Energy Disaggregation of Household Appliances

|  |  |
| --- | --- |
| Disaggregation Team  DAEN 690 Summer 2019 | |
| Role | Member |
| Product Owner | Rosa Donat |
| Scrum Master | David Croghan |
| Developer | Nima Latifi |
| Developer | David Jones |
| Developer | Mark Richard |

Abstract

Household energy disaggregation is the process of inferring the electrical power consumption of individual household appliances from the total power measurement of a residential smart meter. Disaggregation algorithms can be used to break down household energy consumption data into appliance level measurements and predict the appliance state and energy consumption. Widespread deployments of residential smart meters have caused energy disaggregation to be a popular research topic and a potential tool for electric utilities to develop demand-side incentives to modify customer energy consumption behavior. This project uses cutting edge deep learning techniques and probabilistic graphical models to develop NILM algorithms that predict individual appliance energy usage using only aggregate smart meter interval load data. The models were trained using the REFIT dataset, which includes approximately two years of aggregate and sub-metered load from 20 households in the UK. In addition to developing an energy disaggregation model, the team implemented an unsupervised anomaly detection algorithm to identify deviations from typical appliance consumption patterns. The combined disaggregation and anomaly detection model could be used by electric utilities to make residential customers more aware of their energy usage and provide notification of potential appliance problems. The models were trained using open source Python libraries such as hmmlearn, Tensorflow, and Keras and they were deployed in the George Mason ARGO Cluster and AWS cloud environment.

Table of Contents

Abstract 2

Introduction 5

Primary Objective 5

Modeling Approach 6

Dataset 6

Field Descriptions 7

Data Quality Assessment 8

Data Conditioning 8

Algorithms 9

Hidden Markov Models 10

Neural Network Models 19

Modeling Results 30

Regression Model Performance Metrics 30

Regression Model Training 31

Regression Model Results 33

Classification Model Performance Metrics 33

Probabilistic Classification Model Training 34

Probabilistic Classification Model Results 36

Analysis of Results 41

Anomaly Detection 42

Proposed Applications 44

Interactive Energy Bill Dashboard 45

Data Collection 46

Model Training 47

Create Data Pipeline 47

Create Visualizations 47

Model Retraining 48

References 49

Taspinar, Anhet, A guide for using the Wavelet Transform in Machine Learning, 2018, 49

Appendix 50

Example: Metadata for one house 50

Example: Statistics for House 18 51

# Introduction

Climate change has inspired politicians, researchers, utility companies, and technology vendors to innovate new solutions deliver clean, reliable, and resilient energy. One proposed tool to encourage residential customers to use energy more efficiently is known as household energy disaggregation. This is the process of using pattern recognition algorithms to infer the energy consumption of individual appliances using only the net household energy consumption read by a smart meter. Breaking down energy usage into appliance-level measurements will allow customers to have a more detailed understanding of their energy usage and discover ways they can alter their behavior to reduce their energy bill.

Utilities have been collecting residential smart meter data for more than a decade, but have so far only used the data to create electric bills and detect outages. This disaggregated appliance load can be used for several other use cases such as creating an itemized energy bill broken down by each appliance, or detecting appliance anomalies. By showing disaggregated energy usage for each appliance, a homeowner can understand specific actions they can take to improve their energy usage, such as using an energy intensive appliance at night, when prices are cheaper, instead of in the late afternoon.

## Primary Objective

The project team proposes to develop a model that will infer the energy consumption of individual appliances based upon the household aggregate measurements. An anomaly detection model will also be developed to recognize abnormal appliance operation, with the goal of reducing energy wastage and minimizing appliance downtime (Reference 3).

The combined disaggregation and anomaly detection model results can be presented using a dashboard that will allow a user to interactively explore their energy usage on a more granular level. A utility company can use this dashboard to send notifications to homeowners to check their appliances in order to avoid excessive charges. Alternatively, the dashboard could be used to provide an energy bill with usage broken down by appliance.

## Modeling Approach

The team will attempt to develop two types of energy disaggregation models: one based upon the probabilistic graphical model framework and one using neural network techniques.

Probabilistic based methods, such as Factorial Hidden Markov Models have proven effective in performing energy disaggregation because of their ability to exploit context-based features such as usage patterns and external data like temperature and climate information.

Some researchers have used various neural network techniques, which have shown very good performance on several benchmark datasets (Reference 5)[[1]](#footnote-2). Since a number of neural network techniques can be used, the team will explore two general methodologies:

1. Using a denoising autoencoder neural network model, and

2. Using a convolutional neural network model.

# Dataset

The REFIT Electrical Load Measurements dataset includes cleaned electrical consumption data in Watts for 20 households at aggregate and appliance level, timestamped and sampled at 8 second intervals, for 2013 through 2015. This work was created as part of the REFIT project: a consortium of three universities - Loughborough, Strathclyde and East Anglia - and ten industry stakeholders funded by the Engineering and Physical Sciences Research Council (EPSRC) under the Transforming Energy Demand in Buildings through Digital Innovation (BuildTEDDI) funding program.

## Field Descriptions

Table 1: REFIT Dataset Field Descriptions

| **Field Name** | **Description** |
| --- | --- |
| Time | Timestamp in GMT |
| Aggregate | Smart meter aggregate electricity usage interval data (6-8 sec intervals) |
| Appliance 1 | Interval electricity consumption data for an appliance (6-8 sec intervals) |
| Appliance 2 | Interval electricity consumption data for an appliance (6-8 sec intervals) |
| Appliance 3 | Interval electricity consumption data for an appliance (6-8 sec intervals) |
| Appliance 4 | Interval electricity consumption data for an appliance (6-8 sec intervals) |
| Appliance 5 | Interval electricity consumption data for an appliance (6-8 sec intervals) |
| Appliance 6 | Interval electricity consumption data for an appliance (6-8 sec intervals) |
| Appliance 7 | Interval electricity consumption data for an appliance (6-8 sec intervals) |
| Appliance 8 | Interval electricity consumption data for an appliance (6-8 sec intervals) |
| Appliance 9 | Interval electricity consumption data for an appliance (6-8 sec intervals) |

## Data Quality Assessment

The team used the Data Management Association (DAMA) data quality dimensions to provide an assessment of the REFIT data set quality.

The dataset, as provided, is fairly clean. The authors of the dataset listed some actions that were taken to prepare the data such as:

* During the study there were a few periods of missing data (notably February 2014). The outages were due to several factors, including household internet failure, hardware failures, network routing issues.
* The authors forward filled NaN values and removed values which exceed 4000 Watts on the IAM data streams since that would be above the maximum possible power draw
* Any timestamp duplicates in the dataset were merged
* NaN values have been forward filled (< 2 minute gaps) or zeroed (> 2 minute gaps)

## Data Conditioning

The raw time-series measurement data in the REFIT dataset contains some spikes, gaps, and anomalies which were smoothed during preprocessing. The team explored using smoothing and filtering to remove unwanted outliers. However, once the time-series data was resampled to one-minute intervals, most of the high frequency noise and spikes were smoothed out.

An additional benefit of downsampling was that it significantly reduced the preprocessing and model training time, which allowed for more house data to be used during model development. The downsampling was performed by taking the mean of all data points in each one-minute interval. Although median resampling was also explored to remove distortion from spikes, a comparison between methods showed no significant difference in the resampled output.

The Hidden Markov and Factorial Hidden Markov Models required labels of appliance ON/OFF states during training. To prepare the data labels for our analysis we identified and labeled the ON and OFF states of each appliance in the data set.

# Algorithms

As mentioned, the energy disaggregation approach includes probabilistic graphical models and neural network models to identify appliance load shapes in time-series smart meter data, along with unsupervised learning techniques to identify anomalies in appliance usage patterns. The neural network models were initially developed and trained using the Google Colab platform, which includes free GPU resources. The George Mason ARGO cluster was used to train and test the neural network models on the full REFIT dataset. The probabilistic graphical models were developed on an Amazon Web Services cloud environment.

The proposed analytics processes include:

1. Supervised classification of appliance states using Hidden Markov Models and Factorial Hidden Markov Model (FHMM).
2. Supervised regression of the appliance load using Convolutional Neural Networks
3. Supervised regression of the appliance load using an Autoencoder Neural Network
4. Anomaly detection of the disaggregated appliance load using Convolutional Neural Networks
5. Develop conceptual design for a scalable energy disaggregation and anomaly detection system.

## Hidden Markov Models

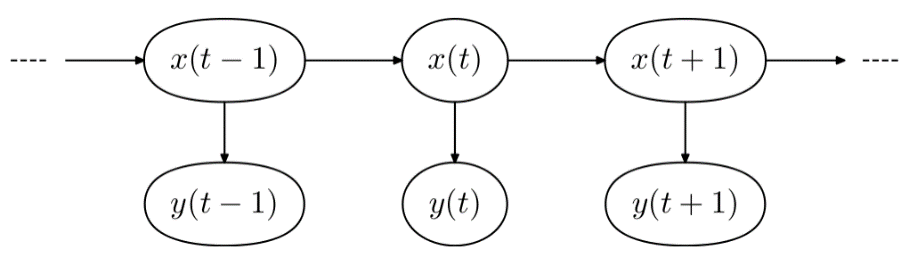


Figure 1: Summary of Probabilistic Model Training

Hidden Markov Models (HMMs) are probabilistic graphical models, useful for modeling discrete time series applications that require estimating the state of a variable that you don’t directly observe[[2]](#footnote-3). The unobserved variable as referred to as being ‘hidden’ or ‘latent’ in our model and we estimate the state of it from observed values. These observations, which can be discrete or continuous, are used to reason the probability of that hidden variable being in a particular state at a particular time. This assumes within the HMM structure that “the observed data is dependent on the hidden state at that particular time.”

To construct a Hidden Markov Model, you must have prior information about the possible states of your hidden variable. For our disaggregation problem the structure of our HMM model is as follows:

* The ‘state of our appliance’ is the hidden variable with possible values ‘On’ or ‘Off’.
* The aggregated power consumption of the house, captured over consistent time intervals, are our observations.

The following sections describe how HMM’s and Factorial HMMs, were used to estimate the state of an appliance for our disaggregation of energy problem.

### *Graphical Notation*

Fig. 2 is a graphical notation of a Hidden Markov Model. The graph displays two random variables and , a representation for time and the relationships between them.

Below is a description of the main variables in an HMM as depicted in Fig. 2:

* ~ Represents a hidden random variable whose state we would like to estimate.
* ~ Represents a random variable for which we have observed data.
* ~ The discrete time step.
  + For the disaggregation problem (t) was sampled during various minute intervals.

As mentioned previously, for our disaggregation of energy problem, we treated the hidden stateof the appliance as either “On” or “Off” at each time stamp . The graphical structure of the model describes the relationship as “the future state of the variable is conditionally independent of its history given the current state.”

where:

= the future state of the hidden variable

= the current state of the hidden variable

The aggregate power consumption of the house are the observations of our model where each output of the home is conditionally independent of the other given the current state of the appliance .

Formally, the relationship between the observations and the hidden states are represented as: .

Finding the joint distribution of these random variables is possible due to the conditional independence assumptions that are present within the structure of the Hidden Markov Model. The joint distribution breaks down into a simplified factorization of local distributions. Using the example depicted in Fig. 1, the factored representation is below:

### *Parameters of a HMM*

A Hidden Markov Model consists of three parameters :

* = The initial state distribution at time
  + The initial probability of each state.
* = The state transition matrix:
  + The probability of transitioning from current state to future state of the hidden variable.
* = The emission matrix:
  + The probability of observation given the hidden state at time .

### *Inference in a Hidden Markov Model*

Inference in an HMM is obtained by determining:

1. What is the probability that the given observations were generated by the model parameters ? This is the Evaluation problem for which we use the “Forward pass” of the ‘Forward-Backwards’ algorithm.
2. What is the most likely state sequence that produced the observations given the model parameters ? The Viterbi algorithm is used for decoding the most probable sequence.
3. The ‘Baum-Welch’ algorithm is used for estimating the parameters that best maximizes the probability of the observations being generated by the model?

### *The Evaluation Problem*

Determining the likelihood of the model having generated the observations is the evaluation problem, . Answering this question effectively requires determining the probability of every possible combination of hidden states that could have been generated by your Hidden Markov Model and summing them up. To arrive at this inference in an efficient and tractable manner we use the ‘Forward pass’ of the ‘Forward-Backward’\*[[3]](#footnote-4)[[4]](#footnote-5) algorithm which simplifies the complexity of the computation to .

* = the length of your observations
* = the number of states in your model

The ‘Forward’ pass, also called alpha , provides the probability of being in state at time after the first observations in a recursive manner.

Breakdown:

* = the probability of being in the initial state .
  + For our disaggregation problem this is the initial probability of the “On” or “Off” state.
* = the state transition matrix
  + - The probability of transitions from state to state
* = the total number of hidden states
* the emission matrix
  + The probability of an observation (home power output at time ) given state
* = the alpha calculated for state at the previous timestep

Finding the probability of the model having generated the observations is found by taking the log-likelihood of the sum of alpha’s from the ‘Forward’ pass of each state at time

### *The Decoding Problem*

After determining how likely the model is to have generated the observations we then answer the question of, “what’s the most likely sequence to have generated the observed sequence?” For our disaggregation problem we are looking to estimate the sequence of appliance states that most likely produced the aggregated home energy usage. This question is answered by using a dynamic programming algorithm as well, the Viterbi algorithm.

Instead of summing up the probabilities of each possible sequence at time step as the forward algorithm, the Viterbi algorithm uses the most probable state at each time step (t) to determine the most probable sequence of hidden states to have generated the observations. The algorithm is listed below:

Breakdown:

* = the Viterbi path probability for the current state at timestamp
* the emission matrix
  + - The probability of seeing your observations given state
* = the max of the product for each state:
  + - the Viterbi path probability for the previous state at timestamp
    - = The probability of transitioning from state to state
* = the total number of hidden states

### *The Baum-Welch Algorithm*

The first two problems assumed that the “parameters of the model were known” however the team that developed hidden markov models, led by L.E. Baum, designed the Baum-Welch algorithm to deal with the most likely use case; where the parameters of the model are unknown. The algorithm works by estimating the parameters of the model that maximizes the likelihood of your model generating the sequence of observed values. This is very useful for the energy disaggregation problem where the total house power consumption at time is the only information available to infer the state of various appliances in the home at time .

Before diving into the Baum-Welch algorithm we introduce the ‘Backward’ Pass of the ‘Forward-Backward’ algorithm. The Backward pass calculates the probability of the remaining observations given the current state: .

Breakdown:

* the emission matrix;
  + - The probability of seeing the observation at (t+1) given the state at .
* = the state transition matrix;
  + - The probability of transitions from state to state
* = the total number of hidden states
* = the beta calculated for state at timestamp

Having introduced each pass of the ‘Forward-Backward’ algorithm, we now discuss how the parameters of the model are estimated using the Baum-Welch algorithm.

The Baum-Welch algorithm is composed of two steps. The ‘E’-step which calculates the probability of the observations given the model and the ‘M’-step which updates the parameters of the model for the next iteration based on the results from the ‘E’-step. This cycle repeats itself until the probability of the model converges or a limit is set on the number of iterations.

The E-step:

Breakdown:

* = The probability of being in state at time
* = from the ‘Forward’ algorithm for state at time step
* =Beta from the ‘Backward’ pass or state at timestamp

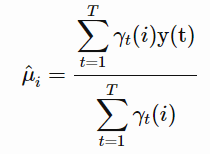
Breakdown:

* = The probability of being in state at time and state at time
* = from the ‘Forward’ algorithm for state at time step
* = from the ‘Forward’ algorithm for state at time step
* =Beta from the ‘Backward’ pass or state at time step

The M-step:

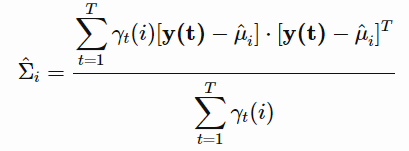
Breakdown:

* = The probability of transitioning from state to state .
* = The total probability of transitioning from state at time to state at time
* = The total probability of transitioning from state at time to any possible at time



Breakdown:

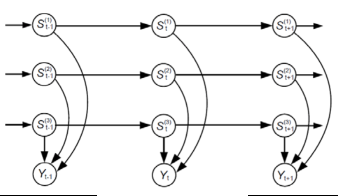
* = The probability of being in state at time
* = the observation at time



Breakdown:

* = The probability of being in state at time
* = the observation at time
* = the predicted mean of state

### *Factorial Hidden Markov Models*



Factorial Hidden Markov Models were introduced in 1997 by Ghahramani and Jordan as a “generalization of HMM’s in which the [hidden] state is factored into multiple state variables and is represented in a distributed manner”[[5]](#footnote-7). For our disaggregation problem, this allows us to account for all appliances in a single model versus an HMM where we would have to have a model per appliance.

Inference in FHMM’s are obtained using the same algorithms mentioned above although approximation is typically required for the Baum-Welch algorithm. The FHMM library that we utilized, from the ‘Non-Intrusive Load Monitoring Tool Kit’ (NILMTK)[[6]](#footnote-8), “stitches” together individual appliance HMM’s to form an “Additive” FHMM. This allows for exact inference.[[7]](#footnote-9)

## Neural Network Models

Deep Neural Networks (DNN) use a graph of neurons to create a non-linear model capturing unique patterns in the training data. When trained, a deep learning model uses gradient descent to update the parameters of the graph in the direction which will minimize the difference between our expected outcome and the true outcome. With the recent rise of distributed and cloud computing resources along with user-friendly APIs, deep learning has become a mainstream technique for modeling complex problems in engineering, finance, cybersecurity, and other domains.

An advantage of deep neural networks over traditional machine learning model is that manual feature identification is not necessary. The neural network will automatically identify underlying patterns in the data during model training by adjusting the weights and biases of the network graph.

Some neural network methods that have been explored for energy disaggregation are:

* Autoencoders, and
* Convolutional neural networks

### *Neural Network Architecture*

A neural network consists of a graph of neurons, each with associated weights and biases, which are activated when features are detected in the model input data. During model training, the weights and biases of the neural network are updated during a process called backpropagation, where the model output is compared to the target variables. Using the principles of the chain rule, an optimization algorithm performs gradient descent to minimize a loss function, which is some measure of distance between the model output and the target variable. After each epoch, or processing the entire training dataset, the model’s weights and biases are updated, to minimize the loss function. Over the course of multiple epochs of training, the neural network begins to identify nonlinear patterns in the training data to predict the output.

By training a neural network on aggregate and appliance load data from a given household, the model will “learn” patterns in the aggregate that are associated with each appliance. Moreover, by training the model on several houses, the neural network will learn to recognize several different patterns associated with an appliance, such as different washer dryer cycles or manufacturers. Convolutional neural networks use sliding filters trained to identify specific features and are particularly effective at recognizing unique patterns in time-series data.

Initially, a separate neural network was created for each appliance. The networks were trained on overlapping and non-overlapping windows of time-series data from the aggregate smart meter. This window should be long enough to capture a full duty cycle of the appliance, but not too long where the DNN captures noise from other appliances. Additionally, the window length must be long enough to be able to use convolutional layers that capture identifying features of the appliance.

### *Convolutional Neural Networks*

A convolutional neural network is any neural network that employs a mathematical operation known as convolution: a specialized kind of linear transformation that is different from the typical matrix multiplication performed in sequential dense layers.

Convolutional neural networks (CNN) have shown exceptional performance performing image classification due to their ability to recognize features such as lines and shapes. For two dimensional convolutional layers, the dot product between of an image pixel matrix and a filter matrix is calculated for each convolutional layer. An output matrix of smaller dimensions is created by “sliding” the filter across the image one pixel at a time. For time-series convolutional networks, the same principle is used, except in one dimension. The sliding filters extract segments of the time series data and identify waveform patterns that uniquely identify an appliance type.

In a convolutional neural network layer, a kernel “slides” across the input sequence for the boundary, which is padded with zeros so that the input length equals the output. Parameter sharing used by the convolution operation of stacked convolutional layers means that rather than learning a separate set of parameters for every layer, each layer learns how specific features relate to each other. Thus, stacking multiple convolutional layers to create a deep network will allow the model to learn several different patterns that are associated with an appliance. This capability is useful for appliances that may have different cycles such as the dishwasher and washing machine.

A convolutional layer’s output shape is affected by the shape of its input as well as the choice of kernel shape, zero padding and strides, and the relationship between these parameters requires tuning by trial and error. Prior to training the model, some preprocessing steps are required to reshape the data in the correct format for a CNN to consume.

A neural network architecture using successive convolutional layers followed by fully connected dense layers produced superior results compared to previous energy disaggregation models, including those using LSTM networks. A distinct advantage of CNN over LSTM models is that they can take advantage of graphical processing units (GPUs) to significantly reduce the required training time. Based on the superior results of previous research, the group selected to use a CNN-based architecture to perform load disaggregation.

### *Autoencoder Neural Networks*

Autoencoders are networks that aim to reproduce the output from the input. They do this by a specific type of architecture made up of an Encoder and a Decoder. The Encoder layers of the network compress the input into a latent space representation, while the Decoder reconstructs the input from the latent space representation. This is done through progressively smaller layers in the Encoder, and progressively larger layers in the Decoder. The Encoder constrains the network and creates a bottleneck which forces a compressed representation of the original input.

Autoencoder networks are useful for dimensionality reduction and data denoising. The process of learning a compressed version of the input data essentially produces a noise-free reproduction at the output. In the case of load disaggregation, the house-level aggregate signal is the “noisy” input and the individual appliance load is the target to be reproduced. The autoencoder network reproduces the appliance load by removing the parts of the aggregate signal that don’t correspond to it.

### *Hyperparameter Tuning*

The team took a structured approach to performing hyperparameter tuning of the neural network architectures to provide optimal performance. Initially the convolutional and autoencoder models were tuned by training and testing on individual houses. Next the models were tuned by training on multiple houses and testing on unseen houses. This approach was chosen to ensure that the models could be adequately trained on a variety of appliance types and usage patterns and be able to perform disaggregation for unseen houses.

There were a number of tunable parameters that were analyzed to select the best performing combination:

* Loss function
* Number of Conv layers
* Number of Max pooling layers
* Kernel size
* Number of filters
* Learning rate
* Batch size
* Window size
* Number and size of dense layers

Combinations of hyperparameters that did not require graph architecture changes, such as learning rate and batch size, were explored using a random search of the variable space to identify near optimal parameters.

### *Sequence-to-point Convolutional Model Architecture*

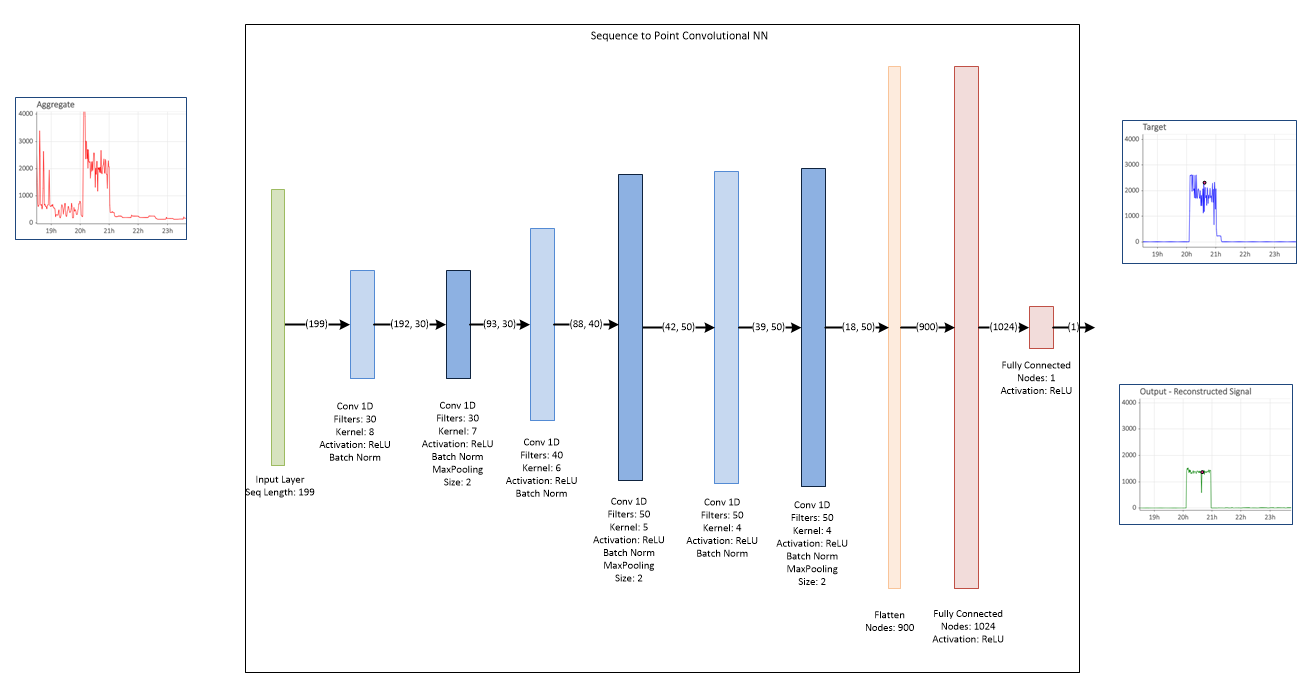


Figure 2: Sequence-to-point Model Architecture

For the sequence-to-point model architecture, the aggregate meter data is placed into an array of sequences of a specified length, where each sequence moves forward one timestep. As each sequence is passed to the model, the corresponding appliance load is selected at the midpoint of each sequence and is designated as the target variable. As the model trains, it compares the sequence to the single output value and adjusts the network weights and biases to minimize the loss between the predicted and actual value. For this model the selected loss was the mean squared error between the predicted and actual appliance load.[[8]](#footnote-10)

### *Convolutional Layers*

One-dimensional convolutional layers identify features in the aggregate time series data that uniquely represent the target appliance load. The appropriate kernel size of each convolutional layer depends on length of the input sequence and the previous layer. During model tuning, a number of window lengths, convolutional kernel sizes, and different numbers of convolutional windows were evaluated.

Computation time was also considered in selecting the kernel size. Larger kernel lengths significantly increased training time per epoch, without substantially improving model performance. This increase in computation time can be attributed to the dot product operation between the filter and the sequence, which increases exponentially as the kernel size increases. For the number of filters, the team found that between 30 and 50 filters per convolutional layer yielded the most accurate results.

The final model used a total of six convolutional layers, each with a progressively larger number of filters and smaller kernel size. Increasing the number of filters at each sequential convolutional layer allows the model to learn more granular features of patterns detected in the previous convolutional layers.

### *Pooling Layers*

The main goal for a pooling layer is to reduce the size of the feature maps by downsampling the input sequences. This approach produces deeper representations in the successive layers while avoiding overfitting. Pooling layers also reduce the total number of model parameters, thus reducing the time required to train the network.

During model tuning, a number of architectures with different combinations of pooling layers were explored. Initially, a max pooling layer was placed after each one-dimensional convolutional layer. However, this architecture reduced the width of the learned sequences too quickly and prevented the network from learning detailed waveform features. A revised approach placed pooling layers after every two convolutional layers. This architecture produced more accurate test results when training and testing the model on multiple houses.

### *Batch Normalization and scaling*

Batch normalization layers were added after every convolutional layer to scale the data and prevent covariate shift from developing during mode training. Batch normalization acts as a standard scaler for each batch that is passed to the model during training. Although the batch normalization algorithm adds some computation time during training, this time is made up by making the model converge more quickly.

The model input sequences were scaled using the Standard Scaler function from the python Sci-kit Learn package. This function subtracts each load value from the input mean and divides by the input standard deviation. Reducing the range of the input using a scaler makes model training more stable and typically allows the model to converge more quickly.

### *Dense layers*

The models used a flatten layer followed by a dense layer to combine the patterns learned by the convolutional and pooling layers. Initially the team used multiple dense layers in this portion of the network graph, but observed that more than one dense layer resulted in overfitting. Thus, the final model used only one dense layer after flattening the output from the convolutional layers.

### *Output Layer*

The model uses a single output neuron which represents the appliance load at the midpoint of the input sequence. Since the model performs regression, a rectified linear unit (ReLU) activation function was used with this single neuron to represent the inferred appliance load. The benefit of the ReLU activation function is that any negative predictions are floored at zero.

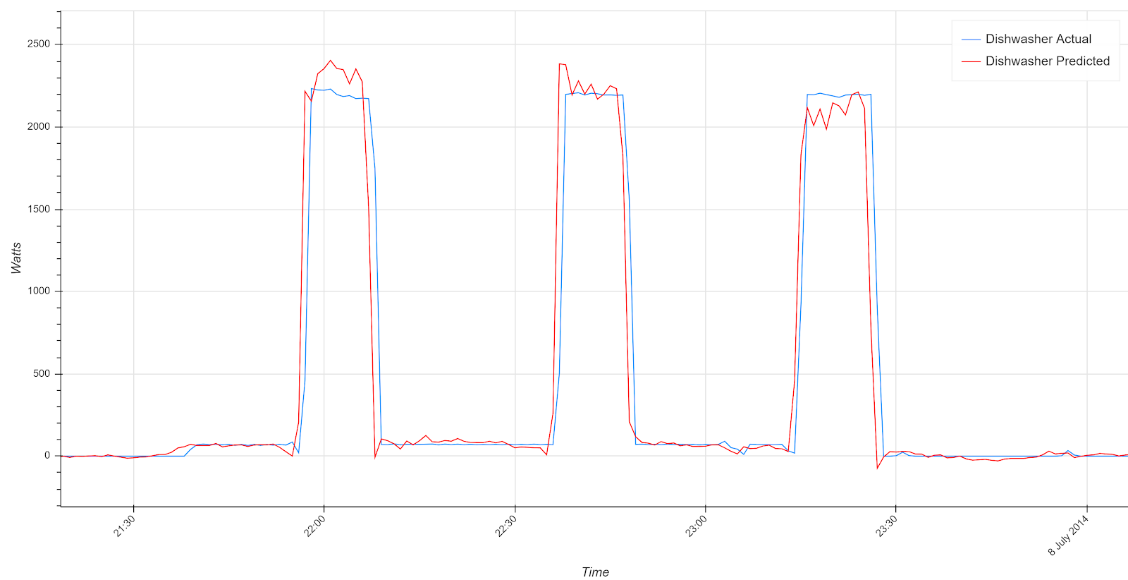


Figure 3: Sequence-to-point CNN Model Dishwasher Predicted and Actual Load Shapes

### *Single House vs Multiple House Performance*

It was discovered during model tuning that a neural network architecture that provides accurate results when training and testing on a single house does not necessarily perform as well when trained and tested on multiple houses. The model architecture must have enough complexity to recognize different appliance patterns from multiple homes, and not just a single home. To create a network large and complex enough to be scalable across a region, the model must be trained on submetered data from as many houses as possible to learn representative usage patterns. Since the REFIT dataset contains submetered data from 20 houses, the model was trained on ten or more houses per appliance type, which allowed for a variety of appliance manufacturers and usage patterns to be learned.

### *Learning Rate, Batch Size, and Number of Epochs*

The batch size hyperparameter specifies the number of rows to be processed before the model weights and biases are updated. Several different combinations of batch size were explored to determine the parameters that would provide the fastest convergence. With batch sizes above 1500, model loss became erratic. A batch size of 1000 rows allowed the model to converge quickly and smoothly for most appliances.

The learning rate hyperparameter specifies the step size used to update weights and biases while performing gradient descent. A learning rate that is too large will overstep the minimum point while a learning rate that is too small will make the model converge slowly. Based on best practices from previous research, the Tensorflow Adam optimizer was chosen to perform gradient descent. This optimizer keeps track of an exponentially decaying average of past gradients and an exponentially decaying average of past squared gradients, adjusting the learning rate as the model converges. The robust and adaptive nature of the Adam optimizer makes the initial learning rate hyperparameter less important. Thus, after trying several different learning rates, the recommended default learning rate of 0.001 was selected.

The Keras EarlyStopping callback feature was used to prevent overfitting. The stopping criteria was set to stop training if mean absolute error did not change by more than 0.1 in 3 epochs. Typically, the models required between 35-45 epochs before meeting the stopping criteria requirements.

### *Autoencoder Model Architecture*

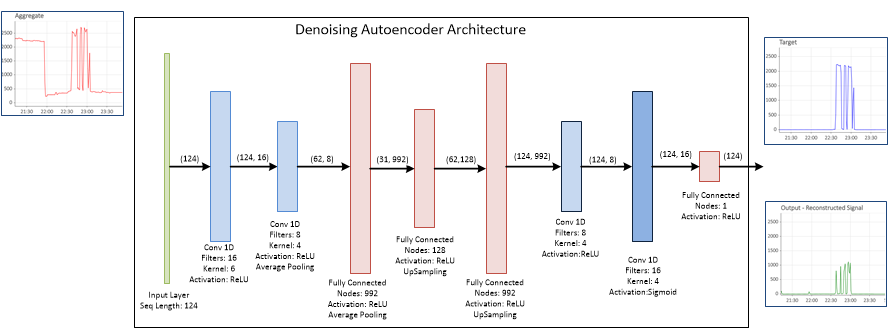


Figure 4: Autoencoder Model Architecture

### *Input Layer*

The input to the Autoencoder is a time window of the aggregate data, with a sequence length that varies per appliance. Some appliances of very short activation cycle (i.e. a toaster with duration of 2 min) can be captured in a short time sequence, whereas some appliances (i.e. washing machine at 120 min) need a much larger time window to properly capture the activation.

### *Scaling the Target*

Standard scaling was applied to the target values, which resulted in much smaller values of the loss and the reconstructed output. The result of the scaling is a signal with a mean of 0 and standard deviation of 1. Doing so provided 2 advantages: much smaller loss values and much faster convergence.

### *Encoder*

The Encoder section of the Autoencoder consists of two convolutional layers and two fully connected layers. The convolutional layers find the initial broad features in the aggregate signal input. The first fully connected layer is sized such that each of the feature maps generated corresponds to one node. The next fully connected layer reduces the number of nodes drastically, and this layer is known as the code layer or “bottleneck” where the network is forced to learn a compressed representation of the data. The activation on each of these layers is Rectified Linear Unit (ReLU), which preserves the computed values and does not allow negative values.

### *Decoder*

The Decoder section of the Autoencoder consists of one fully connected layer and two convolutional layers. The fully connected layer reconstructs the target from the latent space representation. The convolutional layers recreate the final reconstructed signal that most closely aligns with the target. The sigmoid activation on the output is intended to activate each output neuron and although it takes longer to converge than ReLU, it cannot saturate at zero like ReLU can. The final fully connected layer delivers the output shape of the reconstructed signal.

### *Batch Normalization Layers*

These layers are used throughout the model after each of the convolutional and fully connected layers. These layers do not affect the output shape or contribute any new learning parameters. They simply normalize the outputs at each layer and thus prevent the vanishing/exploding gradients problem.

### *Pooling Layers*

Normally pooling layers are introduced to reduce the data size at a given layer and prevent overfitting. The typical max pooling approach of taking the maximum value within a window did not seem appropriate to this task since we are working with continuous values of aggregate load and not discrete features. Hence, average pooling was used in which the average value within a given window is assigned and the data size at that layer is reduced in half. Introducing these layers did not actually improve performance however, because the bottleneck layer was reduced too severely. Combining the average pooling layers in the Encoder with upsampling layers in the Decoder to increase the data shape at a given layer did yield an improvement in performance when the model complexity increased during development.

### *Model Training improvement by comparing output to aggregate*

Most previous energy disaggregation researchers have used a linear approach, where each appliance type is disaggregated using a separate model. To account for dependencies between the appliance states during training, the outputs of the individual appliance models are added and compared to the aggregate of the appliances during backpropagation This additional constraint forces the model to resolve conflicts when the sum of the appliances is greater than the aggregate. Based on preliminary results, this method did not show an improvement in model accuracy. However, the proposed method should be explored f

### *Effect of Architecture on Training Time*

In order to deliver a scalable algorithm, the model should not require a significant amount of training time. Some architecture combinations were explored that had the potential to yield more accurate results, but these significantly increased the training time required. For example, increasing the kernel size in convolutional layers and the step size of the input sequences exponentially increase the training time per epoch. The final model architecture was a compromise between performance that was good enough and model that was not too complex.

### *Single-input-multiple-output Model*

Each of the four models were combined into a single model that takes the aggregate data as an input and produces the inferred load for each appliance as an output. Creating a “wide and deep” neural network will allow a single model to be deployed rather than multiple models for each appliance.

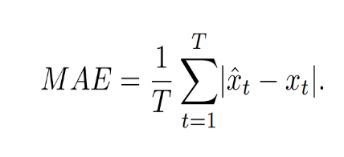
# Modeling Results

The quality of the disaggregation models was assessed in a few ways. For the HMM and FHMM (classification models), we measured the ability of the model to accurately predict the state of the appliance. For the neural networks (regression models) we measured performance upon whether the disaggregation algorithm can correctly detect the time intervals when the target appliance consumes energy and the actual load value at that time. How well the disaggregation algorithm reproduced the shape and magnitude of the target appliance load were also used to evaluate model performance.

## Regression Model Performance Metrics

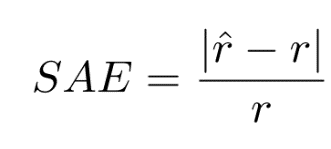
The following performance metrics were used to evaluate the performance of the disaggregation algorithms:

* Mean Absolute Error (MAE):



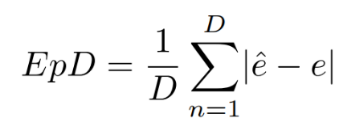
Breakdown:

* + is the predicted load
  + is the ground truth at every time step.
* Signal Aggregate Error (SAE):

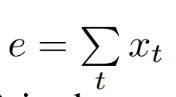


Breakdown:

* + is the total energy consumption of the appliance
  + is the predicted total energy of the appiance.
* Energy Per Day (EPD) Error:



Breakdown:

* +  the energy consumed in a one-day period
  + represents the total number of days

These metrics were used to compare model performance to the baseline models included in the NILMTK as well as models from other energy disaggregation researchers.

## Regression Model Training

Convolutional Model

Table 2: Convolutional Sequence-to-point Model Training and Test Houses

| **Appliance** | **Training Houses** | **Test Houses** |
| --- | --- | --- |
| Dishwasher | 1, 6, 7, 9, 10, 13, 15, 16, 18, 21 | 2, 3, 20 |
| Washing Machine | 1, 6, 7, 9, 13, 15, 16, 17, 19, 21 | 2, 3, 20 |
| Refrigerator | 1, 7, 8, 9, 10, 15, 16, 17, 19, 21 | 2, 3, 20 |
| Freezer | N/A | 2, 3, 20 |
| Kettle | 4, 6, 7, 8, 9, 12, 13, 15, 17, 19 | 2, 3, 20 |
| Microwave | 4, 6, 8, 9, 10, 13, 15, 17, 19 | 2, 3, 20 |
| Toaster | N/A | 2, 3, 20 |
| Tumble Dryer | N/A | 2, 3, 20 |

Autoencoder Model

Table 3: Autoencoder Training and Test Houses

| **Appliance** | **Training Houses** | **Test Houses** |
| --- | --- | --- |
| Dishwasher | 5, 9, 16, 13, 7, 3 | 6, 2, 1 |
| Washing Machine | 15, 18, 20, 13, 7 ,6 | 9, 4 |
| Refrigerator | 2, 15, 16,20, 21, 18 | 19, 17, 7 |
| Freezer | 4, 17, 3, 20 | 18, 6 |
| Kettle | 12, 13, 9, 8, 11, 15 | 4 |
| Microwave | 20, 15, 8, 11, 13, 17 | 4, 19, 12 |
| Toaster | 2, 6, 19, 8, 12, 5 | 7, 15 |
| Tumble Dryer | 15, 20 | 7 |

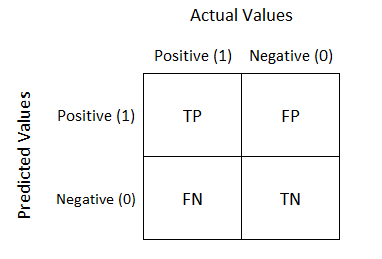
## Regression Model Results

Table 4: Combined Neural Network Model Test Results

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Convolutional Model** | | | **Autoencoder Model** | | |
| **Appliance** | **MAE** | **SAE** | **EpD** | **MAE** | **SAE** | **EpD** |
| Dishwasher | 29.76 | 0.26 | N/A | 24.38 | 0.16 | 262.38 |
| Washing Machine | 17.08 | 0.42 | N/A | 20.57 | 0.42 | 267.12 |
| Refrigerator | 20.54 | 0.21 | N/A | 36.21 | 0.72 | 390.47 |
| Freezer | N/A | N/A | N/A | 52.70 | 1.06 | 592.60 |
| Kettle | 8.48 | 0.089 | N/A | 13.48 | 0.07 | 107.11 |
| Microwave | 3.68 | 0.5 | N/A | 8.26 | 0.52 | 85.94 |
| Toaster | N/A | N/A | N/A | 2.93 | 1.59 | 30.80 |
| Tumble Dryer | N/A | N/A | N/A | 67.26 | 0.25 | 692.42 |

## Classification Model Performance Metrics

The following performance metrics were used to evaluate the performance of the Factorial Hidden Markov Model:



* Sensitivity:
* Specificity:
* Precision:
* Accuracy:
* F1 Score:

## Probabilistic Classification Model Training

Each house was split into a training and test set, 75% and 25% respectively. Each appliance in the home was trained on its individual power usage which was provided in the REFIT data set. Once the model was constructed, the aggregated house consumption in the test set was used to predict the states of each model during the time range in the test set. The results from the model to the ground truth of the appliance state in the test set were compared.

Table 5: Summary of Probabilistic Model Training

|  |  |  |
| --- | --- | --- |
| **House #** | **# of Appliances Trained** | **Sampling Rate** |
| 5 | 9 | 1-min, 15-min |
| 15 | 9 | 1-min, 15-min |
| 18 | 9 | 1-min, 15-min |

## Probabilistic Classification Model Results

Table 6: Appliance References

| **Appliance Name** | **Abbreviation** |
| --- | --- |
| Fridge-Freezer | FF |
| Washing Machine | WM |
| Dishwasher | DW |
| Computer Site | CPU |
| Television Site | TV |
| Microwave | MW |
| Kettle | K |
| Toaster | T |
| Refrigerator | R |
| Freezer | F |
| Dryer | D |

Table 7: Probabilistic Model 15-minute Results

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **15 Minute Intervals** | | | | | | | | | | |
| **House 15** | | | | | | | | | | |
|  | **Overall** | **FF** | **D** | **WM** | **DW** | **CPU** | **TV** | **MW** | **K** | **T** |
| **Sensitivity** | 55.5% | 53.6% | 27.8% | 16.5% | 23.3% | 65.2% | 56.3% | 61.5% | 55.0% | 42.9% |
| **Specificity** | 77.6% | 30.5% | 99.0% | 97.1% | 91.2% | 60.8% | 58.6% | - | 97.7% | 60.0% |
| **Precision** | 40.9% | 44.5% | 16.8% | 45.7% | 1.9% | 5.5% | 31.2% | 100.0% | 37.6% | 0.2% |
| **Accuracy** | 72.8% | 42.3% | 98.4% | 86.8% | 90.7% | 61.0% | 58.0% | 61.5% | 96.7% | 60.0% |
| **F1** | 47.1% | 48.6% | 20.9% | 24.3% | 3.5% | 10.1% | 40.1% | 76.2% | 44.7% | 0.4% |
|  | | | | | | | | | | |
| **House 5** | | | | | | | | | | |
|  | **Overall** | **FF** | **D** | **WM** | **DW** | **CPU** | **TV** | **MW** | **K** | **T** |
| **Sensitivity** | 39.4% | 43.5% | 23.9% | 48.8% | 21.0% | 48.1% | 51.7% | 62.3% | 50.7% | 43.7% |
| **Specificity** | 68.3% | - | 83.7% | 70.6% | 81.6% | 50.9% | 51.4% | 52.8% | 75.4% | 81.5% |
| **Precision** | 40.7% | 100.0% | 76.7% | 29.6% | 35.6% | 59.1% | 16.3% | 15.3% | 14.7% | 6.2% |
| **Accuracy** | 58.1% | 43.5% | 42.3% | 66.2% | 61.8% | 49.2% | 51.5% | 54.0% | 73.5% | 80.4% |
| **F1** | 40.0% | 60.7% | 36.4% | 36.9% | 26.4% | 53.1% | 24.8% | 24.6% | 22.8% | 10.8% |
|  | | | | | | | | | | |
| **House 18** | | | | | | | | | | |
|  | **Overall** | **R** | **F** | **FF** | **D** | **WM** | **DW** | **CPU** | **TV** | **MW** |
| **Sensitivity** | 44.2% | 46.9% | 55.7% | 57.5% | 25.8% | 62.8% | 48.1% | 45.4% | 42.7% | 23.4% |
| **Specificity** | 76.4% | 55.5% | 63.6% | 0.0% | 78.3% | 74.3% | 96.9% | - | 100.0% | - |
| **Precision** | 69.5% | 34.2% | 65.6% | 99.9% | 0.7% | 4.5% | 35.1% | 100.0% | 100.0% | 100.0% |
| **Accuracy** | 58.7% | 52.7% | 59.2% | 57.5% | 77.9% | 74.0% | 95.3% | 45.4% | 42.7% | 23.4% |
| **F1** | 54.0% | 39.6% | 60.3% | 73.0% | 1.4% | 8.3% | 40.6% | 62.5% | 59.8% | 37.9% |

Table 8: Probabilistic Model 1-minute Results

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1 Minute Intervals** | | | | | | | | | | |
| **House 15** | | | | | | | | | | |
|  | **Overall** | **FF** | **D** | **WM** | **DW** | **CPU** | **TV** | **MW** | **K** | **T** |
| **Sensitivity** | 32.6% | 90.7% | 23.1% | 38.7% | 8.0% | 94.9% | 81.3% | 0.7% | 39.4% | 22.6% |
| **Specificity** | 73.3% | 5.4% | 99.6% | 95.5% | 98.6% | 48.6% | 2.8% | 99.0% | 99.5% | 99.2% |
| **Precision** | 21.3% | 30.7% | 16.3% | 30.8% | 1.5% | 5.5% | 20.0% | 99.3% | 25.7% | 0.8% |
| **Accuracy** | 65.9% | 32.3% | 99.3% | 92.8% | 98.3% | 50.0% | 20.9% | 1.1% | 99.2% | 99.1% |
| **F1** | 25.7% | 45.9% | 19.1% | 34.3% | 2.5% | 10.4% | 32.1% | 1.4% | 31.1% | 1.6% |
|  | | | | | | | | | | |
| **House 5** | | | | | | | | | | |
|  | **Overall** | **FF** | **D** | **WM** | **DW** | **CPU** | **TV** | **MW** | **K** | **T** |
| **Sensitivity** | 33.5% | 36.6% | 21.2% | 41.3% | 16.9% | 37.9% | 49.7% | 44.3% | 63.8% | 57.6% |
| **Specificity** | 75.4% | 58.1% | 84.6% | 62.9% | 84.2% | 61.8% | 59.4% | 53.4% | 96.0% | 93.7% |
| **Precision** | 37.8% | 99.7% | 52.9% | 20.4% | 33.3% | 58.6% | 15.6% | 8.4% | 19.1% | 5.6% |
| **Accuracy** | 62.4% | 36.7% | 56.0% | 58.9% | 62.8% | 47.8% | 58.2% | 52.6% | 95.5% | 93.5% |
| **F1** | 35.5% | 53.6% | 30.2% | 27.3% | 22.5% | 46.1% | 23.7% | 14.1% | 29.4% | 10.2% |
|  | | | | | | | | | | |
| **House 18** | | | | | | | | | | |
|  | **Overall** | **R** | **F** | **FF** | **D** | **WM** | **DW** | **CPU** | **TV** | **MW** |
| **Sensitivity** | 37.0% | 42.7% | 43.1% | 44.0% | 33.7% | 47.0% | 36.4% | 41.2% | 47.7% | 12.8% |
| **Specificity** | 82.5% | 59.7% | 71.6% | 74.0% | 95.9% | 81.5% | 98.6% | 50.0% | 55.8% | 82.4% |
| **Precision** | 66.9% | 13.7% | 40.4% | 95.1% | 1.4% | 3.7% | 40.7% | 100.0% | 99.9% | 100.0% |
| **Accuracy** | 60.2% | 57.5% | 62.8% | 46.4% | 95.8% | 81.0% | 97.0% | 41.2% | 47.7% | 12.8% |
| **F1** | 47.6% | 20.7% | 41.7% | 60.1% | 2.7% | 6.8% | 38.4% | 58.3% | 64.6% | 22.7% |

# Analysis of Results

The model test results showed that the convolutional and autoencoder models can successfully infer appliance load with reasonable accuracy using one-minute sampling intervals. Since model test results can vary depending on the house, three houses were set aside for testing and their test results were averaged. The models showed the highest accuracy for appliances with distinct patterns and relatively infrequent usage such as the dishwasher, washing machine, and kettle. For appliances that are frequently cycling, such as refrigerators and freezers, the models produced less accurate but still sufficient to provide an approximate energy usage estimate.

The probabilistic classification models performed differently than the neural nets when comparing energy patterns. The model favored appliances that were always on compared to those that had a distinct energy pattern. For example, the fridge-freezer F1-score outperformed the other appliances as it was always on and consumed majority of the homes aggregate output. The toaster, which was rarely on and consumed the least of the aggregate output, consistently had the lowest F1-score.

In order to improve the performance of the Factorial Hidden Markov Models we would recommend making the following adjustments:

* + Allow the number of states for each appliance to expand to 3 more. We assumed that each appliance only has two states, ON and OFF. It’s possible that an appliance could have an ‘ON-High’, ‘ON-Low’, and ‘OFF’.
  + Introduce additional variables in the model that the hidden state may be dependent on such as:
    - Time of Day
    - Day of the Week
    - Temperature
  + Account for a longer duration of the appliance state by modeling the hidden state as a semi-Markov process.

# Anomaly Detection

This section presents a prototype unsupervised anomaly detection algorithm that uses the disaggregated load data to detect appliance-specific anomalies. Previous sections demonstrated how autoencoders can be used to infer individual appliance energy consumption from the aggregate smart meter data. The same autoencoder architecture can also be used to simply to reproduce the input. After training this autoencoder on several examples of the same appliance from different houses, the model will learn typical patterns associated with the appliance. Therefore, if the trained model is tested using an input that does not represent the training appliance, the model will become “confused” and will not accurately reproduce the input. This application of autoencoders is similar to techniques used in fraud detection for credit card transactions.

Once the appliance input has been reconstructed, a metric must be selected to determine the distance between the input sequence and reconstructed sequence. The method selected utilizes a mathematical technique known as the discrete wavelet transform to measure the difference between the two waveforms. Wavelet transforms are similar to [Fourier analysis](https://en.wikipedia.org/wiki/Fourier_analysis) in that it breaks down a target function into several different waveforms that, when combined, reconstruct the original signal. The advantage of wavelet transforms over Fourier transforms is that they are more applicable to low frequency time-series data.[[9]](#footnote-11)

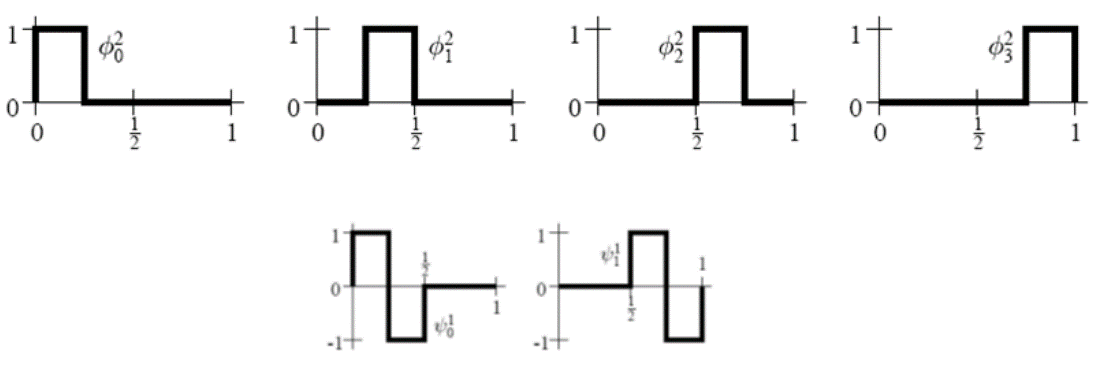
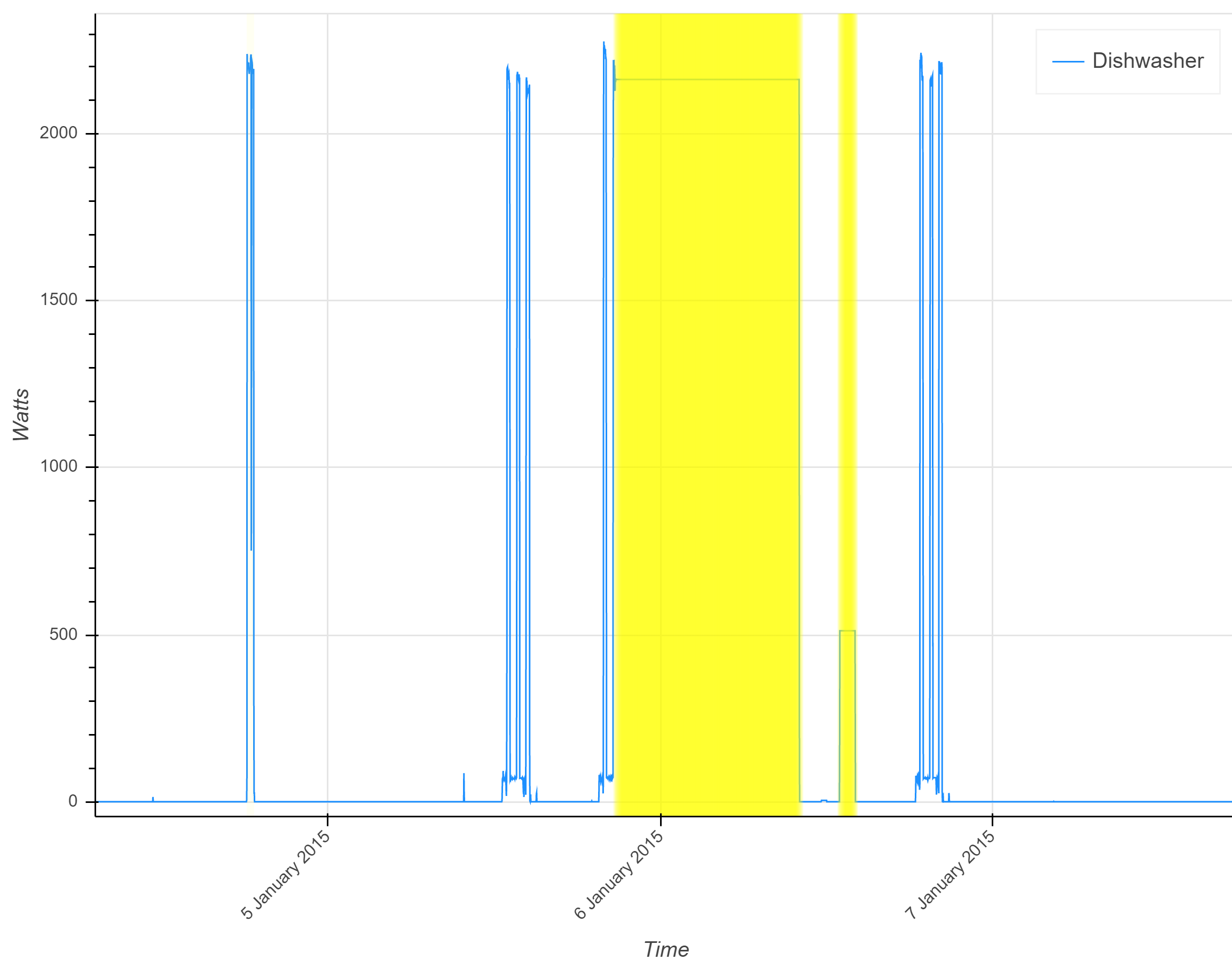


Figure 5 – Example Haar Wavelet Square Wave Basis Functions

The Haar discrete wavelet transform decomposes the input sequence into a series of square waveform shapes, each with a coefficient which weights its importance to the overall waveform shape. The mean squared error of the coefficients serves as a measure of the difference between the input and reconstructed sequence. Therefore, the discrete wavelet transform coefficients are calculated for a list of sequences that move forward one step at a time. If the mean squared error of a sequence exceeds a specified threshold, then the entire sequence is marked as an anomaly.

A challenge with implementing an unsupervised anomaly detection algorithm is that there are no labeled anomalies against which to measure performance. To provide some measure of performance, the model was tested on some sensor error anomalies in the REFIT dataset. An example anomaly is shown below.

  
Figure 6 - Example Anomaly labeled by anomaly detection model

The anomaly detection model could be used to provide near real-time notifications to customers or annotate anomalous periods on an energy bill so users can prevent future anomalies. The cause of the anomaly may be either a problem with the appliance or an error with the smart meter reading. A scalable anomaly detection model will require further testing and model tuning. However, the prototype autoencoder and discrete wavelet transform anomaly detection model has shown promising preliminary results.

# Proposed Applications

The following sections show visualizations for use cases of how disaggregated energy data can be used to make homeowners aware of their energy usage and prompt them to be more efficient and save money on their energy bill. An interactive energy usage dashboard would highlight ways that users could shift their appliance usage away from system peak load times (typically 4-10 pm) to minimize their energy bill and reduce stress on the system.

## Interactive Energy Bill Dashboard

The energy usage dashboard would use disaggregated appliance load data to create an itemized energy bill, showing a breakdown of household energy usage by appliance. By visualizing the energy usage in an interactive dashboard, the user can explore their household data and gain insights into how they can modify their behavior to reduce their energy bill. The Tableau visualization tool provides many built-in interactive widgets which make it easy for the user to drill down to specific time periods and appliances. The dashboard may also show how much money can be saved by shifting appliance usage to off-peak times.

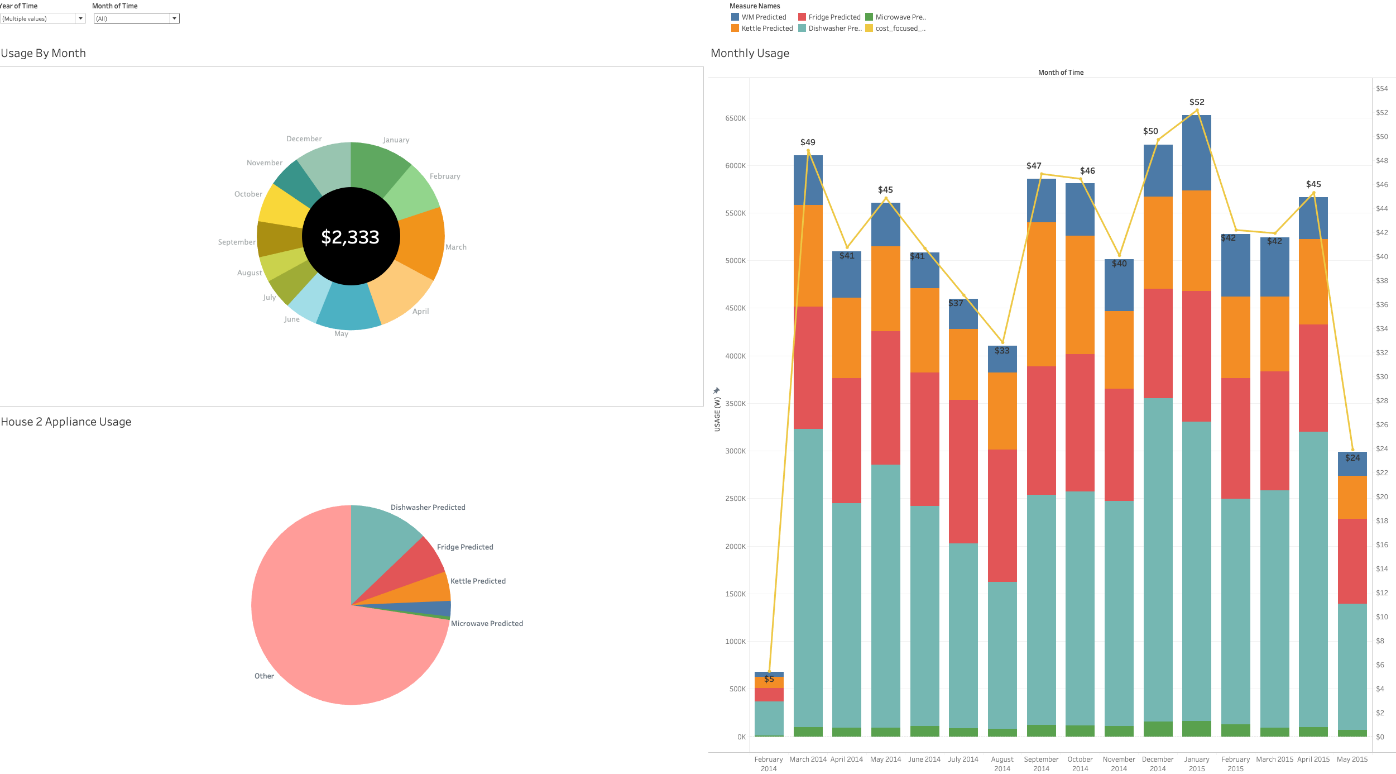


Figure 7: Prototype Itemized Energy Bill Dashboard

### Histogram of Disaggregated Energy Usage

Another portion of the interactive dashboard is a histogram of energy usage for each appliance placed in bins representing each hour of the day. Overlaid on top of the histogram is the Time-of-use energy rates used to bill the customer from the local electric utility for each hour of the day. This visualization can show users how using an appliance at peak times significantly increases their monthly electricity bill. Users can also identify times when energy is cheaper and shift their appliance usage to those times.

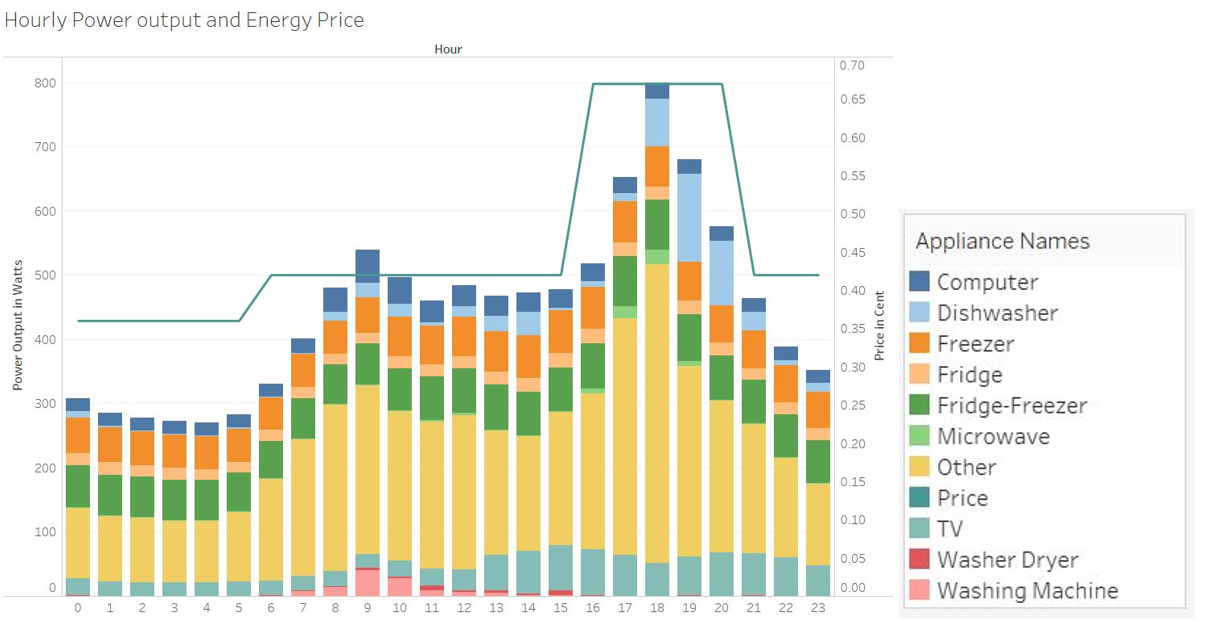


Figure 8: Histogram of Disaggregated Energy Usage and Time-of-Use Rates

### Conceptual Design for a Scalable Energy Disaggregation System

This section outlines the steps required to deploy an energy disaggregation system for an electrical utility’s entire service territory.

## Data Collection

The first step to deploy disaggregation algorithms is to create a pilot program to collect data for model training. The utility company should use incentives to identify at least 100 homes willing to install sensors on their appliances for a period of 6-12 months. Inexpensive power measurement sensors can be used for data collection as long as sampling intervals are one minute or less. The houses selected for the program should include a variety of locations and residence types so that the dataset is a representative sample of the customer base. As seen in the REFIT dataset, there may be a number of sensor failures and power outages, so the data collection system should be resilient to disturbances. In addition to the submetered appliance data, the aggregate smart meter data for the houses should also be collected at intervals of one minute or less.

## Model Training

After the data collection period, the data should be inspected and cleaned to remove spikes, gaps, and anomalies. If a large portion of data from a house is corrupted, the house should be excluded from the dataset. After cleansing the data, the individual appliance models should be trained on as many houses as feasible using GPUs from a cloud or on-premise cluster. To reduce the training time, a transfer learning approach could be used by starting with pre-trained models from other open-source datasets, such as REFIT.

## Create Data Pipeline

To put the model into production, utility data engineers will need to create a data pipeline between the Meter Data Management System (MDMS), the disaggregation model, and the visualization platform. For a large utility, this may require using a data migration system such as Talend to perform Extract, Transform, and Load (ETL) processes between the smart meter data historian and a time-series database such as MongoDB.

The pipeline should establish batch jobs to be run at regular intervals, such as weekly or monthly, to perform disaggregation on the smart meter data. This disaggregated data will also need to be stored in time-series database, which can be accessed by other applications via an enterprise service bus.

## Create Visualizations

The last step will be to develop the web applications that customers and utility employees can use to visualize and explore the disaggregated appliance data. The applications may also include dashboards created using applications such as Tableau or Microsoft PowerBI that summarize appliance usage for districts and regions. The disaggregated appliance data could also be used to create an itemized usage visualization included in monthly energy bills.

## Model Retraining

To ensure that models are accurate and able to detect patterns in new appliances, the data collection and model training process should be repeated every 3-5 years. The retraining process should incorporate any new appliances or energy resources that have shown substantial growth since the previous training period. For example, at some point utilities may want to add an electric vehicle model during retraining due to the substantial growth in electric vehicle adoption in the US.

### *Conclusions*

In this Capstone project our team met the objective of performing energy disaggregation on smart meter data. Several accurate energy disaggregation models were developed, trained, and tested on houses in the REFIT dataset. An anomaly detection model was developed that could identify anomalous patterns in appliance performance or usage. Since anomalies are by definition unpredictable, this model could be enhanced with further training and labeled anomalies. The single-input-multiple-output (SIMO) model represents an effort to combine individual appliance models into an ensemble of models that can capture interactions between appliances and provide results on an entire household at once. Although this model could benefit from further development, it is a very promising concept and would be a novel approach to energy disaggregation that has not been seen in the literature.

With the rationale for why energy disaggregation can be beneficial to both energy providers and consumers, our team has set out a plan for disaggregation at scale, which would include a whole infrastructure and training plan that an energy producer could implement.

# References

Daniel Jurafsky, J. H. (2018). Speech and Language Processing. Ebook.

Haroon Rashid, V. S. (n.d.). Annotated load anomalies from the REFIT Dataset. Retrieved from University of Strathclyde Glasgow: https://pureportal.strath.ac.uk/en/datasets/annotated-load-anomalies-from-the-refit-dataset

Haroon Rashid, V. S. (n.d.). Evaluation of non-intrusive load monitoring algorithms for appliance-level anomaly detection. Retrieved from University of Strathclyde Glasgow: https://pureportal.strath.ac.uk/en/publications/evaluation-of-non-intrusive-load-monitoring-algorithms-for-applia

Haroon Rashid, V. S. (n.d.). REFIT: Electrical Load Measurements (Cleaned). Retrieved from University of Strathclyde Glasgow: https://pureportal.strath.ac.uk/en/datasets/refit-electrical-load-measurements-cleaned

Jack Kelly, W. K. (2015). Neural NILM: Deep Neural Networks Applied to Energy Disaggregation. ACM BuildSys’15. Seoul, South Korea.

NILMTK Team. (2018, August 18). NILMTK: Non-Intrusive Load Monitoring Toolkit. Retrieved from GitHub: https://github.com/nilmtk/nilmtk/blob/master/README.md

Zoubin Ghahramani, & Jordan, M. I. (1997). Factorial Hidden Markov Models. Machine Learning, 1-37.

Chaoyun Zhang, Mingjun Zhong, Zongzuo Wang, Nigel Goddard, and Charles Sutton. Sequence-to-point learning with neural networks for nonintrusive load monitoring. In The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), 2018.

# Taspinar, Anhet, *A guide for using the Wavelet Transform in Machine Learning*, 2018,

<http://ataspinar.com/2018/12/21/a-guide-for-using-the-wavelet-transform-in-machine-learning/> , Accessed on 7/14/2019.

###### Appendix

***Dataset Information***

## Example: Metadata for one house

|  |  |
| --- | --- |
| **House Number** | **18** |
| Building Type | Detached |
| Energy Improvements | None |
| Heating | N/A |
| Construction year | 1965-1974 |
| Size | 3 bedrooms |
| Number of occupants | 2 |
| Appliance owned | 34 |
| Metered Appliances | 14 |
| Number of meters | 9 |
| Date first measurement | March 1, 2013 |
| Date last measurement | March 15, 2015 |
| Total duration (days) | 745 |
| Uptime proportion | 98% |
| Proportion of energy submetered | 54% |

## Example: Statistics for House 18

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Appliance** | **# of activations** | **Avg power of activation [w]** | **Median power of activation [w]** | **Avg duration of activation [sec]** | **Median duration of activation [sec]** |
| Fridge(garage) | 10014 | 105.14 | 103.5 | 655.96 | 420.00 |
| Freezer(garage) | 10289 | 161,89 | 160.68 | 1249.26 | 1080.00 |
| Fridge-Freezer | 22602 | 111.30 | 116.45 | 832.93 | 780.00 |
| Washer Dryer | 98 | 482.18 | 223.49 | 605.79 | 120.00 |
| Washing Machine | 238 | 321.18 | 323.81 | 1907.65 | 1920.00 |
| Dishwasher | 1177 | 624.40 | 114.62 | 514.66 | 300.00 |
| Desktop Computer | 1128 | 22.00 | 19.08 | 31890.37 | 4620.00 |
| Television Site   1. Television 2. DVD player 3. Sky Box 4. Speakers 5. Lamp 6. HiFi | 2120 | 76.20 | 56.23 | 7121.87 | 6230.00 |
| Microwave | 393 | 482.15 | 456.11 | 357.40 | 120.00 |

1. Note\*: The team did not find any papers to date that have used deep learning to perform load disaggregation on the REFIT dataset. [↑](#footnote-ref-2)
2. (Daniel Jurafsky, 2018) [↑](#footnote-ref-3)
3. The ‘Forward-Backward’ algorithm is defined in the literature under “Dynamic Programming” which essentially means that it’s an “optimization algorithm”. With regards to HMM’s, it used to determine the probability of the hidden state variable given your observations, . [↑](#footnote-ref-4)
4. The ‘Junction Tree’ algorithm, used for exact inference in a Bayesian Networks, is a generalization of the ‘Forward-Backward’ algorithm. [↑](#footnote-ref-5)
5. (Zoubin Ghahramani & Jordan, 1997) [↑](#footnote-ref-7)
6. (NILMTK Team, 2018) [↑](#footnote-ref-8)
7. To allow for exact inference in a Factorial HMM the number of state variables was kept low, nine appliances max, and the number of possible states per random variable at two. [↑](#footnote-ref-9)
8. Chaoyun Zhang, Mingjun Zhong, Zongzuo Wang, Nigel Goddard, and Charles Sutton. Sequence-to-point learning with neural networks for nonintrusive load monitoring. In The Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), 2018. [↑](#footnote-ref-10)
9. # Taspinar, Anhet, *A guide for using the Wavelet Transform in Machine Learning*, 2018, <http://ataspinar.com/2018/12/21/a-guide-for-using-the-wavelet-transform-in-machine-learning/> , Accessed on 7/14/2019.

   [↑](#footnote-ref-11)