Signal Processing **Michaelmas 2014**

Time Series Forecasting

Student: Ankush Gupta

CO_2 Level Forecasting

I followed two approaches for forecasting this time series — (1) decomposing the series into long-term and short-term trends (2) using gaussian process regression.

Series Decomposition

Clearly, there are (at least) two trends in the given time series — (1) $f_L(t)$ Long term increase (2) $f_S(t)$ Short term almost sinusoidal trend. Hence, my analysis for this time series was aimed at finding these two components such that $f_{CO_2}(t) \approx f_L(t) + f_S(t)$

Fitting Long Term Trend

Since, many natural phenomena follow the exponential trend because of their dynamics being $\frac{dx}{dt} \propto \alpha x$, I first tried fitting an exponential curve $f_L(t) = \beta e^{\alpha t}$, by taking logs and using ordinary least squares. The fit didn't look right [1], so I tried decomposing it in terms of the polynomial basis $\{1, x, x^2, x^3, ...\}$ using least-squares.

To get rid of the sinusoidal trend, I used a low-pass filter to uncover the long-term trend [2] (more specifically, I used a Butterworth filter with the cut-off frequency determined heuristically and then applied the filter using `filtfilt` to get rid of phase-shift effects). Then I found the coefficients by regressing the curve on the polynomial basis. Using cross-validation, it was found that degree 2 basis basis $\{1, x, x^2\}$ gave the best fit and *generalisation* into the future.

In conclusion, the long term trend was determined to be $\boxed{f_L(t) = a_0 + a_1 x + a_2 x^2}$

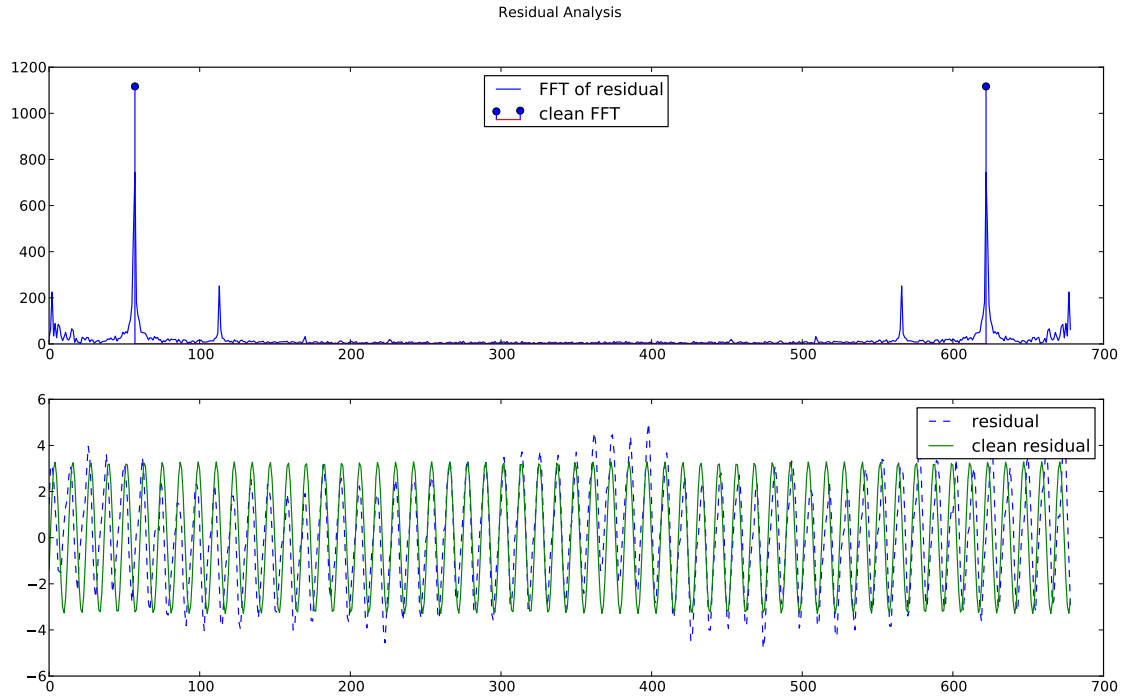
Fitting the Short Term (almost) Sinusoidal Trend

The short-term trend was modelled as a pure sinusoidal i.e., $f_S(t) = f_{CO_2} - f_L(t)$ was modelled as a sine wave of **one** frequency. This decision of using just one frequency was based on analysing the spectrum of the residual $f_{CO_2} - f_L(t)$ (Figure 1.1a). As the maximum peak is relatively bigger than the rest of coefficients, only one frequency was used to model the residual.

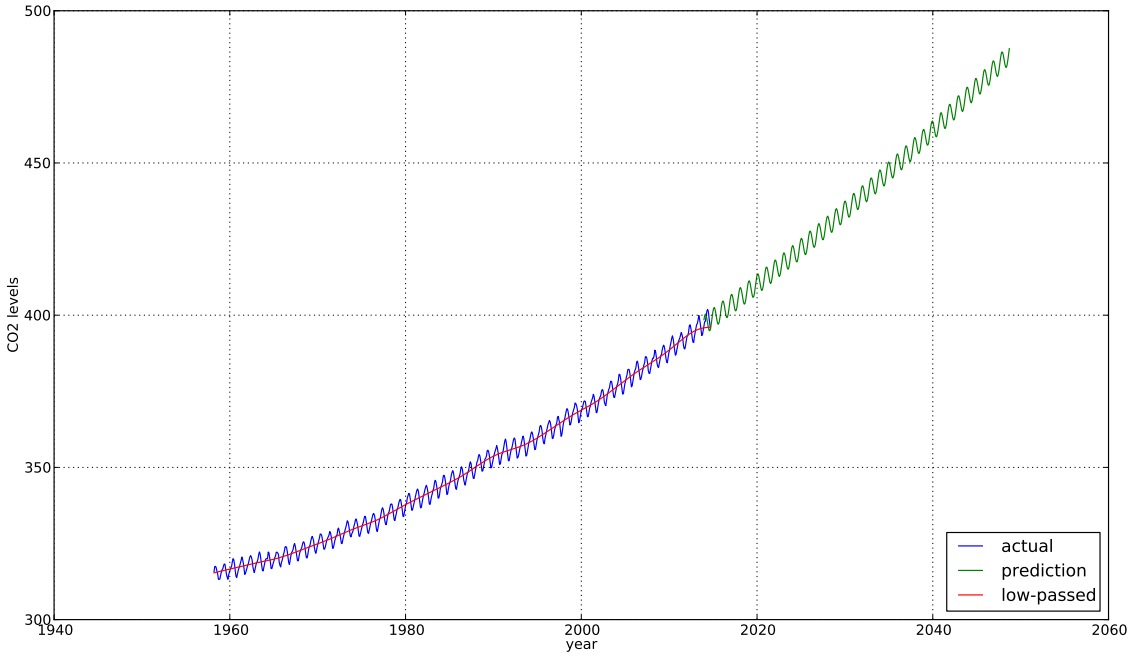
Figure 1.1b shows the final result — predictions up to the year 2050. Also shown is the low-passed version of the original signal used for fitting the long-term trend.

[1] which was quite surprising for me.

[2] Note: I could have used the raw-curve (without any pre-processing/ filtering) to find coefficients of the polynomial basis elements.



(a) Spectrum of the residual (short-term trend) = $f_{CO_2}(t) - f_L(t)$.



(b) Prediction of CO_2 time series data.

Gaussian Process Regression for CO_2

The code written for the *Data Estimation and Inference* course was used here. The polynomial fit for the long-term trend was used as the mean-function. For the covariance, I analysed two Kernels — (1) Square-Exponential = $\sigma^2 exp\left(-(x_1 - x_2)^2/2a\right) + \sigma_y^2 I$ and, (2) Periodic = $\sigma^2 exp\left(-l^2 \sin^2(2\pi/b|x_1 - x_2|)\right)$. The hyper-parameters were optimised using a combination of conjugate gradient methods and human-in-the-loop-stochastic-gradient-descent.

Figure 1.2 shows the results (please see the figure caption for discussion).

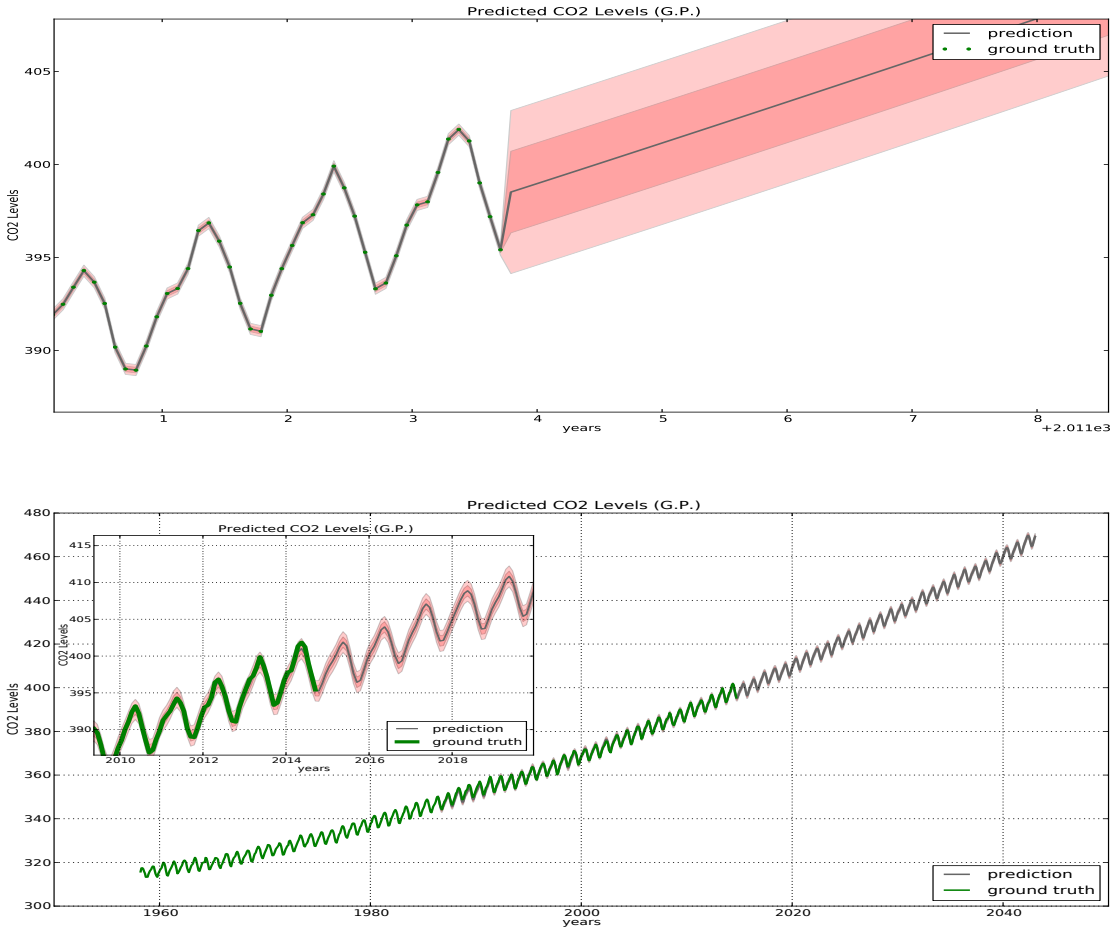


Figure 1.2: [Top] GPR using the square-exponential kernel. Note the high-uncertainty in prediction. The future predictions basically just track the mean function. [Bottom] GPR prediction using periodic kernel. Since this periodic kernel captures the periodicity in the data, the future predictions have low uncertainty (inset). There is some finite variance at the points where observations are available because an observation uncertainty term was added to the kernel to make it numerically robust. The two red intervals indicate regions of one and two standard deviations.

Sunspots Activity Forecasting

This Sunspots Activity series was forecasted using two techniques — (1) Using $AR(p)$ models trained using Yule-Walker equations and, (2) using 1-hidden layer feed-forward neural network regression with p lagged inputs, trained using back-propagation.

A neural-network was used, because auto-regressive modes can be seen as a one linear neuron (perceptron) network. Hence, I was curious whether adding further complexity to the model improves the prediction accuracy. The neural network architecture was heuristically [3] chosen to have one-hidden layer, with p *tanh* input units, $p/2$ *tanh* hidden units and one linear output unit. The inputs to the network were p delayed values of the signal $\{x[t-p], \ldots, x[t-1]\}$ and the output was trained to match $x[t]$. The data was normalised to have a mean of zero and a maximum value of one to avoid network saturation.

Both the AR model and the neural-network were trained for one-step predictions. However, 12 sequential predictions (corresponding to one full year) were made using rolling regression in calculating the error rate. Figure 1.3 below shows the prediction output of $AR(15)$ model.

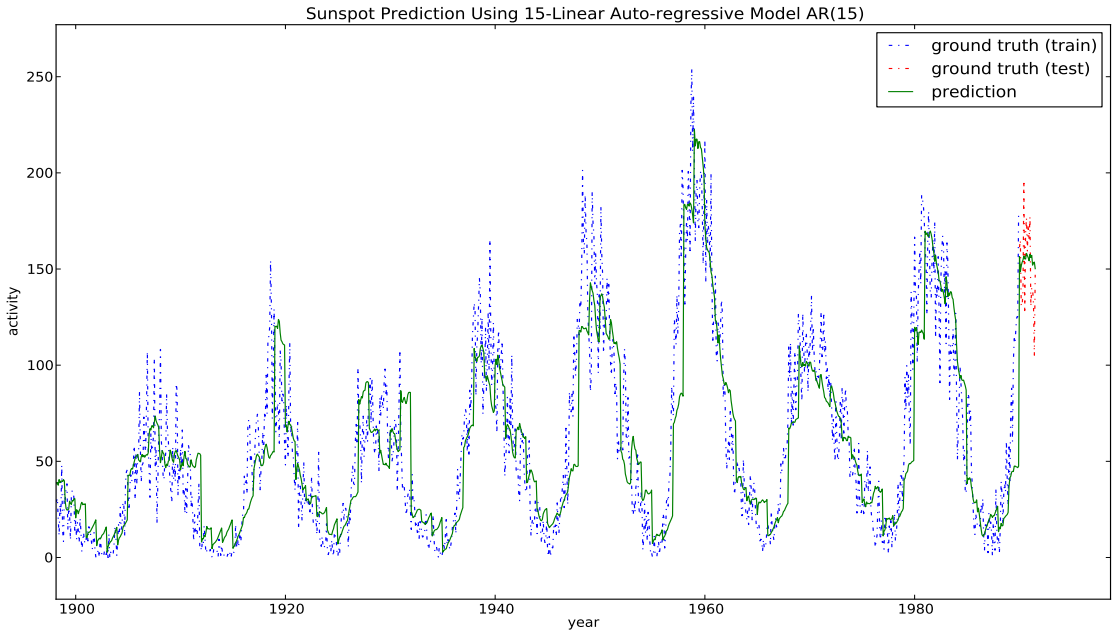


Figure 1.3: Sunspot data prediction using Auto-Regressive ($p = 15$) model.

[3] keeping in mind the size of training data available and the time required to train these networks.

Figure 1.4 and compares the accuracy of the AR model and neural-network for increasing values of p. We note that for small p neural network tracks the error rate of the AR models but tends to overfit and do worse for larger $p > 100$.

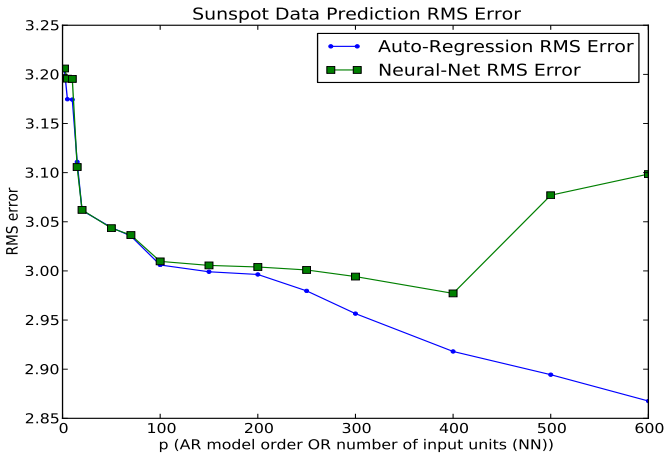


Figure 1.4: Sunspot data prediction using Auto-Regressive ($p = 15$) model.

Both AR models and NN (for regression) invariably introduce some "lag", i.e., more concretely, the predictions of these models is better matched if is shifted in time by some small amount. To quantify this shift, I found the peak of the correlation function between the predicted signal and the actual signal. Figure 1.5 shows the correlation function and peak for AR(15) model. Table 1.1 shows the amount of shift for optimum matching as a function of p for AR models.

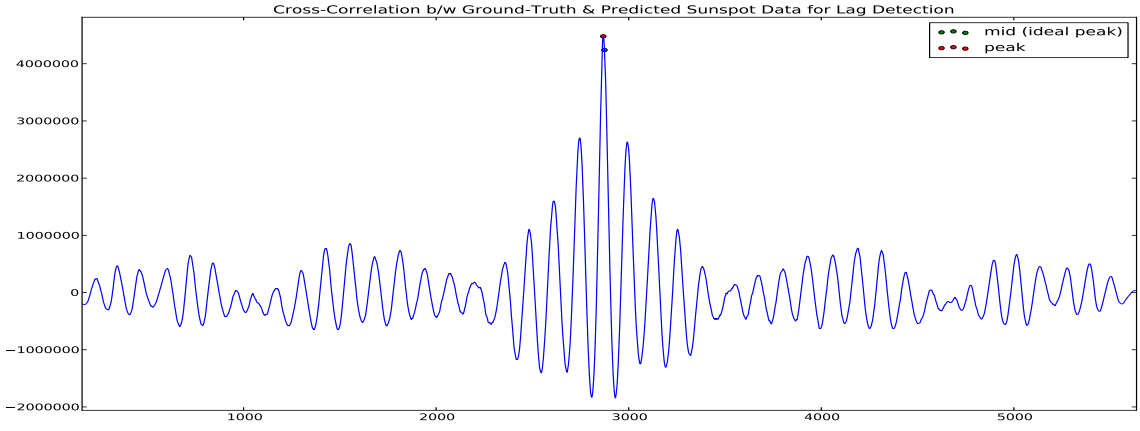


Figure 1.5: Correlation between prediction and the actual signal for detecting lag.

p	2	5	10	15	20	70	100	300	500	600
Lag	-5	-6	-6	-6	-5	-4	-4	-3	0	0

Table 1.1: Lag for best correlation (or matching) as a function of p.

Financial Data

A distinctive feature of this data-set is that it is discrete[4] — the financial data goes up or down in discrete units. Hence, predicting this time series can be seen as a classification problem instead of a regression problem — given the past data, predict whether the change in the next tick will be $\{-4, \ldots, 0, \ldots, 4\}$.

All classification problems rely on find good features for their solution. Given the time constraints and my past (good) experience with using them for MNIST digit classification, I once again tried using neural networks[5], but with little success. I tried a bunch of different architectures but none could do better than random guessing — i.e. an error rate better than $1/|\{-4, \ldots, 0, \ldots, 4\}| = 1/9$. Figure 1.6 shows the error-rate from a typical run — we note that the neural network overfits to the training data and generalises poorly to the test-data. Perhaps, techniques to avoid overfitting would have helped here.

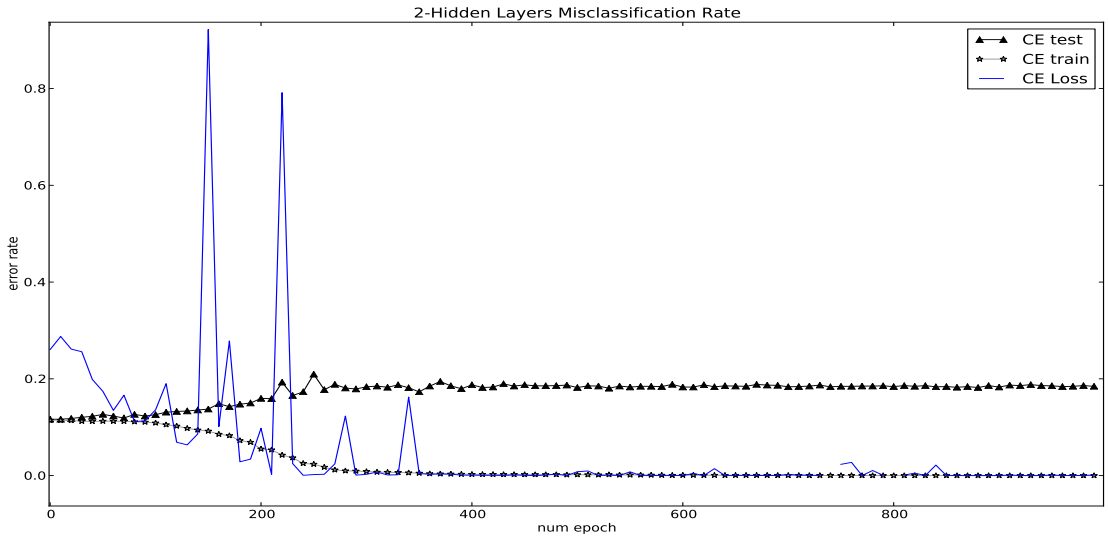


Figure 1.6: Cross-Entropy loss function error rate for a feed-forward neural network on the financial time series data.

[4]Thanks to James Thewlis and the TAs for this insight.
[5]neural-networks = classification sledge-hammers?

Mackey-Glass Time Series

Unfortunately, I could not find the time to really delve into this series. While reading about solving this problem I came across the ideas of recovering embedding dimension and attractors — things I would love to explore further in the future.

Code

All the code was written in python. No time-series-forecasting/ machine-learning libraries were used. All the predictors — AR model, gaussian process regression, neural nets were written by me and the source code is attached with this report. Instructions on how to run the code are present in `README.txt`

Thank You!