The 6th Intl. Workshop on Spoken Language Technologies for Under-Resourced Languages
29-31 August 2018, Gurugram, India

Text Normalization for Bangla, Khmer, Nepali, Javanese, Sinhala and Sundanese Text-to-Speech Systems

Keshan Sodimana, Pasindu De Silva[2]*, Richard Sproat*[1]*, Theeraphol Wattanavekin*[1]*, Chenfang Li*[3]*, Alexander Gutkin*[1]*, Supheakmungkol Sarin*[1]*, Knot Pipatsrisawat*[1]

[1]Google,[2]Google (on contract from Teledirect Pte Ltd),[3]Google (on contract from Optimum Solutions Pte Ltd)

{ksodimana,pasindu,rws,twattanavekin,chenfangl,agutkin,mungkol,thammaknot}@google.com

Abstract

Text normalization is the process of converting non-standard words (NSWs) such as numbers, and abbreviations into standard words so that their pronunciations can be derived by a typical means (usually lexicon lookups). Text normalization is, thus, an important component of any text-to-speech (TTS) system. Without text normalization, the resulting voice may sound unintelligent. In this paper, we describe an approach to develop rule-based text normalization. We also describe our open source repository containing text normalization grammars and tests for Bangla, Javanese, Khmer, Nepali, Sinhala and Sundanese. Finally, we present a recipe for utilizing the grammars in a TTS system.

Index Terms: text normalization, text-to-speech, low-resourced languages

1. Introduction

Recently, text-to-speech (TTS) systems are becoming popular, powerful and more human-like, thanks to novel techniques like WaveNet [1] and to the availability of high quality datasets. These systems have to be able to handle text in various forms, including text with non-standard words (NSWs) such as abbreviations, numbers, and telephone numbers [2]. Typically, the systems cannot identify the pronunciations of these NSWs by using a lexicon/dictionary or a letter-to-pronunciation system. Thus, a text normalization component is needed to identify and convert any NSWs encountered in the input text into standard words so that their pronunciations can be easily produced. Creating a text normalization component for a new language typically requires a collaboration between native speaker(s) of the language and an engineer, because it is challenging to find a person with sufficient technical and linguistic knowledge in a target language. In general, it is important for any teams wishing to scale their work to multiple low-resourced languages (such as in an industrial setting) to be able to spend less effort acquiring linguistic data for each new language.

In this paper, we describe a process for creating a text normalization component for 6 low-resourced languages: Bangla (Bangladeshi Bengali), Javanese, Khmer, Nepali, Sinhala and Sundanese. We worked with native speakers in these languages to create grammar rules for normalizing NSWs from frequent semiotic classes. These grammars can be utilized by open source text normalization framework called Sparrowhawk [3] and be integrated into actual TTS systems. We open sourced these grammars in the hope that they will empower local communities and researchers who are interested in developing TTS voices for these low-resourced languages.

The rest of the paper is organized as follows. In the next section, we discuss related work in the area of text normalization and highlight the key contributions of our work in this paper. Then, we describe a process for creating text normalization component for a new language. In Section 4, we list and explain the resources that we open source. Then, in Section 5, we provide a recipe for utilizing these resources in an actual TTS system. Finally, we end the paper with some discussions and conclusions.

2. Related Work

Although a considerable amount of work has been done in the area of text normalization for well-resourced languages, there is a limited amount of literature devoted to text normalization for low-resourced languages. A regular expression-based text normalization system for Bangla is described in [4]. The authors showed that the system had 99% accuracy for tokens in three classes: floating point numbers, currency and time. An implementation of tokenization and text normalization component for Sinhala using Festival text-to-speech framework is described in [5]. A minimally supervised approach to text normalization for Khmer is presented in [6]. However, the work focused only on number normalization and the system was not made available publicly. There is no significant work focusing on text normalization of a wide range of semiotic classes for Javanese, Nepali and Sundanese. In [7], the authors described a Bangla TTS system whose text normalization component becomes a part of this work. All the text normalization rules discussed in this paper were written using Sparrowhawk which is the open source version of Kestrel, a component of the Google text-to-speech synthesis system [3]. In this system, the text normalization rules are compiled into weighted finite-state transducers (WFSTs) and applied to input text to produce outputs.

The main contributions of this paper are as follows:

1. we describe a general method for working with native speakers in identifying patterns and grammars needed to normalize text,
2. we make available text normalization grammars and their test cases for a wide range of common semiotic classes for 6 low-resourced languages,
3. we provide a recipe for utilizing these grammars and for integrating them into actual text-to-speech systems.

We believe that these contributions could be very useful for researchers working in the field of natural language processing for low-resourced languages.

Table 1: *Semiotic classes covered by this work.*

Semiotic class	Description	Example inputs (in English)
ABBREVIATION	Abbreviations	Dr., Mr., Ms.
ADDRESS	Address expressions	34 Main st.
CARDINAL	Normal numbers	3479, 90,581
DATE	Date expressions	3/5/2018, 2018-01-02
DECIMAL	Numbers with decimal points	234.79, 11.98
DIGIT	Digit sequences which read digit-by-digit	007, 123-4
ELECTRONIC	Email addresses and URLs	hello@example.com, www.google.com
EMOTICON	Emoticons/Emojis	:-), 8-)
FRACTION	Fractions with numerators and denominators	$\frac{2}{5}$, $1\frac{3}{4}$
LSEQ	Letter sequences	FBI, IMF
MEASURE	Quantities with units	10 km, 30 sq.m.
MONEY	Quantities with currency symbols	$2.5, 7.38$
ORDINAL	Ordinal numbers	1st, 3rd
TELEPHONE	Phone numbers	+(94) 123 4567, 310-200-4791
TIME	Time expressions	14:55, 4.33pm
VERBATIM	Text that should be read by symbol names	$\Delta X + \Delta Y$

3. Implementing text normalization

Text normalization, as described in this paper, takes segmented text as input. For languages where words are already separated by whitespaces (namely, Bangla, Javanese, Nepali, Sinhala, Sundanese), this input is just the plain text. However, for languages that do not use whitespaces to separate words (namely, Khmer), the input to the text normalization component is the output from a word segmenter [8]. Word segmentation is beyond the scope of this work.

3.1. System structure and semiotic classes

In this study, we follow the structure of text normalization system described in [3]. Text normalization is divided into two phases. First, input text is analyzed and NSWs are *classified* into semiotic classes [9]. In this phase, some input tokens may be grouped together. For example, input text "15 km" may be grouped together and classified as a measurement token. Then, *verbalizer* grammar for each semiotic class will convert the classified NSWs into standard text accordingly. Table 1 shows the semiotic classes covered by our grammars. We focused on these classes because they are commonly found in TTS inputs and are standard classes implemented throughout TTS systems for various languages at Google.

After identifying the semiotic classes, we reached out to native speakers in each language with a questionnaire. The questionnaire contains a set of questions for each semiotic class. The questions were designed to capture the writing and verbalizing system of the language. First, we started with understanding the basics of cardinal numbers. We ask questions like how to read numbers from 0 – 20, hundreds, thousands, millions, billions and other irregular number names. The native speakers can supply additional information for special numbers not present in the questionnaire. For ordinal numbers, we ask whether there is any special prefix or suffix to indicate ordinality and whether any inflection is involved when reading these numbers. For date/time, we ask for all common date/time formats used in the language. For example, what orderings of year, month, day are possible and whether the time expressions use 12h or 24h format and whether there are any time period indicators (such as "a.m." or "p.m."). For less complex classes such as ABBREVIATIONS, VERBATIM, and EMOTICONS, we simply asked for translations of various input symbols in the language.

3.2. Language-specific considerations

When working across many languages, there are many nuances that we need to be cautious about. In this part of the paper, we highlight some language-specific considerations that affected the implementation of text normalization.

- Bangla: Bangla is an inflection language. Therefore, the grammars need to handle all different inflection cases properly. Moreover, the language has 4 different time indicators (similar to "a.m./p.m.") that are commonly used in time expressions.
- Khmer: Since word segmentation is required for Khmer prior to text normalization, the grammars need to sometimes cover the possibilities of incorrect segmentation outputs. The zero-width space character (U+200B) is used inconsistently in writing and needs to be cleaned up for lexicon lookup to succeed.
- Nepali: Nepali uses native calendar in addition to the Gregorian calendar. These 2 calendar systems have distinct month names. The grammar needs to pay special attention to indicator words and date formats to identify the correct calendar system used in each date expression.
- Sinhala: All measurement units are verbalized before the quantities. For example, the input "10 km" will be verbalized as "kilometer ten" in Sinhala.
- Javanese and Sundanese: These 2 languages are relatively simpler, because they use Latin characters and Arabic numbers. Moreover, their writing and verbalization systems are both similar to those of Indonesian. We most needed to focus on translating words into the target languages.

Besides the above considerations, all languages in this set, other than Javanese and Sundanese, have their own alphabets and digits. Bangla uses the Bengali alphabet. Khmer uses the Khmer script. Nepali uses Devanagari and Sinhala uses the Sinhala alphabet. As mentioned above, Khmer writing contains inconsistent usages of the zero-width space character. Bengali, Nepali and Sinhala also utilize zero-width non-joiner (U+200C) and joiner (U+200D). The use of these unique character sets and symbols require special attention when classifying and tokenizing text in these languages.

The unique phenomenon mentioned above show that asking the native speakers (who often do not have technical background) the right questions is vital to correctly classifying and converting NSWs in these languages.

Table 2: *Example input and output of each step in the text normalization pipeline for Bangla.*

Stage		Example
Input text		$1.20
	↓	
Classifier output (protocol buffer)		money { currency: "usd" amount { integer_part: "1" fractional_part: "20"} }
	↓	
Verbalizer output (standard words)		এক ডলার বিশ সেন্ট

3.3. Grammars and unit tests

Based on the answers to the questions, we also obtained test cases as important by-products of this process. These test cases are used to create unit tests for the grammars. So, we make sure to solicit both common and corner cases and include them in our tests to verify that our rules cover everything correctly.

Once we have the test cases and explanations about how to classify and verbalize different semiotic classes, we create grammar rules to do the classification and verbalization. The grammars that we created are Thrax grammars [10], which consist of mostly regular expressions and context-dependent rewrite rules. The grammars can then be compiled with the Thrax grammar compiler, which turns them into archives of finite state transducers (known as “FARs” – Finite State Archives).

First, the input text will be classified into different semiotic classes. The output of this classification step is a protocol buffer[1] which contains information about the original token and its semiotic class. Then, the protocol buffer will be passed to the verbalizer component, which converts the token into one or more standard words based on the verbalization rules for the semiotic class associated with the token.

For example, the classification of input "100 m" will output the following protocol buffer:

```
measure { decimal { integer_part: "100" }
units: "meter" }
```

This protocol buffer indicates that this input text is a kind of "measure" token. Notice that the protocol buffer contains sufficient information (in this case, the quantity and the unit) to verbalize this token at a later stage. The verbalizer will then take this as its input and apply the rules to convert it into output text that consists only of standard words, which can be found in the lexicon.

Inside the classifier component, many languages share the same rules for some semiotic classes. For example, all 6 languages have similar formats for telephone numbers; the input can start with an optional ’+’ sign, followed by a country code and then a 10-digit number with optional spaces, hyphens and parentheses in between. Similarly, some of the time and date expressions share similar formats. For language-agnostic items such as email addresses, web addresses, symbols (verbatim), and emoticons, the rules are extensively shared across all languages. On the other hand, since the outputs of the verbalization step are words in the target language, verbalizers have to be written for each semiotic class in each language. Table 2 shows an example that illustrates the end-to-end flow of data in our system in Bangla.

As mentioned earlier, all of our grammars come with unit tests for individual classifiers and verbalizers. Table 3 lists the number of test cases for each semiotic class.

[1]Google’s Data Interchange Format, `http://code.google.com/p/protobuf`.

4. Open-sourcing the resources

We have open sourced the implementation of the classifiers and verbalizers described in the previous section along with the test cases to verify the rules for all 6 languages mentioned above. Unless otherwise noted, all original files are licensed under the Apache License, Version 2.0 [11] and where specifically noted, some datasets are licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0) [12].

5. Usage recipe

In this section, we will provide instructions for utilizing our grammars in an actual TTS system.

5.1. Prerequisites

Bazel: This is the build system we use to compile Thrax grammars. The installation guide is provided in [13].
Language resources: This is the repository that contains the open source grammars and other helper scripts. Its contents should be downloaded from [14].

5.2. Compiling and running

There are two ways to run the text normalization system. The first way is running individual semiotic grammars. The second way is running all grammars in an end-to-end Sparrowhawk system. One may wish to run grammars individually for testing and debugging purposes. Individual grammars can be compiled and run with the following commands:

- Compiling a grammar file. This can be done by
  ```
  $ bazel build //<lang>/textnorm/classifier:
  <grammar name>_thrax_compile_grm
  ```
  For example,
  ```
  $ bazel build //si/textnorm/classifier:
  measure_thrax_compile_grm
  ```
  would compile the measure classifier grammar for Sinhala and produce `measure.far`.
- Building `thraxrewrite-tester` tool. This is the tool for utilizing and interacting with the above generated *FAR* file.
  ```
  $ bazel build @thrax//:thraxrewrite-tester
  ```
- Using the `thraxrewrite-tester` tool to run the *FAR* file.
  ```
  $ bazel-bin/external/thrax/
  thraxrewrite-tester
  --far=bazel-genfiles/si/textnorm/
  classifier/measure.far
  --rules=MEASURE_MARKUP
  ```
  This command will display an interactive prompt where users can enter arbitrary text in Sinhala and receive output from classification using the measure grammar.

Table 3: *Number of test cases for each semiotic class.*

Semiotic class	Bangla	Javanese	Khmer	Nepali	Sinhala	Sundanese
ABBREVIATION	15	n/a	n/a	n/a	n/a	n/a
CARDINAL	190	70	274	261	265	70
DATE	66	15	45	16	50	15
DECIMAL	20	10	40	36	36	10
DIGIT	20	19	28	17	17	18
FRACTION	15	18	15	15	14	15
ELECTRONIC	14	n/a	15	22	22	n/a
EMOTICONS	n/a	34	35	35	35	34
LSEQ	15	20	24	59	69	20
MEASURE	19	23	174	20	50	23
MONEY	19	16	75	12	37	16
ORDINAL	20	44	n/a	28	19	45
TELEPHONE	12	15	12	12	12	15
TIME	37	14	13	14	14	14
VERBATIM	27	23	28	30	27	23

To run the grammars in an end-to-end system, where classification and verbalization are applied to each input text, we need to utilize Sparrowhawk, which is an open source text normalization system for TTS. Sparrowhawk is distributed as part of the language resource repository listed under prerequisites. In the language resource repository, we have also included necessary files needed to set up Sparrowhawk for each language. For example, Sparrowhawk configurations for Sinhala can be found at

```
https://github.com/googlei18n/
language-resources/tree/master/si/sparrowhawk
```

Configurations for other languages can be found in their respective folders. To build and run Sparrowhawk for a language, follow these steps (using Sinhala as an example):

1. Build the files required by Sparrowhawk
   ```
   $ bazel build si/sparrowhawk:si_sparrowhawk
   ```

2. Run Sparrowhawk interactively
   ```
   $ bazel-bin/si/sparrowhawk/si_sparrowhawk
   ```

Sparrowhawk can also be used as a command line tool to process one input sentence. For example, to process input text `"www.google.com"` with Sinhala text normalization, run

```
$ echo www.google.com |
bazel-bin/si/sparrowhawk/si_sparrowhawk
```

This method allows Sparrowhawk to be quickly integrated with any TTS system as it allows users to treat Sparrowhawk as a component that takes in segmented text and outputs normalized text. The authors of Sparrowhawk also provided instructions for integrating it with Festival [15] as well [16]. This allows users to adds a Sparrowhawk Festival module which can be used by Festival recipes to normalize text. Describing this process is beyond the scope of this paper.

6. Discussion

In this work, we focused on creating a set of text normalization grammars for 6 low-resourced languages. The grammars were aimed at covering semiotic classes that were commonly found as TTS inputs. We ensured good coverage and robustness of our system with unit tests for each component of each language. A future direction would be to evaluate these grammars more systematically against some standard (unseen) text corpora so that they can be compared against other text normalization approaches.

Another area for future work is to further improve the coverage of our grammars. For example, currently, our measurement grammars focus on commonly used measurement units in each language. More units could be included in our grammars in future versions. For example, we are currently lacking test cases for abbreviations in many languages. At the same time, there could be many more specialized classes of non-standard words that could be added to the grammars. Examples include flight numbers, bank account numbers, highway numbers, etc. The importance of these other classes varies from language to language and warrants further investigations.

7. Conclusions

In this paper, we described a process for creating text normalization grammars for low-resourced languages. Moreover, we presented text normalization grammars, both classifiers and verbalizers, for 6 low-resourced languages: Bangla, Javanese, Khmer, Nepali, Sinhala, and Sundanese. These grammars, along with test cases, are available for free to the public. We also described a process for testing and utilizing these grammars through Sparrowhawk text normalization system.

8. Acknowledgments

We would like to thank Daan van Esch and Viviana Montoya for creating the questionnaire that was used as a basis of our work. We would also like to thank Pattara Sukprasert, Khan Md Anwarus Salam, Saroj Kumar Dhakal, Jaka Aris Eko Wibawa, and Makara Sok for their contributions to creating text normalization grammars presented in this paper.

9. References

[1] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," in *Arxiv*, 2016. [Online]. Available: https://arxiv.org/abs/1609.03499

[2] R. Sproat, A. W. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards, "Normalization of Non-Standard Words," *Computer Speech and Language*, vol. 15, no. 3, pp. 287–333, Jul. 2001.

[3] P. Ebden and R. Sproat, "The Kestrel TTS text normalization system," *Natural Language Engineering*, vol. 21, pp. 333–353, 2015.

[4] F. Alam, S. Habib, and M. Khan, "Text normalization system for Bangla," in *Conference on Language and Technology*, 2009.

[5] R. Weerasinghe, A. Wasala, V. Welgama, and K. Gamage, “Festival-si: A Sinhala Text-to-Speech System,” in *Text, Speech and Dialogue, 10th International Conference, TSD 2007, Pilsen, Czech Republic, September 3-7, 2007, Proceedings*, 2007, pp. 472–479.

[6] K. Gorman and R. Sproat, “Minimally supervised number normalization,” *Transactions of the Association for Computational Linguistics*, vol. 4, pp. 507–519, 2016.

[7] A. Gutkin, L. Ha, M. Jansche, K. Pipatsrisawat, and R. Sproat, “TTS for Low Resource Languages: A Bangla Synthesizer,” in *10th edition of the Language Resources and Evaluation Conference*, Portorož, Slovenia, May 2016, pp. 2005–2010.

[8] V. Chea, Y. K. Thu, C. Ding, M. Utiyama, A. Finch, and E. Sumita, “Khmer Word Segmentation using Conditional Random Fields,” *Khmer Natural Language Processing*, pp. 62–69, 2015.

[9] P. Taylor, *Text-to-Speech Synthesis*. Cambridge University Press, 2009.

[10] B. Roark, R. Sproat, C. Allauzen, M. Riley, J. Sorensen, and T. Tai, “The OpenGrm Open-source Finite-state Grammar Software Libraries,” in *Proceedings of the ACL 2012 System Demonstrations*, ser. ACL ’12. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 61–66.

[11] “Apache license, version 2.0,” http://www.apache.org/licenses/LICENSE-2.0.

[12] “Creative Commons Attribution 4.0 International License,” https://creativecommons.org/licenses/by/4.0.

[13] “Installing Bazel,” https://docs.bazel.build/versions/master/ install.html.

[14] “Google Internationalization Language Resources,” https://github.com/googlei18n/language-resources.

[15] A. W. Black and P. A. Taylor, “The Festival Speech Synthesis System: System documentation,” Human Communciation Research Centre, University of Edinburgh, Scotland, UK, Tech. Rep. HCRC/TR-83, 1997, avaliable at http://www.cstr.ed.ac.uk/projects/festival.html.

[16] “Integrating Sparrowhawk with Festival,” https://github.com/google/sparrowhawk/tree/master/ documentation#integrating-sparrowhawk-with-festival.