Contents

List of Figures

List of Tables

List of Abbreviations

1. **ACF** - Autocorrelation Function
2. **AR** - Autoregressive
3. **ARMA** - Autoregressive Moving Averages
4. **ARIMA** - Autoregressive Integrated Moving Averages
5. **AIC** - Akaike Information Criterion
6. **EWAF** - Exponentially Weighted Average Forecaster
7. **GAM** - Generalized Additive Models
8. **GBM** - Gradient Boosting Model
9. **GCV** - Generalized Cross Validation
10. **KPSS** - Kwiatkowski–Phillips–Schmidt–Shin (KPSS) tests
11. **OCO** - Online Convex Optimization
12. **OGD** - Online Gradient Descent
13. **ONS** - Online Newton Step
14. **REML** - Restricted Maximum Likelihood
15. **S&P500** - The Standard & Poor's 500

Chapter 1

Introduction

There is a rich source of literature in the field of economics and finance when it comes to predicting commodity futures. Indeed, studies have approached this topic in a multitude of ways ranging from co-movement analysis of different metal prices and commodity cycles [29], to application of standard time-series models such as ARMA [9] and GARCH [10]. Although these traditional perspectives have proven to be effective in their own right as a source of insight and prediction, they tend to fall short in exploiting the non-linear nature underlying the movement of commodity prices.

The above issue persisted until the advancement of machine learning and data mining techniques introduced new paradigms of non-linear modelling like decision trees and recurrent neural networks into the financial domain. [23], [30] and [27] utilized these methods in generating signals for trade and predicting copper spot prices. [23] compares the predictive power of linear models like ARIMA against non-linear forecasters like neural networks, and finds that latter class of models comprehensively outperform their linear counterparts. What is important to note here is that despite the change in statistical framework from linear to non-linear modelling, these approaches fall firmly in the area of offline learning. That is, we assume that the entire sequence of data is available to us. We then train our model on this batch of data and derive one final model with the least error metric on the validation set.

However, financial data usually does not come in a batch. Instead, we receive it in the form of a continuous data stream with varying frequencies (seconds, minutes, days, etc). This setting, referred to as *online* setting, poses potential problems when used in forecasting as models that are effective in the batch learning scenario may become unbounded and fail to converge to an optimal solution in the online scenario. An example of this is the ARMA model which becomes unbounded while making predictions in real time as its noise terms $\{\epsilon_t\}_{t=1}^{T}$ are missing. Anava et al.[8] proposed a solution to bound these ARMA models through a regret minimization framework where the potential solutions are limited using online convex optimization algorithms. Their proposed solution was then applied to a number of toy data sets for robustness checks, and finally applied to temperature and stock market data for real world experiments which yielded mixed success.

In this paper we will attempt to recreate the experiments of Anava et al., and then apply their predictive framework on the change in closing prices of copper and aluminium futures as listed in Shanghai Futures Exchange. We will then take our analysis into the online expert settings. Here, we will take advantage of the flexibility offered by online learning and combine the forecasting powers of several different linear and non-linear time series algorithms and check for improvement in our predictive performance. Finally, we will compare these results to the best AR and ARMA models in the batch learning setting.

The main research questions we are attempting to answer through the above setup are the following:

- Do we see convergence in error to the best time-series model in hindsight for our commodities data in the online setting, and are these results useful from a predictive standpoint?
- Can we improve the predictive performance of our online models by considering a mixture of experts?
- And finally, how do these models perform when compared to learners in

the batch setting?

Through our results we will show that despite the online models reducing error over time, their performance initially falls well below our baseline models due to some form of misspecification. However, we see vast improvements once we expand the modeling space by considering the online experts setting with performance well above our offline models.

One of the main theoretical advantages of online learning algorithms is that they make no assumptions with regards to Gaussian distribution of noise terms, and even forego the identically distributed and independence assumptions. If our models yield satisfactory or even comparable results to the models in the offline setting it would imply that we have found a more robust class of models to replace the standard time series formulations.

It is worth noting that use of online learning is not new in the field of finance with [8] and [3] having already applied it to predict S&P500 and conduct portfolio management respectively, with varying degrees of success.[1] However, to the best of our knowledge online learning has never been applied specifically to predict commodity prices. In this regard, these experiments were carried out determine the expected degree of success in the application of online learning in this niche domain.

This rest of the dissertation is laid out across 6 more chapters. The following chapter contains exploratory data analysis, assessing feature importance, and our best models in the offline setting. Chapter 3 and 4 contain the theoretical foundations of online learning for time-series analysis, along with experiments on artificial and real world data sets. Chapter 5 and 6 take our online learning to the expert setting and assess their performance on our copper and aluminium data set. Chapter 7 contains our concluding remarks. It is important to note

[1] [8] did not see success in applying online ARMA models to S&P 500 data with the online model performing poorly in terms of reducing mean square error. [3] looked at online investment algorithms and achieved almost the same result as the best constant-rebalanced portfolio in hindsight.

that this study does not have a dedicated literature review. Instead we cover the literature throughout as and when new concepts are introduced. For this dissertation we assume that the reader has familiarity with basic time series and machine learning concepts like ARMA models, random forests, splines, and gradient boosting machines. The bulk of the programming is done in R using the caret, forecast, mgcv. The expert settings are modelled using R's opera packages. Replication of experiments in [8] has been done in python using scikit learn, and numpy libraries.

Chapter 2

Exploratory Data Analysis

2.1 Origin of data

The copper and aluminium futures data used in this study are drawn from the Shanghai Futures Exchange. This data set contains information with regards to the daily opening price, closing price, and intraday high and low prices, along with volume of traded contracts. Of these variables, only closing price of the futures contract is of interest to us. Both the copper and aluminium database contain a total of 3000 entries starting from February 2006 to May 2018. As mentioned earlier, the frequency of observation is on a day-by-day basis with no observations for weekends due to market holidays. There is also no missing data to report in our variable of interest.

At this point we make some important observations and clarify certain terminology related to our database. To begin with, a futures contract represents a legal agreement to buy or sell a commodity for a predetermined price in the future. Hence, the contract represents an expectation of upcoming price volatility and protects against upside and downside risks. Consequently, the time horizon that a contract is focusing on becomes a pertinent factor in contract valuation. Another important definition is that of closing price: which refers to the last transaction price on the contract market during current day's trading.[32] It is important to note that closing price represents the demand and supply of contracts rather than demand and supply of commodities, though the two are

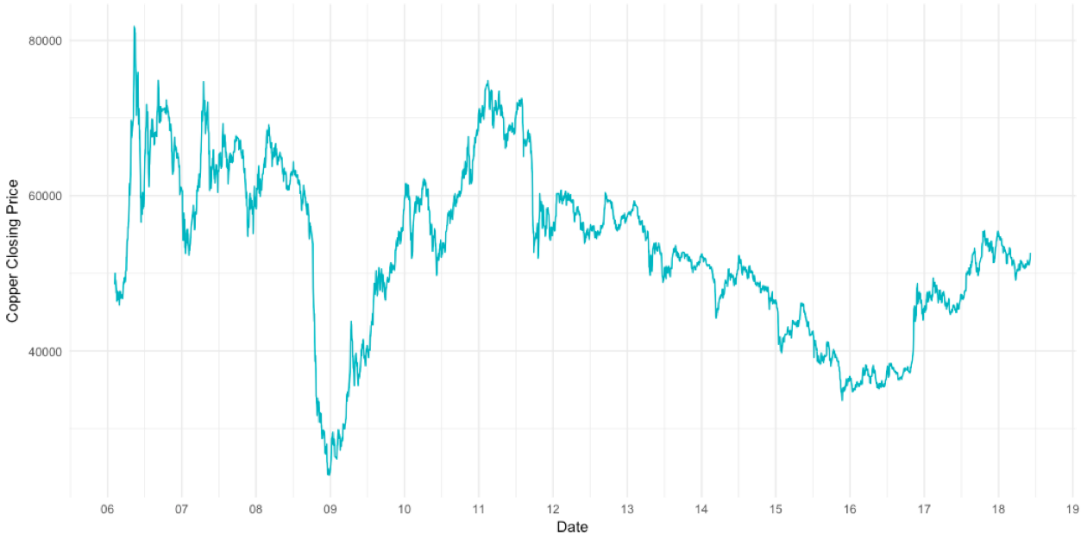
closely related.

We also notice that in the 12 years of time covered in our data set, the commodities have been subjected to a number of economic shocks and business cycle fluctuations. These periods are categorized by sharp upward or downward reversals in price trends and overall increase in volatility until a new equilibrium is reached. Studies analysing these trends have attributed the rise in copper and aluminium prices (and commodities in general) to the increased demand for these goods in the emerging markets, particularly China.[1] This phenomenon persisted from 2000-2014 and is termed as the *commodities super cycle*. The rise in prices was not linear, however, with the global financial crisis and sovereign debt crisis in 2008-09 leading to sharp price drops before a recovery occurred in latter half of 2009.[16]

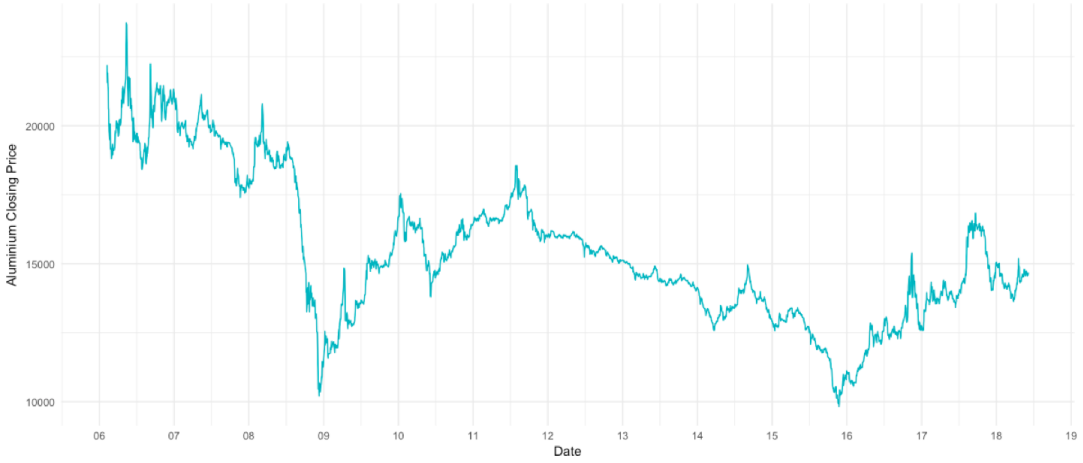
Below, we have plotted the closing prices of copper and aluminium over 12 year period from 2006 to 2018. From Figure 2.1(a) and (b) we can observe that the price trends in copper and aluminium are quite similar indicating that they react in a similar manner to cyclical forces. We also see the emergence of a new rising price trend starting in 2016, which is considered to be the beginning of a new commodities cycle.[5]

2.2 Data cleaning and processing

In order to work with time series data it is imperative that we enforce stationarity on our variable of interest – closing price. Stationarity helps to ensure that closing price will have a more consistent mean, variance, and autocorrelation structure as time progresses. In our study, we achieve this by working with percentage returns on closing price ($:= \frac{ClosingPrice_t}{ClosingPrice_{t-1}} - 1$) rather than closing price itself.



(a) Closing Price of Copper



(b) Closing Price of Aluminium

Figure 2.1: Commodity Closing Price

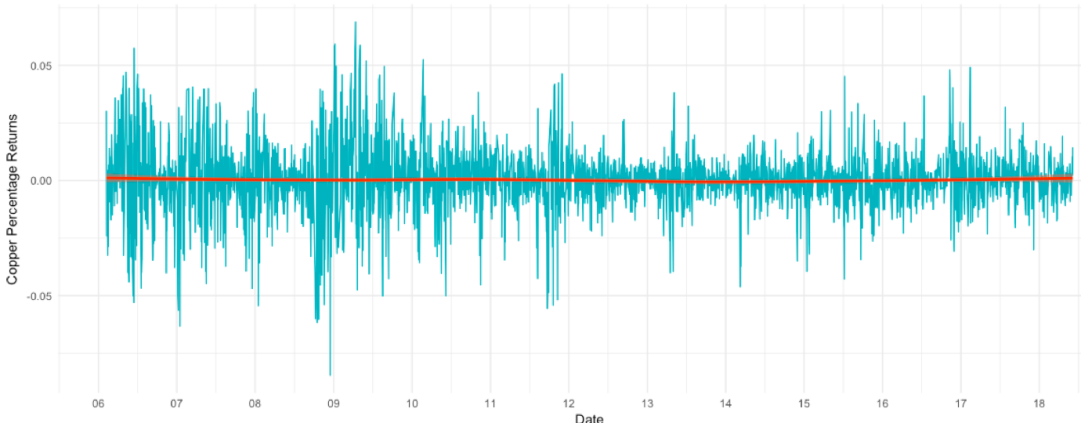
As noted earlier, our data set does not have any missing variables. However, once we start to work with lags of percentage returns in our online experts setting we will drop the first few observations equivalent to number of lags under consideration. This is because when the number of lags exceeds the time frame previous to it it can create missing values in the data set.

Before moving further, it is worth outlining some notation that will be used for the rest of this study. For each metal, we denote the observed percentage returns at the current time point as X_T and its lags at previous time points as X_{T-1}, X_{T-2} etc. We also assume that returns are generated by a Gaussian

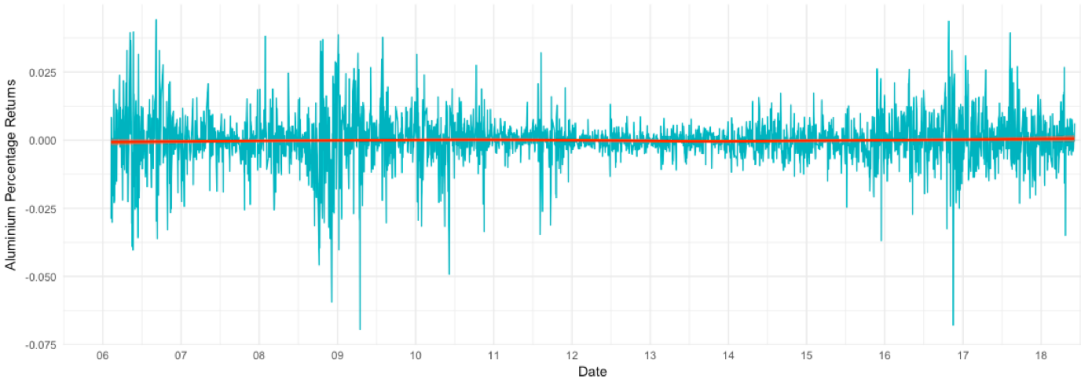
process defined as $X_t + \epsilon_t$, where ϵ_t is the noise term whose mean is 0 and is independent of X_t. This implies that $\mathrm{E}[X_t + \epsilon_t] = X_t$ for $t = 1...T$. As we move to online setting from chapter 3 onward, we will learn to relax these assumptions.

2.3 Descriptive statistics of metals data

To gain insight into the true nature of return on closing prices for aluminium and copper, we plot below the percentage returns on each of these metals:



(a) Closing Price Change - Copper



(b) Closing Price Change - Aluminium

Figure 2.2: Change in Closing Price

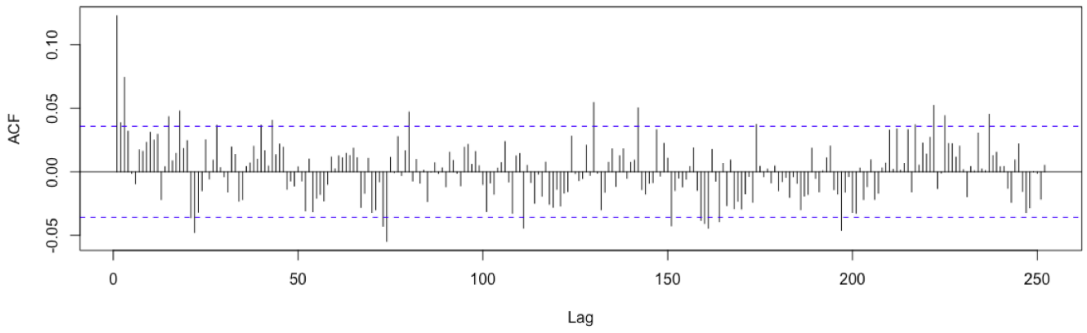
Looking at the figures above, few things immediately jump out. First, we see that the red trend line which highlights the path of returns on closing price for aluminium and copper is fixed close to zero mean. There also does not seem to be a prolonged upward or downward trend after our variable transformation to percentage returns. However, both sets of data show high degree of volatility,

which seem to cluster at different time intervals. The clustering is no surprise since, as explained in chapter 1, copper and aluminium markets have been subjected to changes in business cycle and economic shocks. What is interesting to note is the different positions at which these clusters occur for our metals.

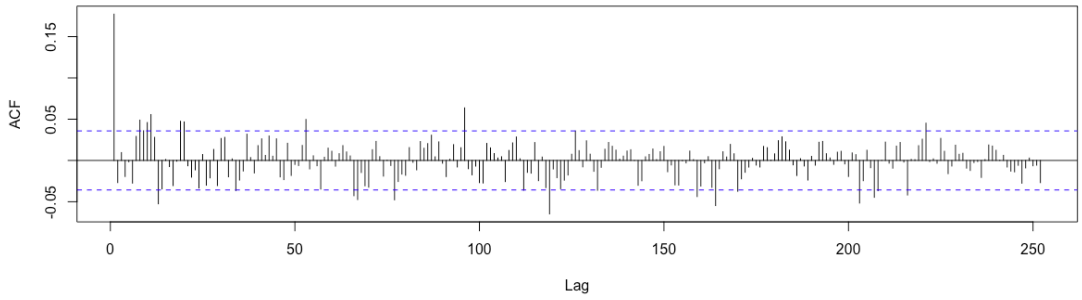
A cursory look at the volatility clusters tells us that aluminium overall has marginally lesser volatility than copper. This is particularly true for the years between 06-08 and 12-14. However, post 2014 the volatility in returns on closing price for aluminium has surpassed that of copper. For the latter half of 08 up till year 10 we see the effects of global financial crisis increasing the price volatility in both sets of markets at almost identical levels.

Since futures contracts are financial instruments that explicitly take into account various time horizons, it is worth exploring how much memory returns on closing price is carrying in its value. We can achieve this by assessing the possible dependence between the percentage returns on closing price and the noise terms across all previous time lags, that is, dependence between $X_T, X_{T-1}, X_{T-2}...$ and $\epsilon_T, \epsilon_{T-1}, \epsilon_{T_2}...$ etc.

In Figure 2.3, we have plotted the autocorrelation function of percentage returns on closing price, that is, $\text{corr}(X_t, X_t - k)$, for the two metals under study. We have extended the plot to 252 lags as this roughly represents the number of active market days in a year. The dotted blue lines represent the 95% confidence interval levels. Bars that are above or below this line indicate a high degree of significance with respect to percentage returns in current time period. Immediately we begin to see interesting patterns emerge in the two autocorrelation plots. In particular, we see signs of a long market memory with significant lags being present throughout our plot. Copper, being the more volatile commodity, shows higher number of significant lags beyond the first working month (22 lags) when compared to aluminium, with no signs of tapering. Instead, we begin to see a crude form of seasonality emerge in both commodities, which should not be altogether surprising given the volatile and cyclical nature of this market as



(a) ACF Copper



(b) ACF Aluminium

Figure 2.3: ACF of commodities

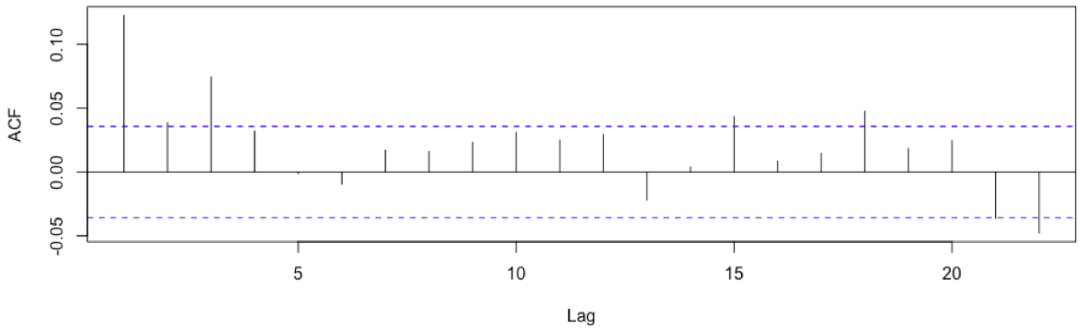
discussed earlier.

The patterns observed above require us to conduct tests for stationarity and trend stationarity before we construct our time-series models. We performed both the Augmented Dickey-Fuller (ADF) test and the KPSS tests which indicate that lags trail off into noise after the 14th and 12th lag respectively, for both sets of commodities.[1] In the Augmented Dickey-Fuller test the alternative hypothesis indicates stationarity and we achieved a p-value of 0.01. For the KPSS trend stationarity test, the null hypothesis indicates stationarity and we achieved p-value of 0.1. In comparison, when we ran the Dickey-Fuller test on our original closing price we achieved a p-value of 0.12, and the KPSS test gave us a p-value of 0; thus indicating non-stationary behaviour.

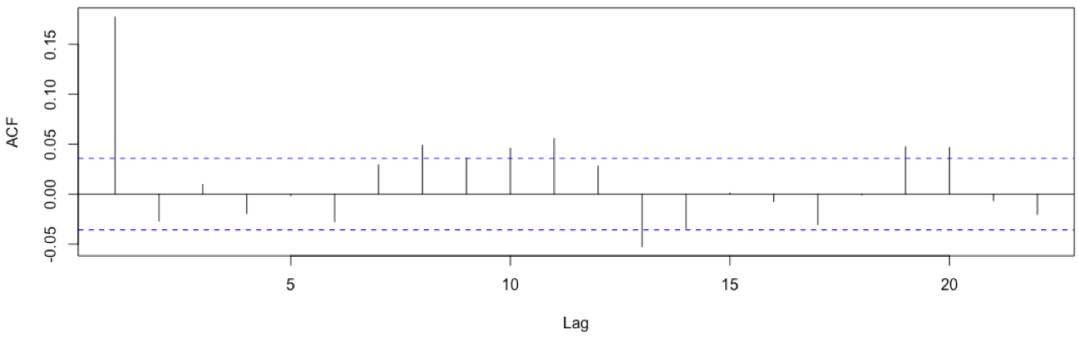
Our study will focus on the first 22 lags, as we are interested in forecasting

[1]We conducted both ADF and KPSS tests because ADF is a unit root test. It has difficulty assessing trend whether time series is trend stationary. KPSS test overcomes this issue by testing for trend stationarity as well.

intervals of 1 day, 1 week and 1 month.[2]. We re-plot our ACF correlogram in Figure 2.4 and focus on the first 22 lags:



(a) Copper Monthly ACF



(b) Aluminium Monthly ACF

Figure 2.4: Monthly ACF of Commodities

In both these plots we see that lag 1 is far above the confidence bounds, which indicates a significant positive temporal relationship between the returns at the previous and current time period, or the random process, or both. We test this significance further by using the Ljung-Box statistics to look for signs of autocorrelation between current percentage returns on closing price and its first lag. The p-value obtained for copper and aluminium is approximately 0. Thus, strengthening our evidence of strong positive autocorrelation at the first lag. Both our commodities have 7 significant lags within the month long time frame under consideration, and majority of these show positive autocorrelation.

[2]22 lags roughly indicate the number of market days in a month

2.4 Feature importance

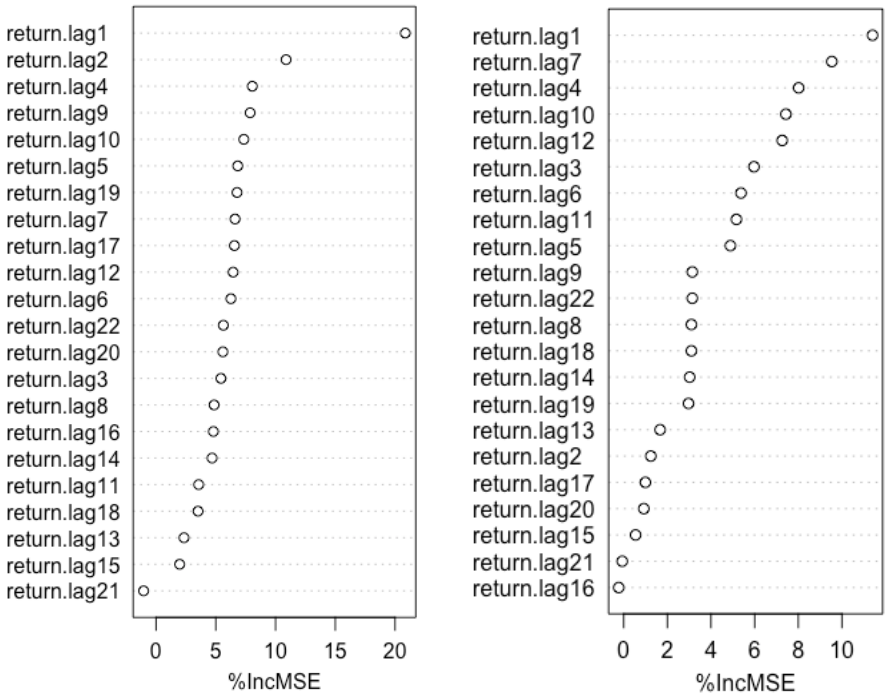
One of the eventual aims of this study is to compare offline and online models in terms of performance and how the independent variables are assigned importance in these two settings. To do this, we would like to compare the importance of different lags in our offline autoregressive model with that of online setting, where we treat these lags as individual experts and allow their weights to fluctuate.

In this section, we will setup our random forest algorithm to assess the importance of our 22 lag features. Random forests use out of bag samples to construct their variable importance measure. The idea behind this process, simply, is to change the value of a variable that is considered important with another random value that is generated using bootstrap aggregation of the data. The decrease in accuracy as a result of this permuting is averaged across all the trees in the random forest and an importance measure is constructed. This randomization has the same impact as setting coefficient of a variable to zero in a linear model. [21]

The importance ranking of lags we see in Figure 2.5 is quite different from the significant lags we obtained from our autocorrelation function. This is evidenced by the fact that for copper only lags 1 and 2 overlap in both those measurements, whereas for aluminium only lags 1 and 10 overlap. This is not entirely unexpected as autocorrelation is looking at linear relation between current variable value and its lags, while the feature selection algorithm is comparing importance of individual variables and simultaneously contextualizing their impact with other independent variables in our data set. [21]

2.5 Offline models

Having done our exploratory data analysis, we can now begin constructing our time series models. A time series model measures a sequence of signals assumed to be spaced at uniform intervals. In this section we will focus on constructing



(a) Lag Ranking Copper

(b) Lag Ranking Aluminium

Figure 2.5: Feature Importance of Commodities

our learners in the offline setting, that is, we will consider the dataset in its entirety when training our model. As mentioned earlier, we will denote our output signal as X_t measured at time t, and let ϵ_t be the noise term at t.

The significant lags observed in the ACF plots of change in closing price for both our commodities indicates a strong temporal relationship with its past values. One way to model these relationships is through $AR(k)$(short for autoregressive) processes, which are parameterized by a time horizon k and a coefficient vector $\alpha \in \mathbb{R}^k$. The mathematical formulation for AR(k) process with zero mean noise term ϵ_t is as follows:

$$X_t = \sum_{i=1}^{k} \alpha_i X_{t-i} + \epsilon_t \tag{2.1}$$

Essentially, the model assumes that each output X_t is a linear combination of previous k signals mixed with some noise term. A more advanced iteration of this model is the ARMA(k,q) (short for autoregressive moving average) process, which is parameterized by two horizon terms k,q and coefficient vectors $\alpha \in \mathbb{R}^k$

and $\beta \in \mathbb{R}^q$. Mathematically, we can express ARMA models as:

$$X_t = \sum_{i=1}^{k} \alpha_i X_{t-i} + \sum_{i=1}^{q} \beta_i \epsilon_{t-i} + \epsilon_t \tag{2.2}$$

where e_t are zero mean noise terms. The AR(k) model is a special case of ARMA(k,q) model with β coefficients turned to zero.The main assumption that governs the ARMA model is that of weak stationarity, which implies the mean and variance of the random variable's distribution is constant across time. This is a strong assumption that needs to be upheld in the batch learning setting but will be relaxed in the online setting.

In this study we build both the AR and ARMA models that best fit the copper and aluminium datasets, and compare them to our baseline model which will consist of a learner that only predicts 0 and another that only predicts the mean. The best lags in our AR and ARMA models will be chosen based on the Akaike information criterion (AIC)[3] that ranks models based on 1) goodness of fit, and 2) the complexity of the model.[13] Usually, in time series analysis acf and partial-acf values are considered when determining the ideal number of lags as model input. But given the lack of decay observed in our acf plots, we have decided to use AIC as it helps enforce parsimony and prevents the lag space from being unnecessarily large and computationally complex.

2.5.1 Modeling copper prices

Table 2.1 contains the results of our best ARMA model for copper along with comparisons to the baselines:

Here, we make two key observations: first, our top performing ARMA model only marginally outperforms the baseline models. Second, the moving average component of our ARMA model plays no role in modeling change in closing price during the current time period. Hence, our leading AR and ARMA model are the

[3]The models were constructed using the auto.arima() command in R's *forecast* library

Type of Model	Parameter of Model	MSE $\times 10^{-4}$
Best Model	ARMA(3,0)	1.964
Baseline Models	Mean	2.004
	Predict 0	2.005

Table 2.1: Copper Offline Model Results

same. The best model yields average mean square error (MSE) of 1.964×10^{-4}.

We shall now conduct some residual diagnostics to determine the robustness of our models. The results of which can be seen in Fig 2.6. The time plot shows changing variation in residuals across time. The volatility is high during the first 1000 observations before reducing. This heteroskedasticity is a potential sign of inconsistency in our estimator. The histogram and acf plots however, show more evidence of robustness. The residual mean is roughly around 0 and there is virtually no sign of severe autocorrelation in the lags. This is further substantiated when we run the Ljung-Box statistic on our residuals to check for autocorrelation and get insignificant results.

2.5.2 Modeling aluminium prices

The important results implementing ARMA models for our price changes in aluminium are shown in Table 2.2:

Type of Model	Parameter of Model	MSE $\times 10^{-5}$
Best ARMA Model	ARMA(1,1)	8.29
Best AR Model	ARMA(2,0)	8.30
Baseline Models	Mean	8.604
	Predict 0	8.605

Table 2.2: Aluminium Offline Model Results

There are two important takeaways from the table above: first, similar to the

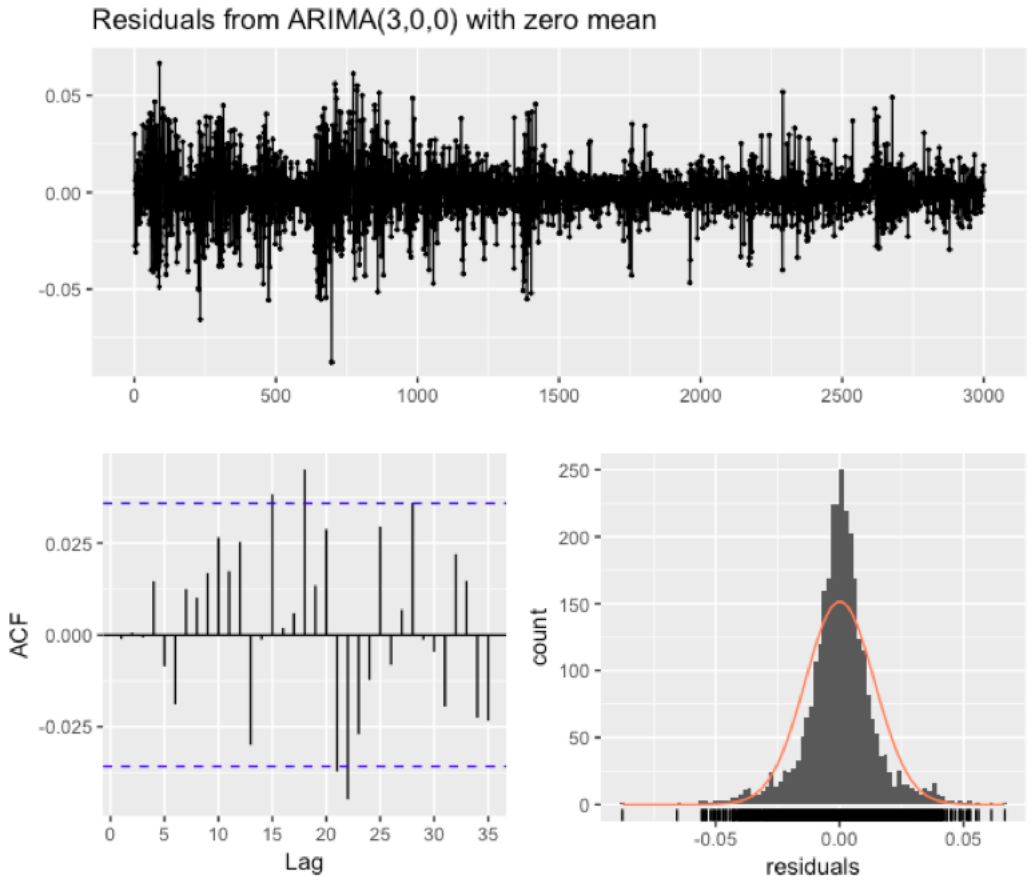
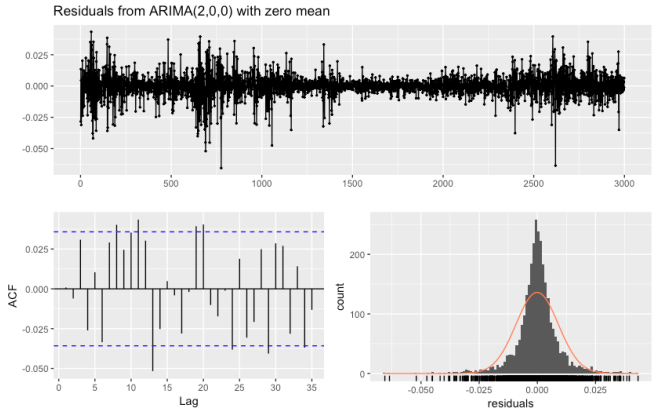


Figure 2.6: Copper Residual Plots

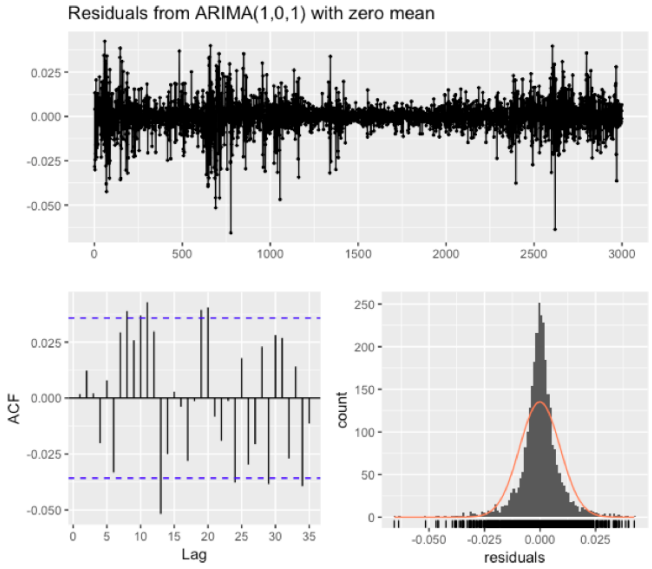
ARMA models for our copper database, the time series models here only yield marginal gains. Second, the existence of a moving average lag seems to yield a slightly better MSE than the best autoregressive model without any moving average terms. The best ARMA model here yields an average MSE of 8.29×10^{-5}.

Finally, we conduct our residual diagnostics on our aluminium database to check the robustness of our AR and ARMA models in 2.7(a) and (b). Both the time plots show clear fluctuations in temporal variance, and to a greater degree than copper. The volatility is high during the first 1200 observations before tapering and then rising again at around the 2200 observation mark. Hence, our ARMA models are extremely inconsistent due to strong prevalence of heteroskedasticity. Further evidence of this inconsistency can be seen from the residual plots, which are skewed with a long left tail (although the ARMA(1,1) graph is slightly more consistent that AR(2) graph). Running the Ljung-Box statistic on our residuals also shows evidence of autocorrelation for both our models as they fall well

within the 1% significance range.



(a) AR(2,0) Residuals



(b) AR(1,1) Residuals

Figure 2.7: Residuals of Offline Model-Aluminium

Although the ARMA models constructed in the offline settings outperform the baseline models, they do so at the price of violating their own underlying assumptions of weak stationarity. Hence, the inconsistencies caused by these violations make the results obtained unreliable. In the next chapter we will take our models to the online setting and show how, given convergence, we can achieve far more robust results than the offline setting.

Chapter 3

Online Learning

3.1 Background and setup

The complexity posed by the ever-changing demand and supply forces acting on a market environment like commodities futures makes it unfeasible to lay out single theoretical models, and solve them using classical algorithmic theory and optimization.[22] This is clearly evidenced by the offline ARMA models we constructed in chapter 2 which were rigid in their assumptions and failed their own quality checks. In order to come up with better learners it is imperative that we apply more robust approaches and utilize optimization methods that learn along the way as more aspects of the problem are revealed.

Online learning is a method that can be used in situations that require an algorithm to adapt dynamically to new patterns of data, or when the data itself is generated as a function of time; as seen in our dataset.[28] In this setting, the learning algorithm is fed a sequence of data $(x_1, y_1), (x_2, y_2), ...(x_t, y_t)$ in order. Specifically, the algorithm observes x_1 and tries to predict y_1. After making its prediction, our algorithm sees the true value of y_1. At this point, the algorithm will try to *learn* from any mistake it made during prediction. The algorithm is then shown the next input x_2 and again asked to make a prediction on output, after which y_2 is revealed. This proceeds until we reach the last data point (x_t, y_t) while the algorithm learns from its mistake each step of the way.[26] Thus, it is useful in applications where algorithms have to make predictions

while simultaneously learning, like on our commodities data, which is updated daily.

3.2 Online learning vs offline learning

Apart from its adaptability, online learning has several other theoretical and practical advantages over offline learning. For starters, we can relax the Gaussian distribution, and no autocorrelation assumptions on our residuals. As we saw in our offline model diagnostics, these strong assumptions are easy to violate in real world settings where data commonly morphs and drifts as a function of time. In terms of application, memory management is an important aspect of optimizing a learner's performance. In this regard, online learning can be described as data efficient, since we no longer need to store large data sets in a computer's memory for statistical learning to take place.[6] Instead, we are learning at every step and discarding the past data.

3.3 Online learning for time series

To convert our offline time series models into the online setting we will follow the work done by Anava et al [8]. In this section, we will go in-depth into their reasoning on how the underlying assumptions and mathematical representation of ARMA models can be adapted for making sequential predictions. The main challenge tackled here is ensuring our solutions are convex in the online setting. In the next chapter, we will attempt to recreate their experiments, and finally apply it to our own commodities data set.

3.3.1 Game theoretic framework

Before expounding on the framework used by Anava et al., we will describe some general conventions used in online learning. The online learning setup uses an adversarial framework where a continuous game is being played between a fore-

caster and his environment. Here we analyse if this approach leads to convergent solutions, while the performance measure shifts from minimizing MSE to minimizing average *regret*.[15] Here, regret is the difference between a forecaster's accumulated loss compared to some designated *expert*. The notion of an expert is more nebulous and can vary based on application. This can range from an abstract definition where the expert is a black box model, with unknown computational power, to well defined statistical forecasters optimizing over given data.[15] In [8] and our study we use the latter scenario.

We start our discussion of Anava et al's work by illustrating their adversarial framework. In the first step, the adversary (environment) fixes ARMA parameters (α, β). Then at each iteration t, the adversary samples noise ϵ_t according to some arbitrary distribution, and generates the resulting signal X_t using our ARMA equation 2.2. At no point during this process is the true (α, β) revealed to our forecaster. Next, our online player (forecaster) makes a prediction $\hat{X}_t$, after which the real X_t is revealed and a loss is suffered according to some convex loss function $l_t(X_t, \hat{X}_t)$. The aim here is to minimize the sum of losses suffered over a set horizon T.

The performance benchmark Anava et al. pick in their paper is to converge as close to the best ARMA model in hindsight, as possible. Hence, the loss function at time t can be written as:

$$f_t(\alpha, \beta) = l_t(X_t, \hat{X}_t(\alpha, \beta)) = l_t(X_t, (\sum_{i=1}^{k} \alpha_i X_{t-i} + \sum_{i=1}^{q} \beta_i \epsilon_{t-i})) \tag{3.1}$$

Regret can now be defined as the difference between the sum of loss suffered by the online player against the sum of loss suffered by an optimal player who has found the best combination of α and β. Mathematically, we express regret as:

$$R_T = \sum_{t=1}^{T} (X_t, \hat{X}_t) - \min_{\alpha,\beta} \sum_{t=1}^{T} l_t(X_t, \hat{X}_t(\alpha, \beta)) \tag{3.2}$$

An efficient algorithm would be one, where regret per round sublinearly tends

to zero with increase in time.

Here, we encounter the first major challenge when we take our offline ARMA models to the online setting: The noise terms $\{\epsilon\}_{t=1}^{T}$ are unknown. Thus, our solution set becomes unbounded preventing us from using standard online convex optimization to find (α, β), which in turn prevents us from generating predictions $\hat{X}_t$ and even defining the best model in hindsight.

3.3.2 Assumptions of the model

Before describing the learning framework, we list out the model assumptions used by Anava et al. What is important to note here is that these assumptions are far weaker than the ones made in offline setting, namely i.i.d. Gaussian:

1. The noise terms are stochastically and independently generated, satisfying the condition $E[|\epsilon_t|] < M_{max} < \infty$ and $E[l_t(X_t, X_t - \epsilon_t)] < \infty$ for all t.
2. The parameter α is bounded such that $|\alpha_i| < c < \infty$. This does not lead to any loss in generality.
3. The parameter β is in unit ball, that is $\sum_{j=1}^{q} |\beta_j| < 1$. This assumption is needed to prevent exploding signals, but not strictly necessary.
4. The loss function l_t is Lipshitz continuous for some Lipshitz constant $L > 0$. A standard assumption in compact convex loss functions.

3.3.3 Learning through online convex optimization

In this framework a decision maker picks a point in the online convex set and incurs a loss proportional to the convex function related to chosen point. We illustrate this in fig 3.1 below:

The online player is continuously picking points $x_1, x_2 \ldots$ etc and subsequently incurs loss $f_1(x_1), f_2(x_2) \ldots$ etc related to those points for a total loss of $\sum_t f_t(x_t)$.

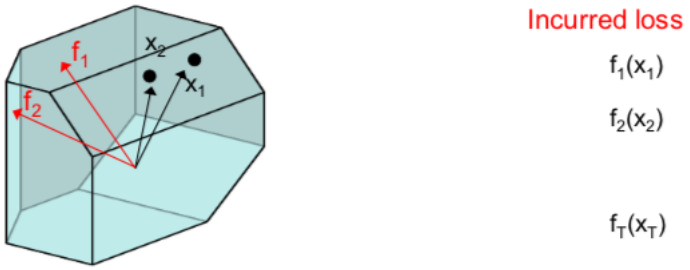


Figure 3.1: Convex Bounded Cost Function [19]

Regret can now be measured as the total loss decision maker suffers throughout all iteration against the total loss suffered picking the best model in hindsight: $\sum_t f_t(x_t) - \min_x \sum_t f_t(x^*)$. Applying this setup to the ARMA model, Anava et al. consider the decisions x_t as parameters (α_t, β_t) and apply it to the signal in the previous time period.

3.3.4 Regret minimization algorithms

One simple and commonly used algorithm to minimize regret is the online gradient descent (OGD) algorithm. In general terms we can describe this algorithm as follows: Suppose our player makes a decision in the convex set of our parameters and observes a particular loss function. The player would then move in the direction of the loss function's gradient. However, this might take the player outside the bounds of the convex set. In which case, we would need to project the player back to the point inside the convex set that is closest to our current point outside. We illustrate this in Figure 3.2.

Earlier, we stated that an efficient algorithm is one where regret sublinearly tends to zero. Zinkevich in [38] laid the theoretical foundation for OGD where he proved that by tuning the parameters of this algorithm we can obtain a regret bounded by the square root of number of iterations:

$$\sum_t f_t(x_t) - \sum_t f_t(x^*) = O(\sqrt{T}) \tag{3.3}$$

Apart from its simplicity, the algorithm is also very efficient and takes linear time to compute apart from the projection step.

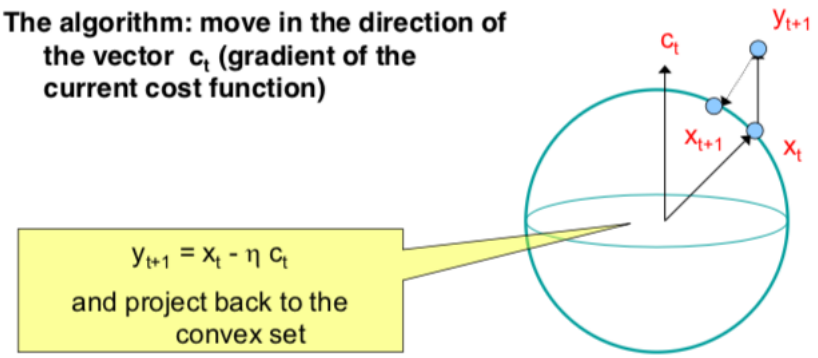


Figure 3.2: Online Gradient Descent [19]

3.3.5 Applying online convex optimization

In this section we will briefly go over how Anava et al. adapted the model agnostic online convex optimization setting above to the specific case of ARMA models. In our study we will only briefly describe the math they used in bounding the ARMA models. For detailed proofs it is worth going over the reference material [8].

As discussed in equation 3.1, the naïve way to write the ARMA loss function would be the following:

$$f_t(\alpha, \beta) = l_t(X_t, (\sum_{i=1}^{k} \alpha_i X_{t-i} + \sum_{i=1}^{q} \beta_i \epsilon_{t-i}))$$

There are two issues here:

1. We do not know the noise term ϵ_t
2. The function is not convex in general

To cope with not knowing the noise term, we will redefine it as a recursive function of noise signals during all the previous periods:

$$\hat{X}_t^{\infty}(\alpha,\beta) \equiv \sum_{i=1}^{k} \alpha_i X_{t-i} + \sum_{j=1}^{q} \beta_j (X_{t-j} - \hat{X}_{t_i}^{\infty}(\alpha,\beta)) \tag{3.4}$$

We can now write a new loss function, which takes into account our redefinition of the noise term:

$$f_t^{\infty}(\alpha,\beta) = l_t(X_t, \sum_{i=1}^{t} c_i(\alpha,\beta) X_{t-i}) \tag{3.5}$$

Although, we have overcome the hurdle of not having a solvable definition for our noise term in the current time period, we have raised a new issue related to *infinite memory*. In order for our algorithm to determine the loss function in the current time period, we now require all the loss functions throughout history. Add to this, our solution set is still not convex.

To tackle these persistent problems Anava et al. employ a method known as *improper learning*. Improper learning can briefly be described as a technique where we use a more robust and general model to learn another related model. To implement this, they replace the (α, β) coefficients of the ARMA model with a single coefficient γ, and re-write the loss term as follows:

$$f_t^{\infty}(\gamma^t) = l_t(X_t, \sum_{i=1}^{t} \gamma_i^t X_{t-i}) \tag{3.6}$$

The second thing Anava et al. do is clip the memory to a restricted time horizon m such that $m \in N$ and at each time point t, they choose an $(m+k)$-dimensional coefficient vector such that $\gamma \in \mathbb{R}^{m+k}$. Hence, the second term in the loss function (3.6) can be written as:

$$\hat{X}_t(\gamma^t) = \sum_{i=1}^{m+k} \gamma_i^t X_{t-i}$$

This prevents the infinite memory problem.

3.3.6 Main observations

Having covered the theoretical aspects of the paper written by Anava et al., we can now turn to their main observations. The first thing they observe is that it is now possible to show $ARMA(k, q)$ models, in the regret sense, as a subset of $AR(m + k)$ models when signals are generated from a larger time horizon $(m+k)$. To put it another way, by observing a longer sequence of ARMA measurements we can minimize regret as if we have a more general AR model but with fewer coefficients.[1]

It is now expected that minimizing regret bounds over $AR(m+k)$ gives you a regret bound with respect to its subset $ARMA(k, q)$. This is the *improper* aspect of their learning framework. With the new $AR(m + k)$ model established, we now have convex solution sets along with bounded memory. Consequently, we can now start using first order learning techniques such as OGD algorithms, and even second order techniques such as Online Newton Step (ONS) to establish regret bounds similar to.[38]

In our next chapter, we will look at the details and implementation of gradient descent algorithms in learning the $AR(m + k)$ model. The loss function we will try to minimize is the following:

$$l_t^m(\gamma^t) = l_t(X_t, \hat{X}_t(\gamma^t)) = l_t(X_t, (\sum_{i=1}^{m+k} \gamma_i^t X_{t-i})) \tag{3.7}$$

We will then attempt to recreate the experiments conducted by Anava et al. to assess the prediction effectiveness of their algorithm on various data sets. Finally, we will apply these algorithms to our own commodities data sets and test for loss convergence.

[1] Detailed proof of this available in the original paper [8]

Chapter 4

Experiments With Online Learning

As we briefly discussed in the previous chapter, one way to learn online models is through online convex optimization algorithms like OGD. In this chapter, we will recreate the experiments conducted by Anava et al. on 6 different data sets to test whether the OCO algorithms are adapting to the evolving function underneath, and optimizing to a satisfactory level. Since the focus of our study is testing how online models perform against our offline ARMA models, we will not recreate the Yule-Walker and recurrent least squares (RLS) baselines used in their study. Instead, we will test our optimizer's performance against the best model in hindsight and where possible, our offline models.

In total, we will test the performance of two OCO algorithms: our standard OGD, and Online Newton Step (ONS) for special case of exp-concave loss. By implementing ONS we wish to provide more context about the advantages and disadvantages of OGD algorithms. After replicating these experiments we will apply the two optimizers to our commodities dataset and check for convergence in loss.

In the following sections we give details about our two online convex optimization algorithms.

4.1 Algorithm parameters

We start our description of the online optimizers by first establishing some important parameters. Let K denote a decision set of candidates for $(m+k)$ - dimensional coefficient vectors, where:

$$K = \{\gamma \in R^{m+k}, |\gamma_j| \leq 1, j = 1, \ldots, m\}$$

From our bounding assumptions 2 and 3 on α and β in (3.3.2), we can now restrict our improper learning parameter γ. Let D be the diameter of our decision set K, which we bound using:

$$D = \sup_{\gamma_1, \gamma_2 \in K} ||\gamma_1 - \gamma_2||_2 = \sqrt{2(m+k)}$$

We next set the upper bound G for our loss function $||\Delta l_t^m(\gamma)||$ for all $t \in K$. This value can change based on the type of loss function. For squared loss $G = 2\sqrt{m+k}D$.

4.2 ARMA online gradient descent (ARMA-OGD)

OGD is the standard algorithm used in online convex optimization that finds its application in most general online settings.[22] This algorithm was first proposed by Martin Zinkevich in [38] and draws its inspiration from the gradient descent algorithm used commonly in the batch learning setting. As explained in section 3.3.4, the algorithm works by taking a step from a point in the previous time period towards the gradient of its cost function. If the algorithm arrives at a point that is outside the convex set we have defined for our data, then we project it back towards the closest point on the convex set. The remarkable aspect of this method is that despite the fact that the underlying cost function could always be changing with each new observation, the regret attained by this

algorithm will still be sublinear.[1]

For the case of ARMA models, Anava et al. use OGD to find the optimal coefficient γ^t at each new data point. The implementation details for ARMA-OGD algorithm is given below. Note that $\prod_k$ refers to the Euclidean projection onto K, that is, $\prod_K(y) = argmin_{x \in K}||y - x||_2$.

Algorithm 1 ARMA-OGD(k,q)

1: Input: ARMA order k,q. Learning rate η
2: Set $m = q.\log_{1-\epsilon}((TLM_{max})^{-1})$
3: Choose $\gamma^1 \in K$ arbitrarily
4: **for** $t = 1$ to $(T-1)$ **do**
5: Predict $\hat{X}_t(\gamma^t) = \sum_{i=1}^{m+k} \gamma_i^t X_{t-i}$
6: Observe X_t and suffer loss $l_t^m(\gamma_t)$
7: Let $\nabla = \nabla l_t^m(\gamma_t)$
8: Set $\gamma^{t+1} \leftarrow \prod_K(\gamma^t - \frac{1}{\eta}\nabla_t)$
9: **end for**

From Algorithm 1 they show the following theorem holds[2] :

Theorem 4.2.1 *Let $k, q \geq 1$ and set $\eta = \frac{D}{G\sqrt{T}}$. Then for any data sequence $\{X_t\}_{t=1}^T$ that satisfies the assumptions from 3.3.2, algorithm 1 generates an online sequence $\{\gamma^t\}_{t=1}^T$ for which the following holds:*

$$\sum_{t=1}^{T} l_t^m(\gamma^t) - \min_{\alpha,\beta} \sum_{t=1}^{T} \mathbb{E}[f_t(\alpha, \beta)] = O(GD\sqrt{T})$$

There are two important observations to make from theorem 4.2.1. First, this theorem indicates that we can make regret converge to the best $ARMA(k, q)$ model in hindsight, using only $AR(m+k)$ models. Second, we use an expectation operator since the noise terms ϵ_t are unknown.

[1] For proof it is worth seeing Theorem 3.1 in [22]
[2] For proof see Theorem 3.5 in [8]

4.3 ARMA online newton step (ARMA-ONS)

Commodity futures, like other ETFs have been studied as part of portfolio theory with the aim of ensuring highest expected future profit.[11] This means that we can apply universal portfolio algorithms to optimize their return.[2] Universal portfolio algorithms are some of the most widely discussed continuous decision making algorithms in the field of finance and information theory. These algorithms seek to learn adaptively from historical data and maximize the log-optimal growth rate in the long run.[33] This is highly reminiscent of the online convex optimization framework we have been looking at in our study, and here we can adopt regret as a performance metric.[22] The only difference is that our online player is now focused on gain maximization rather than loss minimization.

$$regret_T = \max_{x^* \in K} \sum_{t=1}^{T} f_t(x^*) - \sum_{t=1}^{T} f_t(x_t)$$

As a maximization problem, the solution set we are now focused is concave rather than convex. This implies that the loss function of portfolio selection is no longer strongly convex. However, the Hessian of the function is large and in the direction of the gradient. This property is known as exp-concavity.

Anava et al. consider a quasi-Newton approach where the OCO algorithm attempts to approximate this Hessian.[3] Fundamentally, it is similar to the OGD algorithm in its mechanism. However, where the OGD moves in the direction of previous cost function, the ONS moves in the direction in which Newton-Raphson method would move in the offline setting to optimize the previous cost function. That direction being inverse Hessian times the gradient: $A_t^{-1}\Delta_t$. The projection $\prod_k$ now takes place towards the norm defined by the Hessian A_t rather than the Euclidean norm of OGD.

Although computationally less efficient than OGD because of the need to calcu-

[3] Despite trying to solve a second order derivative, it is important to note that the algorithm is still first order as it is still using only gradient information.[22]

late the inverse Hessian, ONS offers logarithm regret guarantee for exp-concave loss functions thus yielding better theoretical and empirical performance. The algorithm for ARMA version of ONS is given below. We will denote λ as the exp-concavity parameter of the loss function $\{l_t^{mT}{}_{t=1}\}$. For squared loss $\lambda = \frac{1}{m+k}$ is used.

Algorithm 2 ARMA-ONS(k,q)

1: Input: ARMA order k,q. Learning rate η; an initial$(m+k)\times(m+k)$ matrix A_0
2: Set $m = q.\log_{1-\epsilon}((TLM_{max})^{-1})$
3: Choose $\gamma^1 \in K$ arbitrarily
4: **for** $t = 1$ to $(T-1)$ **do**
5: Predict $\hat{X}_t(\gamma^t) = \sum_{i=1}^{m+k} \gamma_i^t X_{t-i}$
6: Observe X_t and suffer loss $l_t^m(\gamma_t)$
7: Let $\nabla = \nabla l_t^m(\gamma_t)$, update $A_t \leftarrow A_{t-1} + \nabla_t \nabla_t^\top$
8: Set $\gamma^{t+1} \leftarrow \prod_K(\gamma^t - \frac{1}{\eta}A_t^{-1}\nabla_t)$
9: **end for**

For algorithm 2 they prove the following:

Theorem 4.3.1 *Let $k, q \geq 1$, and set $A_0 = \epsilon I_{m+k}$, $\epsilon = \frac{1}{\eta^2 D^2}$, $\eta = \frac{1}{2}min4GD, \lambda$. Then, for any data sequence $\{X_t\}_{t=1}^T$ that satisfies the assumption from 3.3.2, Algorithm 2 generates an online sequence $\{\gamma^t\}_{t=1}^T$, for which the following holds:*

$$\sum_{t=1}^{T} l_t^m(\gamma^t) - \min_{\alpha,\beta} \sum_{t=1}^{T} \mathbb{E}[f_t(\alpha,\beta)] = O((GD + \frac{1}{\lambda})log(T))$$

The implication of this theorem are the same as that of theorem 4.2.1.[4]

4.4 Demonstration

In this section we will present the results obtained from trying to replicate the experiments done by Anava et al. It is important to note that the authors of

[4]See [8] for detailed proof. Also see section 4.2 of that study for proof how average square loss suffered in this algorithm converges to average square loss suffered by best ARMA prediction in hindsight.

the study have not made the source code available, therefore some nuances of implementation may be different. It is also worth mentioning that the real world data sets used in their study have changed over time, or are no longer available. Given these restrictions, we will attempt to keep our results as qualitatively similar and coherent to the spirit of their paper as possible. In all the cases, it's important to ensure that our optimizer is flexible enough to adapt to most situations within the limiting assumptions, and we obtain some form of convergent result towards the best model in hindsight.

4.4.1 Demonstration with artificial data

For the experiments below we observe the convention followed by the original paper, and consider the following parameters as standard: in order to ensure confidence in our results we present the mean output of 20 runs. We also choose our $(m + k)$ window for our AR models as 10 under all scenarios. The x-axis in each case represents the number of samples, and the y-axis represents square loss. Additionally, to get a more interpretable result for the trend followed by our optimization algorithms we apply a smoothing function with a window frame of 10% of the sample size.

Setting 1

To ensure our algorithm is implemented correctly we do a sanity check by generating some Gaussian noise. We generate a stationary ARMA distribution with the parameters $\alpha = [0.6, -0.5, 0.4, -0.3]$ and $\beta = [0.3, -0.2]$. The noise terms in this are i.i.d normal with the form $\mathcal{N} = (0, 0.3^2)$. In regression analysis, noise ϵ_t is the unexplained aspect of the model, hence our best predictor will have an average error equivalent to the noise being generated, which we have kept as 0.09.

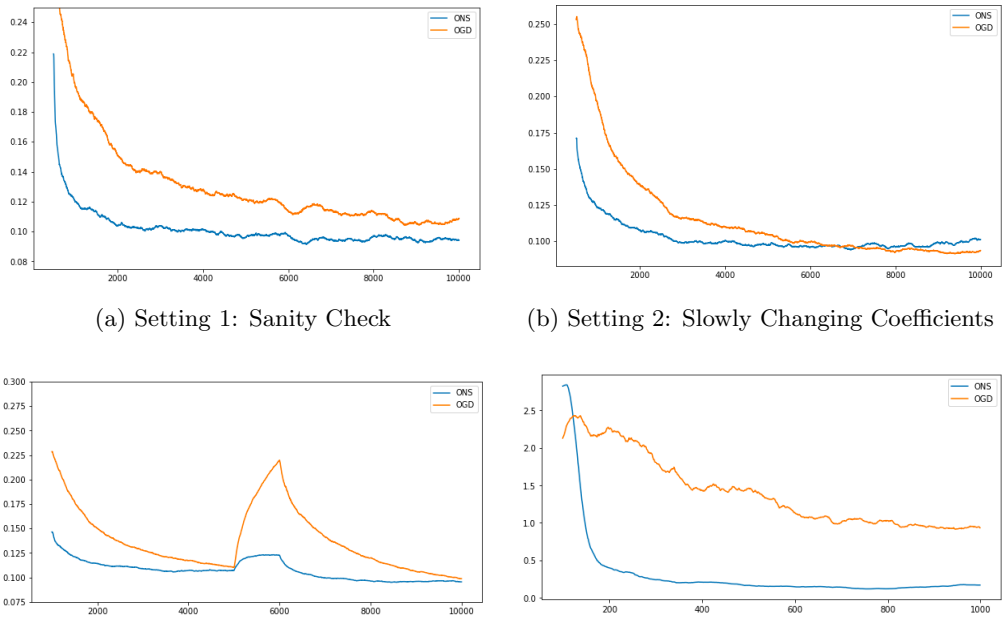
From Fig 4.1(a) we see sublinear convergence towards the best model. ARMA-ONS performs better than ARMA-OGD throughout, as we proved in the previous section for exp-concave loss function. However, ONS roughly takes 7% more computation time.

Setting 2

We now consider our first adaptability test for our OCO algorithms. In this setting we break the strict stationarity rule of offline time-series models and generate a non-stationary ARMA process with coefficients $\beta = [0.32, -0.2]$ and

$$\alpha(t) = [-0.4, -0.5, 0.4, 0.4, 0.1] * (\frac{t}{10^4}) + [0.6, -0.4, -0.5, 0.4] * (1 - \frac{t}{10^4})$$

The equation above illustrates the slow changing nature of our coefficients. Our noise terms are independent with normal distribution $Uni[-0.5, 0.5]$. In this setting the perfect predictor suffers an error rate of 0.0833 due to our unexplained noise. ARMA-ONS outperforms ARMA-OGD on average (0.104 vs 0.118) as it converges faster towards the best predictor but ARMA-OGD overtakes ARMA-ONS for performance in the long run. This is illustrated in Fig 4.1 (b).



(a) Setting 1: Sanity Check

(b) Setting 2: Slowly Changing Coefficients

(c) Setting 3: Abrupt Change

(d) Setting 4: Correlated Noise

Figure 4.1: Experimental results for artificial data.
(x-axis: samples. y-axis: square loss)

Setting 3

Continuing our tests on non-stationary ARMA processes, we now assess the behavior of our optimizers when the change in underlying function is extremely

abrupt. To conduct this experiment we consider a dataset generated from two ARMA processes. The first half of the data set has the parameters $\alpha = [0.6, -0.5, 0.4, -0.4, 0.4]$ and $beta = [0.3, -0.2]$, while the second half is generated with coefficients set $\alpha = [-0.4, -0.5, 0.4, 0.4, 0.1]$ and $\beta = [-0.3, 0.2]$. The noise terms continue to be uncorrelated with distribution of $Uni(-0.5, 0.5)$. Similar to previous setting, the best predictor will suffer a loss of 0.0833. ARMA-ONS comprehensively outperforms ARMA-OGD in this situation, as ARMA-ONS does not suffer the abrupt change on its way to the best model as much as ARMA-OGD. However, both algorithms eventually do show convergence as seen from Fig 4.1(c).

Setting 4

With our next artificial dataset we will test for correlated noise terms which breaks the first assumption of our model. The data was generated using an ARMA process with parameters $\alpha = [0.11, -0.5]$ and $\beta = [0.41, -0.39, -0.685, 0.1]$. The noise terms follow a normal distribution with the expectation that is generated from previous noise term and variance 0.3^2 (noise terms show positive correlation). Here, our implementation begins to break down as neither ONS nor OGD come close to converging towards the best hindsight model[5]. This has been illustrated in Fig 4.1(d).

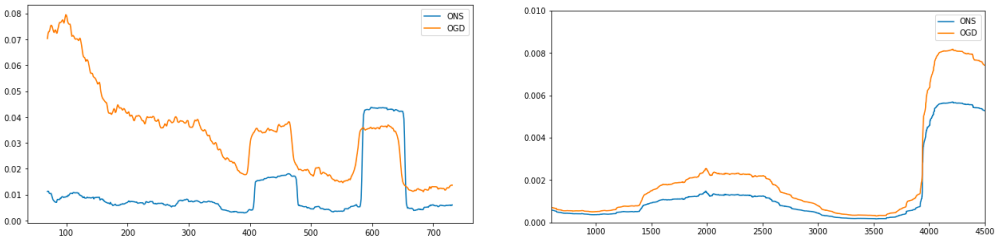
4.4.2 Experiments with real data

Surface temperature

We will now test the robustness of our online algorithms on real world data from different time series domains. The first dataset we look at is taken from weather research. In particular, we are looking at monthly average temperature of the sea surface measured at different time points. The data set we use is a more updated version of the data set used by Anava et al and it is taken from the El Nino database in Python's statsmodel library.

[5]The implementation in the original paper [8] does not seem to suffer such high errors although it shows no signs of convergence either

Weather datasets provide interesting corollary to stock market data as they are both subjected to their own cyclical forces. Hence, there is an underlying pattern that our algorithm can try to learn. One important advantage of weather data over stock market data is that it is less volatile in comparison, hence making a good baseline case to test the effectiveness of our model on real data. As we can see from Figure 4.2(a), our algorithms manage to get really close to learning the weather pattern fully with ARMA-ONS showing more accurate results than ARMA-OGD.



(a) Monthly average temperature

(b) S&P500 daily returns

Figure 4.2: Experimental results for real data.
(x-axis: samples. y-axis: square loss)

S&P500

Our next real world data set comes from the field of stock markets. Here, we consider the daily returns observed on the S&P500 market index. The data is taken from the Yahoo finance API available in python *pandas* library. It is important to note that the data has changed since the time the original paper was written. However, to maintain relevance we will take the first 4500 observations similar to the original paper.

The result from Fig 4.2(b) indicates that the online model we have used so far performs poorly on the financial markets data set. At this point it is important to note that the theoretical guarantees of convergence in online learning is against the best AR predictor. If we have a well specified AR predictor for our data then our online model will converge to it and yield good performance. However, if the AR models are a poor fit for the underlying data then our online

learner will converge only to the best AR model.

There are multiple reasons why an AR model might not be a good fit for the underlying data. One possible explanation is that the $AR(m+k)$ model used is not sophisticated enough to model the volatility observed in the stock market index, especially on the rigid time horizon of $m+k=10$ under consideration. Another reason related to model sophistication is that we need to go beyond lags and look for more information on the exogenous variables affecting S&P500 like the Federal Reserve interest rate, GDP outlook, unemployment etc to model daily returns more accurately.

4.4.3 Application on commodities data

Copper

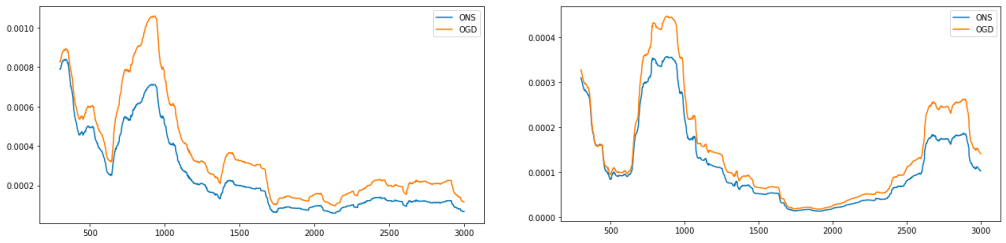
We now apply our OCO algorithms to our commodities data sets. Earlier in the study, we stated that the commodities sector has been subjected to three broad and overlapping trends in the 12 years of data under our purview.[6]

From Fig 4.3(a) we see the model trying to adapt to the different cyclicality it is observing with time. But despite showing a convergent trend it fails to perform as well as the offline ARMA (MSE $= 1.96 \times 10^{-4}$) or even the baseline models. The ONS algorithm outperforms the OGD in terms of mean square loss (2.7×10^{-4} vs 3.7×10^{-4}), however it takes 20% more time to compute.

Aluminium

From Fig 4.3(b) we see similar trends in aluminium as we did in copper, however at around the 2000 sample mark their MSE curves start to diverge. Whereas copper stayed roughly consistent throughout during this period, the MSE on aluminium seems to be rising. This could indicate that the algorithm is having difficulty learning the new commodities super-cycle that aluminium is facing. Both ONS and OGD fail to reach the performance of the best offline ARMA model (MSE $= 8.3 \times 10^{-5}$) or even the baselines (8.6×10^{-5}). ONS marginally

[6] two commodity super-cycles and global financial crisis



(a) Copper daily returns

(b) Aluminium daily returns

Figure 4.3: Result with commodities data.
(x-axis: samples. y-axis: square loss)

outperforms OGD (13×10^{-5} vs 16×10^{-5}).

4.4.4 Discussion of results

From the results obtained on our real world data sets like the S&P500 and applications of OCO algorithms on our commodities data, we begin to see a familiar pattern emerge where the algorithms are unable to fully converge or even outperform the baseline when the data is highly volatile. As stated earlier, this could be due to several reasons revolving around model specification and sophistication. To replicate the experiments done by Anava et al. we followed their setup and kept the $(m+k)$ parameter equal to 10. There is no particular reason why this number has been chosen for our study other than ease of comparison. This can be problematic if it leads to a misspecified model, as convergence is not guaranteed under such scenarios.[17]

In the following chapters we shall attempt to tackle some of these problems. We will start by expanding the number of AR models under consideration and try to optimize them in parallel by using *expert advice* algorithms for online learning. This will give our model far more flexibility in deciding the optimal combination of time horizons when faced with high levels of volatility and overlapping cyclicality in our data. This also meshes well with our initial aim of finding the decision horizons most suited for consideration when making investment choices.

To further tackle the issue of model sophistication we will compare our online-ARMA with other models such as generalized additive models and gradient boosting techniques. We will also introduce certain economic and logistics related exogenous variables in an attempt to increase the information depth contained in our time series models. And finally, we will compare the performance of our OGD algorithm with a commonly used exponentially weighted average forecaster (EWAF) to look for improvement in convergence to the best expert in hindsight.

Chapter 5

Learning From Experts

We ended our previous chapter discussing the need to expand the model space under consideration by our online learning algorithm to get better prediction (and converging) results. A potential way to achieve this would be forecasting closing price changes of our commodities using a group of experts that work together to solve this problem instead of just one $AR(m + k = 10)$ model.

5.1 Expert advice algorithm

In the field of online learning, expert advice algorithms are another form of algorithms related to sequential decision making. Similar to our description in section 3.1, an expert f_E makes predictions continuously as the stream of data flows in. The performance of this expert, at each time t, is then compared against the performance of a set of other accessible experts $\{f_{E,t} : E \in \mathcal{E}\}$ where $f_{E,t} \in \mathcal{D}$ and $\mathcal{E}$ indicates the index of an expert.[15] Once the performance of these experts are aggregated and weighted, our forecaster makes a prediction $\hat{X}_t$. We then reveal to our forecaster the actual output value X_t and compute a nonnegative loss function $l : \mathcal{D} \times \mathcal{Y} \to \mathbb{R}$. Algorithm 3 outlines the procedural details of making predictions with a mixture of experts.

At this point it is important to note that we are still in a game theoretic setting where the forecaster is trying to make predictions $\hat{X}_t$ against the environment,

Algorithm 3 PREDICTION WITH EXPERT ADVICE

Parameters: decision space $\mathcal{D}$, outcome space $\mathcal{Y}$, loss function l, set $\mathcal{E}$ of expert indices.

for each round $t = 1, 2.....$ **do**

the environment chooses the next oucome X_t and the expert advice $\{f_{E,t} : E \in \mathcal{E}\}$; the expert advice is revealed to the forecaster;

the forecaster chooses the prediction $\hat{X}_t \in \mathcal{D}$

the environment reveals the next outcome $X_t \in \mathcal{Y}$;

the forecaster incurs loss$l(\hat{X}_t, X_t)$and each expert E incurs loss $l(f_{E,t}, X_t)$

end for

which sets the expert advice $\{f_{E,t} : E \in \mathcal{E}\}$ and fixes output X_t. The forecaster's goal is to minimize regret of each expert within the decision set. Mathematically, we can denote this as:

$$R_{E,n} = \sum_{t=1}^{n} (l(\hat{X}_t, X_t) - l(f_{E,t}, X_t)) = \hat{L_n} - L_{E,n} \tag{5.1}$$

Where $\hat{L_n}$ is the total loss suffered by the forecaster and $L_{E,n}$ is the total loss of expert E.[15] Therefore, regret is the difference in total loss suffered by the forecaster and an expert E after n samples. In a similar notion we define instantaneous regret of an expert E at time t as $r_{E,t} = l(\hat{X}_t, X_t) - l(f_{E,t}, X_t)$. Then, $R_{E,n} = \sum_{t=1}^{n} r_{E,t}$.

5.2 Creating our mixture of experts

Given the nebulous definition of experts as discussed in section 3.3.1, it is worth analysing what shapes a good mixture of experts for our data set. Ultimately, our aim is to increase the modelling power of our forecaster, that we saw in previous chapters and protect ourselves against model misspecifications.

5.2.1 Lags as experts

In the previous chapter we looked at AR(m+k) models to predict the change in closing price of our commodities data set. We speculated that one of the reasons that approach did not work well was due to model misspecification, and likely the (m+k = 10) decision space under consideration was incorrect. In this part of the study, we will try and take advantage of the online expert setting and explore a larger lag space. We will model different lags in our data set using a more "restricted" form of AR(n) models where all coefficient α not corresponding to the n_{th} lag will be set to 0. We will then weight the experts using one of the aggregating functions available as part of online expert forecasting. An approach where set of experts is created using some variant of autoregressive models is not uncommon. Theoretically, it has been shown that a mixture of AR models can be treated as *universal approximators* with properties similar to neural networks in certain gated settings.[25] [37] The technique used in our study is simpler. Mathematically, our lags X_{t-n} are being modelled in the following way to produce a prediction for unknown output X_t:

$$AR(1) \rightarrow \hat{X}_{t1} = \alpha_1 X_{t-1} + \epsilon_t$$

$$AR(2) \rightarrow \hat{X}_{t2} = \alpha_2 X_{t-2} + \epsilon_t$$

$$AR(3) \rightarrow \hat{X}_{t3} = \alpha_3 X_{t-3} + \epsilon_t$$

$$\vdots$$

$$AR(n) \rightarrow \hat{X}_{tn} = \alpha_n X_{t-n} + \epsilon_t$$

For our study we will consider upto 22 lags as this represents the number of working days in a month. Therefore, our first set of experts E_1 can be represented by:

$$E_1 = [AR(1), AR(2), ..., AR(22)]$$

Our forecaster F will aggregate the set of experts and weigh them to produce a final prediction $\hat{X}_t$ for our unknown actual output X_t. Assuming additive noise, we can mathematically represent this as:

$$\hat{X}_t = F(AR(1), AR(2), ...AR(22)) + \epsilon_t$$

Our actual output X_t will then be revealed and the forecaster will attempt to minimize the loss by reweighing our lag experts. This process will occur continuously in the online setting until we reach the end of our data set.

There are several reasons apart from exploring a larger lag space that we have chosen this mixture of experts. First, by modelling lags individually online, we can ascertain how each decision horizon becomes important at different phases of the business cycle. Second, we can compare the final weights obtained on each of these lags against the feature importance algorithm which we tested in the offline setting (Figure 2.5). It would be interesting to see if there are differences in the lag importance when full database is available to us compared to making prediction real-time in the online setting. Third, exploring large lag horizons comes with computational costs in the offline setting as the solutions are being obtained analytically. Computational complexity is less of a factor online. It would be interesting to see if the prediction results we obtain here outperform the best ARMA models we computed in the offline setting.

5.2.2 Non-linear experts

In our study so far, we have only focused on using past values to determine future change in closing prices. However, this approach is clearly lacking, as there are a whole host of exogenous variables that affect commodity price changes, ranging from economic to logistic factors.[14] One way to model these exogenous variables would be to naively add them into our autoregressive models or introduce them through a vector autoregressive (VAR) framework. However, we will take this opportunity to introduce these exogenous variables using two popular non-linear time series models: generalized additive models (GAM) and gradient boosting model (GBM).[4] [31] [36]

GAM is a technique that adds the input variables using a splines, which can

be linear or non-linear in form depending on the nature of underlying data. These splines act as smoothing function which attempt to generalize the data into smooth curves by local fitting subsections of the data.[20] Because of their adaptability, these smoothing functions are also known as non-parametric functions. Mathematically, we can represent GAM using the following equation:

$$g(\mathbb{E}(Y)) = \beta_0 + f_1(x_1) + f_2(x_2) + ... + f_m(x_m)$$

where $g()$ is some linking function and $f_1(x_1) + f_2(x_2) + ... + f_m(x_m)$ are the smoothing functions on our exogenous variables. For our study we implemented GAM using R's *mgcv* package. This uses a thin plate regression spline where the smoothing parameter λ is being chosen via generalized cross-validation (GCV).[34] One drawback of GCV is that it tends to develop multiple minima and hence, we observe more variable results for smaller data sets.[35] To counter this we attempted to use restricted maximum likelihood (REML) for estimating our smoothing parameter. However, REML consistently performed worse than GCV, hence we have decided not to use it in our models.

In this study we will implement GAM on the following exogenous variables and try to predict change in closing price for copper and aluminium:

1. AUD-USD exchange rate
2. CLP-USD exchange rate
3. 3 month treasury bill rates
4. TED spread
5. Goldman Sachs commodity index
6. Volatility index (VIX)
7. Baltic dry index
8. WTI

No interaction terms or isotropy is assumed when modelling our exogenous variables.

Our second non-linear model is based on the stochastic gradient boosting approach proposed by Friedman.[18] GBMs attempt to generate the best learner that approximates the data by using an ensemble of weaker learners. It does this by essentially training the weak learners on the gradient of the errors generated by other models in the ensemble set. The idea behind this approach is to try and make the weak learners compensate for each other's mistake and come together to form a more accurate stronger learner.

We implemented GBM using the *caret* package in R and the parameters were kept at their standard setting.[1] The input for our GBM are the exogenous variables listed above and our algorithm is again trying to generate a prediction for an unknown output X_t.

We can now represent our non-linear set of experts E_2 by:

$$E_2 = [GAM, GBM]$$

Similar to our first set of experts, our forecaster F will aggregate our non-linear models, weigh them, and output a final prediction $\hat{X}_t$.

$$\hat{X}_t = F(GAM, GBM) + \epsilon_t$$

Our actual output X_t will then be revealed and the forecaster will attempt to minimize the loss by reweighing our non-linear experts. This process will continuously occur for each point in the data set.

5.2.3 Combined experts

For our final mixture of experts we will combine our previous expert sets E_1 and E_2, and assess whether their combination increases the predictive power of

[1] See https://topepo.github.io/caret/model-training-and-tuning.html#basic-parameter-tuning

our online models as a whole. We will also compare these models against our baseline and best offline models obtained in chapter 2. Hence, our final expert set E_3 can be represented as:

$$E_3 = [E1, E2] = [AR(1), AR(2), ..., AR(22), GAM, GBM]$$

The forecaster F will aggregate the result of our experts and make a prediction $\hat{X}_t$ for our unknown output X_t.

$$\hat{X}_t = F(AR(1), AR(2), ...AR(22), GAM, GBM) + \epsilon_t$$

Once the output is revealed at each new data point, the forecaster will minimize loss and reweigh our set of experts. The result of our experiments are available in chapter 6.

5.3 Choice of forecaster

With the choice of forecaster we are effectively determining the optimization algorithm used to weigh our experts. For this part of the study, we again consider OGD as this is the standard online convex optimization framework, and for comparison we will use another popular optimizer known as exponentially weighted average forecaster (EWAF).

5.3.1 OGD-again

We will only briefly review the OGD algorithm as the basic framework remains the same as discussed in section 3.3.4, only instead of a distribution over data points, we now have a distribution over experts. To put it succinctly, consider a case where we have n different experts, each of which offers a model to estimate the underlying function of the data coming in, and some cost attached to this choice. At each round, we have a probability distribution (weight) assigned to each expert. After generating a prediction, the weight is updated in the direction of the cost function produced by these mixture of experts. Suppose, if $f \in \mathbb{R}^n$ is defined as an expert f_E having a probability distribution that our forecaster

selects E, then the set of all probability distributions is convex. Hence, we can see online convex programming as an extension of solving the best expert problem. [24] [38]

5.3.2 Exponentially weighted average forecaster (EWAF)

Intuitively, by using weighted majority algorithms as an aggregation rule we are ensuring that we do not disregard an expert completely when it makes a mistake. Instead, we will just lower its weight for the following predictions, and then readjust continuously as the right values are revealed to us. If we treat these weights as probabilities, over time we can hope to smooth away the influence of poorly performing experts and perform as close to the best model in hindsight, as possible. [12]

In this regard, EWAF is a special type of weighting algorithm that predicts the next point using the following formula:

$$\hat{p_t} = \frac{\sum_{i=1}^{N} exp(\eta(\hat{L}_{t-1} - L_{i,t-1}))f_{i,t}}{\sum_{j=1}^{N} exp(\eta(\hat{L}_{t-1} - L_{j,t-1}))} \tag{5.2}$$

where η is the learning rate, and it is multiplied with the term inside the bracket which represents instantaneous regret. For our EWAF algorithm, η will also be calibrated online. The algorithm for EWAF update can be seen in Table 4:

In the next chapter we will run our three sets of expert mixtures with both these forecasters on the copper and aluminium data sets, and assess their result against the baseline models we formed in chapter 2.

Algorithm 4 EWAF

1: Init: a poll of experts$f_i, i = 1...., N$ and $L_{0,i} = 0, i = 1, ..., N$ and learning rate η

2: **for** $t = 1, 2.....T$ **do**

3: The environment chooses the next outcome X_t and the expert advice$f_{i,t}$;

4: The expert advice is revealed to the forecaster

5: The forecaster chooses the prediction

$$\hat{p_t} = \frac{\sum_{i=1}^{N} exp(\eta(\hat{L}_{t-1} - L_{i,t-1}))f_{i,t}}{\sum_{j=1}^{N} exp(\eta(\hat{L}_{t-1} - L_{j,t-1}))}$$

6: The environment reveals the outcome X_t;

7: The forecaster incurs a loss $l(\hat{X}_t, X_t)$

8: Each expert incurs a loss $l(f_{i,t}, X_t)$

9: The forecaster updates the cumulative loss $L_{i,t} = L_{i,t-1} + l(f_{i,t}, X_t)$

10: **end for**

Chapter 6

Predictions With Experts

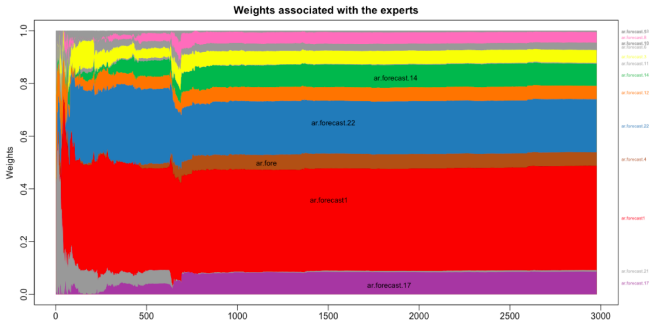
We will begin our performance analysis of expert models by first applying it on our copper database followed by aluminium. For both commodities we will apply the following three combinations of experts: 1) Lags as experts 2) Non-linear time series experts 3) Mixture of experts from first two settings. We will try and optimize the weights on these experts using both OGD and EWAF algorithms, and then compare their performance against the baseline and best offline models. All models have been run 20 times each to ensure stability in our final results.

6.1 Copper

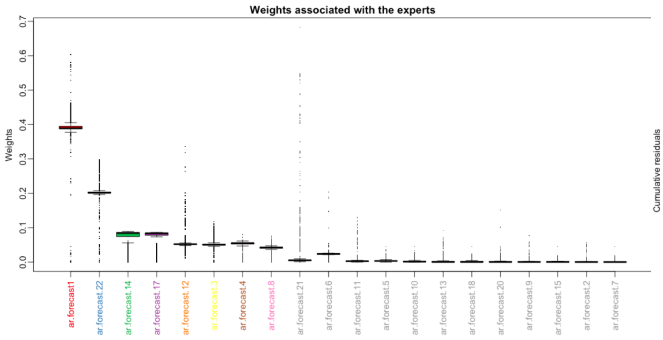
6.1.1 Lags as experts

As mentioned earlier in the study, the decision horizon we are interested in consists of 22 lags, which roughly represents the working days in a month. When running our expert models, we see from Figure 6.1(a) and 6.2(a) that the weights change in dynamically different ways for these two optimizers. The weight evolution is far more stable in OGD when compared to EWAF, despite our underlying variable of interest – change in closing price is very volatile. The weight change in EWAF seems to reflect this volatility. This is theoretically consistent with the way weights are updated in OGD and EWAF. In OGD we are slowly moving in the direction of cost function to minimize regret, whereas EWAF is trying to quickly smooth out the effects of non-performing experts, which change dra-

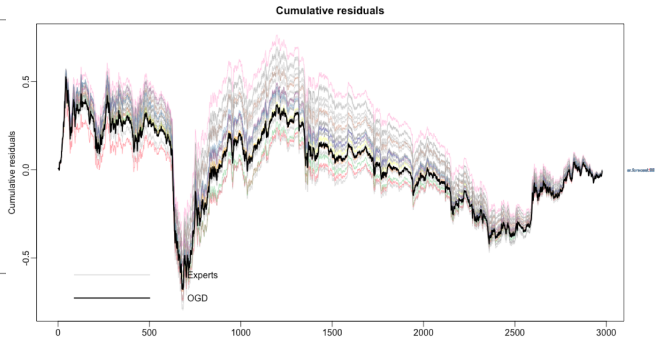
matically based on the stage of the business cycle. A good example of this is lag 22 and lag 3 in Fig 6.2(a), which seems to be very important during initial volatile phase representing the global financial crisis, and then again where we see the commodity super-cycle pick up again after 500 sample points.



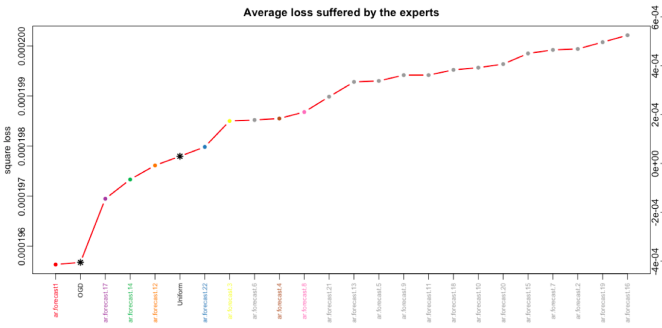
(a) Evolution of Weights



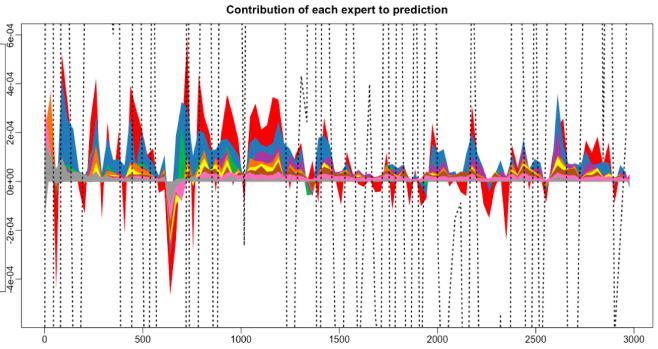
(b) Confidence Intervals on Expert Weights



(c) Prediction Residuals



(d) Mean Square Error of Experts

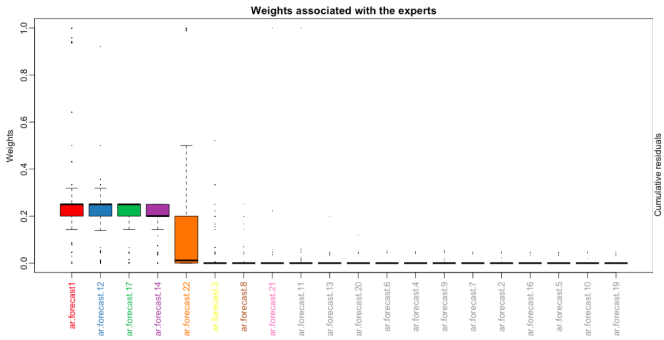


(e) Contribution of Experts to Prediction

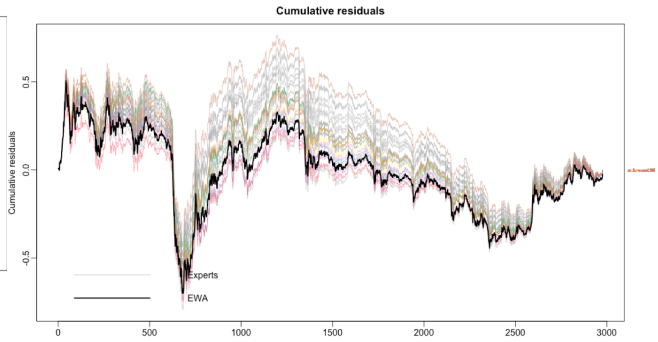
Figure 6.1: OGD-Copper Lags as Experts

We can see a further reflection of the volatile way in which our weights are updated from the confidence interval plots in Fig 6.1(b) and 6.2(b). The box plots for OGD's weight seem far more consistent when compared to EWAF. We also observe that OGD tends to put a small but observable amount of weight on each

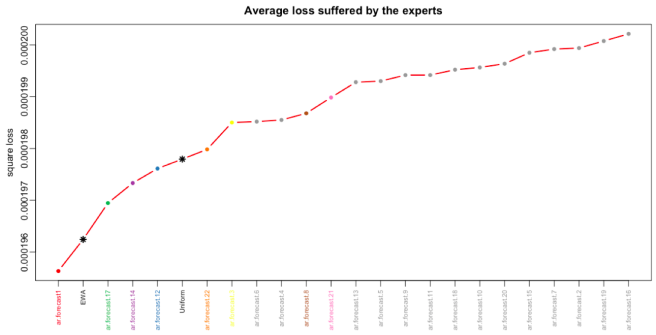
(a) Evolution of Weights



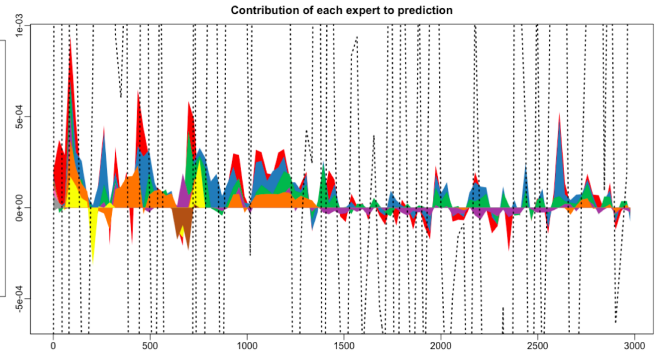
(b) Confidence Intervals on Expert Weights



(c) Prediction Residuals



(d) Average Loss



(e) Contribution of Experts to Prediction

Figure 6.2: EWAF-Copper Lags as Experts

expert, while EWAF puts almost the entire bulk of weights on just 5 experts. A detailed look at the distribution of weights amongst experts is available in the appendix Table A.1, and average loss of experts is available in Fig 6.1(d) and 6.2(d). Another important point of distinction is the decision horizon that each optimizer considers important. Both OGD and EWAF rank the first lag as most important but the following rankings are completely different.

In terms of performance, we want our experts to converge as close to the best model in hindsight as possible to minimize our regret. Fig 6.1(d) and 6.2(d)

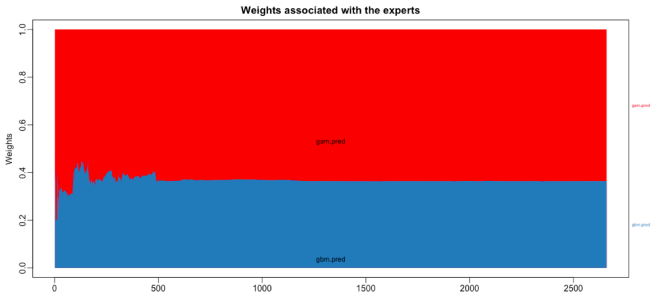
show that both OGD and EWAF perform well in this regard, with OGD performing slightly better than EWAF. A full description of key model performance indicators is available in Table 6.1. Both optimizers outperform the uniform setting where equal weight is given to each expert.

6.1.2 Non-linear experts

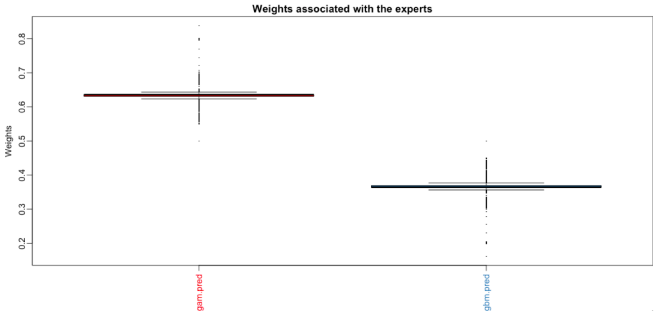
From the results in Fig 6.3(a) and 6.4(a) we can see that both our non-linear experts share roughly equal weights under EWAF and OGD forecasters, with GAM being slightly higher for OGD. This is pattern is also quite consistent as evidenced from the confidence intervals on our weights and the residual plot which almost bisects both non-linear models in Fig 6.3(c) and 6.4(c).

OGD however, shows a fair bit of variation in assigning weights between the experts in the initial time samples. Although gradient descent eventually settles at a slightly higher weight for additive models, during the peak phase of global financial crisis[1] we see it give more preference to gradient boosting. EWAF is far more consistent than OGD other than very specific fluctuations around the 500th sample point.

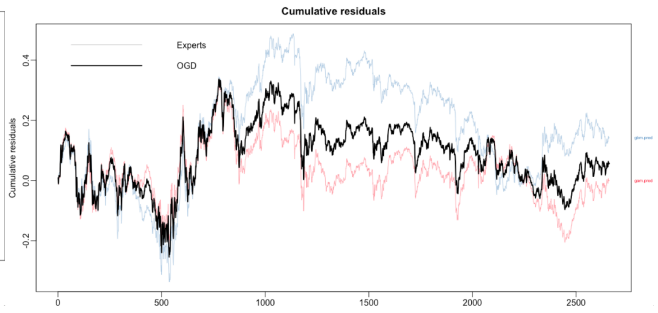
In terms of performance we see negative regret for the first time in Fig 6.3 (d), as OGD slightly outperforms the uniform model and the best model in hindsight, whereas EWA slightly underperforms the uniform model and best hindsight model in Fig 6.3(d). Given that our forecasters are weighing the two experts almost equally on a consistent basis, values so close to the uniform model are unsurprising. What is important however, is that both models significantly outperform our lag experts (Table 6.1).



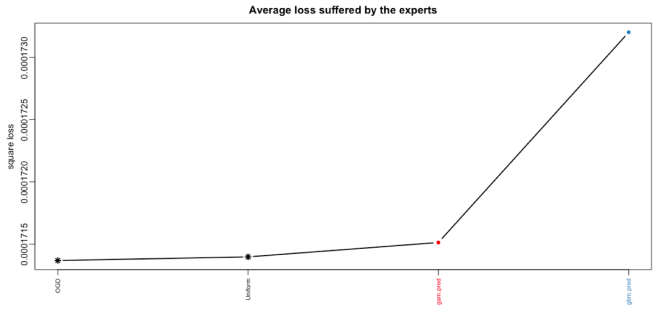
(a) Evolution of Weights



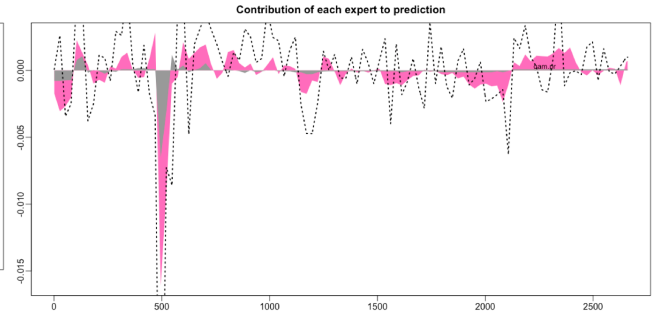
(b) Confidence Intervals on Expert Weights



(c) Prediction Residuals



(d) Mean Square Error of Experts



(e) Contribution of Experts to Prediction

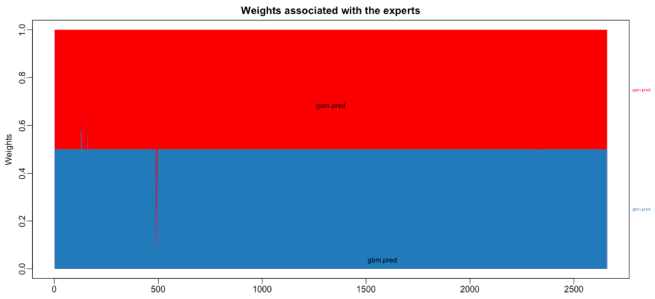
Figure 6.3: OGD -Copper Non-Linear Time Series Experts

6.1.3 Combined set of experts

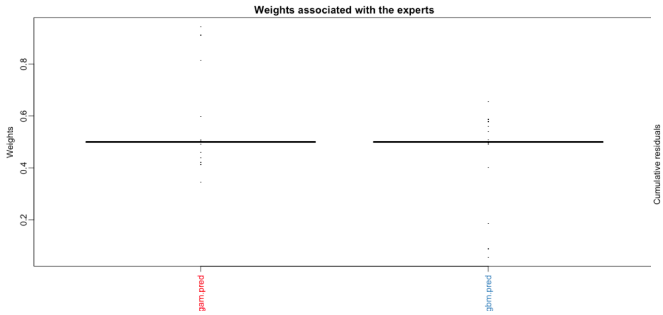
We will now combine the two sets of experts we modelled in previous subsections and test the performance of all the experts taken together.[2] Immediately, we see from Figure 6.5(a) and 6.6(a) that the contrast between the final weights on experts, as well as the evolution of weights throughout time is quite stark between our forecasters. OGD is again far more stable in terms of assigning weights to experts even during times of volatility, while EWAF is initially quite

[1] First 0 to 500 points here represents a time-line from end of 2006 to almost the beginning of 2009

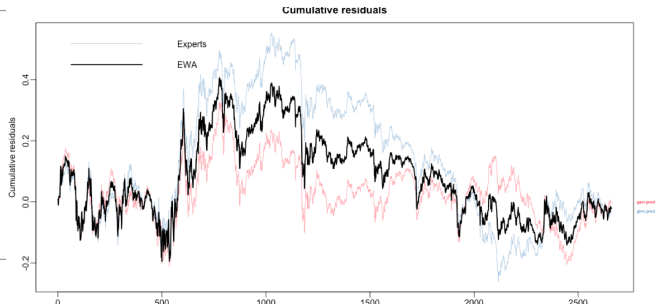
[2] It is important to note that our data points have reduced due to some missing variables from 2999 to 2661.



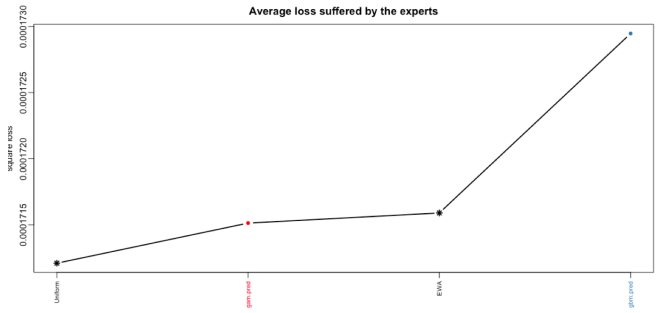
(a) Evolution of Weights



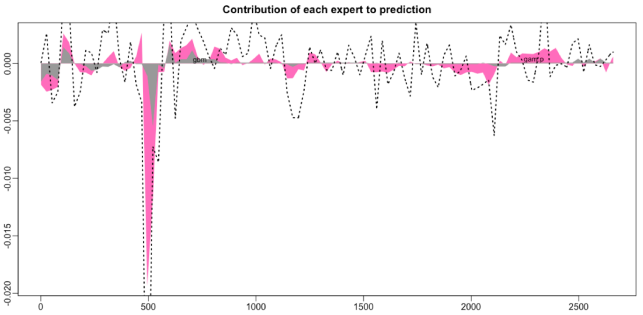
(b) Confidence Intervals on Expert Weights



(c) Prediction Residuals



(d) Mean Square Error of Experts

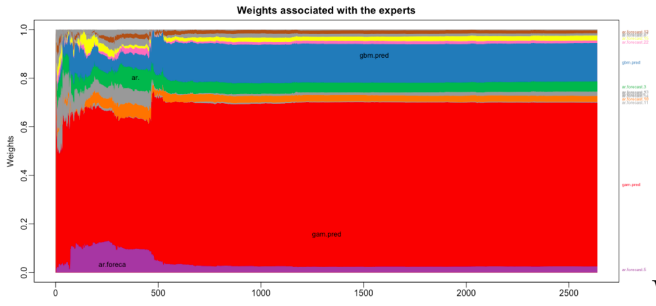


(e) Contribution of Experts to Prediction

Figure 6.4: EWAF-Copper Non-Linear Time Series Experts

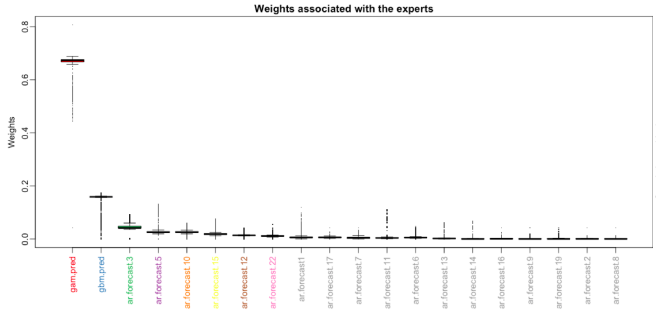
unstable before settling into a balance where lag experts are almost completely discounted. We can infer this further from the confidence interval plots.

However, it is no surprise that non-linear models dominate given the performance increase over lags we saw in previous section. But what is interesting to observe is that GAM now plays a much more significant role in prediction compared to GBM, especially in EWAF where it takes over as the main expert quite abruptly. We see from Table Table A.3 that GAM now occupies roughly 81.6% of the predictive weight in EWAF, and 67.4% for OGD.

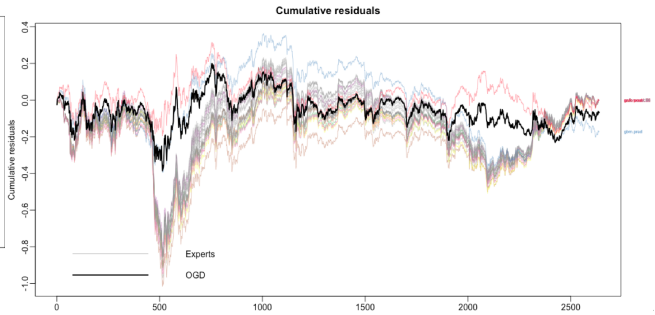


Weight.png

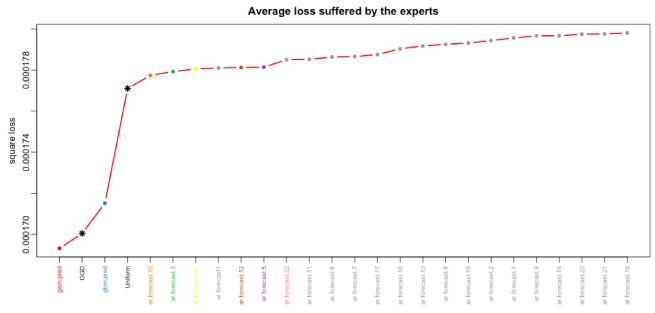
(a) Evolution of Weights



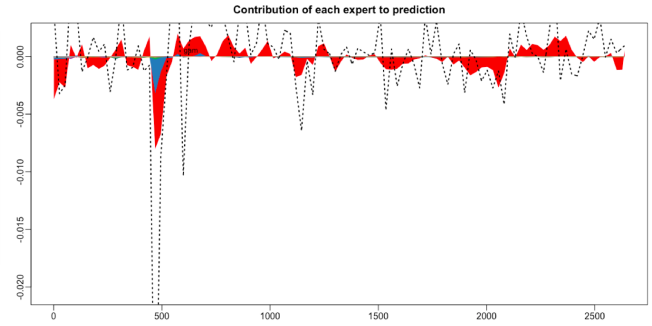
(b) Confidence Intervals on Expert Weights



(c) Prediction Residuals



(d) Mean Square Error of Experts

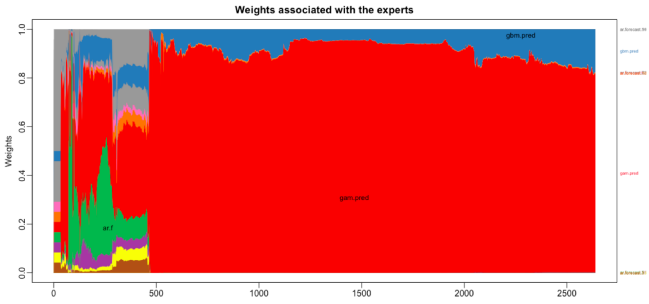


(e) Contribution of Experts to Prediction

Figure 6.5: OGD - Copper Combined Expert Set

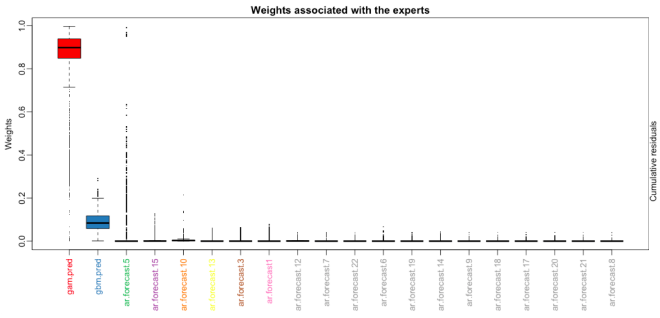
To probe this behaviour more closely, we conducted several experiments where we introduced our AR(n) experts individually into our non-linear expert set. We observed that when poor performing lags are introduced, OGD compensates by reallocating weight more slowly towards the best performing expert compared to EWAF. For example, when we added the first AR experts into our non-linear dataset, we see the weight on best performing non-linear model increase only by 2-5% for OGD, whereas EWAF showed ranges from 15-20% in favour of the best performing model.

Both optimizers outperform the uniform setting and reach close to the best

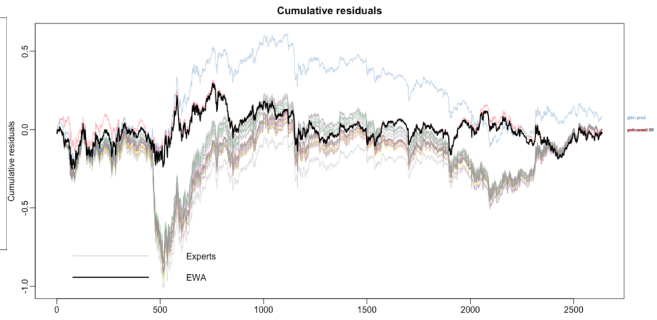


Weight.png

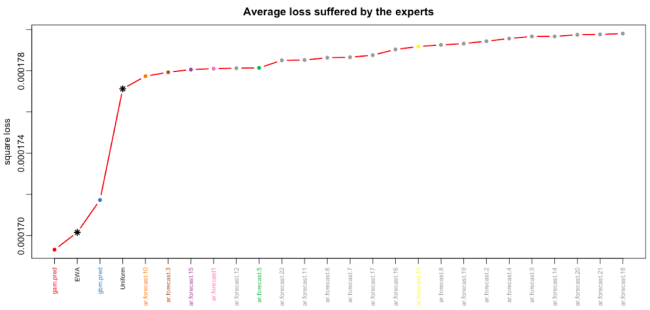
(a) Evolution of Weights



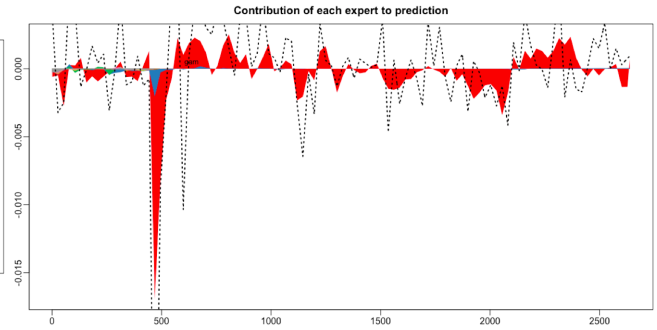
(b) Confidence Intervals on Expert Weights



(c) Prediction Residuals



(d) Mean Square Error of Experts



(e) Contribution of Experts to Prediction

Figure 6.6: EWAF- Copper Combined Expert Set

model in hindsight which is expected as additive models dominate most of the weight in both forecasters. Nevertheless, we should not discount lags completely as the performance of our model does improve from purely non-linear settings (Table 6.1).

6.1.4 Comparison with baseline

One of the questions we initially wanted to answer was what decision horizon is important when making a forecast on future change in closing price. From our

experiments with lags as experts we see that the best time-line to consider in the online setting (Table A.1) is completely different from the important lags in offline setting (Figure 2.5(a)). The offline setting gives a greater preference to immediate lags such as 2,4,9 and 10 compared to online setting, which focuses far more on distant lags such as 12,14,17 and 22. However, both models end up considering lag 1 as the most important.

It is difficult to ascertain why the online setting considers a farther decision horizon as more important when compared to the offline setting. Given the dynamic nature of online learning, it could be more susceptible to noise as we saw in experiments conducted on our artificial data sets in chapter 3. However, the online model with lag experts does slightly outperform the best batch ARMA models as we can see from Table 6.1, hence the result cannot be completely discounted.

Overall, we can see that even the worst online model marginally outperformed the best batch learned models. This is because our offline algorithms take complexity into consideration and compare different models against their AIC values. Hence, a lower number of lags are considered for the best ARMA model. However, complexity is not a major consideration in the online setting since we are evaluating one data point at a time. A naïve offline learning approach where we put 22 past lags in a linear regression yielded slightly better results than our worst online models (MSE $= 1.937 \times 10^{-4}$). Our final online mixture of experts improved on the baselines by around 18.63% for OGD and 17.9% for EWAF. Online gradient descent marginally outperformed exponentially weighted averages at each step, while also being a bit faster. It is also worth noting that the expert models outperformed our stand-alone online models in chapter 3 which did worse than even our baseline models. The biggest jump in improvement can be seen after the introduction of non-linear models with exogenous variables and combining the non-linear models with lags can see further marginal improvement.

Mode of Learning	Model	Forecaster	MSE $\times 10^{-4}$
Online	Combined Set Of Experts	OGD	1.69
		Uniform Weights	1.77
		Best Hindsight Model-GAM	1.68
		EWAF	1.7
		Uniform Weights	1.77
		Best Hindsight Model-GAM	1.68
	Non-Linear Experts	OGD	1.714
		Uniform Weights	1.714
		Best Hindsight Model-GAM	1.715
		EWAF	1.718
		Uniform Weights	1.713
		Best Hindsight Model-GAM	1.715
	Lags as Experts	OGD	1.956
		Uniform Weights	1.98
		Best Hindsight Model-AR(1)	1.956
		EWAF	1.962
		Uniform Weights	1.98
		Best Hindsight Model-AR(1)	1.956
Offline	Best ARMA Model	ARMA(3,0)	1.964
	Baseline Model	Predict Mean	2.004
		Predict 0	2.005

Table 6.1: Comparison Between All Copper Models

6.2 Aluminium

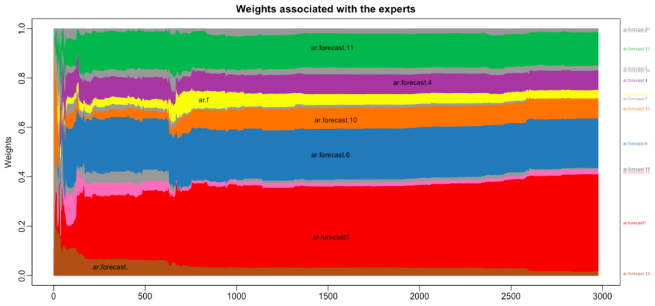
6.2.1 Lags as experts

At first glance, the evolution of weights in our lag experts seems to evolve similarly for aluminium and copper. Indeed, OGD starts to show same level of stability in assigning weights for both commodities, and EWAF shows similar volatility as seen in 6.7(a) and 6.8(a). This result is not unexpected as business cycles affecting copper also affect aluminium and their change in closing price roughly follows a similar pattern as we saw in Fig 2.2. Hence, much of the analysis regarding confidence intervals and residuals done for copper data set overlaps with the aluminium. What is different, however, is the lags considered important by aluminium are not the same as copper. Earlier, we saw that after lag 1, our optimizer put higher weights on lags at distant decision horizons like 22, 14, 17 and 12 for copper. But for aluminium we see a higher bias towards lags at nearer horizons such as 4,6, 10, and 11 for both OGD and EWAF. Full detail of lag weights is given in Figure A.4.

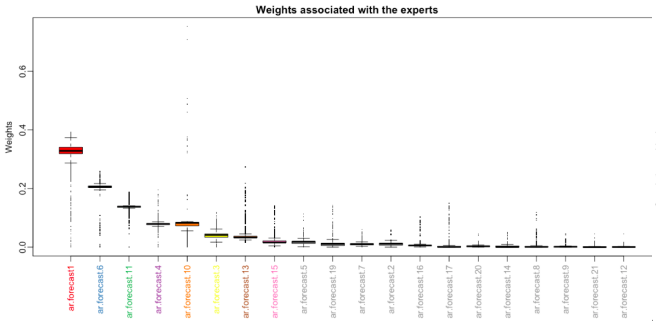
When we plotted our commodity closing price changes in Fig 2.2, we observed that there was an increased volatility in aluminium at the end of our data set after roughly 2500 samples. This is unique to the aluminium database. At this range, it is seen that EWAF almost entirely prefers the first lag to any other, while OGD continues to distribute its weight on multiple experts. From purely performance point of view, both OGD and EWAF get close to the best expert in hindsight and comfortably outperform the uniform weights setting. EWAF marginally outperforms OGD as seen from Table 6.2.

6.2.2 Non-linear experts

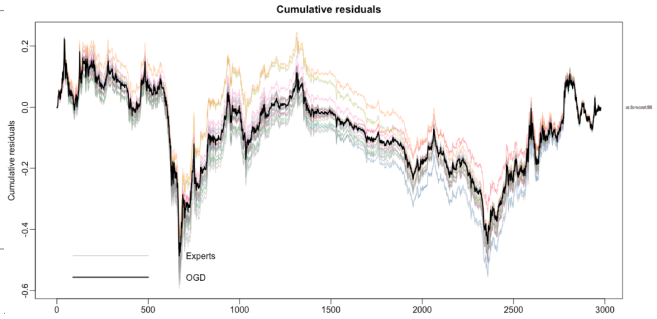
The non-linear experts and exogenous variables chosen for aluminium are same as the ones used in copper. From the results in Fig 6.9, we see that OGD for aluminium and copper are almost exactly identical with initial market volatility slightly favouring GBM before gradual settling with GAMs being slightly ahead.



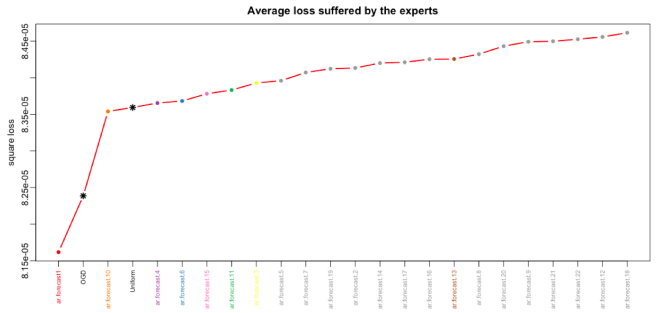
(a) Evolution of Weights



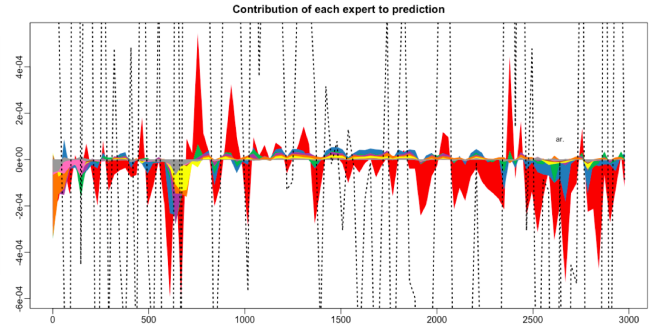
(b) Confidence Intervals on Expert Weights



(c) Prediction Residuals



(d) Mean Square Error of Experts

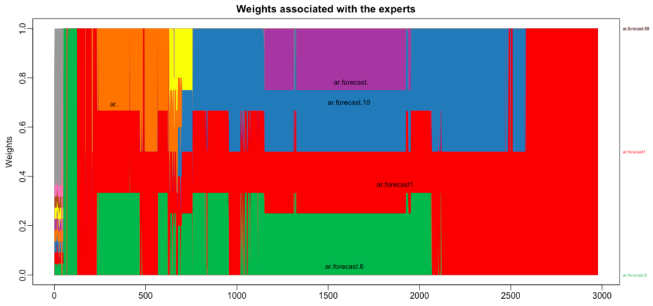


(e) Contribution of Experts to Prediction

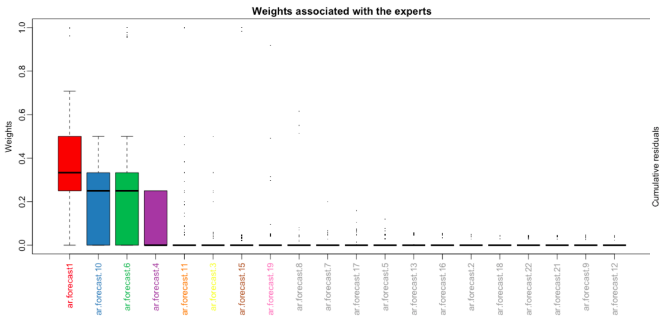
Figure 6.7: OGD-Aluminium Lags as Experts

The confidence intervals are consistent with this observation, and once again we observe negative regrets as OGD slightly outperforms the uniform model and the best model in hindsight.

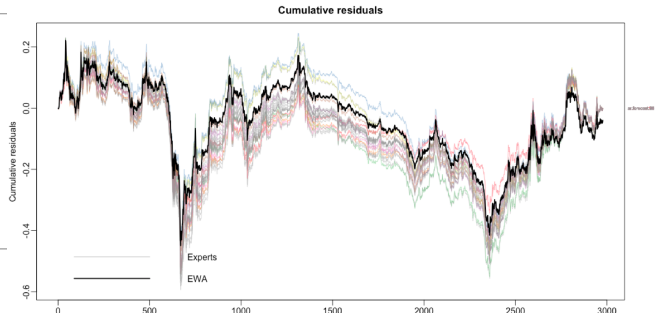
The EWAF optimization for non-linear experts follows a completely different weight evolution for aluminium when compared with copper. We no longer continue to see the equal and consistent distribution of weights of copper. Instead, we see the weights alternate fully between the two experts in the initial 500 sample points, and then again in the last 500 samples points in Fig 6.10(b). This large variation is reflected in the size of our confidence interval box plots.



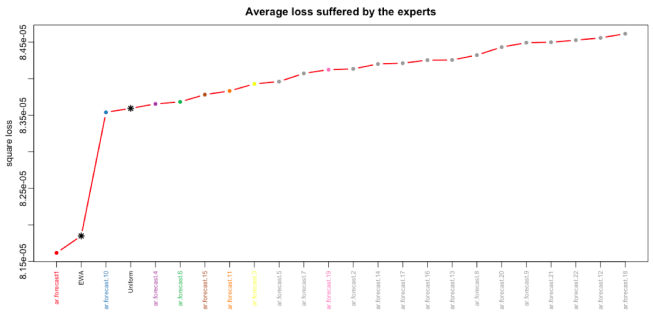
(a) Evolution of Weights



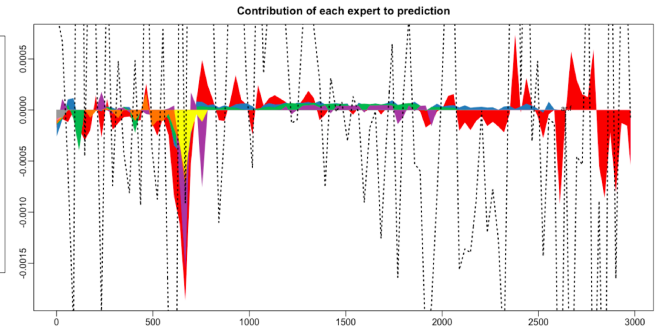
(b) Confidence Intervals on Expert Weights



(c) Prediction Residuals



(d) Mean Square Error of Experts



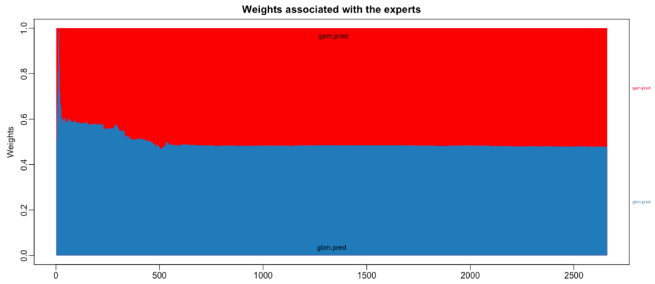
(e) Contribution of Experts to Prediction

Figure 6.8: EWAF-Aluminium Lags as Experts

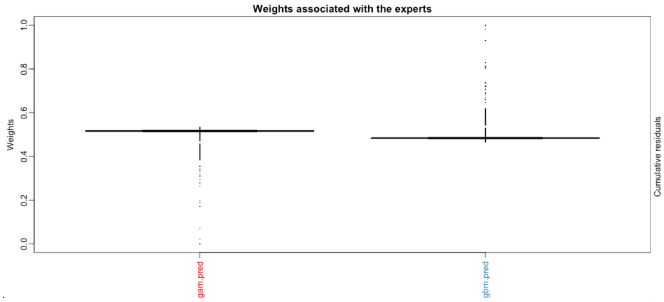
From performance perspective, exponentially weighted average forecaster marginally fails to outperform the uniform model and best model in hindsight.

6.2.3 Combined set of experts

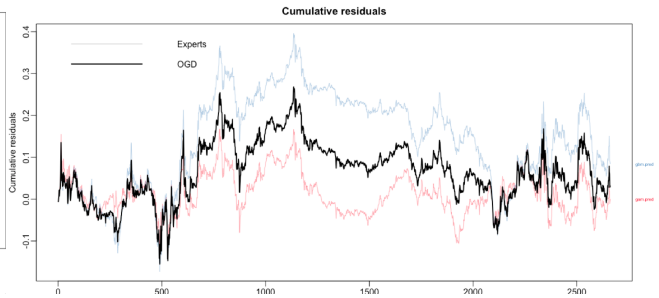
We now combine the lag and non-linear models to create a new mixture of experts for our aluminium database. From Figure 6.11 we see OGD has a very similar weight evolution when compared to copper. OGD continues to assign a visible mass of weights to multiple forecasters but like we saw in copper, additive



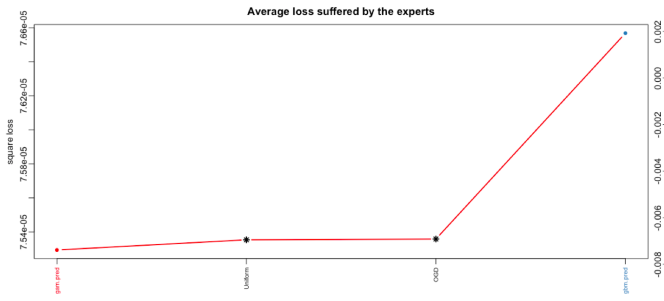
(a) Evolution of Weights



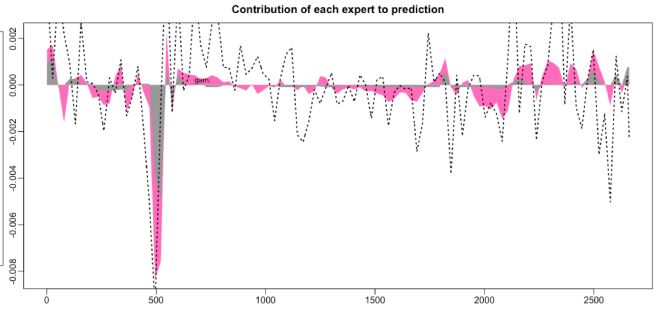
(b) Confidence Intervals on Expert Weights



(c) Prediction Residuals



(d) Mean Square Error of Experts

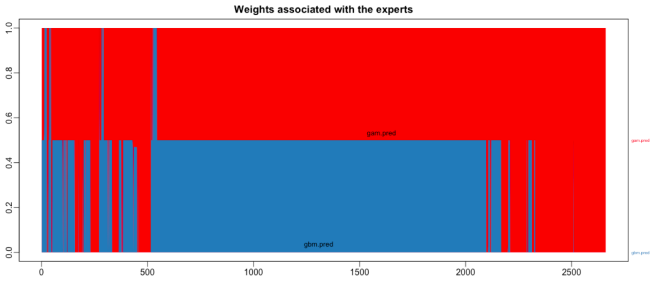


(e) Contribution of Experts to Prediction

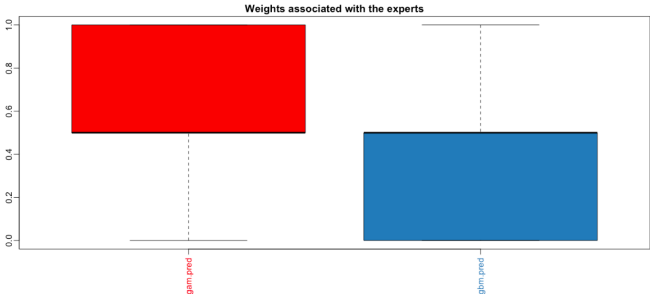
Figure 6.9: OGD -Aluminium Non-Linear Time Series Experts

models heavily dominate it after the first 500 samples. It is interesting to note that unlike copper, the role of gradient boosting machines is greatly diminished and decision horizons like lag 13 and lag 1 play a far more dominant role, especially during the volatility of global financial crisis in the early part of our data set, and in the last 300 samples with the start of new commodity super-cycle.

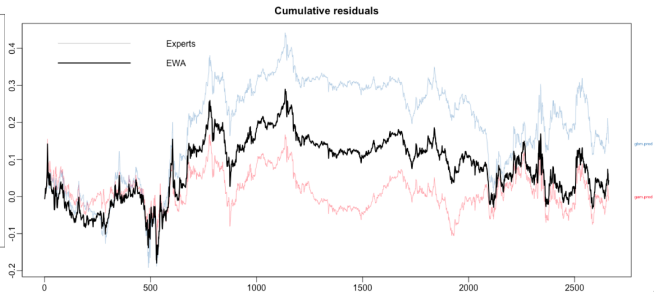
For the case of EWAF in Figure 6.12, we see a complete change in expert preference from copper EWAF, as well as the aluminium OGD. Although GAM dominates in the first 500 samples along with other forecasters, gradient boosting machine fully takes over after this point and this continues even during the new commodity super-cycle observed at the end of the data set. This is



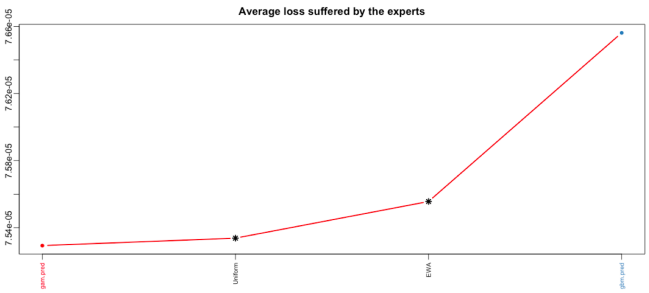
(a) Evolution of Weights



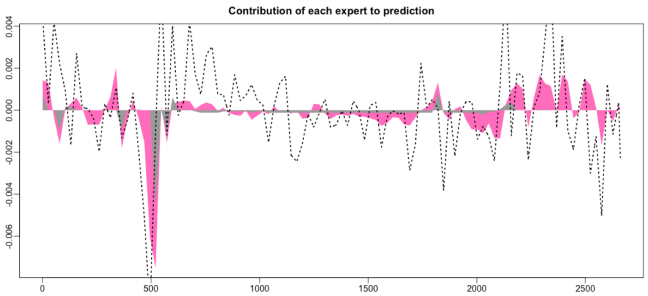
(b) Confidence Intervals on Expert Weights



(c) Prediction Residuals



(d) Mean Square Error of Experts

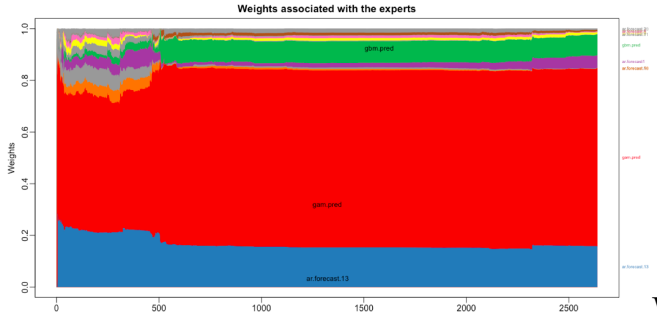


(e) Contribution of Experts to Prediction

Figure 6.10: EWAF-Aluminium Non-Linear Time Series Experts

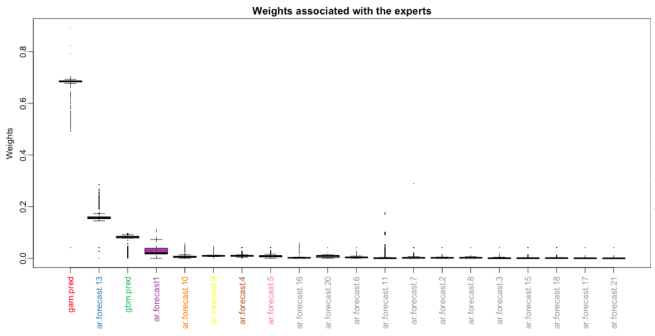
markedly different from copper EWAF where GAM was dominant expert in both its forecasters. The long tails of our confidence interval Figure 6.12(b) shows this reversal taking place.

Performance wise, both our forecasters outperform the uniform weights setting and get really close to the best model in hindsight. EWAF slightly outperforms OGD as seen in Table 6.2.

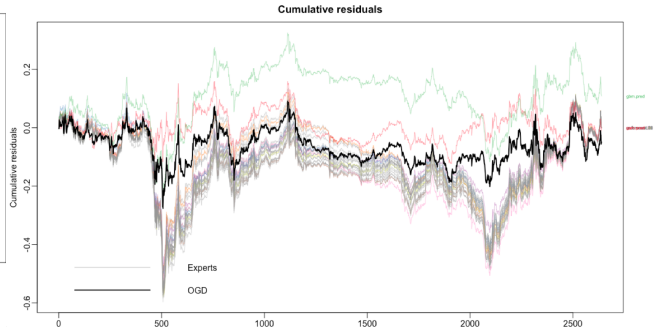


Weight.png

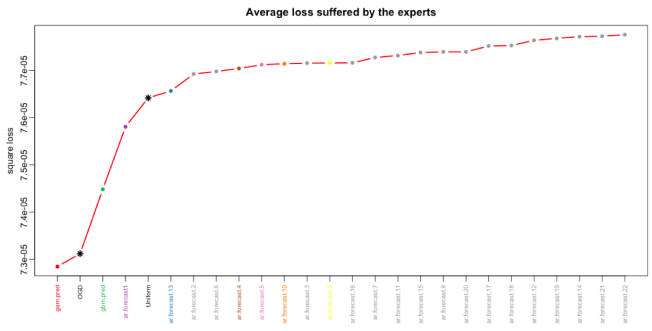
(a) Evolution of Weights



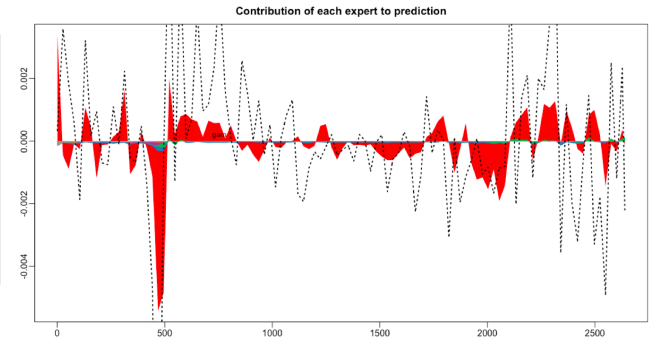
(b) Confidence Intervals on Expert Weights



(c) Prediction Residuals



(d) Mean Square Error of Experts

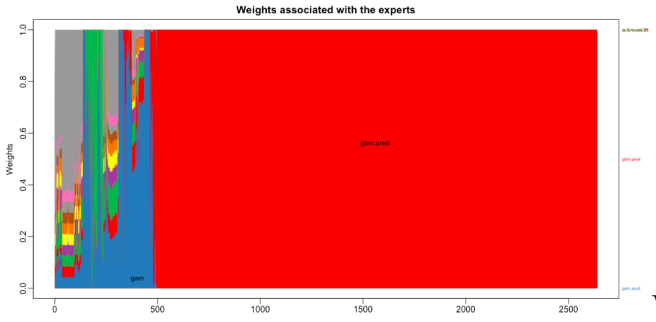


(e) Contribution of Experts to Prediction

Figure 6.11: OGD - Aluminium Combined Expert Set

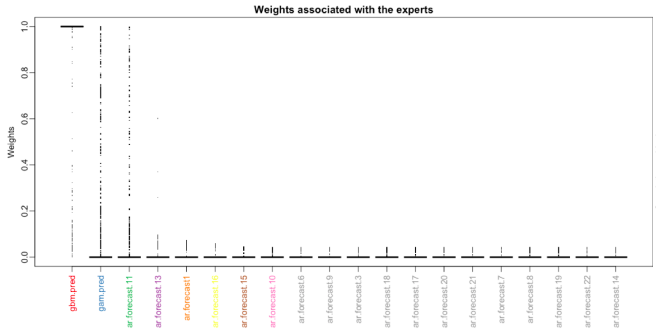
6.2.4 Comparison with baseline

We start our comparison with baseline models by first comparing the feature importance of lags in the offline setting (Figure 2.5(b)) against the weights given to our lag experts in the online setting (Table A.4). Earlier, for copper we noticed a dichotomy in preference where shorter lag horizons were given more importance in offline setting and longer time horizons in online setting. In the case of aluminium we do not observe this dichotomy. Instead, we see the offline and each online forecasters prefer early decision horizons with lags 1, 4, and 10 commonly recurring in top 5 most important lags. Hence, in terms of lag importance, we

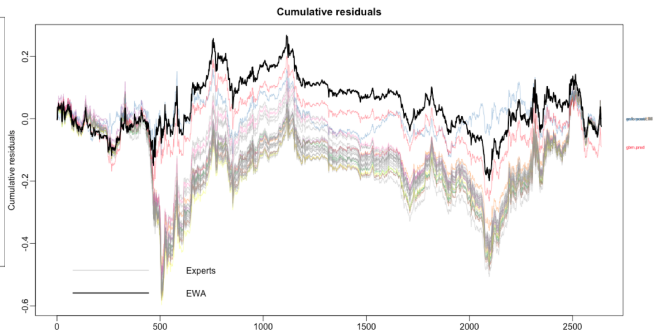


Weight.png

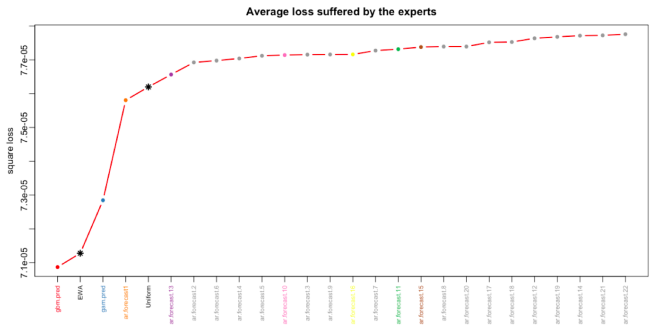
(a) Evolution of Weights



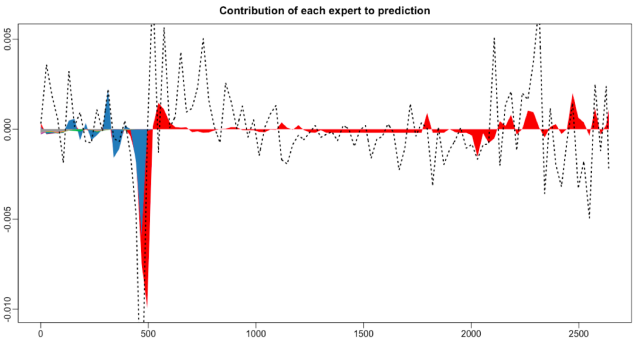
(b) Confidence Intervals on Expert Weights



(c) Prediction Residuals



(d) Mean Square Error of Experts



(e) Contribution of Experts to Prediction

Figure 6.12: EWAF- Aluminium Combined Expert Set

observe a strong correlation in the functioning of online and offline models.

It is difficult to ascertain why we obtain diverging results between copper and aluminium given similar business cycles act on both commodities. However, when we looked closely at our closing price change plots in Fig 2.2, we saw that aluminium is slightly less noisy than copper throughout. This also lends some evidence to our earlier reasoning that online models are more susceptible to noise; hence you see a divergence in results for copper but not for aluminium.

Mode of Learning	Model	Forecaster	MSE $\times 10^{-5}$
Online	Combined Set Of Experts	OGD	7.31
		Uniform Weights	7.65
		Best Hindsight Model-GAM	7.30
		EWAF	7.11
		Uniform Weights	7.8
		Best Hindsight Model-GBM	7.09
	Non-Linear Experts	OGD	7.53
		Uniform Weights	7.53
		Best Hindsight Model-GAM	7.52
		EWAF	7.559
		Uniform Weights	7.54
		Best Hindsight Model-GAM	7.52
	Lags as Experts	OGD	8.23
		Uniform Weights	8.35
		Best Hindsight Model-AR(1)	8.15
		EWAF	8.18
		Uniform Weights	8.35
		Best Hindsight Model-AR(1)	8.15
Offline	Best ARMA Model	ARMA(1,1)	8.25
	Best AR Model	AR(2,0)	8.30
	Baseline Model	Predict Mean	8.604
		Predict 0	8.605

Table 6.2: Comparison Between Aluminium Models

Overall, from Table 6.2 we see that the worst online model marginally outperformed the best offline AR and ARMA models. Similar to copper, our offline models are taking complexity into account and preferring ARMA models with lower lags; a consideration we do not need to focus heavily on in the online setting. A naïve offline implementation where we put all the lags in a linear regression without considering complexity yields slightly better results than our worst online experts (MSE $= 8.04 \times 10^{-5}$). Our final mixture of experts improved on the baseline by 17.7% for OGD and 21% for EWAF, with largest single increase coming after the introduction of non-linear experts and exogenous variables. Our combined set of experts marginally improved on the purely non-linear expert setting. In terms of best forecaster for our experts, we obtain a mixed result with EWAF performing better against OGD for pure lags and combined set of experts, whereas OGD performed faster and gave better results for non-linear experts.

Chapter 7

Conclusion

To conclude our study we will now summarise our main results. The first question we wanted to answer was whether online time series models have comparable predictive power to their offline counter parts, and if we can extend their application to our commodities data. In order to bring our offline time series models (ARMA) to the online setting we made use of the improper learning framework of Anava et al., with the objective of getting over the non-convexity issue when transitioning between the two settings. Our experiments on artificial data set and temperature data showed promising results, however our results were poor for the stock market and commodities data set as our online model failed to perform anywhere close to the offline models. We speculated that the reason for this is due to a lack of model sophistication at some level.

This led us to our second question: can we improve the predictive power of our online models by taking them to the expert setting? Here, we tried to optimize using different mixtures of online models simultaneously. Our first mixture tackled the issue of model sophistication by considering a larger set of autoregressive lags for optimization, while the second mixture approached the issue by introducing new exogenous variables modelled by two non-linear experts: GAM and GBM. And our final mixture combined the first two mixtures together. These changes led to a significant improvement in the prediction power of our online models.

Finally, we compared the results of our online experts against the baseline and best offline models. We observed our first expert set of AR mixtures was now marginally outperforming the best offline ARMA models. When we combine these with our non-linear experts, our final online models show significant improvement of roughly 18% for copper, and 17.7-21% for aluminium over the baseline depending on the forecaster used.

As we can see from the improvements in our result, online learning, and especially expert learning have a tremendous scope for application in the field of finance. Given the gains in performance achieved after the introduction of non-linear models, one obvious direction to continue our research would be to create better combinations of non-linear online experts. In this regard, our next goal would be to explore techniques like multiple kernel learning and finding more effective splines than the thin plate spline used in this study.

We must point out that we do not want to completely disregard linear models for online learning. Techniques such as generalized autoregressive conditional heterskedasticity (GARCH) have been effective in the past at predicting stock market prices and their theoretical foundation for application in online setting has been recently laid.[7] Thinking beyond the immediate framework of this study, we may look at more model agnostic experts for online setting such as those available in the field of reinforcement learning.

Ultimately, given the results of this study, we believe that machine learning and online learning has a lot of potential for application in predicting price of commodity futures.

Appendix A

Distribution of weights for our expert models

All values below $\times 10^{-4}$ are omitted.

Expert	Weights OGD	Weights EWAF
AR(1)	0.395	0.2
AR(2)	0.00286	0
AR(3)	0.0489	0
AR(4)	0.0518	0
AR(5)	0.0032	0
AR(6)	0.0242	0
AR(7)	0.00202	0
AR(8)	0.0401	0
AR(9)	0.000818	0
AR(10)	0.000943	0
AR(11)	0.00322	0
AR(12)	0.0504	0.2
AR(13)	0.00205	0
AR(14)	0.0839	0.2
AR(15)	0.00073	0
AR(16)	0	0
AR(17)	0.0859	0.2
AR(18)	0.00229	0
AR(19)	0	0
AR(20)	0.000485	0
AR(21)	0.00577	0
AR(22)	0.201	0.2

Table A.1: Distribution of Copper Lag Weights

Expert	Weights OGD	Weights EWAF
GAM	0.636	0.364
GBM	0.5	0.5

Table A.2: Distribution of Copper Non-Linear Expert Weights

Expert	Weights OGD	Weights EWAF
GAM	0.674	0.816
GBM	0.159	0.177
AR(1)	0.00964	0.00007
AR(2)	0.0004	0
AR(3)	0.041	0.0001
AR(4)	0.0001	0
AR(5)	0.0254	0.0001
AR(6)	0.00429	0
AR(7)	0.00654	0
AR(8)	0.000372	0
AR(9)	0.00023	0
AR(10)	0.0245	0.00396
AR(11)	0.00372	0
AR(12)	0.0119	0.0008
AR(13)	0.0014	0
AR(14)	0	0
AR(15)	0.0204	0.00151
AR(16)	0.00143	0
AR(17)	0.00531	0
AR(18)	0	0
AR(19)	0	0
AR(20)	0	0
AR(21)	0	0
AR(22)	0.0102	0

Table A.3: Distribution of Copper All Expert Weights

Expert	Weights OGD	Weights EWAF
AR(1)	0.395	1
AR(2)	0.12	0
AR(3)	0.349	0
AR(4)	0.0786	0
AR(5)	0.0188	0
AR(6)	0.201	0
AR(7)	0	0
AR(8)	0.000764	0
AR(9)	0.0001	0
AR(10)	0.0814	0
AR(11)	0.136	0
AR(12)	0	0
AR(13)	0.0176	0
AR(14)	0.00044	0
AR(15)	0.0191	0
AR(16)	0.000305	0
AR(17)	0.000854	0
AR(18)	0	0
AR(19)	0.0055	0
AR(20)	0.0006	0
AR(21)	0.000247	0
AR(22)	0.00017	0

Table A.4: Distribution of Aluminium Lag Weights

Expert	Weights OGD	Weights EWAF
GAM	0.525	1
GBM	0.475	0

Table A.5: Distribution of Aluminium Non-Linear Expert Weights

Expert	Weights OGD	Weights EWAF
GAM	0.69	0
GBM	0.0809	1
AR(1)	0.0474	0
AR(2)	0	0
AR(3)	0.0027	0
AR(4)	0.0060	0
AR(5)	0.0020	0
AR(6)	0.002	0
AR(7)	0	0
AR(8)	0	0
AR(9)	0.0088	0
AR(10)	0	0
AR(11)	0.0002	0
AR(12)	0	0
AR(13)	0.158	0
AR(14)	0	0
AR(15)	0	0
AR(16)	0	0
AR(17)	0	0
AR(18)	0.0004	0
AR(19)	0	0
AR(20)	0	0
AR(21)	0	0
AR(22)	0	0

Table A.6: Distribution of Aluminium All Expert Weights

Bibliography

[1] *2000s commodities boom*. https://en.wikipedia.org/wiki/2000s_commodities_boom#Copper. Accessed: 22-08-2018.

[2] Amit Agarwal and Elad Hazan. *Efficient Algorithms for Online Game Playing and Universal Portfolio Management*. Tech. rep. 2006.

[3] Amit Agarwal et al. "Algorithms for portfolio management based on the Newton method". In: *ICML*. 2006.

[4] Francesca Dominici et al. "On the Use of Generalized Additive Models in Time-Series Studies of Air Pollution and Health". In: *American Journal of Epidemiology* 156 (2002).

[5] *Aluminium prices approach 6-year high amid commodity boom*. https://www.ft.com/content/80928124-88a8-11e7-8bb1-5ba57d47eff7. 2017.

[6] *An Introduction To Online Machine Learning*. https://dziganto.github.io/data%20science/online%20learning/python/scikit-learn/An-Introduction-To-Online-Machine-Learning/. 27-07-2017.

[7] Oren Anava and Shie Mannor. "Heteroscedastic Sequences: Beyond Gaussianity". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 755–763. URL: http://dl.acm.org/citation.cfm?id=3045390.3045471.

[8] Oren Anava et al. "Online Learning for Time Series Prediction". In: *eprint arXiv:1302.6927* (2013).

[9] Chakriya B. and Aasim M. "Forecasting Commodity Prices: Futures Versus Judgment". In: *IMF WORKING PAPER* (2004).

[10] Jean B. et al. "Forecasting Commodity Prices: GARCH, Jumps, and Mean Reversion". In: *Bank of Canada Working Paper* 2006-14 (2006).

[11] Wolfgang Bessler and Dominik Wolff. "Do commodities add value in multi-asset portfolios? An out-of-sample analysis for different investment strategies". In: *Journal of Banking and Finance* 60 (2015), pp. 1–20.

[12] Avrim Blum. "On-line Algorithms in Machine Learning". In: *Developments from a June 1996 Seminar on Online Algorithms: The State of the Art*. Berlin, Heidelberg: Springer-Verlag, 1998, pp. 306–325. ISBN: 3-540-64917-4. URL: http://dl.acm.org/citation.cfm?id=647371.723908.

[13] Hamparsum Bozdogan. "Model selection and Akaike's Information Criterion (AIC): The general theory and its analytical extensions". In: *Springer* (1987).

[14] Daniel Buncic and Carlo Moretto. "Forecasting copper prices with dynamic averaging and selection models". In: *North American Journal of Economics and Finance* 33 (2015).

[15] Nicolo' Cesa-Bianchi and Gabor Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[16] *Commodities super-cycle is 'taking a break'*. https://www.scmp.com/business/commodities/article/1279041/super-cycle-taking-break. Accessed: 22-08-2018.

[17] Ashok Cutkosy and Róbert Busa-Fekete. "Distributed Stochastic Optimization via Adaptive SGD". In: *eprint arXiv:1802.05811* (2018).

[18] Jerome H. Friedman. "Stochastic Gradient Boosting". In: *Comput. Stat. Data Anal.* 38.4 (Feb. 2002), pp. 367–378. ISSN: 0167-9473. DOI: 10.1016/S0167-9473(01)00065-2. URL: http://dx.doi.org/10.1016/S0167-9473(01)00065-2.

[19] Dan Garber and Elad Hazan. *Projection Free Online Learning*. https://www.slideserve.com/thetis/projection-free-online-learning.

[20] Trevor Hastie and Robert Tibshirani. "Generalized Additive Models". In: *New York: Chapman Hall* (1990).

[21] Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The Elements of Statistical Learning*. Stanford University Library, 2013.

[22] Elad Hazan. *Introduction to Online Convex Optimization*. NOW, 2016.

[23] Sanchez Lasheras et al. "Forecasting the COMEX copper spot price by means of neural networks and ARIMA models". In: *Resource Policy* 45.10 (2015), pp. 37–43.

[24] Nick Littlestone and Manfred Warmuth. "The Weighted Majority Algorithm". In: *Information and Computation* 108 (1994).

[25] Jordan M. and Jacobs R. "Hierarchical Mixtures of Experts and the EM Algorithm". In: *Neural Computation* 6 (1994).

[26] Andrew Ng. "CS229 Lecture Notes. The perceptron and large margin classifiers". In: (2017).

[27] Adegbenga Olayiwola. "Forecasting Copper Spot Prices: A Knowledge-Discovery Approach". In: (2016).

[28] *Online Machine Learning*. https://en.wikipedia.org/wiki/Online_machine_learning. Accessed: 22-08-2018.

[29] Anja Rossen. "What are metal prices like? Co-movement, price cycles and long-run trends". In: *Resource Policy* 45.10 (2015), pp. 255–276.

[30] Lukas Schulze. "Using Artificial Neural Networks to generate trading signals for crude oil, copper and gold futures". In: (2016).

[31] Souhaib T. and Rob H. "A gradient boosting approach to the Kaggle load forecasting competition". In: *International Journal of Forecasting* 30 (2014).

[32] *Trading Rules*. http://www.shfe.com.cn/en/Rules/SHFERules/TradingRules/. Accessed: 22-08-2018.

[33] *Universal Portfolio Algorithm*. https://en.wikipedia.org/wiki/Universal_portfolio_algorithm. Accessed: 22-08-2018.

[34] Simon N. Wood. "Thin Plate Regression Splines". In: *Journal of the Royal Statistical Society. Series B (Statistical Methodology)* 65.1 (2003), pp. 95–114. ISSN: 13697412, 14679868. URL: http://www.jstor.org/stable/3088828.

[35] S.N Wood. *Generalized Additive Models: An Introduction with R*. 2nd ed. Chapman and Hall/CRC, 2017.

[36] Goude Y., Nedellec R. and Kong N. "Local Short and Middle Term Electricity Load Forecasting With Semi-Parametric Additive Models". In: *IEEE TRANSACTIONS ON SMART GRID* 5 (2014).

[37] Assaf Zeevi, Ron Meir and Robert Adler. "Time Series prediction Using Mixtures of Experts". In: *NIPS 96* (1996).

[38] Martin Zinkevich. "Online Convex Programming and Generalized Infinitesimal Gradient Ascent". In: *ICML* (2003).