Part II Project - Research Summary

Alex Coplan

22 November 2016

This document summarises the research I have carried out in preparation for the implementation phase of the project.

Statistical Models

Conklin & Witten introduce multiple viewpoint systems in their 1995 paper [2], which is the primary paper I used for this part of the research. In this work, the authors make note of the strong links between predictive statistical models and compression. In particular, to quote the introduction of this paper:

> "The conjecture of this paper is that highly *predictive* theories for [music] will also generate original, acceptable works. The predictiveness of a theory (model) can be precisely measured using entropy."

The authors suggest an estimate of entropy as given in equation (1), where e_i is an event, c_i is its context, and the summation is taken over N subsequences in the language of the model, which we denote L.

$$H(L) = -\sum_{i=1}^{N} \log_2 \mathbb{P}(e_i|c_i) \tag{1}$$

The more subsequences used in this estimate, the better the estimate of the entropy. For more on this link see Cleary & Witten's 1984 Paper on the PPM algorithm [1]. As we will see, the PPM algorithm is of considerable importance to multiple viewpoint systems.

Context Models

The fundamental building block in a multiple viewpoint system is the *context model*. Context models are defined to be comprised of:

- A database of sequences over an event space, each having an associated frequency count.
- An inference method which is used to calculate the probability of an event in context.

Notation

e_n^{n+k}	abbreviates the sequence $(e_n, e_{n+1}, \ldots, e_{n+k-1}, e_{n+k})$.
$()$	denotes the empty sequence.
ξ	denotes the *event space*: the set of all representable events.
$c :: e$	is a *snoc* operation: it denotes the list c with e appended to it.
τ	a type
$[\tau]$	the set of all syntactic members of type τ
S^*	the set of all finite sequences drawn from the set S

If $C(x)$ denotes the count of a sequence x in our database, then we perform deductive inference from a context model using equation (2).

$$\mathbb{P}(e|c) = \frac{C(c :: e)}{C(c)} \tag{2}$$

This equation gives us a probability for an event e in the context c, and is the obvious definition when one considers the frequentist view of probability.

The primary data structure used to efficiently implement context models is the *trie* (or prefix tree). If we have a context model for events of type τ, and we let $[\tau]$ denote the syntactic values of type τ, then the underlying trie for the context model maps sequences $e_1^k \in [\tau]^*$ to natural numbers. The root of the trie corresponds to the empty sequence (), and a path from the root to a node uniquely determines a sequence e_1^k, with the value at that node corresponding to how many times the context model has seen the sequence e_1^k in training.

The underlying trie for a context model also has the property that a node's value is given by the sum of its children's values, i.e.

$$\forall e_1^k \in [\tau]^*.\ C(e_1^k) = \sum_i C(e_1^k :: \text{child}(e_1^k)_i)$$

where $\text{child}(e_1^k)$ denotes the set of children of the node corresponding to the sequence e_1^k in the trie.

To induct a context model from a sequence e_1^n, one trains the model by presenting it with the examples:

$$\langle (), e_1 \rangle, \langle (e_1), e_2 \rangle, \langle (e_1, e_2), e_3 \rangle, \ldots, \langle e_1^{n-1}, e_n \rangle \tag{3}$$

The context model can learn from an example using the following procedures (written in a sort of ML/C++ hybrid pseudocode).

```
1   DB::addOrIncrement(e_1^k) =
2     if self.has(e_1^k) then
3       DB[e_1^k].count++
4     else
5       self.store(e_1^k)
6       self[e_1^k].count = 1
7
8   DB::learn(e_1^k as e_1::e_2^k) =
9     self.addOrIncrement(e_1^k)
10    self.learn(e_2^k)
11| DB::learn([])  =
12    self.addOrIncrement(())
```

Namely, to train a context model on the example $\langle e_1^{k-1}, e_k \rangle$, one would call `db.learn(`$e_1^k$`)`. The procedure for training the model on an entire sequence e_1^n is then to generate the examples as per (3), and to call `DB::learn` on each.

Conklin and Witten refer to line 3 of the above (incrementing the count of an existing sequence) as *statistical specialisation* of the model, and lines 5-6 (adding a new sequence) as *structural specialisation* of the model.

Note that we associate a count with (), and this simply acts as a normalising constant. In particular, `db[()].count` gives the total number of examples the model has been trained on, and normalises the counts of sequences of length one to give a probability as per (2).

As given, these procedures would lead to the context model becoming somewhat unwieldy. In particular, for a single example $\langle e_1^{n-1}, e_n \rangle$, the procedure `DB::learn` can store up to $n+1$ sequences in the database. To prevent this, we limit the maximum length of a sequence that can be stored by the model to some number h. We say that such a context model is of order $h-1$. To implement this, when given an example $\langle e_1^{k-1}, e_k \rangle$, instead of calling `db.learn(`$e_1^k$`)`, we call `db.learn(`$e_{k-h+1}^k$`)`, which processes the sequences:

$$(e_{k-h+1}, \ldots, e_n), (e_{k-h+2}, \ldots, e_n), \ldots, ().$$

One question that might arise from equation (2) is what to do when $C(c) = 0$ for some context c. That is, how do we infer the next event when we don't have its context in our database?

The solution proposed by Conklin & Witten is the partial match algorithm (PPM) [1]. This actually solves two problems for us simultaneously: it solves the *zero-count problem* stated above, but also makes our model *non-exclusive*. This is useful for a number of reasons, not least that it allows us to take logs of any conditional probability (and therefore calculate entropy).

A model is said to be non-exclusive if there is a non-zero probability attached to every possible sequence $e_1^n \in \xi^*$. In order for a context model to be non-exclusive, we need to allocate some probability to an event e_i which might be *novel* for its context c. This includes the case when c is the empty sequence (). This means we must also assign a non-zero probability to every event (even those without context).

PPM solves these problems as follows. Instead of allocating probabilities to events as per equation (2), we allow for one count in the context to be used for novel subsequent events. This gives rise to equation (4).

$$\mathbb{P}(e|c) = \frac{C(c :: e)}{C(c) + 1} \tag{4}$$

Thus, the probability left over to be distributed among events that are novel to their context, known as the *escape probability*, is given by:

$$1 - \sum_i \mathbb{P}(e_i|c) = 1 - \sum_i \frac{C(c :: e_i)}{C(c)+1} = 1 - \frac{C(c)}{C(c)+1} = \frac{1}{1+C(c)}$$

We can then distribute this escape probability among all the events which are novel for their context. Formally, the distribution is as follows:

$$\mathbb{P}(e'|e_1^k) = \begin{cases} \frac{C(e_1^k :: e')}{1+C(e_1^k)} & e' \in \text{child}(e_1^k) \\ \frac{1}{1+C(e_1^k)} \cdot \mathbb{P}(e'|e_1^{k-1}) & \text{otherwise} \end{cases}$$

Note that a nice property of this formulation is that the more a context model is trained, the lower the likelihood that it will predict a note that is novel for its context. Conversely, a poorly-trained model is more likely to "guess" a note out of context.

This particular method is actually known as PPM version A. In their original paper [1], Cleary & Witten introduce versions A and B. Since then, there have been several other versions introduced [3]. Initially, I intend to implement version A before considering other versions.

Multiple Viewpoint Systems

We introduce the formalism underlying multiple viewpoint systems, starting with a few definitions.

Definition. A *type* is an abstract property of events. To use a musical example, events might have types such as *scale degree* or the melodic interval of a note (event) with its predecessor.

Definition. A *viewpoint* of type τ consists of:

1. A partial function $\Psi_\tau : \xi^* \rightharpoonup [\tau]$ known as the *projection function*.
2. A *context model* of sequences in $[\tau]^*$.

A collection of viewpoints forms a multiple viewpoint system.

One immediate observation about multiple viewpoint systems comprised only of simple types is that they cannot model the interactions between types (such as interdependence of pitch and rhythm). To model such behaviour, we need to introduce the notion of *product types*.

Definition. A *product type* $\tau_1 \otimes \tau_2 \otimes \ldots \otimes \tau_n$ is itself a type τ where the elements of the product type are elements of the cross product:

$$[\tau] = [\tau_1] \times [\tau_2] \times \ldots \times [\tau_n]$$

and the projection Ψ_τ is defined only when all of the individual projection functions are:

$$\Psi_\tau(e_1^k) = \begin{cases} \perp & \text{if } \Psi_{\tau_i}(e_1^k) \uparrow \text{ for any } i \in \{1, \ldots, n\} \\ \langle \Psi_{\tau_1}(e_1^k), \ldots, \Psi_{\tau_n}(e_1^k) \rangle & \text{otherwise} \end{cases}$$

A *linked viewpoint* is then a viewpoint whose underlying type is a product type. We can now build multiple viewpoint systems out of both linked and primitive viewpoints.

The behaviour of the projection function Ψ_τ is dependent on the individual viewpoint, but the intuition and typical definition is that Ψ_τ takes a sequence e_1^k, finds the last (most recent) event in the sequence to have a property of type τ defined, and projects out this value, some member of $[\tau]$. Of course, it may be the case that no event or combination of events in e_1^k specifies a property of type τ, in which case, $\Psi_\tau(e_1^k) \uparrow$.

Recurrent Neural Networks

I plan to use TensorFlow to implement the RNN. As such, one of the goals of the research phase was to gain familiarity with the library. This has been achieved, in that I have

implemented the MNIST classifier as demonstrated in the introductory tutorial on the TensorFlow website [1].

References

[1] John Cleary and Ian Witten. "Data compression using adaptive coding and partial string matching". In: *IEEE transactions on Communications* 32.4 (1984), pp. 396–402.

[2] Darrell Conklin and Ian H Witten. "Multiple viewpoint systems for music prediction". In: *Journal of New Music Research* 24.1 (1995), pp. 51–73.

[3] Raymond Peter Whorley et al. "The construction and evaluation of statistical models of melody and harmony". PhD thesis. Goldsmiths, University of London, 2013.

[1] `https://www.tensorflow.org/versions/r0.11/tutorials/mnist/beginners/index.html`