FORECASTING CUSTOMER'S ENERGY DEMAND USING MACHINE LEARNING

SAIFUL ABU

Department of Computer Science

APPROVED:

Christopher Kiekintveld, Chair, Ph.D.

M. Shahriar Hossain, Ph.D.

Paras Mandal, Ph.D.

Charles Ambler, Ph.D.
Dean of the Graduate School

to my

MOTHER and FATHER

with love

FORECASTING CUSTOMER'S ENERGY DEMAND USING MACHINE LEARNING

by

SAIFUL ABU

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Computer Science

THE UNIVERSITY OF TEXAS AT EL PASO

July 2016

Acknowledgements

I would like to express my deep-felt gratitude to my advisor, Dr. Christopher Kiekintveld of the Computer Science Department at The University of Texas at El Paso, for his constant support. I was very new to the research area, and found the early days very depressing. All my attempts to solve the problem were failed at the attempts of first six months. I gave up. But he did not give up on me. He set aside a time for weekly meeting to know my progress. He listened to my works and results. I found him a reliable person to share my research related as well as personal problem. If he was not there, I could not have finished the thesis.

I also wish to thank the other members of my committee, Dr. Shahriar Hossain of the Computer Science Department and Dr. Paras Mandal of the Electrical Engineering Department, both at The University of Texas at El Paso. Their suggestions, comments and additional guidance were invaluable to the completion of this work.

Additionally, I want to thank The University of Texas at El Paso Computer Science Department professors and staff for all their hard work and dedication, providing me the means to complete my degree and prepare for a career as a computer scientist. This includes (but certainly is not limited to) the following individuals:

Dr. Olac Fuentes

I worked with Dr. Fuentes as teaching assistant (TA) for the course algorithms and data structures. He made the TAs to attend the classes. I attened all his classes. I am very much influenced by his teaching. He is very detail oriented. He is strict at the same time supportive. The classes helped me learn the concepts related to algorithms and data structures. With his support, an extreme introvert like me was able to run the lab section for the course and was able to develop public relational skills. I express my deepest gratitude to him.

Oscar Veliz

I am forever in debt to Oscar, my colleague in the IASRL lab. Though he is a busy PhD student, he found time to go through my initial and sloppy thesis draft. I found him very hard working, helpful and encouraging. He always had good suggestions for all my queries. I regret that I we did not have more conversation during my stay in the lab.

Dr. Ivan Gris

Ivan is one of the smartest person I have ever seen. He is very practical and supportive. I received numerous suggestions and advices from him. I found him very reliable and hard working. His mere presence influences a crowd around him very positively.

My Roommates

I received tremendous amount of support from my two room mates Porag and Sunny. They were peaceful. Porag reviewed my draft thesis several times. I thank them for all the things they did to me.

And finally, I must thank my dear wife for all her supports and understandings.

NOTE: This thesis was submitted to my Supervising Committee on the July 21, 2016.

Abstract

Accurate electricity demand forecasting is an important problem as the failure to do so may be costly for both economic and environmental reasons. Power TAC simulation system provides a no risk platform to do research on smart grid based energy generation and distribution. Brokers are important components of the system. The brokers work as self-interested entities that try to maximize profits by trading electricity in various markets. To be successful, a broker has to forecast the electricity demand about its customers as accurately as possible, otherwise it will operate ineffectively. This proposed forecasting method uses a combination of cluster and classifiers. At first, the customers are clustered based on their weekly average usage. After that, energy usage history and related weather related information are combined together to train classifier for the cluster. To forecast for a new customer, the proposed method needs at least a week's energy usage history of the customer. The system assigns the new customer to one of the clusters based on its electricity usage history. The classifier for that cluster will be used to forecast the customer. This approach produced 13 % error compared to 31% relative absolute error observed against the moving average baseline predictor. The Power TAC system has six different types of customer such as customers with demand shifting capabilities, customer with no demand shifting capabilities, electric vehicles, thermal storage, wind production and solar production. Previous approaches to demand forecasting treated all types of customers equally. This work shows that a good forecasting system should treat customers of different type differently, otherwise the system will experience more error.

Table of Contents

List of Tables

List of Figures

Chapter 1

Smart Grid and PowerTAC Competition

In this chapter, I will describe the smart grid [5] and Power TAC [14].

1.1 Traditional Electricity Distribution and Consumption System

In traditional power grids, there are three subsystems electricity generation, transmission, and distribution [5]. In electricity generation subsystem, the generator rotates a turbine in a magnetic field which generates electricity. The turbine rotates through the power of kinetic energy of water falling from a waterfall or a river with strong current, or from the energy of nuclear power plant, or energy received from burning coal or oil. Traditional energy generation system then transmits the electricity through transmission grid and electricity gets distributed through the distribution grid. This generation system is one way meaning a the electricity flow occurs from source node to consumption node only.

1.2 Smart Grid

In contrast to the traditional electricity generation system, Smart Grid are two-way [5]. So, any node in the distribution grid can produce electricity and push it to the distribution grid if necessary. The NIST report [5] states that the SG would make the electricity generation and supply robust against generator or distribution node failure, use renewable energy widely and efficiently, reduce greenhouse gas emission, reduce oil consumption by encouraging usage of electric vehicles, it will give customers more freedom to choose among energy sources. Smart grids will encourage usage of the electric vehicle as these vehicles have the ability to store power in a battery and transmit the power to the distribution grid if there is a necessity. The major challenge with the usage of renewable energy is it is uncertain. This uncertainty causes the ability to predict how much energy the SG can produce in a future time slot hard. The success of SG will need efficient methods to predict energy production [22].

1.3 Smart Grid and Renewable Energy

One of the major focus of Smart Grid will be using renewable energy. There are challenges involved with using this abundant source of energy [25]. People are already showing strong motivation to use renewable energy as indicated by the statistics that 20% of total energy is from the renewable sources which are second after coal 24%. Consumers are using renewable energy due to economic reward and environmental concern. A major challenge with renewable energy is the amount of the energy produced is greatly variable. Since the energy produced is volatile there must be a storage mechanism that balances out the surplus energy. The usage of rechargeable electric vehicles might serve the purpose of storage. Accurate prediction of the renewable energy might enable the electric car users to absorb surplus energy and push it back to the grid in peak hours if necessary.

1.4 Power TAC System

Power Trading Agent Competition (Power TAC) [14], [15], is a low-risk system that simulates a smart grid based energy system. This simulation system models a competitive and liberal energy trading market. The power TAC simulation has several components such as wholesale market, brokers, customers, distribution utility and weather service [15].

The brokers publishes tariff plans for electricity consumers and producers. It then buys electricity from the wholesale and balancing market to meet customers need. The system is trained on customers behavior of past years and uses real weather data. The following sections give a brief explanation of each component of the Power TAC. Figure 1.1 shows a block diagram of the components of the powerTAC simulation environment.

1.4.1 Broker

In Power TAC system, participants implement their broker logics. Here are the list of actions that a broker can take in Power TAC simulation -

- At any hour of the simulation, a broker can publish a new tariff. Each tariff is targetted to a specific category of the customers. The tariffs contain information about which category of customers it is targetted, expiration date of the tariff, signing bonus, penalty for early withdraw and rate of periodic payment.

- At any moment of time, a broker can modify its published tariffs. It can adjust payment rates and withdraw penalty. The broker can also revoke a tariff that is not profitable.

- To meet customer's demand, a broker takes part in the electricity auction in the wholesale market. There, it specifies how much energy it needs and how much it is ready to pay for. Based on the asks and bids from other brokers in the simulation, the broker's bid may or may not get cleared.

At any moment of time the brokers are aware of the following information -

- Participating brokers in the simulation

- Customers present in the simulation.

- Bootstrap data of customers and wholesale market.

- Published tariff in the simulation.
- Information about tariff modification or revocation
- Wholesale market clearing prices of last time slot.
- Energy transaction information of subscribed customers.
- Transaction of balancing market.
- Current bank balance of itself.

1.4.2 Customers

A customer represents an entity that buys energy from the brokers. A customer has the following attributes -

- An unique name.
- Number of individuals it represents. This number can range from one to several thousands.
- A power type that specifies which category of customers (producer or consumer) does it fall into. Producer and consumer categories have several subcategories.

A customer can take the following actions during a simulation -

- Evaluate available tariffs in tariff market.
- Subscribe or abandon a tariff. Customers try to maximize their economical gain so if there is a lucrative tariff in the market, a customer might try to subscribe to it.
- Generate meter reading based on produced or consumed energy. The system then sends this meter reading to the broker it subscribed to.
- Customers with demand shifting capabilities can shift their demand to favorable time slot.

1.4.3 Weather Service

The weather service broadcasts weather forecast of future hours and weather report of current hour to the brokers. The weather report contains information such as wind speed, cloud cover, temperature, day of week and month of week. Power TAC uses real weather data from the past that makes the simulation more realistic. Brokers can use these information to forecast demand for the weather sensative consumers. The weather information also makes it possible to forecast about renewable energy producers.

1.4.4 Markets and Distribution Utility

There are three different types of markets in Power TAC simulation, namely, wholsale market, tariff market and balancing market. The wholesale market is the bidding place for buying energy. Bulk energy producers and brokers take part in the wholesale market auction. Brokers can submit their bids for 24 future timeslots in the wholesale market by specifying the price it is prepared to pay. If the bid was successful, the broker receives its desired amount by paying the money. At each time slot, the system notifies the broker about the wholesale market clearing prices. Brokers publish their tariff plans in the tariff market. A tariff holds information about the pricing of the energy. Customers, upon analyzing available tariffs, subscribe to their mostly suited tariff plan. Balancing market represents the market from where the broker can buy energy in case of emergency. For example, if a broker has bought less amount of energy for a given timeslot and it finds it needs more energy then it can buy the necessary amount of energy from the balancing market. Usually, the balancing market transactions are costly for brokers than the wholesale market.

The distribution utility has two main objective. First, it supplies energy to the consumers from whole sale market and from the renewable producers. Secondly, it works as a default broker that publishes default tariffs at the start of the game. This makes sure no customer is ever out of energy. Other brokers are supposed to publish lucrative tariffs to attract customers.

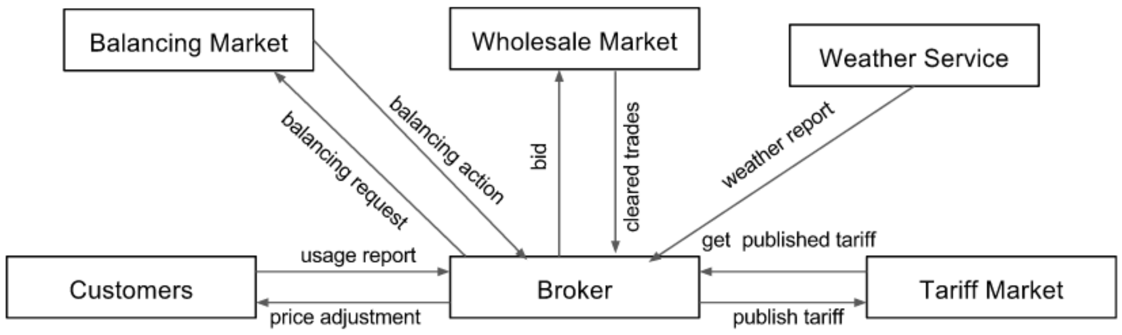

Figure 1.1: PowerTAC simulation environment.

1.5 Importance of Accurate Demand Forecasting in Power TAC

A broker has to make bids and asks in the wholesale market. The amount of electricity it asks depends on the demand forecast of its subscribed customers. If the broker fails to make accurate demand forecast, it will not be able to ask for proper amount of electricity. So it will end up asking more or less energy than the required amount in the wholesale market. In this case, the broker will have to buy energy from the balancing market in a higher price or has to sell surplus energy in a lower rate. As a result, it will face monetary losses. This unwanted scenario can be avoided through demand forecasting as accurate as possible. This thesis investigates ways to find a better demand forecasting mechanism.

Chapter 2

Related Works

In this chapter, I have described different methods of energy load forecasting for long term and short term in the literature. It is hard to know the state of the art electricity consumption mechanism in Power TAC as most of the researchers did not publish their demand forecasting mechanis [16], [20], [28], [19]. So I mostly describe the works done for real world electricity demand forecasting mechanisms.

2.1 Variables in Electricity Demand

Studies such as [10], [7] and [4] have found that temperature has effect on electricity demand. The study in [7] was done in a region of Australia. It was found that, in a lower temperature the customers tend to use heaters and in a higher temperature they tend to use coolers. As a result, the increase or decrease of temperature from a certain point will cause the consumption of electricity to increase. In study [4], two demand forecasting models were proposed. One was univariate Auto Regressive Integrated Moving Average (ARIMA) and another one was univariate ARIMA model along with temperature depended transfer function. The model with temperature variable did better forecasting than the one without the temperature variable. On the other hand, the study in [3] showed that inclusion of temperature variable in forecasting model actually introduced more error in demand prediction. The aim of the study was to make forecasting about electricity usage of January based on past five years training data using a Support Vector Machine (SVM) forecaster. The reason behind of getting more error after including temperature variable may be because during Januray the temperature did not change much and the inclusion might have caused overfitting.

Weather variables such as wind speed and cloud cover has effect on electricity demand [10], [26]. As cloud cover increases, the demand for light increases too. The increased lighting demand causes increased electricity demand. The period where cloud cover was low, the electricity demand was also low [10]. High speed wind across wet walls help cool houses. High speed wind thus may cause reduced electricity demand due to reduced demand of air cooling [26].

In the survey article [6], the authors reported that the day of the week and the month of the year is highly correlated with customer's energy demand. The electricity load demand can be higher or lower based on the day of week. The weekends usually have different load demand pattern than the week days. Also, based on the hour of a given day, the load demand can be higher or lower too. The season also showed impact on electricity demand.

2.2 Electricity Load Forecasting Using Statistical Method

To make electricity load forecast, researchers have used statistical methods such as statistical average, Auto Regressive Integrated Moving Average (ARIMA) and exponential smoothing. Agent TACTEX'13 [29], the winner of the PowerTAC competition in 2013, used the statistical average to make electricity demand forecasting for an hour of a day of a week. In a week a customer has $24 * 7 = 168$ hours or slots where it can consume electricity. TACTEX'13 kept track of average usage of 168 weekly slots for each customer. To predict a future time slot, their agent would look at which weekly slot the future time slot would fall in. Then the agent used that weekly slot's average usage as the forecast of the future time slot. [4] have used an ARIMA model for load forecasting. The ARIMA model uses both moving average and auto regression to forecast the demand. To make a forecast about a future time slot, the auto regression model uses some previously observed time slots values based on its degree. Moving average scheme would use the average of all the known time series data points to make a prediction about a future time slot. In the study [2], a short term load demand was proposed that uses several ARIMA models. For the combination of week day and temperature level, 16 short term load forecasting models were used. This scheme made better forecasting than a single ARIMA model. The authors in [13] used modified Halt Winter Exponential Smoothing for demand forecasting. The modified exponential method was cabaple of dealing with weekly and daily seasonality pattern present in the data.

2.3 Load Forecasting using Machine Learning

Support Vector Machine(SVM) proved to be an effective tool for load forecasting [27], [3]. In [3], the authors used SVM to forecast electricity demand of January based on past 5 years electricity consumption data. Separate SVM models for separate were proposed. SVM model trained with data from January was able to make better load forecasting. Artificial Neural Network (ANN) is another favorite load forecasting mechanism among the researchers [12], [23], [11]. The author [23], used ANN to model Prediction Interval (PI) to forecast renewable energy forecasting. The author [12] used ANN to figure out the time horizon suitable for solar energy production. [21] the authors used various machine learning techniques to make 24 hours ahead load forecast for the Power TAC simulation. They found that hour of week, weather related features such as temperature cloud cover were influential to the electricity load demand. The forecasting modules made low error while forecasting for the customers that showed regularity in their energy consumption behavior. The application of linear regression [17] hernandez2012classification and Kalman Filtering [1] also appeared for demand forecasting in the literature.

2.4 Load Forecasting using Clustering

Clustering can be used to group consumers with same electricity demand pattern [8]. The authors [17], applied clustering on 6 months electricity usage of household consumers in

Ireland. Application of clustering generated common load patterns called load profiles which were used to forecast about future load demand. The authors [4] noticed that customers can be categorized to improve accuracy of demand forecasting. They manually clustered the customers in four groups namely, commercial, office, residential and industrial customers. For Power TAC environment, the authors [30] proposed a broker that clusters the bootstrap data of customers and generates a linear regression classifier for demand prediciton for each cluster. The authors [8] used kMeans and Self Organaizing Map to cluster industrial park's consumption in Spain to understand micro environments present in a larger environment.

2.5 Expert System based Load Forecasting

Expert system based electricity demand prediction contains variables that are likely to affect electricity demand [24], [9]. This system then mimicks a human operator's steps to load forecast. This load forecasting mechanism appeared applicable for short term load forecasting [24], [9], [18].

From the review of the literature, the importance of weather related variables such as temperature, cloud cover and wind speed is evident. Also, the hour of the day and day of the week are highly correlated with the load demand. A combination of machine learning classifiers and clustering algorithms appears to be a better idea. For the methodology of [21] it will take a large number of predictors for the simulation system. Also, those predictors will not work if the name of the customer is changed or a new customer is introduced as each predictor is hard coded with a specific customer. It sounds reasonable to cluster the data first and then train machine learning classifier for each cluster. This approach will hold generality. Instead of training only on bootstrap data as proposed in [30], a wealth of data generated from the simulations can be used to train the cluster. Since the clustering is done offline, the proposed approach will not suffer from the problem of having a time limit that the broker has to face if the cluster is trained during the competition. After the clustering is done, for each cluster, different machine learning classifiers can be trained to figure out which one performs the best. So, the broker will no longer stick to linear regression as in [30].

Chapter 3

Customer Description

In this chapter, I will describe the customers present in the PowerTAC simulation system and some statistics their attributes.

3.1 Customer Categories

In Power TAC simulation, a customer can be electricity consumer or producer based on the power type it has. A customer evaluates the tariff plans targetted for its power type and can look for the tariff that gives it maximum monetary benefit. There are several types of customers in the Power TAC simulation such as consumption, interruptible consumption, thermal storage, solar production, wind production and electric vehicle. Each power type has its own characterstics. For example, interruptile consumption customers can shift their electricity demand to some off peak hour, the solar production customers can produce energy based on the weather condition. As opposed to previous methods on demand forecasting, I argue that each category of customers based on the power type should be treated differently. One load forecasting method can be suitable for a category of customers while it may be unsuitable for other categories because each category behaves differently. In the below section, I describe the characterstics of the customers.

- **Consumption:** A customer with power type consumption are the most common customers. They use the energy when they need it. They cannot shift their demand to a future timeslot. Usually, they have a regular pattern of their energy usage. Often, they show a similar pattern for weekdays. They have similar kind of usage pattern for the weekends.

- **Interruptible Consumption:** Interruptible customers are smart enough to shift their energy demand in a timeslot where they can buy electricity at a reduced price. Because of this shifting capability, they don't show a regular usage pattern as the consumption customers do.

- **Thermal Storage:** Thermal storage customers show a weekly pattern in their electricity usage. Also, during a day, their electricity usage in a day depends much on the energy they used in the last timeslot.

- **Solar Production** The solar energy production customer's energy production depends on the cloud cover. They are highly likely to produce energy during the day time.

- **Wind Production** Wind production customers generate energy from the wind.

- **Electric Vehicle** An electric vehicle customer represents one electric vehicle. Their usage of energy is quite irregular and hard to predict.

To discuss the behavior of different power typed ustomers, a representative customer was chosen from each type. The table 3.1 shows the customer chosen to discuss -

Power Type	Customer Name
Consumption	downtown offices
Interruptible Consumption	village 1 ns
Thermal Storage	sf2
Solar Production	sunnyHill
Wind Production	windmill 1
Electric Vehicle	high income 1

Table 3.1: Representative customer from each power type

Before diving into the problem of solving forecasting methods, I found it useful to take a look at how the customers behave. The log extractor program extracts all the energy consumption and production by all the customer on all the timeslot. At the end of the game, it makes a report on normalized usage of all the customers in all the week slots. Normalized values are useful because even if the amount of energy usage among the customers vary, the pattern of usage can be captured through it and normalized usages of different customers can be plotted in the same graph. The figures 3.1 to 3.3, shows normalized electricity demand or supply of Mondays. 0 in the x axis means hour 12:00 am. From the figures 3.1 to 3.3, it is clear that some customer's electricity usage is higly correlated with time of the hour of the day. The customers of type consumption and solar energy can be example of these types. The other customer types demands did not seem to have correlation with hour of a day. In both consumption and solar energy customers the consumption or production curve grew smoothly till it reached a peak point. After the reaching the peak, the production or consumption reduced smoothly. The customers with types interruptible consumption, wind production, thermal storage and electric vehicle had irregular demand/supply pattern.

In figure 3.4 to 3.6, normalized electricity usages during the week are shown. Hour 0 to 23 represents all the hours of Monday from 12:00 am to 11:00 pm. It appears that, the electricity demand for consumption customer, interruptible consumption, thermal storage had repeatative demand pattern everyday. The solar energy production customer showed repeatative production pattern. Moreover some customers such as the consumption type and the thermal storage type showed different pattern during the weekend. During the weekend they usually had lower energy demand than the weekdays. In the case of electric vehicle and wind energy production customers the demand/supply patterns were not regular. From these observations, it can be assumed that the consumption customer and the solar energy type customers can be forecasted with the most accuracy. Due to the irregular patterns of other customers, it will be harder to make accurate demand forecast about them.

3.2 Statistics

In this section, I present some statistics on the customers available in the system.

- **Customer Vs PowerType**

  The figure 3.7 shows number of customers in each power type for a typical Power TAC simulation game. The electric vehicle power type has the most number of customers. This is because the electric vehicle represents a population of size 1. Consumption and interruptible consumption power type customers follow the electric vehicle power type customers in terms of number of customers.

- **Population Vs PowerType** From figure 3.8 by far the powertype of consumption has the most number of population. Some customers can represent thousands of individuals. For this reason, even though there are only a few numbers of consumption and solar energy customers, they can represent a population of size several thousands.

- **Total Energy Consumed Vs PowerType** The figure 3.9 and **??** show the contribution in energy transaction by different power type customers in a typical Power TAC simulation. From the figures, we can see that the consumption type customers are responsible for the most amount of energy transaction (more than 60%). After the consumption type customers, the interruptible consumption type customers trade 18% of total traded energy. The solar production customers caused the 11% of total energy transaction. Undoubtedly, a successful broker needs to forecast the consumption type customer with prime importance because of the bulk of energy they transact. Due to this fact, I concentrated mostly on forecasting about consumption type customers' demand.

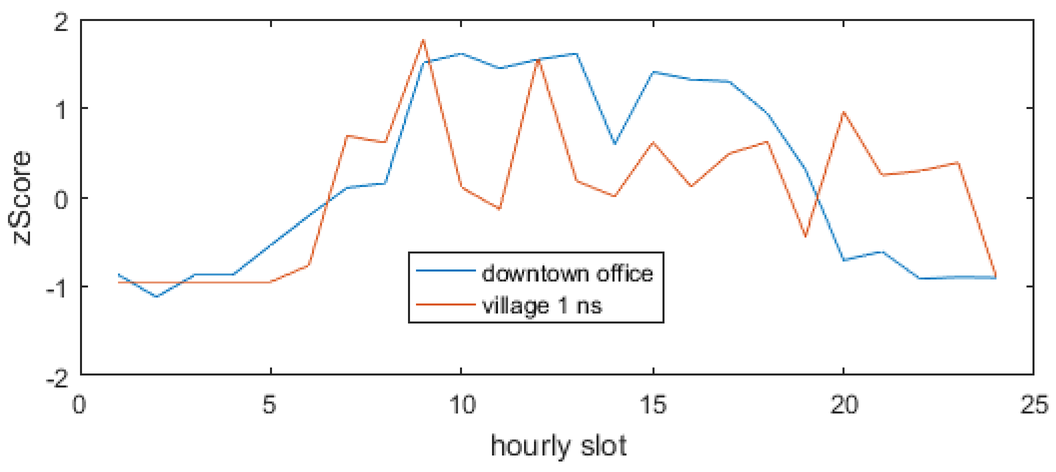

Figure 3.1: Caption 1

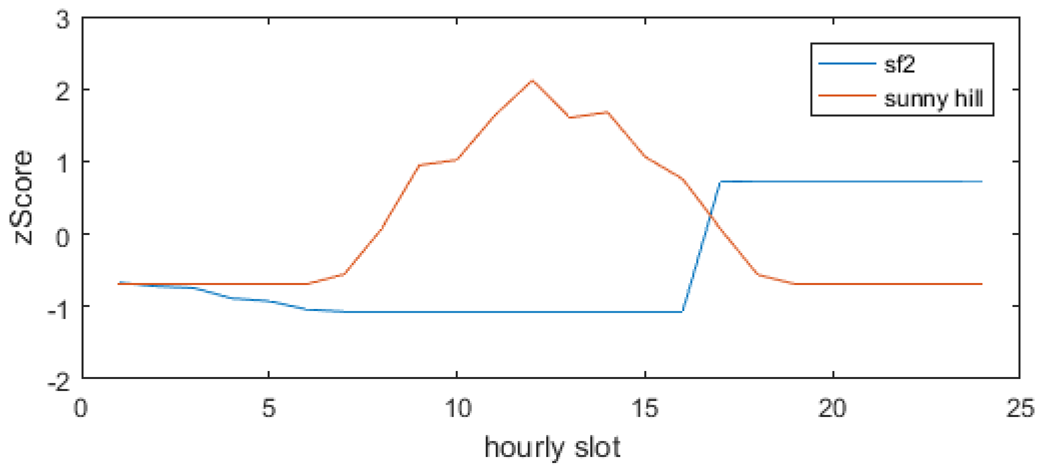

Figure 3.2: Caption 1

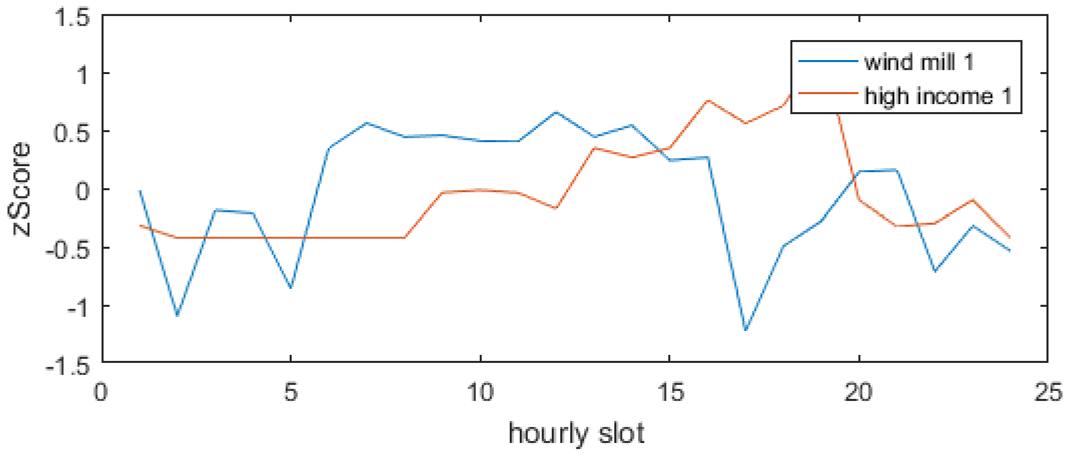

Figure 3.3: Caption 1

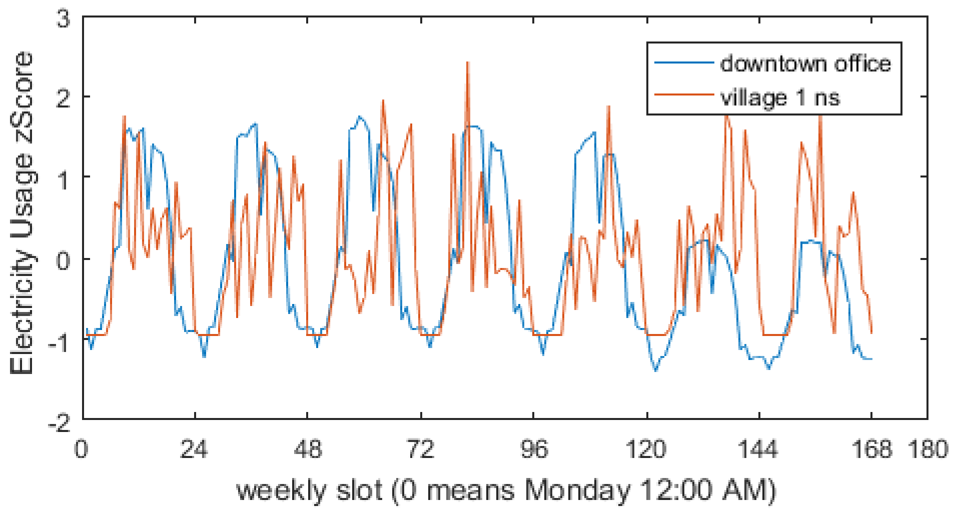

Figure 3.4: Caption 1

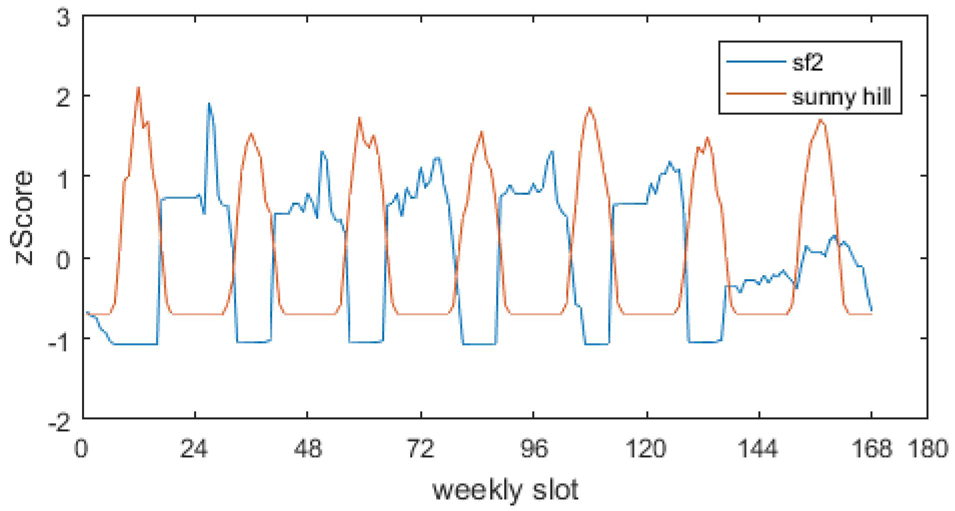

Figure 3.5: Caption 1

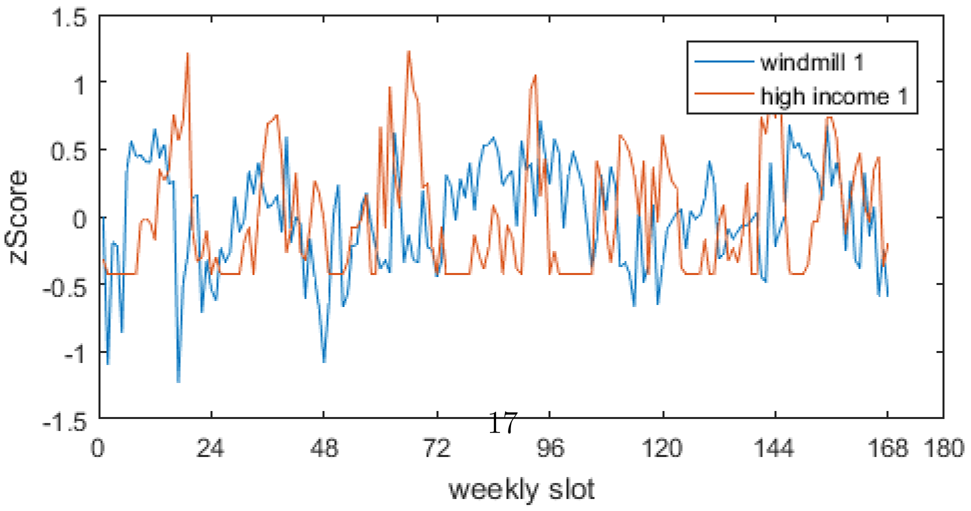

Figure 3.6: Caption 1

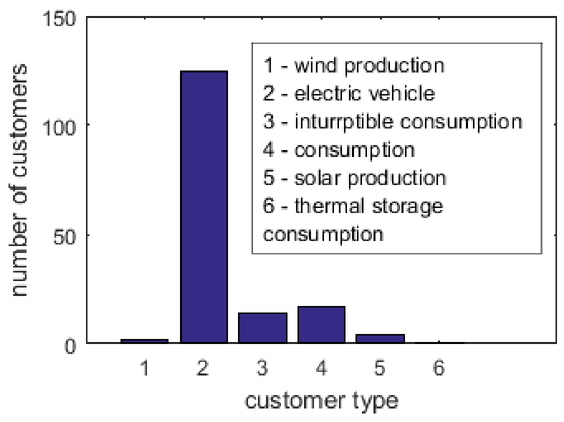

Figure 3.7:
Number of customers vs Powertype.

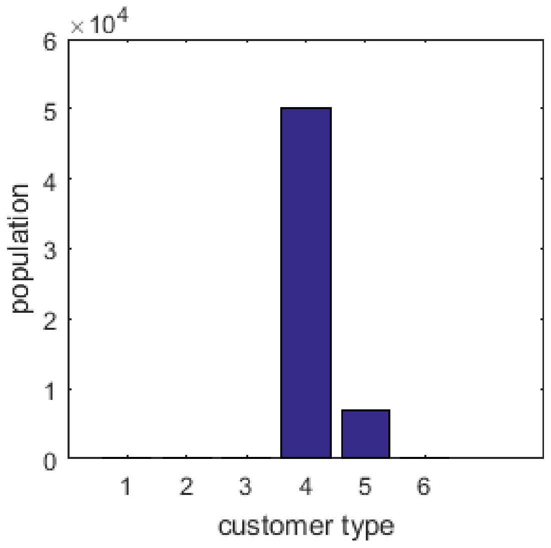

Figure 3.8: Population vs Powertype

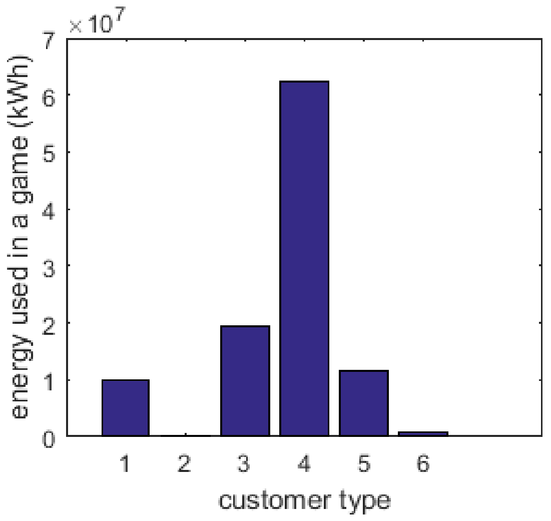

Figure 3.9: Energy vs PowerType.

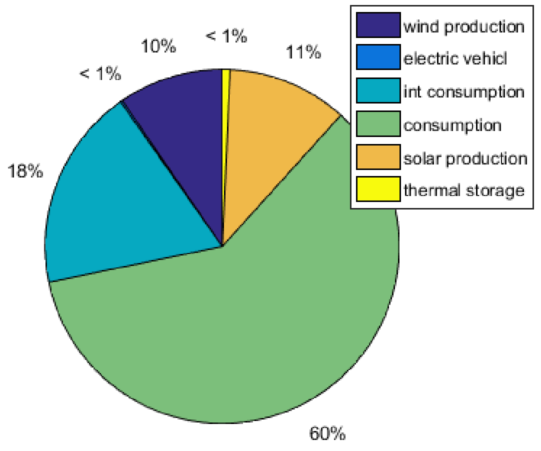

Figure 3.10:
Energy share for each power type.

Chapter 4

Methodology and Results

In this chapter, I have described the methodology used to forecast consumption type customer's demand. Traditionally, a single type of predictor served to predict the energy demand of all power type customers. Since each power type customers acts differently, I planned to make different forecasting mechanism for different power typed customers.

4.1 The Baseline Electricity Forecasting Mechanisms

The first baseline energy forecasting mechanism is the default prediction mechanism provided by the PowerTAC system. It exploits the fact that usage of a timeslot of a customer in a specific date is highly correlated with the day of the week and the time slot. To make a prediction it stores the average energy usage of an hour of a week. So, for each customer, it uses $24 * 7 = 168$ memory to remember average usages. As soon as it learns about a new usage information of an hour of a week, it updates old average using the following algorithm.

Algorithm 1 Update average usage for $customer_i$ for day d and timeslot t, $newUsage$

```
1: avgUsage = get average usage of $customer_i$ at day d and time slot t
2: $avgUsage = 0.7 * avgUsage + 0.3 * newUsage$
```

Algorithm 2 forecast usage for day d and timeslot t for $customer_i$

```
1: avgUsage = get average usage of $customer_i$ at day d and time slot t
2: return $avgUsage$
```

The second baseline forecasting mechanism is designed to make energy forecasts for a single customer. In general, if there are n customers in the system, we will need n energy forecasters each one trained on the data of a single customer. I went further by checking different machine learning algorithms such as M5Tree, Linear Regression, M5P rules and REP tree for each customer and picked the best performing one for each customer.

4.2 Proposed Electricity Demand Forecasting Mechanism

In this section, I will describe how I attempted to make energy demand forecaster for consumption power type customers.

4.2.1 Demand Forecasting for Consumption Type Customer

For the consumption type customers, algorithm 3 describes the proposed method of forecasting energy demand and how it was compared to the baseline methods.

Algorithm 3 Make electricity demand forecasting for consumption type customer

```
1: Extract features for each time slot for each customer [algorithm 4, 5 and 6 ]
2: train kmeans cluster for different sizes of k [algorithm 7]
3: train linear regression classifier for each cluster and compute error [algorithm 8]
4: pick suitable value for k by observing the errors
5: for each cluster, find the best performing predictor for that cluster [algorithm 9]
6: train individual classifer for each customer to make the second baseline [algorithm
   10]
7: evaluate performance using test data [algorithm 11]
```

Algorithm 3 begins with extracting information from the game log files. All the activities that occurred in a game can be found in a game log. Activities such as buying or selling electricity occur during a time slot. At the beginning of a time slot, the system notifies the broker that a new time slot is about to begin. The system also notifies the brokers with weather forecast about the future time slots. As a time slot ends, the broker receives information about its customer's energy usage which is called tariff transaction report. Algorithm 4 refers how the extraction program retrieves necessary information from tariff transaction report. As the broker gets notification of the beginning of a new time slot, the extraction program has all the information related to energy usage and weather data of the previous time slot available by this time. The extractor program extracts the following features -

- Temperature
- Cloud Cover
- Wind Speed
- Average of the Slot
- Standard Deviation of the Slot

Algorithm 5 shows the procedure of writing the information of the known time slot's information in training instance file. Once the simulation ends, the extraction program knows the average energy usage of all the customers during a week. The extraction program writes all the 168 hourly averages of a week to a file. This is explained in algorithm 6.

Algorithm 4 extract information from transactionReport sent to broker after each time slot through TariffTransactionHandler call back method

```
1: timeSlot = get time slot from transactionReport
2: customerName = get customer name from transactionReport
3: energyUsed = get energy used from trom transactionReport
4: addUsage(customerName, timeSlot, energyUsed)
```

Next, all the average weekly usages are combined together to make training set for the clustering algorithm. I have used kMeans clustering algorithm to cluster the training set. I

Algorithm 5 write extracted data after timeSlot update message received from TimeSlotUpdateHandler call back method

```
1: knownTimeSlot = timeSlot - 1
2: for each customer do
3:     day = get day of knownTimeSlot
4:     hour = get hour of knownTimeSlot
5:     statisticalData = get statistics of the customer of day and hour
6:     weatherData = get weather data of knownTimeSlot
7:     trueUsage = get true usage of customer in knownTimeSlot
8:     trainingInstance = create training instance by combining statistical data, weather
       data and true usage
9:     writeToFile(trainingInstance)
10: end for
```

Algorithm 6 write average electricity usage of the customers of each hour of the week

```
Require: information of all timeslots has been received
1: for each customer do
2:     trainingInstance = create empty training instance
3:     for each day of week do
4:         for each hour of day do
5:             averageUsage = get average usage of day and hour of customer
6:             append averageUsage to the trainingInstance
7:         end for
8:     end for
9:     writeToAvgUsageFile(trainingInstance)
10: end for
```

have trained clusters of sizes 4, 5, 6, 7, 8, 9, 10 and 11. Algorithm 7 describes the procedure of making clusters from the training instances. Once a k-means of cluster size k is made, a program groups the hourly usages of the customers in the same cluster and combines them to make training set for machine learning classifier. This training set is used to train linear regression classifier. To test the performance of the classifiers, I have separated five game logs and they were not used for training purposes. Algorithm 8 describes how the cluster based predictor's performance was evaluated.

Algorithm 7 create kmeans cluster of size k from weekly usage training instance file

```
1: data = load weekly average usage file
2: kmeansCluster = build kmeans cluster of size k
3: save kmeansCluster
```

Based on the errors observed from different kMeansCluster based predictions, I fixed the number of clusters. Once the number of the clusters was fixed, a program creates several machine learning predictors to see which one performs best for a given cluster. The machine learning classifiers that were tried out are linear regression [31], M5P rules [31], M5 Tree[31], REP tree[31]. In the runtime, a customer will be grouped in a cluster based on its weekly usage. Once the program knows the cluster assigned to a customer, the program will load the corresponding demand forecaster to make electricity demand forecast about the customers.

At this phase, I have the proposed cluster-based customer's demand forecaster. Next, the baseline predictor that needs a machine learning classifier for each customer is built. At first, the training instances are combined based on the name of the customer. This means for n customers n training set is constructed, each of the training set has only the information of a single customer. A training set related to a customer is used to create machine learning classifiers for that customer. Several classifiers had been tried out to figure out which classifier performs the best for a customer. The best performing classifier was chosen to predict about a customer. Algorithm 10 explains the procedure of getting the best classifier.

The next phase is testing the performance of the proposed and baseline methods. For testing, I had used five game logs that were not used for training purposes. For each test instance, all three methods output was observed to figure out the performance. Algorithm

Algorithm 8 find error of kmeans clusters of different size

```
1: for each cluster size k do
2:     get the kMeansCluster of size k
3:     for cluster in KMeansCluster do
4:         combine slot based training instances of that cluster
5:         train linear regression classifier based on the combined data
6:         save the classifier for cluster
7:     end for
8: end for
9: for each training instance do
10:    compute error of the instance using each kMeansCluster
11: end for
```

Algorithm 9 find best classifiers of each cluster of kmeans cluster of size k

```
1: for each cluster in kMeansCluster do
2:     combine slot based data of the all the customers in cluster
3:     train available classifiers on the combined data using 10 fold cross validation
4:     choose the classifier with minimum error
5:     save the classifier for making demand forecasting for cluster
6: end for
```

Algorithm 10 find best classifiers created for each individual customer

```
1: for each customer do
2:     combine all slot based training instance of the customer
3:     train available classifiers on the combined data using 10 fold cross validation
4:     choose the classifier with minimum error
5:     save the classifier for making prediction about the customer
6: end for
```

11 shows the mechanism of testing.

Algorithm 11 performance evaluation of each method

```
1: for each test instance do
2:     classify the test instance using moving average usage [algorithm 2]
3:     classify the test instance using individual prediction mechanism
4:     classify the test instance using cluster based predictor
5:     calculate and accumulate errors of each mechanism [algorithm12]
6:     update moving average baseline predictor based on the information from the test
       instance [algorithm 1]
7: end for
8: find average error from the accumulated errors for each forecasting mechanism
```

Algorithm 12 calculate error from the predicted value and the true value

```
1: absoluteError = abs(predictedValue - trueValue)
2: relativeAbsoluteError = (absoluteError / trueValue ) * 100 %
```

4.3 Result

The following subsections describe the results at each stage of experiments. The stages are finding optimal number of clusters. Once the number of clusters has been fixed, different classifier needs to be made for each cluster to see which one makes the best forecast for a particular cluster. After that, the baseline predictor that needs a classifier for each customer needs to be built. At this point, for each customer several classifier has been tried out to see which classifier makes best demand forecast about that customer. After that, the proposed mechanism has been tested against the two baseline demand forecasting methods.

4.3.1 Finding number of clusters

At first, I have segmented the customer using KMeans clustering algorithm with cluster sizes = 4, 5, 6, 7, 8, 9, 10 and 11. For KMeans with size k, we will have k clusters. For each of the k clusters, I had a linear regression predictor. I observed the relative percentage error and absolute average the above cluster sizes. It turned out that the size of the cluster does not have a big impact on the prediction performance. To keep things simple, I have decided to choose Kmeans cluster of size 4. When k = 4 was chosen, table 4.1 shows the cluster assignment for each customer. It can be seen that, cluster-0 held most of the offices, cluster 2 held most of the village types, cluster 3 held the medical center, cluster 1 held large housing such as brooksidehomes, centerville homes and large offices such as downtown offices and centerville offices.

Customer Name	Assigned Cluster Number
BrooksideHomes	0
CentervilleHomes	0
DowntownOffices	1
EastsideOffices	1
OfficeComplex 1 NS Base	0
OfficeComplex 1 SS Base	0
OfficeComplex 2 NS Base	0
OfficeComplex 2 SS Base	0
Village 1 NS Base	2
Village 1 RaS Base	2
Village 1 ReS Base	2
Village 1 SS Base	2
Village 2 NS Base	2
Village 2 RaS Base	2
Village 2 ReS Base	2
Village 2 SS Base	2
MedicalCenter@1	3

Table 4.1: Assigned cluster for each customer

4.3.2 Finding best predictor for each cluster

Once the features are extracted, I have tried out M5Tree, Linear Regression, M5P rules and REP tree machine learning classifiers to see which one performs the best for each of the 4 clusters. Figure 4.1, 4.3, 4.5, 4.7 show that M5P, M5P, REPTree and M5RULES are the best predictors for cluster 0, 1, 2 and 3 respectively.

The next step is to find the best classifiers for each of the customers. Based on the data from each of the customers, the four types of classifiers described in previously were tried out. For each customer, the following classifiers performed the best.

The figure 4.9 shows error percentage of each of the predictors type for each of the customer types.

Finally, the cluster based forecasting and the two baselines were tested with data extracted from 5 test files that were not used for training. From Figure 4.10 we can see that cluster based prediction mechanism performed almost as good as the mechanism where n predictors are needed for n customers. And it did well than the default moving average prediction scheme.

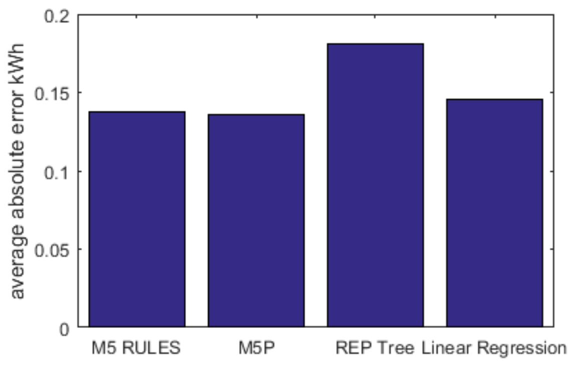

Figure 4.1:
cluster 0 average absolute error of 4 classifiers

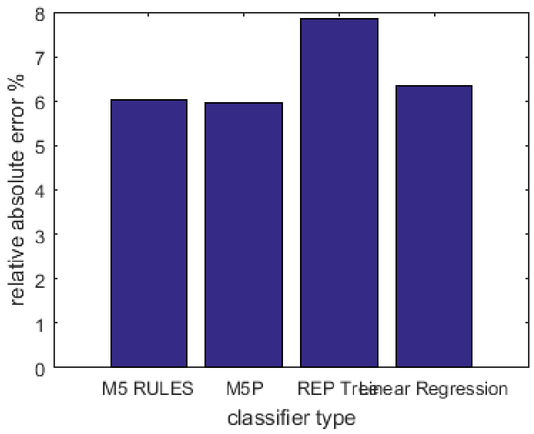

Figure 4.2:
cluster 0 average relative absolute error of 4 classifiers

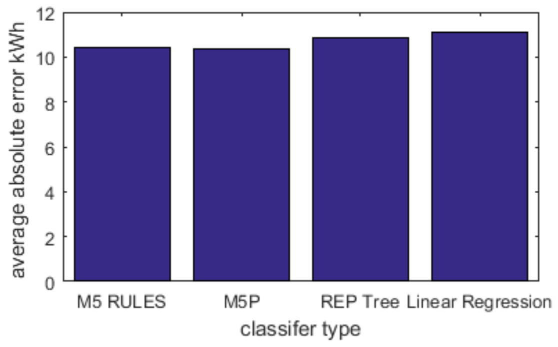

Figure 4.3:
cluster 1 average absolute error of 4 classifiers

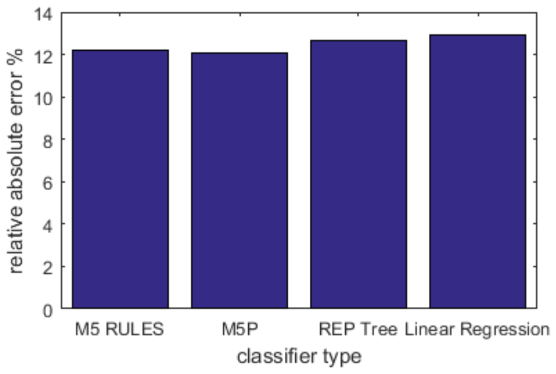

Figure 4.4:
cluster 1 average relative absolute error of 4 classifiers

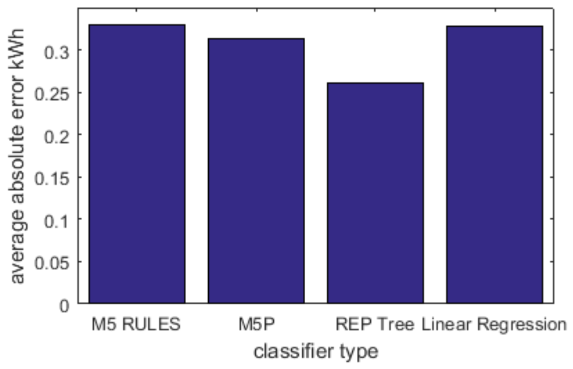

Figure 4.5:
cluster 2 average absolute error of 4 classifiers

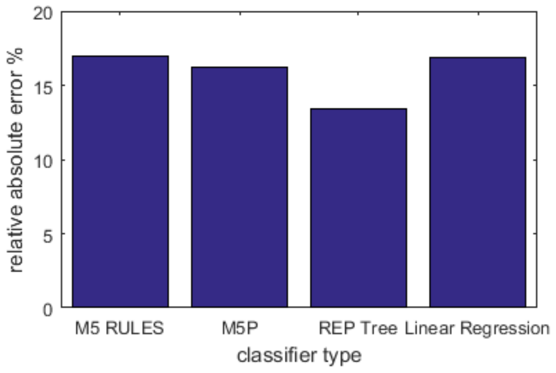

Figure 4.6:
cluster 2 average relative absolute error of 4 classifiers

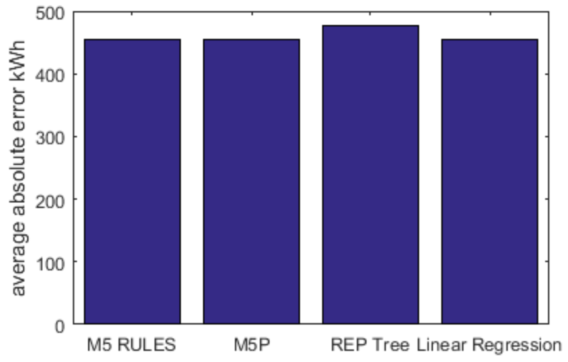

Figure 4.7:
cluster 3 average absolute error of 4 classifiers

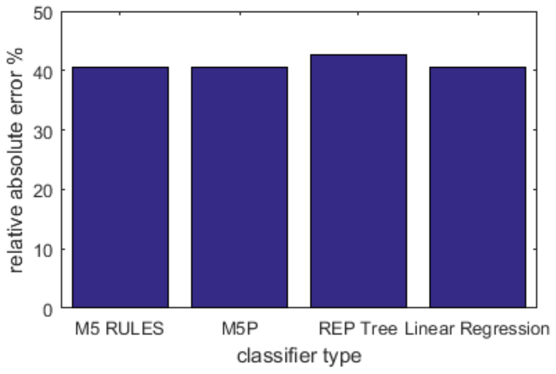

Figure 4.8:
cluster 3 average relative absolute error of 4 classifiers

Customer Name	Best Predictor Type
BrooksideHomes	M5P
CentervilleHomes	M5P
DowntownOffices	M5P
EastsideOffices	M5P
OfficeComplex 1 NS Base	LinearRegression
OfficeComplex 1 SS Base	LinearRegression
OfficeComplex 2 NS Base	LinearRegression
OfficeComplex 2 SS Base	LinearRegression
Village 1 NS Base	M5P
Village 1 RaS Base	LinearRegression
Village 1 ReS Base	M5P
Village 1 SS Base	M5P
Village 2 NS Base	LinearRegression
Village 2 RaS Base	M5P
Village 2 ReS Base	M5P
Village 2 SS Base	M5P
MedicalCenter@1	M5P

Table 4.2: Best individual predictor for each customer

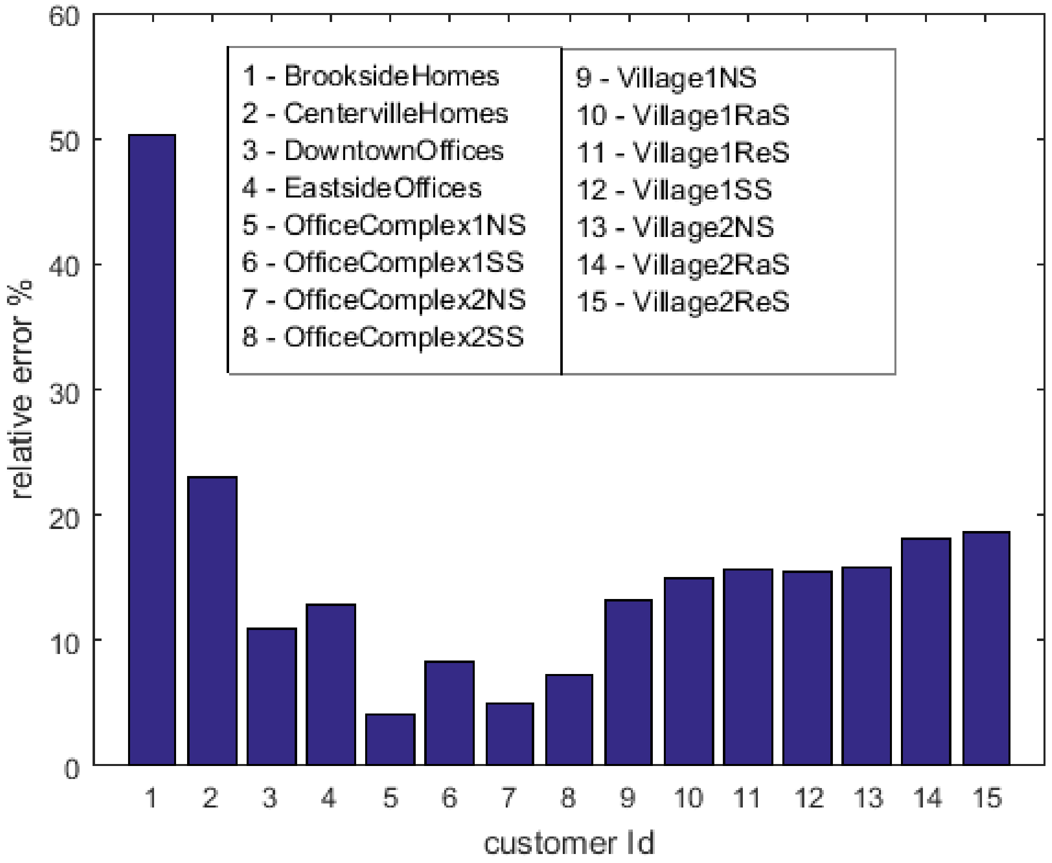

Figure 4.9: Performance of the best classifier for each customer type. Customer Medical center was excluded as it was showing huge error.

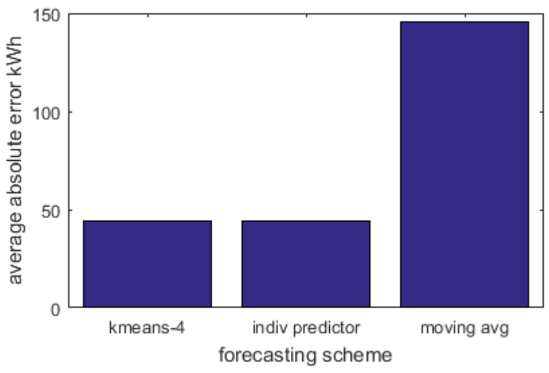

Figure 4.10: average absolute error

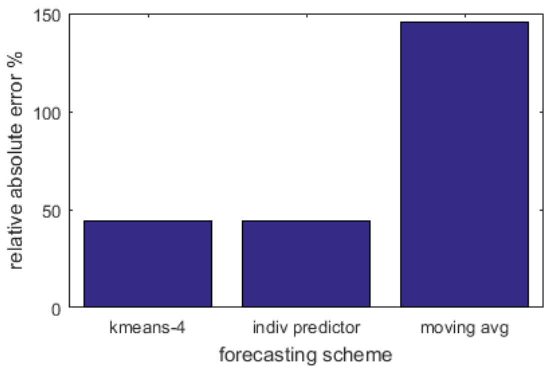

Figure 4.11:
average percent relative absolute error

Chapter 5

Conclusions and Future Work

5.1 Significance of the Result

This work showed that for each power type, a broker should use different demand forecasting mechanism. The work showed that for consumption type customers in the Power TAC simulation, the size of the cluster does not matter. The baseline with individual predictors can be considered as a mechanism with n customer, where n is the number of consumption customers. The proposed forecasting methodology was able to achieve almost similar demand forecasting mechanism using only 4 clusters. The above mentioned baseline has a serious fault, the demand predictors are hardcoded by the customer names. If during the simulation the name of a customer is changed, this mechanism will not work. On the other hand, the proposed mechanism can be trained on previous game logs and does not have the problem mentioned above.

5.2 Future Work

This work only deals with demand forecasting about the consumption customers. The proposed mechanism seems to be applicable for solar energy production customers with a slight change. For customer with irregular demand pattern such as customers with demand shifting capabilities and the electric vehicle customers, different technique of demand forecasting has to be figured out.

References

[1] HM Al-Hamadi and SA Soliman. Short-term electric load forecasting based on kalman filtering algorithm with moving window weather and load model. *Electric power systems research*, 68(1):47–59, 2004.

[2] Nima Amjady. Short-term hourly load forecasting using time-series modeling with peak load estimation capability. *IEEE Transactions on Power Systems*, 16(3):498–505, 2001.

[3] Bo-Juen Chen, Ming-Wei Chang, and Chih-Jen Lin. Load forecasting using support vector machines: A study on eunite competition 2001. *Power Systems, IEEE Transactions on*, 19(4):1821–1830, 2004.

[4] MY Cho, JC Hwang, and CS Chen. Customer short term load forecasting by using arima transfer function model. In *Energy Management and Power Delivery, 1995. Proceedings of EMPD'95., 1995 International Conference on*, volume 1, pages 317–322. IEEE, 1995.

[5] Xi Fang, Satyajayant Misra, Guoliang Xue, and Dejun Yang. Smart gridthe new and improved power grid: A survey. *Communications Surveys & Tutorials, IEEE*, 14(4):944–980, 2012.

[6] Heiko Hahn, Silja Meyer-Nieberg, and Stefan Pickl. Electric load forecasting methods: Tools for decision making. *European Journal of Operational Research*, 199(3):902–907, 2009.

[7] Melissa Hart and Richard de Dear. Weather sensitivity in household appliance energy end-use. *Energy and Buildings*, 36(2):161–174, 2004.

[8] Luis Hernández, Carlos Baladrón, Javier M Aguiar, Belén Carro, and Antonio Sánchez-Esguevillas. Classification and clustering of electricity demand patterns in industrial parks. *Energies*, 5(12):5215–5228, 2012.

[9] Ku-Long Ho, Yuan-Yih Hsu, Chuan-Fu Chen, Tzong-En Lee, Chih-Chien Liang, Tsau-Shin Lai, and Kung-Keng Chen. Short term load forecasting of taiwan power system using a knowledge-based expert system. *IEEE Transactions on Power Systems*, 5(4):1214–1221, 1990.

[10] Ching-Lai Hor, Simon J Watson, and Shanti Majithia. Analyzing the impact of weather variables on monthly electricity demand. *IEEE transactions on power systems*, 20(4):2078–2085, 2005.

[11] Che-Chiang Hsu and Chia-Yon Chen. Regional load forecasting in taiwan—-applications of artificial neural networks. *Energy conversion and Management*, 44(12):1941–1949, 2003.

[12] Ercan Izgi, Ahmet Öztopal, Bihter Yerli, Mustafa Kemal Kaymak, and Ahmet Duran Şahin. Short–mid-term solar power prediction by using artificial neural networks. *Solar Energy*, 86(2):725–733, 2012.

[13] Nur Adilah Abd Jalil, Maizah Hura Ahmad, and Norizan Mohamed. Electricity load demand forecasting using exponential smoothing methods. *World Applied Sciences Journal*, 22(11):1540–1543, 2013.

[14] Wolfgang Ketter, John Collins, and Prashant Reddy. Power tac: A competitive economic simulation of the smart grid. *Energy Economics*, 39:262–270, 2013.

[15] Wolfgang Ketter, John Collins, and Mathijs De Weerdt. The 2016 power trading agent competition. *ERIM Report Series Reference*, 2016.

[16] Bart Liefers, Jasper Hoogland, and Han La Poutré. A successful broker agent for power tac. In *Agent-Mediated Electronic Commerce. Designing Trading Strategies and Mechanisms for Electronic Markets*, pages 99–113. Springer, 2014.

[17] Fintan McLoughlin, Aidan Duffy, and Michael Conlon. A clustering approach to domestic electricity load profile characterisation using smart metering data. *Applied energy*, 141:190–199, 2015.

[18] Ibrahim Moghram and Saifur Rahman. Analysis and evaluation of five short-term load forecasting techniques. *IEEE Transactions on power systems*, 4(4):1484–1491, 1989.

[19] S Ozdemir and R Unland. Agentude: The success story of the power tac 2014s champion. In *AAMAS Workshop on Agent-Mediated Electronic Commerce and Trading Agents Design and Analysis (AMEC/TADA 2015)*, 2015.

[20] Serkan Ozdemir and Rainer Unland. A winner agent in a smart grid simulation platform. In *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 2, pages 206–213. IEEE, 2015.

[21] Jaime Parra Jr and Christopher Kiekintveld. Initial exploration of machine learning to predict customer demand in an energy market simulation. In *Workshops at the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[22] Cameron W Potter, Allison Archambault, and Kenneth Westrick. Building a smarter smart grid through better renewable energy information. In *Power Systems Conference and Exposition, 2009. PSCE'09. IEEE/PES*, pages 1–5. IEEE, 2009.

[23] Hao Quan, Dipti Srinivasan, and Abbas Khosravi. Short-term load and wind power forecasting using neural network-based prediction intervals. *IEEE transactions on neural networks and learning systems*, 25(2):303–315, 2014.

[24] Saifur Rahman and Rahul Bhatnagar. An expert system based algorithm for short term load forecast. *Power Systems, IEEE Transactions on*, 3(2):392–399, 1988.

[25] Andre Richter, Erwin van der Laan, Wolfgang Ketter, and Konstantina Valogianni. Transitioning from the traditional to the smart grid: Lessons learned from closed-loop supply chains. In *Smart Grid Technology, Economics and Policies (SG-TEP), 2012 International Conference on*, pages 1–7. IEEE, 2012.

[26] Ina Rüdenauer and Carl-Otto Gensch. Energy demand of tumble driers with respect to differences in technology and ambient conditions. *Öko-institut, Freiburg*, 2004.

[27] Nicholas I Sapankevych and Ravi Sankar. Time series prediction using support vector machines: a survey. *IEEE Computational Intelligence Magazine*, 4(2):24–38, 2009.

[28] Jonathan Serrano, Enrique Munoz de Cote, and Ansel Y Rodríguez. Fixing energy tariff prices through reinforcement learning.

[29] Daniel Urieli and Peter Stone. Tactex'13: a champion adaptive power trading agent. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1447–1448. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[30] Xishun Wang, Minjie Zhang, Fenghui Ren, and Takayuki Ito. Gongbroker: A broker model for power trading in smart grid markets. In *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 2, pages 21–24. IEEE, 2015.

[31] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2005.

Appendix A

Source Code

```
package tools;

import java.util.Arrays;
import java.util.LinkedList;

import weka.classifiers.Classifier;
import weka.classifiers.functions.LinearRegression;
import weka.classifiers.rules.M5Rules;
import weka.classifiers.trees.M5P;
import weka.classifiers.trees.REPTree;

public class Settings {

public static final int NUM_FOLD_CROSS_VALIDATION = 4;
// stats files.
// Phase 1 : separate training instances of slot based files based on
// clusters
// Phase 3: combine training instances of slot based files
static String CUSTOMER_NAME = "cluster-test";
public static String TRAINING_ROOT_FOLDER_NAME = "TrainingData";
public static String OVERALL_STATS_FOLDER_NAME = "OverallStats";
static String OVERALL_STATS_HEADER_LOCATION = "header/header-overallstats.txt
public static String CLUSTER_FOLDER_NAME = "Clusters";
public static String SLOT_BASED_TRAINING_SET_FOLDER_NAME = "SlotBasedTrainin
public static String INDIVIDUAL_PREDICTOR_FOLDER_NAME = "IndividualPredictors
// log extracted file's setting
public static String LOG_EXTRACTED_ROOT_FOLDER_NAME = "SimulationData"; //
// public static String OVERALL_STATS_FOLDER_NAME = "OverAllStats"; //
public static String AVERAGE_FOLDER_NAME = "AverageUsage";
public static String zScoreUsageFolder = "ZScoreUsage";
public static String bootStrapFolder = "BootStrapUsage";
public static String slotBasedFolderName = "SlotBasedUsage";
public static String CLUSTER_PREDICTOR_FOLDER_NAME = "ClusterPredictors";
// public static String SLOT_BASED_HEADER_LOCATION =
// "header/header_V6-feature-set-1.txt";
public static String SLOT_BASED_HEADER_LOCATION = "header/header_V6-feature-

public static String[] CLASSIFER_NAME = { "M5RULES", "M5P", "REPTree",
// "AdditiveRegression", "KStar",
"LinearRegression"
// , "MultiLayerPerceptron"
};
public static LinkedList<Classifier> CLASSIFIER_LIST = new LinkedList<Classi
Arrays.asList(new M5Rules(), new M5P(), new REPTree(),
// new AdditiveRegression(), new KStar(),
new LinearRegression()
// , new MultilayerPerceptron()
));
}// end class
```

```
package org.powertac.trialmodule;

/*14, June 2016*/
/*14 June 2016, written by Saiful Abu
 * Extracts only five features from the log files.
 *
 * Extracts data from all the customers. and put in this format
 * SimulationData
 * |
 * | --- SimulationFileName
 * | ------------AverageUsage
 * | -------------------customer1
 * | -------------------custumer2
 *       | ------------SlotBasedUsage
 *       | -------------------customer1 ...
 *
 *       Original file was Clustering_Data_Extractor
 *
 * */

import java.awt.Toolkit;
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;

import org.apache.commons.io.FileUtils;
import org.apache.log4j.Logger;
import org.joda.time.DateTime;
import org.powertac.common.ClearedTrade;
import org.powertac.common.CustomerInfo;
import org.powertac.common.TariffTransaction;
import org.powertac.common.TimeService;
import org.powertac.common.Timeslot;
import org.powertac.common.WeatherReport;
import org.powertac.common.enumerations.PowerType;
import org.powertac.common.msg.TimeslotUpdate;
import org.powertac.common.repo.BrokerRepo;
import org.powertac.common.repo.CustomerRepo;
import org.powertac.common.repo.OrderbookRepo;
import org.powertac.common.repo.TimeslotRepo;
import org.powertac.common.repo.WeatherReportRepo;
import org.powertac.common.spring.SpringApplicationContext;
import org.powertac.logtool.LogtoolContext;
import org.powertac.logtool.common.DomainObjectReader;
import org.powertac.logtool.common.NewObjectListener;
import org.powertac.logtool.example.MktPriceStats;
import org.powertac.logtool.ifc.Analyzer;
import org.powertac.trialmodule.CustomerUsageStorer.SlotRecord;

import tools.Settings;
```

```
/**
 * Logtool Analyzer that reads ClearedTrade instances as they arrive and buil
 * an array for each timeslot giving all the market clearings for that
 * timeslot,s indexed by leadtime. The output data file has one line/timeslot
 * formatted as<br>
 * [mwh price] [mwh price] ...<br>
 * Each line has 24 entries, assuming that each timeslot is open for trading
 * times.
 *
 * Usage: MktPriceStats state-log-filename output-data-filename
 *
 * @author John Collins
 */
public class ReducedFeatureExtractor extends LogtoolContext implements Analy

static int TEMP_TEST_TIME_SLOT = 0;
int START_FROM_SLOT = 600;
static String[] weekDay = { "mon", "tue", "wed", "thu", "fri", "sat", "sun"
static String[] hourDay = { "0", "1", "2", "3", "4", "5", "6", "7", "8",
"9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19",
"20", "21", "22", "23", };

// public static String[] customerName = { "SolarLeasing@1",
// "SunnyhillSolar2", "MedicalCenter@2", };
// CustomerUsageStorer custUsage = new CustomerUsageStorer(customerName[0],
// null);
static String logFileNames[] = { "", "01.powertac-sim-1",
"02.powertac-sim-2", "03.powertac-sim-3", "04.powertac-sim-4",
"05.powertac-sim-5", "06.powertac-sim-6", "07.powertac-sim-7",
"08.powertac-sim-8", "09.powertac-sim-9", "10.powertac-sim-10",
"11.powertac-sim-11", "12.powertac-sim-12", "13.powertac-sim-13",
"14.powertac-sim-14", "15.powertac-sim-15", "16.powertac-sim-16",
"17.powertac-sim-17", "18.powertac-sim-18", "19.powertac-sim-19",
"20.powertac-sim-20", "21.powertac-sim-21", "22.powertac-sim-22",
"23.powertac-sim-23", "24.powertac-sim-24", "25.powertac-sim-25",
"26.powertac-sim-26-my-sim", "27.powertac-sim-27-my-sim",

};
static int fileIndex = 10;
static String logFileName = logFileNames[fileIndex];
static private Logger log = Logger.getLogger(MktPriceStats.class.getName());

// service references
private static TimeslotRepo timeslotRepo;
private TimeService timeService;
private WeatherReportRepo weatherReportRepo;
private OrderbookRepo OrderbookRepo;
private DomainObjectReader dor;
private Timeslot timeslot;
int counter = 0;

private static CustomerRepo customerRepo;

// Data
private TreeMap<Integer, ClearedTrade[]> data;
private TreeMap<Integer, CustomMarketTransaction> marketData;
private int ignoreInitial = 0; // timeslots to ignore at the beginning
private int ignoreCount = 0;
private int indexOffset = 0; // should be
// Competition.deactivateTimeslotsAhead - 1

// private PrintWriter output = null;
private String dataFilename = "clearedTrades.data";
public double brokerID;
private BrokerRepo brokerRepo;

// saif
int totalSlot = 360;
int previousTimeSlot = 0;
int totalTariffTransaction = 0;
Map<Integer, WeatherReport> weatherData;
// only extract from customers of this type
static PowerType[] powerType = { PowerType.CONSUMPTION };
static HashMap<String, CustomerInformationHolder> customerInformationHolder;

// /////////
int MAXIMUM_PREVIOUS_WEEK_OFFSET = 6;

// PowerUsageRepo powerUsageRepo;

/**
 * Main method just creates an instance and passes command-line args to its
 * inherited cli() method.
 */
public static void endProgramExecution() {

// close all file headers
for (String customerName : customerInformationHolder.keySet()) {
customerInformationHolder.get(customerName).output.close();
}// end for
// any additional file writing goes here
// write average file

for (String customerName : customerInformationHolder.keySet()) {
// constitue average usage string
// game#, powerType, customer name
String headerInfo = "\""
+ logFileName
+ "\""
+ ","
+ "\""
+ customerInformationHolder.get(customerName).customerInfo
.getPowerType().toString() + "\"" + "," + "\""
+ customerName + "\"";
String averageUsage = headerInfo;
String zScoreUsage = headerInfo;
for (int slot = 0; slot < 24 * 7; slot++) {
double avgUsageValue = customerInformationHolder
.get(customerName).custUsage.weekSlot[slot].stats
.getMean();
double zScoreValue = customerInformationHolder
.get(customerName).custUsage.getZScore(slot / 24 + 1,
slot % 24);
if (averageUsage.isEmpty()) {
averageUsage += avgUsageValue;
} else {
averageUsage = averageUsage + "," + avgUsageValue;
}

if (zScoreUsage.isEmpty()) {
zScoreUsage += zScoreValue;
} else {
zScoreUsage = zScoreUsage + "," + zScoreValue;
}
}// end for
// write the average usage to file
CustomerInfo customerInfo = customerInformationHolder
.get(customerName).customerInfo;
writeOverAllStats(customerInfo, averageUsage, "avg");
writeOverAllStats(customerInfo, zScoreUsage, "zscr");

}// end for
System.out.println("... Data Extraction finished from " + logFileName);
}// end of method

public static void writeOverAllStats(CustomerInfo customerInfo,
String overAllStatsString, String extension) {
String customerName = customerInfo.getName();
```

```
// create file now

String outputFileLocation = Settings.LOG_EXTRACTED_ROOT_FOLDER_NAME
+ "/" + logFileName + "/" + Settings.OVERALL_STATS_FOLDER_NAME
+ "/" + customerInfo.getPowerType().toString() + "/"
+ extension + "/" + customerName + "--game" + fileIndex + "."
+ extension;
try {
File file = new File(outputFileLocation);
if (file.getParentFile().exists() == false) {
file.getParentFile().mkdirs();
}// end if
PrintWriter output = new PrintWriter(new File(outputFileLocation));
output.write(overAllStatsString + "\n");
output.flush();
output.close();
// Files.createFile(Paths.get(outputFileLocation));
} catch (Exception e) {
// TODO Auto-generated catch block
System.out.println("could not create the file "
+ outputFileLocation);
}// end try catch
}// end of method

public static void main(String[] args) {
System.out.println("Data Extracting from log file " + logFileName
+ " ...");
new ReducedFeatureExtractor().cli(args);
```

```
endProgramExecution();
Toolkit.getDefaultToolkit().beep();
}// end of main

public static int[] converDayHour(int timeSlot) {

DateTime dt = timeslotRepo.getDateTimeForIndex(timeSlot);
int dayOfWeek = dt.getDayOfWeek();
int hourOfDay = dt.getHourOfDay();
int[] retVal = { dayOfWeek, hourOfDay };
return retVal;
}// end of method

/**
 * Takes two args, input filename and output filename
 *
 *
 */
private void cli(String[] args) {

// create folder by name
// SimulationData <SimulationFileName> [AverageUsage SlotBasedUsage]
if (Files.exists(Paths.get(Settings.LOG_EXTRACTED_ROOT_FOLDER_NAME)) == fals
try {
Files.createDirectory(Paths
.get(Settings.LOG_EXTRACTED_ROOT_FOLDER_NAME));
} catch (IOException e) {
// TODO Auto-generated catch block
System.out.println("Could not create root folder");
e.printStackTrace();
}// end try catch
}// end if

try {
if (Files.exists(Paths.get(Settings.LOG_EXTRACTED_ROOT_FOLDER_NAME
+ "/" + logFileName))) {
FileUtils.forceDelete(new File(
Settings.LOG_EXTRACTED_ROOT_FOLDER_NAME + "/"
+ logFileName));
}
FileUtils
.forceMkdir(new File(
Settings.LOG_EXTRACTED_ROOT_FOLDER_NAME + "/"
+ logFileName));
FileUtils.forceMkdir(new File(
Settings.LOG_EXTRACTED_ROOT_FOLDER_NAME + "/" + logFileName
+ "/" + Settings.OVERALL_STATS_FOLDER_NAME));
FileUtils.forceMkdir(new File(
Settings.LOG_EXTRACTED_ROOT_FOLDER_NAME + "/" + logFileName
+ "/" + Settings.slotBasedFolderName));
// Files.createDirectory(Paths.get(rootFolder + "/" + logFileName));
// Files.createDirectory(Paths.get(rootFolder + "/" + logFileName
// + "/" + avgFolderName));
// Files.createDirectory(Paths.get(rootFolder + "/" + logFileName
// + "/" + slotBasedFolderName));
} catch (IOException e) {
```

```
// TODO Auto-generated catch block
System.out.println("Could not create avg files folder");
e.printStackTrace();
}// end try catch

String logFolder = "log";

String logFileExtension = "state";
String logFileLocation = logFolder + "/" + logFileName + "."
+ logFileExtension;
super.cli(logFileLocation, this);
// csvWriter.closeWriter();
}// end of method

/*
 * (non-Javadoc)
 *
 * @see org.powertac.logtool.ifc.Analyzer#setup()
 */
@Override
public void setup() {
dor = (DomainObjectReader) SpringApplicationContext.getBean("reader");
timeslotRepo = (TimeslotRepo) getBean("timeslotRepo");
// weatherReportRepo = (WeatherReportRepo) getBean("WeatherReportRepo");
timeService = (TimeService) getBean("timeService");
brokerRepo = (BrokerRepo) SpringApplicationContext
.getBean("brokerRepo");
// registerNewObjectListener(new BrokerHandler(), Broker.class);
```

```
registerNewObjectListener(new TimeslotUpdateHandler(),
TimeslotUpdate.class);
registerNewObjectListener(new TariffTransactionHandler(),
TariffTransaction.class);
// registerNewObjectListener(new MarketTransactionHandler(),
// MarketTransaction.class);
// registerNewObjectListener(new BalancingTransactionHandler(),
// BalancingTransaction.class);
// registerNewObjectListener(new TimeslotHandler(), Timeslot.class);
registerNewObjectListener(new WeatherReportHandler(),
WeatherReport.class);
// registerNewObjectListener(new OrderbookHandler(), Orderbook.class);

ignoreCount = ignoreInitial;
data = new TreeMap<Integer, ClearedTrade[]>();
// saif
weatherData = new HashMap<Integer, WeatherReport>();
// powerUsageRepo = new PowerUsageRepo();

// saif
marketData = new TreeMap<Integer, CustomMarketTransaction>();
customerInformationHolder = new HashMap<String, CustomerInformationHolder>()
// ///////////////////
}

/*
 * (non-Javadoc)
 *
 * @see org.powertac.logtool.ifc.Analyzer#report()
 */
@Override
public void report() {
boolean ret;
ret = true;
if (ret) {
// for now wont use the report method
return;
}// end if

for (Map.Entry<Integer, CustomMarketTransaction> entry : marketData
.entrySet()) {
String delim = "";
Integer timeslot = entry.getKey();
CustomMarketTransaction trades = entry.getValue();

}

}

/*********** Handlers Begin ********/
// ----------------------------------------
// catch TimeslotUpdate events
class TimeslotUpdateHandler implements NewObjectListener {

@Override
```

```
public void handleNewObject(Object thing) {

int currentTimeSlot = timeslotRepo.currentSerialNumber();

// update previous slot
int prevSlotBefore = currentTimeSlot - 1;
int[] dayHour = converDayHour(prevSlotBefore);

int slotOfInterest = prevSlotBefore;// currentTimeSlot - 8; // - 2;
if (slotOfInterest % 100 == 0) {
System.out.println("Processing Time Slot " + slotOfInterest
+ " ...");
}// end if
if (slotOfInterest > START_FROM_SLOT) {
for (String customerName : customerInformationHolder.keySet()) {
processSlot(
slotOfInterest,
customerInformationHolder.get(customerName).custUsage,
customerInformationHolder.get(customerName).output);
}
}// end if
// System.out.println("will write to file for slto " +
// slotOfInterest);

if (prevSlotBefore >= 360) {
// Update record for all the customers
for (String customerName : customerInformationHolder.keySet()) {
customerInformationHolder.get(customerName).custUsage
.updateRecord(dayHour[0], dayHour[1]);
}// end for
}// end if
}// end of method

/*
 * Forms supervised learning example for a timeSlotIndex.
 * [temporalString] [temperature of 5 slots starting from the slot of
 * interest] [previous week's usage 5 features centered at the same slot
 * as the slot of interest] [actual usage]
 */
void processSlot(int timeSlotIndex, CustomerUsageStorer custUsage,
PrintWriter output) {
String instance = "";
DateTime dt = timeslotRepo.getDateTimeForIndex(timeSlotIndex);
String dateTime = "\"";
dateTime += dt.getYear() + "-" + dt.getMonthOfYear() + "-"
+ dt.getDayOfMonth() + " " + dt.getHourOfDay() + "\"";

int dayOfWeek = converDayHour(timeSlotIndex)[0];
int hourOfDay = converDayHour(timeSlotIndex)[1];
int slotOfWeek = (dayOfWeek - 1) * 24 + hourOfDay;
String day = weekDay[dayOfWeek - 1];
String hour = hourDay[hourOfDay];
SlotRecord record = custUsage.getRecord(dayOfWeek, hourOfDay);

double totalUsage = record.completeUsage;
```

```
// excluding time related feats
// instance += dateTime + "," + slotOfWeek + "," + day + "," + hour
// + ",";

// weather instance
String weatherString = formWeatherString(weatherData,
timeSlotIndex, 1);
instance += weatherString;

String zScoreString = "";
int totalZScores = 24;
int startSlot = (int) ((dayOfWeek − 1) ∗ 24 + hourOfDay);
for (int i = 0; i < totalZScores; i++) {
int currentSlot = (168 + startSlot − i) % 168;
int dayOfWeekTemp = currentSlot / 24 + 1;
int hourOfDayTemp = currentSlot % 24;
double zScore = custUsage.getZScore(dayOfWeekTemp,
hourOfDayTemp);
zScoreString += zScore + ",";
}// end for
// excluding zscore related feats
// instance += zScoreString;
double meanOfSlot = custUsage.getMean(dayOfWeek, hourOfDay);
double stdDevOfSlot = custUsage.getStdDev(dayOfWeek, hourOfDay);
double overAllMean = custUsage.getOverAllMean();
double overAllStdDev = custUsage.getOverAllStdDev();
double overAllZ = custUsage.overAllZ();
// excluding statis related feats
// instance += meanOfSlot + "," + stdDevOfSlot + "," + overAllMean
// + "," + overAllStdDev + "," + overAllZ + ",";
instance += meanOfSlot + "," + stdDevOfSlot + ",";
double avgUsage = 0;
if (record.completePopulation > 0) {
avgUsage = totalUsage / record.completePopulation;
}// end if
// instance += avgUsage + ",";
instance += avgUsage;
double recentUsageZScore = custUsage.getZScoreRecentUsage(
dayOfWeek, hourOfDay);
// exluding recent z score
// instance += recentUsageZScore;

writeALine(instance, output);
if (timeSlotIndex == 601) {
System.out.println(instance);
}

}// end of method

void writeALine(String string, PrintWriter output) {
output.write(string + "\n");
output.flush();
}// end of method

/*
 * weatherString = <temp-0><temp-1>...<temp-(n-1)> temp-(i) means
 * temperature of t-i slots temperature where t is the current slot.
 *
 * @param weatherData = list of weather report for each slot i
 *
 * @param currentTimeSlot = the slot we are predicting for
 *
 * @param totalReports = number of reports we want to incorporate in the
 * training example
 */
String formWeatherString(Map<Integer, WeatherReport> weatherData,
int currentTimeSlot, int totalReports) {
String weatherReportString = "";
for (int i = 0; i < totalReports; i++) {
int timeSlot = currentTimeSlot - i;
WeatherReport weatherReport = weatherData.get(timeSlot);
double cloudCover = weatherReport.getCloudCover();
double temperature = weatherReport.getTemperature();
double windspeed = weatherReport.getWindSpeed();
weatherReportString = weatherReportString + cloudCover + ","
+ temperature + "," + windspeed + ",";
}// end for
return weatherReportString;
}// end of method

/*
 * forms a string of the following data: <consumption slot 0>
 * <consumption slot -1> .... <consumption slot - (totalslots-1)>
 */
```

```
String formPreviousAvgUsageString(
PowerUsageRepoForOfficeComplex powerUsageRepo,
int startingIndex, int totalSlots) {
String usageStr = "";
for (int i = 0; i < totalSlots; i++) {
int slot = startingIndex - i;
// double averageUsage = powerUsageRepo.getUsage(slot) /
// powerUsageRepo.getPopulation(slot);
double averageUsage = powerUsageRepo.getAvgUsage(slot);
usageStr = usageStr + averageUsage + ",";
}// end for
return usageStr;
}// end of method

}// end of class

class WeatherReportHandler implements NewObjectListener {

@Override
public void handleNewObject(Object thing) {

WeatherReport wr = (WeatherReport) thing;

// System.out.println("In the weather report handler");

int timeSlotIndex = wr.getTimeslotIndex();
// System.out.println("Putting weather " + timeSlotIndex);
weatherData.put(timeSlotIndex, wr);
```

```
}// end of method
}// end of class

class TariffTransactionHandler implements NewObjectListener {

@Override
public void handleNewObject(Object thing) {

TariffTransaction tariffTransaction = (TariffTransaction) thing;
DateTime dt = timeslotRepo.getDateTimeForIndex(tariffTransaction
.getPostedTimeslotIndex());
int dayOfWeek = dt.getDayOfWeek();
int hourOfDay = dt.getHourOfDay();
int month = dt.getMonthOfYear();
int dayOfMonth = dt.getDayOfMonth();

CustomerInfo customerInfo = tariffTransaction.getCustomerInfo();
if (customerInfo == null) {
// System.out.println("customer info is null");
return;
}// end if

// boolean containsValidType = false;
// for (PowerType pt : powerType) {
// if (customerInfo.getPowerType() == pt) {
// containsValidType = true;
// }// end if
// }// end for
// if (containsValidType == false) {
// return;
// }// end if

double usageKwh = Math.abs(tariffTransaction.getKWh());
int userCount = tariffTransaction.getCustomerCount();

if (tariffTransaction.getPostedTimeslotIndex() == 420) {
System.out.println("at 420 slot for customer "
+ tariffTransaction.getCustomerInfo().getName()
+ " population " + tariffTransaction.getCustomerCount()
+ " broker name "
+ tariffTransaction.getBroker().getUsername());
}// end if
String customerName = tariffTransaction.getCustomerInfo().getName();
int reportOfSlot = tariffTransaction.getPostedTimeslotIndex();
if (customerInformationHolder.containsKey(customerName) == false) {
// create file now
String outputFileLocation = Settings.LOG_EXTRACTED_ROOT_FOLDER_NAME
+ "/"
+ logFileName
+ "/"
+ Settings.slotBasedFolderName
+ "/"
+ customerInfo.getPowerType().toString()
+ "/"
+ customerName + "--game" + fileIndex + ".extracted";
```

```
try {
File file = new File(outputFileLocation);
if (file.getParentFile().exists() == false) {
file.getParentFile().mkdirs();
}
// Files.createFile(Paths.get(outputFileLocation));
} catch (Exception e) {
// TODO Auto-generated catch block
System.out.println("could not create the file "
+ outputFileLocation);
}
customerInformationHolder.put(customerName,
new CustomerInformationHolder(customerName,
outputFileLocation, customerInfo));
}
CustomerUsageStorer custUsage = customerInformationHolder
.get(customerName).custUsage;
customerInformationHolder.get(customerName).custUsage.addUsage(
dayOfWeek, hourOfDay, month, dayOfMonth, usageKwh,
userCount, reportOfSlot);

}// end of method
}// end of class

/********************** Handler End *****************************/

class CustomMarketTransaction {
int timeslotIndex;
```

```
double boughtMWh;
double soldMWh;
double mWh;
double price;
double boughtprice;
double soldprice;
int count;
double balancingTrans;
int day;
int hour;
double temp;
double cloudCoverage;
double windSpeed;
double windDirection;
double clearingPrice;
double[] clearings = new double[8720];

CustomMarketTransaction() {
timeslotIndex = 0;
boughtMWh = 0.0;
soldMWh = 0.0;
boughtprice = 0.0;
soldprice = 0.0;
mWh = 0.0;
price = 0.0;
count = 0;
balancingTrans = 0.0;
day = 0;
hour = 0;
temp = 0.0;
cloudCoverage = 0.0;
windSpeed = 0.0;
windDirection = 0.0;
clearingPrice = 0.0;
}
}
}

package tools;


/*
 * Created by Saiful Abu on May 2016.
 * Run several phases.
 *  Phase 0 create training instances from over all -->
 do clustering manually using weka on this data
 phase 1 separates slot based training instances
 based on the cluster
 phase 2: manually check if there exists any outlier in the training set.
 then run phase 2 to combines the slot
 based files stored in cluster folders. This phase will figure out the
 best performing classifier for each cluster
 phase 3 --- separate customer data in separate folder and combine them
 phase 4 --- after checking manually the aggregated data, make prediction
 models that makes best
 prediction for that customer.
 RUN MovingAvgPerformanceLogger.java AFTER ALL THE PHASE ARE COMPLETE
 *
 * */
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.FilenameFilter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;

import org.apache.commons.io.FileUtils;

import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.functions.LinearRegression;
import weka.classifiers.functions.MultilayerPerceptron;
import weka.classifiers.lazy.KStar;
import weka.classifiers.meta.AdditiveRegression;
import weka.classifiers.rules.M5Rules;
import weka.classifiers.trees.M5P;
import weka.classifiers.trees.REPTree;
import weka.clusterers.SimpleKMeans;
import weka.core.Instances;
import weka.core.converters.ArffSaver;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.Remove;

public class FileCombiner_ClusterVersion {
static int PHASE = 4; // Phase 0 create training instances from over all -->
// do clustering manually using weka on this data
// phase 1 separates slot based training instances
// based on the cluster
// phase 2: manually check if there exists any outlier in the training set.
// then run phase 2 to combines the slot
// based files stored in cluster folders. This phase will figure out the
// best performing classifier for each cluster
// phase 3 --- separate customer data in separate folder and combine them
// phase 4 --- after checking manually the aggregated data, make prediction
// models that makes best
// prediction for that customer.
// RUN MovingAvgPerformanceLogger.java AFTER ALL THE PHASE ARE COMPLETE
static String OUT_FILE_NAME_START_PORTION = "training_set_"
+ Settings.CUSTOMER_NAME;
```

```
// static String OUT_FILE_NAME_START_PORTION =
// "error_office_complex1_ns_base_";

public static void main(String[] args) {
// combine();
if (PHASE == 0) {
phase0();
} else if (PHASE == 1) {
phase1();
} else if (PHASE == 2) {
phase2();
} else if (PHASE == 3) {
phase3();
} else if (PHASE == 4) {
phase4();
}
}// end of main

public static void phase0() {
String tempFolderName = "temp";
if (Files.exists(Paths.get(Settings.TRAINING_ROOT_FOLDER_NAME)) == false) {
try {
Files.createDirectory(Paths
.get(Settings.TRAINING_ROOT_FOLDER_NAME));
} catch (IOException e) {
// TODO Auto-generated catch block
System.out.println("Could not create root folder");
e.printStackTrace();
}// end try catch
}// end if

if (Files.exists(Paths.get(Settings.TRAINING_ROOT_FOLDER_NAME + "/"
+ tempFolderName))) {
try {
FileUtils.forceDelete(new File(
Settings.TRAINING_ROOT_FOLDER_NAME + "/"
+ tempFolderName));

} catch (IOException e) {
// TODO Auto-generated catch block
System.out.println("Could not delete training folder");
e.printStackTrace();
}// end try catch
}// end if

try {
Files.createDirectory(Paths.get(Settings.TRAINING_ROOT_FOLDER_NAME
+ "/" + tempFolderName));
} catch (IOException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
System.out.println("could not create temp folder");
}
File[] files = new File(Settings.LOG_EXTRACTED_ROOT_FOLDER_NAME)
.listFiles();
```

```
for (File file : files) {
// go inside overall stats folder

File[] foldersInsideOverAllStats = new File(file.toString() + "/"
+ Settings.OVERALL_STATS_FOLDER_NAME).listFiles();
for (File fileOverAllStats : foldersInsideOverAllStats) {
String powerTypeFolderName = fileOverAllStats.getName();
if (Files.exists(Paths.get(Settings.TRAINING_ROOT_FOLDER_NAME
+ "/" + tempFolderName + "/" + powerTypeFolderName)) == false) {
try {
Files.createDirectory(Paths
.get(Settings.TRAINING_ROOT_FOLDER_NAME + "/"
+ tempFolderName + "/"
+ powerTypeFolderName));
} catch (IOException e) {
// TODO Auto-generated catch block
System.out
.println("Could not create power type folder inside temp folder");
e.printStackTrace();
}// end try catch
}// end if
String destinationFolder = Settings.TRAINING_ROOT_FOLDER_NAME
+ "/" + tempFolderName + "/" + powerTypeFolderName;
// String sourceFolder = fileOverAllStats.toString() + "/zscr";
String sourceFolder = fileOverAllStats.toString() + "/avg";
copyFromDirectoryToDirectory(sourceFolder, destinationFolder);
}// end for
}// end for
```

```
// upto this point all the files are in temp folder based on the power
// type
String overAllStatsTrainingFolder = Settings.TRAINING_ROOT_FOLDER_NAME
+ "/" + Settings.OVERALL_STATS_FOLDER_NAME;
// create clustered folder
String clusterFolder = Settings.TRAINING_ROOT_FOLDER_NAME + "/"
+ Settings.CLUSTER_FOLDER_NAME;

ifExistDeleteThenCreate(overAllStatsTrainingFolder);
ifExistDeleteThenCreate(clusterFolder);

files = new File(Settings.TRAINING_ROOT_FOLDER_NAME + "/"
+ tempFolderName).listFiles();
for (File file : files) {
String powerType = file.getName();
ifExistDeleteThenCreate(overAllStatsTrainingFolder + "/"
+ powerType);
// create powertype folders in cluster directory
ifExistDeleteThenCreate(clusterFolder + "/" + powerType);
combine(file.toString(), overAllStatsTrainingFolder + "/"
+ powerType, Settings.OVERALL_STATS_HEADER_LOCATION);
}// end for

deleteIfExists(Settings.TRAINING_ROOT_FOLDER_NAME + "/"
+ tempFolderName);

}// end of method
```

```
/*
 * loads the cluster model files. separates training instances of slot based
 * files based on the cluster assignments.
 */
public static void phase1() {
System.out.println("running phase 1");
String clusterFolder = Settings.TRAINING_ROOT_FOLDER_NAME + "/"
+ Settings.CLUSTER_FOLDER_NAME;
File[] files = new File(clusterFolder).listFiles();
ifExistDeleteThenCreate(Settings.TRAINING_ROOT_FOLDER_NAME + "/"
+ Settings.SLOT_BASED_TRAINING_SET_FOLDER_NAME);
ifExistDeleteThenCreate(Settings.TRAINING_ROOT_FOLDER_NAME + "/"
+ Settings.CLUSTER_PREDICTOR_FOLDER_NAME);

for (File cluster : files) {
String clusterType = cluster.getName();
String slotBasedTrainingInstanceFolder = Settings.TRAINING_ROOT_FOLDER_NAME
+ "/"
+ Settings.SLOT_BASED_TRAINING_SET_FOLDER_NAME
+ "/"
+ clusterType;
String predictorStorerFolder = Settings.TRAINING_ROOT_FOLDER_NAME
+ "/" + Settings.CLUSTER_PREDICTOR_FOLDER_NAME + "/"
+ clusterType;
ifExistDeleteThenCreate(slotBasedTrainingInstanceFolder);
ifExistDeleteThenCreate(predictorStorerFolder);
for (File file : cluster.listFiles()) {
```

```
if (file == null) {
continue;
}
if (file.listFiles().length == 0) {
continue; // don't do anything if the folder does not have a
// cluster model in it
}// end if
String powerType = file.getName();
// System.out.println("has cluster " + powerType);
// create folder for the training instance with appropriate
// power
// type
ifExistDeleteThenCreate(slotBasedTrainingInstanceFolder + "/"
+ powerType);
// creating predictor storage folder
ifExistDeleteThenCreate(predictorStorerFolder + "/" + powerType);

// read corresponding .inst file and make a map that points to
// instance to file location
// TrainingData\OverallStats\CONSUMPTION\CONSUMPTION.inst
String instFileLocation = "TrainingData/OverallStats/"
+ powerType + "/" + powerType + ".inst";
HashMap<Integer, String> instanceToFileLocationMap = new HashMap<Integer, St

FileInputStream fis;
try {
fis = new FileInputStream(new File(instFileLocation));
// Construct BufferedReader from InputStreamReader
// BufferedReader
BufferedReader br = new BufferedReader(
new InputStreamReader(fis));

String line = null;
while ((line = br.readLine()) != null) {
// System.out.println(line);
String[] parts = line.split(",");
int instanceNumber = Integer.parseInt(parts[0]);
String logFolderName = parts[1].substring(1,
parts[1].length() - 1);
String customerName = parts[3].substring(1,
parts[3].length() - 1);
int simulationFileNumber = Integer
.parseInt(logFolderName.substring(0, 2));
String arffFileName = customerName + "--game"
+ simulationFileNumber + ".extracted";
String logFileLocation = Settings.LOG_EXTRACTED_ROOT_FOLDER_NAME
+ "/" + logFolderName + "/"

+ Settings.slotBasedFolderName + "/"

+ powerType + "/" + arffFileName;
instanceToFileLocationMap.put(instanceNumber,
logFileLocation);
}// end of while

br.close();
```

```
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
System.out.println("could not read inst file");
System.exit(0);
}// end of try catch
// hashmap is ready for use now: it has instance —> file
// name
// in it

// now load the cluster file
// TrainingData\Clusters\CONSUMPTION\CONSUMPTION.model
String clusterPath = Settings.TRAINING_ROOT_FOLDER_NAME + "/"
+ Settings.CLUSTER_FOLDER_NAME + "/"
+ cluster.getName() + "/" + powerType + "/" + powerType
+ ".model";
SimpleKMeans model = null;
try {
// Vector v = (Vector) SerializationHelper.read(MODEL_PATH);
// model = (Classifier) v.get(0);
// Instances header = (Instances) v.get(1);
model = (SimpleKMeans) weka.core.SerializationHelper
.read(clusterPath);
// System.out.println("loaded " + clusterPath);

int clusterCounts = model.getNumClusters();
int[] assignments = model.getAssignments();
```

```
// create cluster folder in each powertype folders
for (int i = 0; i < clusterCounts; i++) {
ifExistDeleteThenCreate(slotBasedTrainingInstanceFolder
+ "/" + powerType + "/" + "cluster-" + i);
ifExistDeleteThenCreate(predictorStorerFolder + "/"
+ powerType + "/" + "cluster-" + i);
}// end for

// now copy slot based files in proper cluster folders
for (int instIndex = 0; instIndex < assignments.length; instIndex++) {
int assignedCluster = assignments[instIndex];
String sourceFileLocation = instanceToFileLocationMap
.get(instIndex);
String destFolderLocation = slotBasedTrainingInstanceFolder
+ "/"
+ powerType
+ "/"
+ "cluster-"
+ assignedCluster;
// System.out.println(sourceFile + " --- " + destFile);
copyFileToDirectory(sourceFileLocation,
destFolderLocation);
}// end for
} catch (Exception e) {
// TODO Auto-generated catch block
System.out.println("Error Message: " + e.getMessage());
System.exit(0);
}// end of try catch
}// end for
}// end for
}// end of method phase 1

/*
 * combines the slot based information files stored in different cluster
 * folders.
 */
public static void phase2() {

String slotBasedFolderName = combineFileNamesToMakePathLocation(
Settings.TRAINING_ROOT_FOLDER_NAME,
Settings.SLOT_BASED_TRAINING_SET_FOLDER_NAME);
for (File clusterType : new File(slotBasedFolderName).listFiles()) {
String clusterTypeName = clusterType.getName();
System.out.println("******************************");
System.out.println("Cluster Type " + clusterTypeName);
for (File powerTypeDir : clusterType.listFiles()) {

String powerType = powerTypeDir.getName();
for (File clusterDir : new File(
combineFileNamesToMakePathLocation(slotBasedFolderName,
clusterTypeName, powerType)).listFiles()) {
String clusterName = clusterDir.getName();

System.out.println(powerType + " " + clusterName);
String outputFilePath = combineFileNamesToMakePathLocation(
clusterDir.getAbsolutePath(), "Combined");
deleteIfExists(outputFilePath);
combine(clusterDir.getAbsolutePath(), outputFilePath,
Settings.SLOT_BASED_HEADER_LOCATION);

String predictorFile = combineFileNamesToMakePathLocation(
Settings.TRAINING_ROOT_FOLDER_NAME,
Settings.CLUSTER_PREDICTOR_FOLDER_NAME,
clusterTypeName, powerType, clusterName,
"bestPredictor.model");
String arffFileLocation = new File(outputFilePath)
.listFiles(new FilenameFilter() {
public boolean accept(File dir, String name) {
return name.toLowerCase().endsWith(".arff");
}
})[0].getAbsolutePath();
// System.out.println(predictorFile);
makeBestClassifierForEachCluster(arffFileLocation,
predictorFile);
}// end for
}// end for
System.out.println("*****************************");
}// end for

}// end of method phase 2

public static void makeBestClassifierForEachCluster(
String instanceFileLocation, String outputFileLocation) {
```

```
// System.out.println("Examining " + instanceFileLocation);
try {
BufferedReader reader = null;
reader = new BufferedReader(new FileReader(instanceFileLocation));
Instances inst = null;
inst = new Instances(reader);
inst.setClassIndex(inst.numAttributes() - 1);
// code for reduction from big feat to reduced feat

// Instances instNew;
// Remove remove;
//
// remove = new Remove();
// remove.setAttributeIndices("5,6,7,32,33,37");
// remove.setInvertSelection(new Boolean(true));
// remove.setInputFormat(inst);
// instNew = Filter.useFilter(inst, remove);
//
// instNew.setClassIndex(instNew.numAttributes() - 1);
//
// ArffSaver saver = new ArffSaver();
// saver.setInstances(instNew);
//
// deleteIfExists(Paths.get(instanceFileLocation + "--reduced.arff")
// .toString());
// saver.setFile(new File(instanceFileLocation + "--reduced.arff"));
// saver.writeBatch();
// end of red feat set
// test classifier creation from the instance set
// we are going to use 7 classifiers and choosing the best among
// them
Classifier bestClassifier = null;
double minRelativePercentError = Double.POSITIVE_INFINITY;
double minMeanAbsError = Double.POSITIVE_INFINITY;

LinkedList<Classifier> classifierList = (LinkedList<Classifier>) Settings.CL
.clone();

int i = 0;
int bestClassiferINdex = -1;
for (Classifier classifier : classifierList) {

// classifier.buildClassifier(instNew);
// Evaluation eval = new Evaluation(instNew);
// eval.crossValidateModel(classifier, instNew, 4, new
// Random(1));
classifier.buildClassifier(inst);
Evaluation eval = new Evaluation(inst);
eval.crossValidateModel(classifier, inst, 4, new Random(1));
double relativeAbsError = eval.relativeAbsoluteError();
double meanAbsError = eval.meanAbsoluteError();
System.out.println("Making " + Settings.CLASSIFER_NAME[i]);
if (relativeAbsError < minRelativePercentError) {
bestClassiferINdex = i;
bestClassifier = classifier;
minRelativePercentError = relativeAbsError;
minMeanAbsError = meanAbsError;
}// end if else
System.out.println(Settings.CLASSIFER_NAME[i]
+ " relative error : " + relativeAbsError
+ " abs error : " + meanAbsError);
i++;

}// end for

// System.out.println("For customer " + instanceFileLocation
// + " The best classifier "
// + Settings.CLASSIFER_NAME[bestClassiferINdex]
// + " with error " + minRelativePercentError);

// write down the classifer model to file
weka.core.SerializationHelper.write(outputFileLocation,
bestClassifier);
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}// end of try catch

}// end of method

// combines information of customers with same name together and then
// creates the best
// predictor model out of it.
```

```
public static void phase3() {
String temporaryFolder = combineFileNamesToMakePathLocation(
Settings.TRAINING_ROOT_FOLDER_NAME, "TempIndiv");
String indivFolder = combineFileNamesToMakePathLocation(
Settings.TRAINING_ROOT_FOLDER_NAME,
Settings.INDIVIDUAL_PREDICTOR_FOLDER_NAME);
ifExistDeleteThenCreate(temporaryFolder);
ifExistDeleteThenCreate(indivFolder);

HashMap<String, Boolean> customerNameMap = new HashMap<String, Boolean>();
File[] files = new File(Settings.LOG_EXTRACTED_ROOT_FOLDER_NAME)
.listFiles();
for (File file : files) {
// go inside slotbased usage folder
File[] consumptionFolder = new File(file.toString() + "/"
+ Settings.slotBasedFolderName + "/" + "CONSUMPTION")
.listFiles();
for (File customerLog : consumptionFolder) {
String customerName = customerLog.getName().substring(0,
customerLog.getName().indexOf("--"));
customerNameMap.put(customerName, true);
// System.out.println(customerName);
if (Files.exists(Paths
.get(temporaryFolder + "/" + customerName)) == false) {
try {
Files.createDirectory(Paths.get(temporaryFolder + "/"
+ customerName));
} catch (IOException e) {
```

```
// TODO Auto-generated catch block
System.out
.println("Could not create separate folders for individual customers");
e.printStackTrace();
}// end try catch
}// end if
String destinationFolder = temporaryFolder + "/" + customerName;
// System.out.println(destinationFolder);
String sourceFile = customerLog.toString();
copyFileToDirectory(sourceFile, destinationFolder);
}// end for
}// end for
// at this point cusotmers of consumption type are copied to folders
// containing the customername
// now combine them to form training instance
File[] customerLogFolder = new File(temporaryFolder).listFiles();
for (File customerLog : customerLogFolder) {
String customerName = customerLog.getName();
combine(customerLog.getAbsolutePath(),
combineFileNamesToMakePathLocation(indivFolder,
customerName), customerName,
Settings.SLOT_BASED_HEADER_LOCATION);
}// end for

// at this point we have all the customers data combined together
// now make best performing classifier for each customer type

}// end of method
```

```
// loads the arff file for each customer.
// ignores unneccsary attributes
// checks which prediciton model among 7 models has the least absolute error
// write that predictor to file
public static void phase4() {
// go to each customer folder and load the arff file
String individualPredictorFolder = combineFileNamesToMakePathLocation(
Settings.TRAINING_ROOT_FOLDER_NAME,
Settings.INDIVIDUAL_PREDICTOR_FOLDER_NAME);
File[] indivCustomerFolder = new File(individualPredictorFolder)
.listFiles();
for (File customerFolder : indivCustomerFolder) {
// System.out.println(customerFolder.getName());
// load the arff file
try {
BufferedReader reader = null;
reader = new BufferedReader(new FileReader(
customerFolder.getAbsolutePath() + "/"
+ customerFolder.getName() + ".arff"));
Instances inst = null;
inst = new Instances(reader);
inst.setClassIndex(inst.numAttributes() - 1);

// Instances instNew;
// Remove remove;
// remove = new Remove();
// remove.setAttributeIndices("5,6,7,32,33,37");
// remove.setInvertSelection(new Boolean(true));
// remove.setInputFormat(inst);
// instNew = Filter.useFilter(inst, remove);
// instNew.setClassIndex(instNew.numAttributes() − 1);
//
// ArffSaver saver = new ArffSaver();
// saver.setInstances(instNew);
//
// deleteIfExists(Paths.get(
// customerFolder.getAbsolutePath() + "/"
// + customerFolder.getName() + "−−reduced.arff")
// .toString());
// ;
// saver.setFile(new File(customerFolder.getAbsolutePath() + "/"
// + customerFolder.getName() + "−−reduced.arff"));
// saver.writeBatch();

// test classifier creation from the instance set
// we are going to use 7 classifiers and choosing the best among
// them
Classifier bestClassifier = null;
double minRelativePercentError = Double.POSITIVE_INFINITY;
double minMeanAbsError = Double.POSITIVE_INFINITY;
LinkedList<Classifier> classifierList = (LinkedList<Classifier>) Settings.CL
.clone();

int i = 0;
int bestClassiferINdex = −1;
```

```
for (Classifier classifier : classifierList) {
// classifier.buildClassifier(instNew);
// Evaluation eval = new Evaluation(instNew);
// eval.crossValidateModel(classifier, instNew, 10,
// new Random(1));
classifier.buildClassifier(inst);
Evaluation eval = new Evaluation(inst);
eval.crossValidateModel(classifier, inst,
Settings.NUM_FOLD_CROSS_VALIDATION, new Random(1));
double relativeAbsError = eval.relativeAbsoluteError();

if (relativeAbsError < minRelativePercentError) {
bestClassiferINdex = i;
bestClassifier = classifier;
minRelativePercentError = relativeAbsError;
minMeanAbsError = eval.meanAbsoluteError();
}// end if else
i++;
}// end for

System.out.println("For customer " + customerFolder.getName()
+ " The best classifier "
+ Settings.CLASSIFER_NAME[bestClassiferINdex]
+ " with error " + minRelativePercentError
+ " mean abs error " + minMeanAbsError);

// write down the classifer model to file
weka.core.SerializationHelper.write(
customerFolder.getAbsolutePath() + "/"
+ customerFolder.getName() + ".model",
bestClassifier);
} catch (Exception e) {
// TODO Auto-generated catch block
e.printStackTrace();
}// end of try catch
}
}// end of method

public static void testAttribDeletetion() throws Exception {
// for test make a classifier based reduced training set
BufferedReader reader = null;

reader = new BufferedReader(
new FileReader(
"TrainingData\\IndividualPredictors\\BrooksideHomes\\BrooksideHomes.arff"));

Instances inst = null;
inst = new Instances(reader);

Instances instNew;
Remove remove;

remove = new Remove();
remove.setAttributeIndices("5,6,7,32,33,37");
remove.setInvertSelection(new Boolean(true));
remove.setInputFormat(inst);
```

```
instNew = Filter.useFilter(inst, remove);

instNew.setClassIndex(instNew.numAttributes() - 1);

ArffSaver saver = new ArffSaver();
saver.setInstances(instNew);
deleteIfExists(Paths
.get("TrainingData\\IndividualPredictors\\BrooksideHomes\\BrooksideHomes-redu
.toString());
;
saver.setFile(new File(
"TrainingData\\IndividualPredictors\\BrooksideHomes\\BrooksideHomes-reduced-
saver.writeBatch();

// test classifier creation from the instance set
// we are going to use 7 classifiers and choosing the best among them
Classifier bestClassifier = null;
double minRelativePercentError = Double.POSITIVE_INFINITY;

LinkedList<Classifier> classifierList = new LinkedList<Classifier>();

String[] classifierName = { "M5RULES", "M5P", "REPTree",
"AdditiveRegression", "KStar", "LinearRegression",
"MultiLayerPerceptron" };
classifierList.add(new M5Rules());
classifierList.add(new M5P());
classifierList.add(new REPTree());
classifierList.add(new AdditiveRegression());
classifierList.add(new KStar());
classifierList.add(new LinearRegression());
classifierList.add(new MultilayerPerceptron());

int i = 0;
int bestClassiferINdex = -1;
for (Classifier classifier : classifierList) {
classifier.buildClassifier(instNew);
Evaluation eval = new Evaluation(instNew);
eval.crossValidateModel(classifier, instNew, 10, new Random(1));
double relativeAbsError = eval.relativeAbsoluteError();

if (relativeAbsError < minRelativePercentError) {
bestClassiferINdex = i;
bestClassifier = classifier;
minRelativePercentError = relativeAbsError;
}// end if else
i++;
}// end for

System.out.println("The best classifier "
+ classifierName[bestClassiferINdex] + " with error "
+ minRelativePercentError);

}// end of method

public static String combineFileNamesToMakePathLocation(String... strings) {
if (strings.length <= 0) {
```

```
return null;
}
;
String output = strings[0];
for (int i = 1; i < strings.length; i++) {
output += "/" + strings[i];
}// end for
return output;
}// end of method

public static void deleteIfExists(String filePath) {
if (Files.exists(Paths.get(filePath))) {
try {
FileUtils.forceDelete(new File(filePath));

} catch (IOException e) {
// TODO Auto-generated catch block
System.out.println("Could not delete folder " + filePath);
e.printStackTrace();
}// end try catch
}// end if
}

/*
 * source -- folder that contains files to be combined destination -- the
 * combined file name headerLocation -- location of the header file
 * generates a random name for the output file name
 */
```

```
public static void combine(String source, String destination,
String headerLocation) {
Path outputFilePath = Paths
.get(new File(getOutPutFileName(destination)).getAbsolutePath());

List<Path> inputs = new LinkedList<Path>();
inputs.add(Paths.get(new File(headerLocation).getAbsolutePath())); // add
// header
// file's
// location
// at
// the
// beginning
inputs.addAll(getPathofAllFilesFromFolder(source));
concatenateFiles(inputs, outputFilePath);

}// end of emethod

/*
 * source -- folder that contains files to be combined destination -- the
 * combined file name headerLocation -- location of the header file
 * generates a random name for the output file name outputFileName --
 * creates the outputfile in the destination Directory
 */
public static void combine(String source, String destination,
String outputFileName, String headerLocation) {
Path outputFilePath = Paths.get(new File(destination + "/"
+ outputFileName + ".arff").getAbsolutePath());
List<Path> inputs = new LinkedList<Path>();
inputs.add(Paths.get(new File(headerLocation).getAbsolutePath())); // add
// header
// file's
// location
// at
// the
// beginning
inputs.addAll(getPathofAllFilesFromFolder(source));
concatenateFiles(inputs, outputFilePath);

}// end of emethod

public static void ifExistDeleteThenCreate(String filePath) {

deleteIfExists(filePath);

try {
Files.createDirectory(Paths.get(filePath));
} catch (IOException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
System.out.println("could not create folder " + filePath);
}
}// end of method

// copies a file to a Directory
```

```
public static void copyFileToDirectory(String srcFileLocation,
String destinationDirectoryLocation) {
Path file = Paths.get(srcFileLocation);
Path to = Paths.get(destinationDirectoryLocation);
try {
Files.copy(file, to.resolve(file.getFileName()));
} catch (IOException e) {
// TODO Auto-generated catch block
System.out.println("could not copy file to a directory");
e.printStackTrace();
}
}// end of method copy file to directory

// copies content of one directory to antoher directory
public static void copyFromDirectoryToDirectory(String src,
String destination) {
File source = new File(src);
File dest = new File(destination);
try {
FileUtils.copyDirectory(source, dest);
} catch (IOException e) {
e.printStackTrace();
}
}// end of method

public static void combine() {
int headerVersion = 6;
```

```
String featureSet = "-feature-set-1";
String headerFolder = "header";
// String headerFileName = "header_V" + headerVersion + "" + featureSet
// + ".txt"; //
String headerFileName = "header-overallstats.txt";
// header.txt
// String headerFileName = "header-error.txt"; // header.txt

// String featureFolder = "error-customer";
// String featureFolder = "test-set-extracted-feature-set-1" + "-v"
// + headerVersion;
String featureFolder = "extractedFeatureFile-feature-set-1-V"
+ headerVersion;
// String trainingSetFolder = "trainingSet";
String trainingSetFolder = "trainingSet-feature-set-1-v"
+ headerVersion;
// String trainingSetFolder = "error-training-set";
String outputFileName = getOutPutFileName(trainingSetFolder);
Path outputFilePath = Paths.get(new File(outputFileName)
.getAbsolutePath());

List<Path> inputs = new LinkedList<Path>();
inputs.add(Paths.get(new File(headerFolder + "/" + headerFileName)
.getAbsolutePath())); // add header file's location at the
// beginning
inputs.addAll(getPathofAllFilesFromFolder(featureFolder));

// testOutputFileGen(outputFilePath.toString());
```

```
print("Started Conacatenating...");
concatenateFiles(inputs, outputFilePath);
print("...Finished Concatenating");
}// end of emethod

public static void concatenateFiles(List<Path> inputs, Path output) {
if (Files.notExists(output.getParent())) {
try {
Files.createDirectories(output.getParent());

} catch (Exception e) {
print("Could not create file");
e.printStackTrace();
}// end try catchh
}// end if

Path instanceInfoFile = Paths.get(output.getParent().toAbsolutePath()
.toString()
+ "/" + output.getParent().getFileName().toString() + ".inst");
try { // always create a new file
Files.createFile(output);
Files.createFile(instanceInfoFile);
} catch (IOException e1) {
// TODO Auto-generated catch block
e1.printStackTrace();
}// end try catch

// instance file
// System.out.println("instance file " + instanceInfoFile.toString());

// Charset for read and write
Charset charset = StandardCharsets.UTF_8;
// Join files (lines)
int curIndex = 0;
for (Path path : inputs) {

try {
List<String> lines = Files.readAllLines(path, charset);
Files.write(output, lines, charset, StandardOpenOption.CREATE,
StandardOpenOption.APPEND);

if (path.getFileName().toString().endsWith(".txt")) { // skip
// for
// header
continue;
}// end if
// the instance file will be garbase for folders other than
// overallstats
String[] words = lines.get(0).split(",");
List<String> instanceLine = new LinkedList<String>();
instanceLine.add(curIndex + "," + words[0] + "," + words[1]
+ "," + words[2]);
Files.write(instanceInfoFile, instanceLine, charset,
StandardOpenOption.CREATE, StandardOpenOption.APPEND);
curIndex++;
```

```
} catch (IOException e) {
print("Error reading files");
// TODO Auto-generated catch block
e.printStackTrace();
}// end of try catch
}// end of for
}// end of method

public static void testOutputFileGen(String str) {
print(str);
}// end of method

public static String getOutPutFileName(String trainingSetFolder) {
DateFormat dateFormat = new SimpleDateFormat("MM_dd_HH_mm_yyyy");
Date date = new Date();
// System.out.println(dateFormat.format(date)); //2014/08/06 15:59:48
String endPortion = dateFormat.format(date);
String OUT_FILE_NAME = trainingSetFolder + "/"
+ OUT_FILE_NAME_START_PORTION + "_" + endPortion + ".arff";
return OUT_FILE_NAME;
}// end of method

public static List<Path> getPathofAllFilesFromFolder(String directory) {
List<Path> path = new LinkedList<Path>();

File[] files = new File(directory).listFiles();
for (File file : files) {
if (file.isFile()) {
```

```
path.add(Paths.get(file.getAbsolutePath()));
}// end if
}// end for
return path;
}// end of method

public static void print(String str) {
System.out.println(str);
}// end of method
}// end of method
```

```
package org.powertac.trialmodule;

/*
 * Created by Saiful Abu
 On April 19, 2016
 This program writes prediction and true value for moving average prediction
 customers.

 For each time slot > 360 + 168, this program writes down the following in a
 time slot —— customerName —— powerType —— true avg usage —— predicted a

 It works on some test files. For each file you have to run the program once
 appended to a output file. If the output file is not empty already. For the
 the output file if exists deleted then created, for other files, the program
 predictions to the file.

 Name of the output file is movingAvgPerformance.csv
 At the end, if analysis of the file is true, it loads the csv file and figu
 of the predictors. And finally writes down teh prediction errors.
 * */
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FilenameFilter;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;

import org.apache.commons.io.FileUtils;
import org.apache.log4j.Logger;
import org.joda.time.DateTime;
import org.powertac.common.ClearedTrade;
import org.powertac.common.CustomerInfo;
import org.powertac.common.TariffTransaction;
import org.powertac.common.TimeService;
import org.powertac.common.Timeslot;
import org.powertac.common.WeatherReport;
import org.powertac.common.enumerations.PowerType;
import org.powertac.common.msg.TimeslotUpdate;
import org.powertac.common.repo.BrokerRepo;
import org.powertac.common.repo.CustomerRepo;
import org.powertac.common.repo.OrderbookRepo;
import org.powertac.common.repo.TimeslotRepo;
import org.powertac.common.repo.WeatherReportRepo;
import org.powertac.common.spring.SpringApplicationContext;
import org.powertac.logtool.LogtoolContext;
import org.powertac.logtool.common.DomainObjectReader;
import org.powertac.logtool.common.NewObjectListener;
import org.powertac.logtool.example.MktPriceStats;
import org.powertac.logtool.ifc.Analyzer;

import tools.Settings;
import weka.classifiers.Classifier;
import weka.clusterers.Clusterer;
import weka.core.Attribute;
import weka.core.FastVector;
import weka.core.Instance;
import weka.core.Instances;

public class MovingAvgPerformanceLogger extends LogtoolContext implements
Analyzer {

HashMap<String, PowerUsageRepo> customerPowerUsageMap;
HashMap<String, CustomerUsageStorer> customerStatisticsStorer;
int START_FROM_SLOT = 600;
int agentIdForSanityCheck = 4470; // agent ude
int slotAgentSanityCheck = 610; // 441; // 610;

static boolean ANALYZE_PERFORMANCE_IN_THE_END = true;
// debug purpose variables

static String logFileNames[] = {

"", "01.test-01", "02.test-02", "03.test-03", "04.test-04", "05.test-05",

};
static int fileIndex = 3;
static String logFileName = logFileNames[fileIndex];
static private Logger log = Logger.getLogger(MktPriceStats.class.getName());

// service references
private TimeslotRepo timeslotRepo;
private TimeService timeService;
private WeatherReportRepo weatherReportRepo;
private OrderbookRepo OrderbookRepo;
private DomainObjectReader dor;
private Timeslot timeslot;
int counter = 0;

private CustomerRepo customerRepo;

// Data
private TreeMap<Integer, ClearedTrade[]> data;
private TreeMap<Integer, CustomMarketTransaction> marketData;
private int ignoreInitial = 0; // timeslots to ignore at the beginning
private int ignoreCount = 0;
private int indexOffset = 0; // should be
// Competition.deactivateTimeslotsAhead - 1

private PrintWriter output = null;
private static String dataFilename = "clearedTrades.data";
public double brokerID;
private BrokerRepo brokerRepo;

// saif
int totalSlot = 360;
int previousTimeSlot = 0;
int totalTariffTransaction = 0;
Map<Integer, WeatherReport> weatherData;

// /////////
int MAXIMUM_PREVIOUS_WEEK_OFFSET = 6;
// PowerUsageRepo powerUsageRepo;
HashMap<Long, PowerUsageRepo> brokerProfile;

// stores predictorType (eg simplekmeans-2 ---> Hashmap(powertype, cluster)
static HashMap<String, HashMap<String, HashMap<Integer, Classifier>>> cluster
static HashMap<String, Classifier> individualPredictorMap = new HashMap<Strin
static HashMap<String, HashMap<String, Clusterer>> clusterModelMap = new Hash

public static void calculateError() throws Exception {
```

```
FileReader fileReader = new FileReader(
"OfflinePerformanceEvaluation\\MovingAvgOffline.csv");

// Always wrap FileReader in BufferedReader.
BufferedReader bufferedReader = new BufferedReader(fileReader);
String line = bufferedReader.readLine(); // read the first line
String[] predictors = line.split(","); // the last slot is not predictor
int numberPredictors = predictors.length - 2;

double[] totalPercentError = new double[numberPredictors];
double[] totalError = new double[numberPredictors];
double[] predictedValue = new double[numberPredictors];
int numInstances = 0;

// debug variable
int numPercentWhereKmeansWasBetter = 0;
double KMeansPercentError = 0;
while ((line = bufferedReader.readLine()) != null) {
if (line.isEmpty()) {
continue;
}// end if
String[] tokens = line.split(",");
double trueValue = Double.parseDouble(tokens[tokens.length - 1]);
// checking the effect of the true value. for now avoiding the true
// value if less than 1.
if (trueValue < 1) {
continue;
}
```

```
// end if

for (int predNum = 0; predNum < numberPredictors; predNum++) {
predictedValue[predNum] = Double
.parseDouble(tokens[predNum + 1]);
double error = Math.abs((trueValue - predictedValue[predNum])
/ trueValue) * 100;
totalPercentError[predNum] += (error);
totalError[predNum] += Math.abs(trueValue
- predictedValue[predNum]);
// debug to check why kemans-2 percent error is inferior
// than
// moving avg
if (predNum == 1) {
KMeansPercentError = error;
}// end if
if (predNum == 4) {
if (error > KMeansPercentError) {
numPercentWhereKmeansWasBetter++;
}
}// end if
// end debug
}// end for

numInstances++;
}// end while

if (numInstances != 0) {
```

```
for (int predNum = 0; predNum < numberPredictors; predNum++) {
double avgPercentError = totalPercentError[predNum]
/ numInstances;
double avgError = totalError[predNum] / numInstances;
System.out.print(predictors[predNum + 1] + " :: ");
System.out.print("Avg abs error "
+ String.format("%.2f", avgError));
System.out.println(" -- Avg % abs error "
+ String.format("%.2f", avgPercentError) + " %");
}// end for
}// end if

// debug to check kmeans-2 vs movign avg
// System.out.println("K-Means-2 was better "
// + numPercentWhereKmeansWasBetter + " out of " + numInstances);
// debug
bufferedReader.close();
}// end of method

public static String combineFileNamesToMakePathLocation(String... strings) {
if (strings.length <= 0) {
return null;
}
;
String output = strings[0];
for (int i = 1; i < strings.length; i++) {
output += "/" + strings[i];
}// end for
```

```
return output;
}// end of method

public static void loadPredictors() {
String clusterTypeFolderRoot = combineFileNamesToMakePathLocation(
Settings.TRAINING_ROOT_FOLDER_NAME,
Settings.CLUSTER_PREDICTOR_FOLDER_NAME);
for (File clusterTypeFolder : new File(clusterTypeFolderRoot)
.listFiles()) {
String clusterTypeName = clusterTypeFolder.getName();
HashMap<String, HashMap<Integer, Classifier>> powerTypeMap = new HashMap<Stri
clusterPredictorMap.put(clusterTypeName, powerTypeMap);
for (File powerTypeFolder : clusterTypeFolder.listFiles()) {
String powerTypeName = powerTypeFolder.getName();
HashMap<Integer, Classifier> classfierMap = new HashMap<Integer, Classifier >
powerTypeMap.put(powerTypeName, classfierMap);
for (File clusterNumberFolder : powerTypeFolder.listFiles()) {
File classiferFile = clusterNumberFolder.listFiles()[0];
// System.out.println(classiferFileName);
int clusterNumber = Integer
.parseInt(clusterNumberFolder.getName()
.substring(
clusterNumberFolder.getName()
.indexOf('-') + 1));
try {
Classifier classifer = (Classifier) weka.core.SerializationHelper
.read(classiferFile.getAbsolutePath());
classfierMap.put(clusterNumber, classifer);
// System.out.println("loaded one classifier");
} catch (Exception e) {
// TODO Auto-generated catch block
System.out
.println("Could not load the classifier for cluster");
System.out.println("file name "
+ classiferFile.toString());
System.exit(0);
}// end of try catch
}// end for
}// end for
}// end for

// now load the classiiers for individual predictors
String individualPredictorFolder = combineFileNamesToMakePathLocation(
Settings.TRAINING_ROOT_FOLDER_NAME,
Settings.INDIVIDUAL_PREDICTOR_FOLDER_NAME);
for (File customerFolder : new File(individualPredictorFolder)
.listFiles()) {
File classiferFile = customerFolder.listFiles(new FilenameFilter() {
public boolean accept(File dir, String name) {
return name.toLowerCase().endsWith(".model");
}
})[0];

try {
Classifier classifer = (Classifier) weka.core.SerializationHelper
.read(classiferFile.getAbsolutePath());
```

```
individualPredictorMap.put(customerFolder.getName(), classifer);
System.out.println(classiferFile.getName());
} catch (Exception e) {
// TODO Auto-generated catch block
System.out.println("Could not load the classifier for cluster");
System.out.println("file name " + classiferFile.toString());
System.exit(0);
}// end of try catch

}// end for

// now load clusterModels
String clusterRootFolder = combineFileNamesToMakePathLocation(
Settings.TRAINING_ROOT_FOLDER_NAME,
Settings.CLUSTER_FOLDER_NAME);
for (File clusterFolder : new File(clusterRootFolder).listFiles()) {
String clusterTypeName = clusterFolder.getName();

HashMap<String, Clusterer> powerTypeClusterMap = new HashMap<String, Clustere
clusterModelMap.put(clusterTypeName, powerTypeClusterMap);
for (File powerTypeFolderFile : clusterFolder.listFiles()) {
if (powerTypeFolderFile.listFiles().length == 0) {
continue;
}// end if
String powerTypeName = powerTypeFolderFile.getName();
try {
Clusterer clusterModel = (Clusterer) weka.core.SerializationHelper
.read(powerTypeFolderFile.listFiles()[0]
.getAbsolutePath());
powerTypeClusterMap.put(powerTypeName, clusterModel);
System.out.println("loaded cluster model");
} catch (Exception e) {
// TODO Auto-generated catch block
System.out.println("Could not load the cluster model");

System.out.println("file name "
+ powerTypeFolderFile.listFiles()[0].toString());
System.exit(0);

}// end of try catch
}// end for
}// end for

}// end of method

/**
 * Main method just creates an instance and passes command-line args to its
 * inherited cli() method.
 *
 * @throws Exception
 */
public static void main(String[] args) throws Exception {
System.out.println("Data Extracting from log file " + logFileName
+ " ...");
new MovingAvgPerformanceLogger().cli(args);
```

```
if (ANALYZE_PERFORMANCE_IN_THE_END) {
calculateError();
}// end if
System.out.println("... Data Extraction finished from " + logFileName);
}// end of main

/**
 * Takes two args, input filename and output filename
 */
private void cli(String[] args) {
// load cluster predictors and individual predictors
loadPredictors();
// create the csv file for tariff transaction inspection
// String tariffTranspectionFileHeader =
// "Transaction_Type,Tariff_Id,Tariff_Type,Customer_Name,Customer_PowerType,
// csvWriter = new CSVWriter(
// "C:/Users/iasrluser/Google Drive/Research/result/tariff_transaction_inspe
// + slotAgentSanityCheck
// + "_agent_"
// + agentIdForSanityCheck + ".csv",
// tariffTranspectionFileHeader);

String logFolder = "log";

String logFileExtension = "state";
String logFileLocation = logFolder + "/" + logFileName + "."
+ logFileExtension;
```

```
String outputFolderBase = "OfflinePerformanceEvaluation";
String outputFileName = "MovingAvgOffline.csv";
String outputFileLocation = outputFolderBase + "/" + outputFileName;

/*
 * if (args.length != 2) {
 * System.out.println("Usage: <analyzer> input-file output-file");
 * return; }//end of if dataFilename = args[1]; super.cli(args[0],
 * this);
 */
if (fileIndex == 1) {
if (Files.exists(Paths.get(outputFolderBase)) == true) {
try {
// Files.deleteIfExists(Paths.get(outputFolderBase));
FileUtils.forceDelete(new File(outputFolderBase));
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}// end try catch
}// end if

// now create the directory
try {
Files.createDirectory(Paths.get(outputFolderBase));
Files.createFile(Paths.get(outputFileLocation));
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}// end of folder creation
dataFilename = outputFileLocation;
super.cli(logFileLocation, this);
output.close();
// csvWriter.closeWriter();
}// end of method

/*
 * (non-Javadoc)
 *
 * @see org.powertac.logtool.ifc.Analyzer#setup()
 */
@Override
public void setup() {
dor = (DomainObjectReader) SpringApplicationContext.getBean("reader");
timeslotRepo = (TimeslotRepo) getBean("timeslotRepo");
// weatherReportRepo = (WeatherReportRepo) getBean("WeatherReportRepo");
timeService = (TimeService) getBean("timeService");
brokerRepo = (BrokerRepo) SpringApplicationContext
.getBean("brokerRepo");
// registerNewObjectListener(new BrokerHandler(), Broker.class);

registerNewObjectListener(new TimeslotUpdateHandler(),
TimeslotUpdate.class);
registerNewObjectListener(new TariffTransactionHandler(),
TariffTransaction.class);
// registerNewObjectListener(new MarketTransactionHandler(),
// MarketTransaction.class);
// registerNewObjectListener(new BalancingTransactionHandler(),
// BalancingTransaction.class);
// registerNewObjectListener(new TimeslotHandler(), Timeslot.class);
registerNewObjectListener(new WeatherReportHandler(),
WeatherReport.class);
// registerNewObjectListener(new OrderbookHandler(), Orderbook.class);

ignoreCount = ignoreInitial;
data = new TreeMap<Integer, ClearedTrade[]>();
// saif
weatherData = new HashMap<Integer, WeatherReport>();
// powerUsageRepo = new PowerUsageRepo();
brokerProfile = new HashMap<Long, PowerUsageRepo>();
// saif
marketData = new TreeMap<Integer, CustomMarketTransaction>();
customerPowerUsageMap = new HashMap<String, PowerUsageRepo>();
customerStatisticsStorer = new HashMap<String, CustomerUsageStorer>();
// ///////////////////

try {

// output = new PrintWriter(new File(dataFilename));
output = new PrintWriter(new FileOutputStream(
new File(dataFilename), true /* append = true */));
if (fileIndex == 1) {
String header = "CustomerName,";
for (String clusterTypeName : clusterModelMap.keySet()) {
```

```
header += clusterTypeName + ",";
}
header = header + "IndivPredictor, MovingAvg, True-Usage";
output.println(header);
}// end if

} catch (FileNotFoundException e) {
log.error("Cannot open file " + dataFilename);
}
}

/*
 * (non-Javadoc)
 *
 * @see org.powertac.logtool.ifc.Analyzer#report()
 */
@Override
public void report() {
boolean ret;
ret = true;
if (ret) {
// for now wont use the report method
return;
}// end if

for (Map.Entry<Integer, CustomMarketTransaction> entry : marketData
.entrySet()) {
String delim = "";
```

```
Integer timeslot = entry.getKey();
CustomMarketTransaction trades = entry.getValue();

// if (trades.length != 24)
// log.error("short array " + trades.length);
// for (int i = 0; i < trades.length; i++) {

if (null == trades) {
output.print(delim + "[0.0 0.0]");
} else {

output.format("%s %d,%d,%.2f,%.2f, %.2f, %.2f, %.2f", delim,
trades.day, trades.hour, trades.temp,
trades.cloudCoverage, trades.windDirection,
trades.windSpeed, trades.clearingPrice);
}

delim = " ";
// }

output.println();

}
output.close();

}

/*********** Handlers Begin ********/
```

```
// ———————————————————————————————
// catch TimeslotUpdate events
class TimeslotUpdateHandler implements NewObjectListener {

@Override
public void handleNewObject(Object thing) {

int slotOfInterest = timeslotRepo.currentSerialNumber() − 1;
if (slotOfInterest % 100 == 0) {
System.out.println("Processing Time Slot " + slotOfInterest
+ " ...");
}// end if
// if (slotOfInterest < START_FROM_SLOT) {
if (slotOfInterest < 361) {
return;
}// end if
processSlot(slotOfInterest);
}// end of method

/*
 * Forms supervised learning example for a timeSlotIndex.
 * [temporalString] [temperature of 5 slots starting from the slot of
 * interest] [previous week's usage 5 features centered at the same slot
 * as the slot of interest] [actual usage]
 */
void processSlot(int timeSlotIndex) {
// for (PowerUsageRepo powerUsageRepo : brokerProfile.values()) {
// for (Long brokerIdentifier : brokerProfile.keySet()) {
```

```
for (String customerName : customerPowerUsageMap.keySet()) {
PowerUsageRepo powerUsageRepo = customerPowerUsageMap
.get(customerName);
if (powerUsageRepo == null) {
// System.out.println("No record for broker "
// + brokerIdentifier);
continue;
}

double trueAvgUsage = powerUsageRepo.getAvgUsage(timeSlotIndex);
int day = timeslotRepo.getDateTimeForIndex(timeSlotIndex)
.getDayOfWeek();
int hour = timeslotRepo.getDateTimeForIndex(timeSlotIndex)
.getHourOfDay();
double movingAvgUsage = powerUsageRepo.getMovingAvgUsage(day,
hour);

powerUsageRepo.addtoMovingAvgUsage(day, hour, trueAvgUsage);
String instance = customerName + ",";
// System.out.println(instance);

// load the proper customer Usage storage
CustomerUsageStorer custUsage = customerStatisticsStorer
.get(customerName);

FastVector attribs = new FastVector(6);
attribs.addElement(new Attribute("cloudcover"));
attribs.addElement(new Attribute("temperature"));
attribs.addElement(new Attribute("windspeed"));
attribs.addElement(new Attribute("mean"));
attribs.addElement(new Attribute("stddev"));
attribs.addElement(new Attribute("prediction"));
// form instance string for predictor
Instances dataset = new Instances("TestInstances", attribs, 0);
// Assign the prediction attribute to the dataset. This
// attribute will
// be used to make a prediction.
dataset.setClassIndex(dataset.numAttributes() - 1);

double[] attributes = new double[6];
WeatherReport weatherReport = weatherData.get(timeSlotIndex);
attributes[0] = weatherReport.getCloudCover();
attributes[1] = weatherReport.getTemperature();
attributes[2] = weatherReport.getWindSpeed();
attributes[3] = custUsage.getMean(day, hour);
attributes[4] = custUsage.getStdDev(day, hour);

Instance instanceToPredict = new Instance(1.0, attributes);

dataset.add(instanceToPredict);
instanceToPredict.setDataset(dataset);

// form cluster string for cluster
double[] weeklyAvgUsage = custUsage.getWeeklyAvgOfAllSlots();
Instance instanceToCluster = new Instance(1.0, weeklyAvgUsage);
```

```
// predict for each cluster instance
String customerPowerType = powerUsageRepo.customerInfo
.getPowerType().toString();
for (String clusterTypeName : clusterModelMap.keySet()) {
Clusterer clusterer = (Clusterer) clusterModelMap.get(
clusterTypeName).get(customerPowerType);

try {
int clusterIndex = clusterer
.clusterInstance(instanceToCluster);
Classifier classifer = (Classifier) clusterPredictorMap
.get(clusterTypeName).get(customerPowerType)
.get(clusterIndex);
double prediction = classifer
.classifyInstance(instanceToPredict);
// System.out.println("successfully predicted");
instance += prediction + ",";

} catch (Exception e) {
// TODO Auto-generated catch block
System.out.println("could not predict the instance "
+ e.getMessage());
}
}// end for

// predict using the individual predictors
Classifier classifer = individualPredictorMap.get(customerName);
try {
```

```
double prediction = classifer
.classifyInstance(instanceToPredict);
// System.out.println("successfully predicted");
instance += prediction + ",";
} catch (Exception e) {
// TODO Auto-generated catch block
System.out.println("could not predict the instance");
}

instance += movingAvgUsage + "," + trueAvgUsage;

// all prediction has been made so update record now
custUsage.updateRecord(day, hour);
if (timeSlotIndex > START_FROM_SLOT)
writeALine(instance);
}// end for
}// end of method

void debug_avg_usage(int slot, double usage, long brID, String instance) {
if ((slot != slotAgentSanityCheck))
return;
if (brID != agentIdForSanityCheck) {
return;
}// end if
DateTime dateTime = timeslotRepo
.getDateTimeForIndex(slotAgentSanityCheck);
int year = dateTime.getYear();
int month = dateTime.getMonthOfYear();
int dayOfMOnth = dateTime.getDayOfMonth();
int dayOfWeek = dateTime.getDayOfWeek();
int hourOfDay = dateTime.getHourOfDay();
WeatherReport weatherReport2 = weatherData
.get(slotAgentSanityCheck);
WeatherReport weatherReport1 = weatherData
.get(slotAgentSanityCheck - 1);
WeatherReport weatherReport0 = weatherData
.get(slotAgentSanityCheck - 2);

System.out.println(instance);
}// end of method

void writeALine(String string) {
output.write(string + "\n");
output.flush();
}// end of method

void testWriteLine(int currentSlot, int slotToTest, String str) {
if (currentSlot == slotToTest) {
writeALine(str);
}// end if
}// end of method

/*
 * weatherString = <temp-0><temp-1>...<temp-(n-1)> temp-(i) means
 * temperature of t-i slots temperature where t is the current slot.
 *
 * @param weatherData = list of weather report for each slot i
 *
 * @param currentTimeSlot = the slot we are predicting for
 *
 * @param totalReports = number of reports we want to incorporate in the
 * training example
 */
String formWeatherString(Map<Integer, WeatherReport> weatherData,
int currentTimeSlot, int totalReports) {
String weatherReportString = "";
for (int i = 0; i < totalReports; i++) {
int timeSlot = currentTimeSlot - i;
WeatherReport weatherReport = weatherData.get(timeSlot);
double temperature = weatherReport.getTemperature();
weatherReportString = weatherReportString + temperature + ",";
}// end for
return weatherReportString;
}// end of method

String formInstanceString(int timeSlot, PowerUsageRepo powerUsageRepo) {
String instance = "";
int previousSlot = timeSlot - 1;
String temporalString = formTemporalString(timeSlot);
String weatherString = formWeatherString(weatherData, timeSlot, 6);
String immediateUsage = formPreviousAvgUsageString(powerUsageRepo,
previousSlot, 12);
String previousWeekUsage = formPreviousAvgUsageString(
powerUsageRepo, timeSlot - 24 * 7
```

```
+ MAXIMUM_PREVIOUS_WEEK_OFFSET, 13);
double currentSlotUsage = powerUsageRepo.getAvgUsage(timeSlot);
instance = temporalString + weatherString + immediateUsage
+ previousWeekUsage + currentSlotUsage;
return instance;
}// end of method

void testFormInstanceString(int currentSlot, int slotToTest,
PowerUsageRepo powerUsageRepo) {
if (currentSlot == slotToTest) {
System.out.println(formInstanceString(slotToTest,
powerUsageRepo));
}// end if
}// end of method

/*
 * forms a string of the following data: <consumption slot 0>
 * <consumption slot -1> .... <consumption slot - (totalslots-1)>
 */
String formPreviousAvgUsageString(PowerUsageRepo powerUsageRepo,
int startingIndex, int totalSlots) {
String usageStr = "";
for (int i = 0; i < totalSlots; i++) {
int slot = startingIndex - i;
// double averageUsage = powerUsageRepo.getUsage(slot) /
// powerUsageRepo.getPopulation(slot);
double averageUsage = powerUsageRepo.getAvgUsage(slot);
usageStr = usageStr + averageUsage + ",";
```

```
}// end for
return usageStr;
}// end of method

/*
 * @param timeSlot forms the string of the following format <month
 * [1-12]> <dayOfMonth[1-31]> <dayOfWeek[1-7]> <hour[0-23]>
 */
String formTemporalString(int timeSlot) {

String temporalString = "";
int monthOfYear = timeslotRepo.getDateTimeForIndex(timeSlot)
.getMonthOfYear();
int dayOfMonth = timeslotRepo.getDateTimeForIndex(timeSlot)
.getDayOfMonth();
int dayOfWeek = timeslotRepo.getDateTimeForIndex(timeSlot)
.getDayOfWeek();
int hourOfDay = timeslotRepo.getDateTimeForIndex(timeSlot)
.getHourOfDay();
temporalString = monthOfYear + "," + dayOfMonth + "," + dayOfWeek
+ "," + hourOfDay + ",";
return temporalString;
}// end of method

void testFormTemporalStringMethod(int currentSlot, int slotToTest) {
if (currentSlot == slotToTest) {
System.out.println(formTemporalString(slotToTest));
}// end if
}// end of method

void testFormUsageString(int currentSlot, int slotToTest,
PowerUsageRepo powerUsageRepo) {
if (currentSlot == slotToTest) {
System.out.println(formPreviousAvgUsageString(powerUsageRepo,
currentSlot, 1));
System.out
.println(formPreviousAvgUsageString(powerUsageRepo,
currentSlot - 24 * 7
- MAXIMUM_PREVIOUS_WEEK_OFFSET, 13));

TimeSlotRecord record = powerUsageRepo.getRecord(currentSlot);
System.out.println(record.toString());
System.out.println(formPreviousAvgUsageString(powerUsageRepo,
currentSlot, 1));
}// end if
}// end of method

void testWeatherFormMethod(int currentSlot, int slotToTest) {
if (currentSlot == slotToTest) {
String str = formWeatherString(weatherData, slotToTest, 2);
System.out.println(str);
WeatherReport report = weatherData.get(slotToTest);
report.toString();
report = weatherData.get(slotToTest - 1);
report.toString();
}// end if
}// end of test method

void testDayFormats(int currentSlot) {
int previousSlot = currentSlot - 1;

//
int monthOfYear = timeslotRepo.getDateTimeForIndex(currentSlot)
.getMonthOfYear();
int dayOfMonth = timeslotRepo.currentTimeslot().getStartTime()
.getDayOfMonth();
int dayOfWeek = timeslotRepo.currentTimeslot().dayOfWeek();
int dayHour = timeslotRepo.currentTimeslot().slotInDay();
System.out.print(monthOfYear + " " + dayOfMonth + " " + dayOfWeek
+ " " + dayHour + " : ");

monthOfYear = timeslotRepo.getDateTimeForIndex(previousSlot)
.getMonthOfYear();
dayOfMonth = timeslotRepo.getDateTimeForIndex(previousSlot)
.getDayOfMonth();
dayOfWeek = timeslotRepo.getDateTimeForIndex(previousSlot)
.getDayOfWeek();
dayHour = timeslotRepo.getDateTimeForIndex(previousSlot)
.getHourOfDay();
System.out.println(monthOfYear + " " + dayOfMonth + " " + dayOfWeek
+ " " + dayHour);
}// end of method

void testUsageInsertion(int currentSlot, PowerUsageRepo powerUsageRepo) {
```

```
if (currentSlot == 540) {
double usage = powerUsageRepo.getUsage(currentSlot);
int customerCount = powerUsageRepo.getPopulation(currentSlot);
System.out.println("*** usage : " + usage + " customerCount "
+ customerCount + " ***");
}// end if
}// end of method
}// end of class

class WeatherReportHandler implements NewObjectListener {

@Override
public void handleNewObject(Object thing) {

WeatherReport wr = (WeatherReport) thing;

// System.out.println("In the weather report handler");

int timeSlotIndex = wr.getTimeslotIndex();
// System.out.println("Putting weather " + timeSlotIndex);
weatherData.put(timeSlotIndex, wr);
}
}// end of class

class TariffTransactionHandler implements NewObjectListener {

@Override
public void handleNewObject(Object thing) {
```

```
TariffTransaction tariffTransaction = (TariffTransaction) thing;

// if (tariffTransaction.getTxType() !=
// TariffTransaction.Type.CONSUME) {

CustomerInfo customerInfo = tariffTransaction.getCustomerInfo();
if (customerInfo == null) {
return;
}// end if
if (customerInfo.getPowerType() != PowerType.CONSUMPTION) {
return;
}// end if

PowerUsageRepo powerUsageRepo;
CustomerUsageStorer custUsage;
if (customerPowerUsageMap.containsKey(customerInfo.getName()) == true) {
powerUsageRepo = customerPowerUsageMap.get(customerInfo
.getName());
custUsage = customerStatisticsStorer
.get(customerInfo.getName());
} else {
powerUsageRepo = new PowerUsageRepo();
powerUsageRepo.customerInfo = customerInfo;
customerPowerUsageMap.put(customerInfo.getName(),
powerUsageRepo);

custUsage = new CustomerUsageStorer(customerInfo.getName(),
tariffTransaction.getTariffSpec());
customerStatisticsStorer.put(customerInfo.getName(), custUsage);
}// end if else

double usageKwh = Math.abs(tariffTransaction.getKWh());
int userCount = tariffTransaction.getCustomerCount();
if ((usageKwh == 0) || (userCount == 0)) {
return;
}// end if

int reportOfSlot = tariffTransaction.getPostedTimeslotIndex();
powerUsageRepo.addUsage(reportOfSlot, usageKwh, userCount);
int dayOfWeek = timeslotRepo.getDateTimeForIndex(reportOfSlot)
.getDayOfWeek();
int hourOfDay = timeslotRepo.getDateTimeForIndex(reportOfSlot)
.getHourOfDay();
int month = timeslotRepo.getDateTimeForIndex(reportOfSlot)
.getMonthOfYear();
int dayOfMonth = timeslotRepo.getDateTimeForIndex(reportOfSlot)
.getDayOfMonth();
custUsage.addUsage(dayOfWeek, hourOfDay, month, dayOfMonth,
usageKwh, userCount, reportOfSlot);
}// end of method

}// end of class

/********************** Handler End *****************************/

class CustomMarketTransaction {
```

```
int timeslotIndex;
double boughtMWh;
double soldMWh;
double mWh;
double price;
double boughtprice;
double soldprice;
int count;
double balancingTrans;
int day;
int hour;
double temp;
double cloudCoverage;
double windSpeed;
double windDirection;
double clearingPrice;
double[] clearings = new double[8720];

CustomMarketTransaction() {
timeslotIndex = 0;
boughtMWh = 0.0;
soldMWh = 0.0;
boughtprice = 0.0;
soldprice = 0.0;
mWh = 0.0;
price = 0.0;
count = 0;
balancingTrans = 0.0;
day = 0;
hour = 0;
temp = 0.0;
cloudCoverage = 0.0;
windSpeed = 0.0;
windDirection = 0.0;
clearingPrice = 0.0;
}
}
}

/**************** class definition **********************/
class TimeSlotRecord {
int timeSlotIndex;
double usage;
int population;

TimeSlotRecord(int timeSlotIndex) {
this.timeSlotIndex = timeSlotIndex;
usage = 0;
population = 0;
}// end of constructor

@Override
public String toString() {
return "timeSlot : " + timeSlotIndex + " usage : " + usage
+ " population : " + population;
}// end of method
}// end of class

// this class stores energy usage for each customer
// a hashmap timeslot ---> usage
// also keeps a moving average data structure for the customer
class PowerUsageRepo {
HashMap<Integer, TimeSlotRecord> slotUsageMap;
double[] weeklyMovingAvgUsage = new double[168];
CustomerInfo customerInfo;

double getMovingAvgUsage(int day, int hour) {
int slot = (day - 1) * 24 + hour;
return weeklyMovingAvgUsage[slot];
}// end of method

void addtoMovingAvgUsage(int day, int hour, double recentAvgUsage) {
int slot = (day - 1) * 24 + hour;
weeklyMovingAvgUsage[slot] = .7 * weeklyMovingAvgUsage[slot] + .3
* recentAvgUsage;

}// end of method

PowerUsageRepo() {
slotUsageMap = new HashMap<Integer, TimeSlotRecord>();
}// end of constructor

void addUsage(int slotIndex, double usageAmount, int customerCount) {
usageAmount = Math.abs(usageAmount); // take positive
TimeSlotRecord slotRecord = slotUsageMap.get(slotIndex);
if (slotRecord == null) {
slotRecord = new TimeSlotRecord(slotIndex);
slotUsageMap.put(slotIndex, slotRecord);
}// end if
// modify old record
slotRecord.usage += usageAmount;
slotRecord.population += customerCount;
}// end of method

double getUsage(int timeSlotIndex) {
TimeSlotRecord slotRecord = slotUsageMap.get(timeSlotIndex);
if (slotRecord == null) {
return 0;
} else {
return slotRecord.usage;
}// end if else
}// end of method

int getPopulation(int timeSlotIndex) {
TimeSlotRecord slotRecord = slotUsageMap.get(timeSlotIndex);
if (slotRecord == null) {
return 0;
} else {
return slotRecord.population;
}// end if else
}// end of method
```

```
double getAvgUsage(int timeSlotIndex) {
TimeSlotRecord slotRecord = slotUsageMap.get(timeSlotIndex);
if (slotRecord == null) {
// System.out.println("Slot record is null");
return -1; // no slot record
} else {
if (slotRecord.population == 0) {
// System.out.println("population 0 for slot "
// + slotRecord.timeSlotIndex);
return -2; // no user
} else {
return slotRecord.usage / slotRecord.population;
}
}// end if else
}// end of method

TimeSlotRecord getRecord(int timeSlotIndex) {
TimeSlotRecord record;
record = slotUsageMap.get(timeSlotIndex);
return record;
}// end of method

class CustomerInformationFormation {
double prediction;
double trueAvgUsage;
double[] weeklyMovingAvgUsage = new double[168];

}// end of class
}// end of class
```

Curriculum Vitae

Saiful Abu was born on December 25, 1988. He went to the Bangladesh University of Engineering and Technology to study Bachelor of Science in Computer Science. He graduated from there in 2012. After working for a few years in the industry, he came to the University of Texas at El Paso in year 2014 during the fall. He graduated with a MS degree in Computer Science in 2016.

Permanent address: 52, Jahidur Rahman Sarak
Khulna, Bangladesh.