TIME INDEPENDENT SWITCHING BETWEEN TWO SOURCE CODES

Steven de Rooij[1] *and Wouter Koolen*[1,2]

[1]Centrum Wiskunde & Informatica (CWI)
Science Park 123, P.O. Box 94079, 1090 GB Amsterdam, Netherlands. Email: s.de.rooij@cwi.nl
[2] Royal Holloway, University of London, Department of Computer Science
Egham, Surrey, TW20 0EX, United Kingdom. Email: wouter@cs.rhul.ac.uk

ABSTRACT

Switching algorithms for combining sequential codes usually come with bounds, relating their performance to that of the best fixed strategy that switches m times. The overhead is parameterized by the number of outcomes n and the number of switches m; it may be difficult to judge whether this is acceptable. We present an alternative algorithm whose overhead does not depend on n; instead our bound expresses the overhead in terms of the benefit of switching in the first place.

1. INTRODUCTION

In 1998, Paul Volf and Frans Willems published an algorithm that runs two separate source coding algorithms, say, the Context Tree Weighting and LZ77 data compression schemes, in parallel, determining on-line how their predictive models should be combined so that any part of the data is compressed almost as well as if we used the best of the two algorithms for it [1].

Around the same time, many related results on "expert tracking" appeared in the online learning community. A general overview is provided in [2]; publications most relevant to this work are [3, 4, 5, 6].

For all these approaches, performance guarantees of the following form are given. Let $L_\gamma(x^n)$ denote the number of bits a code γ uses to encode outcomes $x^n = x_1, \ldots, x_n$. Fix two codes A and B with length functions L_A and L_B, and let $L_{\boldsymbol{t}}$ denote the number of bits used by the algorithm that switches between A And B at fixed times $\boldsymbol{t} = t_1, \ldots, t_m$. For all tracking algorithms in the listed references, bounds on the maximal performance difference are given. For example, for Volf and Willems' algorithm, one may show that

$$L_{\text{sm}}(x^n) - L_{\boldsymbol{t}}(x^n) \leq nH(m/n) + \tfrac{1}{2}\log n + 3, \qquad (1)$$

where $H(p) = -p\log p - (1-p)\log(1-p)$ is the binary entropy function. This bound can be interpreted as the number of bits required to encode where the switches occur.

Being valid for *any* time sequence $\boldsymbol{t}$ and data sequence x^n, performance guarantees such as (1) are very robust: no assumptions about the data, stochastic or otherwise, are required. However, it can be difficult to relate the overhead expressed by the bound to the benefit of switching in the first place. For example, suppose that A starts out a little better than B — there is an n_0 such that $L_A(x^{n_0}) < L_B(x^{n_0}) - C$; but B is better than A in the long run — $L_B(x^n) \ll L_A(x^n)$ from some n onwards. Then we can improve our code length by C bits by switching once from A to B at the appropriate time, but if we use Volf and Willems' algorithm to do so, this improvement is dwarfed by the maximal overhead of $(3/2)\log n + O(1)$, so the bound does not tell us whether or not using the algorithm would be wise.

In our paper "Switching Investments" [7] we develop an alternative strategy for which, roughly speaking, the maximal impact of each pair of switches on the regret is measured in terms of the benefit of having those two switches in the first place. (The paper is presented in terms of investment strategies, but the setting is equivalent to coding, as explained in Section 4.) We will now look at an example to explain the rough idea, before discussing our algorithm and its regret bound in more detail in the last two sections.

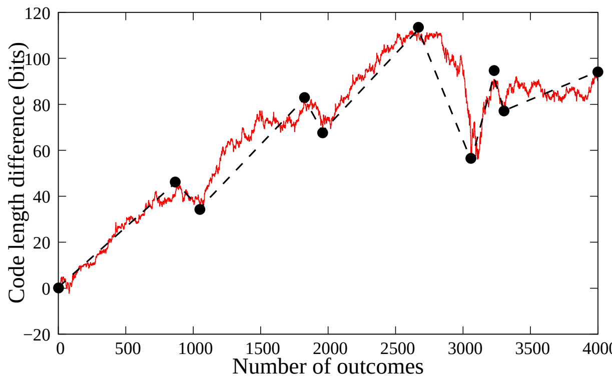

Figure 1. Example code length difference for prefixes

2. EXAMPLE

We have created some fictitious code lengths for coding 4 000 outcomes with A and B. The solid line in Figure 1 shows the difference $\Lambda(i) = L_A(x^i) - L_B(x^i)$ as a function of the number i of processed outcomes. On the whole code B clearly outperforms code A, but there are substantial intervals where the situation is reversed. Ideally, we would switch *every* time Λ changes direction, but that is clearly not feasible. But it looks as if it might be worthwhile to switch from one code to the other at the points indicated by the dashed black line, which can be thought of as a regularisation, or smoothing, of the actual behaviour of the code length difference.

With (1) we can bound the overhead in terms of the numbers on the horizontal axis, which may or may not have anything to do with the sizes of the fluctuations in code length. Our main idea is to parameterize the reference strategy not by its switching times $\boldsymbol{t}$, but instead by a sequence $\boldsymbol{\delta} = \delta_1, \ldots, \delta_m$ specifying the number of *bits* that Λ must increase or decrease between subsequent switches, as measured along the vertical axis. Note that we can identify any sequence of local extrema using such a sequence, and it provides the information we need to switch at the appropriate times as we process the data sequentially. The strategy that corresponds to the dashed line has $\boldsymbol{\delta} = 42, 6, 46, 14, 44, 55, 33, 13, 13$.

3. REGRET BOUND

Our algorithm, with code length function denoted L_{si}, uses a fixed prior density function π (that has to satisfy some mild restrictions). We prove the following bound for all $\boldsymbol{\delta}$, including the one highlighted in Figure 1:

$$L_{\text{si}}(x^n) - L_{\boldsymbol{\delta}}(x^n) \leq \sum_{i=1}^{m} \Big(-\log \pi(\delta_i) \Big) + (m-1)\alpha + \beta, \quad (2)$$

where m is the length of $\boldsymbol{\delta}$ and α and β are small constants that depend on the used prior density function (for example, for $\pi(x) = 3(x+3)^{-2}$ we have $\alpha = 0.042$ bits per switch and $\beta = 6.13$ bits). Note that the second term is constant per switch, and the last two terms are additive constants; the main contribution to the regret comes from the first term, which can now be interpreted as the number of bits required to encode $\boldsymbol{\delta}$ (to an appropriate precision).

Comparing (2) to (1), we find that all dependence on the sample size has disappeared; instead the overhead is expressed in terms of the *sizes of the fluctuations* of the code length difference. If we use a fat-tailed prior, the overhead is only logarithmic in the fluctuation size, which means that if the fluctuations are sufficiently large our switching approach is certain to yield an improvement over A and B.

4. THE ALGORITHM

The algorithm is easiest to explain as the implementation of a Bayesian prediction strategy. By the Kraft inequality, each code length function has an associated probability distribution such that the $-\log$ of the probability of a sequence of outcomes is equal to the code length. Thus, our results can be stated either in terms of coding, or in terms of prediction with logarithmic loss. In the paper we use a third formalism, that of online investment — which turns out to be equivalent yet again.

Now that we think about the set of reference codes a set of random processes $\mathcal{M} = \{P_{\boldsymbol{\delta}} \mid \boldsymbol{\delta} \in \Delta\}$, where $\Delta = [0, \infty)^\infty$ is the set of all infinite sequences of positive differences, we can define a universal model by putting a prior distribution π on the elements of $\mathcal{M}$. The marginal probability of a sequence of outcomes becomes

$$P_{\text{si}}(x^n) = \int_{\mathcal{M}} \pi(\boldsymbol{\delta}) Psi_{\boldsymbol{\delta}}(x^n) \, \mathrm{d}\delta,$$

and as usual we can predict the next outcome by conditioning

$$P_{\text{si}}(X_{n+1} = x \mid x^n) = \frac{P_{\text{si}}(x^n, X_{n+1} = x)}{P_{\text{si}}(x^n)}.$$

(From this probabilistic setting we can get back to coding using e.g. arithmetic coding.) It is not immediately clear that these predictions can be evaluated efficiently, but if the prior π is a product distribution $\pi(\boldsymbol{\delta}) = \prod_i \pi(\delta_i)$, then the height and direction of the last switch is a sufficient statistic and we need to maintain only a linear number of weights in the prediction process, reducing the running time to $O(n)$ per outcome. In fact, if the density π is chosen to be exponential, even more weights can be lumped together and the running time is further reduced to amortized $O(1)$ per outcome.

5. REFERENCES

[1] P.A.J. Volf and F.M.J. Willems, "Switching between two universal source coding algorithms," in *Proceedings of the Data Compression Conference, Snowbird, Utah*, 1998, pp. 491–500.

[2] Nicolò Cesa-Bianchi and Gábor Lugosi, *Prediction, Learning, and Games*, Cambridge University Press, 2006.

[3] Mark Herbster and Manfred K. Warmuth, "Tracking the best expert," in *Proceedings of the 12th Annual Conference on Learning Theory (COLT 1995)*, 1995, pp. 286–294.

[4] Wouter M. Koolen and Steven de Rooij, "Combining expert advice efficiently," in *Proceeding of the 21st Annual Conference on Learning Theory*, 2008, pp. 275–286.

[5] C. Monteleoni and T. Jaakkola, "Online learning of non-stationary sequences," *Advances in Neural Information Processing Systems*, vol. 16, pp. 1093–1100, 2003.

[6] Steven de Rooij and Tim van Erven, "Learning the switching rate by discretising Bernoulli sources onlin e," in *JMLR Workshop and Conference Proceedings*, 2009, vol. 5: AISTATS, Available at http://jmlr.csail.mit.edu/proceedings/papers/v5.

[7] Wouter M. Koolen and Steven de Rooij, "Switching investments," in *Proceedings of the 21st International Conference on Algorithmic Learning Theory (ALT 2010), LNAI 6331*, 2010, pp. 239–254, An extended version will appear in the special issue on Algorithmic Learning Theory in Theoretical Computer Science.