Adaptive calibration for binary classification

Vladimir Vovk, Ivan Petej, and Alex Gammerman

практические выводы
теории вероятностей
могут быть обоснованы
в качестве следствий
гипотез о *предельной*
при данных ограничениях
сложности изучаемых явлений

On-line Compression Modelling Project (New Series)

Working Paper #35

First posted July 4, 2021. Last revised July 6, 2021.

Project web site:
http://alrw.net

Abstract

This note proposes a way of making probability forecasting rules less sensitive to changes in data distribution, concentrating on the simple case of binary classification. This is important in applications of machine learning, where the quality of a trained predictor may drop significantly in the process of its exploitation. Our techniques are based on recent work on conformal test martingales and older work on prediction with expert advice, namely tracking the best expert.

Contents

1 Introduction

A common problem in applications of machine learning is that, soon after a predictor is trained, the distribution of the data may change, and the predictor may need to be retrained. There are efficient ways of online detection of a change in distribution, such as using conformal test martingales [15], but there are inevitably awkward gaps between the change in distribution and its detection and between the detection of the change and the deployment of a retrained predictor.

This paper proposes a way of preventing a catastrophic drop in the quality of the trained predictor when the data distribution changes. Given a base predictor, our procedure gives an enhanced predictor that is more robust to changes in the data distribution. To use Anscombe's [1] insurance metaphor (repeatedly mentioned in [8]), our procedure provides an insurance policy (hopefully not too expensive) against such changes.

The case of regression was discussed in an earlier paper [16], and in this note we concentrate on the simpler case of binary classification. We will assume that the label space is $\{0, 1\}$ (except for Section 5, in which we will use a dataset, `Bank Marketing`, with label space $\{1, 2\}$). Suppose we are given a predictive system that maps past data and an object x to a number $p \in [0, 1]$, interpreted as the predicted probability that x's label is 1. We will refer to it as our *base predictive system*.

We will be interested in two seemingly different questions about the base predictive system:

Online testing Can we gamble successfully against the base predictive system (at the odds determined by its predicted probabilities)? We are interested in online testing [15], i.e., in constructing test martingales with respect to the base predictive system that take large values on the actual sequence of observations.

Online prediction Can we improve the base predictive system, modifying its predictions p_n to better predictions p'_n?

If the quality of online prediction is measured using the log-loss function [6], the difference between the two questions almost disappears, as we will see in Sections 3 and 5.

After discussing online testing in Section 2 and online prediction in Section 3, we will give an example of a theoretical performance guarantee for our prediction procedure (a straightforward application of a known result) in Section 4. In Section 5 we report encouraging experimental results, and Section 6 concludes.

2 Testing predictions by betting

We consider a potentially sequence of *actual observations* $z_1, z_2, \ldots$, each consisting of two components: $z_n = (x_n, y_n)$, where $x_n \in \mathbf{X}$ is an *object* chosen

Algorithm 1 Jumper betting martingale ($(p_1, p_2, \dots) \mapsto (S_1, S_2, \dots)$)

$C_\epsilon := 1/|\mathbf{E}|$ for all $\epsilon \in \mathbf{E}$
$C := 1$
for $n = 1, 2, \dots$:
    for $\epsilon \in \mathbf{E}$: $C_\epsilon := (1 - J)C_\epsilon + (J/|\mathbf{E}|)C$
    for $\epsilon \in \mathbf{E}$: $C_\epsilon := C_\epsilon B_{f_\epsilon(p_n)}(\{y_n\})/B_{p_n}(\{y_n\})$
    $S_n := C := \sum_{\epsilon \in \mathbf{E}} C_\epsilon$

from an *object space* $\mathbf{X}$, and $y_n \in \{0, 1\}$ is a binary label. A *predictive system* is a function that maps any object x and any finite sequence of observations $z_1, \dots, z_i$ (intuitively, the past data) for any $i \in \{0, 1, \dots\}$ to a number $p \in [0, 1]$ (intuitively, the probability that the label of x is 1). Fix a *base predictive system*, and let $p_1, p_2, \dots$ be its predictions for the actual observations: p_n is the prediction output by the base predictive system on x_n and $z_1, \dots, z_{n-1}$; it is interpreted as the predicted probability that $y_n = 1$. (In this note we do not need any measurability assumptions; in particular, $\mathbf{X}$ is not supposed to be a measurable space.)

In this paper we are mostly interested in the special case where the output p of the base predictive system depends only on x and not on $z_1, \dots, z_i$. In this case we will say that our predictive system is a *prediction rule*. A typical way in which prediction rules appear in machine learning is as result of training a prediction algorithm. In Section 5 we will be only interested in a prediction rule, but for now we do not impose this restriction.

Our online testing procedure is given as Algorithm 1. One of its two parameters is a finite family $f_\epsilon : [0, 1] \to [0, 1]$, $\epsilon \in \mathbf{E}$, of *calibrating functions*. The intuition behind f_ϵ is that we are trying to improve the base predictions p_n, or *calibrate* them; the idea is to use a new prediction $f_\epsilon(p_n)$ instead of p_n. In the experimental Section 5 we will use a subset of the family

$$f_\epsilon(p) := p + \epsilon p(1 - p), \tag{1}$$

where $\epsilon \in [-1, 1]$. For $\epsilon > 0$ we are correcting for the forecasts p being underestimates of the true probability of 1, while for $\epsilon < 0$ we are correcting for p being overestimates. Our family is required to be finite, and we choose $\mathbf{E} := \{-1, -0.5, 0, 0.5, 1\}$.

We do not know in advance which f_ϵ will work best, and moreover, it seems plausible that suitable values of ϵ will change over time. Therefore, we use the idea of "tracking the best expert" [7]. Algorithm 1 uses the notation B_p, $p \in \{0, 1\}$, for the Bernoulli distribution on $\{0, 1\}$ with parameter p: $B_p(\{1\}) = p$. To each sequence $\theta = (\theta_1, \theta_2, \dots)$ of elements of $\mathbf{E}$ corresponds the *elementary test martingale*

$$\prod_{i=1}^{n} \frac{B_{f_{\theta_i}(p_i)}(\{y_i\})}{B_{p_i}(\{y_i\})}, \quad n = 0, 1, \dots. \tag{2}$$

The other parameter of the Jumper betting martingale of Algorithm 1 is $J \in$

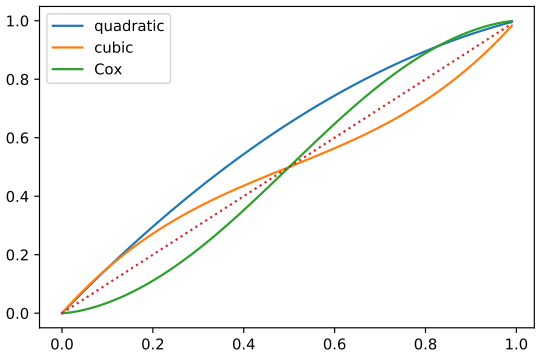

Figure 1: Examples of calibration functions.

$(0,1]$, the *jumping rate*. This martingale is obtained by "derandomizing" (to use the terminology of [13]) the stochastic test martingale corresponding to the probability measure μ on $[0,1]^\infty$ defined as the probability distribution of the following Markov chain with state space $\mathbf{E}$. The initial state θ_1 is chosen from the uniform probability measure on $\mathbf{E}$ (line 1 of Algorithm 1), and the transition function prescribes maintaining the same state with probability $1-J$ and, with probability J, choosing a new state from the uniform probability measure on $\mathbf{E}$ (line 4).

We derandomize the stochastic test martingale by averaging,

$$S_n := \int \prod_{i=1}^{n} \frac{B_{f_{\theta_i}(p_i)}(\{y_i\})}{B_{p_i}(\{y_i\})} \mu(\mathrm{d}\theta),$$

which gives us a deterministic test martingale.

In Section 5 we will see an example where already the simple choice (1) leads to very successful betting for a benchmark dataset. However, there are numerous other natural calibration functions, some of which are shown in Figure 1. The function in blue is in the quadratic family (1); these functions are fully above, fully below, or (for $\epsilon = 0$) situated on the bisector of the first quadrant (shown as the dotted line). In many situations other calibration functions will be more suitable. For example, it is well known that untrained humans tend to be overconfident [9, Part VI, especially Chapter 22]. An example of a calibration function correcting for overconfidence is the cubic function

$$f_{a,b}(p) := p + ap(p-b)(p-1), \tag{3}$$

where $(a,b) \in [0,1]^2$. An example of such a function is shown in Figure 1 in orange. The meaning of the parameters is that b is the value of p (such as 0.5) that we believe does not need correction, and that a indicates how aggressively we want to correct for overconfidence ($a < 0$ meaning that in fact

Algorithm 2 Jumper predictor $((p_1, p_2, \dots) \mapsto (p'_1, p'_2, \dots))$

$C_\epsilon := 1$ for all $\epsilon \in \mathbf{E}$
for $n = 1, 2, \dots$:
  $C := \sum_{\epsilon \in \mathbf{E}} C_\epsilon$
  for $\epsilon \in \mathbf{E}$: $C_\epsilon := C_\epsilon / C$
  for $\epsilon \in \mathbf{E}$: $C_\epsilon := (1 - J) C_\epsilon + J / |\mathbf{E}|$
  $p'_n := \sum_{\epsilon \in \mathbf{E}} f_\epsilon(p_n) C_\epsilon$
  for $\epsilon \in \mathbf{E}$: $C_\epsilon := C_\epsilon B_{f_\epsilon(p_n)}(\{y_n\})$

we are correcting for underconfidence). If the predictor predicts a p that is close to 0 or 1, we correct for his overconfidence (assuming $a > 0$) by moving p towards the neutral value b. Alternatively, we could use Cox's [2, Section 3] calibration functions, one-parameter

$$f_\beta(p) := \frac{p^\beta}{p^\beta + (1-p)^\beta}, \tag{4}$$

where $\beta \in \mathbb{R}$, or two-parameter

$$f_{\alpha,\beta}(p) := \frac{p^\beta \exp(\alpha)}{p^\beta \exp(\alpha) + (1-p)^\beta}, \tag{5}$$

where $\alpha, \beta \in \mathbb{R}$. An example of a function in the class (4) is shown in Figure 1 in green.

3 Prediction algorithms based on adaptive calibration

For any predictive system, we define its *gale* [12, Chapters 15–17] as a function mapping any finite sequence of observations to the product $B_{p_1}(y_1) \cdots B_{p_n}(y_n)$, where n is the number of observations, $y_1, \dots, y_n$ are their labels, and $p_1, \dots, p_n$ are the predictions for those observations. We regard the gale as the capital process of a player playing an extremely challenging game: his capital cannot go up, and for it not to go down he has to predict with the probability measure concentrated on the true outcome. The gale of the base predictive system will be referred to as the *base gale*.

Remark 1. The notion of a gale is very similar to Cox's [3] notion of partial likelihood, but we cannot say that a gale is partial in any sense (since it is not part of a fuller likelihood function: there is no probability measure on the objects [12, Section 10.5]).

For simplicity, we will discuss only positive gales and martingales (i.e., those that do not take zero values). This will be sufficient for the considerations of Section 5. Each test martingale with respect to the base predictive system is

the ratio of a gale to the base gale, and vice versa. This establishes a bijection between test martingales and gales. Algorithm 2 is the predictive system whose gale corresponds to the test martingale of Algorithm 1.

Algorithm 1 is a special case the Aggregating Algorithm (AA) [13] corresponding to the *log-loss function*

$$\lambda(y,p) := \begin{cases} -\log p & \text{if } y = 1 \\ -\log(1-p) & \text{if } y = 0 \end{cases} \tag{6}$$

(the logarithm is typically natural, but in Section 5 we will consider decimal logarithms). Analogously to (2), to each sequence $\theta = (\theta_1, \theta_2, \dots)$ corresponds the *elementary predictor* that outputs, at each step n,

$$p'_n := f_{\theta_n}(p_n), \quad n = 1, 2, \dots,$$

as its prediction. The AA is described in [13, Section 2], and in our case of the log-loss function the optimal in a natural sense exponential learning rate is $\beta := \exp(-1)$, and the AA coincides with the APA ("Aggregating Pseudo-Algorithm"). The prior distribution μ on the elementary predictors is as described in the previous section.

At the beginning of step n, the prior weight of the elementary predictor θ is multiplied by

$$\prod_{i=1}^{n-1} B_{f_{\theta_i}(p_i)}(\{y_i\}),$$

in the sense that the posterior distribution is

$$\mu'(\mathrm{d}\theta) = \mu(\mathrm{d}\theta) \prod_{i=1}^{n-1} B_{f_{\theta_i}(p_i)}(\{y_i\}).$$

Let $D_n(\epsilon)$ be the total posterior (unnormalized) weight of the elementary predictors θ that are in state ϵ at the beginning of step n, i.e., $\theta_n = \epsilon$. We start from $D_1(\epsilon) = 1/|\mathbf{E}|$ (for all ϵ), and the recursion is

$$D_n(\epsilon) := (1-J) D'_n(\epsilon) + \frac{J}{|\mathbf{E}|} \sum_{\epsilon' \in \mathbf{E}} D'_n(\epsilon'),$$

where

$$D'_n(\epsilon') := B_{f_{\epsilon'}(p_{n-1})}(\{y_{n-1}\}) D_{n-1}(\epsilon')$$

for all $\epsilon' \in \mathbf{E}$. We can see that $D_n(\epsilon) \propto C_\epsilon$, where C_ϵ is computed in line 4 of Algorithm 1 at iteration n, and $\propto$ means coincidence to within a positive factor independent of ϵ. The AA prediction can now be computed as

$$\frac{\sum_\epsilon f_\epsilon(p_n) D_n(\epsilon)}{\sum_\epsilon D_n(\epsilon)}.$$

This again results in the prediction algorithm given as Algorithm 2. The variables C_ϵ in it are different from the C_ϵ in Algorithm 1, but the difference is not essential (a positive factor independent of ϵ); the former are the normalized versions of the latter.

4 An example of a theoretical guarantee

Theoretical performance guarantees for Algorithm 2 and related procedures is potentially a big topic, but we will give only a very simple result, a special case of a known result for the AA. In the context of the AA, a gale is $\exp(-L)$, where L is a loss process. The following lemma is the main property of the AA.

Lemma 1. *The gale of the AA is the average of the elementary predictors' gales.*

Proof. For a proof, see [14, Lemma 1]. □

Let us use the notation

$$\mathrm{Loss}(p_1,\dots,p_n \mid y_1,\dots,y_n) := \sum_{i=1}^{n} \lambda(y_i,p_i)$$

for the cumulative log-loss of predictions $p_i \in [0,1]$ on labels $y_i \in \{0,1\}$, where λ is defined by (6). The simplest performance guarantee for Algorithm 2 is

$$\begin{aligned}\mathrm{Loss}(p'_1,\dots,p'_n \mid y_1,\dots,y_n) \le{}& \mathrm{Loss}(f_{\theta_1}(p_1),\dots,f_{\theta_n}(p_n) \mid y_1,\dots,y_n)\\ &+ \log|\mathbf{E}| + k\log(|\mathbf{E}|-1) + k\log\frac{1}{J} + (n-k-1)\log\frac{1}{1-J},\end{aligned}$$

for any n and any sequence $f_{\theta_1},\dots,f_{\theta_n}$ of calibrating functions (from the family (f_ϵ)), where $k = k(\theta_1,\dots,\theta_n)$ is the number of switches,

$$k := \left|\{i \in \{1,\dots,n-1\} : \theta_i \ne \theta_{i+1}\}\right|.$$

If we further average Algorithm 2 over the uniform probability measure over $\epsilon \in [0,1]$, we obtain the guarantee

$$\begin{aligned}\mathrm{Loss}(p'_1,\dots,p'_n \mid y_1,\dots,y_n) \le{}& \mathrm{Loss}(f_{\theta_1}(p_1),\dots,f_{\theta_n}(p_n) \mid y_1,\dots,y_n)\\ &+ \log|\mathbf{E}| + k\log(|\mathbf{E}|-1) + (k+1)\log n,\end{aligned}$$

again for any n and any sequence of calibrating functions, with k defined in the same way. This follows from the results of [13, Section 3.3] (such as Theorem 2) and [7], and can be easily deduced from Lemma 1.

5 Experimental results

In this section we report results of our experiments with the Bank Marketing dataset (the only dataset in the top twelve most popular datasets at the UC Irvine Machine Learning Repository that fits the scenario of Section 1; we will use, however, the full version of this dataset as given at the `openml.org` repository, since it is easy to do from `scikit-learn`). The dataset consists of 45,211 observations representing telemarketing calls for selling long-term deposits offered by a Portuguese retail bank, with data collected from 2008 to 2013 [10].

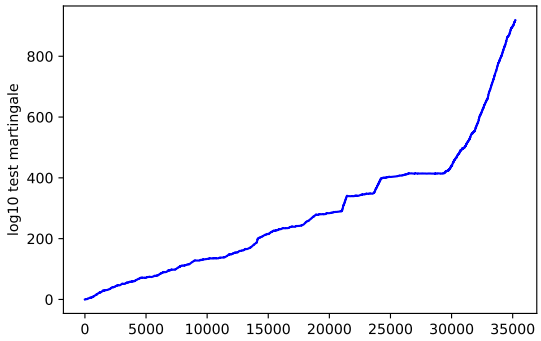

Figure 2: The Jumper test martingale.

The labels are 1 or 2, with the 2s (indicating a successful sale) comprising only 12% of all labels.

The observations are listed in chronological order. We took the first 10,000 observations as the training set and trained a random forest with default parameters and random seed 2021 on it (the random forest method gives the best results for this dataset in our preliminary experiments; this element of data snooping appears harmless since we are interested in improving the base predictive system, and it is natural to expect similar or better results for less successful prediction algorithms). The random forest often outputs probabilities of success that are equal to 0 or 1, and when such a prediction turns out to be wrong (which happens repeatedly), the log-loss is infinite. It is natural to truncate a probability $p \in [0,1]$ of 2 to the interval $[\epsilon, 1-\epsilon]$ replacing p by

$$p^* := \begin{cases} \epsilon & \text{if } p \le \epsilon \\ p & \text{if } p \in (\epsilon, 1-\epsilon) \\ 1-\epsilon & \text{if } p \ge 1-\epsilon, \end{cases}$$

where we set $\epsilon := 0.1$ (in `scikit-learn`, $\epsilon = 10^{-15}$, but $\epsilon := 0.1$ leads to significantly better results). The resulting prediction rule is our base predictive system. After we find it, we never use the training set again, and the numbering of observations starts from the first element of the test set (i.e., the dataset in the chronological order without the training set).

Figure 2 shows the trajectory of $\log_{10} S_n$, $n = 1, \dots, 35211$, where S_n is the value of the Jumper test martingale over the test set with the jumping rate $J := 0.01$ and the family (1) with $\epsilon \in \mathbf{E} := \{-1, -0.5, 0.0.5, 1\}$. It is interesting that the steepest growth of the test martingale (on the log scale) starts towards the end of the dataset, long after the financial crisis of 2007–2008 ended. The final value of the test martingale in Figure 2 is approximately $10^{919.3}$.

Figure 3 gives the ROC curve for the random forest and the random forest enhanced by Algorithm 2. We can see that the improvement is substantial. In

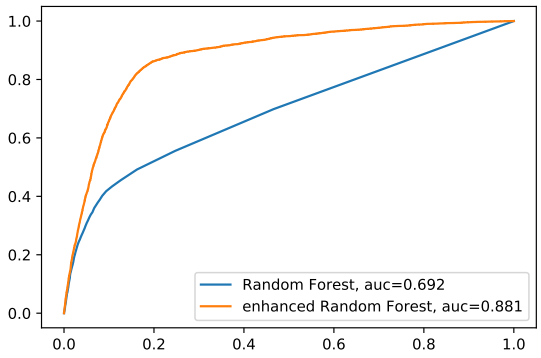

Figure 3: The ROC curve for the Jumper enhancement.

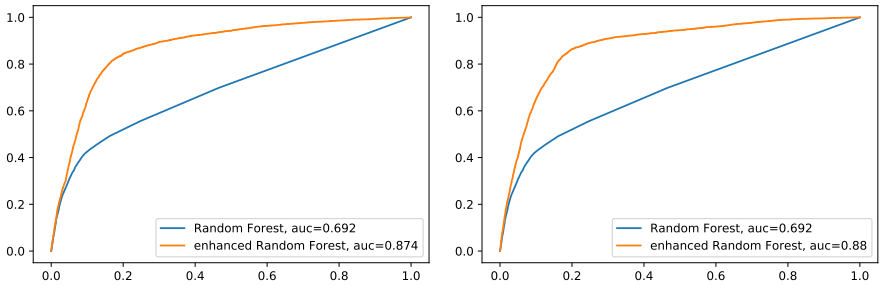

Figure 4: The analogues of Figure 3 for $J := 0.1$ on the left and $J := 0.001$ on the right.

terms of the log-loss function and decimal logarithms, the loss goes down from 5684.1 to 4764.8 (the difference between these two numbers being, predictably, the exponent 919.3 in the final value of the test martingale in Figure 2).

The value $J = 0.01$ is the one that has been used most commonly in the existing papers, but the dependence of our results on the values of parameters is weak: see, e.g., Figure 4, which shows results for jumping rates 0.1 and 0.001. However, using a specific value of J may be risky in that the Jumper test martingale loses capital exponentially quickly if the base prediction algorithm is already ideal (which makes the insurance policy discussed in Section 1 expensive). A safer option is to use the Mean Jumper procedure averaging the Jumper test martingales over a small set of J including $J = 1$ [16, Section 3].

6 Conclusion

The methods of adaptive calibration that we propose in this note need to be validated on other datasets and for other calibrating functions, such as (3), (4), and (5). Notice that calibrating functions may depend not only on the current predicted probability p but also on the current object x. (So that "calibration" may be understood in a very wide sense, as in [4], and include elements of "resolution" [5].)

A natural direction of further research is to extend our methods and results to multiclass classification. Notice that Cox's calibrating functions (4) and (5) immediately extend to the multiclass case.

Acknowledgments

We are grateful to Glenn Shafer for his advice. This research has been partially supported by Stena Line. In our computational experiments we used `scikit-learn` [11].

References

[1] Francis J. Anscombe. Rejection of outliers. *Technometrics*, 2:123–147, 1960.

[2] David R. Cox. Two further applications of a model for binary regression. *Biometrika*, 45:562–565, 1958.

[3] David R. Cox. Partial likelihood. *Biometrika*, 62:269–276, 1975.

[4] A. Philip Dawid. Calibration-based empirical probability (with discussion). *Annals of Statistics*, 13:1251–1285, 1985.

[5] A. Philip Dawid. Probability forecasting. In Samuel Kotz, N. Balakrishnan, Campbell B. Read, Brani Vidakovic, and Norman L. Johnson, editors, *Encyclopedia of Statistical Sciences*, volume 10, pages 6445–6452. Wiley, Hoboken, NJ, second edition, 2006.

[6] I. J. Good. Rational decisions. *Journal of the Royal Statistical Society B*, 14:107–114, 1952.

[7] Mark Herbster and Manfred K. Warmuth. Tracking the best expert. *Machine Learning*, 32:151–178, 1998.

[8] Peter J. Huber and Elvezio M. Ronchetti. *Robust Statistics*. Wiley, Hoboken, NJ, second edition, 2009.

[9] Daniel Kahneman, Paul Slovic, and Amos Tversky, editors. *Judgment under Uncertainty: Heuristics and Biases*. Cambridge University Press, Cambridge, 1982.

[10] Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.

[11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] Glenn Shafer and Vladimir Vovk. *Game-Theoretic Foundations for Probability and Finance*. Wiley, Hoboken, NJ, 2019.

[13] Vladimir Vovk. Derandomizing stochastic prediction strategies. *Machine Learning*, 35:247–282, 1999.

[14] Vladimir Vovk. Competitive on-line statistics. *International Statistical Review*, 69:213–248, 2001.

[15] Vladimir Vovk. Testing randomness online, On-line Compression Modelling project (New Series), `http://alrw.net`, Working Paper 24, June 2019. Journal version: *Statistical Science* (to appear).

[16] Vladimir Vovk. Enhancement of prediction algorithms by betting, On-line Compression Modelling project (New Series), `http://alrw.net`, Working Paper 34, May 2021.