Computer Assignment 1: Supervised Learning for Linear Prediction

Christopher Robbiano — Due: March 6, 2018 — ECE656

1 Introduction

The goal of this computer assignment is to investigate how supervised learning can be used to estimate the unknown parameters of an Nth order linear autoregressive (AR) model used as a linear predictor using the data directly. This assignment was completed in Matlab, and used data from the S&P 500 futures index from the years 2015, 2016 and 2017.

2 Theory

We use a 3rd order autoregressive model (AR) to predict the futures price using the following relationship

$$y(n) = a_1 y(n-1) + a_2 y(n-2) + \ldots + a_N y(n-N) + e(n) \tag{1}$$

where N is the order of the AR process, the driving term $e(n) \sim \mathcal{N}(0, \sigma_e^2)$ and the a_i's are real valued coefficients that need to be estimated through training. The optimal minimum mean squared error solution (MMSE), known as the Weiner-Hopf (WH) solution, for the optimal weight vector $\mathbf{w}^* = \begin{bmatrix} a_1^* & a_2^* & \ldots & a_N^* \end{bmatrix}$ is given by $\mathbf{w}^* = \mathbf{R}_{xx}^{-1}\mathbf{R}_{xd}$. The WH solution assumes that the process is ergodic.

Two learning rules are compared in this report. The first is the least mean squares (LMS), and the second is the recursive least squares (RLS). Both learning rules assume a linear neuron model, where the output of the model is $o(k) = \mathbf{w}(k)^T\mathbf{x}(k)$, and the typical weight update equation of $\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w}$. In the case of both the RLS and LMS learning rules, it can be shown that the learned weight vectors converge to the WH solution under certain conditions.

2.1 Least Mean Squares

The least mean squares algorithm uses a squared error cost function of $J\big(\mathbf{w}(k)\big) = \big(d(k) - \mathbf{w}(k)^T\mathbf{x}(k)\big)^2$ and borrows the general gradient descent learning rule to form the weight update equation of $\mathbf{w}(k+1) = \mathbf{w}(k) + \mu\mathbf{x}(k)\mathbf{x}(k)^T\mathbf{w}(k)$.

2.2 Recursive Least Squares

The recursive least squares algorithm uses a sum of squared error cost function with a forgetting factor. The cost function is given by $J\big(\mathbf{w}(k)\big) = \frac{1}{2}\sum_{i=1}^{k}\gamma^{k-i}\big(d(i) - o(i)\big)^2$. The output is passed through a threshold-logic activation function such that

$$o(k) = f\big(\mathbf{w}(k)^T\mathbf{x}(k)\big) = \begin{cases} 0 & \mathbf{w}(k)^T\mathbf{x}(k) \le 0 \\ \frac{\mathbf{w}(k)^T\mathbf{x}(k)}{a} & 0 < \mathbf{w}(k)^T\mathbf{x}(k) < a \\ 1 & \mathbf{w}(k)^T\mathbf{x}(k) > a \end{cases} \tag{2}$$

Differentiating J with respect to $\mathbf{w}$ gives the normal equation of $\frac{1}{a}\sum_{i=1}^{k}\gamma^{k-i}\mathbf{x}(i)\big(d(i) - \frac{1}{a}\mathbf{x}(i)^T\hat{\mathbf{w}}(k)\big)$. Iteratively solving this equation gives rise to the following general update equations for calculating the gain, inverse correlation matrix and finally the weight vector.

$$\mathbf{K}(k) = \frac{\mathbf{P}(k-1)\mathbf{x}(k)}{\gamma + \mathbf{x}(k)^T\mathbf{P}(k-1)\mathbf{x}(k)} : \quad \text{Gain Calculation} \tag{3}$$

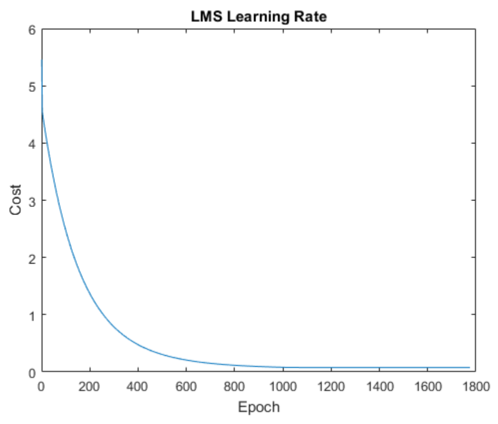
$$\mathbf{P}(k) = \frac{1}{\gamma}\big(\mathbf{I} - \mathbf{K}(k)\mathbf{x}(k)^T\big)\mathbf{P}(k-1) : \quad \text{Updating Inverse Correlation} \tag{4}$$

$$\hat{\mathbf{w}}(k) = \hat{\mathbf{w}}(k-1) + \mathbf{K}(k)\big(d(k) - \frac{1}{a}\mathbf{x}(k)^T\hat{\mathbf{w}}(k-1)\big) : \quad \text{Weight Updating} \tag{5}$$

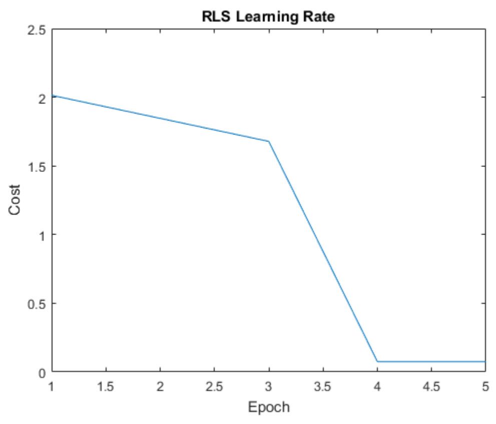
3 Data Processing and Training

Prior to training, the price data was normalized to have a mean of 0 and a variance of 1. This reduced the number of epochs required to reach convergence of both algorithms and allowed for a larger learning rate in each case. The data was split into a training and testing set, with the training data being composed of one third of the original data and the testing data being composed of the remaining two thirds.

The weight update equations were iterated on until the change in the cost function from iteration k to iteration $k+1$ was less than some predetermined threshold Υ. A couple of different values of Υ were empirically tested and it was found that $\Upsilon = 10^{-8}$ worked well. The learning rate for each learning rule is presented in Figures 1a and 1b. It is clear that the RLS learning rule converged to a solution *much* quicker than the LMS learning rule, with RLS converging after just 4 iterations while LMS took over 1,700.



(a) LMS learning rate



(b) RLS learning rate

Figure 1: Learning rates for different learning rules

3.1 Effect of Learning Rate on LMS

A number of different learning rates μ were used while training the LMS algorithm. For any $\mu > 10^{-4}$, the weight update equation diverged instead of converged, but it wasn't until $\mu < 10^{-7}$ that the weight vector produced by the LMS algorithm actually produced a reasonable approximation to the optimal weights. As μ decreased, the convergence time (required number of epochs for convergence) increased substantially. To automatically tune

μ, I generated code that would automatically increase and decreased μ depending on if the cost of the current iteration was greater than or less than the previous iteration. If the cost was greater than the previous iteration then μ would be decreased and the current iteration would be thrown out. I found that this drastically increased the convergence time while still allowing the solution accuracy of a smaller μ.

4 Results and Discussion

The result of the training produced the weight vectors $\mathbf{w}_{LMS}$ and $\mathbf{w}_{RLS}$ that were then used for the testing phase. Using those two weight vectors, predictions for futures prices were made and them compared to the predictions made with the optimal weight vector $\mathbf{w}^*$ as well as the actual futures prices. This can be seen in Figure 2. From the figure, it is apparent that the optimal prediction is just a naive prediction, where the next predicted price is generated by guessing the current price. It is clear that the RLS estimate is very close to the optimal estimate, but the LMS estimate is somewhat lacking. The results for the LMS that are presented are from a training phase that used the zero vector as an initial starting point. I was able to produce better results for the LMS algorithm by using a two phase approach. In phase one I used a random search with randomized initial weights and kept the weight vector that produced the lowest cost function. In phase two I began the line search with the randomized vector selected in phase one.

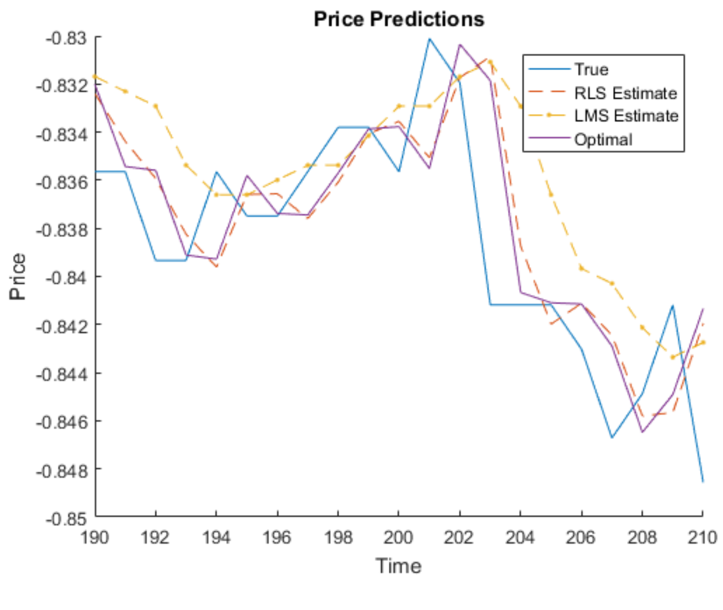
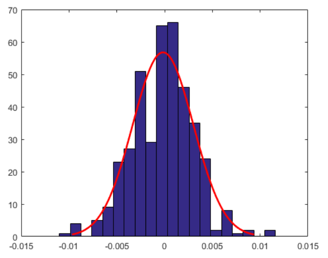
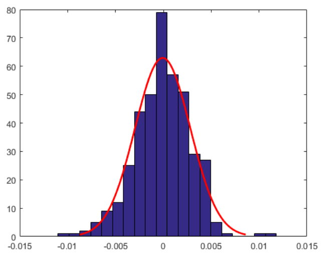


Figure 2: Predictions using different weight vectors compared to actual values

The MSE testing errors of the MMSE solution, the LMS algorithm and the RLS algorithm are $1.0079(10^{-5})$, $8.335(10^{-6})$ and $7.691(10^{-5})$, respectively. The distribution of the error from the LMS and RLS algorithms is shown in Figures 3a and 3b. It can be seen that RLS algorithm produces a slightly tighter error distribution, lending itself to be the better prediction method. The MMSE error distribution can also be seen in Figure 3c for comparison, and it can be seen that the MMSE solution has only a slightly tighter error distribution than the RLS algorithm.


(a) LMS error distribution


(b) RLS error distribution

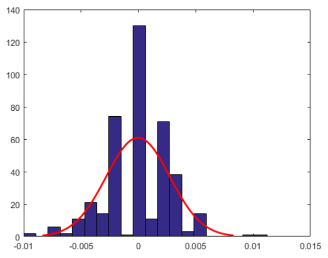

(c) MMSE error distribution

Figure 3: Error distributions for different learning rules

There are some issues associated with the assumptions made in these predictors that effect the reliability of the predictions. The first is that the data is assumed to be ergodic, which it is not. There may be small windows of time in which the data is locally stationary, but the whole dataset is not stationary. Also, the ensemble mean is most certainly not equivalent to the time average. In order to try and use this local stationarity property to better predict the futures price, an adaptive learning algorithm was used which is based on the RLS algorithm. This new algorithm adds a sliding window of P samples in time to completely forget samples of data that are too old. The cross-correlation of the data with itself provides a way to estimate how big the sliding window should be, and the window size is selected to be the number of samples before the data is no longer correlated with itself. From Figure 4, it can be seen that the data is no longer correlated after about $P = 25,000$ samples. Instead of using a training and testing phase, the new algorithm operates on-line and uses only the previous P samples and also continuously updates. Unfortunately, the changes to using a windowed weighted RLS algorithm did not produce any notable improvements in the predictions, and thus the results are not presented.

5 Conclusion

Two different learning rules were evaluated against each other as well as being evaluated against the optimal MMSE solution in this computer assignment. It was found that the RLS learning rule converged to the optimal solution after only a small number of epochs while the LMS learning rule converged to the optimal solution only after a two phase random then line search approach was taken. The LMS learning rule convergence speed was also dependent on the learning rate and the faster the convergence of the algorithm the worse the accuracy of the solution. Although both learning rules converged to the optimal solution, the optimal solution for this dataset is a naive predictor that takes the current value and guesses it for the next value. Given that at best we will be one time step behind in the data, it makes sense to not use any of these prediction methods on this data.

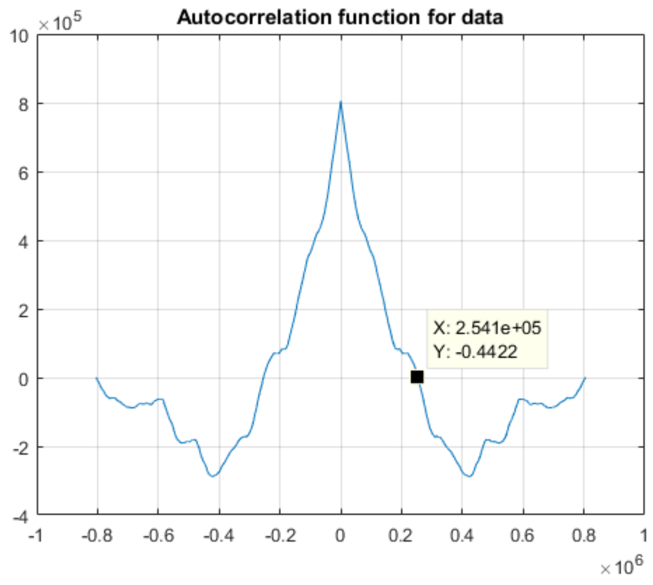


Figure 4: Cross-correlation of price data

6 References

Neural Networks and Learning Machines
ECE656 Class Notes