A Fast On-line Causal-State Splitting Reconstruction Algorithm

John A Brown.
nworbnhoj@gmail.com
www.nhoj.info

Abstract

This paper presents an algorithm which implements the method of nonlinear prediction of discrete random sequences proposed by Shalizi [1]. The algorithm is designed to construct and refine a Hidden Markov model under minimal structural assumptions "online" as each new observation becomes available. This algorithm offers improved time complexity compared to the original CSSR algorithm.

1. Introduction

Shalizi, Shalizi and Crutchfield [1, 2 & 3] have presented a method for nonlinear prediction of discrete random sequences under minimal structural assumptions in the form of Hidden Markov Models (HMM). The associated Causal-State Splitting Reconstruction (CSSR) algorithm is described below in close alignment with their notation and prose.

CSSR starts by "assuming" that a process produces an independent, identically-distributed sequence, with one causal state, and adds states when statistical tests show that the current set of states is not sufficient.

The null hypothesis is that the process generating the sequence x is Markovian on the basis of the set of states Σ; that is,

$$P(X_t|_{t-L}^{t-1} = a x_{t-L+1}^{t-1}) = P(X_t|\hat{S} = \hat{\epsilon}(x_{t-L+1}^{t-1})) \quad (1)$$

for all a ∈ finite alphabet A.

Thus, each state is a set of history strings from the sequence which have the same conditional distribution (morph) for the next observation independent of any prefix.

Each history string is initially mapped to the state containing the longest suffix of the history. However, if the null hypothesis is rejected, then CSSR falls back on a restricted alternative hypothesis and maps the history string to any other state for which the null hypothesis is accepted. If there is no such state, then a new state is created and the history string added to it.

The CSSR algorithm has three phases. Phase I initializes Σ to a single state, which contains only the null suffix ∅. The morphs are generated for every suffix of length L_{max}. Phase II iteratively tests the successive versions of the null hypothesis, Eq. 1, beginning with suffix's with L=1 and increasing with each iteration. At the end of II, ϵ is (approximately) next-step sufficient. Phase III makes ϵ recursively calculable, by splitting the states until they have deterministic transitions.

The time complexity of the CSSR algorithm is $O(k^{2Lmax+1}) + O(N)$ where N is the length of the sequence, k is the size of the alphabet A, and L_{max} is the length of the longest history considered.

The CSSR algorithm is designed and implemented to processes an entire sequence of historical observations as a batch. There may be situations where it is advantageous to have the HMM available after each successive observation is made. However, the computational cost of re-running the entire CSSR algorithm after each observation may become prohibitive for larger N, k and L_{max}.

The on-line Causal-State Splitting Reconstruction (oCSSR) algorithm presented here relies on entirely the same basis as CSSR. However, three important changes are made to improve its suitability to on-line processing.

1. The task of recording the probability distributions for the next observation is decoupled from the subsequent steps.
2. CSSR Phase III is avoided entirely thereby removing its significant contribution to the time complexity, which results in a faster algorithm.
3. oCSSR implements the null hypothesis slightly differently to CSSR but in keeping with the underlying theory.

This paper describes string histories with familiar family terminology in an attempt to increase comprehension but perhaps with some loss of precision. Firstly, a history is a sequence of observations with the most recent on the right. Removing the left most

observation from a child history reveals the parent history. Siblings have the same parent. A family of histories share the same ultimate parent which is referred to here as a SuffiX. Similar reference is made to the morphs of these histories as in "the child is homogeneous with the parent" but the distinction between history and morph should always be clear from the context.

Decoupling. CSSR is fundamentally based on the probability distributions for the next observation. oCSSR stores the probability distributions as frequency counts. Decoupling this initial step from the subsequent steps may allow high rate processes to be handled with more limited computational and memory resources. Once the frequency counts have been updated the original sequence data can be discarded resulting in a progressively larger memory saving as N increases. Also, the subsequent steps might be completed only periodically, as required or in batch at the end to reduce computational load.

Avoiding CSSR Phase III. Phase III introduces significant time complexity, $O(k^{2L_{max}}+1)$, as it checks each state for determinism but restarts from the beginning each time a state is split. This continues until no more splits are required. oCSSR avoids the need for Phase III by pre-emptively splitting states to maintain determinism rather than waiting until the end and having to search through the states for non-deterministic transitions.

When the null hypothesis is rejected and a state is split, oCSSR identifies if the determinism of a prior state is disrupted and splits that state immediately. This, in turn, may disrupt the determinism of a yet prior state. So, the step is repeated - a maximum of L_{max} times.

Null hypothesis implementation. oCSSR frequently tests the homogeneity of a child with its immediate parent. In addition, oCSSR calculates the morph of a state as the weighted average of the SuffiX's in that state. By way of contrast, CSSR calculates the morph of a state as the weighted average of all of the the histories in the state.

2 On-line Causal-State Splitting Reconstruction.

This section contains a description of the on-line Causal-State Splitting Reconstruction (oCSSR) algorithm, that estimates the HMM with the properties summarised above and more fully described in [3].

oCSSR can be implemented as two threads. The first thread maintains the morphs for each history and identifies any child which is no longer homogeneous with it's immediate parent. The second thread, takes each identified child suffix, identifies the necessary alterations to the existing states and reconstructs the HMM accordingly.

Thread I. Each morph is maintained as a frequency count for the next observation (rather than a probability distribution). This enables both rapid updates with each new observation and easy use of empirical significance tests. Each morph is attached to a node of a trie representing the history string. The most recent observation is at the root of the morph trie.

When a new observation arrives, the morph trie is traversed from root to leaf, guided by the recent history of length L_{max}. At each node, the morph (count) is incremented according to the observation before being compared to that of its parent node. If there is a significant difference between the two morphs then the node is flagged for processing by Thread II.

Thread II. There are three steps involved: a) Identifying SuffiX's, b) Updating States and c) Updating Transitions.

Identifying SuffiX's. This step implements the null hypothesis but with the additional constraint of maintaining determinism.

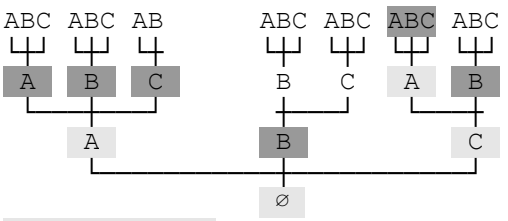
As the morph trie is populated, each node is noted as a member of one of three mutually exclusive sets (see Figure 1).

- *Family nodes* are all of those nodes closest to the leaves of the morph-trie. Each *family node* is collecting information in its morph that may (with additional observations) develop a significant difference from its parent. When the morph-trie is initiated, all nodes are *family nodes* except for the root.
- *SuffiX nodes* exist, one on each branch, between the family nodes and the root of the morph-trie. They represent a history which is the ultimate parent of a family where all children are homogeneous with their immediate parents. Each *SuffiX node* will later be mapped to a state. When the morph-trie is initiated, there is just one *SuffiX node* at the root.
- *Idle nodes* exist only between the root and the SuffiX node on each branch of the morph-trie. *Idle nodes* represent a history that is too short to be mapped deterministically to a state and hence its morph is of no value. When the morph-trie is initiated, there are no *idle nodes.*

A node can only progress from *family* to *SuffiX* to *idle* and not return.

When a *SuffiX node* develops such that it is no longer homogeneous with its state, then it remains a *SuffiX node* but is assigned to a different (possibly new) state.

Figure 1. A morph-trie for histories with L_{max} = 3 & alphabet = {A,B,C}



Idle nodes = {∅, A, C, AC}
SuffiX nodes = {AA, BA, CA, B, AAC, BAC, CAC, BC}
Family nodes = {AAA, BAA, CAA, ABA, BBA, CBA, ACA ..... CBC}

When a *family node* X develops such that it is no longer homogeneous with its parent, then it becomes a *SuffiX node* and the following changes to the morph-trie are necessary to maintain homogeneity:

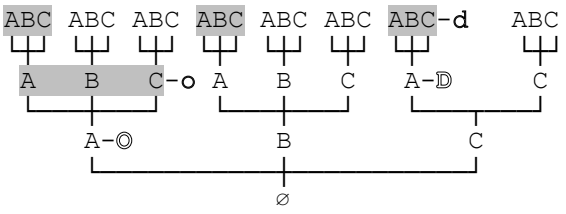
- The parent node, P_x of X must become an *idle node* as P_x no longer represents a homogeneous family.
- The whole sibling group S which contains X, $\{S_1, S_2 \ldots X \ldots S_k\}$, must also become *SuffiX nodes* as they are each now the minimal representation of a homogeneous family.
- This must be repeated for every family back down the morph-trie to the root with the parent of each family becoming an *idle node*.

In addition, the following changes to the morph-trie are necessary to maintain determinism.

- There will be a group of sibling nodes T_{S-1} which will transition to the sibling group $S = \{S_1, S_2 \ldots X \ldots S_k\}$. This group of sibling nodes T_{S-1} must all become *SuffiX nodes*. This must be further propagated to sibling group T_{S-2} which will transition to T_{S-1} etc.
- This must also be similarly propagated for every sibling group of the Parents P_X, $P_{X-1} \ldots P_0$.

To appreciate the SuffiX's required to maintain determinism, consider an origin SuffiX O that transitions to a destination SuffiX D as a result of an observation (see figure 2). SuffiX D can be at most 1 longer than SuffiX O. By definition, all of the children of SuffiX O will also transition to SuffiX D. Now, if a child d of *SuffiX node* D is converted from a *family node* to a *SuffiX node* then a transition from *SuffiX node* O will no longer be deterministic. Hence the corresponding child o of SuffiX O must also become an *Suffix node* to ensure deterministic transitions.

Figure 2. The highlighted sibling nodes in level 2 of the morph trie will transition to one of the three highlighted sibling node groups in level 3 depending on the next observation A, B or C.



If the CAC node becomes a *SuffiX node* then:

- the AC node becomes an idle node,
- the AAC and BAC nodes become SuffiX nodes, and
- the CA node must also become an *Suffix node*.

The same process is then completed for the CA node causing the A=*idle*, AA=*SuffiX*, BA=*Suffix* and C=*Suffix*. Finally, once again for the C node causing ∅=idle, B=SuffiX.

It turns out that identifying all of these changes is relatively straight-forward:

- every sibling of a *SuffiX node* must be a *SuffiX (or idle) node*.
- every suffix of a *SuffiX node* must be an *idle node.*
- every prefix of an *Suffix node* must be a *SuffiX (or idle) node*.

Updating States. This step implements the backup null hypothesis but within the added constraint of maintaining determinism. SuffiX nodes are assigned to states such that composite states - those containing more than one SuffiX – are homogeneous.

To maintain determinism, States cannot contain SuffiX nodes from more than one family group. To see why this is so, consider again figure 2.

When a *family node* develops such that it is no longer homogeneous with its parent, the all of the siblings in group S become *SuffiX nodes*. These *SuffiX nodes* are assigned to two or more homogeneous States.

The group of sibling nodes T, that transition to S, must be assigned States in a pattern compatible with a deterministic transition to S. The states assigned to the nodes in T must have a one-one or many-one relationship with the states in S. A one-many relationship would represent a non-deterministic relationship.

To achieve this deterministic relationship between the States assigned in T and S, a State Template is made of the states assigned in S and applied to those in T.

Updating Transitions. The final step of creating the transitions from one state to the next is trivial. Each state is deterministic so if the disrupted States are tracked in the prior steps then only those transitions need be remapped.

3 Time Complexity

The oCSSR algorithm is designed to be run for each new observation made of a process so for N observations, the oCSSR will run N-1 times.

Thread I takes each new observation, updates the morphs (frequency counts) in the morph-trie of depth L_{max} and tests the homogeneity of each node with its parent. This procedure is therefore $O(L_{max})$ but this will reduce as nodes are set idle.

Thread II identifies SuffiX's, assigns States and constructs transitions but only when a child is found to be significantly different to its parent. This can happen at most once for every parent node on the morph trie.

$$\sum_{x=1}^{L_{max}} k^x < 2\,k^{L_{max}}$$

Identifying SuffiX's and assigning homogeneous States will process at most L_{max} sibling groups. Each sibling group can be split into a maximum of k States requiring k(k-1) steps. The worst case time is $O(k^2L_{max})$.

The state templates will be applied to at most $\frac{1}{2}L_{max}(k-1)$ sibling groups and each application might touch up to k siblings in a single pass. Hence, the worst case time is $O(\frac{1}{2}k^2L_{max})$.

Updating Transitions might process at most k transitions for each of L_{max} sibling groups which each have a maximum of k states. The worst case time is less than $O(k^2L_{max})$

Hence the time complexity of Thread II is $O(2\frac{1}{2}k^2L_{max})$.

Overall. When oCSSR processes an observation, Thread I has time complexity of $O(L_{max})$.

On a maximum of $2k^{Lmax}$ occasions, Thread II will run with time complexity of $O(2\frac{1}{2}k^2L_{max})$

These estimates exclude the complexity of the significance test.

4. oCSSR in batch mode.

oCSSR can easily be modified to operate in batch mode. This would involve:

- Running Thread I but omitting the morph checks until all N observations were recorded in the morph-trie. $O(NL_{max})$
- Checking the nodes of the completed morph-trie for child nodes which are not homogeneous with the parent node. $O(2k^{Lmax})$
- Running Thread II as is over a maximum of k^{Lmax} non-homogeneous child nodes will result in a run time of $O(2\frac{1}{2}L_{max}k^{Lmax+2})$. Some additional gains are available by running the 3 steps in parallel rather than sequentially for each of the k^{Lmax} non-homogeneous child nodes.

Hence, oCSSR operating in batch mode will run within $O(NL_{max} + 2\frac{1}{2}L_{max}k^{Lmax+2})$.

This is a performance improvement on the original CSSR batch algorithm which runs with $O(k^{2Lmax+1}) + O(N)$.

5. Discussion.

The oCSSR algorithm is well suited for use as an exploratory tool. Not only is the HMM constructed as the observations are made but processing and memory resources might easily be directed to areas of interest. For example, if the transitions from one particular state is of interest then L_{max} could be adjusted "on the fly" for the corresponding SuffiX's to take a "deeper look" at its causal nature. Conversely, L_{max} might be reduced for a state with a polarised morph to release the computational and memory resources. Adjusting L_{max} is a powerful tool in this respect as it has a dominant influence on both the computational time complexity and memory requirements.

References

[1] C. R. Shalizi. Causal Architecture, Complexity and Self-Organization in Time Series and Cellular Automata. PhD thesis, University of Wisconsin-Madison, 2001. http://bactra.org/thesis/.

[2] C. R. Shalizi, K. L. Shalizi, and J. P. Crutchfield. An algorithm for pattern discovery in time series. Technical Report 02-10-060, Santa Fe Institute, 2002. arxiv.org/abs/cs.LG/0210025.

[3] C. R. Shalizi, and K. L. Shalizi. Blind Construction of Optimal Non-linear Recursive Predictors for Discrete Sequences. University of Michigan 6 June 2004. arXiv:cs.LG/0406011v1.