COMPGI15: Information Retrieval & Data Mining
Time Series Forecasting Group Project
A hierarchical load forecasting problem: backcasting and forecasting hourly loads for a US utility with 20 zones.

Rupert Chaplin
rupert.chaplin.15@ucl.ac.uk

Artemis Dampa
artemis.dampa.15@ucl.ac.uk

Megane Martinez
megane.martinez.15@ucl.ac.uk

Abstract

We present a range of techniques for tackling hierarchical forecasting of a time series for electricity loads for a US utility, incorporating temperature as an explanatory variable. We consider Multiple Linear Regression, ARIMA, Gradient Boosting and Neural Network approaches. We present results that are comparable to the best achieved in the 2012 Kaggle competition tackling this forecasting problem.
Code is available at https://github.com/rupchap/IRDM2016.

Introduction

A time series can be defined as a set of observations measured successively over a continuous time interval. Every measurement is taken at a specific time which is equally distant to its subsequent. The analysis of time series takes into consideration the ordering of the data points and constitutes a useful tool in order to extract information such as the causal effect of time change on the variable which is examined. After describing and analysing the time series one can gain an understanding of the data and thus, define a model with the best possible fit at the data based on the past values of the examined variable and/or the past and present observations of other variables which might influence the former. Using the created model one can, finally, make predictions and forecast the future values of the examined variable or even use models to backcast and possibly fill in missing values. The ultimate goal of time series analysis is to control and manage the process involved in the time series.

Load forecasting and especially, short term load forecasting which can predict energy load values ranging from one day to one fortnight at hourly or sub hourly intervals, has always been of great importance for managing power systems. It is used in economic generation scheduling, fuel scheduling, unit commitment and system security, as well as in transmission and distribution operations. There are several machine learning methods which can be applied in order to achieve short term load forecasting, such as ARIMA, Multiple linear regression, Gradient boosting and Neural networks.

The dataset used in order to implement the time series techniques for forecasting (and backcasting) is provided by the Load Forecasting track of Global Energy Forecasting Competition 2012. The aim of the competition was to collect different approaches for energy forecasting in order to overcome forecasting challenges in the smart grid world. More specifically, the load forecasting track was addressing a hierarchical load forecasting problem. The goal was to use forecasting (and backcasting) in order to predict hourly loads (in kW) for a US utility with 20 zones.

Data availability

In developing this work, we embraced the same constraints on data use imposed upon the original Kaggle challenge - confining ourselves to use of the originally provided datasets, which are:

- Time series for the electricity load in 20 zones, at hourly intervals between 1 January 2004 and 30 June 2008.

- Time series of temperature readings at 11 stations, at hourly intervals over the same time period (locations not provided)

- Dates of US holidays falling during the period

The task is to generate estimates of electricity load in each zone, and for the region as a whole, for eight hold out weeks within the period. Additionally, forecasts are required for the week 1 July 2008 to 7 July 2008, for which no temperature readings are available.

Temperatures forecasting

In this project, most of the implemented models, use the values provided in the temperature dataset as features for predicting energy loads values. However, the last week of temperature values is missing (2008-07-01 to 2008-07-07). Therefore, predicting the values for this week was a critical challenge for the results would impact the quality of the results provided by the implemented models. Teams that took part in the Kaggle competition usually used naive techniques for forecasting temperature values such as taking the average of the previous values at the same time of year. In this project, we also explored use of ARIMA modelling to forecast the missing temperatures. This approach represents an improvement from what has been done in the Kaggle competition and was one of the challenges to overcome.

Consideration of feature selection

Before determining the features of the model and after performing data cleansing and editing, it is essential to create and examine the time plot of the data. The time plot can show the behavior of the data and reveal important features such as seasonal variation and trend which is defined as an increasing normal number assigned to each observation in chronological order.

Exploratory analysis of patterns within the time series, and the relation between temperature and load levels, are included in the poster which accompanies this paper.

In our case the time plot showed various fluctuations with an annual seasonality which led us to develop a set that combines several features; trend, hour with 24 levels (representing the hours of a day), weekday with seven levels (representing the days of a week), month with 12 levels (representing the months of a year), temperature of the current hour, temperature of the current hour and cubic temperature of the current hour. In addition to those main features, interaction between them was taken into account forming several cross effects; weekday×hour, temperature×month, temperature^2×month, temperature^3×month, temperature×hour, temperature^2×hour and temperature^3×hour. After measuring the feature importance, we plotted the results as shown in Figure 1.

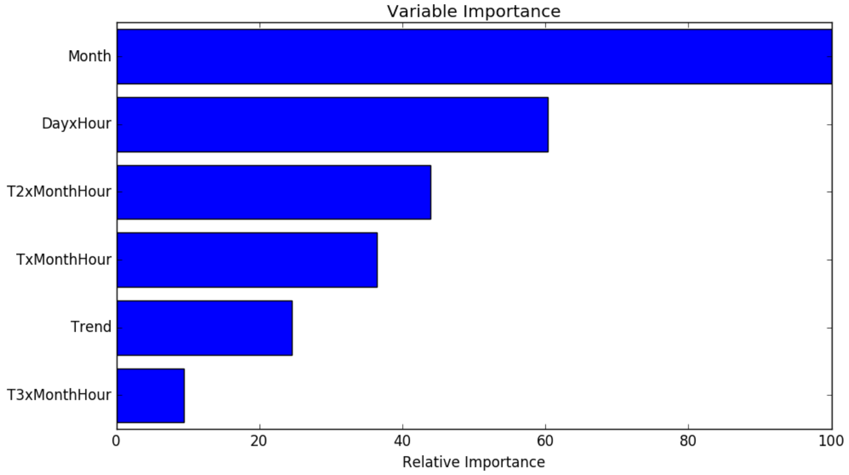

Figure 1: Shows the feature importance measurements over the features

'Variable importance' here is a proxy measure taken as the coefficient value (mean coefficient value in the case of one-hot measures) of a Multiple Linear Regression model, where input variables have first had standardised scaling applied to ensure comparability.

Evaluation Methods

To compare the predictive power of our models, we have adopted the same evaluation method used by the original Kaggle competition. This allows us to objectively compare predictions from the different approaches we consider, and also compare out results to those achieved by those participating in the original competition.

This approach is a Weighted Root Mean Square Error, with a weighting of 1 for each zonal prediction within the period covered by training data, and a weighting of 8 for predictions within the forecast period. Weightings of 20 and 160 are applied for each prediction of total regional load.

Root Mean Square Error is a common evaluation method for regression problems - it is a measure of average absolute distance between predicted and actual results. It has the advantage of being expressed in the same units as that of the data we are trying to predict. Unlike some other approaches, it is indifferent to whether overall discrepancy is due to large discrepancies on a small number of distant outliers, or due to smaller errors affecting a large number of datapoints.

The addition of weighting for this problem means that succesful models will be those which are particularly effective at providing forecasts over future periods for which not all explanatory variables are available.

Monitoring the evaluation metric also allows us to monitor models for 'overfitting'. Overfitting occurs when a model becomes tuned to the specific characteristics of a training data set, and does not generalise well to unseen test data. Overfitting would be signalled if the error measure is higher for a test dataset than it is for training data.

Modelling Approaches

ARIMA

The model

What makes time series different from other regression problems is that they are time dependent. Therefore, instead on trying to find some correlations between the value to predict and external features, one can try to find correlations between the data points in a time series among themselves in order to predict future values. ARIMA (Auto Regressive Integrated Moving Average) is a time series modelling method that uses this approach and has become popular in the seventies. Before we discuss how ARIMA actually works, lets see what the prerequisites are for a time series to be effectively modelled with ARIMA.

Typically, time series can be divided into three components: the trend, the seasonality and the residuals. The trend is the long term direction of the series (upwards or downwards), the seasonality represents recurrent patterns that can be quarterly, monthly, daily and so on. The residuals are the random noise that is left after extraction of the other two components. The interference of the trend, seasonality and residuals produces the final time series. Most of time series modelling approaches, ARIMA included, works on the assumption that the series is stationary, i.e, its mean and variance remain constant over time. In order to use ARIMA to model a time series, one needs to, first, make it stationary on both mean and variance. One of the way of making a time series stationary on mean, is to remove the trend by using first order differencing: $Y'(t) = Y(t) - Y(t-1)$.

Most of time series become stationary after first order differencing. If the residuals time series still has a trend, one can use 2nd order differencing. In order to make a series stationary on variance, one usually uses log transformation: $Y'(t) = \log_{10} Y(t)$. Finally, the combination of both 1st order differencing and log transformation makes a time series stationary on both mean and variance.

ARIMA is divided in three components, each with its parameter. AR stands for Auto Regressive and its parameter in p. I stands for integrated and its parameter is d and MA stands for Moving Average and its parameter is q. As seen previously, ARIMA uses past time series data points to predict the future values. The auto regressive part of the model defines lags of dependent variable. For instance if p = 2, AR will use the past two values to predict a third one. The parameter d of the Integrated part of the ARIMA model simply defines the order of difference. If the first order differencing is enough to make the series stationary one can use the differenced data and put k = 0 or use the original log transformed data and put k = 1. The Moving Average part of the model defines the number of lagged errors used to predict the next value. If q = 2, its means that to predict a value the model will use the past two errors, i.e, differences between the moving average and the actual value.

Design choices

Several statistical libraries or packages offer tools to implement ARIMA modelling. However some are more comprehensive than others. For example, in Python, the package `statsmodels.tsa.arima_model` allows one to create ARIMA modelling but does not have a way to find the parameters (p,d,q) of the best fit model automatically. The package `forecast` in R however does provide this possibility with the `auto.arima()` function.

In our team, none of us were very familiar with R so we decided to use python for the overall implementation and to use an R package (`PypeR`) to still benefit from the additional capabilities of R.

In order to check the stationarity of the time series we used the Dickey-Fuller test where the null hypothesis is that the time series is non-stationary. If the test result is below the critical value we can reject the null hypothesis and say the time series is stationary and suitable for ARIMA modelling.

Checking the effectiveness of the ARIMA models was done by plotting and analysing the ACF (Auto Correlation Fuction) and PACF (Partial Auto Correlation Function) of both the differenced and log transformed data and then the ARIMA output.

The ACF plot represents the correlation coefficient between a time series and lags of itself. For instance, by looking at an ACF plot, one can know if for any data point $Y(t)$ of the series, $Y(t)$ is correlated with $Y(t-1)$, $Y(t-2)$ and so on.

The PACF plot represents the part of the correlation between two data point $Y(t)$ and $Y(t-n)$ that is not explained by their mutual correlation with the $Y(t-1)$ to $Y(t-n+1)$.

By differencing and log transforming the raw data and plot ACF and PACF one can see if there is still some correlation left in the residuals that would be removed by using ARIMA. Plotting the ACF and PACF on the ARIMA results allows to check that there is no more correlation in the residuals and that the ARIMA model is working well.

During this project, ARIMA was used for two purposes: predict the missing energy load values and predict the missing temperature values in order to use them as features in other prediction algorithms.

ARIMA for loads forecasting

At first, it was decided to try implementing ARIMA modelling directly on the energy loads time series to predict the missing values.

However, since the energy loads dataset contains hourly data from 2004-01-01 to 2008-06-30 for 20 different zones in the United States, each of this zone needed to be treated as an independent time series. Moreover, within that time range, 8 separate weeks of data were missing for each zone and needed to be forecasted as well as the week following the last data point of the set.

Most of ARIMA modelling implementation usually assume that there is no missing data within the time range of the given time series. Since the energy loads data sets was missing 8 weeks, it required modelling every range of data in between those missing weeks as an independent time series as well.

As a result, the implemented ARIMA algorithm dealt with 9 independent time series for each zone, thus, 180 time series in total.

Although the implemented algorithm dealt with a lot of independent time series, the results were constant over them. In the following are shown results for the time series from 2004-01-01 to 2005-03-06. The following plot represents the raw time series that does not look stationary, if the overall trend is difficult to identify, there are clearly some patterns present.

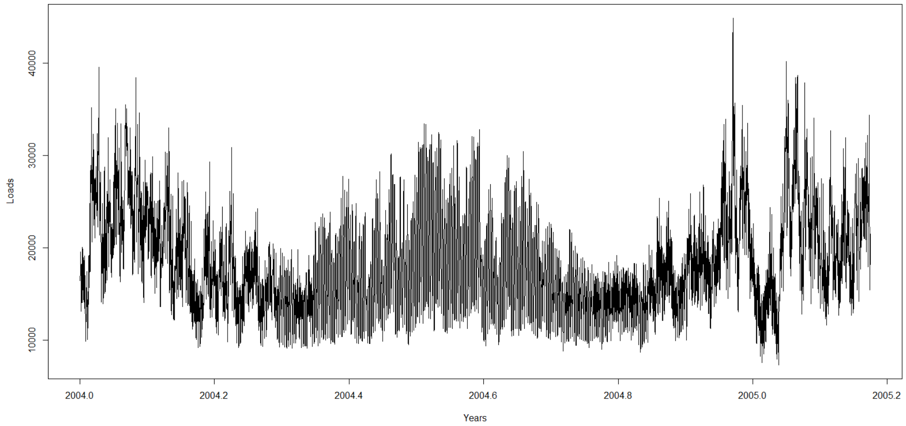

Figure 2: Shows the raw data for the first date range of zone 1

The next plot represents the 1st order differenced log transformed data. This time, the series looks much more stationary.

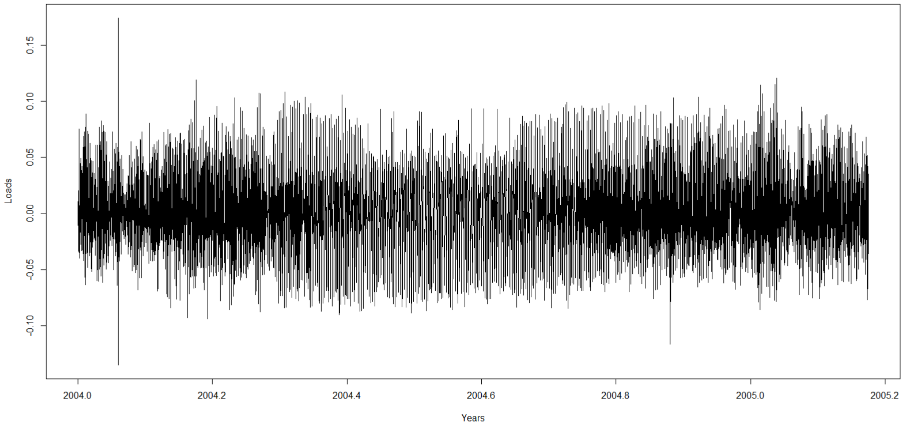

Figure 3: Shows the 1st order differenced log transformed data for the first date range of zone 1

In order to confirm that the series is stationary, the algorithm runs the Dickey-Fuller test. The output is the following:

```
Results of Dickey-Fuller Test:
Test Statistic: -1.154495e+01
p-value: 3.559772e-21
#Lags Used: 27
Number of Observations Used: 10279.000000
Critical Value 5\%: -2.861821
Critical Value 1\%:  -3.430986
Critical Value 10\%: -2.566920
```

The test statistic is smaller than the critical value of 1% which means the rejection of the non-stationarity hypothesis has less than 1% chance to be wrong. One can conclude with confidence that the time series is stationary.

The following charts, show the ACF and PACF values of the time series. Both plots show a lot of spikes going out of the insignificant zone represented by the dotted horizontal lines. It means that the time series residuals has some information left to be extracted by the ARIMA model. One could also notice the daily seasonality represented by the spikes at lag 24 (24 hours seasonality).

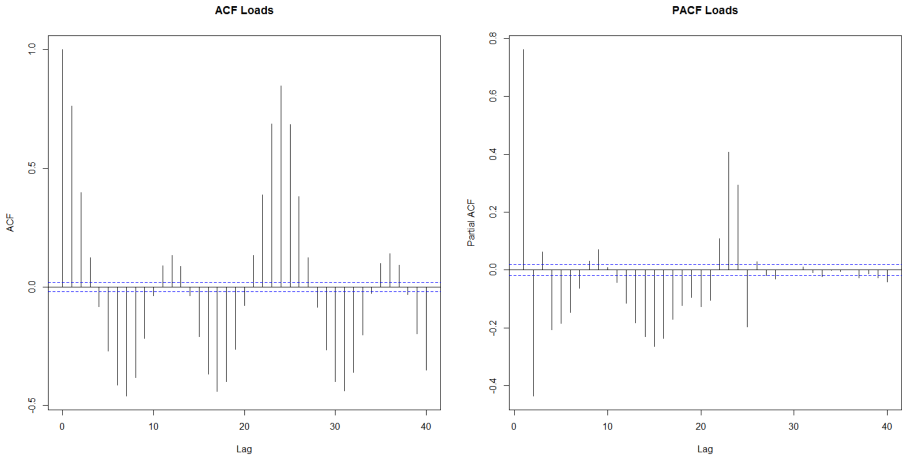

Figure 4: Shows the ACF and PACF values for the differenced log transformed data for the first date range of zone 1

By using the `auto.arima()` function provided by R, the algorithm finds and uses the best ARIMA model for the given time series, i.e, the parameter order that provides the lowest Bayesian information criterion (BIC). In R, one can obtain a summary of the best ARIMA model. In this case, the output was the following:

```
Series: log10(data)
ARIMA(4,1,3)

Coefficients:
         ar1      ar2     ar3      ar4      ma1      ma2      ma3
      1.7072  -1.1621  0.5679  -0.2606  -0.6993  -0.1086  -0.1276
s.e.  0.0547   0.0921  0.0613   0.0229   0.0556   0.0442   0.0317

sigma^2 estimated as 0.000333:  log likelihood=26612.67
AIC=-53209.33   AICc=-53209.32   BIC=-53151.42

Training set error measures:
                ME         RMSE        MAE         MPE          MAPE
Training set 2.2533e-05 0.01824026 0.01375096 -0.000907019 0.3242475

MASE      ACF1
0.1096394 0.001905331
```

Here, the parameters of the model are p=4, d=1 and q=3 but it is important to note it was not the case for all of the independent time series the algorithm dealt with. If the d was always equal to one, the value of p and q varied.

The following plot represents a subset of the raw data for the given time series in black and the predicted values for the following week in blue. One can see that the first predicted values reflect some of the variation from the actual time series but then the remaining values become linear quickly.

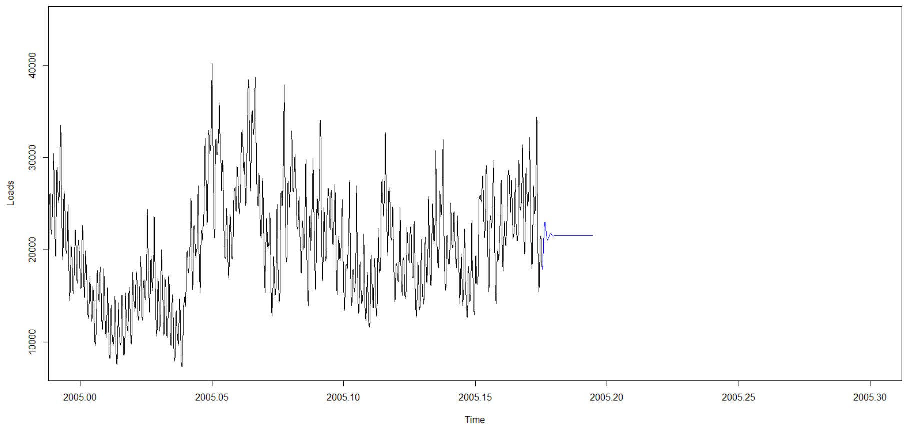

Figure 5: Shows predicted values for the week after for the first date range of zone 1

If one plots the ACF and PACF of our best fit ARIMA model, one can see that although the spikes are less pronounced, some of them are still out of the insignificant zone which means that there is still information left in the residuals and explains why the model does not produce very good results.

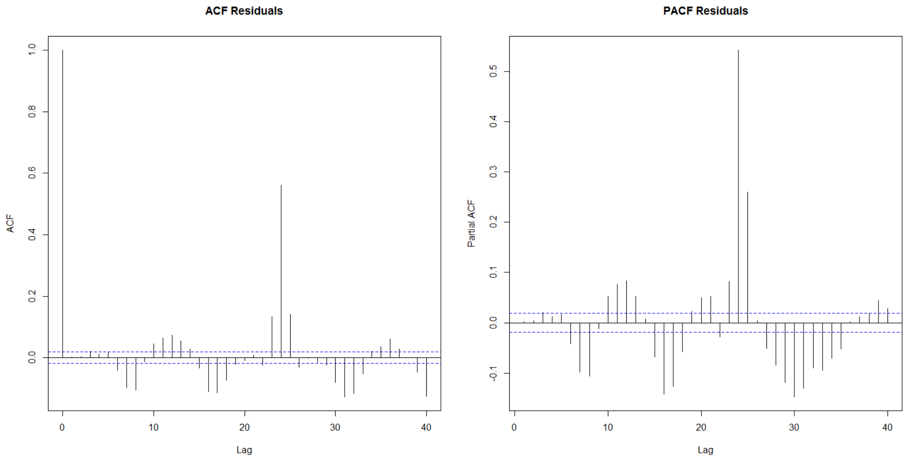

Figure 6: Shows ACF and PACF values for the ARIMA residuals for the first date range of zone 1

The results of the implemented ARIMA model for energy loads could come from several reasons. First, the fact that the algorithm dealt with several independent time series for the same zone in order to deal with missing values might have result in a loss of information. As said before, the best fit ARIMA model was not the same for each independent time series. Without the missing values the algorithm would have dealt with only one time series per zone and it would have been possible to see the bigger picture instead of focusing one smaller ranges.

The second reason why the ARIMA did not work very well might be because the data were too detailed. Indeed, measuring the energy loads hourly over four years produces a lot of information and a lot of white noise which could explain why there was information left in the residuals of the best fit model. Finally, some time series are simply not suitable for ARIMA modelling and some other approaches need to be considered.

ARIMA for temperature forecasting

The second try of ARIMA modelling was for the temperatures predictions. In addition to the energy data set, the temperature data set provided hourly temperature measurements for 11 different stations from 2004-01-01 to 2008-06-30 without missing values. This data set was used in other algorithms with the temperatures as features of the value to predict (energy load). One of the weeks for which to predict the energy loads was the one at the end of the data set (2008-07-01 to 2008-07-01). The temperatures for this week were not provided, thus it was necessary to predicted them in order to pass them to our energy loads predictive algorithms.

This time, the implemented ARIMA model dealt with only one time series per zone which was less tedious than for the energy loads model.

The results of the ARIMA modelling were constant across the different zones. As an example are presented the results for the station number 1. The following charts represents the raw temperature data with the very clear yearly pattern but no noticeable trend.

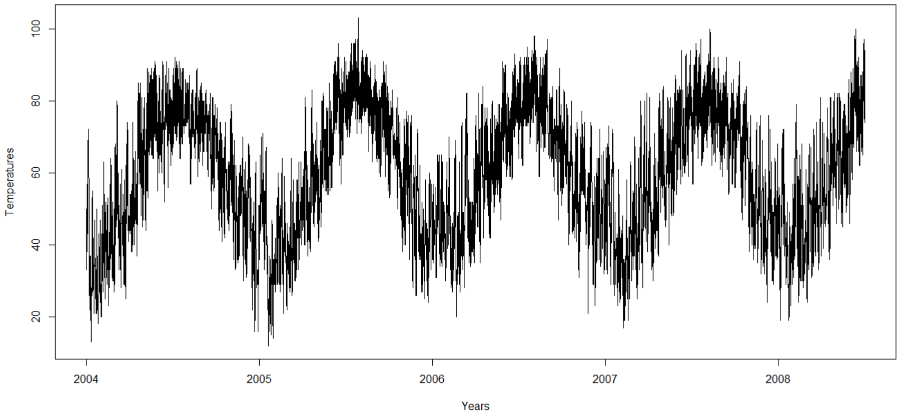

Figure 7: Shows the raw temperature data for station 1

Then the time series is made stationary with 1st order differencing and log transformation. The results are shown in the following plot.

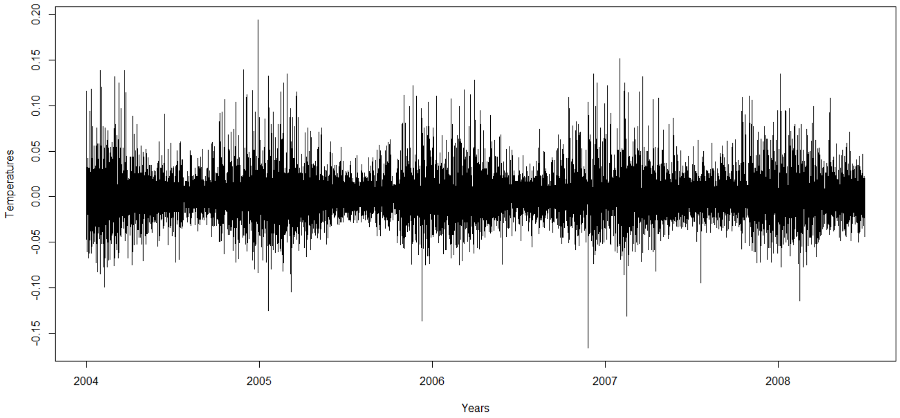

Figure 8: Shows the 1st order differenced log transformed temperature data for station 1

The Dickey-Fuller test was used to check if the transformed series was indeed stationary with following output:

```
Test Statistic: -33.513700
p-value: 0.000000
#Lags Used: 54.000000
Number of Observations Used: 39358.000000
Critical Value 5\%:  -2.861613
Critical Value 1\%: -3.430516
Critical Value 10\%: -2.566809
```

The test statistic is smaller than the critical value of 1% so one can assume that the series is stationary.

On the following charts one can observe the ACF and PACF values of the series. Here as well, several spikes are going out of the insignificant zone which means there is information to exploit for the ARIMA model.

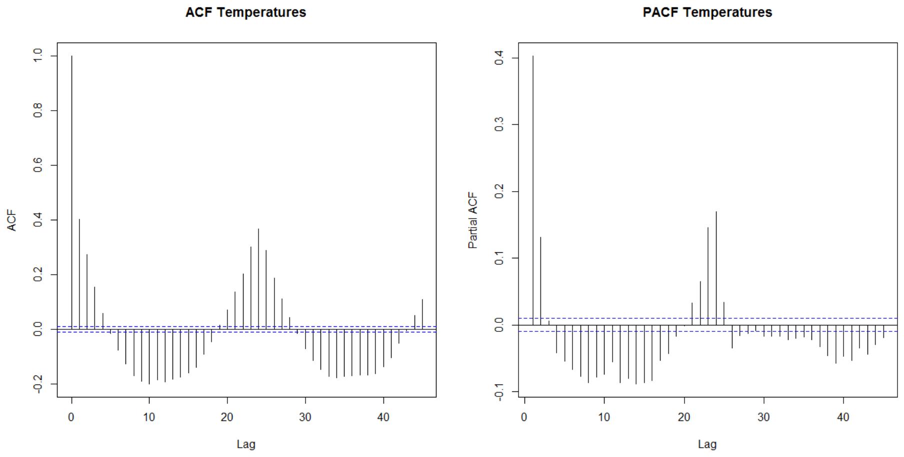

Figure 9: Shows the ACF and PACF values for station 1

Like for the energy loads time series the auto.arima() R function gives the best fit model for the given time series, the summary of this model is the following output:

```
Series: log10(data)
ARIMA(2,1,2)

Coefficients:
         ar1      ar2      ma1     ma2
      1.7959  -0.8652  -1.5442  0.5978
s.e.  0.0072   0.0071   0.0128  0.0128

sigma^2 estimated as 0.0002299:  log likelihood=109173.7
AIC=-218337.4   AICc=-218337.4   BIC=-218294.4

Training set error measures:
                ME            RMSE        MAE         MPE           MAPE
Training set 8.343304e-06 0.01516216 0.01011681 -0.004802065 0.596134
MASE      ACF1
0.1328162 0.03907321
```

The parameters of the model are p=2, d=1 and q=2.

The following plot represents a subset of the raw data for the given temperature time series in black and the predicted values for the following week in blue. One can see that, just like the predictions of energy loads, the first predicted values reflect some of the variation from the actual data but then the remaining values become linear quickly.

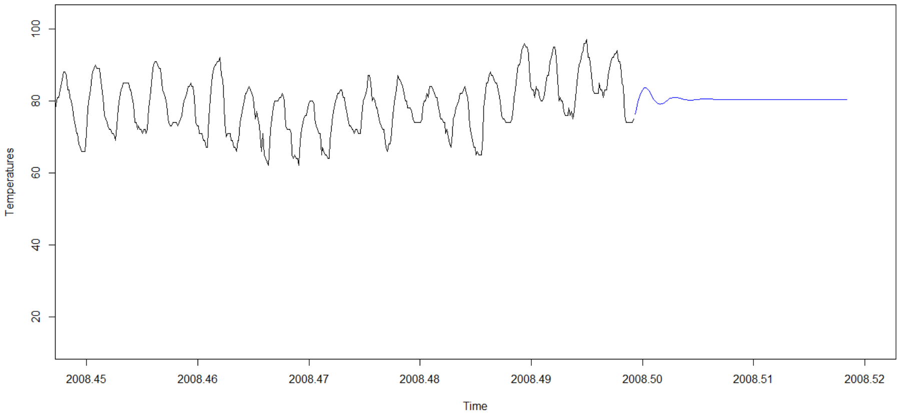

Figure 10: Shows predicted values for the week after the end of the data set for station 1

This last plot shows the ACF and PACF values of the residuals of the ARIMA model. One can see that a lot of spikes are going out of the insignificant zone which means there is information left to extract and that the best fit ARIMA model is not working well.

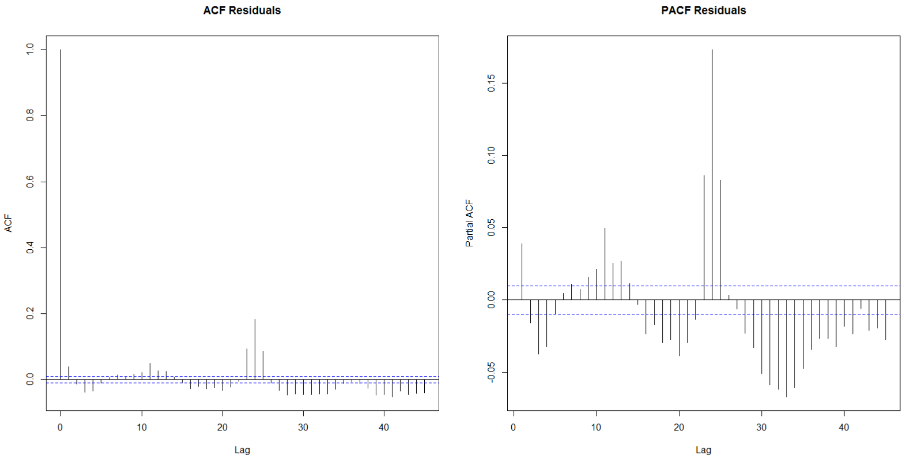

Figure 11: Shows ACF and PACF values for the ARIMA residuals for station 1

As it has been said before, maybe the ARIMA did not work very well because the data were too detailed. Indeed, measuring the temperatures hourly over four years produces a lot of information and a lot of white noise which could explain why there was information left in the residuals of the best fit model. This time series might not be a good fit for ARIMA modelling.

0.1 Reflection on ARIMA and seasonality

ARIMA is a powerful and widely used technique for time series forecasting. In our consideration here, we have had some success in processing timeseries data to accommodate the impact of autocorrelation between time periods. However, we have not fully accommodated the multiple seasonal effects within the data (present at both a yearly and daily frequency in temperature data, and at a yearly, weekly and daily frequency within load data).

Extensions to ARIMA can account for seasonal effects. These haven't been fully explored here. Indeed, we were not able to easily identify suitable libraries which could assist with the multiple-frequency seasonality patterns that we identified within these data series. Incorporating such extensions would be an important enhancement of this work.

Multiple Linear Regression

Multiple Linear Regression is a technique which finds the best fitting linear relationship between several input variables and a scalar output variable - in this case Electricity Load. The best fit is determined as that linear combination of candidate features which minimises the Mean Square Error between predictions and true values.

As has been seen earlier, there is clearly not a direct linear relationship between the raw available inputs and electricity load. We introduced non-linear elements by construction of polynomial and cross-features, and allow these features to be combined in the regression model.

Model design

We replicated the Benchmark model proposed by Tao Hong as convener of the Kaggle competition. The inclusion of 3rd ordered polynomial terms for temperature captures the non-linear relationship between load and temperature. One-hot terms are introduced to accommodate the seasonal elements of the time series - cycles of variation are observed over daily, weekly and yearly periods - captured by the Hour (of day), Day (of week) and Month (of year) terms. The Trend term is linear in time over the training and forecast period.

$$\begin{aligned} Y = \beta_0 &+ \beta_1 \times Trend + \beta_2 \times Day \times Hour + \beta_3 \times Month \\ &+ \beta_4 \times Month \times T + \beta_5 \times Month \times T^2 + \beta_6 \times Month \times T^3 \\ &+ \beta_7 \times Hour \times T + \beta_8 \times Hour \times T^2 + \beta_9 \times Hour \times T^3 \end{aligned}$$

We are provided with temperature readings for 11 different locations. Consistent with Hong's benchmark model, we select the readings from just one temperature station to use in generating forecasts for each zone. These selections are made by first running models for each combination of zone and temperature station, and selecting the temperature station which provides best fit training predictions for each zone. To assess best fit, we consider the R^2 statistic (which is the default scoring mechanism for sci-kit learn's implementation of Linear Regression). This statistic measures the proportion of variability in the time series which is explained by the model - which we seek to maximise.

We use estimated temperature data to fill in for missing data over the prediction period. In common with Hong's benchmark model, we estimate these data by taking averages over available history - using mean temperature for the same station, day of year and hour of day over previous years. We also used temperature estimates from the ARIMA model discussed above - these results are presented in the final results section.

Results

The model delivered an overall Weighted Root Mean Square Error on test data of 98,877 (using historic mean temperature estimates). This is very close to that reported by Hong for his baseline model.

The output below shows the temperature station selected for each zone, along with R^2 statistics over both training and test data:

```
zone =  1  tempstn = 10  train R2 = 0.91454  test R2 = 0.85273
zone =  2  tempstn =  9  train R2 = 0.90491  test R2 = 0.83671
zone =  3  tempstn =  9  train R2 = 0.90491  test R2 = 0.83671
zone =  4  tempstn =  2  train R2 = 0.86619  test R2 = 0.78987
zone =  5  tempstn =  9  train R2 = 0.91561  test R2 = 0.84053
zone =  6  tempstn =  9  train R2 = 0.90898  test R2 = 0.84415
zone =  7  tempstn =  9  train R2 = 0.90491  test R2 = 0.83671
zone =  8  tempstn = 11  train R2 = 0.91704  test R2 = 0.83712
zone =  9  tempstn =  6  train R2 = 0.24259  test R2 = 0.35444
zone = 10  tempstn =  1  train R2 = 0.49582  test R2 = 0.42738
zone = 11  tempstn =  3  train R2 = 0.93155  test R2 = 0.90417
zone = 12  tempstn =  5  train R2 = 0.93978  test R2 = 0.90279
zone = 13  tempstn =  2  train R2 = 0.89164  test R2 = 0.81793
zone = 14  tempstn =  4  train R2 = 0.90972  test R2 = 0.84503
zone = 15  tempstn =  6  train R2 = 0.89341  test R2 = 0.83153
zone = 16  tempstn =  7  train R2 = 0.92038  test R2 = 0.88541
zone = 17  tempstn =  4  train R2 = 0.91826  test R2 = 0.84122
zone = 18  tempstn =  7  train R2 = 0.93430  test R2 = 0.89398
zone = 19  tempstn = 10  train R2 = 0.91545  test R2 = 0.85268
zone = 20  tempstn = 11  train R2 = 0.90992  test R2 = 0.85622

Root Mean Square Error (zone level only), test: 10947.36093
Weighted Root Mean Square Error (including system level), test: 98877.18172
```

It can be seen that the model provides a good fit for most zones, with some element of over-fitting evidenced by the difference between results on training and test data.

There are some zones, notably zone 9 and zone 10, where the R^2 statistics are considerably lower. The patterns of energy use in these zones are different from the norm - reflecting usage which is less subject to variation corresponding with temperature and time of day/year. This may be indicative of energy usage in these zones being dominated by industrial rather than commercial or residential users. It should be noted that R^2 is a measure of the proportion of variance explained, rather than absolute discrepancy between predicted and actual values. In the case of these zones there is less variance over time, and so despite the low R^2 values, the overall impact on our key evaluation metric of root mean square error may be minimal.

Implementation

The multiple regression model is implemented using the Pipeline and Feature Union structures of the scikit-learn library for Python. These flexible structures allow for the piecewise extraction and transformation of different features, which can then be passed through to different machine learning models. While we have not explored many variations on feature selection and model choice in the course of this work, the implementation approach we have used means that further exploration of such changes would be straightforward to implement.

Gradient Boosting

Gradient Boosting is an ensemble learning method which builds additive regression models in a forward stage-wise manner. In each stage it fits a parameterised base learner to the gradient of the loss function which decreases in regard to the model values of every training sample. Gradient boosting can handle and process various data types and many different features. The factor of randomisation of gradient boosting also improves its accuracy and execution speed. Moreover, through its robust loss functions it can eliminate noise by removing outliers and thus, prevent the training model from overfitting to the data samples.

In our approach, a set of features was introduced to the gradient boosting regression model in order to best fit the data. The features are composed by the month (1-12), weekday (1-7), hour (1-24), summer (true-false), trend, holiday falling and the temperature measurements of each of the 11 temperature stations. After training the model the root mean squared error of the testing was generated and was approximately equal to 76,676 whilst its weighted root mean squared error (incorporating predictions at system as well as zonal level) was approximately equivalent to 112,081. Finally, the error of the training set and test set was computed at each iteration and can be summarised in Figure 12, whilst the importance of all features was measured and can be observed in Figure 13.

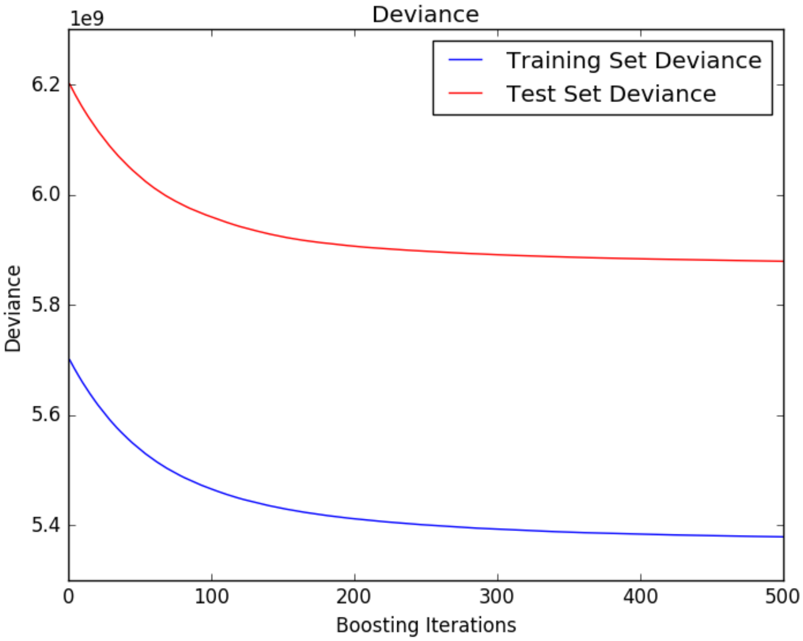

Figure 12: Shows the deviance of the training and test set in gradient boosting

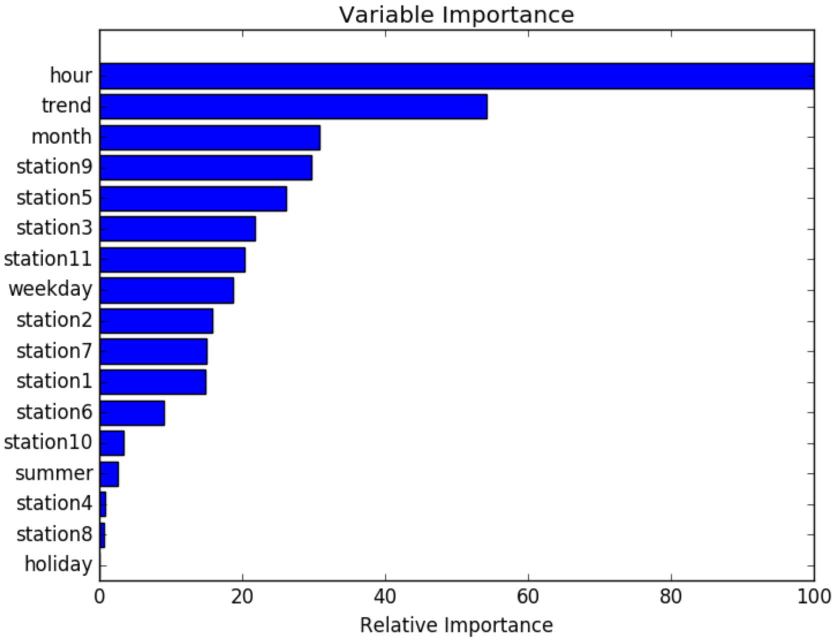

Figure 13: Shows the feature importance measurements over the features in gradient boosting

Neural Networks

Neural networks can be used to tackle regression problems. By including multiple connected hidden layers, neural nets can can effectively learn complex non-linear relationships between explanatory variables. Training of neural networks is computationally expensive, requiring extended training runs during which back-propagation techniques are used to modify weight parameters in a direction that reduces our loss function (a measure of absolute difference between predicted and true values).

Model design

Our neural network includes three densely connected hidden layers, consisting of 304, 152 and 38 nodes respectively. Between the first two layers we used a $tanh$ activation function, with outputs from the final layer then being mapped to a single output node which was trained against actual load values.

We trained the model against a Mean Square Error loss function using batch training and optimising ADAM, a stochastic gradient descent method.

The network we built is relatively deep considering the scope of the problem, and so there was a risk of overfitting during training. We tackled this by including drop-out layers between the hidden layers. Drop-out mitigates against overfitting, by randomly omitting a proportion of nodes at each epoch of training, preventing individual routes through the network from becoming overly specialized, and encouraging a more generalized solution.

There are many parameters that can be tuned in the configuration of neural networks, and in many respects the theory guiding appropriate choice of parameters and network topology is still under development. We achieved competitive results with this network design, but clearly there is great scope to further refine the model through cross-validation.

Implementation

We used the Python library Keras to construct the neural network. Keras provides a suite of tools to quickly construct neural networks, which can then be executed using GPU hardware to run calculations in parallel.

The model was defined as follows:

```
model = Sequential()
model.add(Dense(input_dim*4, input_dim=input_dim))
model.add(Activation(’tanh’))
model.add(Dropout(0.5))
model.add(Dense(input_dim*2))
model.add(Activation(’tanh’))
model.add(Dropout(0.5))
model.add(Dense(38))
model.add(Dense(1))
model.compile(loss=’mse’, optimizer=’adam’)
model.fit(X, y, nb_epoch=200, batch_size=100)
```

Input variables were preprocessed using a sci-kit learn pipeline, including extracting categorical features from the date/time of measurements, allowing the model to discover relationships involving hour of day, day of week and month of year. We also incorporated whether the day was a US holiday. In contrast to the Multiple Linear Regression approach, temperature readings from all 11 weather stations were available for use in predicting for each zone.

After applying one-hot transformations to the categorical variables in the input data, there was a total of 78 input variables.

The model was then run for 200 epochs, carrying out ADAM optimisation over batch sizes of 100 records.

Comparative Results

The table below shows the comparative results achieved for a selection of our models. Note that RMSE is a simple (unweighted) Root Mean Square Error across all zonal predictions for the test periods. WRMSE is a Weighted Root Mean Square Error including errors on system level (i.e. sum across zones), with additional weighting on system level and unseen period forecasts.

Along with results from using temperature estimates from both the historic mean and ARIMA approaches, we also present results from using actual temperature measures for the forecast period. These actuals were released after the conclusion of the Kaggle competition.

Method	Source of temperature estimates	RMSE	WRMSE
Baseline - Linear Regression	Historic mean	10,947	98,877
	ARIMA	12,027	154,804
	Actuals	10,226	78,098
ARIMA for loads			280,608
Gradient Boosting	Historic mean	76,676	112,081
Neural Network	Historic mean	12,712	87,731
Neural Network	ARIMA	14,114	165,947
Neural Network	Actuals	12,235	77,363

Table 1: Summarises the Root Mean Square Error and the Weighted Root Mean Square Error of each technique

Our best result was achieved using a Neural Network incorporating temperature estimates based on means observed during the historic dataset. The WRMSE score of 87,731 compares well with those achieved during the Kaggle competition - this result would have placed in the top 20 on the final leaderboard.

Conclusions

We explored several different techniques for forecasting energy usage, given inputs limited to date/time and temperature readings from weather stations of unknown geographic location.

We discovered that there were strong non-linear relationships between explanatory variables and the measure of interest.

We successfully replicated the benchmark model proposed for the 2012 Kaggle competition using this data.

We considered several more sophisticated approaches to improve on this benchmark model, including consideration of ARIMA techniques to forecast both energy load directly, and to predict future temperature patterns as an input to other modelling techniques. While this technique did not ultimately add value over more naive approaches in our implementation, we consider that there is great potential for this approach, if the multi-level seasonality within the data can be more fully accommodated.

We showed the potential of Deep Learning / Neural Network techniques as applied to regression problems of this type, achieving results that were on a par with the best results achieved during the Kaggle competition.

We used Machine Learning libraries developed for Python to construct flexible models that can easily be developed to explore other parameter settings and incorporation of additional features.

Potential areas for future work include the refinement of ARIMA or other time series forecasting techniques to provide more robust esitmates of temperature trends (particularly modelling the highly seasonal aspects of this data), and exploration of alternative neural network topologies and parameter settings through cross-validation.

References

Kaggle Global Energy Forecasting Competition 2012
`https://www.kaggle.com/c/global-energy-forecasting-competition-2012-load-forecasting`

Chatfield, Chris. Time-series forecasting. CRC Press, 2000.

Friedman, Jerome H. ”Stochastic gradient boosting.” Computational Statistics & Data Analysis 38.4 (2002): 367-378

Hong, Tao, Pierre Pinson, and Shu Fan. ”Global energy forecasting competition 2012.” International Journal of Forecasting 30.2 (2014): 357-363.

Taieb, Souhaib Ben, and Rob J. Hyndman. ”A gradient boosting approach to the Kaggle load forecasting competition.” International Journal of Forecasting 30.2 (2014): 382-394.

Hon, Tao. ”Short Term Electric Load Forecasting”
`http://www.lib.ncsu.edu/resolver/1840.16/6457`

Kingma Diederik, P. and Lei Ba, Jimmy. ”ADAM: A method for stochastic optimization”
`http://arxiv.org/pdf/1412.6980.pdf`