An Algorithm for Segmenting Categorical Time Series into Meaningful Episodes

Paul Cohen. Department of Computer Science. University of Massachusetts. cohen@cs.umass.edu
Niall Adams. Department of Mathematics. Imperial College, London. n.adams@ic.ac.uk

Abstract. This paper describes an unsupervised algorithm for segmenting categorical time series. The algorithm first collects statistics about the frequency and boundary entropy of ngrams, then passes a window over the series and has two "expert methods" decide where in the window boundaries should be drawn. The algorithm segments text into words successfully, and has also been tested with a data set of mobile robot activities. We claim that the algorithm finds *meaningful* episodes in categorical time series, because it exploits two statistical characteristics of meaningful episodes.

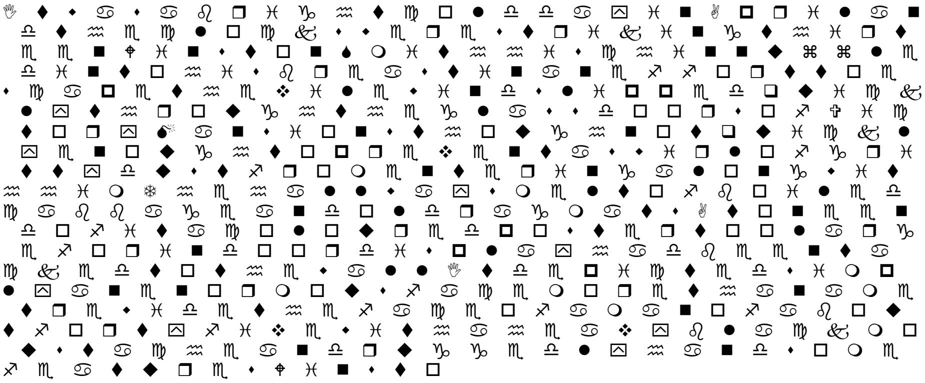
Introduction

Most English speakers will segment the 29 characters in "itwasabrightcolddayinapriland" into nine words. We draw segment boundaries in eight of the 28 possible locations so that the sequences of characters between the boundaries are meaningful. In general, there is an exponential number of ways to draw segment boundaries in ways that produce *meaningless* segments (e.g., "itw" "asab" "rig" …) but we somehow manage to find the "right" segmentation, the one that corresponds to meaningful segments. It seems likely that we do it by *recognizing* words in the sequence: The task is more difficult if the characters constitute words in an unknown language, or if they are transliterations of roman characters into a new font. For example, "♓♦⬥♋⬧♋♌❒♓♑♒♦♍□●♎♎♋⍓♓■♋◻❒♓●♋■♎" is formally (statistically) identical with "itwasabrightcolddayinapriland"; the two sequences have the same ascii representation but are rendered in different fonts. One is easily segmented, the other is not.

This paper asks whether there is a way to find meaningful units in time series other than recognizing them. It proposes two statistical characteristics of meaningful units, and reports experiments with an unsupervised segmentation algorithm based on these characteristics. We offer the conjecture that these characteristics are domain-independent, and we illustrate the point by segmenting text in two languages, and also time series of perceptual data produced by a mobile robot.

The Segmentation Problem

Suppose we remove all the spaces and punctuation from a text, can an algorithm figure out where the word boundaries should go? To illustrate the difficulty of the task, here are the first 500 characters of George Orwell's *1984*, spaces and punctuation removed, translated into a strange font, so you cannot use your knowledge of English to identify words:



To an algorithm that doesn't know English, the actual characters of Orwell's text are no more meaningful than the symbols above are to you[1]. Nevertheless, it can insert boundaries between the characters that *are* meaningful. Here is the result of running the algorithm on the first 500 characters of *1984*. The ✲ symbols are induced boundaries:

```
Itwas ✲ a ✲ bright ✲ cold ✲ day ✲ in ✲ April ✲ andthe ✲ clockswere ✲ st
✲ ri ✲ king ✲ thi ✲ rteen ✲ Winston ✲ Smith ✲ his ✲ chin ✲ nuzzl ✲
edinto ✲ his ✲ brea ✲ st ✲ in ✲ aneffort ✲ to ✲ escape ✲ the ✲ vilewind
✲ slipped ✲ quickly ✲ through ✲ the ✲ glass ✲ door ✲ sof ✲ Victory ✲
Mansions ✲ though ✲ not ✲ quickly ✲ en ✲ ought ✲ oprevent ✲ aswirl ✲
ofgrit ✲ tydust ✲ from ✲ ent ✲ er ✲ inga ✲ long ✲ with ✲ himThe ✲ hall
✲ ways ✲ meltof ✲ boiled ✲ cabbage ✲ and ✲ old ✲ ragmatsA ✲ tone ✲ endof
✲ it ✲ acoloured ✲ poster ✲ too ✲ large ✲ for ✲ indoor ✲ dis ✲ play ✲
hadbeen ✲ tack ✲ ed ✲ tothe ✲ wall ✲ It ✲ depicted ✲ simplya ✲ n ✲
enormous ✲ face ✲ more ✲ than ✲ ametre ✲ widethe ✲ faceof ✲ aman ✲ of ✲
about ✲ fortyfive ✲ witha ✲ heavy ✲ black ✲ moustache ✲ and ✲ rugged ✲
ly ✲ handsome ✲ featur
```

The segmentation clearly is not perfect: Words are run together (Itwas, aneffort) and broken apart (`st ✲ ri ✲ king`). More seriously, some words are split between segments ("to" in `en ✲ ought ✲ oprevent`), although the algorithm loses only a small fraction of words this way.

To define the segmentation problem, we must first distinguish *patterns* from *episodes*. An episode is a pattern that means something in a domain. This definition holds for any notion of pattern and any domain. Thus, "black" and "un" are patterns and also episodes

[1] The sense in which the algorithm knows and does not know English is described in the following section. Obviously, it knows *something*, otherwise its segmentation would be random, but we argue that what it knows is not specific to English or even to human languages, and is, in any case, acquired from the data.

(both are morphemes), whereas “bla” is a pattern but not an episode. In computer vision, induced lines are patterns, and some of them are also episodes by merit of corresponding to edges in the scene. In chess, sixteen pawns arranged neatly in a square in the center of the board doubtless constitute a pattern, but the arrangement is meaningless in the domain, so it is not an episode. The segmentation problem is: *Given a time series of categorical data that contains episodes but no episode boundary markers (e.g., spaces, punctuation) insert boundary markers so that the subsequences between the markers are episodes, i.e., meaningful.*

It is not difficult to write algorithms to find patterns in time series of categorical data, but the majority of these patterns will not be episodes. For example, the following subsequences are the 100 most frequently-occurring patterns in the first 10,000 characters of Orwell’s text, but most are not morphemes, that is, meaningful units:

```
th  in  the  re  an  en  as  ed  to  ou  it  er  of  at  ing  was  or  st
on  ar  and  es  ic  el  al  om  ad  ac  is  wh  le  ow  ld  ly  ere  he  wi
ab  im  ver  be  for  had  ent  itwas  with  ir  win  gh  po  se  id  ch  ot
ton  ap  str  his  ro  li  all  et  fr  andthe  ould  min  il  ay  un  ut
ur  ve  whic  dow  which  si  pl  am  ul  res  that  were  ethe  wins  not
winston  sh  oo  up  ack  ter  ough  from  ce  ag  pos  bl  by  tel  ain
```

Related work

Many methods have been developed for segmenting time series. Of these, many deal with continuous time series, and so are not directly comparable to the problem we are considering here. Some methods for categorical series are based on compression (e.g., [5,8]), but as we just saw, compression finds common, not necessarily meaningful subsequences.   Some methods are trained to find instances of patterns or templates (e.g., [2,4]) but we wanted an unsupervised method. There is some work on segmentation in the natural language and information retrieval literature, for instance, techniques for segmenting Chinese, which has no word boundaries in its orthography. The method in [8], is similar to ours, though it requires training on very large corpora. Magerman and Marcus’ [3] approach to parsing based on mutual information statistics is similar to our notion of boundary entropy (see below). We know of no related research on characteristics of *meaningful* subsequences, that is, statistical markers of boundaries of meaning-carrying subsequences.

Characteristics of Episodes

Although we are far from a theory of episodes – a theory to tell us which subsequences of a series are meaningful – we have observed four empirical characteristics of episodes.

Two of them are not implemented in the current algorithm: one of these relies on the fact that random coincidences are rare, so coincidences often mark episode boundaries [1]; the other exploits the idea that adjacent episodes are often generated by processes that have different underlying probability distributions, so one can infer episode boundaries by looking for points at which the sequences to the left and right of the boundary have different estimated distributions [6,7]. The two characteristics of episodes that we have implemented here are called *boundary entropy* and *frequency*:

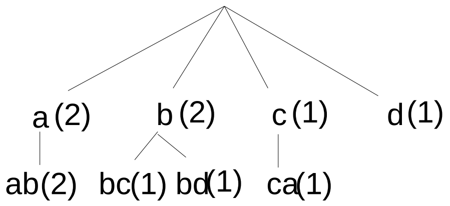
Boundary entropy. Every unique subsequence is characterized by the distribution of subsequences that follow it; for example, the subsequence “en” in this sentence repeats five times and is followed by tokens c and ”. This distribution has an entropy value (0.72, as it happens). In general, every subsequence of length n has a *boundary entropy*, which is the entropy of the distribution of subsequences of length m that follow it. If a subsequence S is an episode, then the boundary entropies of subsequences *of S* will have an interesting profile: They will start relatively high, then sometimes drop, then peak at the last element of S. The reasons for this seem to be, first, that the predictability of elements within an episode increases as the episode extends over time; and, second, that the element that immediately follows an episode is relatively uncertain. Said differently, within episodes, we know roughly what will happen, but at episode boundaries we become uncertain.

Frequency. Episodes, recall, are meaningful sequences, they are patterns in a domain that we call out as special, important, valuable, worth committing to memory, worth naming, etc. One reason to consider a pattern meaningful is that one can use it for something, like prediction. (Predictiveness is another characteristic of episodes nicely summarized by entropy.) Rare patterns are less useful than common ones simply because they arise infrequently, so all human and animal learning places a premium on frequency. In general, episodes are common patterns, but not all common patterns are episodes, as we saw earlier.

The Voting Experts Algorithm

The voting experts algorithm includes experts that attend to boundary entropy and frequency, and is easily extensible to include experts that attend to other characteristics of episodes. The algorithm simply moves a window across a time series and asks, for each location in the window, whether to “cut” the series at that location. Each expert casts a vote. Each location takes n steps to traverse a window of size n, and is seen by the experts in n different contexts, and may accrue up to n votes from each expert. Given the results of voting, it is a simple matter to cut the series at locations with high vote counts. Here are the steps of the algorithm:

Build an ngram tree of depth n+1. Nodes at level i of an ngram tree represent ngrams of length i. The children of a node represent the extensions of the ngram represented by the node. For example, *a b c a b d* produces the following ngram tree of depth 2:



Every ngram of length 2 or less in the sequence *a b c a b d* is represented by a node in this tree. The numbers in parentheses represent the frequencies of the subsequences. For example, the subsequence *ab* occurs twice, and every occurrence of *a* is followed by *b*.

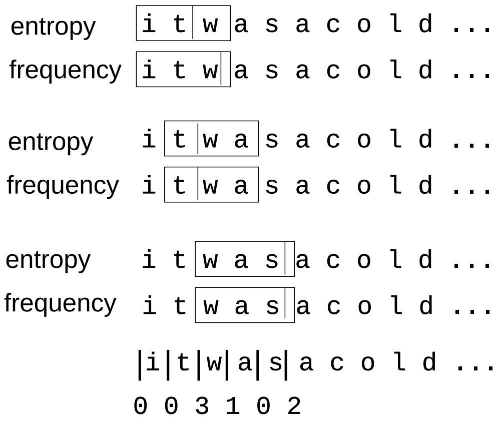
For the first 10,000 characters in Orwell's text, an ngram tree of depth 7 includes 33774 nodes, of which 9109 are leaf nodes. That is, there are over nine thousand unique subsequences of length 7 in this sample of text, although the average frequency of these subsequences is 1.1 – most occur exactly once. The average frequencies of subsequences of length 1 to 7 are 384.4, 23.1, 3.9, 1.8, 1.3, 1.2, and 1.1.

Calculate boundary entropy. The boundary entropy of an ngram is the entropy of the distribution of tokens that can extend the ngram. The entropy of a distribution of a random variable *x* is just $-\sum p(x)\log p(x)$. Boundary entropy is easily calculated from the ngram tree. For example, the node *a* has entropy equal to zero because it has only one child, *ab*, so $\log p(ab) = \log 1.0 = 0$ whereas the entropy of node *b* is 1.0 because it has two equiprobable children, *bc* and *bd*. Clearly, only the first n levels of the ngram tree of depth n+1 can have node entropy scores.

Standardize frequencies and boundary entropies. In most domains, there is a systematic relationship between the length and frequency of patterns; in general, short patterns are more common than long ones (e.g., on average, for subsets of 10,000 characters from Orwell's text, 64 of the 100 most frequent patterns are of length 2; 23 are of length 3, and so on). Our algorithm will compare the frequencies and boundary entropies of ngrams of different lengths, but in all cases we will be comparing how *unusual* these frequencies and entropies are, relative to other ngrams of the *same* length.

To illustrate, consider the words “a” and “an”. In the first 10000 characters of Orwell’s text, “a” occurs 743 times, “an” 124 times, but “a” occurs only a little more frequently than other one-letter ngrams, whereas “an” occurs *much* more often than other two-letter ngrams. In this sense, “a” is ordinary, “an” is unusual. Although “a” is much more common than “an” it is much less unusual relative to other ngrams of the same length. To capture this notion, we standardize the frequencies and boundary entropies of the ngrams. To standardize a value in a sample, subtract the sample mean from the value and divide by the sample standard deviation: $z_i = (x_i - \bar{x})/s_x$. This has the effect of expressing the value x_i as the number z_i of standard deviations s_x it is away from the sample mean $\bar{x}$. Standardized, the frequency of “a” is 1.1, whereas the frequency of “an” is 20.4. In other words, the frequency of “an” is 20.4 standard deviations above the mean frequency for sequences of the same length. We standardize boundary entropies in the same way, and for the same reason.

Score potential segment boundaries. In a sequence of length k there are $k-1$ places to draw boundaries between segments, and, thus, there are 2^{k-1} ways to divide the sequence into segments. Our algorithm is greedy in the sense that it considers just $k-1$, not 2^{k-1}, ways to divide the sequence. It considers each possible boundary in order, starting at the beginning of the sequence. The algorithm passes a window of length n over the sequence, halting at each possible boundary. All of the locations within the window are considered, and each garners zero or one vote from each expert. Because we have two experts, for boundary-entropy and frequency, respectively, each possible boundary may garner up to 2n votes. This is illustrated below. A window of length 3 is passed along the sequence *itwasacold*….



Initially, the window covers *itw*. The entropy and frequency experts each decide where they could best insert a boundary within the window (more on this, below). The entropy expert favors the boundary between *t* and *w*, while the frequency expert favors the boundary between *w* and whatever comes next. Then the window moves one location to the right and the process repeats. This time, both experts decide to place the boundary between *t* and *w*. The window moves again and both experts decide to place the boundary after *s*, the last token in the window. Note that each potential boundary location (e.g., between *t* and *w*) is seen n times for a window of size n, but it is considered in a slightly different context each time the window moves. The first time the experts consider the boundary between *w* and *a*, they are looking at the window *itw*, and the last time, they are looking at *was*.

In this way, each boundary gets up to 2n votes, or n votes from each of two experts, for a window of size n. The *wa* boundary gets one vote, the *tw* boundary, three votes, and the *sa* boundary, two votes.

The experts use slightly different methods to evaluate boundaries and assign votes. Consider the window *itw* from the viewpoint of the boundary entropy expert. Each location in the window bounds an ngram to the left of the location; the ngrams are *i, it*, and *itw*, respectively. Each ngram has a standardized boundary entropy. The boundary entropy expert votes for the location that produces the ngram with the highest standardized boundary entropy. As it happens, for the ngram tree produced from Orwell's text, the standardized boundary entropies for *i, it,* and *itw* are 0.2, 1.39 and –0.02, so the boundary entropy expert opts to put a boundary after the ngram *it*.

The frequency expert places a boundary so as to maximize the sum of the standardized frequencies of the ngrams to the left and the right of the boundary. Consider the window *itw* again. If the boundary is placed after *i*, then (for Orwell's text) the standardized frequencies of *i* and *tw* sum to 1.73; if the boundary is placed after *it*, then the standardized frequencies of *it* and *w* sum to 2.9; finally, if it is placed after *itw*, the algorithm has only the standardized frequency of *itw* to work with; it is 4.0. Thus, the frequency expert opts to put a boundary after *itw*.

Segment the sequence. Each potential boundary in a sequence accrues votes, as described above, and now we must evaluate the boundaries in terms of the votes and decide where to segment the sequence. Our method is a familiar "zero crossing" rule: If a potential boundary has a locally maximum number of votes, split the sequence at that boundary. In the example above, this rule causes the sequence *itwasacold…* to be split after *it* and *was*. We confess to one embellishment on the rule: The number of votes for a boundary must exceed a threshold, and be a local maximum. We found that the algorithm splits too often without this qualification. In the experiments reported below, the threshold was always set to n, the window size. This means that a location must garner half the available votes (for two voting experts) and be a local maximum to qualify for splitting the sequence.

Let us review how the design of the experts and the segmentation rule, to see how they test the characteristics of episodes described earlier. The boundary entropy expert assigns votes to locations where the boundary entropy peaks, locally, implementing the idea that entropy increases at episode boundaries. The frequency expert tries to find a “maximum likelihood tiling” of the sequence, a placement of boundaries that makes the ngrams to the left and right of the boundary as likely as possible. When both experts vote for a boundary, and especially when they vote repeatedly for the same boundary, it is likely to get a locally-maximum number of votes, and the algorithm is apt to split the sequence at that location.

Evaluation

We removed spaces and punctuation from text and assessed how well the voting experts algorithm could induce word boundaries. In these experiments, boundaries stand in six relationships to episodes. To illustrate these relationships, let us adopt the convention that a horizontal line denotes an episode, and vertical lines denote induced boundaries. The relationships are:

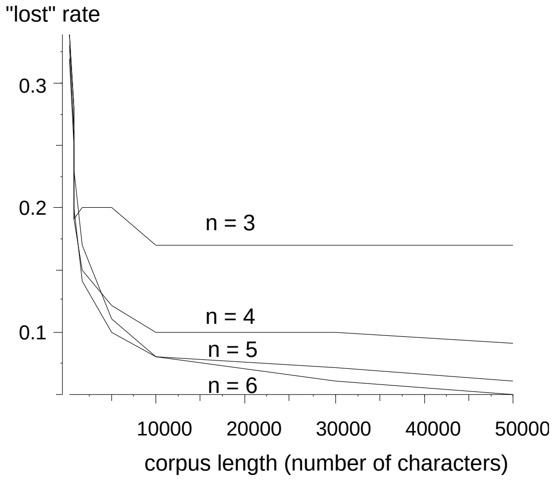
<table>
<tr><td>|______|</td><td>Case 1: The boundaries coincide with the beginning and end of the episode</td></tr>
<tr><td>| ____| or |_____ |</td><td>Case 2: The episode falls entirely within the boundaries and begins or ends at one boundary.</td></tr>
<tr><td>| ________ |</td><td>Case 3: The episode falls entirely within the boundaries but neither the beginning nor the end of the episode correspond to a boundary.</td></tr>
<tr><td>|_____|_____| or |_____|_____|…|_____|</td><td>Case 4: One or more boundaries splits an episode, but the beginning and end of the episode coincide with boundaries.</td></tr>
<tr><td>|_____|_____ | or |_____|_____|…|_____ | or
| _____|_____| or | _____|_____|…|_____|</td><td>Case 5: Like case 4, in that boundaries split an episode, but only one end of the episode coincides with a boundary.</td></tr>
<tr><td>| _____|_____ | or | _____|_____|…|_____ |</td><td>Case 6: The episode is split by one or more boundaries and neither end of the episode coincides with a boundary.</td></tr>
</table>

The cases can be divided into three groups. In cases 1 and 4, boundaries correspond to both ends of the episode; in cases 2 and 5, they correspond to one end of the episode; and in cases 3 and 6, they correspond to neither end. We call these cases *exact*, *dangling*, and *lost* to evoke the idea of episodes located exactly, dangling from a single boundary, or lost in the region between boundaries.

We ran the voting experts algorithm on the first 50,000 characters in Orwell's 1984, spaces and punctuation removed. The window length was 6. The algorithm induced 11210 boundaries, for a mean episode length of 4.46. The mean word length in the text was 4.49. The algorithm induced boundaries at 74.9% of the true word boundaries (the hit rate) missing 25.1% of the word boundaries. 25.4% of the induced boundaries did not correspond to word boundaries (the false positive rate). Exact cases, described above, constitute 55.2% of all cases; that is, 55.2% of the words were bounded at both ends by induced boundaries. Dangling and lost cases constitute 39.5% and 5.3% of all cases, respectively. Said differently, only 5.3% of all words in the text got lost between episode boundaries. These tend to be short words, in fact, 58% of the "lost" words are of length 3 or shorter. In contrast, *all* of the words for which the algorithm found exact boundaries are of length 3 or longer.

It is easy to ensure that all word boundaries are found, and no word is lost: Induce a boundary between each letter. However, this strategy would induce a mean episode length of 1.0, much shorter than the mean word length, and the false-positive rate would approach 100%. In contrast, the voting experts algorithm finds roughly the same number of episodes as there are words in the text, and loses very few words between boundaries.

The effects of the corpus size and the window length are shown in the following graph. The proportion of "lost" words (cases 3 and 6, above) is plotted on the vertical axis, and the corpus length is plotted on the horizontal axis. Each curve in the graph corresponds to a window length, n. The proportion of lost words becomes roughly constant for corpora of length 10,000 and higher.



Said differently, corpora of this length seem to be required for the algorithm to estimate boundary entropies and frequencies accurately. As to window length, recall that a window of length n means each potential boundary is considered n times by each expert, in n different contexts. Clearly, it helps to increase the window size, but the benefit diminishes.

The appropriate control conditions for this experiment were run and yielded the expected results: The algorithm performs very poorly given texts of random words, that is, subsequences of random letters. The algorithm performs marginally less well when it is required to segment text it has not seen. For example, if the first 10,000 of Orwell's text are used to build the ngram tree, and then the algorithm is required to segment the next 10,000 characters, there is a very slight decrement in performance.

As one test of the generality of the algorithm, we ran it on corpora of Roma-ji text and a segment of Franz Kafka's *The Castle* in the original German. Roma-ji is a transliteration of Japanese into roman characters. The corpus was a set of Anime lyrics, comprising 19163 roman characters[2]. For comparison purposes we selected the first 19163 characters of Kafka's text and the same number of characters from Orwell's text. As always, we stripped away spaces and punctuation, and the algorithm induced word boundaries. Here are the results:

	Hit rate	False positive rate	Exact	Dangling	Lost
English	.71	.28	.49	.44	.07
German	.79	.31	.61	.35	.04
Roma-ji	.64	.34	.37	.53	.10

Clearly, the algorithm is not biased to do well on English, in particular, as it actually performs best on Kafka's text, losing only 4% of the words and identifying 61% exactly. The algorithm performs less well with the Roma-ji text; it identifies fewer boundaries accurately (i.e., places 34% of its boundaries within words) and identifies fewer words exactly. The explanation for these results has to do with the lengths of words in the corpora. We know that the algorithm loses disproportionately many short words. Words of length 2 make up 32% of the Roma-ji corpus, 17% of the Orwell corpus, and 10% of the Kafka corpus, so it is not surprising that the algorithm performs worst on the Roma-ji corpus and best on the Kafka corpus.

[2] We are grateful to Ms. Sara Nishi for compiling this corpus.

As a further test of the generality of the algorithm, we gave it a time series of 22535 consecutive states of a mobile robot, generated as the robot engaged repeatedly in a simple "locate an object, move to it, push it" activity. Each of 18 unique states of the robot (e.g., `(near-obstacle moving-forward pushing)`) was assigned a number, and the algorithm was run on the sequence of numbers. The dataset includes 355 episodes. Unlike text, in which words are relatively short, the mean episode length in the robot data set was 63.33. We were interested to learn whether the voting experts algorithm would successfully find the boundaries in this data set even with window sizes far smaller than 63. With a window size of 14, the algorithm induced 956 boundaries, and correctly identified 58% of the 355 episode boundaries, for a false positive rate of nearly 80%. This is probably not acceptable performance for most applications: While the algorithm lost only 20% of the episodes, identifying the boundary of one end or another 80% of the time, the false positive rate means that 4 out of 5 boundaries are not true episode boundaries.

	Hit rate	False positive rate	Exact	Dangling	Lost
Robot dataset	.58	.79	.31	.49	.2

The mean length of induced episodes was 23.54, less than half the size of true episodes, which explains the high false-positive rate. One might expect performance to improve as the window size gets bigger, but the improvement is bounded by the amount of available data: Long ngrams are uncommon, so if we build an ngram tree for ngrams of length, say, 30, then the frequencies (and thus boundary entropies) of nodes lower in the tree become miniscule. Interestingly, the algorithm does not perform much worse with a smaller window size, despite the length of episodes.

Conclusion

The voting experts algorithm segments characters into words with some accuracy, given that it is a greedy, unsupervised algorithm that requires relatively little data. It performs less well identifying long episodes in robot data. In future work, the algorithm will be augmented with other experts besides the two described here, and we are hopeful that these will improve performance on datasets like those generated by our robot. In particular, we have identified two other features of episodes, meaning-carrying subsequences, and we are building experts to detect these features. The idea that meaningful subsequences differ from meaningless ones in some formal characteristics – that syntactic criteria might help us identify semantic units – has practical as well as philosophical implications.

References

1. Cohen, Paul. Fluent learning: Elucidating the structure of episodes. Submitted to IDA2001.

2. M. Garofalakis, R. Rastogi, and K. Shim. Spirit: sequential pattern mining with regular expression constraints. In Proc. of the VLDB Conference, Edinburgh, Scotland, September 1999.

3. Magerman D. and Marcus, M. 1990. Parsing a natural language using mutual information statistics. In Proceedings of AAAI-90, Eighth National Conference on Artificial Intelligence, 984—989

4. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery, 1(3), 1997.

5. Nevill-Manning, C.G. and Witten, I.H. (1997) Identifying Hierarchical Structure in Sequences: A linear-time algorithm, Volume 7, pages 67-82.

6. Tim Oates, Laura Firoiu, Paul Cohen. Using Dynamic Time Warping to Bootstrap HMM-Based Clustering of Time Series. In Sequence Learning: Paradigms, Algorithms and Applications. Ron Sun and C. L. Giles (Eds.) Springer-Verlag: LNAI 1828. 2001

7. Paola Sebastiani, Marco Ramoni, Paul Cohen. Sequence Learning via Bayesian Clustering by Dynamics. In Sequence Learning: Paradigms, Algorithms and Applications. Ron Sun and C. L. Giles (Eds.) Springer-Verlag: LNAI 1828. 2001

8. Teahan, W.J., Y. Wen, R. McNab and I.H. Witten. A compression-based algorithm for Chinese word segmentation. Computational Linguistics, v 26, no 3, September, 2000, p 375-393.

9. Weiss, G. M., and Hirsh, H. 1998. Learning to Predict Rare Events in Categorical Time-Series Data, Proceedings of the 1998 AAAI/ICML Workshop on Time-Series Analysis, Madison, Wisconsin.