

Detection of List-Type Sentences

Taniya Mishra, Esther Klabbers and Jan P. H. van Santen

Center for Spoken Language Understanding
OGI School of Science & Engineering at OHSU
20000 NW Walker Road, Beaverton, OR 97006, USA

mishra@cse.ogi.edu

Abstract

In this paper, we explore a *text type* based scheme of text analysis, through the specific problem of detecting the *list* text type. This is important because TTS systems that can generate the very distinct F_0 contour of lists sound more natural. The presented list detection algorithm uses part-of-speech tags as input, and detects lists by computing the alignment costs of clauses in a sentence. The algorithm detects lists with 80% accuracy.

1. Introduction

The ubiquity of personal computers and the increased use of over-the-phone transactions in the last decade have made the use of text-to-speech (TTS) systems quite popular—almost commonplace. However, the increase in popularity comes accompanied by an increased clamor to make TTS systems more *natural-sounding*. This goal of *naturalness* depends on the sophistication and the robustness of the several underlying mechanisms: text analysis, linguistic and prosodic representation, and signal processing.

This paper focuses on text analysis. State-of-the-art text analysis remains quite error-prone in most TTS systems. The reason for this is that given a unit of text, there are usually many ways of parsing it, depending on its context. For example, consider the problem of distinguishing zip code from street address; the text unit, “19000” is parsed as “one-nine-zero-zero-zero” in the context of a zip code, but as “nineteen thousand” in the context of a street address. Another common *text type* that causes such confusion is abbreviations - particularly noticeable in difficult domains such as classified ads for real estate or matrimonial services, where novel abbreviations with unclear expansions abound.

In our project, “Prosody Generation for Child Oriented Speech Synthesis” we intend to analyze text using the concept of *text type*. A text type is a distinct text region, set apart either by its stylistic variation (e.g. tables, or quotes) or by its prosodic variation (e.g. lists versus parenthetical remarks). This text type based analysis will involve partitioning the text into text regions that have to be made as distinct in the output speech as they are in the input text. This idea was proposed by Sproat and van Santen [5] and has been previously used by Sproat [4] in pre-processing email for audio rendering with good results.

The specific task of text analysis that we are going to discuss in this paper is the task of detecting the *list* text type. Lists are enumerations of similar things (names, actions, attributes, etc.) We chose lists as a candidate for type-based text analysis for three principal reasons: First, they are found quite often in children’s stories, the domain on which we focus in our project.

The second reason which makes lists a particularly motivating candidate is that lists have a specific semantic form, and are uttered with a distinct F_0 contour, which we would like to replicate in our TTS system. For example, consider the sentence: “There were two red lobsters, and a ham, a fish, a pudding and some pears and oranges.” It has a distinct lexical form (that of a list) and is uttered with a distinct F_0 pattern: high for “two red lobsters”, lowered for “ham”, high again for “a fish”, lowered again for “a pudding”, raised slightly for “some pears” and stays at approximately the same height for “and oranges”. The constrained nature of the F_0 contour of lists has previously been evinced by Lieberman and Pierrehumbert [2]. However, the list intonation described in their paper was strictly decreasing, and quite unlike the lively intonation that we have described above. The third reason is that in spite of its semantic uniqueness, there is no distinct typographic signature that sets lists apart from other text, which makes list detection a challenging task.

2. List Detection Technique

Since there are no obvious typographic cues such as formats to help identify lists from other unformatted text, we propose to detect lists regions in a given text using the parallel grammatical structure of the elements of a list. This proposal is guided by the anecdotal evidence that people generally indicate the similarity of the items in a list by using the same pattern of words for every item in the list; the list items are separated by commas (or short pauses when speaking) or by conjunctions such as “and” and “or”. An example of such a parallel grammatical structure can be seen in the repeated use of the adjective-noun pattern for all items of the four-item list, “She had dark hair, fair skin, pearly teeth and blue eyes.” Thus, in this paper, we will present a method of list detection using the part-of-speech (POS) tags of the words in the text. Though a novel application in the area of list detection, part-of-speech tags have been previously used for detection of other prosodically prominent events, such as phrase boundary detection [6].

2.1. Basic Algorithm

In this section, we will demonstrate the basic algorithm using a simple example sentence, “I want blueberries, strawberries, mulberries and blackberries.”

Step 1: Mark the words in the corpora by their part-of-speech tags. We used Eric Brill’s transformation-based learning POS tagger [1].

I/PRP want/VBP blueberries,/NNS strawberries,/NNS mulberries/NNS and/CC blackberries./NNS
(Where, PRP indicates pronoun, NNS indicates noun (plural

This research was conducted with support from NSF grant 0205731 “Prosody Generation for Child Oriented Speech Synthesis”.

form), VBP indicates verb in the present tense, and CC indicates conjunction.)

Step 2: For each sentence, indicated by a period or question mark, partition the sentence into clauses isolated by commas.

Clause 0: I want blueberries,
 Clause 1: strawberries,
 Clause 2: mulberries and blackberries.

Step 3: This step is a modification of the well-known string alignment algorithm [3]. For every pair of clauses, retrieve the associated POS tags and measure the edit distance between the tags of one clause and another. The steps for calculating the edit distance D are given below. Let the tags associated with clauses $S1$ and $S2$ be $T1$ and $T2$ respectively.

1. Insert one space at the beginning of both $T1$ and $T2$.
2. Align $T1$ and $T2$ such that each tag/space in either set of tags is opposite a unique tag/space from the other set of tags.
3. Calculate the distance D using the recursion formula:

$$D(i, j) = \min[D(i-1, j) + 1, D(i, j-1) + 1, D(i-1, j-1) + t(i, j)]$$
 Where,
 (a) $t(i, j) = 0.0$ if $T1(i) = T2(j)$.
 (b) $t(i, j) = 0.5$ if $T1(i)$ *partially mismatches* $T2(j)$, i.e. $T1(i) \neq T2(j)$ but they are from the same family of tags. For example, VBP (verb in present tense) and VBD (verb in past tense) are different but from the same family of tags.
 (c) $t(i, j) = 1.0$ if $T1(i)$ *completely mismatches* $T2(j)$, i.e. $T1(i) \neq T2(j)$ and they are from completely different family of tags. For example, a verb versus a noun.

4. Multiply D by 100 to more clearly reflect the relative differences between different edit distances. Normalize D by the number of tags in the longer clause, thus constraining D to be a value between 0 and 100. Save D in a two-dimensional cumulative distance matrix, having clauses 0 through $n-1$ along one dimension, and clauses 1 through n along the other. Note that this matrix will only be filled along the upper diagonal, because each clause is only being aligned with the clauses following it.

In the example sentence, the distortion matrix is:

| | | |
|---|------|------|
| | 1 | 2 |
| 0 | 50.0 | 50.0 |
| 1 | — | 33.3 |

Step 4: Group the clauses of the sentence into all possible groups, $G(i:j)$, of two or more consecutive clauses; i and j are the clauses marking the beginning and ending of the list.

$G(0:1) = \{0, 1\}$: I want blueberries, strawberries,
 $G(1:2) = \{1, 2\}$: strawberries, mulberries and blackberries.
 $G(0:2) = \{0, 1, 2\}$: I want blueberries, strawberries, mulberries and blackberries.

Step 5: For each of the clause groups $G(i:j)$, compute the largest normalized edit distance $L(i:j)$. For given i and j , $L(i:j)$ is calculated by comparing all values $D(m, n)$ from the cumulative

matrix, such that $i \leq m < j$ and $i < n \leq j$, and selecting the largest.

For the clause groups in the example sentence, the L values are as follows:

$L(0:1) = 50.00$, $L(1:2) = 33.33$, $L(0:2) = 50.00$.

Step 6: Of all the $L(i:j)$ values, find the smallest, and the corresponding group, $G(i:j)$. If the smallest L value corresponds to more than one group G , return one or more group, using the following two rules:

1. If $G1(i:j)$ and $G2(m:n)$ both correspond to the same L value, and $i \leq m$ and $j \geq n$ (i.e. $G2$ is a subset of $G1$), return $G(i:j)$. This rule is based on the idea that we want to detect as many items in a list as possible.
2. If $G1(i:j)$ and $G2(m:n)$ both correspond to the same L value, but either $i > m$ or $j < n$ (i.e. $G1$ and $G2$ are disjoint), return both $G1$ and $G2$. This rule is based on the idea that we want to detect as many lists as possible in the sentence.

A list exists if $j - i + 1 \geq 2$ and $L(i:j) < 100$. This rule is based on the assumption that a list has at least two elements in it, and there is at least some similarity between the items of the list. $G(i:j)$ represents the clauses that are in this list.

Thus, in the example sentence, the smallest $L(i:j)$ value is 33.33. Since $2 - 1 + 1 = 2$ and $33.33 < 100$, a list exists in the sentence, and it is represented by $G(1:2)$. So, the list detected in the sentence is “strawberries, mulberries and blackberries.”

2.2. Limitations of the Basic Algorithm and the Subsequent Refinements

Limitation 1: Existence of an Initial Pre-List Part in Sentence Beginning

The first and most obvious limitation is that the algorithm outputs an incomplete list: it does not contain “blueberries” which is clearly a part of the list. The reason for this is that the Clause 0 has an initial pre-list part, “I/PRP want/VBP” whose associated set of tags are *completely mismatched* with the tag associated with Clause 1, thus raising the associated L value to 50.

Refinement 1: Strip Sentence Beginning

To solve this problem, we remove the initial pre-list part of the Clause 0, using the following simple method: we generate all possible unique subsequences, $S(i)$ of Clause 0 while maintaining the strict ordering of the words in the clause. Next, we align each of the subsequences, $S(i)$ with Clause 1 and obtain an alignment cost (or, edit distance). We replace Clause 0 with the subsequence $S(i)$ that corresponds to the lowest alignment cost. Stripping Clause 0 using this method will result in Clause 0 consisting of just “blueberries/NNS” and yield the following cumulative distance matrix:

| | | |
|---|------|------|
| | 1 | 2 |
| 0 | 00.0 | 33.3 |
| 1 | — | 33.3 |

The associated L values are:

$L(0:1) = 0.00$, $L(1:2) = 33.33$, $L(0:2) = 33.33$.

Thus, the list detected after the refinement is, “I want blueberries, strawberries”.

Limitation 2: Implicit Punctuation of Conjunctions “And” and “Or” is not Recognized

The result output after the last modification points to the next limitation: Clause 2 is not included in the list because of the existence of “and/CC” in it; The “CC” tag is *completely mismatched* with the “NNS” tag of Clause 1, thus raising the L value to 33.33. To solve this problem, we must consider the semantic meaning of the conjunction “and”. We know that just like commas, conjunctions, “and” and “or” are used to separate list items.

Refinement 2: Generate an Explicit Punctuation Space

Should we, then, partition the input sentence using commas as well as “and” and “or” conjunctions? If yes, how would partitioning into clauses affect the nested list structure in the sentence, “Hans arrived before the king with all these things - (with the money, and the gold, and the silver and (the cows, sheep and goats.))”? This method of partitioning would remove the nested list structure of the sentence. Removing the nested list structure erases the relative semantic similarity between “cows”, “sheep” and “goats”; it also erases the relative dissimilarity between “money, gold and silver” and “cows, sheep and goats”, instead, imposing an apparent sense of similarity between the two groups of list items. So, it seems unwise to partition the input sentence in such a manner right away. We decided to first generate an explicit punctuation space, and then remove all clause-initial “and” and “or” conjunctions. The punctuation space is generated in two steps: First, for every sentence, each “and” and “or” not preceded by a comma is marked as a *non-clause-initial conjunction*. The second step is based on the idea that preceding a non-clause-initial conjunction by a comma or leaving it out are both grammatically correct; thus, for n non-clause-initial conjunctions in a sentence, 2^n possible arrangements of commas preceding the non-clause-initial conjunctions are generated.

For the example sentence, “I want blueberries, strawberries, mulberries and blackberries.”, the punctuation space contains two possible types of correct punctuation :

Punctuation 1: I want blueberries, strawberries, mulberries and blackberries.

Punctuation 2: I want blueberries, strawberries, mulberries, and blackberries.

Note that the sentence version generated by Punctuation 2 has a comma after “mulberries”. Thus, partitioning this version using Refinement 1 and 2 will yield the following clause set, (since both the initial pre-list part in Clause 0, and the clause-initial “and” in Clause 3 are removed):

Clause 0: blueberries,

Clause 1: strawberries,

Clause 2: mulberries,

Clause 3: blackberries.

Not surprisingly, the associated L values are

$L(0:1) = 0.00$, $L(1:2) = 0.00$, $L(2:3) = 0.00$, $L(0:2) = 0.00$,
 $L(1:3) = 0.00$, $L(0:3) = 0.00$.

Since all L values are zero, we grab the corresponding group G that represents the maximum number of clauses, $G(0:3)$. Thus, the list output is “blueberries, strawberries, mulberries, and blackberries.”

Limitation 3: Every sentence has a list!

A list is detected by the basic algorithm if $j - i + 1 \geq 2$ and $L(i : j) < 100$. $L(i : j) < 100$ if there is a *partial match* between any two elements of the tagsets associated with any two clauses. Thus, for a sentence like, “Go/VB Katarina/NN, go/VB with/CC Julie/NN.”, where there is clearly no list but there the

“NN” tag associated with “Katrina” of Clause 0 matches the “NN” tag associated with “Julie” of Clause 1. Thus, $L(0:1) < 100$, and $1 - 0 + 1 \geq 2$, and the basic algorithm incorrectly detects a list, “Go Katrina, go with Julie”!

Refinement 3: Stricter Definition of a List

The above problem is solved by redefining the rule for list detection: A list is detected by the algorithm if either $j - i + 1 \geq 3$ and $L(i:j) < 100$, or if $j - i + 1 = 2$ and $L(i:j) = 0$. Thus, the algorithm detects a list if either the list contains three or more clauses that are at least *partially matched*, or if the list has two clauses that are *completely matched*. Using this strict definition of list detection, no list would be detected in “Go/VB Katarina/NN, go/VB with/CC Julie/NN.”, but “blueberries, strawberries, mulberries, and blackberries.” would be detected in “I/PRP want/VBP blueberries./NNS strawberries./NNS mulberries/NNS and/CC blackberries./NNS”

Limitation 4: List Indicators are Ignored

Certain lists are marked by very strong indicators such as numbers, as in, “One sprang under the table, the second into the bed, the third into the stove, the fourth into the kitchen, the fifth into the cupboard, the sixth under the washing-bowl, and the seventh into the clock-case.” or by other comparative adjectives (“eldest”, “youngest”, “first”, “last”, etc.) Lists containing these strong list indicators are very easy to detect, even if the parallel grammatical structure between the clauses is weak. The basic algorithm did not detect the presence of list indicators and relied only on alignment costs for detecting lists. Thus, if the parallel grammatical structure is weak, as in, “The one drew the thread and trod the wheel, the other wetted the thread, the third twisted it, and struck the table with her finger, and as often as she struck it, a skein of thread fell to the ground that was spun in the finest manner possible.”, the algorithm, using the stricter definition of list detection, detected no list in the sentence.

Refinement 4: Detect List Indicators

The basic algorithm is modified so that after the sentence has been partitioned into clauses, it scans the clauses for presence of list indicators. If list indicators are found, a list is detected without even computing the alignment costs.

3. Evaluation

Having made the previously outlined refinements to the basic algorithm, the performance of the algorithm was tested against human subjects.

3.1. Test Data

A set of 160 sentences, each containing 6 clauses on average, were used for testing the performance of the list detection algorithm. These sentences were obtained from Margaret Hunt’s version of the Grimm’s Fairy Tales. Of the 160 sentences, 80 were chosen because they were deemed to contain lists. The remaining 80 were chosen as a control set: they matched the first set of eighty in terms of commas, however they did not contain lists, (as judged by the authors).

These sentences were chosen from Grimm’s Fairy Tales for two reasons: First, we noticed that there is stronger presence of mnemonic devices, like lists, in older stories compared to modern stories; this is possibly because modern stories are often written with a reliance on the accompanying picture book to fuel imagination, while older stories relied only on strong rhetorical devices for that purpose. The second reason is that we found the

lists in Grimm’s stories to be very complex; good performance on sentences this complex bodes good performance in general.

3.2. Experiment

First the 160 sentences were randomized so that list-bearing sentences were mixed in with sentences containing no lists. Then four sets of test stimuli were prepared, each with a different ordering of the test sentences:

- Sequence 1: Sentences 0–79, Sentences 80–159.
- Sequence 2: Sentences 159–80, Sentences 79–0.
- Sequence 3: Sentences 80–159, Sentences 0–80.
- Sequence 4: Sentences 79–0, Sentences 159–80.

Each of the four human subjects was given one of the four sequences. All four subjects were asked to demarcate parts of the sentence that they considered an “intonational list”, meaning lists that are identified as such because they would read it with a particular list-like intonational pattern, rather than just lists that contain strong list indicators. If the subjects found cases where they were several meaningful ways of partitioning the sentence into a list, they were asked to mark the multiple versions of the list structure. and indicate their confidence in each of the versions by a number between 1 and 10; the sum of the confidence levels associated with each version of the sentence had to be 10. The same set of 160 sentences was given to the list detection algorithm as test data, and the output results recorded.

3.3. Results

The results output by the detection algorithm were compared to the results recorded by the human subjects. And the comparative results were analyzed to answer two questions:

1. Can the list detection algorithm detect the presence of a list in a sentence reliably?
2. Given that the algorithm detects lists reliably, how many times is the list detected by the algorithm identical to the lists perceived by humans?

The first question is answered by computing the *majority vote* for each sentence. If three or more subjects agree that there is a list, majority vote = 1, otherwise, majority vote = 0. The majority vote for every sentence is compared to the algorithm’s decision on whether there is a list in that sentence. Table 1 shows the percentage of agreement between majority vote and the algorithm’s decision. The summation of the values along the left diagonal shows that a majority of the subjects agreed with the algorithm’s decision 80% of the time. It is important that a TTS system reads **only** true lists in a list-like intonation, otherwise it will confuse the listeners by implying a greater similarity between some groups of clauses. Whereas if, the TTS system reads the list in a non-list-like manner, the listeners may find the intonation monotonous, nevertheless, they can still detect the list because of strong semantic similarity among the list elements. Thus in our case, having obtained a higher precision rate (92%) compared to the recall rate (82.7%) is encouraging.

| | | Algorithm | |
|---------------|---------|-----------|---------|
| | | List | No list |
| Majority vote | List | 53.75 | 11.25 |
| | No list | 8.75 | 26.25 |

Table 1: Performance of algorithm against the majority vote in terms of detecting presence of a list.

The second question is answered by taking into account only those sentences in which a majority of the subjects identified a list. There are 103 such sentences. For each of these sentences, the list(s) output by the algorithm is compared to those identified by each of the subjects comprising the majority, and classified as one the following: *Misclassified*, meaning that the algorithm detected no list; *Under-detected*, meaning that the list identified by the system is a subset of the list identified by the subject; *Identical*, meaning that the list identified by the algorithm is identical to the list identified by the subject; and *Over-detected*, meaning that the list output by the system is a superset of the list identified by the subject. The results of the comparison are shown in Table 2. Thus, this detection algorithm outputs lists identical to a majority of the subjects 40.5% of the time. This accuracy rate can be considered reasonably good in light of the fact that for these sentences with an average of six clauses, there are 15 (6+5+4+3+2+1) possible lists, implying a base rate performance of just $1/15 = 6.3\%$.

| Misclassified | Under-detected | Identical | Over-detected |
|---------------|----------------|-----------|---------------|
| 17.5% | 35% | 40.5% | 7% |

Table 2: Performance of algorithm against majority, in terms of identical list matches

4. Conclusion

In conclusion, we have shown that a simple algorithm based on the similarity of POS sequences of the clauses in a sentence can be used both to detect whether a sentence has a list type, and which clauses in the sentence comprise the list. The difficulty of the task for humans, as demonstrated by the fact that on 10% of the sentences, the judges were evenly divided in their opinion, sets an upper limit to what can be expected of any detection algorithm.

The next phase of this research will consist of mathematical characterization of the intonation contour used in *list* types.

5. Acknowledgements

We would like to thank Richard Sproat and Alan Black for their helpful suggestions on the subject of this paper, and Lyle Kopnicki for his help with the programming involved in this project.

6. References

- [1] Eric Brill. “Brill: Trainable Part of Speech Tagger”, In <http://www.cs.jhu.edu/~brill/code.html>, last updated 1996.
- [2] M. Lieberman and Janet Pierrehumbert. “Intonational Invariance Under changes of pitch range and length”, In *Language Sound Structure*, MIT Press, Cambridge, Massachusetts, p157-233, 1984.
- [3] D. Sankoff and J. B. Kruskal. “Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison”, Addison-Wesley, Reading, 1983.
- [4] Richard Sproat, J. Hu and H. Chen “Emu: An E-Mail Preprocessor For Text-To-Speech”, In *Proceedings of the IEEE 1998 Workshop on Multimedia Signal Processing*, Los Angeles, California, 1998.
- [5] Richard Sproat, J. van Santen and Joseph Olive. “Further Issues”, In *Multilingual Text-To-Speech Synthesis*, The Bell Labs Approach, p245-254, 1998.
- [6] M. Q. Wang and J. Hirschberg. “Automatic Classification of Intonational Phrase Boundaries”, In *Computer Speech and Language* 6(2), p175-196, 1992.