2009 Ninth IEEE International Conference on Data Mining

Hierarchical Probabilistic Segmentation Of Discrete Events

Guy Shani
Information Systems Engineeering
Ben-Gurion University
Beer-Sheva, Israel
shanigu@bgu.ac.il

Christopher Meek and Asela Gunawardana
Machine Learning and Applied Statistics
Microsoft Research
Redmond, USA
{meek/aselag}@microsoft.com

***Abstract*—Segmentation, the task of splitting a long sequence of discrete symbols into chunks, can provide important information about the nature of the sequence that is understandable to humans. Algorithms for segmenting mostly belong to the supervised learning family, where a labeled corpus is available to the algorithm in the learning phase. We are interested, however, in the unsupervised scenario, where the algorithm never sees examples of successful segmentation, but still needs to discover meaningful segments.**

In this paper we present an unsupervised learning algorithm for segmenting sequences of symbols or categorical events. Our algorithm, Hierarchical Multigram, hierarchically builds a lexicon of segments and computes a maximum likelihood segmentation given the current lexicon. Thus, our algorithm is most appropriate to hierarchical sequences, where smaller segments are grouped into larger segments. Our probabilistic approach also allows us to suggest conditional entropy as a measurement of the quality of a segmentation in the absence of labeled data.

We compare our algorithm to two previous approaches from the unsupervised segmentation literature, showing it to provide superior segmentation over a number of benchmarks. We also compare our algorithm to previous approaches over a segmentation of the unlabeled interactions of a web service and its client.

I. Introduction

In a number of areas data is naturally expressed as a long sequence of discrete symbols or events. Typically, humans find it difficult to identify patterns or chunks within the long sequence. In such cases it may be beneficial to automatically segment the sequence into smaller chunks, resulting in a shorter sequence over a higher level lexicon [1], [2].

For example, many modern software applications maintain logs of events that occur during the execution. Such logs are useful for understanding the behavior of the deployed application under real conditions and for analyzing traces of failures. Typically these logs are very large and understanding the behavior by looking at the sequence of recorded events becomes very difficult. However, if we can automatically identify that certain subsequences (or chunks) of these sequences originate from certain procedures, replacing these chunks by a single procedure name can make understanding these logs more manageable.

We are interested in two specific types of scenarios. The first scenario is the analysis of web service usage; web services are software applications that provide services for software clients. In this application, the web service are instrumented to maintain a log of the client requests. Analyzing these logs can shed light on the behavior of the client applications, allowing us to better optimize the server responses. The second type of scenario is the analysis of user-driven software applications. Examples of such applications are word processors and spreadsheets. In this scenario, the software is instrumented to capture sequences of user actions. Understanding these sequences of actions can (e.g.) help us to construct better user interfaces. In both these examples we do not have access to the high level process — the human user or the client application — that generated the sequences, and therefore we do not know the high level task and subtasks.

In our Hierarchical Multigram approach to segmentation we consider the segmentation task as having two sub-tasks: (i) lexicon identification in which we identify meaningful segments, or chunks and (ii) sequence segmentation in which the sequence is segmented given a current lexicon. In the Hierarchical Multigram approach developed in this paper, we iteratively interleave these two tasks. Given a lexicon, we segment the sequences. We use the new segmentation to identify a new lexicon by selectively concatinating chunks. We continue this process until we have a hierarcy of chunks. We then choose between alternative segmentations (i.e., the level of the hierarchy) by using the likelihood of the observed data.

We compare the performance of our method to two previous algorithms, Sequitur [1] and Voting Experts [2], over standard benchmarks from the literature — an English and Chinese word identification tasks. In both cases our method outperform both Sequitur and Voting Experts.

Finally, we present some results on lexicon acquisition and segmentation over a real data set of service requests from different clients accessing a Microsoft Exchange server. Even though we do not have labeled data for this domain, we show that our hierarchical lexicon acquisition method generates segments that capture much of the structure present in the logs. We measure this by estimating the Shannon entropy of the service request sequences that is captured by the segmentations generated by our algorithm.

1550-4786/09 $26.00 © 2009 IEEE
DOI 10.1109/ICDM.2009.87

II. Motivating Examples

We begin by an overview of two motivating examples of real world domains where event sequences exist, but are currently analyzed only using statistics over the low level events. We explain for both domains why a better understanding of the high level process is needed.

A. *Microsoft Exchange Server*

The Microsoft Exchange Server is a messaging product, supporting e-mail, calendaring, contacts and tasks, and are accessed by many different software clients. An Exchange server keeps a log of all the requests issued by the clients. These logs contain information such as user id, client application id and requested operation. We hence obtain sequences of operations corresponding to a single user interaction with a specific client application.

These sequences contain only low level operations. A server administrator can use these low level operations to compute statistics, such as the frequency of an operation, or the average number of operations per session. Afterwards, these statistics can be used, for example, to execute simulation interactions with the server to test the server performance in extreme conditions.

However, it was noticed by Exchange administrators that these simulations poorly imitate true user sessions. This is because in a true user session low level operations occur within the context of the high level process of the client application. Therefore, understanding the high level behavior of a client can improve these simulations, and allow Exchange administrators to better optimize server responses.

B. *User Interface Driven Applications*

In many cases user application, such as Microsoft Word or Excel, are instrumented to capture user behavior, such as clicks on toolbars or menus. These low level events can be later collected and sent to the application producer through tools such as Microsoft Customer Experience Improvement Program (CEIP)[1]. For example, when a Microsoft Office product experiences a failure, a user can allow a report to be sent back to Microsoft for analysis.

Using these traces, software engineers can reproduce problems locally, thus making it possible to better understand and fix the problem. However, Microsoft, like other producers of widely used software, receive many such reports. Each report then needs to be transferred to a specific group that handles a specific failure. This classification process is very difficult when the only available data is statistics over the low level operations.

Inferring the high level process that generated a trace can help both the classification process of reports, as well as the reproduction of problems in a controlled environment.

[1]http://www.microsoft.com/products/ceip/EN-US/default.mspx

III. Background and Previous Approaches

Let $\vec{e} = < e_0, .., e_n >$ be a sequence of discrete low level events where each e_i belongs to a finite, known, alphabet Σ. Let $\eta = \{\vec{e}_0, ..., \vec{e}_m\}$ be a set of m such sequences.

Let $\vec{i}$ be a segmentation of $\vec{e}$ — a sequence of indexes $i_0, ..., i_k$ s.t. $i_0 = 0$, $i_k = |\vec{e}| + 1$ and $i_j < i_{j+1}$. We say that s is a segment in the sequence $\vec{e}$ with segmentation $\vec{i}$ if there exists j such that $s = e_{i_j}, \ldots, e_{i_{j+1}-1}$. By extension, we say that s is a segment in η if there exists $\vec{e} \in \eta$ with segmentation $\vec{i}$ such that s is a segment in $\vec{e}$ with segmentation $\vec{i}$. The set of all segments found in η will be called the lexicon, denoted by S.

A lexicon S is hierarchical if for every $s \in S$ containing more than a single event, there exist segments $s_1, .., s_k \in S$ such that $s = s_1 + ... + s_k$, where $+$ is the concatenation operator. $s_1, ..., s_k$ are the sub-segments of s.

A natural application where such sequences arise is in the natural language research community, where chunks of characters are concatenated into words, and word boundaries must be identified. Motivated by this domain, the Sequitur algorithm [1] builds a context free grammar — a set of a rules of the form $X_i \rightarrow X_{i1}, ..., X_{ik}$, where X_{ij} is either a rule or a symbol. An expansion of a rule is the replacement of the left hand side of the rule with the right hand side repeatedly, until all rules have been replaced by symbols. These rules can be thought of as an hierarchical lexicon. Now, we can segment a sequence, by applying these rules such that each segment correspond to an expansion of a single rule. For example, we can decide that segment boundaries are placed after the expansion of rules of a fixed depth d.

A second, probabilistic approach, is the Voting Experts algorithm suggested by Cohen et al [2], where a set of independent criterions (experts) decide on segment boundaries. This algorithm uses a sliding window of a fixed length over the sequence. At each position of the window, each 'expert' votes on the most likely segment boundary within the current window. Then, we traverse the sequence of votes, and introduce segment boundaries where the sum of votes for the next position is smaller than the sum of votes for the current position.

Specifically, Cohen et al. use two experts — one that minimizes the internal entropy of the chunk, and one that maximizes the frequency of the chunk among chunks of the same length. Voting Experts was demonstrated to outperform Sequitur on a word boundary discovery from different languages, and in identifying robot motion sequences.

IV. Multigram

A multigram [3], [4] is a model originating from the language modeling community, designed to estimate the probabilities of sentences, given a lexicon of words. A sentence is modeled as a concatenation of independently drawn words. Here, we will model event sequences as "sentences" which are the concatenation of independently drawn segments ("words"). A multigram Θ defines a distribution over a lexicon of segments $S = \{s_1, s_2, ..., s_m\}$ as $\Theta = \{\theta_i\} : \theta_i = p(s_i)$.

The likelihood of a sequence of segments $\vec{s} = s_{j_1}, \ldots, s_{j_k}$ where each $s_{j_i} \in S$ is defined by $\prod_{i=1..k} p(s_{j_i}) = \prod_{i=1..k} \theta_{j_i}$. The likelihood of all possible segment sequences $\{\vec{s}\}$ consistent with a sequence of events $\vec{e}$ is computed by summing the probabilities of the different segment sequences.

Given a lexicon and a sequence we can learn the segment probabilities using the a dynamic programming procedure derived by specializing the Forward Backward algorithm for HMMs [5] to the case of multigrams. We can use the model obtain a segmentation by using the Viterbi algorithm for HMMs.

V. Lexicon Acquisition

As we will later show in our experiments, the lexicon given to the multigram has a significant impact on the multigram accuracy. A lexicon can be defined by selecting a maximal length n and adding each sequence of length n or less that was observed in the data [4]. However, this approach can result in a huge lexicon, a long training time, and possible loss of accuracy due to local minima.

It is therefore better to filter out some of the observed sequences using some function of the sequence probability such as its mutual information [6], as described below.

A. Hierarchical Lexicon Acquisition

We suggest here an iterative method for hierarchical lexicon acquisition. We begin with a lexicon containing all low level events (Σ), and the trivial single event segmentation.

During each iteration we add concatenations of existing segments, where pairs of segments are chosen for concatenation using their mutual information. The mutual information of the concatenation xy of segments x and y is estimated by:

$$I(x;y) = p(xy) \log \frac{p(xy)}{p(x)p(y)} \qquad (1)$$

where probabilities are estimated over the observed data.

For each concatenation whose mutual information exceeds a predefined threshold, we create a new segment, and add it to the lexicon. Then we train a multigram over the expanded lexicon, produce a new segmentation and a new iteration begins. If no sequence passes the mutual information threshold, the threshold is reduced.

The process is stopped when the threshold is less than ϵ and we can then select the lexicon that maximized the likelihood of the data. As this lexicon is created by always joining together sequences of segments from the lexicon, it is hierarchical.

When creating a new segment to augment the lexicon, we remember the segments that were concatenated to create it. We hence maintain for each segment s an ordered list of segments $L_s = s_1, ..., s_k$ such that $s = s_1 + ... + s_k$, and each s_i is in the lexicon. Each segment can participate in the creation of many longer segments.

Our implementation uses mutual information as the criterion for creating new words, but other measurements such as the conditional probability or the joint probability can also be used. The power of the method comes from collecting term frequencies from already segmented data. Term frequencies are thus more accurate than frequencies that were computed over the raw data.

Consider for example the input string $abcdbcdab$, segmented into $ab|cd|b|cd|ab$. In the raw data, the term bc appears twice, while in the segmented data the term does not appear at all. The term bcd appears twice in the raw data but only once in the segmented data.

Algorithm 1 Hierarchical Multigram Learning

```
input $\eta = \{\vec{e}_0, ..., \vec{e}_m\}$
input $\delta$ — MI threshold
  $i \leftarrow 0$
  $\eta_1 \leftarrow \eta$
  $S_0 = \Sigma_\eta$
  while $\delta > \epsilon$ do
    $i \leftarrow i + 1$
    $S_i \leftarrow S_{i-1}$
    for each consecutive pair of segments $s_1, s_2$ in $\eta_i$ do
      if $MI(s_1 s_2) > \delta$ then
        Add $s_1 s_2$ to $S_i$
      end if
    end for
    Initialize a multigram $M$ using $S_i$
    Train $M$ on $\eta_i$
    $\eta_{i+1} \leftarrow \phi$
    for $j = 0$ to $m$ do
      Add the most likely segmentation of $\vec{e}_j$ given $M$ to $\eta_{i+1}$
    end for
    $\delta \leftarrow \frac{\delta}{2}$
  end while
output $\eta_{i-1}$
```

VI. Evaluating Unsupervised Segmentation

In this paper we focus on unsupervised segmentation of discrete event sequences. The typical method for evaluating the quality of a segmentation is to use a labeled data set, such as a set of sequences that were already segmented by an expert, and see whether the unsupervised algorithm recovers those segments. Alternatively, an expert can observe the resulting segmentation of several algorithms and decide which algorithm produced the best segmentation.

However, in many cases obtaining even a relatively small labeled data set, or manually analyzing the output of an algorithm, can be very expensive. In such cases, we cannot evaluate the accuracy of a segmentation algorithm with respect to a true segmentation. We therefore suggest in the lack of labeled data to evaluate how well does the segmentation algorithm capture the underlying statistical structure of the low level event sequences $\vec{e}$.

A natural candidate for estimating the captured structure is the mutual information $I(\vec{E};\vec{I})$ which is an information theoretic measurement of how much information a segmentation $\vec{i}$ provides about the event sequence $\vec{e}$ [7]. We suggest that a segmentation algorithm whose output contains as much information about the event sequence, and hence the highest mutual information with respect to the event sequence should be preferred among the candidate segmentations. We note that this mutual information between the event sequence and its segmentation should not be confused with the mutual information between adjacent segments that was used in constructing the hierarchical lexicon.

The mutual information can be written as

$$I(\vec{E};\vec{I}) = H(\vec{E}) - H(\vec{E}|\vec{I})$$

where $H(\vec{E})$ is the entropy of the event sequence and $H(\vec{E}|\vec{I})$ is the entropy of the event sequence given the segmentation [7]. Since only the second term depends on the segmentation, choosing the segmentation algorithm with the highest mutual information is equivalent to choosing the one with the lowest conditional entropy. The per-event conditional entropy can be estimated as:

$$\hat{H}(\vec{E}|\vec{I}) = -\frac{1}{|\eta|}\sum_{\vec{e}\in\eta}\frac{\log p(\vec{e}|\vec{i}(e))}{|\vec{e}|} \quad (2)$$

$$p(\vec{e}|\vec{i}) = \prod_{(i_j,i_{j+1})\in\vec{i}}\frac{p(e_{i_j},\ldots,e_{i_{j+1}})}{\sum_{w\in D,|w|=i_{j+1}-i_j}p(w)} \quad (3)$$

where $p(w)$ is estimated probability of a word w in the lexicon D.

In order to estimate the conditional entropy of a segmentation, we divide the set of event sequences into a train and test set. We train a multigram over the train set learn the lexicon probabilities, initializing the lexicon to the words (segments) that were observed in the segmentation. Then, we estimate the conditional entropy of the given segmentation on the test set. It is important that the conditional entropy be estimated on a test set distinct from the lexicon training set in order to measure how well a segmentation captures the underlying structure of the event sequences, rather than measuring how close a segmentation comes to memorizing the training event sequences.

A. Datasets and Algorithms

In this section we provide an empirical comparison of three unsupervised algorithms for segmentation — our hierarchical multigram approach, Sequitur [1], and Voting Experts [2]. Our results demonstrate the superiority of the segmentation that we produce both over previous approaches.

We experiment here with two types of datasets; Much research is dedicated to the identification of word boundaries in Chinese text [8]. While this is an important task, and was previously used to demonstrate segmentation algorithms [2], it is not truly an unsupervised task, as many pre-segmented datasets of this type are available. Still, results for datasets that were segmented by experts can shed much light on the power of various approaches. As such, we use Chinese and English text here mainly to provide information about the properties of the various unsupervised segmentation algorithms. For these datasets we can use the annotated dataset to compute precision-recall curves.

A task that is more interesting for us is the segmentation of event sequences from an Exchange server. This is a real world task, for which there is no existing annotated dataset. Indeed, this is the type of task for which unsupervised algorithms are constructed. In evaluating the segmentation of Exchange events, we use the mutual information criteria defined above.

B. Segmenting Text

We begin by a traditional evaluation of the quality of the segmentation produced by the algorithms on a pre-segmented data set, as done by most researchers. The resulting scores may be somewhat misleading, as a supervised algorithm that is trained on the segmented data set can produce much better results. Still, such an evaluation is useful in providing an understanding of the relative performance of the unsupervised algorithms.

We evaluate the three algorithms over two tasks — the identification of word boundaries in English and in Chinese texts [9], [2], [8]. For the English text, we took the first 10 chapters of "Moby Dick"[2] and transformed the text into unlabeled data by removing all the characters that are not letters (spaces, punctuation marks, etc.) and transformed all letters to lower case. For the Chinese text we used the labeled Academia Sinica corpus[3].

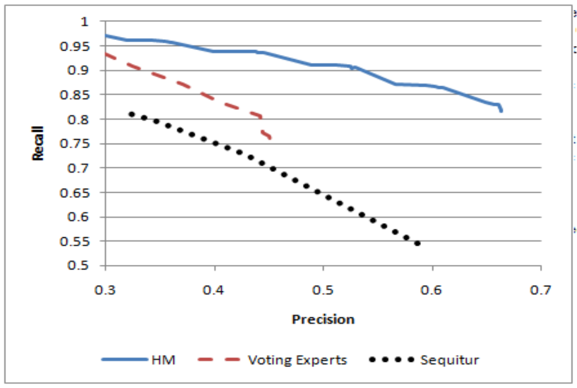
Each of the algorithms has a tunable parameter for the decision on segment boundaries. Voting Expert has a sliding window, whose length affects the segment boundary identification. Sequitur identifies a set of rules, and in order to generate a segmentation from these rules, we place word boundaries after expanding rules up to a finite depth, after which rules are expanded into words. Our lexicon creation procedure is affected by the threshold on the required mutual information for concatenating chunks. We evaluate each algorithm at various settings of its tunable parameter.

In order to compare the accuracy of the algorithms at various settings of their tunable parameters, we employ precision-recall curves, which are a typical measurement for the success of a segmentation algorithm. When identifying segment (or word) boundaries we can either correctly identify a boundary (TP — true positive), fail to identify

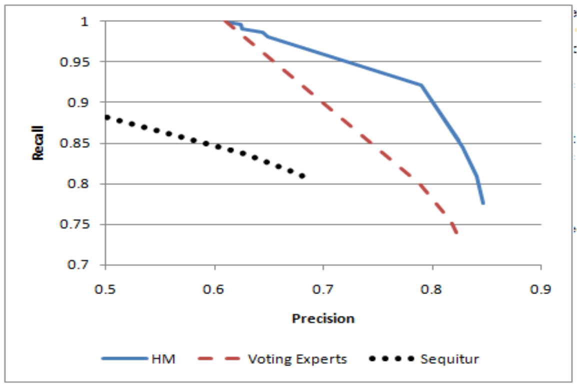
[2]www.physics.rockefeller.edu/~siggia/projects/mobydick_novel
[3]www.sighan.org/bakeoff2003/bakeoff_instr.html

a segment boundary (FN — false negative), or predict a boundary within a segment (FP — false positive). Precision is defined as the portion of true positives among the guessed boundaries — $\frac{\#TP}{\#TP+\#FP}$, while recall is the portion of true positives among all the segment boundaries — $\frac{\#TP}{\#TP+\#FN}$. There is a clear trade-off between precision and recall. At the extreme we can predict no boundary, making no mistake and getting a precision of 1, but identifying no boundaries and getting a recall of 0. At the other extreme we can identify a boundary after each symbol, getting a recall of 1 but a relatively low precision.

Figure 1 shows the precision recall curves resulting from tuning the parameters. As we can see, our method dominates the two other on both data sets. These tasks have some hierarchical structure. For example, we can identify chunks of English words that occur repeatedly, such as "th", "ing", and "tion". Identifying these chunks early in the process can help us to construct the words afterwards.



(a) Moby Dick (English text)



(b) 1998-01-qiefen (Chinese text)

Figure 1. Precision-Recall curves on the two word segmentation problems.

Figure 2 shows some sample output from the hierarchical segmentations yielded by our algorithm on the Moby Dick task. The algorithm does find the correct word boundaries in most cases, and also finds higher level structures such as "more than" and "have been."

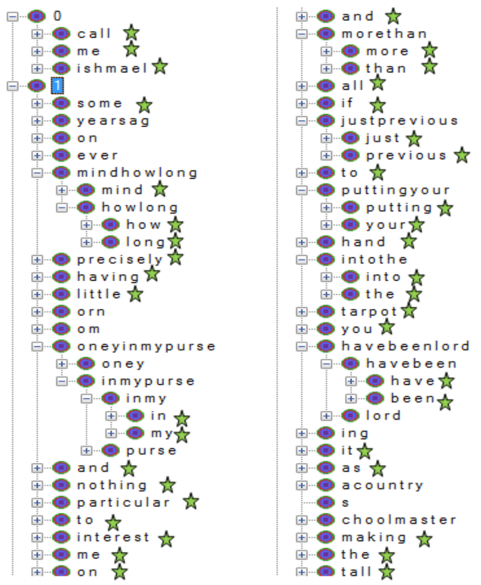


Figure 2. Hierarchical segmentation of the Moby Dick text. Showing the first two sentences (left side) and another random sentence (right side). Correctly identified words are annotated by a star.

Our metric for comparing performance of methods for segmentation over an unlabeled data set is conditional entropy. To validate that this metric correlates with our intuitive notion of segmentation we compute the conditional entropies of the three algorithms for the text segmentation problem. Table I shows the conditional entropies of our approach, and the Sequitur and Voting Experts algorithms on the English and Chinese word segmentation tasks. For each algorithm we picked the segmentation that provided the best F score, defined as $F = \frac{2 \cdot precision \cdot recall}{precision + recall}$. We then trained a multigram using the lexicon derived by the segmentation on a train set (0.8 of the data) and computed the conditional entropy of the given segmentation over the test set (0.2 of the data). In addition, we also evaluated the conditional entropy of the true segmentation using the same procedure.

Algorithm	**English**	**Chinese**
True	1.56	5.73
HM	2.01	6.01
Voting Experts	3.38	6.87
Sequitur	3.00	9.25

Table I
CONDITIONAL ENTROPY OF THE VARIOUS SEGMENTATIONS OF THE THREE ALGORITHMS AND THE TRUE SEGMENTATION ON THE TWO TEXT SEGMENTATION TASKS.

Looking at Table I, we see that the true segmentation gives the most information about the text. This provides evidence that the structure captured by words is highly useful. The true segmentation is followed by our approach, Sequitur, and then Voting Experts for the English text. On the Chinese text Voting Expert outperformed the Sequitur approach. This corresponds well with the precision-recall curves in Figure 1. The ability of Sequitur to achieve higher precision at the expanse of lower recall, resulted in a higher conditional entropy.

Shannon estimated that English had an entropy of about 2.3 bits per letter when only effects spanning 8 or fewer letters are considered, and on the order of a bit per letter when longer range effects spanning up to 100 letters are considered [10]. We estimate the conditional entropy given word segmentations but not taking into account long range effects. Thus, we would expect the conditional entropy to be lower than the entropy taking into account only short range effects, though perhaps not as low as the entropy given longer range effects. Our results using the true segmentations as well as our results using the hierarchical multigram are consistent with this expectation.

C. Segmenting Exchange Sequences

We now move to evaluating the performance of the three algorithms over event sequences gathered from a Microsoft Exchange server. We obtained logs from the interactions of the server with 5 different clients. As we can see in Table II the properties of the clients differ significantly in terms of the functionality that they require from the server (Σ) and the length of a single interaction.

We segmented all the domains using Voting Experts and the hierarchical multigram algorithms, varying the tunable parameters. Note that the Sequitur algorithm failed provide results on these data sets. We report in Table III the conditional entropy of the best segmentation that was achieved by each algorithm.

Client	$\lvert\Sigma\rvert$	**Sessions**	**Avg. Session Length**
AS	22	24,630	19.5
ASOOF	21	17,833	17
OWA	56	9,689	14.7
Airsync	54	27,115	25.9

Table II
PROPERTIES OF THE EXCHANGE CLIENT APPLICATIONS.

Client	**HM**	**Voting Experts**
AS	0.11	0.66
ASOOF	0.069	0.067
OWA	0.42	1.22
Airsync	0.48	1.36

Table III
COMPARING THE CONDITIONAL ENTROPY OF THE HIERARCHICAL MULTIGRAM AND THE VOTING EXPERTS OVER THE VARIOUS EXCHANGE CLIENT APPLICATIONS.

The hierarchical multigram generated segmentations with considerably better conditional entropy in all domains, except for the ASOOF client, which is by far the simplest and most structured domain, as we can see from the very low conditional entropy of both segmentations.

VII. CONCLUSIONS

In this paper we proposed an hierarchical probabilistic segmentation method based on a multigram. Multigram performance is highly dependent on the input lexicon that is provided. We propose a method for iteratively building an hierarchical lexicon. Our method computes the criteria for joining segments based on the current segmentation of the data. As such, the generated term frequencies are more accurate.

We experimented with a text segmentation problem, showing our method to produce superior accuracy, and over real data sets gathered from an Exchange server, showing our method to provide models with lower conditional entropy than a previous algorithms.

Our hierarchical segmentation results in a tree of segments. In the future we would investigate further properties of this tree, such as the best cut through the tree that would produce the best segmentation of the data. We also intend to apply our techniques to the segmentation of user interactions with applications, allowing us to understand the behavior of users when accomplishing complicated tasks.

REFERENCES

[1] C. Nevill-Manning, , and I. Witten, "Identifying hierarchical structure in sequences: A linear-time algorithm," *Journal of Artificial Intelligence Research*, vol. 7, pp. 67–82, 1997.

[2] P. Cohen, N. Adams, and B. Heeringa, "Voting experts: An unsupervised algorithm for segmenting sequences," *Intell. Data Anal.*, vol. 11, no. 6, pp. 607–625, 2007.

[3] S. Deligne and F. Bimbot, "Language modeling by variable length sequences: Theoretical formulation and evaluation of multigrams," in *Proc. ICASSP '95*, 1995, pp. 169–172.

[4] ——, "Inference of variable-length linguistic and acoustic units by multigrams," *Speech Commun.*, vol. 23, no. 3, pp. 223–241, 1997.

[5] L. E. Baum, , T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *Ann. Math. Statist.*, vol. 41, no. 1, pp. 164–171, 1970.

[6] M. Yamamoto and K. Church, "Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus," *Computational Linguistics*, vol. 27, no. 1, pp. 1–30, 2001.

[7] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley and Sons, Inc., 1991.

[8] R. Sproat and T. Emerson, "The first international chinese word segmentation bakeoff," in *The Second SIGHAN Workshop on Chinese Language Processing*, 2003.

[9] H. J. Bussemaker, H. Li, and E. D. Siggia, "Regulatory element detection using a probabilistic segmentation model," in *ISMB*, 2000, pp. 67–74.

[10] C. E. Shannon, "Prediction and entropy of printed english." *The Bell System Technical Journal*, pp. 50–64, January 1951.