Predicting Building Energy Consumption Using Gaussian Process Regression: Algorithms, Visualization and Web Applications

Predicting Building Energy Consumption Using Gaussian Process Regression: Algorithms, Visualization and Web Applications

Bin Yan

A Thesis in the Field of Information Technology

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

March 2018

Abstract

Growing attention has been drawn to energy use forecasting for smart grid applications. Among different modeling methods of predicting building energy use, Gaussian process (GP) regression has its advantages since it not only outputs the mean value of the prediction, but also the confidence range. In this study, I have developed a web application that allows users without programming skills to predict and visualize energy demand through GP regression. The web application implements both baseline prediction and next-day prediction. This study also explores the visualization techniques that facilitate the analysis of energy consumption patterns and transform the data into informative insights. This thesis presents two case studies to demonstrate the use of the web application and discusses the prediction accuracy. The first case study predicts electric energy, chilled water and steam consumption of a campus building. The second case study predicts the next-day electric energy demand of a high-tech industry area.

Acknowledgments

I would first like to thank my thesis advisor Dr. Peter V. Henstock who generously put his time and efforts to help me through the thesis process. He encouraged me to develop a web application so that my research can have more impact and he provided detailed and valuable guidance from technology choices to writing. I really enjoyed this process and have learned a lot. Thank you for all the guidance.

I would also like to acknowledge Dr. Jeff Parker at Harvard Extension School, who helped me with my proposal and provided assistance throughout the thesis process. I am grateful to the staff at the Harvard Extension School who provided administrative assistance. I want to thank my supervisor Prof. Ali Malkawi at Harvard Center for Green Buildings and Cities. I chose a topic related to my full-time research job and Prof. Malkawi has always been supportive of my personal pursuits and career.

Finally, I must express my gratitude to my husband Vilhelm Sjoeberg, who is a computer science research scientist at Yale University. It is a privilege to have a coding expert that I can always turn to for help. For example, he figured out some Heroku issues I had trouble with. Thank you.

Table of Contents

List of Tables

List of Figures

Chapter I.

Introduction

1.1 Background

There is growing attention drawn to forecasting building energy usage as it can be applied to fault detection and diagnosis[1], operation optimization and demand response in a smart grid. Energy use forecasting has been widely used in building commissioning to determine retrofit savings (Cohen & Krarti, 1995; Kissock, Reddy, & Claridge, 1998). It is also essential in fault detection since a baseline prediction is necessary to detect abnormalities in energy consumption (B. Yan & Malkawi, 2013). The ability to accurately forecast demand-side loads plays a critical role in electric power systems, especially since the future power grid is expected to provide unprecedented flexibility in how energy is generated, distributed and managed (Burger & Moura, 2015; Jetcheva, Majidpour, & Chen, 2014). Power plant level forecasting is crucial to the planning and operation of utility companies, while building-level load forecasting is crucial to building owners to reduce the electricity charges by optimizing electricity purchase strategies. Forecasting both levels is important for design of microgirds and intelligent distribution systems as well as the implementation of demand response (Kwac, Flora, & Rajagopal, 2014).

[1] Fault detection and diagnosis is a process of monitoring a system, determining whether a fault has occurred and which fault occurred or the cause of the observed out-of-control status. (Chiang, Leo H., Richard D. Braatz, and Evan L. Russell. Fault detection and diagnosis in industrial systems. Springer Science & Business Media, 2001.)

1.2 Literature Review

The literature is rich with forecasting methods for grid-level and building-level loads. Perhaps the most attention has been given to the approaches such as artificial neural networks (ANN), Support Vector Regression (SVR) and Autoregressive Integrated Moving Average (ARIMA) models (Burger & Moura, 2015). Multiple Linear Regression, Fuzzy Logic, Decision Trees, and k-Nearest Neighbors (k-NN) methods are also widely used (Burger & Moura, 2015; Majidpour, Qiu, Chu, Gadh, & Pota, 2015). There has been a surge of interest in Gaussian process (GP) modeling following recent advances in the machine learning community (Neal, 1995; Rasmussen, 1996). Gaussian Processes have been successful in solving many real-word data modeling problems (MacKay, 1997). Recently, Gaussian process modeling has been adopted to forecast thermal dynamics and energy consumption in buildings (Kim, Ahn, Park, & Kim, 2013; Manfren, Aste, & Moshksar, 2013; J. Yan, Kim, Ahn, & Park, 2013).

Leith, Heidl, and Ringwood (2004) examined models based on GP priors for electrical load forecasting. In their GP models, the features are the order of the data point i and the electrical load of the previous time step y_{i-1}. The covariance functions account for trend, seasonal component, autocorrelation and noise in the data. They compared the prediction accuracy of using Basic Structural Models (BSM) and Seasonal Auto-Regressive Integrated (SARI) models with GP models for weekly electrical demand from a mixture of domestic, commercial and industrial users in Ireland. The GP results were better than BSM and SARI models.

Gray and Schmidt (2016) developed a GP regression model to predict the day-ahead hourly zone temperature in a building. Their study uses synthetic data simulated by

TRNSYS (A TRaNsient SYstems Simulation Program2). The input variables include ambient temperature, solar gains, heating medium mass flow, number of occupants, hour of day, day of week, heating schedule and calculated zone temperature of previous days. There are 18 weeks of data for training and 10 for testing. The study compares models with different training periods: 72-hour (three days), 168-hour (one week), 504-hour (three weeks) and 1008-hour (six weeks). Longer training periods lead to lower day-ahead prediction errors. The study also compares GP regression with a grey-box model. The grey-box model is a series of partial differential equations that describe thermal dynamics. It is based on physical principles but simplifies the mathematical system description by lumping certain parameters. When using a training period of three weeks or longer to predict the day-ahead zone temperature during occupied hours, GP achieves a lower prediction error than the grey-box model. However, in their case study, the grey-box model has a lower total prediction error, requires less training and is less sensitive to input data not present in the training dataset than GP. The sensitivity to unknown data is a limiting factor for GP prediction, especially when a reduced amount of training data is available.

Heo, Choudhary, and Augenbroe (2012) presented a Bayesian approach, which involves GP regression, to calibrate normative energy models. The mathematical model is based on the work of Kennedy and O'Hagan (2001). The GP formulation is used to compute the likelihoods p of observations y given model parameters $\boldsymbol{\theta}$, $p(y|\boldsymbol{\theta})$. In their case study, $\boldsymbol{\theta}$ includes window opening, indoor temperature during heating, infiltration rate and discharge coefficient. The y variable to be calibrated is monthly gas

[2] http://sel.me.wisc.edu/trnsys/

consumption in their study. Later Heo and Zavala (2012) developed a Gaussian process modeling framework to determine energy savings in measurement and verification (M&V) practices. Burkhart, Heo, and Zavala (2014) extended the Heo and Zavala (2012) research by incorporating input uncertainty using a Monte Carlo expectation maximization (MCEM) framework.

1.3 Prior Work

In my PhD dissertation work, I predicted building energy consumption using GP regression (B. Yan, 2013; B. Yan & Malkawi, 2013) with a 90-day building energy demand and weather dataset. The targets were hourly chilled water use rate (W/m^2) and hourly steam use (W/m^2). The input features included outside air dry-bulb temperature (°C), humidity ratio (kg/kg) and hour of day which is represented by $\sin\left(\frac{2\pi \cdot \text{hour}}{24}\right)$ and $\cos\left(\frac{2\pi \cdot \text{hour}}{24}\right)$. The outcomes consisted of predictive mean and uncertainty range in the form of 95% confidence region, as shown in Figure 1. I also compared the accuracy of GP regression and neural networks. The results of ten-fold cross validations of different predicting period are shown in Figure 2. My PhD thesis also explored GP prediction with uncertain predictive input noise and the use of GP prediction as a baseline in fault detection. However, the work in my PhD thesis was preliminary and the dataset was limited. The study mostly relied on EnergyPlus[3] simulation to generate synthetic data. There was no visualization component involved to help discover underlying patterns of data. No web application tools were developed in my PhD thesis.

[3] EnergyPlus™ is a whole building energy simulation program that engineers, architects, and researchers use to model energy consumption. The models in EnergyPlus are physics-based. More details can be found here: https://energyplus.net/

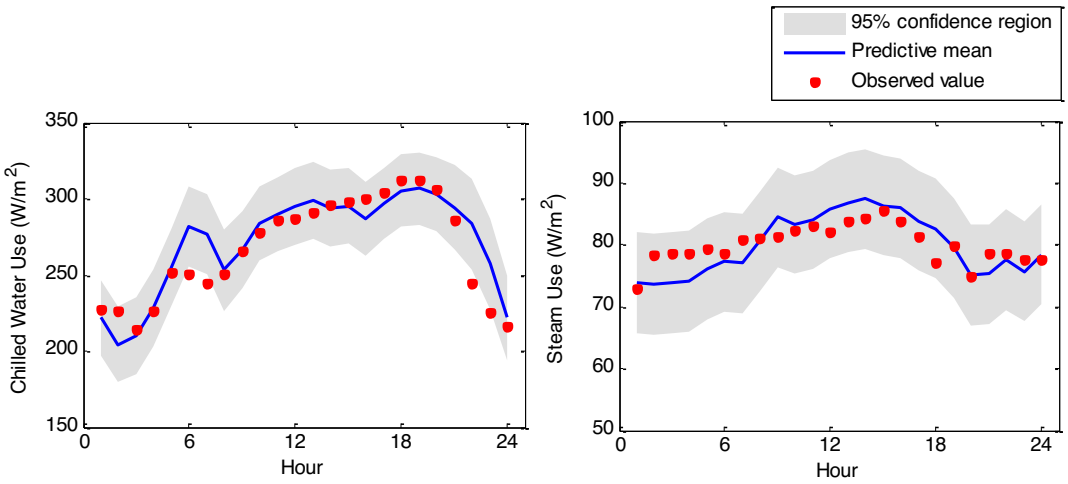

Figure 1 Illustration of 24-hour prediction of chilled water and steam use

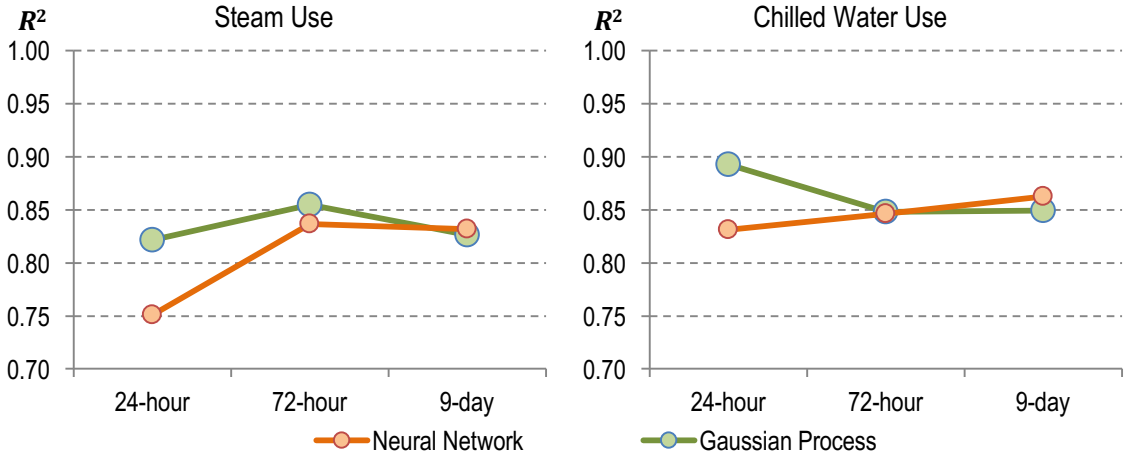

Figure 2 Comparison of R^2 values of Gaussian processes and neural networks

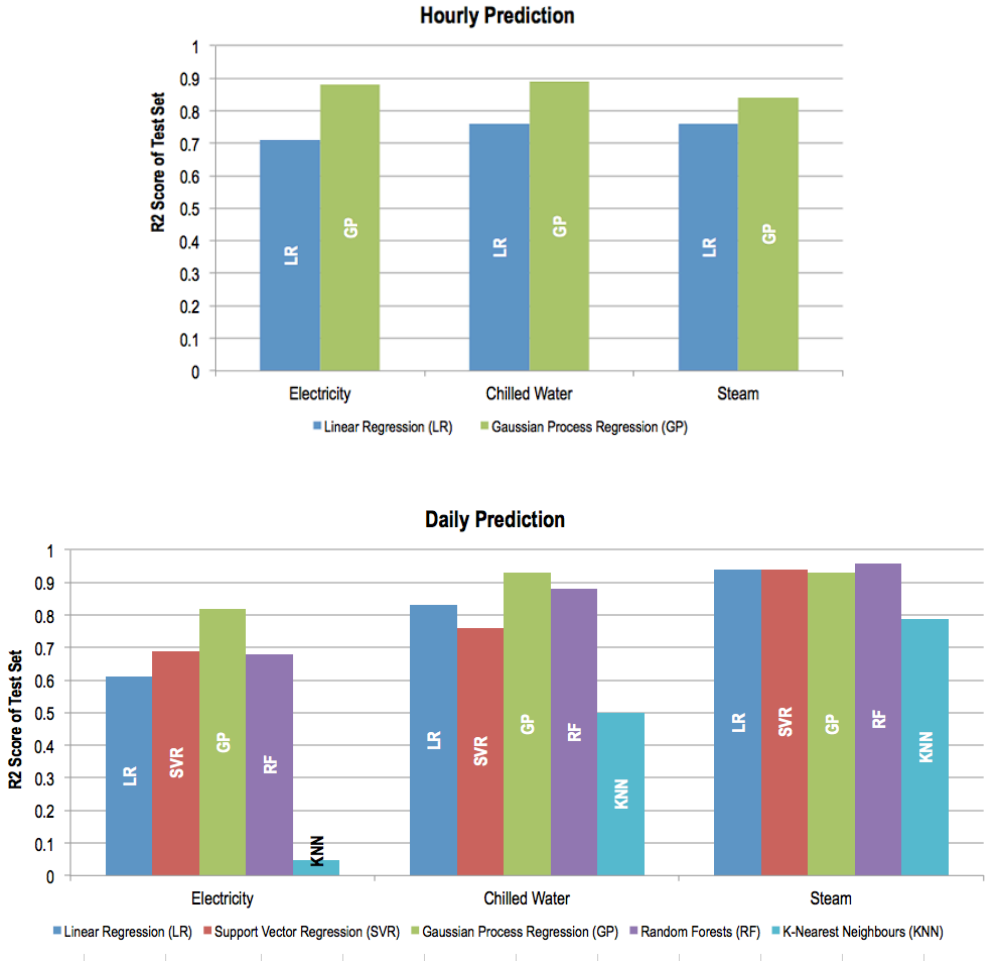

Figure 3 Accuracy comparison of different machine learning methods for hourly and daily prediction

During my course work for this MLA degree, I analyzed the Gund Hall (Harvard Graduate School of Design) data with other two teammates for CS109 (Data Science) final project[4]. Using the energy consumption data from 07/01/2011 to 10/31/2014, we trained different models to predict daily and hourly energy consumption using linear regression, support vector regression, GP regression, random forests and k-nearest

[4] http://cs109-energy.github.io/

neighbours. Figure 3 shows the prediction accuracy of different machine learning methods in this case study. The results of the comparison cannot be readily generalized to other cases without further research. Figures 4 and 5 show some preliminary exploration in visualization. My CS-E81 (Machine Learning) final project explored the prediction of campus-level electric energy consumption using random forests and GP regression. We used data from July 2014 to February 2015, approximately 7 months (one month missing) data as the training set, and March 2015 to May 2015 approximately 3 months data as the testing set. The training R^2 score is 0.95 and the prediction R^2 score is 0.74 ± 0.16 for random forests and 0.77 ± 0.12 for GP regression. We have a website hosted on Github for CS109 final project. After launching the website, I received several inquiries from students and researchers who would like to know more about this project. According to Google Analytics shown in Figure 6, there have been almost 10,000 page views from 10 countries in the past two years.

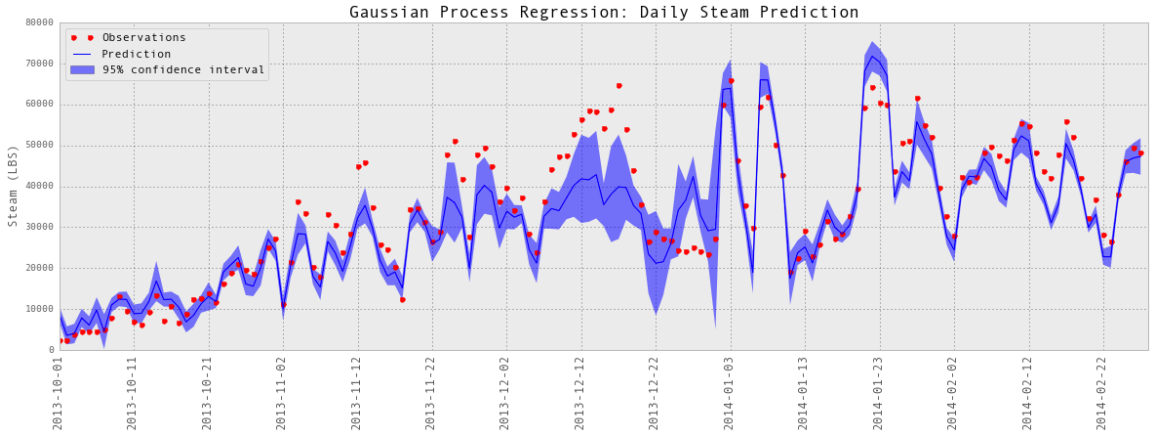

Figure 4 Visualization of Gaussian process prediction implemented in Matplotlib Python module

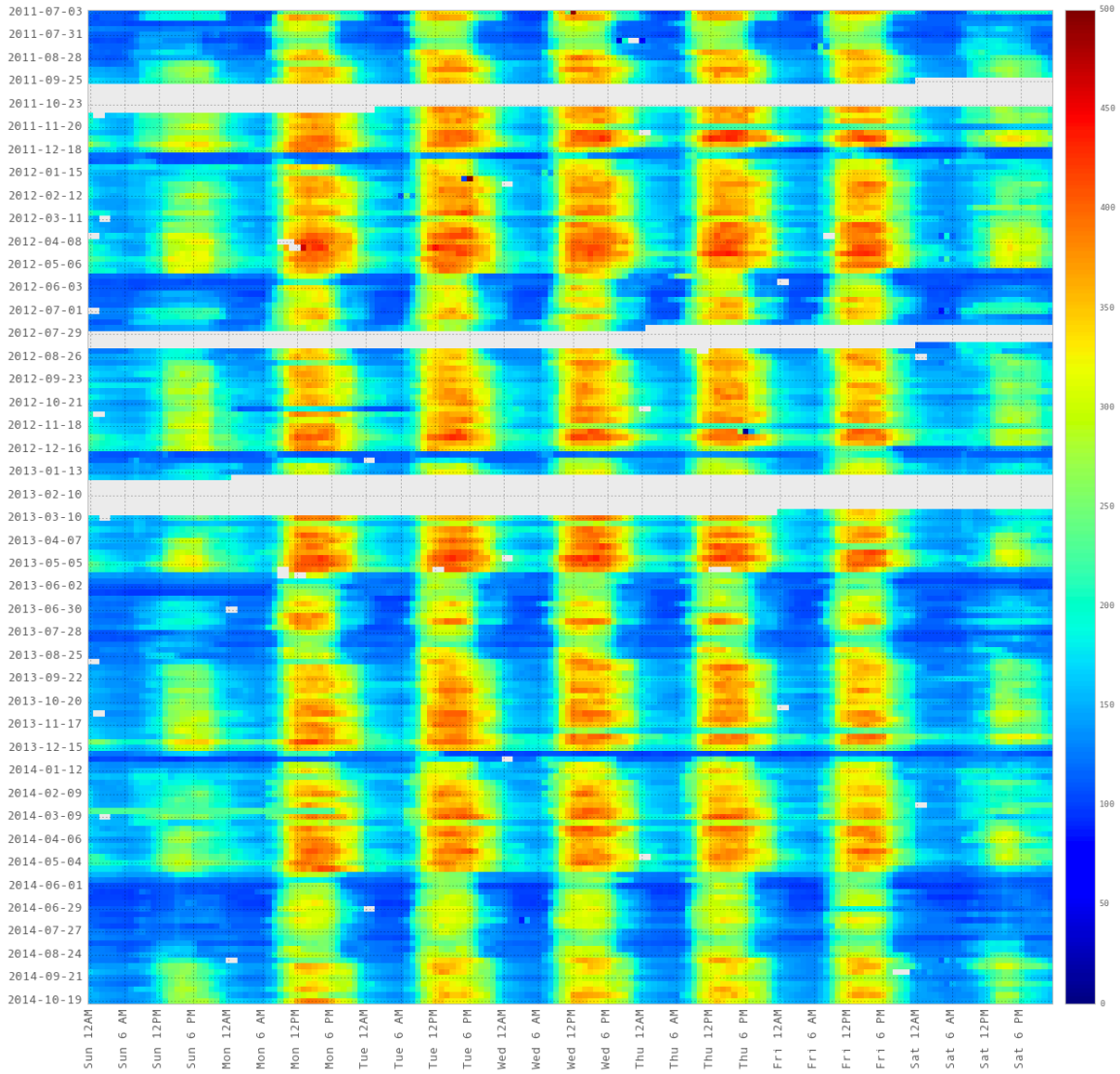

Figure 5 Visualization of electric energy consumption pattern

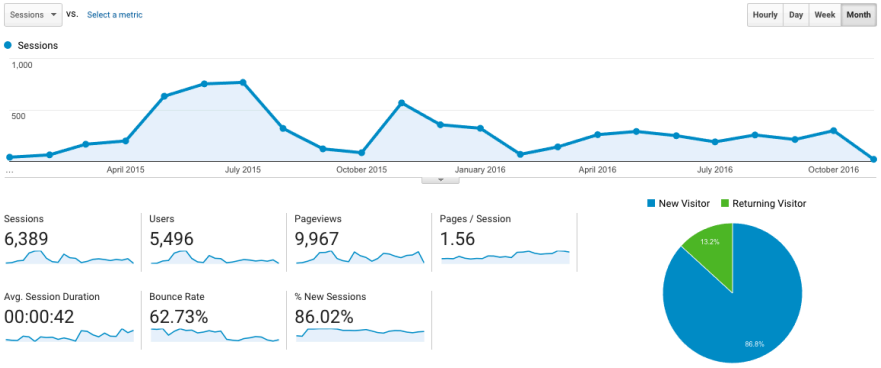

Figure 6 Google analytics statistics of CS109 final project website visits

There are still several limitations in our final projects of CS109 and CS-E81 due to the time constraint. The limitations are in the follow aspects:

1. Algorithm: GP regression could be a useful tool for engineers to forecast building energy demand. The engineers in building or energy industry might not be machine learning experts. Therefore, a guideline on the workflow including how to select features would be helpful. In our CS109 and CS-E81 projects, I didn't perform feature selection but only tried several different feature combinations based on my own domain knowledge. In my PhD and Postdoc projects, I did not investigate the feasibility of adopting time series modeling methods, either.
2. Implementation: For my PhD thesis, I implemented GP regression through a Matlab toolbox developed by Rasmussen and Nickisch (2015). In our CS109 and CS-E81 projects, I switched to a Python module, Scikit-learn[5]. Since the data wrangling is done in Python, it is much easier to use the same programming language for GP regression, but there are downsides. The GP regression in Scikit-learn module is less recognized than the Matlab toolbox developed by Rasmussen and Nickisch (2015), because the work of Rasmussen (2006) is much more frequently cited. The implementation of GP regression in the Scikit-learn module is based on a translation of the DACE Matlab toolbox (Lophaven, Nielsen, & Sondergaard, 2002). Users need to specify the lower and upper bounds of hyperparameters in GP regression, which requires manual tuning and additional cross-validation. I recently found a Python library for GP regression called

[5] http://scikit-learn.org/stable/

GPflow[6]. GPflow uses Google TensorFlow[7] to search for the hyperparameters in GP regression. Its implementation seems to be similar to Rasmussen and Nickisch (2015)'s Matlab toolbox.

3. Visualization: This can help users find underlying patterns in data and thus help with model development. The visualization implemented in Python Matplotlib for CS109 was static. There was no user interaction. The visualization was hard-coded and therefore users will not be able to use that to view their own data such as energy demand patterns and trends unless they change the code.

Inspired by the considerable attention our website of CS 109 project has drawn, I have continued to work on this topic for my MLA thesis. According to my knowledge of the existing research and my experience of two previous projects, there are three areas in which I could make some contributions. First, the existing research about using GP in building energy demand prediction is still very limited compared with that of using other widely used machine learning techniques such as ANN and SVR. A workflow including feature selection is yet to be developed. Second, for users without programming experience, a web application will enable them to develop GP regression models without coding. GP toolboxes in Matlab, Python, R, C and C++ are available (Rasmussen, 2011), but they all require programming to some extent. A user-friendly web application implemented with recent technologies is still lacking. Last, there is no discussion about effective visualization methods that could assist in discovering energy consumption patterns and understanding prediction outcomes.

[6] http://gpflow.readthedocs.io/en/latest/
[7] https://www.tensorflow.org/

1.4 Contribution

The main contribution of this thesis is a web application that allows users to predict and visualize energy demand through Gaussian process regression. Although the web application is designed for energy demand prediction, it can be applied to any time series data with a single output. The prediction types implemented in the web application include baseline prediction[8] and next-day prediction. The web application highlights visualization techniques designed for pattern discovery and model tuning. It also utilizes popular technologies including Tensorflow. This allows the possibilities of integrating other machine learning models, such as a deep neural network into the app in the future.

This thesis advances my prior work by using new technologies to improve the design and the performance of a prediction tool. I also explore the visualization techniques that facilitate the analysis of energy consumption patterns and transform the data into informative insights. In terms of algorithms, the web application adds next-day prediction.

1.5 Outline

Chapter 2 first reviews the mathematical form of Gaussian process regression. Then it discusses how to deal with feature selection and time series predictions in Gaussian process regression. Chapter 3 shift the attention to the web application, which is the core of this thesis. It first presents the overall functions of the web app, then discusses the technology choices. Section 3.3 introduces the implementation details, front-end and back-end design. Chapter 4 illustrates the use of the web application through two case

[8] See Section 2.4 for the definition of baseline prediction.

studies. Finally, Chapter 5 concludes this thesis with a discussion on interesting future directions for the web application as well as the algorithm development.

Chapter II.

Algorithm

This chapter reviews the basic mathematical form of Gaussian process regression and discusses how to deal with feature selection and time series predictions. Section 2.1 describes the GP regression based on existing studies. Section 2.2 is from a publication of my Postdoc work (B. Yan, Li, Shi, Zhang, & Malkawi, 2017) and therefore is not part of the work of this thesis. It is here because the equations and process in Section 2.1 and 2.2 are used in the web application. Section 2.3 introduces the method of time series prediction used in this thesis and Section 2.4 discusses the difference of two types of predictions in the web application.

2.1 Standard Gaussian Process Regression

The Gaussian process regression models used in this thesis are based on the work of Rasmussen & Williams (2006). A Gaussian process is specified by a mean function and a covariance function $k(\mathbf{x}_i, \mathbf{x}_j)$. The choice of mean function in this study is the zero function, and the covariance function is a squared exponential kernel,

$$k(\mathbf{x}_i, \mathbf{x}_j) = \sigma_f^2 \exp\left[-\frac{1}{2}\left(\mathbf{x}_i - \mathbf{x}_j\right)^T W^{-1}\left(\mathbf{x}_i - \mathbf{x}_j\right)\right] \quad (1)$$

where

$$W = \mathrm{diag}[w_1^2, w_2^2, \ldots, w_D^2] \quad (2)$$

with parameter w defining the characteristic length-scale in covariance function. D in Equation 2 is the number of features. The characteristic length-scales roughly define how far apart the input values $x_{i,d}$ and $x_{j,d}$ can be for the response values to become uncorrelated, where $x_{i,d}$ is the value of the dth feature of the ith data point . Inputs that are judged to be close by the covariance function are likely to have similar outputs. A prediction is made by considering the covariance between the predictive case and all the training cases $\mathbf{k}(X, \mathbf{x}^*)$ (Rasmussen, 1996). For a noise-free input $\mathbf{x}^*$, predictive posterior mean and variance are (Rasmussen & Williams, 2006)

$$\mathbb{E}[f_* \mid X, \mathbf{y}, \mathbf{x}^*] = \mathbf{k}(X, \mathbf{x}^*)^T (K + \sigma_n^2 I)^{-1} \mathbf{y} \tag{3}$$

$$\mathbb{V}[f_* \mid X, \mathbf{y}, \mathbf{x}^*] = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}(X, \mathbf{x}^*)^T (K + \sigma_n^2 I)^{-1} \mathbf{k}(X, \mathbf{x}^*) \tag{4}$$

K is an $N \times N$ matrix of covariance functions between each pair of training inputs. σ_n^2 denotes the variance of Gaussian noise in training targets $\mathbf{y}$, I is an $N \times N$ identity matrix. σ_f, σ_n and $w_1, w_2 \ldots w_D$ are hyperparameters to be trained through gradient descent optimization algorithm. Figure 7 describes the relationships between inputs and output, as well as the ones between training points and the points to be predicted in a GP regression.

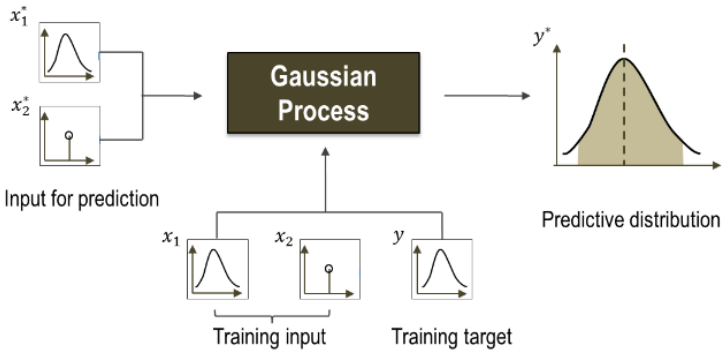

Figure 7 Diagram of Gaussian process regression

A distinguished feature of Gaussian Processes is that the outcomes come in the form of probability distributions, which take uncertainty in the modeling process into account. The variance of a prediction (Equation 4) automatically includes two types of uncertainties. The first type of uncertainty is noise in the targets $\mathbf{y}$. The noise might come from measurement noise. The process to be modeled itself is stochastic, and thus there are random elements in $\mathbf{y}$. The features in an existing model might not fully explain the variance in training targets. There might be some other important features that affect outputs. The second type of uncertainty is interpolation uncertainty. The distance between the inputs associated with a prediction and training inputs affects the magnitude of the variance. GP modeling is an interpolation method. If a new input point lies beyond the scope of the training input domain, the variance will be large in the prediction.

2.2 Feature Selection

Successful feature selection will improve prediction performance, reduce computational cost and provide a better understanding of the underlying prediction process. Feature selection is especially helpful for high-dimensional problems such as text processing of internet documents and gene expression array analysis (Guyon & Elisseeff, 2003). Although energy prediction is not a high-dimensional problem, if there are more than a few thousand training points, the covariance matrix gets large and computational cost could be high. For instance, hourly prediction using one-year historical data requires feature selection in order to reduce computational cost.

There are feature selection methods based on entropy (Che & Wang, 2014; Zheng & Kwoh, 2011). Following the Che and Wang (2014) method, we have tested feature

selection based on correlation coefficients and mutual information for GP regression. However, these algorithm-independent feature selection methods might not be applicable to GP regression or energy demand forecasting discussed in this thesis. Here, we use an alternative approach of feature selection specifically targeting GP regression, which has been discussed in (B. Yan et al., 2017). The idea is to use the hyperparameters which a GP regression learns. Here are the implementation steps.

1. Normalize all features and the target to [-1, 1].
2. Divide the dataset into training, validation and test sets. Use training and validation sets for feature selection. If the dataset is large and computational cost is a concern, select smaller training and validation sets for the purpose of feature selection.
3. Include all the features to train a GP regression and derive the characteristic length-scale w that corresponds to each feature from the training set.
4. Calculate the indicator of each feature as

$$g(\mathbf{x}_d) = \frac{w_d}{\mathrm{std}(\mathbf{x}_d)} \tag{5}$$

where $\mathbf{x}_d$ is the input vector of a certain feature, $d = 1,2, \ldots D$ and D is the number of features. The indicator $g(\mathbf{x}_d)$ is defined as the characteristic length-scale of a feature divided by the standard deviation of all inputs on that feature dimension. As mentioned in the Modeling Methods section, the characteristic length-scale determines how close two points have to be to influence each other. As the characteristic length-scale is normalized by the variation of the training inputs on that feature dimension, it indicates how significant that feature is. If

$g(\mathbf{x}_d)$ is small, it means small changes in input value of a certain feature will have a significant impact on output value. Therefore, a relatively small $g(\mathbf{x}_d)$ indicates that the corresponding feature is important in GP regression.

5. Select $\mathbf{x}_{(1)}, \mathbf{x}_{(2)}, \ldots \mathbf{x}_{(D^*)}$ as

$$\mathbf{x}_{(1)} = \arg\min_{X}[g(\mathbf{x}_d)] \tag{6}$$

$$\mathbf{x}_{(2)} = \arg\min_{X-\{\mathbf{x}_{(1)}\}}[g(\mathbf{x}_d)]$$

$$\vdots \tag{7}$$

$$\mathbf{x}_{(D^*)} = \arg\min_{X-\{\mathbf{x}_{(1)},\mathbf{x}_{(2)},\ldots\mathbf{x}_{(D^*-1)},\}}[g(\mathbf{x}_i)]$$

where (1), (2) and (D^*) denote the rank of features, and X denotes the entire feature sets. The feature with the smallest $g(\mathbf{x}_d)$ is the first feature to select. The number of the features D^* depends on training and validation accuracies, as well as computational cost. In general, if the training and/or validation accuracy of using the selected feature(s) is much lower than the accuracy of using all features, one should include more features in the model. Otherwise, test with fewer features to further reduce computational time.

2.3 Time Series Prediction

Energy demand forecasting appears to be a typical time series modeling problem. Each data point is associated with a timestamp and energy demand at each time step is strongly related to the previous one. For buildings, the energy consumption for heating and cooling at different time steps is not independent due to thermal inertia of buildings.

Such dependencies due to thermal inertia may be more obvious when time intervals are shorter than an hour.

In order to use GP regression for time series prediction, we can continue to use a modification of Equations (1) to (4). To use GP regression for time series prediction adds a few time series features and uses a modified training matrix. This study examines the effect of adding a few time series features and growing training matrix for time series prediction. The time series features tested in this study include:

$\boldsymbol{i}$: the order of data points. By using $\boldsymbol{i}$, more recent training data points get higher weights. The prediction will be more influenced by recent data points.

$\boldsymbol{y_{t-1}}$: measured value of the target at the previous time step. When predicting next-day energy demand, we will use yesterday's measured energy use as a feature.

$\boldsymbol{f_{t-1}}$: predicted value of the target at the previous time step. When performing X-day ahead ($X \geq 2$) predictions, measured $\boldsymbol{y_{t-1}}$ is not available. Predicted value $\boldsymbol{f_{t-1}}$ is used instead.

Most learning tasks use a fixed set of historical data as the training set. In order to predict the trend, it makes more sense to include recent data points when they are available. We use a fixed set of historical data to train a model only once. The hyperparameters in the trained model stay the same, but new data will be added to the training data matrix X in Equations (3) and (4) when available. For example, in order to predict the next day energy demand, today's observed energy consumption along with the observed value of the features will be added to the training matrix.

2.4 Forecasting Type

Whether to add time series features and/or to use a growing training set depends on the type of prediction task. As far as the energy demand prediction is concerned, there are two types of prediction tasks, ***baseline*** prediction and ***trend*** prediction.

In baseline prediction, we use a dataset of one to several years to predict energy demand for another long period of time, e.g. 1-2 years. All the features X and targets Y are observed. The predicted value serves a baseline, in that if all the inputs remain the same, the output will be the predicted energy. Baseline prediction is useful for anomaly detection. It does not make sense to add time series features or use a growing training set for the purpose of baseline prediction. Otherwise, you are not comparing the energy consumption with a fixed period of time.

Predicting the energy demand of next-day, next-30-day or further times in advance is a trend prediction task. When predicting tomorrow's energy demand, some of tomorrow's features X are estimated or predicted, for example, weather. The purpose is to predict future energy demand and make plans no matter whether the energy consumption pattern stays the same. When predicting energy demand of next-30-day energy, a long-term daily weather forecast is usually unavailable. Therefore, in practice, weather might not be included in the feature set. The only features available are time-related features and historical energy use. Adding time series features and using a growing training set might help improve trend prediction accuracy even when energy use patterns are quite different from previous time periods.

Chapter III.

Web Application

3.1 Application Overview

The main purpose of the web application is to allow users without programming skills to train Gaussian process regression models to predict energy demand. First, users need to upload their data in a csv file. The data file needs to follow the format as shown in Figure 8. The first row of the csv file must be the names of the features. The first column must be the date. The program uses the Python **`dateutil`** module to parse date/time strings. This module can return a date/time object in Python for various common date/time input formats. The settings of the parser in the web application interprets the first value in an ambiguous 3-integer date (e.g. 01/05/09) as the month and the last value as the year. All the data of features and target need to be numeric. Once the file is uploaded, the variable names and their values will be read into the database and fed into visualization pages. Then users will be able to view the data, select forecasting type, features and training period, as well as view prediction results through interactive visualizations. After users submit all the model inputs, the back-end runs the GP regression algorithm, trains a model using the features and target specified by users, stores the results in the database and outputs the predictive mean, 95% confidence region, as well as training and test accuracies. The back-end also calculates the importance of each selected feature and presents the information in a bar chart in the front-end. If the prediction accuracy is low, users can use interactive visualization to check the data points with poor predictions. After some diagnosis, users can re-train the model by trying different features and training periods. The current version of the web application

implements the baseline and next-day prediction. The next two sections introduce the technology choices, back-end implementation and front-end design in detail.

date	cooling degrees	solar radiation	dehumidification	occupancy	chilledWater-TonDays
1/1/12	0	95.26086957	0	0	0.961857377
1/2/12	0	87.33333333	0	0.3	0.981724914
1/3/12	0	95.70833333	0	0.3	1.003671785
1/4/12	0	98.75	0	0.3	1.483192383
1/5/12	0	90.75	0	0.3	3.465090988
1/6/12	0	76.54166667	0	0.3	2.850017304
1/7/12	0.375	94.29166667	0	0.3	1.233654041
1/8/12	0	95.5	0	0.3	1.819965162
1/9/12	0	101.5	0	0.3	1.755790202
1/10/12	0	95.79166667	0	0.3	2.031340777
1/11/12	0	98.70833333	0	0.3	2.040937011
1/12/12	0	5.083333333	0	0.3	1.598126892
1/13/12	0	45.75	0	0.3	1.734152099
1/14/12	0	88.75	0	0.3	2.102609237
1/15/12	0	109.7083333	0	0.3	1.956226933
1/16/12	0	104.2916667	0	0.3	2.022602633
1/17/12	0	59.125	0	0.3	1.505160495
1/18/12	0	109.4166667	0	0.3	1.063769172
1/19/12	0	94.79166667	0	0.3	1.417285663
1/20/12	0	117.0833333	0	0.3	1.239107508
1/21/12	0	21.70833333	0	0.3	0.758384332
1/22/12	0	83.33333333	0	0.3	0.794660121
1/23/12	0	44.25	0	0.97083333	1.326296195
1/24/12	0.166666667	91.95833333	0	1	2.118289861
1/25/12	0	114.7916667	0	1	2.094743663
1/26/12	0	85.125	0	1	1.758939562
1/27/12	0	9.541666667	0	1	2.01469655

Figure 8 csv data file format

3.2 Technology Choices

3.2.1 Python and GPflow

Python is used to program the back-end, including algorithm implementation and web service. Python is widely used in the machine learning community. For example, TensorFlow, a popular machine learning library, uses Python. Using Python will allow

the web application to incorporate other machine learning models in the future. The GPflow package is used to implement the GP regression algorithm (Matthews et al., 2017). This package uses TensorFlow to handle all gradient computation. GPflow comes with many covariance functions (also called kernels), such as constant, linear, white, Matern12/32/52, RBF (radial basis function, the one used in this project), cosine and periodic kernels. Kernels can be combined and different kernels can be used on different feature dimensions.

3.2.2 Flask

In order to build a python web application, one needs a web framework to implement all the functionality common to most web application, such as mapping URLs to Python. There are many Python web frameworks; the most popular two are Django and Flask. Django was first released in 2006 and now it has the largest community with over 156k StackOverflow questions as of November 2017. Flask is a much younger framework which started in mid-2010. There are around 19k StackOverflow questions, which is only the one-eighth of Django. On Github, they have a nearly identical number of starts with 29.4k for Django and 30.8k for Flask.

I choose Flask as the web framework for this project since it is great for small projects that need a fast way to make simple web sites. Compared with Django, Flask is more lightweight and it is easier to figure out for new web developers. In addition, Flask lets you decide what components, such as database, to use and how to interact with them.

3.2.3 MongoDB

I choose MongoDB as the database for this project. MongoDB is a NoSQL database. NoSQL databases were developed in late 2000s to deal the limitations of SQL databases. NoSQL database allows the insertion of data without a predefined schema. Compared with a SQL database, it is easier to make significant application changes in real-time to a NoSQL database, which makes development faster. They support auto-sharding, which means they can natively and automatically spread data across an arbitrary number of servers as necessary (MongoDB, 2016).

The debate about whether NoSQL is superior to SQL database is not over. Today SQL seems to be resurging. Google's globally-distributed data management system Spanner transitioned from NoSQL to SQL because "SQL has provided significant additional value in expressing more complex data access patterns and pushing computation to the data" (Bacon et al., 2017). The main reason I decided to use MongoDB is that it makes web application development easier for a small project. MongoDB uses JSON-like documents, which is consistent with the data structure used in web front-end development. A most appealing feature is that by using MongoDB, applications can add new fields to each data document on the fly. Unlike SQL table rows, dissimilar data can be stored together. MongoDB's flexibility is extremely useful for web application projects.

3.2.4 Highcharts.js

The market of charting tools has become highly competitive. I selected Highcharts.js as the main tool to create interactive charts in front-end. Highcharts.js is a charting library written in pure JavaScript that was created by a Norway-based company

Highsoft. Highcharts.js provides a variety of basic charts, line, area, column, bar, pie, scatter and bubble charts. Using the combinations of the basic chart elements, it can also generate gauges, heat and tree maps, polar charts, wind roses, vector plots, Sankey diagrams and word clouds. Highstock.js, a side product of Highcharts.js, optimizes the line charts for financial data visualization and large datasets. The timeline charts in Highstock.js, as shown in Figure 9, have sophisticated navigation options including preset date ranges, date picker, scrolling and panning, which is ideal to visualize time series data such as energy demand in this project.

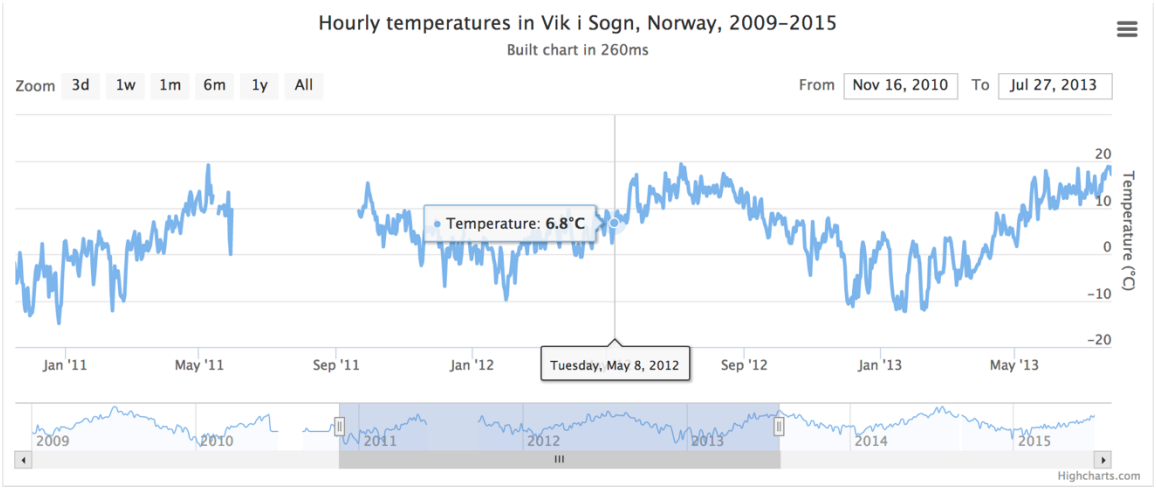

Figure 9 Time series chart in Highcharts.js

Since its first release in 2009, Highcharts.js has been developed to meet evolving needs. The chart tools provided by Highcharts are responsive across devices, touch optimized for mobile and screen devices, and ready for big data through their WebGL-powered Boost module. All the charts are easy to customize and style via JavaScript or CSS. Last but not least, Highcahrts is free for non-commercial use.

In addition to Highcharts.js, D3.js and Tableau are among the most popular tools that produce dynamic and interactive data visualizations. The style of the charts created byD3.js is similar to Highcharts.js, but D3.js provides much more variety and flexibility. When I show my D3 visualizations to people working in machine learning or big data industries, they always ask me, “why not Tableau?” To be honest, I have never used Tableau. I have only seen that one of my co-coworkers without programing skills can create an interactive chart quite easily. The same chart she created using Tableau will take me probably 30~40 hours to develop using D3.js. I have to admit that Tableau is easy to learn and easy to use. One can simply create an interactive chart and publish it on web by drag and drop. No coding is required. It might take many hours and heavy coding to create the same chart in D3. However, Tableau is a closed software package which means when you are developing your own web apps, it would be difficult to connect charts with back-end. Not to mention that Tableau license is very expensive. D3.js, in comparison, is a JavaScript library which uses SVG, HTML5 and CSS standards. Without any doubts, D3.js is made for data visualization in web browsers and suitable for web apps development. D3.js is free and open. It can satisfy high interactivity and customization requisites.

Since programming is not an issue for me, I prefer tools like D3.js which are open source and provide great flexibility. The drawback is that even a standard chart might require moderate coding. Although one can copy and paste most of the code from online resources, the code will become long and messy. For this thesis project, if I had used D3.js, I would have found it difficult to maintain all my HTML and JavaScript code, and probably would have soon lost track of them. I like the idea and style of D3.js, but the

amount of work required has prohibited me from using it for this project. To make coding easier, there are libraries built on top of D3.js, for example C3.js, which could make your code shorter and save you some time. However, the chart design is not elegant. I might end up spending extra time trying to change chart appearance so it is no use.

Plotly is another visualization tool built on top of D3.js. Plotly offers several products which include (1) Plot.ly, a graphical user interface which adopts drag and drop style; (2) Plotly.js, an open source JavaScript library; (3) API libraries for Python, R, MATLAB, Node.js and REST API. Coding is manageable using Plotly.js, but it suffers the same problem – the style in default charts is not optimal, at least in my opinion.

I chose Highcharts.js over C3.js or Plotly.js. I like the design of its charts, especially the design of time series charts as shown in Figure 9. The navigation options in Highcharts.js are more user-friendly than C3.js and Plotly.js. The default time series chart of Highcharts.js provides buttons to select pre-configured ranges in the chart, like 1 week, 1 month, etc. It also provides input boxes where min and max dates can be manually input. A small series below the main series displays a view of the entire data set. It allows users to zoom in and out on parts of the data as well as pan across the dataset. In comparison, the default time series chart of C3.js does not provide any navigation options, as shown in Figure 10. The default time series chart of Plotly.js shown in Figure 11 provides pre-configured date ranges selector and a range slider similar to Highcharts.js. However, the choices of default fonts, colors, line styles and chart layouts are not as good as Highcharts.js.

Compared with D3.js, coding with Highcharts.js is more manageable. Although Highcharts.js is not built on top of D3.js, the style is similar. In this project, I used D3.js together with Highcarts.js when I needed more customization than Highchart.js provides.

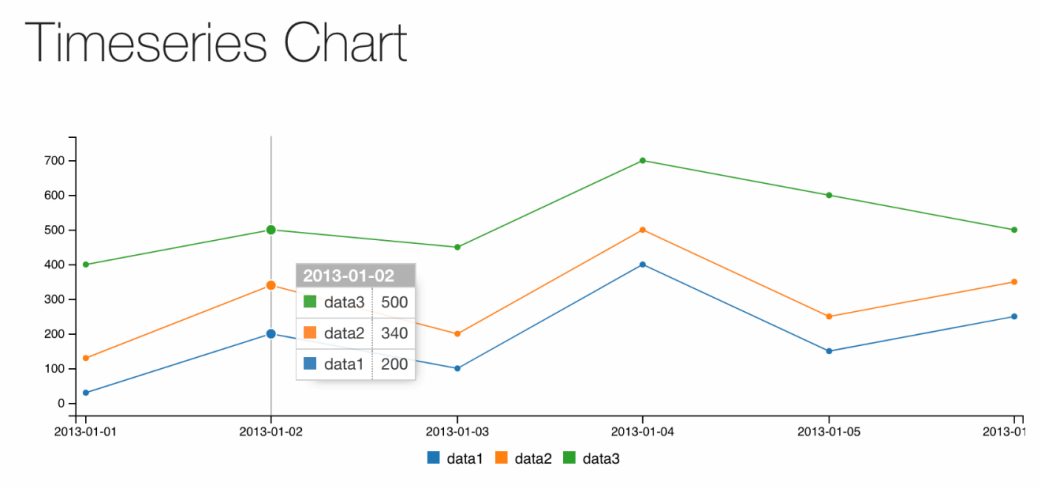

Figure 10 Time series chart in C3.js

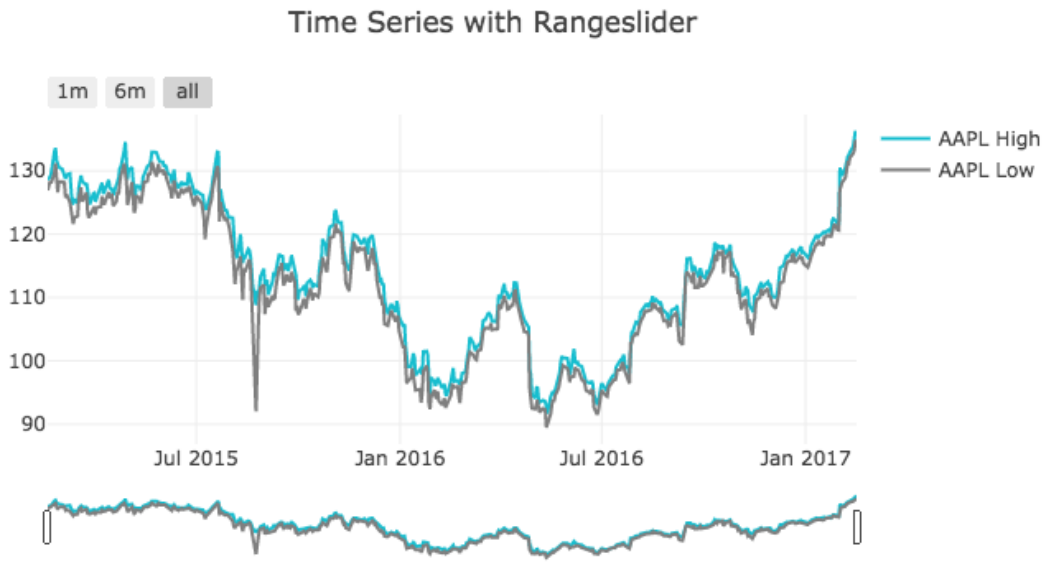

Figure 11 Time series chart in Plotly.js

No single visualization tool could be a perfect solution. The choice depends on many factors including cost and time for development, expertise available or familiarity to the tools, dataset volume and features, level of interactivity required and visual appeal taste. The visual design of the timeline chart in Highcarts.js is a good fit to visualize energy demand data and open source libraries sometimes are ideal.

3.2.5 Heroku

Heroku was selected as the cloud platform as a service (PaaS) solution to deploy the web application. Platforms as Service (PaaS) remove the need for organizations to manage the underlying infrastructure. This helps developers be more efficient as they do not need to deal with resource procurement, capacity planning, software maintenance, patching, or any of the other undifferentiated heavy lifting involved in running applications (Amazon.com, 2017). Heroku, one of the first cloud platforms, was established in 2007. Other PaaS choices include Google App Engine and Amazon Web Service. Among these PaaS solutions, Heroku probably offers developer the simplest path to deploying the web applications online. It can take as short as five minutes and a few lines of commands to deploy an app. Heroku provides a free plan ideal for experimenting with cloud applications, which allows 512 MB RAM and a monthly pool of 1000 free dyno (Heroku's unit of processing power) hours. This study used this free plan for initial web development. If the complexity of this web app evolves in the future, 512 MB RAM might not be sufficient.

3.2.6 Summary

Figure 12 summarizes the technology choices used in the web app.

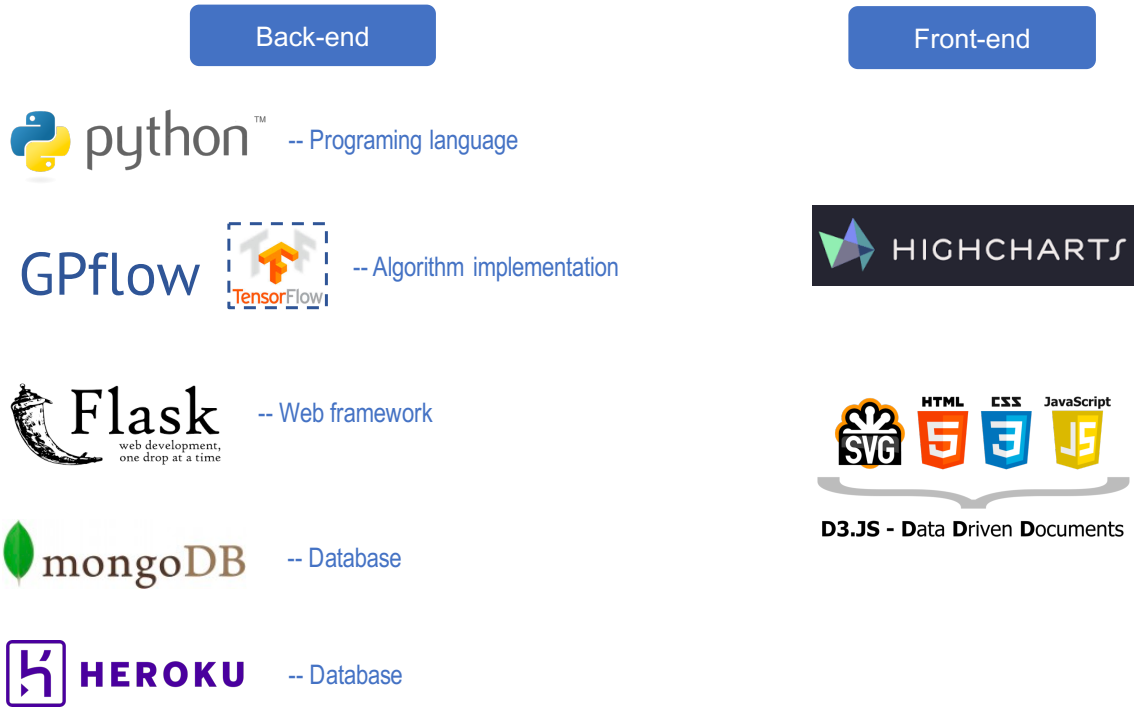

Figure 12 Technologies used in the web app

3.3 Back-end Implementation

The back-end does three things: (1) uploading data files and storing data in the database; (2) querying database and formatting data for RESTful (Representational state transfer) web service; (3) implementing algorithms including feature selection, training and prediction, and storing results in the database. This section first introduces the database design and then discusses the main functions in Flask.

3.3.1 Database

In MongoDB, databases hold collections of documents (MongoDB, 2017a). Everything is stored in one database in this web application. Each dataset uploaded by

users is stored in a collection. A collection contains N documents, where N is the number of data points. Each document holds one row of data in the csv file shown in Figure 8. One dataset only has one GP regression model. The information of all the models is stored in a separate collection, which contains M documents, where M is the number of models/datasets. Therefore, the database contains $M + 1$ collections in total.

When a user uploads a new data file, the database creates a new collection and assigns a name of 6 random letters to the collection. The program checks whether the newly created collection name already exists. In that case, the program generates a new name and checks for uniqueness again until a unique name is generated and assigned to the collection. This 6-letter name is called model ID. The model ID is shown at the end of the link addresses of the training and result pages. Then the database stores each data entry as a BSON document in the collection. A MongoDB document is composed of field-and-value pairs. Its structure is the same as a JSON representation or dictionary in Python. The value of a field can be any of the BSON data types, including other documents, arrays, and arrays of documents (MongoDB, 2017b). In this web application, the value of a field is a date, a numeric value or a Boolean value. For each document, MongoDB assigns an id. This id is not explicitly used in the web application though. "Date" is a required field in the uploaded data file of this web app. As soon as a data file is uploaded, the back-end converts the date to several time features including timestamp in millisecond, year, month, week of year, day of month, and day of week. Timestamp is used as an index, since we need to look up a data entry by its timestamp frequently. This speeds up the query process considerably. Timestamp is also used for x-axis values in time series charts in Hicharts.js. Year and month are used in calendar chart display. Week

of year, day of month, day of week might be useful features in GP model training. The values of features and target are also stored in each document. After a GP regression is trained, predictions are made for both training and test sets. The predicted mean value along with the prediction uncertainty are inserted into each document. If the observed value of the target falls out of the 95% confidence region of prediction, then this point will be marked as “abnormal” in the database. If a model is re-trained, the predictions will be updated. The structure of a collection that stores uploaded data and corresponding processed data is illustrated in Figure 13.

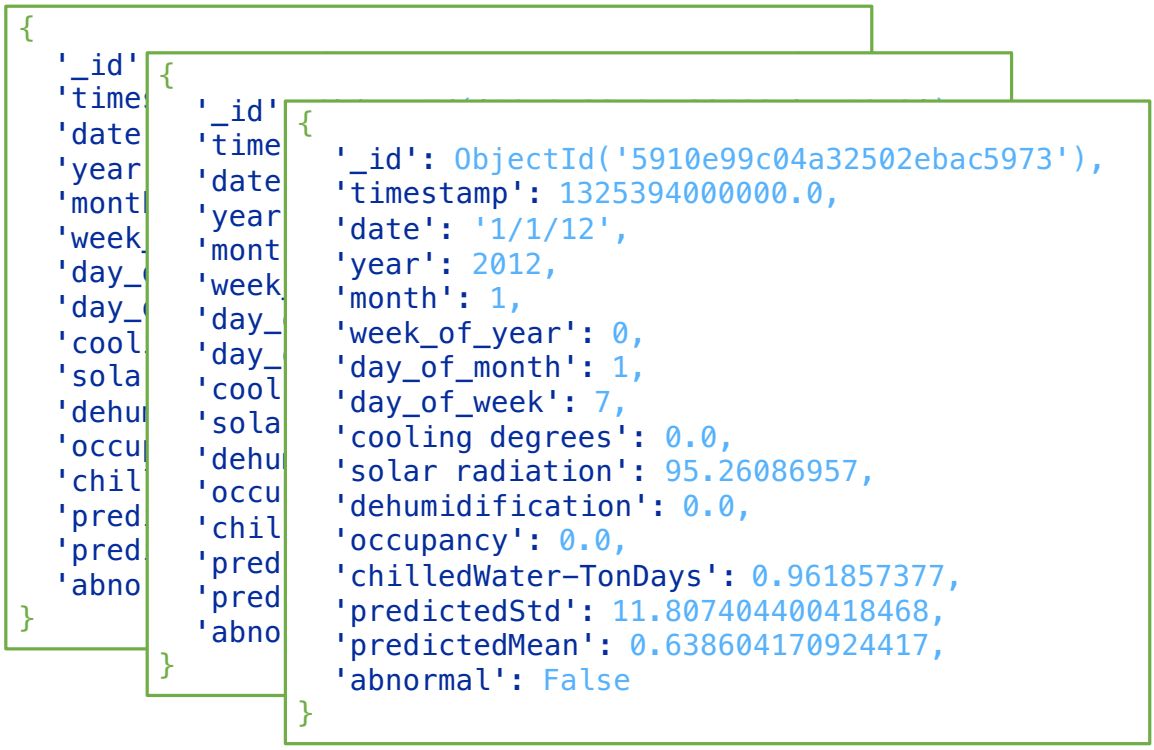

Figure 13 Structure of a collection that stores data uploaded by a user and corresponding processed data

A collection called “models” holds documents of model information including the model ID assigned to a data file, the split of training and test sets, training and test R^2 values (called accuracy in the database), as well as the features and target selected by users. In MongoDB, a collection does not require its documents to have the same schema. The documents in a single collection do not need to have the same set of fields (MongoDB, 2017a). Therefore, the same collection can hold all the data for the model. However, in order to make queries easier, a separate collection was created to hold the information of all models, as shown in Figure 14.

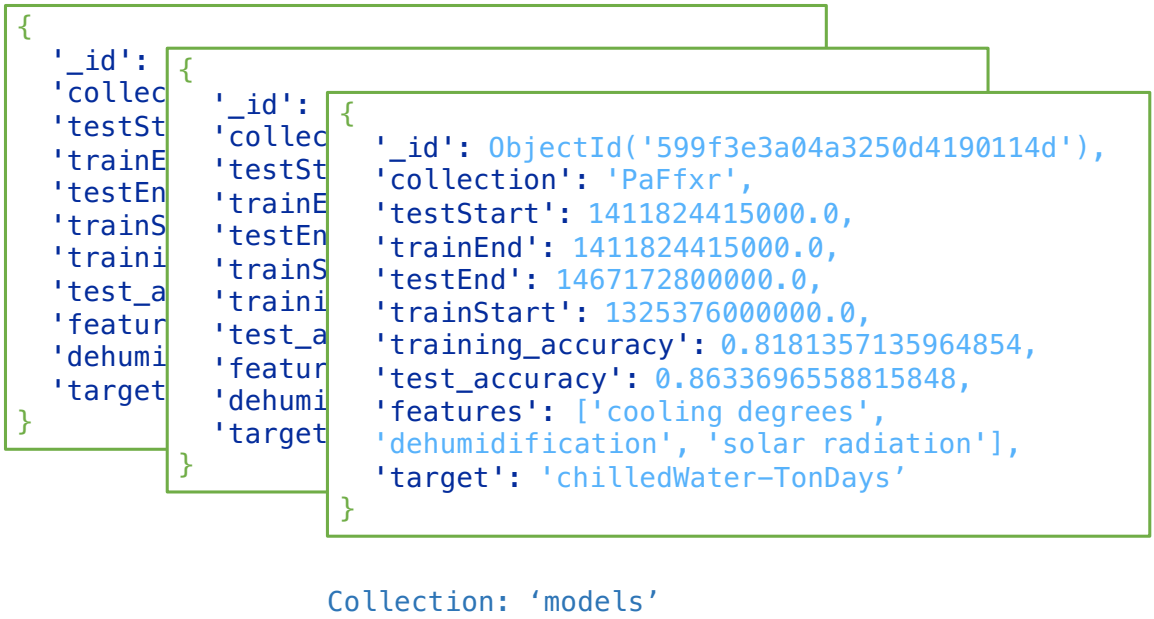

Figure 14 Structure of the collection “models”

3.3.2 Functions

The back-end consists of four types of functions: (1) processing uploaded files; (2) querying database; (3) formatting data; (4) implementing algorithms. These four types of functions are listed in Table 1 and will be described in detail in the following paragraphs.

Table 1 List of functions and their parameters in back-end

Category	Name	Parameters
Process uploaded files	**upload_file**	
Query database	**get_keys**	collection, train_only=False
	get_data	collection, selected_fields=**None**, excluded_fields=**None**
	get_range	collection, field
	get_model	collection
Format data	**format_data**	type_, collection
Implement algorithms	**gp_train**	X_train, Y_train
	gp_predict	X_test, trained_model
	gp_predict_nextDay	X_train, Y_train, X_test, Y_test, trained_model
	gp_get_feature_rank	trained_model, features, X_train
	train_and_predict	collection

The function **`upload_file`** allows users to upload a csv data file or retrieve a previous model. This function first checks whether a user intends to retrieve a model previously uploaded from the database or to upload a data file. In the former case, it will fetch the data and redirect the page to visualize the model if the input model ID exists in the database. Otherwise, it will ask the user to either re-enter a model ID or upload a data file. If a user uploads a csv file, the function will assign a model ID, read the data, parse the date inputs, convert dates to millisecond timestamp, extract time-related features, store all the data including processed timestamp and time-related features into the database and redirect the page to visualize the data. Figure 15 shows the processes of the function **`upload_file`**.

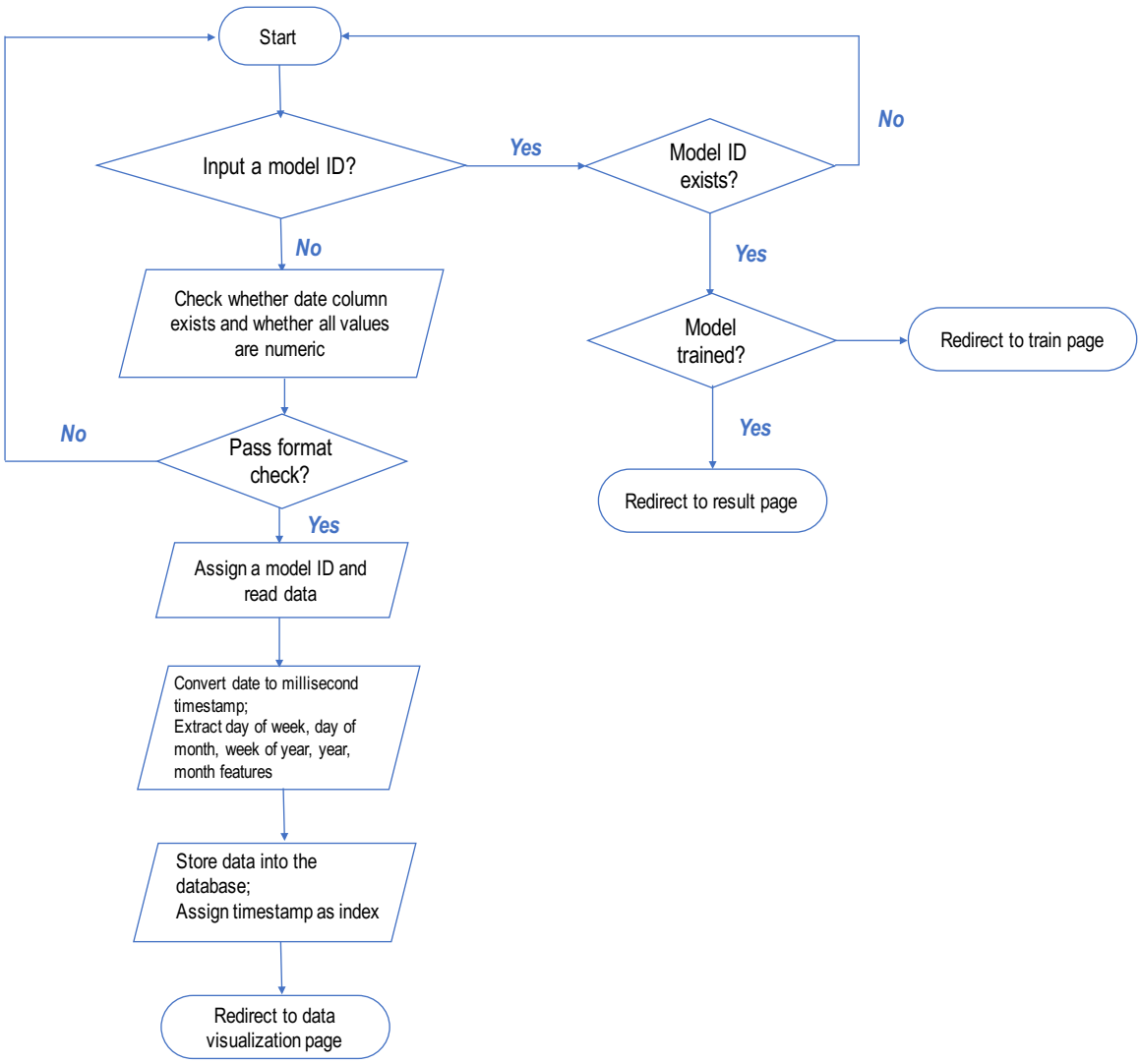

Figure 15 Flowchart of the function `upload_file`

A series of functions query the database for training a model in the back-end and displaying data in the front-end. The function **`get_keys`** gets all the variable names of a dataset, which are used as "fields" of a data record in the MongoDB database. The function **`get_data`** fetches all the data from a model. MongoDB stores data records in the format of BSON documents in "collections". In MongoDB, databases hold collections of documents. Each dataset is stored as a collection in this web application.

By default, this function fetches all the data from a collection in a database. Users can select which features to include or exclude. The function **`get_range`** returns the minimum and maximum data points of the field specified by a user in a collection. The function **`get_model`** returns a JSON structured data of the detailed information of a model including the names of features, target, the start and end dates of the training set and test set, training accuracy, test accuracy and train-to-test data size ratio. The information is displayed in the front-end.

The data returned by the function **`get_data`** is used only in the back-end. The function **`format_data`** converts data to the formats that can be visualized for different kind of charts in the front-end. For each chart, the function **`format_data`** specifies what to display on the x and y axes, the marker type, size and color and returns the data in JSON. The function **`gp_train`** trains a Gaussian regression model using GPflow. The function **`gp_predict`** returns the predicted mean value and standard deviation of given X for baseline prediction. The function **`gp_predict_nextDay`** returns the prediction results for next-day prediction. The function **`gp_get_feature_rank`** returns the indicator of feature importance according to Equation (5) to (7). This information represents the sensitivity of each feature in the GP regression and helps users to select features. The function **`train_and_predict`** processes user inputs from the front-end, calls the function **`gp_train`**, **`gp_predict`**/**`gp_predict_nextDay`** and **`gp_get_feature_rank`**, as well as stores all the results into the database.

Figure 16 illustrates the main interactions between uses, front-end and back-end. Representational state transfer (RESTful) web services are used for communication between client (front-end) and server (back-end). The client obtains formatted data and

results of trained models from the server through GET HTTP methods. The client also sends information such as model ID, selected features and target, as well as specified training and test period to server side through POST method.

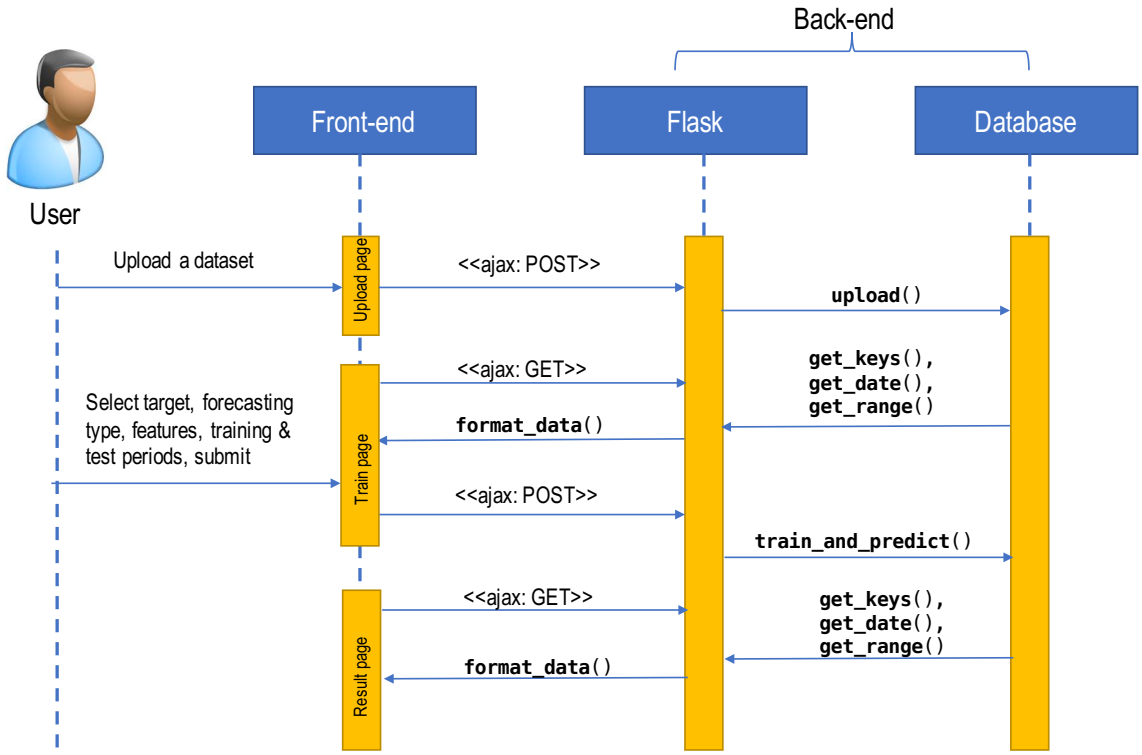

Figure 16 Sequence UML diagram of main interactions between users, front-end and back-end

3.4 Front-end Design

3.4.1 Pages and Layouts

The front-end mainly consists of three web pages: an upload page, training page and result page. The upload page, as shown in Figure 17, allows users to either upload a new csv data file or input a Model ID of an existing model.

Then the web application will direct users to training page if no trained model exists and otherwise to the result page.

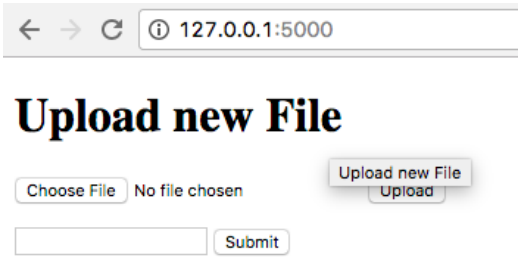

Figure 17 Screenshot of the upload page

Figure 18 shows the 2×2 layout of the training page. In the top left is a time series plot. Users can view target values in the format of a line chart with dates on the x-axis. Users can use the range selector and navigator to view data in certain time periods. The range selector provides buttons to select preconfigured time ranges in the chart, such as 1 month, 3 months and year-to-date (YTD). It also provides input boxes where min and max dates can be manually input. The navigator is a small series below the main line chart, displaying a view of the entire dataset with tools to zoom in and out on parts of the data as well as panning across the dataset. Another important function of the time series chart is for users to select time periods for training and testing. Users need to first place the slider in the navigator over the time period to be selected, and then click the "Set Training Period" or "Set Test Period" button below the navigator. The selected time periods for training and testing will be displayed next to the corresponding buttons. The remaining time after the training period defaults to be the test period. But users can use "Set Test Period" to change that. Below the time series chart is a calendar view. The

calendar view is a heat map[9] which displays target values in a calendar format. In this way, users can easily get an idea of date-related patterns, such as differences between weekdays and weekends. To the right of the time series plot is a scatter plot. Users can select any two variables to display and explore the relationships between different variables, especially between features and target, which might help them to select features for model training.

On the lower right corner is the model setup section. In this section, users can specify the target, choose forecasting type (baseline or next-day prediction) and select features to be included in model training. When the target is selected, the time series chart will update correspondingly to display the selected variable. When the forecasting type is chosen, the features will update accordingly. If next-day prediction is chosen, time series features i and y_{t-1} will show up. After all the model components are set, users can click the "Submit" button to send the information to the back-end. As soon as the model training is finished, the web application will redirect to the result page. Baseline prediction takes fewer than 30 seconds to train a model and make prediction for a dataset of around 1500 points and 4~5 features. Next-day prediction takes at least a few minutes for the same size dataset.

[9] A heat map represents each individual data point contained in a matrix as colors.

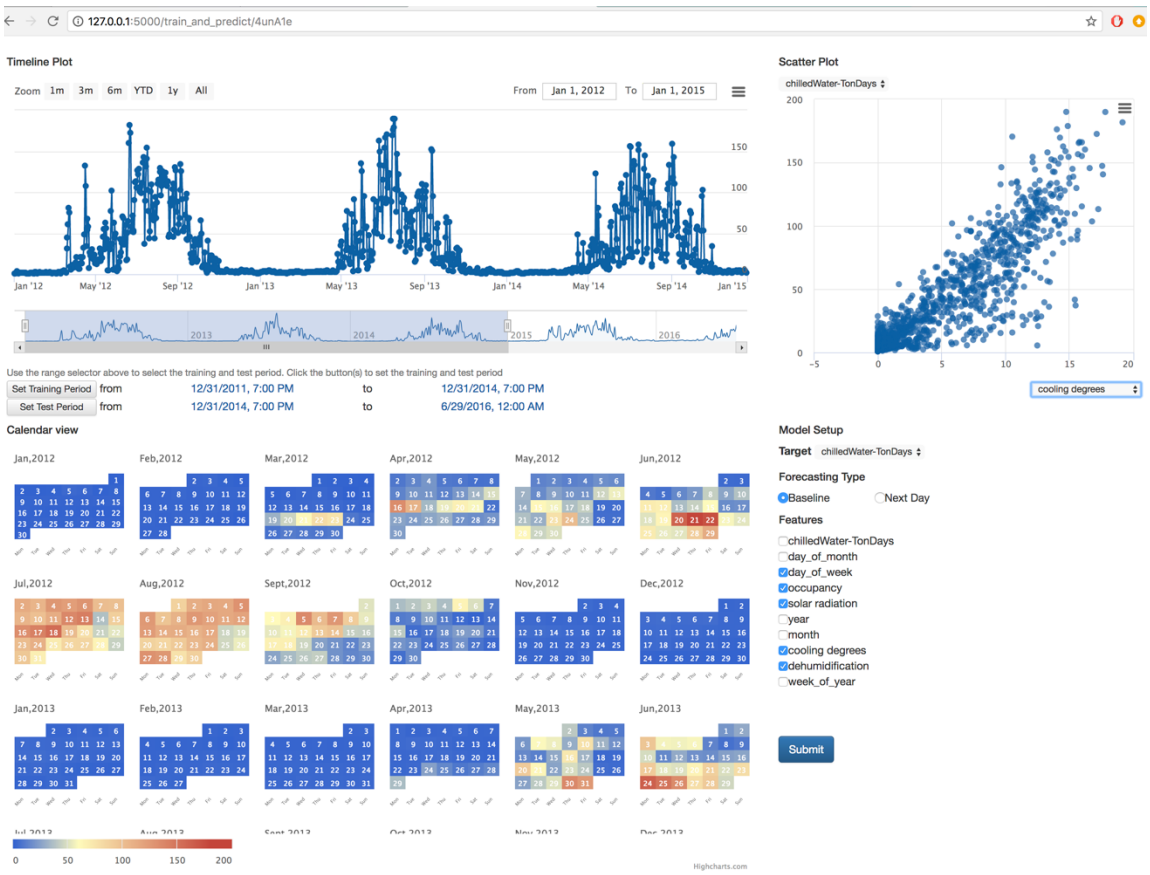

Figure 18 Screenshot of training page

Figure 19 shows the layout of result page. Compared with the training page, there is an additional layer of predicted results on top of the existing time series plot, scatter chart and calendar view. In the time series plot, predicted mean values are marked as blue circles connected with a line and the 95% confidence region is shown as the light blue area. Measured values are marked as blue and orange dots. If a measured point is shown in blue, it means that the measured value falls within the 95% confidence region of the prediction. If it is orange, it is an outlier either caused by an inaccurate prediction or an unexpected energy use. We marked these points as “abnormal” in the database. Similarly, in the scatter plot, the abnormal points are highlighted in orange. In the calendar view, the

dates with abnormal predictions are highlighted with a red circle. When users click on a date in the calendar view, the corresponding data point in time series plot and scatter plot will be highlighted in a larger dot with a black border. This may help users to diagnose their models. The result page also displays information of the trained model. The importance of selected features in GP regression is displayed in a bar chart. Model configurations such as the features and target used in the model, the training and test period, the ratio of training samples to test samples, as well as training and test accuracy are shown in the Model Analysis section.

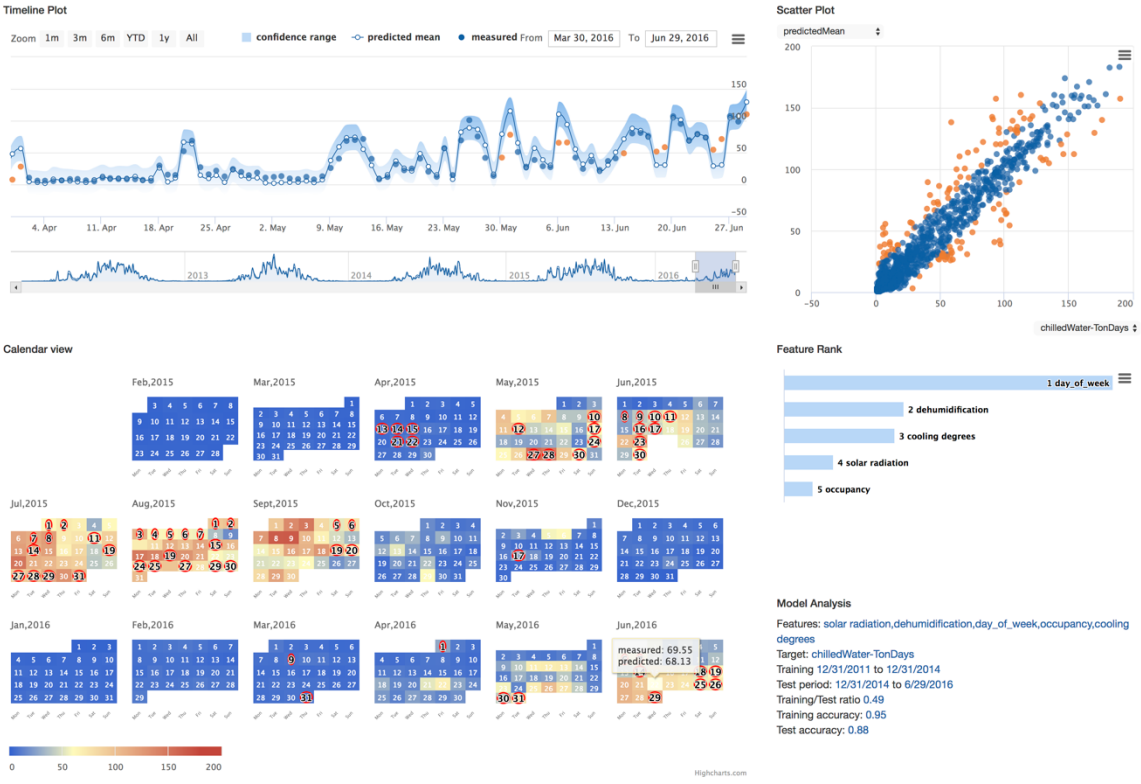

Figure 19 Screenshot of the result page

3.4.2 Visualization Design

In this web application, the visualization focuses on the presentation of raw data and prediction results. The goal is to design interactive and interlinked charts to help discover patterns, improve model training, and interpret results.

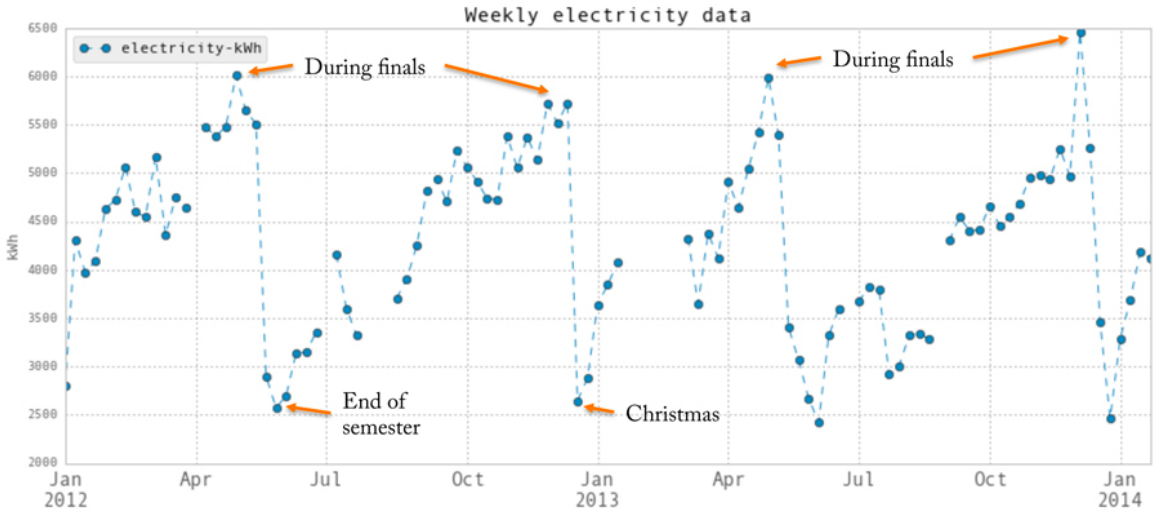

Figure 20 Weekly electric energy consumption of a campus building

Discover Patterns

Even simple charts can help discover interesting patterns in energy consumption. Figures 5 and 19 show the hourly and weekly electrical energy consumption of Gund respectively, a building of Graduate School of Design. In Figure 5, we can see clear differences in electric energy consumption between day and night, as well as weekdays and weekends. The accumulated weekly consumption, as shown in Figure 20, reveals interesting study patterns of the students in design school. During each semester, electricity use ramps up toward a peak at finals, perhaps because the students are working day and night during that week. Additionally, the students are working harder and harder leading up to finals. Then, there is a dip after semesters end, especially during Christmas

vacation. The electric energy consumption is relatively low during January and summer terms, and spring break, when campus is relatively empty. The calendar view helps identify such patterns easily. The calendar view provides users with an easy way to detect patterns related to dates, such as lower energy use in winter/summer recess and higher use during the finals.

Select Features and Training Period

Figures 5 and 19 not only show the energy use pattern, but also highlight time-related features. With the academia schedule, this is crucial for the accurate prediction of electric energy consumption. Moreover, it is important to use at least one year of training data. For example, if one only uses data from January to November in training, then the prediction during Christmas week might be poor. For datasets that show periodic patterns, it is important to include at least a full period for the training set. The cooling energy consumption in Figure 18 shows time-related features do not play a significant role while weather plays a dominant role. In the scatter plot, the cooling energy use has a strong linear relationship with outdoor temperature. As a result, it is important to include data of different seasons in the training set and include outdoor temperature as a training feature.

Interpret Results

After a model has been trained and tested, the model performance is of critical importance. If it predicts some data points poorly, we would like to know which data points and why. For example, does the model perform poorly mostly during unoccupied hours? Or is it because the features of predictive points are not similar to those of training points? These questions can be answered efficiently by interactive and interlinked

visualizations. For example, in the calendar view, users can identify date-related issues caused by holidays, summer/winter recess and other events (e.g., graduation). During holidays or events, occupancy level might be lower or higher than usual. If the unusual occupancy level is not reflected in the features, or there is simply not similar historical data with similar occupancy levels in the training set, then the predictions may fall out of the 95% confidence region of the measured values. Since the "abnormal" predictions are marked with a red circle on the calendar, users might be able to immediately realize that this is due to weekends or holidays. Additional information such as events can be included in the tooltip to help the diagnosis process. When users click a date, the corresponding data point will also be enlarged in the time series and scatter plots. The charts are designed to be interlinked so that users can use additional information such as target measured value, predicted value, predicted confidence region and values of features to further diagnose the model.

In summary, the purpose of visualization is mainly to assist model development, including pattern discovery, feature selection and model evaluation. Ideally, the visualization can help drive the GP in interpreting and understanding the relationship between features and targets, and provide tuning capability and insight into the prediction.

Chapter IV.

Case Studies

4.1 A Campus Building

4.1.1 Dataset

Both campus-level and building-level energy consumption data of Harvard buildings are available through the Energy Witness system[10]. The highest resolution is typically 15-minute but it is different for different buildings and systems. The data can be manually downloaded from the website as a csv file, although data access requires a HarvardKey login.

One building, Gund Hall (Graduate School of Design), was selected for this study. The total area of this building is 15,067m^2. It provides studio and office areas to approximately 500 students and more than 100 faculty and staff. There are also lecture and seminar rooms, a cafeteria, an auditorium and a library in the building. The HVAC (heating, ventilation and air- conditioning) system of this building consists of 9 air handling units (AHU), variable air volume (VAV) boxes and fan coil units (FCU) as terminal units. Three types of energy use are metered, electric energy, chilled water and steam. Electric energy use consists of lighting, plug load, fans and pumps of the HVAC system. Chilled water is supplied from the campus chiller plant and it is used for cooling. Steam is used for heating and domestic hot water[11]. Hourly data from Energy Witness for this building was downloaded and aggregated to daily energy use.

[10] https://www.campusservices.harvard.edu/node/1298

[11] Domestic hot water is hot water used for domestic purpose, for example, sanitation and personal hygiene.

Weather data is collected from a weather station installed on the roof of the building. Dry-bulb air temperature, relative humidity, atmospheric pressure, wind direction, wind speed, gust speed and solar radiation are sampled at 5-minute intervals. Apart from these directly measured variables, dew-point temperature and specific humidity are calculated and included in the potential feature set. Hourly weather features are derived from averaging 5-minute samples, and daily values from averaging hourly samples. Hourly temperature and humidity were transformed to the following variables for cooling and heating forecasting as described below.

Dehumidification h_d (kg/kg):

$$h_d = \begin{cases} h - 0.0087, & h > 0.0087 \\ 0, & h \leq 0.0087 \end{cases} \quad (8)$$

where h (kg/kg) is humidity.

Cooling degrees t_c (°C):

$$t_c = \begin{cases} t - 12, & t > 12 \\ 0, & t \leq 12 \end{cases} \quad (9)$$

where t(°C) is outdoor air dry bulb temperature.

Heating degrees t_h (°C):

$$t_h = \begin{cases} 0, & t \geq 15 \\ 15 - t, & t < 15 \end{cases} \quad (10)$$

The aggregated daily values of these three variables are the sum of hourly values as shown in Equation 11, which is different from deriving other daily weather-related features.

$$h_d = \sum_{i=0}^{23} h_{d,i} \tag{11}$$

where $h_{d,i}$ is hourly sample. Please refer to B. Yan et al. (2017) regarding the reasoning of transforming t to t_c and h to h_d.

Time related features, day of week, day of year and week of year are included in the feature set. An occupancy factor between 0 and 1 is estimated from the academic calendar. For instance, a normal weekday during the semester is assigned as 1, university holidays for staff but not for students are assigned as 0.5 and Christmas is assigned as 0 as the university is completely closed. Note that the value of occupancy is merely an estimate based on the academic calendar, which considers winter/summer recess and holidays but does not consider events.

4.1.2 Prediction Accuracy

Table 2 shows the baseline and trend prediction accuracies of three types of daily energy demand. The training period is from 2012 to 2014. The test period is from January 2015 to June 2016. The base set of features selected for daily prediction is shown in Table 3 (B. Yan et al., 2017). Feature selection and the accuracy of baseline prediction for this case study has been analyzed in B. Yan et al. (2017) This section discusses an analysis of trend prediction accuracy performed for this thesis project. Please refer to Section 2.4 for the terms used in Table 2.

Table 2 Baseline and trend prediction accuracies of three types of energy demand forecasting

	Features				Training set		Forecasting type			Test R^2		
	base set	i	y_{t-1}	f_{t-1}	fixed	growing	base line	next day	30-day ahead	electric energy	chilled water	steam
1	X				X		X			0.52	0.88	0.88
2	X	X			X		X			0.69	0.83	0.77
3	X					X		X		0.73	0.87	0.95
4	X	X				X		X		0.86	0.81	0.95
5	X	X	X			X		X		0.92	0.74	0.96
6	X					X			X	0.44	0.86	0.92
7	X	X				X			X	0.41	0.85	0.75
8	X	X		X		X			X	0.74	0.52	-1.09

Table 3 Base set features selected for daily prediction

	Features selected
Electric energy	Day of year, week of year, weekday, occupancy, heating degrees
Chilled water	Cooling degrees, solar radiation, weekday, dehumidification, occupancy
Steam	Day of year, week of year, heating degrees, weekday, occupancy

The accuracy of baseline prediction for electric energy consumption is poor because the electric energy use meter was malfunctioning in 2016. The electric energy consumption of the investigated building is the sum of readings from two meters. One meter stopped functioning for a few days in October and November 2015, and then completely stopped. As a result, the metered value is lower than its actual consumption and the predicted baseline, as illustrated in Figure 21. This shows that baseline prediction is able to detect anomalies in energy consumption.

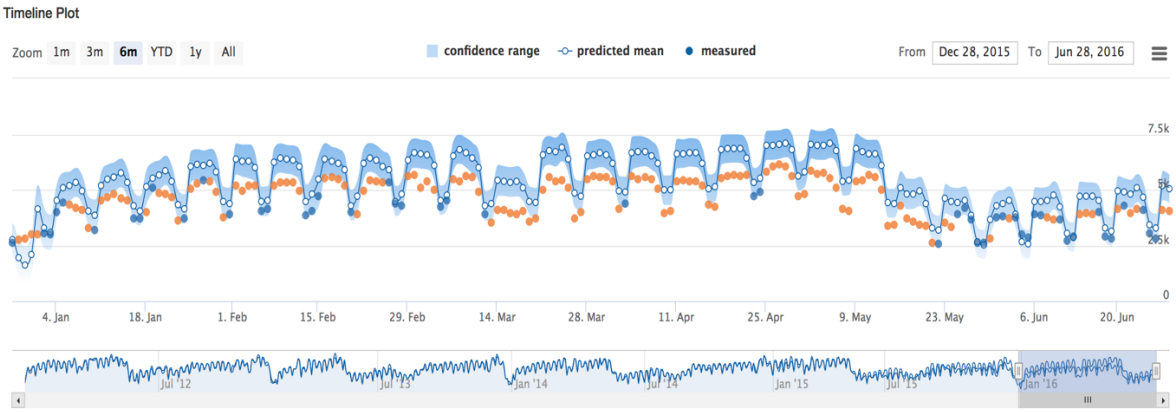

Figure 21 Baseline prediction of electric energy consumption of the Gund building

For electric energy demand prediction, including both feature i (the order of the data points), and the latest data points available in the training set y_{t-1}, improves "prediction accuracy". In this case, claiming "higher accuracy" might be misleading since the one of the two meters stopped logging. On the other hand, a malfunctioning meter is to some extent equivalent to a change of pattern in energy consumption. If half of the building were unknowingly shut down from a certain point of time, would it be possible to predict the energy demand in the next few days? By using i as a feature, new data points get higher weights in the training set. Since the data points to be predicted deviate significantly from old training data points, putting more weight on newer data points improves accuracy. The accuracy of 30-day ahead prediction using $f_{\mathrm{t}-1}$ is much higher than baseline prediction. Next-day prediction which uses $y_{\mathrm{t}-1}$ yields very "accurate" results, as shown in Figure 22. Therefore, the prediction approaches proposed in this thesis are both accurate and robust to changes of pattern/behavior in energy demand.

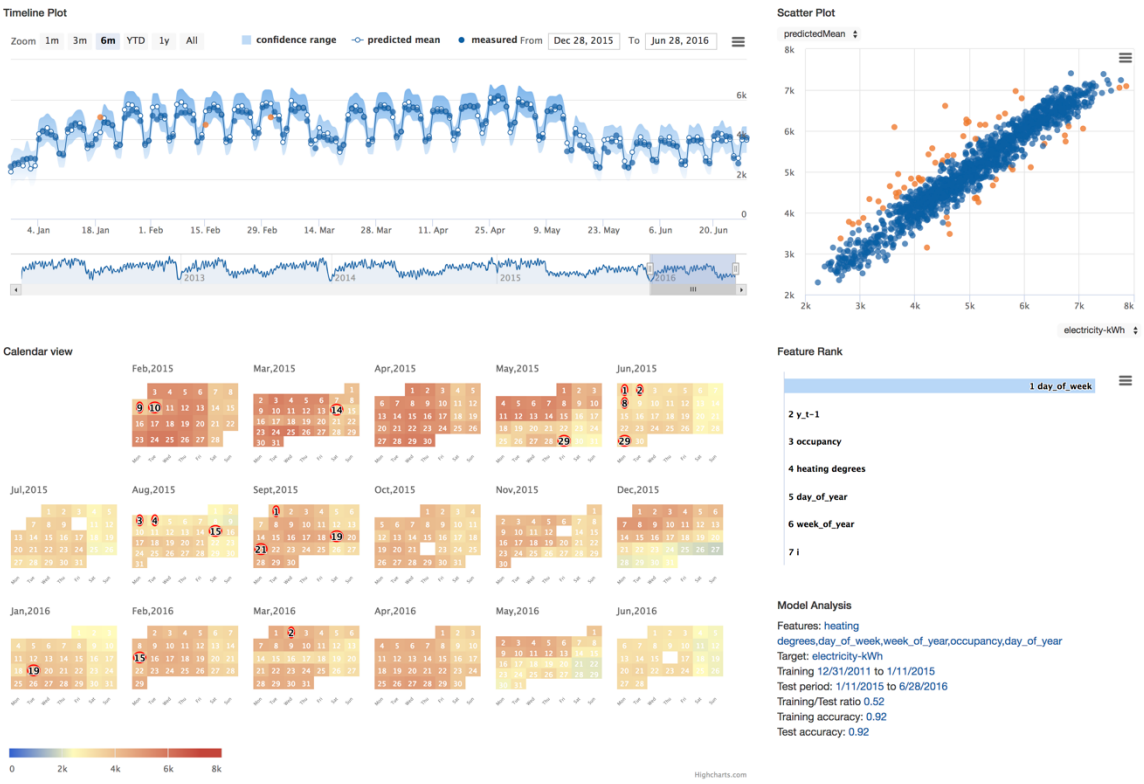

Figure 22 Next-day prediction of electric energy consumption of the Gund building

For chilled water use prediction, including feature i does not help improve prediction accuracy. Including y_{t-1} in next-day prediction or f_{t-1} in 30-day ahead prediction actually make things much worse. For steam use prediction, next-day prediction accuracy is the highest compared to baseline and 30-day ahead prediction, no matter whether including y_{t-1} and i or not. In terms of baseline and 30-day ahead steam use prediction, including feature i decreases the prediction accuracy. The accuracy of 30-day ahead steam use prediction using f_{t-1} is low. Here is a possible explanation. Daily chilled water and steam use has a strong correlation with weather-related features. Using weather-related features as well as some time-related features already yields high accuracy. This also means there is no abnormal pattern in daily cooling and heating

energy use. As a result, putting more weight on more recent data points does not help improve the accuracy of predicting "trend". On the contrary, this might reduce the weight of useful old training data points. When making a prediction, what really matters is finding the historical data points with similar weather and occupancy conditions in the training set. It does not matter whether these most relevant historical data points are old or recent. If we include the order of data points i as a feature, the weights of these useful training data points may decrease if they are not recent. As a result, including i decreases prediction accuracies. Including y_{t-1} might merely add a useless feature. Including f_{t-1} in the feature set is not a good strategy for next-30-day prediction. f_{t-1} is the prediction of the last time step and there is deviation between predictions and actual target values. This inaccurate feature leads to an inaccurate prediction of this time step, which is used as a feature to predict next time step. In 30-day ahead predictions, the accumulated deviation of f_{t-1} from y_{t-1} probably causes poor prediction accuracy. Because the accuracy of 30-day ahead is poor, it is not implemented in the current version of the web app.

4.2 A High-Tech Industry Area

4.2.1 Dataset

The data presented thus far have focused on energy consumption of campus buildings. It is meaningful to test the web app with other types of energy demand data. The dataset used in this case study is from an energy consumption forecasting competition. It consists of daily energy consumption of over 1000 companies in a high-

tech industry area in Jiangsu Province, China from January 2015 to August 2016. The goal of this contest is to forecast the total daily electric energy consumption of the entire industry area, which might be useful for demand response in smart grid applications. The original dataset only consists of dates and daily energy consumption of 1454 companies. Weather data such as daily min and max temperatures can be found online. This case study uses the sum of the daily energy consumption of all the companies as the target and tests the next-day prediction accuracy with and without weather data. The training period is from January 2015 to April 2016; the test period is from May 2016 to August 2016.

Figure 23 shows a screenshot of the training page of this case study. From the time series plot, we can see that the electric energy consumption is constantly around 4 million (unit unknown) except two dips in February 2015 and 2016, which are the Chinese New Year holiday. The calendar view shows that in addition to Chinese New Year, the energy consumption during January 1-2, May 1-2 and October 1-3 are relatively lower. These dates are all important national holidays in China. The energy consumption during March 5 and 6, 2015 is also significantly lower. A possible explanation is that National People's Congress and National Committee of the Chinese People's Political Consultative Conference were held during these dates. It is not unusual to adjust industry activities during these two conferences. However, this pattern does not repeat in 2016, so the actual reason is unknown. The scatter plot shows the relationship between electric energy consumption and minimum daily temperature. When outside temperature is high, the energy consumption increases, but the relationship is not linear.

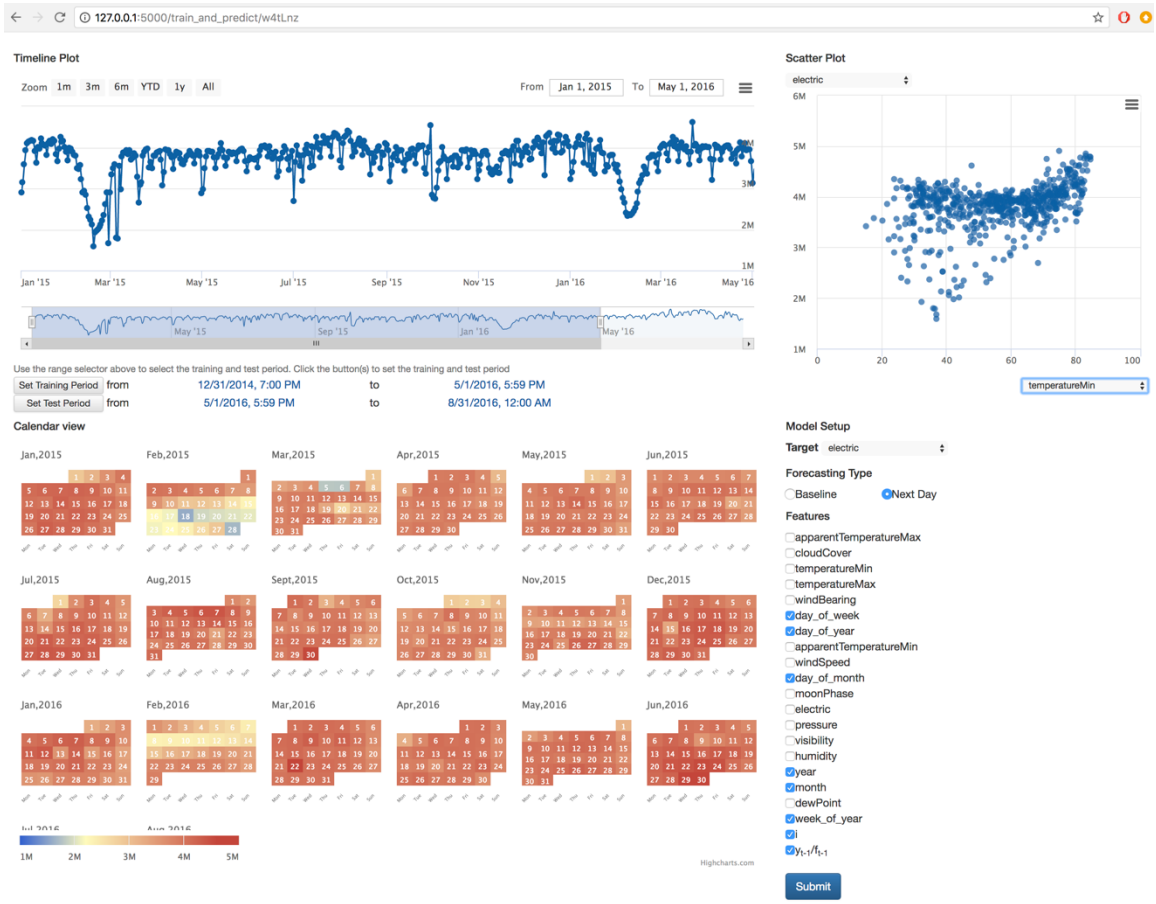

Figure 23 Training page of case study 2

4.2.2 Prediction Accuracy

In this case, we only evaluate next-day prediction accuracy since the goal is to accurately predict future demand, not anomaly detection. Figure 24 shows the prediction results without using weather features. The features included in training are month, week of year, day of week, day of year, day of month, i and y_{t-1}. The R^2 value of test period is 0.60. Assume that weather forecast is relatively accurate. Adding weather related features daily minimum and maximum temperatures increases R^2 to 0.65. The minimum daily temperature ranks as the third most important feature after month and y_{t-1}. Although the

R^2 is not ideal in this case study, the mean absolute percentage error $\frac{100}{\mathrm{n}}\sum_{i=1}^{n}\left|\frac{y_i-f_i}{y_i}\right|$ is only 3.2%.

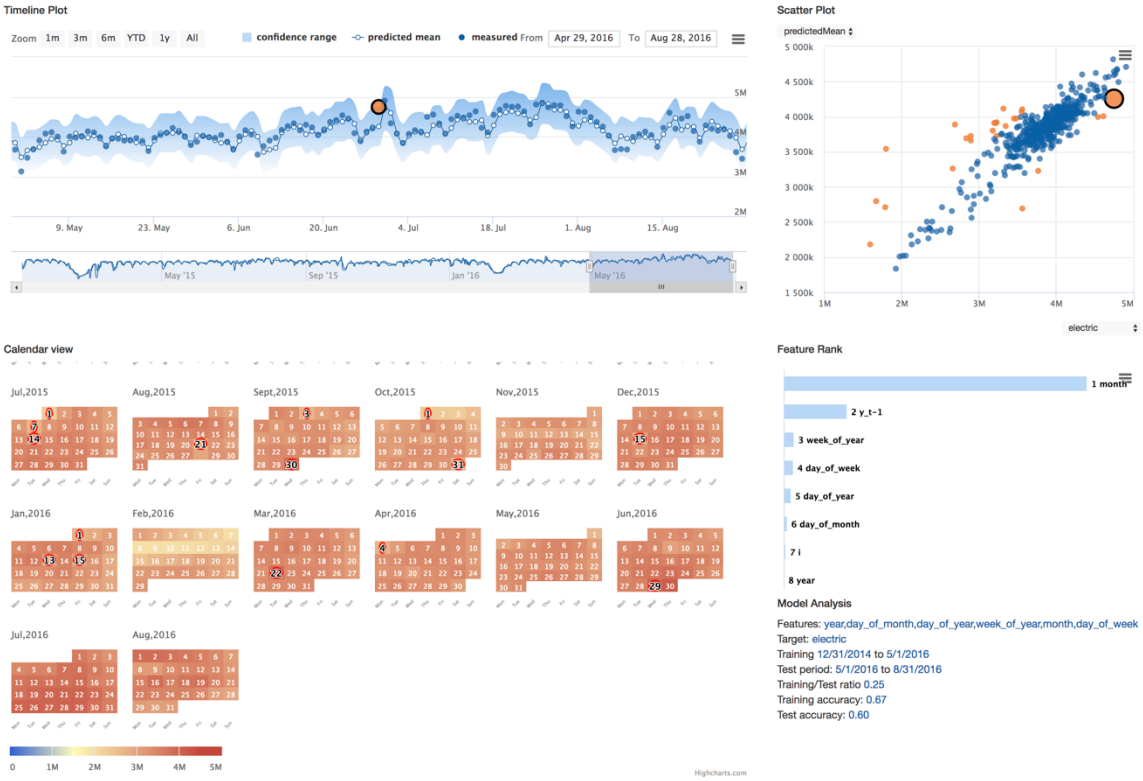

Figure 24 Prediction results of case study 2 without using weather features

Chapter V.

Conclusion and Future Work

5.1 Conclusion

The main contribution of this thesis is a web application for energy demand forecasting using algorithm packages and web app development tools including GPflow, Flask, Highcharts.js, MongoDB and Heroku. Users without programming skills can upload their own time series data, train a Gaussian process regression, and predict baseline energy consumption for anomaly detection, or predict next-day energy demand for smart grid applications such as demand-response. The web interface mainly consists of a time series plot, a scatter plot, and a calendar view. The time series plot shows the original data, predicted mean values and predicted confidence range. The scatter plot assists feature selection by showing the relationship between the target and the features. The calendar view provides users with an easy way to identify patterns related to dates. The charts are designed to be interactive and interlinked so that users can combine the information from different charts to interpret the results and further diagnose the model. The web app also ranks the importance of the selected features in a bar chart. The feature ranking is based on the characteristic length-scales derived from GP training. This information helps users fine-tune feature selection to improve their models.

This thesis demonstrates the use of the web app through two case studies: (1) the daily electric energy, chilled water and steam consumption of a campus building in U.S. and (2) the daily electric energy consumption of a high-tech industry area in China. The results show that weather-related features are the most important in chilled water and steam consumption predictions, while time-related features are the most important to

electric energy usage prediction. Baseline prediction is able to detect anomalies in energy consumption by using the confidence range output by a GP regression. The next-day prediction accuracy is high even when there is an abnormality in energy usage or measurements. The prediction accuracy of next-30-day prediction is low and therefore not implemented in the web application.

5.2 Future Work

There is still a lot of work left to improve the web application. I would like to further improve visualization design, extend the web application to allow datasets larger than a few thousand points, and include additional machine learning algorithms to better compare predictive performance.

5.2.1 Visualization

Although the web application is designed for energy demand prediction, the GP regression algorithm can be applied to any type of dataset as long as all the data points are numeric. Most of the plots are designed to display time series data. Therefore, this web app requires the dataset to contain a column of date and time. The time series chart can visualize data of any time interval. However, currently the calendar view only allows daily energy demand prediction. In the future, the calendar view will need to be redesigned for hourly prediction or other time intervals. Moreover, additional feedback from more users would be valuable in making the visualization more user-friendly.

5.2.2 Large Datasets

The current version of web application has been tested using approximately 1,700 data points. The capability and the limitation of the web application for large datasets need to be tested. A well-known drawback of GP regression is the expensive computation for matrix inversion in training process that scales in $O(N^3)$. Prediction for variances scales in $O(N^2)$ and memory requires $O(ND + N^2)$. In order to work with large datasets, we need to adopt sparse Gaussian processes. The underlying idea in this approach is to use a subset of the data (Banerjee, Dunson, & Tokdar, 2012) to train a model.

Data storage also leads to bottlenecks since currently, all the data and prediction results are stored in database and the free Heroku plan only allows 500MB for the entire web application. This is not a problem for a few thousand data points. But for larger datasets, we need to upgrade the Heroku plan or explore other choices. Rendering a large number of data points in the visualization might also slow down the web application or even cause a problem. Fortunately, the WebGL-powered Boost module in Highcharts.js enables fast rendering for millions of data points. It is able to render a line chart with one million points in less than 500 milliseconds on a MacBook Pro with an AMD Radeon R9 M370X GPU (Highcharts, 2018). In case we need to visualization large datasets, we only need to update to the newest version of Highcharts.js and add one line of code in JavaScript.

5.2.3 Other Algorithms

The web application can be further extended to include other algorithms such as a deep neural network. Since the back-end is written in Python, it is fairly easy to

incorporate TensorFlow for this purpose. It would be ideal to provide other prediction algorithms in addition to GP regression. Users can compare different models and even use ensemble learning.

Appendix

Web app link[12]:

https://calm-falls-41279.herokuapp.com

Demos:

https://calm-falls-41279.herokuapp.com/train_and_predict/c0G0uL

https://calm-falls-41279.herokuapp.com/visualize/nnhNaq/chilledWater-TonDays

Video tutorial:

https://drive.google.com/open?id=1HahVDuf7qxlTobXynRTudYIS5CY8qF-S

Code and data:

https://github.com/bin-yan/energy-demand-forecasting-app

[12] In case the web link changes, I will post updates on Github.

References

Amazon.com. (2017). Retrieved December 06, 2017, from https://aws.amazon.com/types-of-cloud-computing/

Bacon, D. F., Bales, N., Bruno, N., Cooper, B. F., Dickinson, A., Fikes, A., . . . Kogan, E. (2017). *Spanner: Becoming a SQL System.* Paper presented at the Proceedings of the 2017 ACM International Conference on Management of Data.

Banerjee, A., Dunson, D. B., & Tokdar, S. T. (2012). Efficient Gaussian process regression for large datasets. *Biometrika, 100*(1), 75-89.

Burger, E. M., & Moura, S. J. (2015). Gated ensemble learning method for demand-side electricity load forecasting. *Energy and Buildings, 109*, 23-34.

Burkhart, M. C., Heo, Y., & Zavala, V. M. (2014). Measurement and verification of building systems under uncertain data: A Gaussian process modeling approach. *Energy and Buildings, 75*, 189-198.

Che, J., & Wang, J. (2014). Short-term load forecasting using a kernel-based support vector regression combination model. *Applied Energy, 132*, 602-609.

Cohen, D., & Krarti, M. (1995). *A neural network modeling approach applied to energy conservation retrofits.* Paper presented at the Building Simulation Fourth International Conference.

Gray, F. M., & Schmidt, M. (2016). Thermal building modelling using Gaussian processes. *Energy and Buildings, 119*, 119-128.

Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research, 3*(Mar), 1157-1182.

Heo, Y., Choudhary, R., & Augenbroe, G. (2012). Calibration of building energy models for retrofit analysis under uncertainty. *Energy and Buildings, 47*, 550-560.

Heo, Y., & Zavala, V. M. (2012). Gaussian process modeling for measurement and verification of building energy savings. *Energy and Buildings, 53*, 7-18.

Highcharts. (2018). Render millions of chart points with the Boost Module. Retrieved January 2, 2018, from https://www.highcharts.com/blog/news/higcharts-boost-module/

Jetcheva, J. G., Majidpour, M., & Chen, W.-P. (2014). Neural network model ensembles for building-level electricity load forecasts. *Energy and Buildings, 84*, 214-223.

Kennedy, M. C., & O'Hagan, A. (2001). Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology), 63*(3), 425-464.

Kim, Y.-J., Ahn, K.-U., Park, C., & Kim, I.-H. (2013). *Gaussian emulator for stochastic optimal design of a double glazing system.* Paper presented at the Proceedings of the 13th IBPSA Conference, August.

Kissock, J. K., Reddy, T. A., & Claridge, D. E. (1998). Ambient-temperature regression analysis for estimating retrofit savings in commercial buildings. *Journal of Solar Energy Engineering, 120*(3), 168-176.

Kwac, J., Flora, J., & Rajagopal, R. (2014). Household energy consumption segmentation using hourly data. *IEEE Transactions on Smart Grid, 5*(1), 420-430.

Leith, D. J., Heidl, M., & Ringwood, J. V. (2004). *Gaussian process prior models for electrical load forecasting.* Paper presented at the Probabilistic Methods Applied to Power Systems, 2004 International Conference on.

Lophaven, S. N., Nielsen, H. B., & Sondergaard, J. (2002). *DACE A MATLAB Kriging Toolbox* (Technical Report IMM-TR-2002-12). Retrieved from Denmark:

MacKay, D. J. (1997). Gaussian processes-a replacement for supervised neural networks? *Tutorial lecture notes for NIPS 1997*.

Majidpour, M., Qiu, C., Chu, P., Gadh, R., & Pota, H. R. (2015). Fast prediction for sparse time series: Demand forecast of EV charging stations for cell phone applications. *IEEE Transactions on Industrial Informatics, 11*(1), 242-250.

Manfren, M., Aste, N., & Moshksar, R. (2013). Calibration and uncertainty analysis for computer models–a meta-model based approach for integrated building energy simulation. *Applied Energy, 103*, 627-641.

Matthews, A. G. d. G., van der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., . . . Hensman, J. (2017). GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research, 18*(40), 1-6.

MongoDB, I. (2016). *Top 5 Considerations When Evaluating NoSQL Databases*. Retrieved December 20, 2017, from https://www.mongodb.com/nosql-explained

MongoDB, I. (2017a). MongoDB Manual: Databases and Collections. Retrieved December 20, 2017, from https://docs.mongodb.com/v3.2/core/databases-and-collections/.

MongoDB, I. (2017b). MongoDB Manual: Documents. Retrieved December 20, 2017, from https://docs.mongodb.com/v3.2/core/document/.

Neal, R. M. (1995). *Bayesian learning for neural networks.* (PhD PhD thesis), University of Toronto.

Rasmussen, C. E. (1996). *Evaluation of Gaussian processes and other methods for non-linear regression.* (PhD PhD thesis), University of Toronto.

Rasmussen, C. E. (2006). Gaussian processes for machine learning. Cambridge: MIT Press.

Rasmussen, C. E. (2011). The Gaussian Process Web Site: Software. Retrieved November 10, 2017, from http://www.gaussianprocess.org/ - code

Rasmussen, C. E., & Nickisch, H. (2015). Gaussian Process Regression and Classification Toolbox version 3.6. Retrieved November 10, 2017, from http://www.gaussianprocess.org/gpml/code/matlab/doc/

Yan, B. (2013). *A Bayesian approach for predicting building cooling and heating consumption and applications in fault detection.* (PhD PhD thesis).

Yan, B., Li, X., Shi, W., Zhang, X., & Malkawi, A. (2017). *Forecasting Building Energy Demand under Uncertainty Using Gaussian Process Regression: Feature Selection, Baseline Prediction, Parametric Analysis and a Web-based Tool.* Paper presented at the the 15th International Building Performance Simulation Association Conference, San Francisco.

Yan, B., & Malkawi, A. M. (2013). *A Bayesian approach for predicting building cooling and heating consumption.* Paper presented at the 13th International Building Performance Simulation Association Conference.

Yan, J., Kim, Y.-J., Ahn, K.-U., & Park, C.-S. (2013). *Gaussian process emulator for optimal operation of a high rise office building.* Paper presented at the Proceedings of 13th International Building Performance Simulation Association Conference.

Zheng, Y., & Kwoh, C. K. (2011). A feature subset selection method based on high-dimensional mutual information. *Entropy, 13*(4), 860-901.