t}^i$. In this notation, t is the only variable, so we can see that this is a function of t and thus can be treated as a path that can be integrated over another path. Extending this to the second order iterated integral, we can define the following, for any pair of indices (i, j), where i and j are not necessarily distinct. That is, we can also integrate a path over itself.

$$S(X)_{a,t}^{i,j} = \int_{a<s<t} S(X)_{a,s}^i dX_s^j = \int_{a<s<t} \int_{a<r<s} dX_r^i dX_s^j$$

We can extend this to the k-fold iterated integral, for indices $i_1, ..., i_k$:

$$S(X)_{a,t}^{i_1,...,i_k} = \int_{a<t_k<t} \int_{a<t_{k-1}<t_k} ... \int_{a<t_1<t_2} dX_{t_1}^{i_1} ... dX_{t_k}^{i_k}$$

Now we can arrange all these numbers into an infinite length vector:

$$S(X)_{a,b} = (1, S(X)_{a,b}^1, S(X)_{a,b}^2, S(X)_{a,b}^{1,1}, S(X)_{a,b}^{1,2}, ...)$$

$S(X)_{a,b}$ is a series of real numbers indexed by $\{(i_1, ..., i_k) |\ 1 \leq i_1, ..., i_k \leq d\}$. This is the path signature of X.

For each k there are exactly d^k terms that have exactly k superscripts. The n-th order of the signature will have exactly $1 + d + ... + d^n$ entries. Implementing signatures for machine learning is straightforward: a Python package called *iisignature* is provided in the Python Package index https://pypi.org/project/iisignature/ and its documentation explains how you can calculate the signatures from discrete data to get it ready for the machine learning framework. Note that this Python package does not include the first constant 1 entry, so the n-th order signature has $\frac{d(d^{n-1}-1)}{d-1}$ entries in that package. In practice, we need to truncate the signature for several reasons: the signature has infinite length, so in order to use it in machine learning models, only a finite number of features have to be kept and the rest discarded. In addition, only some of the features already selected may turn out to be predictive.

Signature properties

- In practice the signature length does not depend on the frequency with which the data is sampled. Even if we work with high frequency data, the signature is able to encode the

relationships between the variables in a fixed-length vector independent of the sampling frequency. We do not need to downsample the data to have fewer parameters in our models. I can safely gather high frequency data without worrying about a very complex model.

- The path signature determines a path up to tree-like equivalence [4]. In particular, the signature uniquely determines a path that has a monotonic dimension, such as time.
- Adding a constant vector to the path, that is a constant for each dimension, does not change the signature vector. In other words, the signature is invariant under translations.
- The signature is invariant under reparameterization. The speed at which the path is recorded does not change the signature. [5]

Log signature

There is enough motivation to introduce the log signature. For a 2-dimensional path $X : [a, b] \to \mathbb{R}^2$, the following hold:

$$S(X)^{1,1}_{a,b} = \frac{1}{2}\big(S(X)^1_{a,b}\big)^2$$

$$S(X)^1_{a,b} \cdot S(X)^2_{a,b} = S(X)^{1,2}_{a,b} + S(X)^{2,1}_{a,b}$$

Therefore, the signature presents some redundancy in its terms. If we want to use higher order signatures in our machine learning models, too many features can lead not only to a computationally difficult task, but also to a lot of features that turn out not to be predictive. Instead, a better approach is to use a smaller version of the signature, which captures the relationships between the variables just as well as the normal signature, this time by removing the redundancies between its features. This way, the correlation between the features is reduced, the number of features is reduced and I are more likely to find the features that are predictive. In what follows, I will always be using the log signature. For a more detailed explanation on how this is calculated, see this [6].

Signature degree	signature	log signature
2	12	6
3	39	14
4	120	32
5	363	80
6	1092	196
7	3279	508

Table 1: Number of features for a 3-dimensional path using signature and log signature for orders from 2 to 7. These Ire computed using *iisignature.siglength* and *iisignature.logsiglength*.

Example
Suppose we have a two dimensional path sampled discretely:

$$X = (X^1, X^2),\ X^1 = (1, 2, 3, 4),\ X^2 = (4, 1, 3, 1.5)$$

By linearly interpolating between two consecutive points, we can make this path continuous.

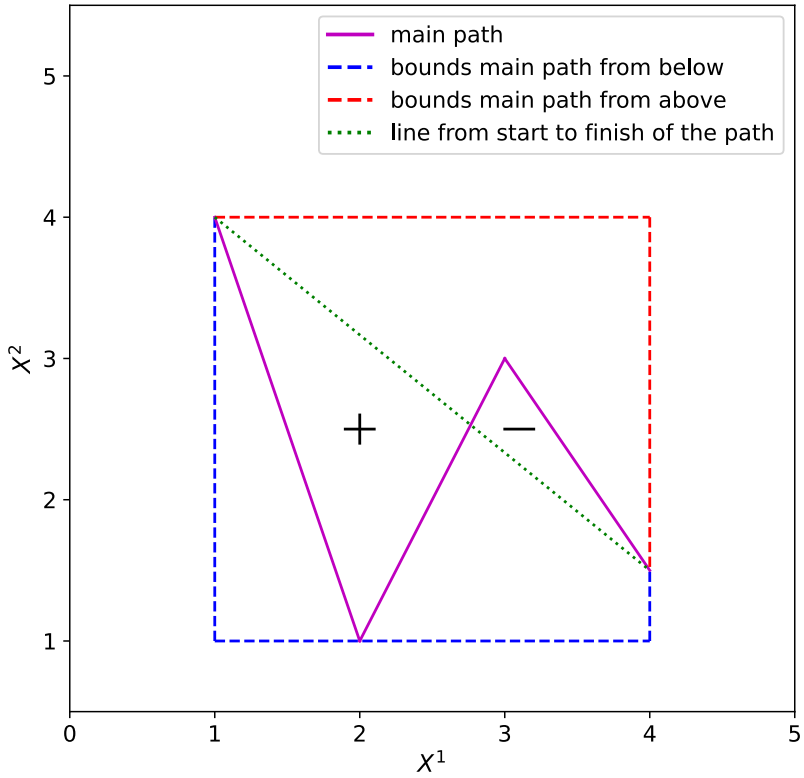

Figure 1: Plot of X^2 against X^1. The graph can be bounded above and below by a rectangle - in this case a square - of minimal area. Note that X^1 does not necessarily have to be a strictly increasing sequence. In this example, X^1 can be interpreted as time. The path is traversed in the order given by X^1.

We can identify some signature terms in the areas present in the figure. For example:

- The area between the path and the bottom part of the square is $S(X)^{2,1}_{1,4}$ and equals 3.75.
- The area between the path and the upper part of the square is $S(X)^{1,2}_{1,4}$ and equals 5.25.
- The Lévy area [6] is defined as the the addition of all the areas between the path and the green dotted line multiplied by 1 or -1, depending on whether the area between these is marked with a plus or a minus sign - if the path moves counterclockwise near the green line, then it is counted as positive, otherwise it is counted as negative. In this case it equals $S(X)^{1,2}_{1,4} - S(X)^{2,1}_{1,4}$.

Note that since $S(X)^1_{1,4} = 3, S(X)^2_{1,4} = 3$, I get $S(X)^1_{1,4} \cdot S(X)^2_{1,4} = S(X)^{1,2}_{1,4} + S(X)^{2,1}_{1,4}$, which is a particular case of the shuffle product [6].

A more detailed introduction to the path signature which also gives a geometric interpretation of the terms can be found in [6], [11]. However, higher order terms are not easy to interpret geometrically.

Data

The data provided is the UK-DALE dataset, which consists of appliance-level electricity demand from five UK homes [7]. Several attributes for a dataset for disaggregation include: simultaneously record the power drawn by most of the individual appliances in each house (this data can be used to train the system, validate the hyperparameters in order to be able to make predictions on the test dataset, which only includes whole-house signals; preferably sampled once every 10s or faster) and record the whole-house voltage and current signals at a very fine scale, like 16000 datapoints per second.

AC circuit terminology

In a simple alternating current (AC) circuit consisting of a source and a linear load, both the current and voltage are sinusoidal. The utility frequency, (power) line frequency (American English) or mains frequency (British English) is the nominal frequency of the oscillations of alternating current (AC) in a wide area synchronous grid transmitted from a power station to the end-user. In large parts of the world this is 50 Hz, although in the Americas and parts of Asia it is typically 60 Hz. Current usage by country or region is given in the list of mains electricity by country [9]. The portion of power that, averaged over a complete cycle of the AC waveform, results in net transfer of energy in one direction is known as *active power*. The active power is what I are going to work with. There are two other notions that exist and the following helps to avoid confusion: reactive and apparent power. The portion of power due to stored energy, which returns to the source in each cycle, is known as *reactive power*. *Apparent power* is the product of the rms values of voltage and current. Apparent power is taken into account when designing and operating power systems, because although the current associated with reactive power does no work at the load, it still must be supplied by the power source [8].

The following are the components of the UK-DALE dataset [7]:

6 second data

This data contains the active power measurements for each individual appliance, corresponding to a UNIX timestamp that can be converted into seconds. Data is sampled once every 6 seconds, although I found, by converting timestamps to seconds and taking the difference between two consecutive measurements, that there can also be 7 or 8 seconds between them.

1 second data

There are four columns in each CSV file recording the whole-house power demand every second:

- UNIX timestamp
- Active power (watts)
- Apparent power (volt-amperes)
- Mains RMS-voltage

All four columns record real numbers. The 1 second data is present as a mains.dat file in houses 1, 2, 5.

16kHz data

This is the most important part of the UK-DALE dataset. The 16 kHz data is compressed using the Free Lossless Audio Codec (FLAC). For houses 1, 2, and 5 UK-DALE records a stereo 16 kHz audio file of the whole-house current and voltage waveforms.The files are labelled vi-<T>.flac where

T is a real number recording the UNIX timestamp with micro-second precision - using an underscore as the decimal place. This timestamp is the time at which the audio file began recording. The recordings are split into hour-sized chunks. More details about how to convert values to voltage (volts) and current (amps) can be found in [7].

I only work with 14 days of the data from House 1, from February 16th 2015 to March 2nd 2015. There exist several approaches for using this data. In our use case, the 16kHz data contains 50 voltage cycles per second. The approach is the following: for each voltage cycle I take the corresponding current cycle. Each cycle will contain around 320 datapoints and is a fixed-length interval. The datapoints can be treated as a function of time. I then create a three dimensional path composed of time, voltage and current. Since I know the timestamp of the start of each flac file, I can find the exact timestamp of each datapoint in the path and hence each cycle is a function of the real time. I compute the signature of each such three dimensional path and this signature will be labelled as either 'on' or 'off', depending on whether the entire cycle belongs to a period of 'on' or 'off' of the fridge, respectively. To do this, I first identify the on/off periods of the fridge, and 'on' is when the power of the fridge is shown to be more than $20W$, according to the dataset. Because I want to have clean on/off periods in our labelled dataset, I take a 10 second margin on each side of the intervals - that is, I lose on average 25 seconds from each on/off interval of the fridge since the data is sampled once every 6 seconds. Since most of the fridge's 'on' periods are about 20 or 30 minutes, this does not reduce our training capacity, but in the end I do not compare our plots/results with the 6 second data, since the latter consists of more datapoints. The intervals I compare our predictions to are the intervals formed by using the labels column from the labelled dataset created by taking the 10 second margin, so the accuracy measured on them is more reliable. Those plots in the figures 9 and 13 are scaled so as to fill in the entire last 20% part of the 14 day period, even though they span at most one 90 minutes less. There is also asynchrony in the sense that between two consecutive cycles in the 16kHz data there is actually $\frac{1}{50} - \frac{\epsilon}{50}$ of a second, where ϵ is very small and varies, so given our downsampling of taking one cycle every 50, there really are $1 - \epsilon$ seconds between two consecutive rows of our dataset. These asynchronies do not reduce our capacity of interpreting the results of the models.

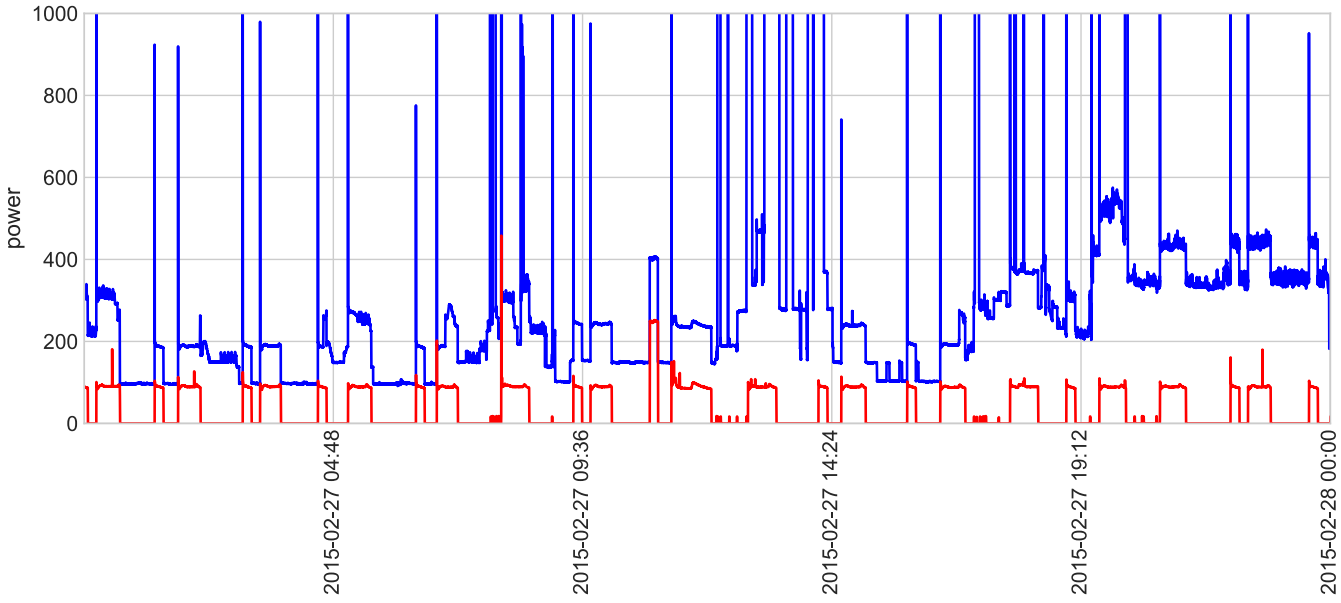

Figure 2: Fridge usage plot (red) over which I superimpose the total active power used (blue) throughout a day. The red plot is done using the individual appliance data, more specifically channel 12 from House 1. The blue plot is done using the mains data.

3 Results

I present three models that make extensive use of the data described above. Firstly, as an introduction, I solve a simple regression task using sklearn.ensemble.RandomForestRegressor to explicitly exploit the iterated integral definition of the signature and show how can its features encode properties of the path. I predict the average total mains poIr used during each voltage cycle. I do this using a downsampled version of our dataset, to 10 voltage cycles per second. Secondly, I train a random forest classifier using sklearn.ensemble.RandomForestClassifier during each voltage cycle: I predict whether the fridge is on or off during the time period of a voltage cycle - the time period is accompanied by the voltage cycle and the current cycle corresponding to it. That is, during $\frac{1}{50}$ of a second, I want to be able to say whether the fridge is on or off. This model is prone to a lot of false positives and false negatives because the time scale is too small. The latter motivates our last approach: I use a convolutional neural network with keras.layers.convolutional.conv1D to predict whether the fridge is entirely on or entirely off during periods of 42 seconds. I believe this approach will more accurately generate the individual fridge usage plots and indeed this is the case, as I will see later. The previous two models use a downsampled version of the dataset to 1 voltage cycle per second.

Segmentation algorithm [12]

I use this algorithm to generate the individual appliance plots by using the predictions that our models do on the testing data. The segmentation algorithm takes as input four things: the base interval, a boolean function that decides whether the algorithm should pass an interval and two types of precision. The basis of the algorithm is the notion of dyadic intervals. For a brief and sufficient description of dyadic intervals, see [13]. The algorithm divides the base interval into subintervals according to a precision. The first type of precision is a positive integer. For example, the precision of 11 means that the smallest subinterval that the segmentation algorithm is testing has length $\frac{1}{2048}$. The base interval is divided into two equal dyadic intervals of maximum length. The function decides whether each of these two intervals should pass. If not, it in turn divides the unpassed intervals into two smaller and equal dyadic intervals, each time reducing the length of the current interval by half. At each step, the function decides whether it should pass the interval. If, say, I have a passed interval and the algorithm finds another dyadic interval on the left side of it, then the function decides whether this bigger interval should be passed. If yes, the process continues. When the algorithm cannot append any other dyadic interval to the sides of the before passed interval, the segmentation algorithm - not the function, since the function has already passed the interval - has to decide whether this bigger interval should appear in the output. I call the "bigger" interval as the interval formed as a union of dyadic intervals by appending dyadic intervals on each side of the current interval as long as this is possible. This is where the other type of precision comes in: it is a positive integer and if it is 9, this means that the latter bigger interval is allowed to appear in the output only if its length is at least $\frac{1}{512}$. If the bigger interval has at least this length, then it will appear in the output of the algorithm and from that point on, each interval tested by the segmentation algorithm will be disjoint from it. In this way, the output of the algorithm is going to contain non-overlapping intervals that satisfy a certain requirement. Since the function is boolean, what is left of the base interval is really the set of intervals for which the function returns false. In our use case, the segmentation algorithm will return the intervals when the fridge is 'on', and so I immediately deduce when the fridge in 'off' according to our predictions, and that is how I produce the generated usage plots of the fridge. It is important to say that the accuracy for the generated plots is measured as the proportion of the *correctly* predicted intervals where the fridge is on *or* off. As an example, for a base interval $[0, 3]$, if the true intervals are $[0, 1.5]$: 'on', $[1.5, 3]$: 'off' and the predicted intervals are $[0, 1]$: 'on', $[1, 2]$: 'off', $[2, 3]$: 'on', then the correctly predicted intervals are only $[0, 1]$: 'on', $[1.5, 2]$: 'off' and so the accuracy is $\frac{1+0.5}{3} \cdot 100\% = 50\%$.

3.1 Random forest regressor

For this task, I need to know the average total active power used by the house during each voltage cycle, because the training model uses a downsampled version of the entire dataset to 5 cycles per second. Since I Ire only provided 1 second data for the active power, I must calculate the power using the 16kHz data. In essence, this is an approximation to the real power used during one voltage cycle but it is enough to illustrate the use of the path signature, because in what follows I explicitly use the integral definition of the signature to encode a feature which is characteristic to the power.

To calculate the average power, I must use an approximation to the true integral, since I know that the average power used from time 0 to time T is $\int_0^T V(t) \cdot I(t)dt$, where $V(t)$ and $I(t)$ measure the instantaneous voltage and current values, respectively. I want to approximate the mean, which is $\frac{1}{T}\int_0^T V(t) \cdot I(t)dt$. If I know that there are 320 datapoints evenly spaced in the interval of interest, say with indices i_k with k from 0 to 319, then a good approximation to the mean of the integral is $\frac{\frac{T}{320}}{T}\sum_{k=0}^{319} V(i_k) \cdot I(i_k) = \frac{1}{320}\sum_{k=0}^{319} V(i_k) \cdot I(i_k)$.

The motivation behind this task is really that if I integrate the current, I obtain the electric charge. Informally, if q is the charge, then $q = \int Idt$. I create a two dimensional path consisting of the voltage and the integral of the current. If I integrate the voltage path over the charge path, I will get the power, which is explained informally in the following two lines:

$$q = \int Idt \implies \frac{dq}{dt} = \frac{d}{dt}\int Idt = I \text{ (fundamental theorem of calculus)} \implies$$

$$\implies \int Vdq = \int V\frac{dq}{dt}dt = \int V \cdot Idt$$

Note that, informally:

$$\int_0^T Vdq = \int_{0<s<T}\int_{0<r<s} dVdq$$

,which is one of our 2nd order signature terms. Then one of our features will always consist of a number proportional to the power, which turns out to be a good predictor in the random forest regressor model.

Training and testing data

For a flac file from the 16kHz data, I choose the first 80% to be the traning data and leave the last 20% for the errors of the prediction measurements, which is calculated by rounding the mean relative error to two decimal places. I compute the log signature of order 2 - which only has 3 features - for each cycle in the dataset. One of the features is indeed proportional to the power and its feature importance rounded to two decimal places according to RandomForestRegressor.feature_importances_ is 100%. I did this task for 20 flac files from a single day - consisting of one hour of data each. For each flac file I took the mean of the relative errors of the predictions, so I get 20 representative numbers.

	Mean	Standard deviation
the 20 mean relative errors for the 20 different flac files	1.24%	1.42%

Table 2: The mean and standard deviation of the sequence of 20 mean relative errors for the predictions on each of 20 flac files belonging to the same day - consisting of one hour each.

3.2 Random forest classifier

Training, validation and testing data

- Training set: first 60% of the data - 741522 rows.
- Validation set: next 20% of data - 247174 rows. This is used to select the signature degree as Ill as the random forest hyperparameters and perform feature selection according to the importance ranking provided by RandomForestClassifier.feature_importances_.
- Testing set: last 20% of data - 247174 rows.

16kHz current graph before and after switch to on

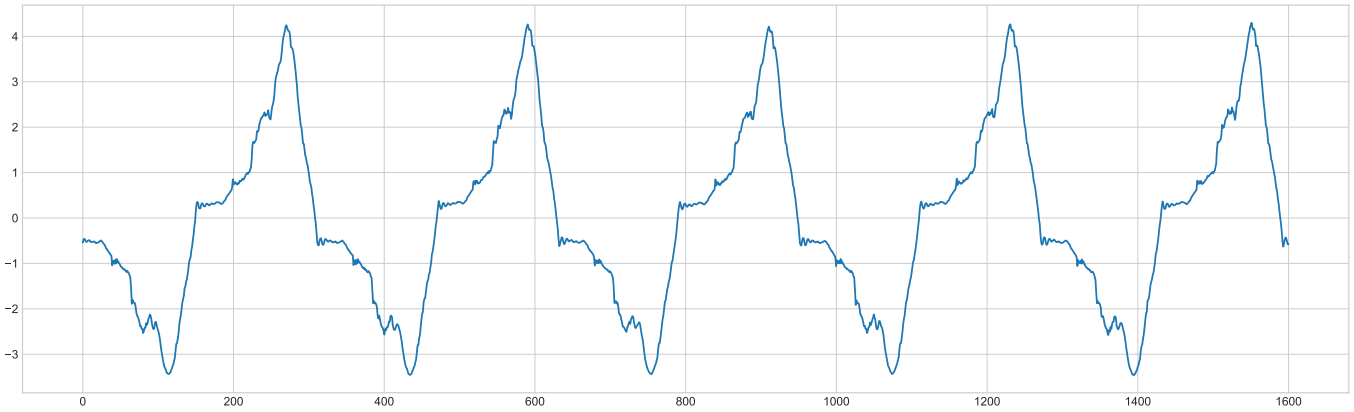

Figure 3: A tenth of a second long graph of the current from a flac file recording the hour 7 pm from February 27th 2015, just before the fridge was turned on. There are a lot of appliances turned on during this time, as I can see in Figure 1.

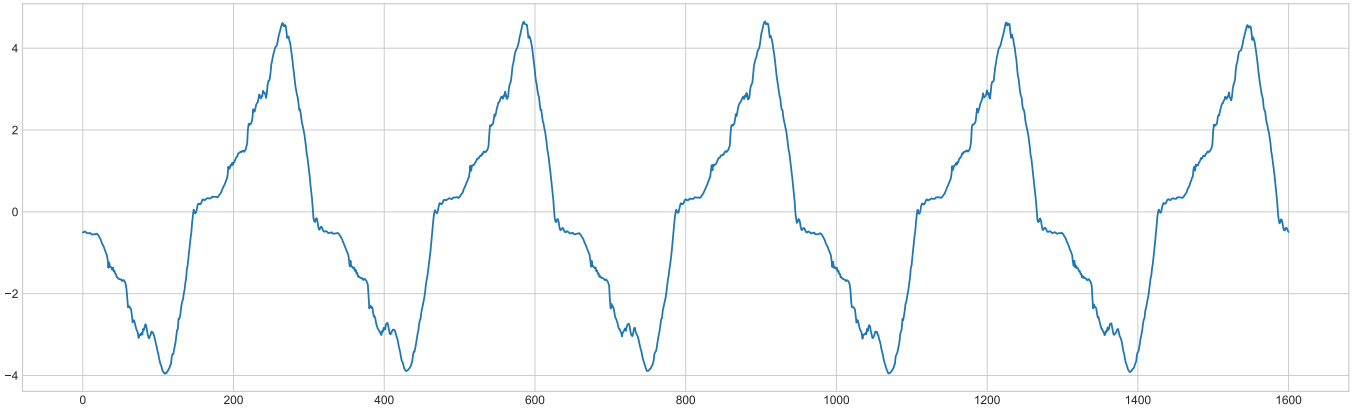

Figure 4: This plot is a tenth of a second long plot of the graph of the current, only one second after the fridge was turned on. That is, this plot is just 1 second ahead in time of the above plot.

Despite the difficulty in visually seeing any particular shapes that the fridge is having on the current plot when it is turned on versus just before that time - even if I are seeing a mere 0.1 second of data, the path signature is doing very Ill in learning these shapes, as I will see. In addition, its predictions are very accurate even on times when multiple appliances are turned on.

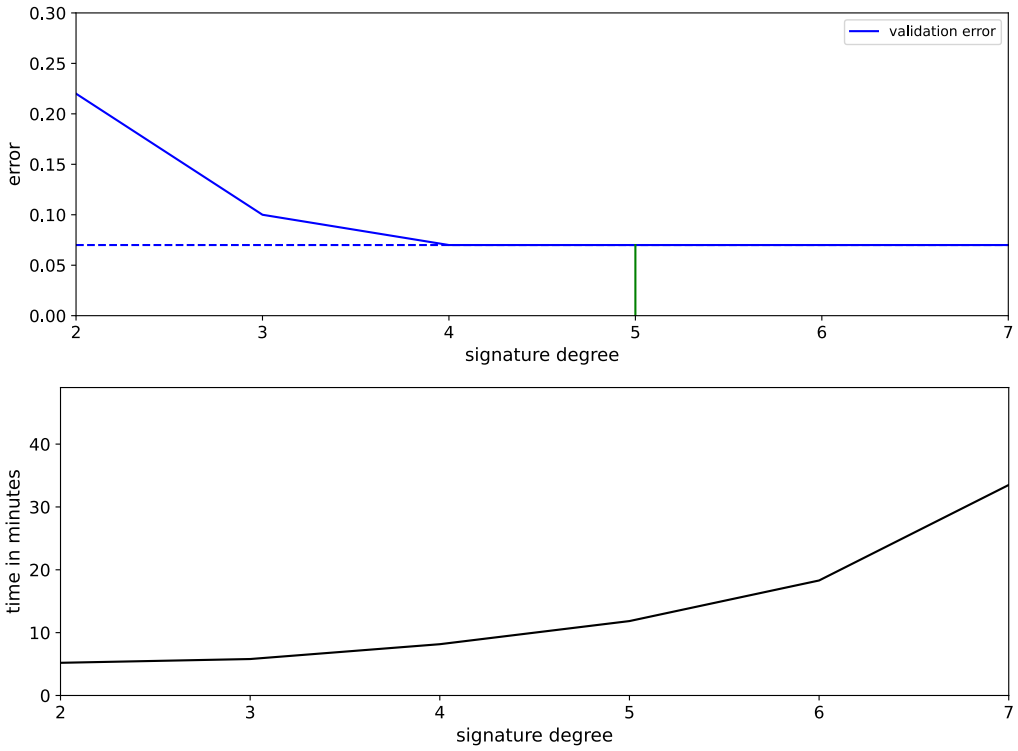

Figure 5: Validation error and training time against the signature degree. In the first plot of the figure, the signature degree 5 gives the smallest error on the validation data. The errors for the 6th and 7th orders of the signature are slightly higher that the one for the 5th order signature. The error is the proportion of the validation data predicted incorrectly.

Signature degree and feature selection

The first plot in figure 5 suggests that I should pick 5 as our signature level. I only select the top 26 features, as explained in the following. I rank the features in the models with 5 and 6 signature levels, respectively and identify in the 6 level model which feature is the first one that does not belong to level 5 of the signature. Feature 89 is ranked 27th in the 6 level signature model and so from now on I select the top 26 features, whose cumulative importance is 83.23%. These features will also be used in the CNN model, which emphasizes that it is useful to try multiple models on the same problem because one model may provide insight about the innate structure of the data that can help in subsequent models. The next plot is shown to see the rate at which the random forest learns. I keep the validation set the same and I only take fractions of the training set to see whether I use too much data or too little.

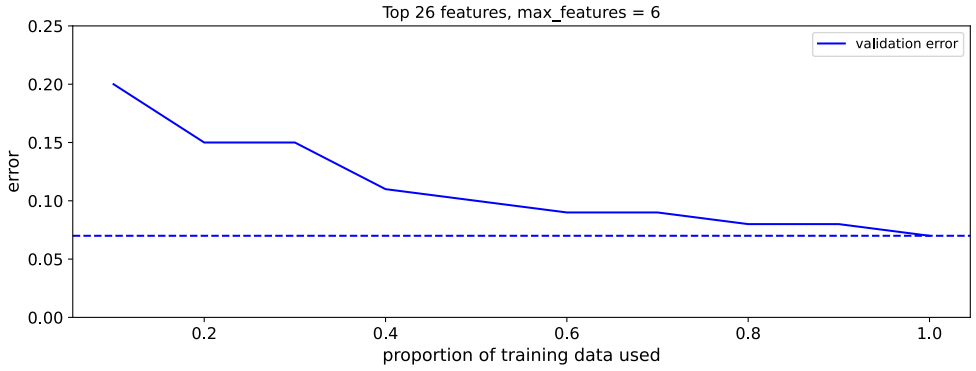

Figure 6: See how is the random forest model learning as I provide it with more training data. The error is the proportion of the validation data predicted incorrectly.

The plot in figure 6 suggests that the algorithm is able to learn and that it is likely that given more data it will do even better. From now on I always use the entire training set.

I have to see whether I are overfitting the data. Since it is a classification problem, at each split I set the maximum number of features to be considered to be 6 (the ceil of the square root of the total number of features). The other popular option is the ceil of the logarithm in base 2 of the number of features, which is 5.

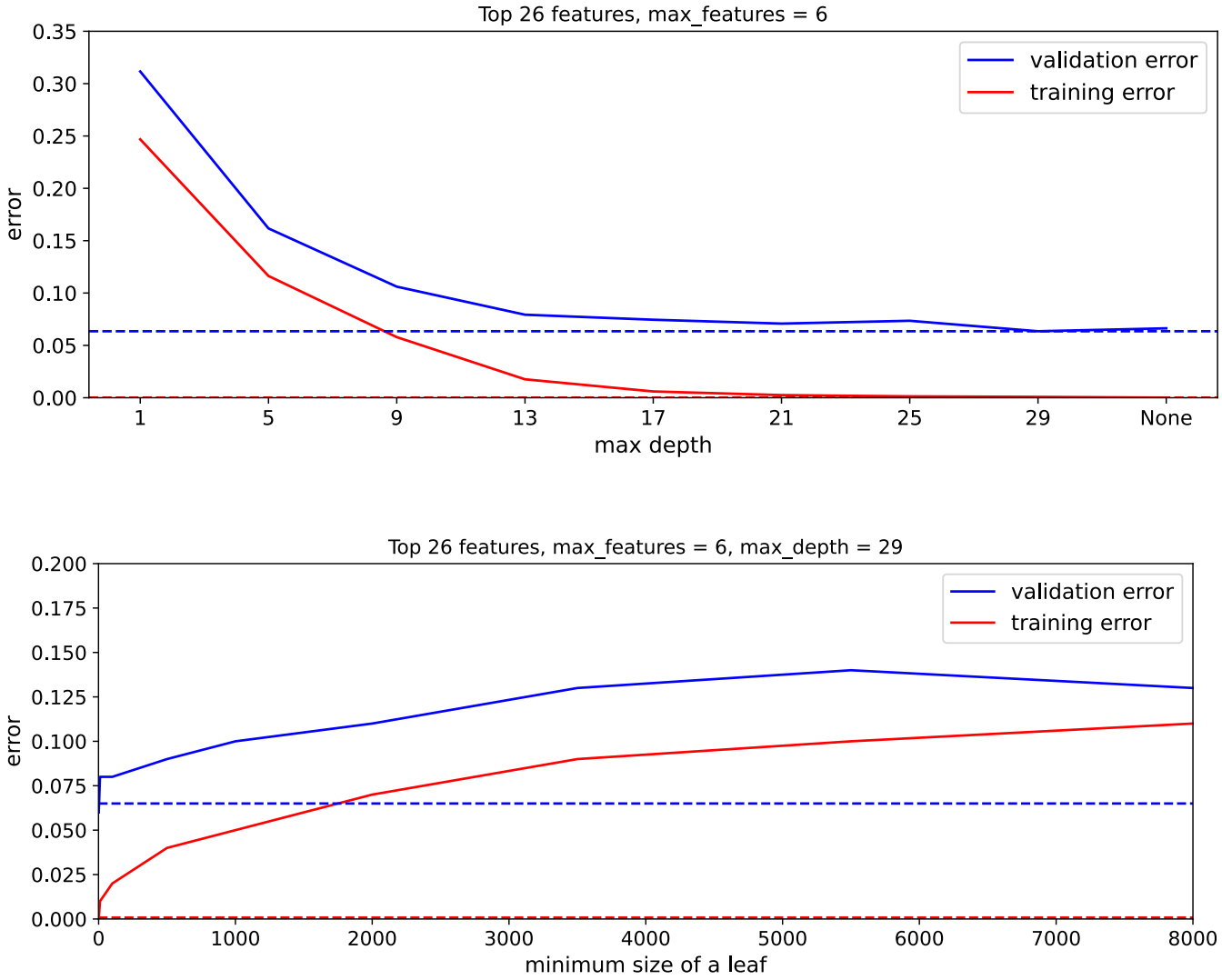

Figure 7: I see two plots that display what happens for various values for the maximum depth as Ill as the minimum size of a leaf in the random forest. The error in the above two plots is again the proportion of the data predicted incorrectly. They suggest that I do not overfit the training data and I can proceed to the prediction step, where I retrain a random forest model with 100 esimators on the training data - without including the validation data - keeping the top 26 important features of a log signature computed from a signature of degree 5, a maximum depth of each tree in the random forest of 29 and the minimum number of samples on each leaf of 1 and the number of features to be selected at each split of 6. The accuracy on the testing data measured as the number of individual cycles predicted correctly is 89.33%.

In using the segmentation algorithm, the identification of the proportions of the true positive rate and false positive rate is helpful, because the function inside the segmentation algorithm is sensitive to wrong predictions.

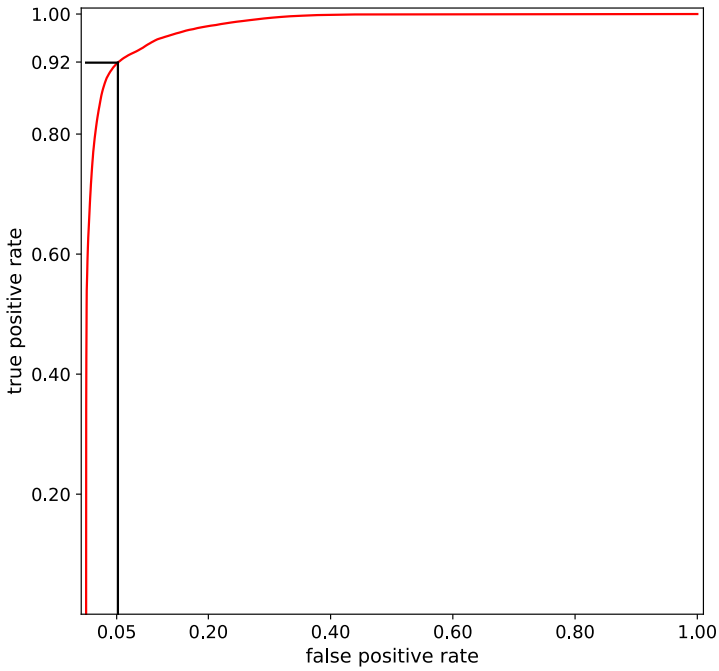

Figure 8: ROC curve for the random forest model done with the selected hyperparameters. The random forest model consists of 100 estimators. During the prediction whether a cycle is on or off, the model takes, by default, the majority vote of each of the 100 estimators, and the threshold is by default 0.5. I plot the ROC curve for this random forest model. To do this, I get the output of each individual tree and compute the proportion of the trees that decide that the cycle is during an 'on' period of the fridge. I turn the majority vote into a probability distribution, which I can compute the ROC curve for. In particular, for the 0.5 default threshold the true positive rate is 0.92 and the false positive rate is 0.052 for the validation set.

Training a model on individual cycles is prone to larger errors than training on chunks of a certain - larger - size. In the random forest model, the size is too small. If one cycle is predicted wrongly, since consecutive cycles have very similar signatures, they will most likely also be predicted wrongly and create false positive or false negative periods for the fridge. Each interval in the segmentation algorithm corresponds to a contiguous block of the testing dataset. This block is composed of consecutive rows, each of which being the signature of the 3-dimensional path during one voltage cycle. This is prone to errors because of the contiguous wrongly predicted sequences of cycles. This can be seen in the plots in figure 9, because sometimes there are some false positive intervals when the fridge is on whose size is comparable with the intervals when the fridge is actually on.

In the following page, I use the segmentation algorithm in order to generate the individual fridge usage plots on the testing data of 2.8 days. The function in the segmentation algorithm returns true for an interval if at least 98% of the cycles it consists of are predicted to be during an 'on' period. Creating the function so that it passes the interval if every cycle it consists of is predicted true produces bad results because the function will reject many large true 'on' intervals. I took a 2% margin to protect against false negatives. It takes 9 seconds to do the segmentation on individual cycles. The accuracy of the plots in figure 9 is 87.45%, so the true intervals and the predicted intervals overlap in that fraction of the base interval. More precisely, I predict 2.45 days of data correctly out of the 2.8 days that the testing set consists of. This is slightly less than the prediction accuracy on individual cycles, which is not unexpected. Recall that by true intervals I mean the intervals that appear in our dataset, taken with a 10 second margin on each side of the individual appliance dataset.

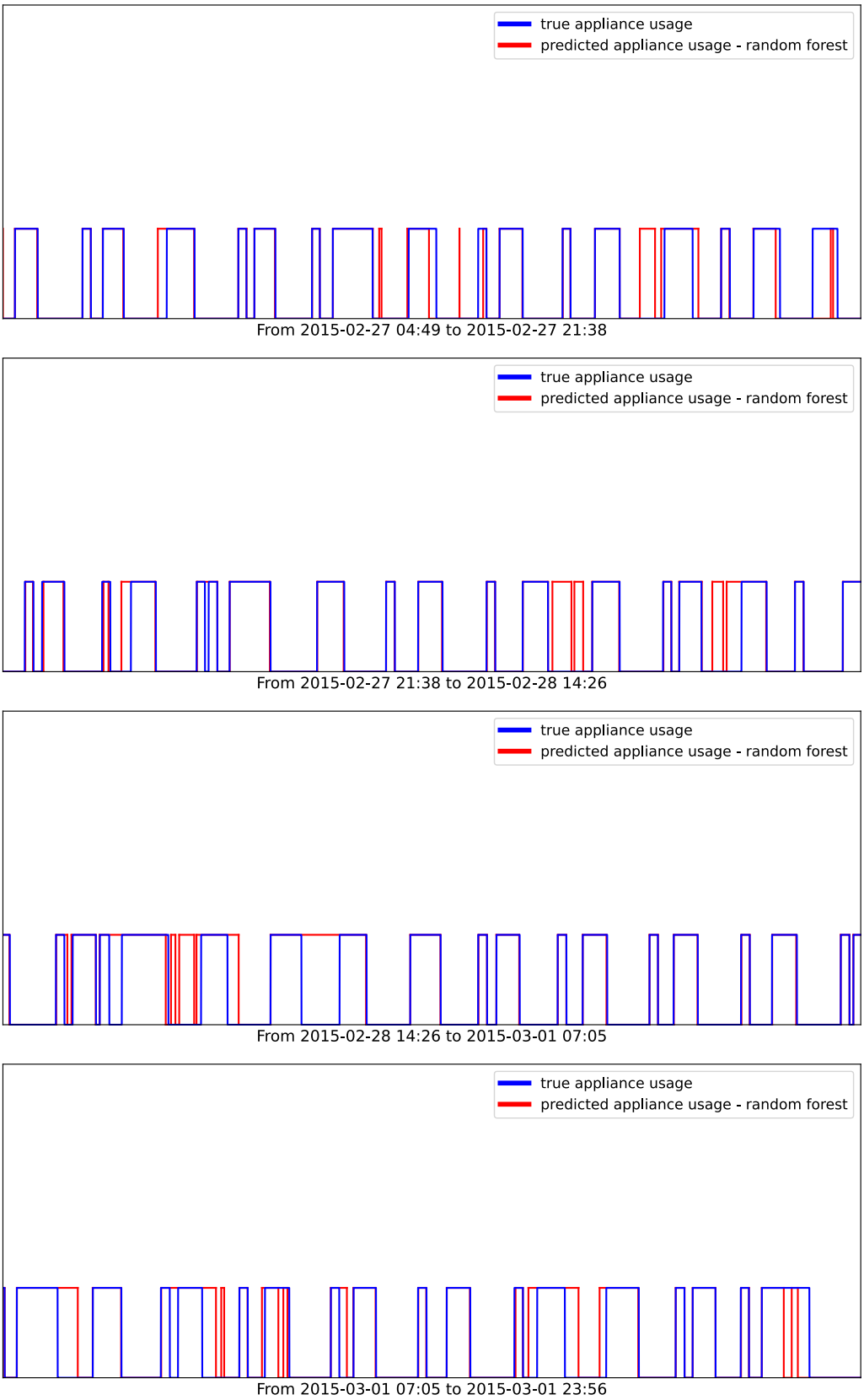

Figure 9: The generated fridge usage plots using the output given by the segmentation algorithm for the random forest model.

3.3 Convolutional neural network

Training on individual cycles makes it hard for the segmentation algorithm to generate the individual appliance plots. Even though I get a high accuracy in predicting the states of individual cycles, this is not good enough because the goal is to accurately predict the time intervals where the fridge is on or off, respectively. Hence, I need to cooperate with the way the segmentation algorithm works in order to get more accurate individual appliance plots.

The same split applies here as Ill: 60% training set, 20% validation set and 20% testing set, but the learning problem has changed. Before I had learned a function which mapped an object of shape $(1, 26)$ to the set $\{0, 1\}$. In the current problem I have to learn a function that maps an object of shape $(42, 26)$ to the set $\{0, 1\}$. Each $(42, 26)$ object consists of 42 contiguous $(1, 26)$ objects, each of which has a label 0 or 1. I assign 1 to a $(42, 26)$ object using the AND operation, namely if each of the $(1, 26)$ objects it consists of has a 1 label. If there is only one $(1, 26)$ object that has a 0 label, then the $(42, 26)$ object that contains it will be assigned a 0 label. For each one of the training set and the validation set, I move in steps of 21 along the rows of the set and select the block of 42 rows starting from the current row. In this way, the shape of the training set is a rank 3 tensor of shape (35309, 42, 26). Similarly the validation set has the shape (11769, 42, 26). I tune the CNN on the validation set. Then, I retrain on the training set only with the selected parameters. It is important that right now I do not do predictions on the testing set, because depending on which intervals the segmentation algorithm is choosing, they will be divided into subintervals of length 42 and the algorithm will use the live prediction from the CNN model in order to decide whether to include the larger interval in the output. That is, each time the segmentation algorithm chooses an interval, it needs to know whether all its non-overlapping subintervals of length 42 are 'on', and it does this in a live manner by calling the prediction method of the CNN model for each such subinterval. The 42 length intervals that have to be predicted are only known at the runtime of the segmentation algorithm. Because of this reason, the segmentation algorithm takes about half an hour to return the intervals when the fridge is on.

The size of the intervals I need to train our data on needs to be optimally chosen. If the size is too small I are prone to relatively large errors in the overlapping accuracy. If the size is too large, then a false positive or a large negative would add a much larger error to the overlapping accuracy in the end. I provide a way to find this optimal value of the size of the time frame I need to train our data on, by strongly collaborating with how the segmentation algorithm works. I need to justify the reason for choosing 42 second intervals in the CNN model. Converted to seconds, $\frac{1}{2048}$ of a day is 42.18 seconds, and the smallest length of any interval considered in the algorithm is 42.18 seconds. If it tests for a larger interval, the boolean function implemented by us divides the interval into non-overlapping contiguous subintervals each of which comprising of 42 seconds and it computes the AND operation, in the sense that the function treats the larger interval to be an 'on' period for the fridge if and only if each of the subintervals is predicted by the CNN model to be an 'on' period for the fridge. On the other hand, when the model given as input in the algorithm is the random forest classifier - trained on individual cycles - an interval considered by the segmentation algorithm will be treated as 'on' if 98% of the individual cycles from it are predicted by the model to be 'on'. The question arises: why train only one model for length 42 second intervals and not train multiple models for the different timeframes considered by the segmentation algorithm - this would remove the division into subintervals part and directly return true or false? The reason is that either a false positive or a false negative of a large interval will produce a large error; for example, if the segmentation algorithm decides that during a 10-minute interval the fridge is 'on' when it is actually off, this will produce a large error in the end. The 42 second precision is smaller than all the 'on' or 'off' periods of the fridge. I ultimately want to reduce the false positives and false negatives of large intervals, as these increase the error in the end. The fact that I divide the larger interval tested by the segmentation algorithm into subintervals of 42 seconds gives us protection against false positives, because it is very unlikely that when a large interval is tested, *all* its subintervals for which the fridge is 'off' are actually predicted to be 'on'. Indeed, the false positives in our plots only create small false

'on' periods of the fridge, as it can be seen in the very small sudden red spikes in the plots in figure 13. The reason why I get an improvement in the CNN model is that I combine it very Ill with the way the segmentation algorithm works.

Layers

- Conv1D layer with 64 filters and the default value 1 of the strides hyperparameter.
- Average pooling layer with pool size set to 2.
- A flattening layer.
- At the end, since I aim for binary classification, I include a dense sigmoid layer.

I compile the model using the binary cross entropy loss and the stochastic gradient gradient optimizer. Most of the log signature features are of the order 10^{-4} and leaving these as they are will lead to a poor performance of the model, most probably because of the numerical instability of the algorithm. I tried to multiply each feature by the same constant: $1, 10, 100, 1000$; 100 achieved by far the best accuracy. In what follows, the results are using the dataset where the entry of every cell is multiplied by 100. Because of the randomness of the optimizer, the following results are reported as the average of 5 models trained for each hyperparameter separately. The standard deviation is always at most 0.02. After having selected the hyperparameters, I only retrain one more time on the training dataset of shape (35309, 42, 26) and go further with that model in the segmentation algorithm.

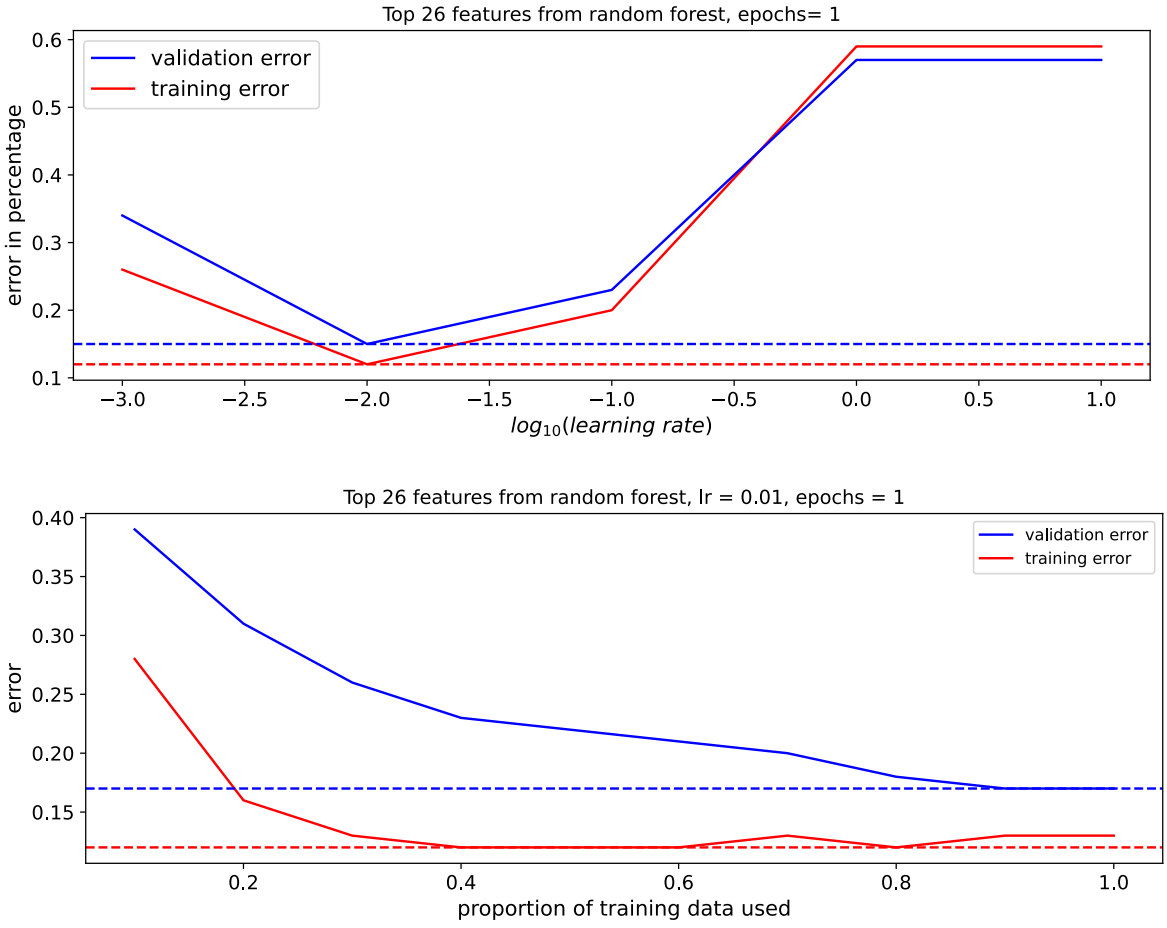

Figure 10: I tune the learning rate "lr" hyperparameter and then I further see how is our model learning the problem using the learning rate that gives the smallest errors. I take fractions of the training data without changing the validation data. The error is the proportion of the data predicted incorrectly.

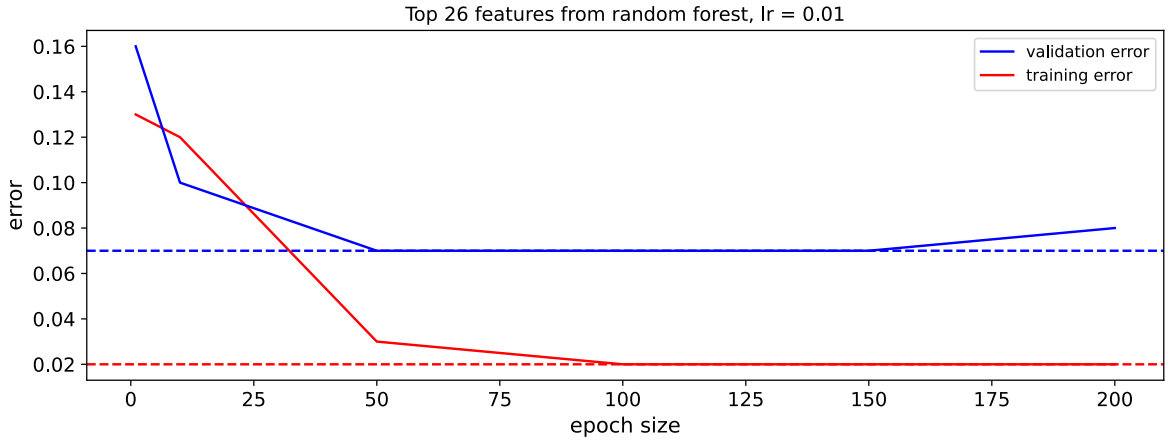

Figure 11: Using the "lr" value that minimizes the error, I further tune the "epochs" hyperparameter. The value 150 is at the bottom of the "U" shape of the validation curve is the value I use from now on. The error is the proportion of the data predicted incorrectly.

Choosing the threshold in the sigmoid layer

It is important to choose a good threshold for the model. Because of the randomness of the stochastic gradient descent optimizer, I will get different ROC curves for different runs of the same algorithm. Choosing a threshold is a subjective task.

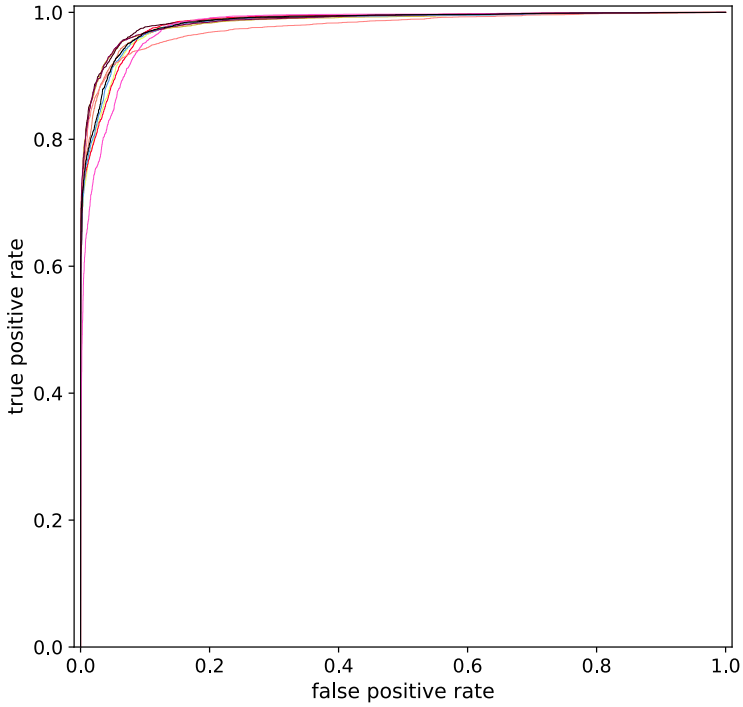

Figure 12: Ten ROC curves corresponding to ten different runs of the CNN model with the selected hyperparameters for the predictions on the validation set. For these ten models, the 0.5 threshold in the sigmoid layer results in an average true positive rate of 0.897 and an average false positive rate of 0.046.

With 10 different runs of the same model, the 0.5 threshold produced true positive rates betIen 0.86 and 0.92 and false negative rates betIen 0.3 and 0.7 on the validation set. In some cases, trying a threshold corresponding to a higher true positive rate than the one given by the 0.5 threshold produced a loIr overlapping accuracy. These numbers show that choosing the natural threshold of 0.5 is a good idea. Retraining a model with these hyperparameters and a threshold of 0.5 gives an accuracy of 93.8% on the validation set. The accuracy in the aligned plots obtained by using the segmentation algorithm with this CNN model is 94.69%.

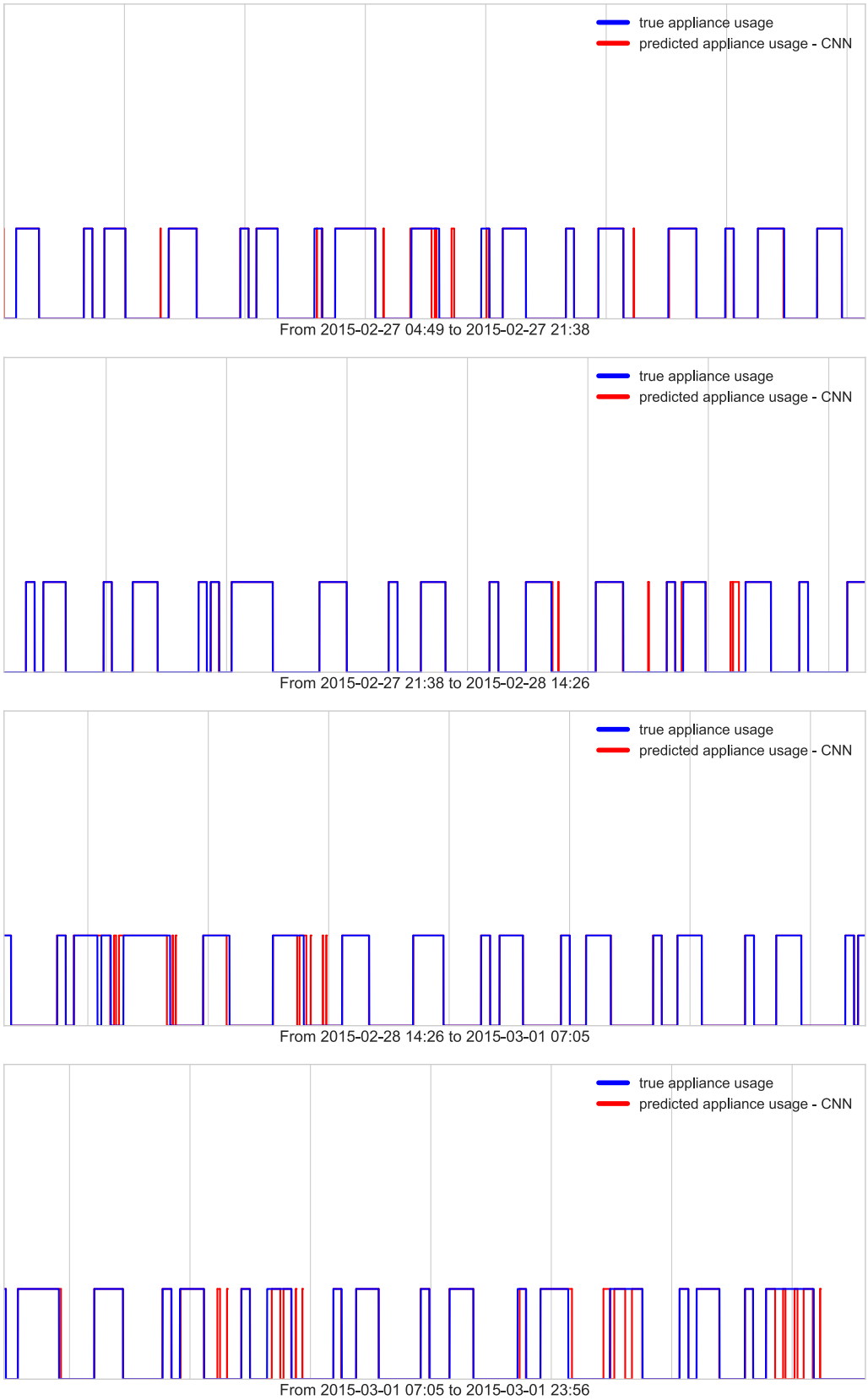

Figure 13: The generated fridge usage plots using the output given by the segmentation algorithm for the CNN model.

Summary for the machine learning approaches

I showed that the path signature can be very useful in producing a thorough summary of a multivariate path with a relatively small number of features, irrespective of the frequency with which the data was sampled. Recall that a cycle has 320 datapoints that characterizes it. I took one voltage cycle as the timeframe reference, added a time path corresponding to the timeframe and then added the current path which encodes the information from the transition periods of the fridge. Recall that a discretely sampled path can be made continuous by linearly interpolating between consecutive datapoints. This resulted in a three dimensional path and I only characterised it with 26 features. Despite having a lot of data to work on, I only chose a contiguous 14-day period to be able to try many approaches, because otherwise it would have taken much more time to try these models, due to the computational time difficulties. Also recall that a second consists of 50 cycles, but I downsampled our data to one cycle per second because this reduces the computational time of a model and another reason is that the other 49 cycles in each second have almost identical signatures. The segmentation algorithm is key to produce the generated individual usage of the fridge plots. The two approaches Ire: train on individual cycles or train groups formed by taking blocks of 42 consecutive cycles. Training on individual cycles is prone to errors because of the frequent false positive and false negative sequences in the predictions. Doing a segmentation on these creates false positive and false negative predictions. Training on chunks of 42 rows of data is better for the segmentation part because I are much less prone to getting false positives on large intervals, due to the fact that I divide the larger interval into contiguous non-overlapping subintervals of 42 seconds and test each one individually. Training on even larger intervals than 42 seconds is prone to even larger errors than in the individual cycle model because a false positive or a false negative of a large interval will add a large error in the end. The 42 is in the same time not too big and small enough to be smaller than every period when the fridge is entirely ‘on‘ or entirely ‘off‘. I used the feature importances given by the random forest to reduce the number of features that I considered in our models. The top 26 features from the 5-order signature Ire used in the primary random forest model as Ill as in the primary CNN model.

	Training time	Accuracy on validation data	Overlapping accuracy of generated plots
Random forest	14 minutes	92.5%	87.45%
CNN	8 minutes	93.8%	94.69%

Table 3: Comparison of the two approaches: random forest on individual cycles and convolutional neural network on blocks of 42 seconds. The CNN model is doing better in generating the individual fridge usage plots because it combines better than the random forest model with the segmentation algorithm.

4 Discussion

I Ire able to generate the fridge usage plots with a high accuracy for a period of 2.8 days by only using the 16kHz data, because the predicted ‘on‘ time intervals of the fridge during that period Ire very close to the truth. Recall that I only selected 14 days of data from February 2015 for the entire model - 60% for training, 20% for validation and the last 20% for generating the fridge usage plots from the 16kHz data. Given that I only selected one such subset, the ability to produce a statistical summary of the results is reduced, because slightly different accuracies will occur for a different 14-day subset of data. Additionally, as I have seen in the second plot of figure 10, it looks like the ability of the CNN model to learn has not plateaued yet, so it seems that adding more training data to the model will make the model stronger - one approach is using a split of 80-10-10 on a 28-day sequential subset of the data. I thus emphasize that the focus of this study is on the method used and not on giving a statistical summary of the results achievable by using the entire UK-DALE dataset. I believe I achieved our goal. Therefore, as a means of feature extraction, the path signature can be used as a tool to provide features of multivariate paths that can be fed into machine learning models for classification and prediction problems.

5 References

1 Xie Z, Sun Z, Jin L, Ni H, Lyons T. Learning spatial-semantic context with fully convolutional recurrent network for online handwritten chinese text recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence. 2017;.

2 Chen KT. Integration of paths–A faithful representation of paths by noncommutative formal poIr series. Transactions of the American Mathematical Society. 1958; 89(2):395–407. https://doi.org/10.2307/1993193

3 Johnsonbaugh, Richard F.; Pfaffenberger, William Elmer (2010). Foundations of mathematical analysis. Mineola, NY: Dover Publications. ISBN 978-0-486-47766-4.

4 B. Hambly and T. Lyons, "Uniqueness for the signature of a path of bounded variation and the reduced path group," Annals of Mathematics, pp. 109-167, 2010.

5 T. Lyons, "Rough paths, Signatures and the modelling of functions on streams," In Proceedings of the International Congress of Mathematicians: Seoul, pp. 163-184, 2014.

6 Chevyrev I, Kormilitzin A. A primer on the signature method in machine learning. arXiv preprint arXiv:160303788. 2016;.

7 Jack Kelly, William Knottenbelt, The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes

8 https://en.wikipedia.org/wiki/AC_poIr

9 https://en.wikipedia.org/wiki/Utility_frequency

10 Xiph.Org Foundation. FLAC: The Free Lossless Audio Codec http://xiph.org/flac (2012).

11 Iixin Yang, Terry Lyons, Hao Ni, Cordelia Schmid, Fellow, IEEE, LianIn Jin, Member, IEEE: Developing the Path Signature Methodology and its Application to Landmark-based Human Action Recognition

12 https://github.com/datasig-ac-uk/pysegments

13 Chapter 2 of: https://arxiv.org/pdf/math/0108030.pdf