Sentence Boundary Detection in Speech for Mobile Computing

Calvin Deutschbein

Abstract— **While sentence boundary detection is AI-hard, with increased prevalence of pervasive sensing in mobile computing, the ability to quickly and easily determine sentence boundaries in speech will become increasing important. While various models have been proposed, most prior models operate on spoken word directly. This paper proposes a new model relying on parts-of-speech in a Hidden Markov Model to reduce computational complexity response time.**

I. INTRODUCTION

This paper explores a possible modification to the OpenLlamaTalk project for speech analysis my mobile computers. OpenLlamaTalk was a project designed to leverage the increased sensing and processing ability of mobile computers to provide always-on speech detection and analysis for mobile users. One aspect of speech analysis by OpenLlamaTalk was a grammar checker that validated diction in spoken sentences. However, sentence boundary detection proved difficult so original versions of OpenLlamaTalk were restricted to running for the duration of a single sentence.

In its current form, OpenLlamaTalk receives speech data as output from Google's Web Speech API and then passes it through Stanford's Log-Linear Part-of-Speech tagger API to determine parts of speech for analysis. Google's Web Speech API does not provide any sentence boundary delimiters so either a different speech-to-text framework was needed or sentence boundary detection would have to be performed on text.

The aim of this paper is to establish the accuracy of sentence boundary detection performed after part-of-speech tagging. As sentence boundary detection may be considered an AI-hard Natural Language Processing problem, simplifying the problem space by analyzing parts-of-speech rather than words directly offered reduced complexity that could potential also lead to increased accuracy. It is worth noting that in experiments with OpenLlamaTalk on Google Glass, existing computation already was producing enough heat to make Glass uncomfortable for the user after just a few sentences of speech, making reducing computation an important goal in ongoing development.

A Hidden Markov Model would be used to detect the transitions from subject subclause of a sentence to predicate subclause of a sentence. In standard English, either a conjunction or a sentence boundary will lie at the transition from predicate to subject. Previous work has noted the importance of detecting conjunctions as the likely beginnings of new sentences in transcripts[1]. As subjects are based around nouns and predicates around verbs, a two state Hidden Markov Model may be able to find parts-of-speech occurring with different probabilities in these two sentence subclauses and transition between states on sentence boundaries and as sentence changes from subject to predicate.

While Conditional Random Fields have demonstrated higher degrees of predictive in power in sentence boundary detection with regard to processing over words[2], the reduced complexity of the problem space as a result of using parts-of-speech instead of words may once again favor Hidden Markov Models, so they are explored here.

II. RELATED WORK

The two current state-of-the-art approaches to sentence boundary detection in speech depending on input type. If audio is monitored directly, sentence boundary detection in vocal pauses was found to be highly effective by Y. Gotoh et al. In the case that transcripts must be analyzed, the application of Conditional Random Fields have been shown to be highly effective in analysis by Y. Liu et al.

Working directly on parts-of-speech, especially in an environment in which computational complexity is even more important, does not appear to have been approached by any other researchers.

III. EXPERIMENTAL DATA

The sentence boundary detector was intended for use with casual everyday speech so for the purpose of this work the example testing set was selected from a web series called "Thorin's Thoughts" by Duncan "Thorin" Shields, an esports journalist and historian. Transcriptions were provided by Reddit user /u/transcribesstuff and their validity was established by

Reddit's own upvote/downvote system. One arbitrarily selected episode, Thorin's Thoughts - Coast and the Outrage Inconsistency (LoL) was used as the testing set. The testing set was 3719 words in length.

While Thorin's work is not fully representative of modern casual speech, it does have a diverse viewing audience across multiple cultures and does have a sufficiently large fan base to ensure availability of quality transcriptions to serve as a gold standard for testing. Moreover, as esports has a wide array of unique internal language it provides testing of validity of the model to be used in different cultural settings.

For the purposes of this paper, human transcriptions were used instead of Google's Web Speech API's output so that the validity of boundary detection could be tested without regard for the accuracy of the text-to-speech infrastructure. It was observed that the automated transcription had sufficient errors that the parts-of-speech tagging diverged significantly even prior to the application of the Hidden Markov Model.

As this was specifically intended to find sentence boundaries in speech, the idea of synthetic data was not particularly meaningful to this problem, so this paper will focus on the application of the method to real data.

IV. HIDDEN MARKOV MODELS

The Hidden Markov Model or HMM is a statistical model in machine learning in which the output of a system being observed is considered to a generated state-machine model $M = (S, A, T, E)$ where S is the array of the states of the model, A is the array containing the alphabet in which the output of the model is encoded, T is a $|S|$-by-$|S|$ matrix where $T(i, j)$ gives the probability that when the model is in state $S(i)$ that it transitions to state $S(j)$ while emitting output, and E is a $|S|$-by$|A|$ matrix where $T(i, j)$ gives the probability that the model in state $S(i)$ emits element $A(j)$ while transitioning to the next state.

A. EXAMPLE MODEL

For an explanatory example, consider generating a series of integers using a fair coin and a fair (six-sided) die.

First, the coin is flipped twice and the number of heads plus one is recorded.

1) If there are two heads, the die is used for the next round.
2) If there are fewer than two heads, the coin is used for the next round.

When the die is rolled, it's value is recorded.

1) If the value is greater than 2, the die is used for the next round.
2) If the value is less than or equal to 2, the coin is used for the next round.

This system can be modeled as $M = ([C, D], [1, 2, 3, 4, 5, 6], T, E)$ where C denotes coin, D denotes die and T and E are given by the following.

$$T = \begin{bmatrix} \frac{3}{4} & \frac{1}{4} \\ \\ \frac{1}{3} & \frac{2}{3} \end{bmatrix}$$

$$E = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \end{bmatrix}$$

For example, when rolling the die, this means there is a probability of $\frac{1}{6}$ that a two is recorded. When flipping a coin, there is a probability of $\frac{1}{4}$ that a die is rolled in the next round.

By observing the output of a model for some time, a Hidden Markov Model can be trained to determine expected probabilities of transmissions and emissions.

B. VITERBI PATH

The Viterbi path calculates, given an instance of output of a model, the most likely set of states the model passed through when generating that output.

V. METHODOLOGY

Processing flow begins with the transcription.

> The brand new League of Legends organization, NRG, owned by Sacramento Kings owner, apparently, signing what was formerly Coast's organization, buying it off them, but not signing the players, ...

Fig. 1. The provided transcript from Reddit user /u/transcribesstuff

The transcript was then part-of-speech tagged. As in OpenLlamaTalk, the tagger provided by Stanford's Computational Linguistics group was used. This provides highly specific part of speech tags optimized for machine learning analysis.

After tagging, the original words are discarded and only the tags are retained in order to provide input to the Hidden Markov Model. This was done by using a regular expression to find all text between a whitespace character and an underscore inclusive and replace it with a (in this case Unicode) space.

The_DT brand_NN new_JJ League_NN of_IN Legends_NNPS organization_NN ,_, NRG_NNP ,_, owned_VBN by_IN Sacramento_NNP Kings_NNPS owner_NN ,_, apparently_RB ,_, signing_VBG what_WP was_VBD formerly_RB Coast_NNP 's_POS organization_NN ,_, buying_VBG it_PRP off_IN them_PRP ,_, but_CC not_RB signing_VBG the_DT players_NNS ,_, ...

Fig. 2. Application of Stanford's Log-Linear Part-of-Speech tagger API

\s(.*?)_ → U+0020

Fig. 3. Regular expression used to parse out words from tagged transcript

The complexity of this operation is $O(l)$ where l is the length of the input in characters. For this testing set, the length was 34848.

Next, all punctuation is stripped from the testing data as it is not present in speech.

A separate reference copy with sentence delimiters is preserved for later comparison. This was generated by first replacing all instances of the period character with the term 'break' and then running the same regular expression over the data.

For the purposes of this experiment, the Hidden Markov Model would be run in MATLAB's Hidden Markov Model Toolbox, so the parts of speech needed to be tokenized by a lisp function into integer tags for easier mathematical analysis.

```
(define pos '(DT NN JJ NN ... ))

(define unique
  (remove-duplicates pos))

(define pos_tags (cons 'break unique))
```

DT NN JJ NN IN NNPS NN , NNP , VBN IN NNP NNPS NN , RB , VBG WP VBD RB NNP POS NN , VBG PRP , IN PRP CC RB VBG DT NNS , ...

Fig. 4. Application of Stanford's Log-Linear Part-of-Speech tagger API

\s\p\s → U+0020

Fig. 5. Regular expression used to parse out words from tagged transcript

DT NN JJ NN IN NNPS NN NNP VBN IN NNP NNPS NN RB VBG WP VBD RB NNP POS NN VBG PRP IN PRP CC RB VBG DT NNS ...

Fig. 6. Punctuation removal

```
(define (get-index x xs i)
  (cond
    [(eq? (first xs) x) i]
    [else (get-index x
     (rest xs) (+ 1 i))]))

(define (pos-to-index p)
  (get-index
   p pos_tags 0))

(map pos-to-index pos)
```

The variable 'pos' contains the tags loaded in after punctuation removal. This script has time complexity $O(n * m)$ where n is the length of the input in tokens and m is the number of unique tokens. This analysis does assume that tokens are of short length as otherwise $(eq?...)$ may begin taking considerably long as a string compare. It could be optimized to run in $O(n*log(m))$ by using a sorting algorithm to index but that was not considered necessary for this data size. For the testing set, $n = 3870$ which is notable as it is greater than the number of words as the tagger would sometimes split words into parts, such as in the case of the following example:

Coast's → Coast_NNP 's_POS → NNP POS

Fig. 7. Some words generate more than one part-of-speech tag

For the testing set, $m = 33$. Across testing sets it seems somewhat regular that length in words would be roughly two orders of magnitude larger than the number of unique parts-of-speech and one order of magnitude smaller than the length in characters. However, for significantly different input lengths the part-of-speech ratio is likely drastically different as it can reasonably be expected to following a pseudo-sigmoid as a the first words of a set are all significantly more likely to be a unique part-of-speech and there a finite number of parts-of-speech so saturation will occur asymptotically.

The final output from the script was lisp list formed of a series of integers separated by spaces.

This series of integers is then loaded into MATLAB as the sequence array for use by the MATLAB HMM

1 2 3 2 4 5 2 6 7 4 6 5 2 8 9 10 11 8 6 12 2 9
13 4 13 14 8 9 1 15 ...

Fig. 8. Tags are encoded as natural integers for portability

Toolbox. First, the MATLAB HMM is trained using the following function from the HMM Toolbox:

```
[T2, E2] = hmmtrain(seq, T1, E1)
```

For this function, 'seq' is the loaded sequence array, 'T2' and 'E2' are the trained transmisson and emission matrices and 'T1' and 'E1' are guesses from transmission and emission matrices generated by the following function:

```
function [T1, E1] = newstart(parts)
T1 = rand(2,2);
T1(1,1) = T1(1,1) / 2 + .5;
T1(1,2) = 1 - T1(1,1);
T1(2,2) = T1(2,2) / 2 + .5;
T1(2,1) = 1 - T1(2,2);
E1 = rand(2,parts);
div = sum(E1,2);
E1 = bsxfun (@rdivide, E1, div);
```

For this function, 'parts' is the number of unique parts-of-speech in the sequence. Note that transmission probabilities are always generated such that remaining in the same state is preferred. For the purposes of illustration, further figures will be populated by data generated from replacing random number number generation with uniform number generation by replacing the call to 'rand' with a call to 'ones' but changing no other aspects of the processing flow.

trans = [1,0;0,1], emis = [0.0303,...;0.0303,...]

Fig. 9. Generated transmission and emission probabilities from newstart

trans = [1,0;0,1], emis = [0.1078,...;0,...]

Fig. 10. Trained transmission and emission probabilities from hmmtrain

After training the Hidden Markov Model to generate 'T2' and 'E2' the array V encoding the Viterbi path through the sequence array is computed using the following function from the HMM Toolbox:

```
V = hmmviterbi(seq, T2, E2);
```

For this function, inputs are as defined previously. This produces as output the states the Viterbi path through the model passes through.

V = [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,...]

Fig. 11. Viterbi states from hmmviterbi

Applying the idea that sentence breaks occur at transitions between states, subtracting the state array from itself with an index change gives an array C in which all non-zero elements are state-changes and the sign of the element gives the direction of the state change for the Viterbi path:

```
C = C(1:size(S,2)-1)
C = C - temp1(2:size(S,2))
```

C = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,...]

Fig. 12. Array encoding changes in state by Viterbi path

The indices at which all transitions or one direction of transition occur can each be determined with a single MATLAB command:

```
all_changes = find(C)
pos_changes = find(C > 0)
neg_changes = find(C < 0)
```

In the case of the example, all of these return the empty array as there are no state transitions in the example.

When newstart is randomized, it becomes desirable to run more than one trial. The computations presented here can be combined into an iterative function that runs multiple trials and combines the results.

```
function [all,pos,neg] = findc(seq)
iter = 20;
len = size(seq,2);
cs = zeros(iter,len-1);
parts = max(seq);
for i=1:iter
    [T,E] = newstart(parts);
    [T,E] = hmmtrain(seq,T,E);
    tmp = hmmviterbi(seq,T,E);
    cs(i,:) = tmp(1:len-1);
    cs(i,:) = cs(i,:)-tmp(2:len);
end
% Try for consistent signs
for i=1:iter
    sign = cs(i,find(cs(i,:),1));
    cs(i,:) =  sign * cs(i,:);
```

```
end
% Find agreement of 95+%
tol = .95;
c = fix(sum(cs)/(tol*iter));
all = find(c);
pos = find(c > 0);
neg = find(c < 0);
```

This is similar to the example with the added consideration of attempting to enforce consistency in state names. As state names are arbitrary in MATLAB, setting the sign of the first phase transition to be positive in all cases enforces consistency in naming convention across all trials if the trials converge to the same path and only changes an arbitrary name in the case that the trials fail to converge.

In general, as the subject is expected to preceed the predicate, 'neg' will be expected to have the highest accuracy. On this testing set, there were usually 412 each of positive and negative transitions for a total of 824.

```
neg = [9,20,28,37,59,70,77,84,99,104,...]
```

Fig. 13. Locations of transitions from the second state back to the first state taken by Viterbi path in 95 or higher percentage of trials

VI. RESULTS

The standard for comparison was determined by an additional lisp function. The second reference copy of the parts-of-speech tags preserved earlier on in the process is loaded into lisp as 'pos_broken' and tagged similarly to the experimental data.

```
(define pos_broken '(DT NN ... ))

(define index_broken
  (map pos-to-index pos_broken))

;; 'break index is 0
(define (get-bs ps i)
  (cond
    [(empty? ps) empty]
    [(zero? (first ps))
     (cons i (get-bs (rest ps) i))]
    [else
     (get-bs (rest ps) (+ i 1))]))

(get-bs index_broken 1)
```

The variable 'pos_broken contains the tags loaded in from the reference copy containing sentence breaks. This gives the sentence index of the sentence breaks determined by /u/transcribesstuff. Note that the index is not incremented on breaks because these breaks were not present in the experimental data and therefore have no associated index relative to the MATLAB data.

```
'(63 131 164 177 228 278 318 388 ...)
```

Fig. 14. Locations of sentence breaks from the transcript

Immediately of note is that the transcription contains fewer sentence breaks than the number of state transitions in either direction by the Viterbi path. This is unsurprising as state transitions would likely still occur around conjunctions, but the degree to which they differ is greater than expected. The transcript only contained 122 sentence breaks, while the Hidden Markov Model experience 412 transitions each in two directions. In this case, a truly random assignment would be expected to find (almost exactly) 13 breaks per direction.

TABLE I
ALL TRANSITIONS

	True	False
Positive	17	807
Negative	2942	105

TABLE II
POSITIVE TRANSITIONS

	True	False
Positive	10	402
Negative	3347	112

TABLE III
NEGATIVE TRANSITIONS

	True	False
Positive	7	405
Negative	3344	115

All told, the Hidden Markov Model finds fewer sentence breaks than would be expected from random guessing. However, more closely examining the data suggests another possibility: phase transitions could sometimes be clustered around a sentence break without detecting directly. For example, in the following phrase there were four distinct phase transitions all

immediately adjacent to either a sentence break themselves or to another phase transition adjacent to a sentence break.

> So first of all, let's just break down the details of (+) this (- & .) It (+) all (-) began last year in 2014,...

Fig. 15. Locations of phase transitions denoted by sign. When colliding with a sentence break both the period and sign are denoted.

To see how close phase transitions were to actual breaks, distances of phase transitions to the nearest actual sentence break and compared it to the distance of every candidate sentence break.

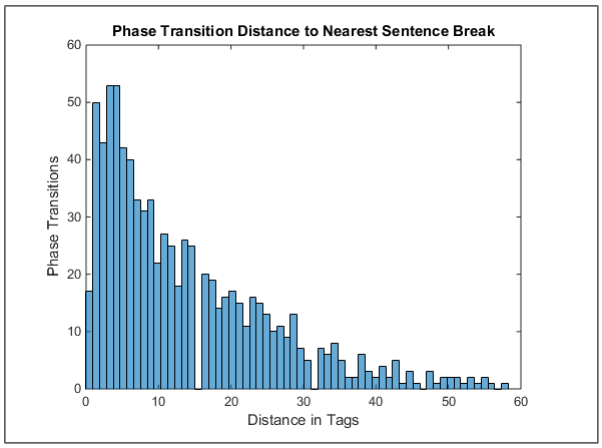

Fig. 16. Distances from phase transitions to nearest actual sentence break measured by number of part-of-speech tags

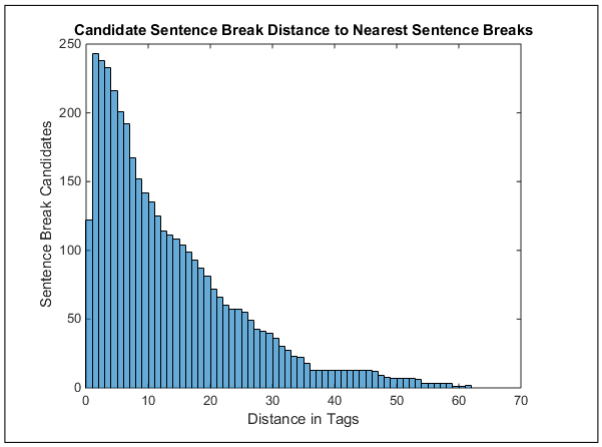

Fig. 17. Distances from every space between part-of-speech tag to nearest actual sentence break measured by number of part-of-speech tags

Overall, it seems as if the Hidden Markov Model over parts-of-speech has very little to no predictive power in determining sentence breaks by any reasonable measure.

TABLE IV
STATISTICS ON PHASE TRANSITIONS AND CANDIDATES

	Q1	Mean	Q3	Max	Std Dev
Transitions	4	10	19	58	11.86
Candidates	4	10	20	62	11.53

VII. CONCLUSIONS

Unfortunately, Hidden Markov Models over parts-of-speech tags appears to have no reasonable ability to determine sentence breaks. While iteration over different numbers of states could be a possibility, it would perhaps be more advisable to attempt to apply Conditional Random Fields instead given their propensity for accuracy when operating on words directly. Also, a different tagging system could be used that is more suggestive of overall sentence structure.

ACKNOWLEDGMENT

The author would like to thank Professor John Goldsmith at the University of Chicago for igniting a passion for Machine Learning, Computational Linguistics, and Natural Language Processing, Laura Macaddino for co-founding the OpenLlamaTalk project, Bodi Li for assisting in in its development, Professor Andrew Chien at the University of Chicago for providing the impetus to begin developing and helping provide access to Google Glass, Google for making their Glass Developer Kit available to the OpenLlamaTalk team, Professor Vladimir Jojic at the University of North Carolina at Chapel Hill for further inspiration and training in Machine Learning and Lee Barnett for helping with the author's first LaTeX document of this scale.

REFERENCES

[1] Y. Gotoh, and S. Renals, Young, Sentence Boundary Detection in Broadcast Speech Transcripts, in Proceedings of the International Speech Communication Association (ISCA) Workshop: Automatic Speech Recognition: Challenges for the New Millennium (ASR-2000), pages -, Paris, September 2000

[2] Y. Liu, A. Stolcke, E. Shriberg, M. Harper, Using Conditional Random Fields For Sentence Boundary Detection In Speech, in Proceedings of the 43rd Annual Meeting of the ACL, pages 451458, Ann Arbor, June 2005.