Text Normalization for Speech Systems for All Languages

Athiya Deviyani, Alan W Black

Language Technologies Institute, Carnegie Mellon University, USA

{adeviyan, awb}@cs.cmu.edu

Abstract

Most text-to-speech systems suffer from the limitation that its inputs should be a set of strings of characters with standard pronunciation, and struggle when given input is in the form of symbols, numbers, or abbreviations that often occur in real text. One of the most common ways to address this problem is to automatically map non-standard words to standard words using statistical, neural and rule-driven methods. However, despite the significant efforts of normalizing such words, there is just too much variability in existing corpora such that it is extremely challenging to capture edge cases. In this work, we propose a tool which aids data collection from (non-programmer) native speakers to allow numbers and other common non-standard words to be mapped to standard words that can be pronounced correctly by a synthesizer, while addressing related problems such as identifying common non-standard words appear in text and how do we ask questions from native speakers to get sufficient information to allow a useful normalization of non-standard words.

Index Terms: text normalization, text-to-speech, speech synthesis, low-resource languages, data collection

1. Introduction

Text-to-speech (TTS) has improved to the level that given a corpus of well recorded, well-spoken and a transcript, it is possible to build a high quality text-to-speech system. However, such systems are almost always limited to input that are 'words', i.e. strings of characters that have a standard pronunciation, rather than symbols, numbers, abbreviations, etc. that appear in real text. If we want to deploy these text-to-speech engines we will need to address the tokens in text that are not simple words.

There have been attempts to automate the mapping of these additional non-standard words to standard words, using various statistical, neural and rule-driven approaches [1][2][3]. These methods certainly work, but have poor generalizability to new languages. The development of commercial synthesizers typically includes significant effort normalizing such non-standard words. But although there are often a set of well-defined cases, there is also an extremely long tail of rare cases that are hard to cover.

In our work we provide robust methods for building synthesizers, typically aimed at low-resource languages, and targeted to be easily deployed as the base text-to-speech engine on cheap (mostly) Android devices. However, although these voice-building techniques typically work well from both found data (e.g. religious books) or using techniques to create and record your own phonetically balanced corpora, these techniques do not address the issues of non-standard words.

Except in specialized limited domain cases, it is impractical to design a recording corpus that covers even as well-defined a domain as number pronunciation. For most languages the pronunciation of numbers (i.e. string of digits) is well defined, and effectively rule driven, but in our case how can we effectively encode that information into our synthesizers without having developers write their own code. Explicit code would require specialized binaries, and the ability to write safe and efficient code, which limits accessibility. We would prefer to be able to collect sufficient data from a native speaker to allow numbers and other common non-standard words to be expanded to standard words that can be pronounced correctly by the synthesizer.

The problem of addressing non-standard words in a low-resource setting brings forward a number of research questions:

- **RQ1:** Which non-standard words appear in text?
- **RQ2:** How do we ask questions to get our answers?
- **RQ3:** How do we make it easy to answer such questions?
- **RQ4:** How do we address variability?
- **RQ5:** How do we extend this to a full front-end?

There might be other solutions to this issue. For example you might be able to predict number pronunciation for an unknown language if you know something about the language family. For example if a written word in text appears next to something that could have a number associated to it, and it starts with "octo-" given knowledge of Latin based languages it is reasonable to consider it to be the word for the digit "8" (well, the 10th month name notwithstanding). Given that it is likely (even in religious texts) that the low numbers appear as words in text, you might be able to derive this. Larger numbers are probably less likely, and floating point numbers even less so (though perhaps how floating point numbers are spoken and somewhat cross-lingually standardized). Another method might be to design a corpus for people to say that contains all the variations for the numbers, but that might be hard in general, and if you only have found data, you might not be able to get access to a speaker (and recording situation).

It is worth mentioning that in many languages as the complexity of the number strings increases there may be a tendency to move to more international standards for numbers (e.g. how exponentiation is spoken, if there is a standard at all). There may even be a tendency to move to the language of education for numbers (e.g. English/French/Mandarin) as arithmetic may have been taught to everyone in that language. And even if that's not standard, if everyone knows the English for numbers, then we can probably better pronounce the numbers in English than in the native language.

2. Motivation

The text normalization task requires converting a written representation of a text into a representation of how that text is to be read aloud. Although the task seems relatively mundane, it is very important as a major degradation of perceived quality in TTS systems can be traced to problems involving text normalization.

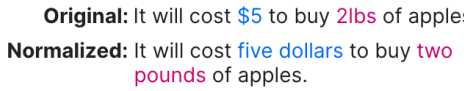

Figure 1: *Text normalization of currency and measure. The colors in the original text correspond to the standard form in the normalized text.*

In the text normalization example shown in Figure 1, the original sentence contains "$5" and "2lbs", which are categorized as non-standard words [4]. For a text to speech system to read the sentence out loud, every occurrence of a non-standard word needs to be normalized. This is a difficult task as non-standard words can appear in different forms or semiotic classes [5], such as measures, currency, date and time, telephone numbers, and more.

Furthermore, the problem seems to prove to be even more challenging in the multilingual setting. This is because every language is likely to have different counting systems. While most languages such as English uses the base-10 counting system, the French language uses the base-20 system where the number "80" is pronounced as "quatre-vingts" which means "four twenties". This is similar to Danish, where "50" is normalized to "halvtreds" which means "two and a half times twenty".

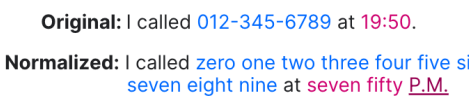

Figure 2: *Text normalization of phone number and time. The colors in the original text correspond to the standard form in the normalized text.*

Each semiotic class needs to be processed differently into its normalized form. This can be shown by the example in Figure 2, where a 10-digit phone number is normalized into individual numbers while an instance of time is broken down into the hour and minute components. This is difficult as phone numbers vary in length, and there are special cases where a three-digit number such as "911" is also a valid phone number. This in turn makes the mapping process highly context-dependent. Additionally, depending on the language and the use case of the normalized form, the time class can be normalized further and added with an AM/PM token by deciding whether or not the value is before or after 12 PM.

Figure 3: *Example of text normalization for the number "125000" across the English, Mandarin, and Hindi languages.*

Even languages with the same base system vary in the way they chunk the numbers. In English, numbers follow the base-10 counting system and are separated into chunks of three. Both Hindi and Mandarin follow the base-10 counting system, however in Hindi, they chunk the numbers into groups of 3 for the last three digits and groups of 2 for the remaining digits, while they chunk the numbers after each ten thousand in Mandarin. This example is highlighted in Figure 3.

There are even more discrepancies in text normalization when we go into the details on how to normalize multilingual semiotic classes, such as how the currency is read out, how the time is divided, positioning of special symbols, length of phone numbers in different countries, and more. In English, assigning the AM/PM token to a time instance depends on whether the time is before or after 12 PM, i.e. either day or night. In the Indonesian language, however, the time is defined into morning, afternoon, evening, and night, depending on when the sun rises and sets. For example, "15:30" will be normalized to "three thirty in the evening" while "06:45" will be normalized to "six forty five in the morning". Additionally, normalizing text belonging to the date semiotic class is also quite tricky. Often, the text "2010" is read out as "twenty ten" while the first ten years within a century (e.g. "2000" up to "2009") will be read out as a normal four-digit number. Although reading out year values as a normal number will not completely alter the meaning of a text, it can potentially throw off users listening to the synthesized text as most English speakers are not familiar with reading "1999" as "one thousand nine hundred ninety nine".

The examples above illustrate the various discrepancies in non-standard to standard text mapping in a very small subset of semiotic classes across very few languages. For example, some currencies such as the United States Dollar have a plural form (dollars), however currencies such as the Indonesian Rupiah are commonly used in its singular form, regardless of the accompanying monetary value. This poses a problem where expanding the text normalization system across multiple languages can increase the number of mapping functions exponentially. From this, we propose a method to collect data regarding the counting system, digit mapping, and relevant information with respect to the different semiotic classes from native speakers of a language to allow for the development of a generalizable text normalization system.

3. Related work

The standard approach of text normalization commonly used by industries involves complex hand-written grammars to verbalize input tokens, such as Google's Kestrel TTS text normalization system [1]. The system works by classifying non-standard tokens into their respective semiotic classes (classification grammars) and taking their context to appropriately verbalize the token (verbalization grammars). The classification and verbalization grammars are then compiled into weighted finite-state transducers (WFSTs) which will be used to pass non-standard text into. Given that this method requires complex hand-written grammars, it is difficult to dynamically scale to multiple languages.

Recent advances in text normalization have led to the development of rapidly generated FST-based verbalizers for ASR and TTS systems through the efficient sourcing of language-specific data collected through questionnaires given to native speakers of a language [6]. This data contains all the necessary information to bootstrap the number grammar induction system to parameterize a verbalizer template, which will then be used by formal language experts to develop FST-based text normalization systems [7]. Our method takes in the idea of text normalization as a data collection problem and improves it further by providing a user-friendly interface in the form of a website and a feature which allows users (both the native speaker source

and the formal language expert) to quickly evaluate the provided information and the text normalization system for any obvious mistakes.

Within the last few years, deep learning has taken over the speech and language technology field. One recent work [2] decided to tackle the text normalization task as a supervised sequence to sequence deep learning task: given a large corpus of written text aligned to its normalized spoken form, train a recurrent neural network (RNN) to learn the correct normalization function. Although the authors did achieve very good results in terms of overall accuracy, they encounter some errors which are problematic as the resulting standardized text conveys the wrong message, thus making it useless when deployed in a speech application. They came up with a solution by employing a filter which is based on finite state transducers (FSTs). The FST-based filter can mitigate such problematic errors and achieve an even higher accuracy. The authors conclude that text normalization is not going to be something that can be solved merely by having huge amounts of annotated text data and feeding the data into a general RNN model.

Only recently, transformer models have been topping the charts in various tasks ranging over different fields, such as machine translation in natural language processing. Zhang et al. [3] suggested that we can treat the text normalization problem as a machine translation task, where the source language is raw text and the target language is normalized text, in the same language as the source language. Their best transformer model achieved an accuracy of 96.63%, however after qualitative evaluation they found that their model still suffers from unrecoverable errors which completely alters the meaning of the text. Additionally, transformers are notoriously known to only perform well when given a large amount of non-standard and standard text pairs, therefore this methodology (and any supervised learning-based methods) will not be transferable in the context of low-resource languages.

4. Methodology

4.1. Data collection

Our main goal is to obtain sufficient information from a native speaker in order to map non-standard numerical text to its standardized form, ideally by asking the minimum number of questions. We would also want the number or type of questions to be generalizable across various languages to cover multilingual mappings.

As mentioned in the previous section, there are a lot of nuances in multilingual text normalization which causes it to be a difficult task to scale across languages. After a careful evaluation of the different counting systems adopted by several (primarily) high resource languages, we observe that the nuances diminish as the number grows. Furthermore, the non-standard numerical texts that are commonly found in most corpora are usually smaller numbers. For instance, it is very rare to observe an 8-digit number with eight significant digits. From this observation, we decided that it is sufficient to collect information on the digits 1-100, as well as the numbers denoting the tens position, such as hundred, thousand, up to a billion.

Additionally, we have requested some additional information from the user regarding the additional symbols that might exist in numerical text existing in the different semiotic classes, for example, the decimal point (.), the negative and positive sign (-,+), comma (,), currency symbol and its normalized form ($ $\rightarrow$ dollar), as well as the percentage sign (%). We also gather information with regards to the chunking of the number. In the future, we hope to collect even more detailed information to cover the mapping of all semiotic classes, such as the different measurement units adopted by the different languages.

We have presented a clean and intuitive website built using the Streamlit framework [8] where users with limited technical proficiency will be able to input text normalization data into a table. The user will be able to pick from over 600 languages from a drop down menu. The list of languages corresponds to the number of unique voices available in the CMU Wilderness dataset [9]. Duplicate voices with the highest Mel Cepstral Distortion score for Random Forest Clustergen synthesizer (MCDR) [10] will be omitted from the list.

Then, the user will be able to download a template in the form of a comma-separated value (CSV) file containing the information we need to perform text normalization in their native language. After filling in the CSV file, they can upload the completed table back onto the website. Their completed table will be stored with a unique ID consisting of the language ID corresponding to the CMU Wilderness voice and the timestamp it was uploaded, allowing it to be used as a non-standard to standard mapping for text synthesizers.

The user will then be able to enter a numeric string such as whole numbers (“1235523”), phone numbers (“+4120987654”), currency (“$23”), floating-point numbers (“21.25”), time (“19:10”), and percentages (“90%”). The string will then be run through a text normalization algorithm which will output its normalized form. The normalized text will be processed by the Flite synthesizer [11] which utilizes the voice corresponding to the language from the CMU Wilderness dataset, generating a .wav file. The .wav file will be played out through an audio player widget on the website.

4.2. Text normalization algorithm

When the user uploads the CSV file containing information on the non-standard to standard text mapping for a particular language, the back-end will parse the information into a hashmap to be normalized using a Python script. At the moment, the algorithm is able to normalize text in the form of whole numbers, phone numbers, currency, floating-point numbers, time, and percentages. The algorithm takes into account how the numbers are chunked and processes each chunk accordingly. Additionally, the algorithm will also take into account the surrounding symbols to identify the semiotic class by querying for specific symbols such as decimal points, colons, and more. The algorithm is agnostic to the position of the symbol, as long as the symbol is attached to the numerical text (not separated by a white space).

For whole numbers, we will break down the numbers with respect to the chunking system adopted by the specified language. Each chunk will be processed individually and joined back together. For example, the number “123456” in English will be divided into “123” and “456”, where “123” will be normalized to “one hundred twenty three” and “456” will be normalized to “four hundred fifty six”. Then, the word “thousand” will be inserted when concatenating the two chunks.

For currencies, floating-point numbers and percentages, the non-standard text will be split depending on their position with respect to the decimal point. The numbers before the decimal point will be read as a whole number, while the numbers after the decimal point will be read out individually. Then, the normalized version of the currency or percentage symbol will be placed accordingly. For example, the amount “$2.52” will

be normalized in the English language to "two point five two dollars". Future iterations of the algorithm will enable the user to incorporate stylistic preferences specific to a semiotic class, such as reading out "$2.52" as "two dollars and fifty two cents" instead of "two point five two dollars".

Phone numbers will be normalized into individual numbers, similar to the numbers behind a decimal point. As mentioned in the previous sections, it is quite difficult to identify which text is an instance of a phone number, especially when symbols such as dashes between the numbers or the plus sign are omitted. For example, "911" is a valid phone number, and so is "1234567890". For now, we have decided that any numerical text containing more than 10 digits will be classified as a phone number and that each number will be read out individually. This solution, however, might not generalize to multiple languages as the length of phone numbers varies across different countries.

Finally, texts belonging to the time semiotic class are usually indicated by a colon in the middle of a string of digits. Since there are a lot of discrepancies in the way multiple languages interpret time, we decide to make the normalization time instances generalizable by omitting the AM/PM token and just reading out the numbers before and after the colon as whole numbers. For example, "20:50" will be read out as "twenty fifty" instead of "eight fifty PM".

4.3. Evaluation and feedback

As our text normalization system mainly adopts a rule-based methodology which is quite stringent and might not generalize well to languages with unique counting systems. To aid this, we provide a feedback collection system where a user will be able to input an erroneous normalization after the .wav file is generated and played on the website. After the user has input the correct normalization, they will be able to listen to the amended normalization and they will be able to submit it once they are satisfied. If the user believes that the error came from a mistake in the input table, they will be able to upload the amended table the same way they have done so previously. Given enough users and feedback from erroneous normalizations, this data will allow us to identify common mistakes that arise from the text normalization of specific languages and make the appropriate amendments to the algorithm or add new fields to collect more information on the CSV file.

5. Results

Based on our review of previous work and our proposed methodology, we will answer the research questions posed in Section 1.

5.1. RQ1: Which non-standard words appear in text?

We have found that the type of non-standard words which involve numerical characters can be grouped into different semiotic classes. The presence of a semiotic class highly depends on the corpus the text originates from. For example, a science textbook will have multiple instances of measure words such as "180kg" or "2000m", while a news article might contain date and time instances such as "20:30". Furthermore, there exists ambiguities between the existing semiotic classes, such as how "20:30" can represent an instance of time, but it can also indicate a verse from the Bible. Without taking into account the context of the text, the system will normalize the non-standard words into its most generic form. So, instead of spelling out "eight thirty PM", the normalized form will be "twenty thirty".

5.2. RQ2: How do we ask questions to get our answers?

Our proposed method relies heavily on user input and feedback, where a user will fill the required information in a table for the initial non-standard to standard text mapping, and then make amendments to erroneous normalizations, which will be used to improve the algorithm as well as to evaluate whether we would need to gather more information by adding fields to the table. The table will contain information such as the digit normalization from 1-100, common symbols such as currency and percentage, decimal point, number chunking and more. Given access to a native speaker of a low-resource language, this process is relatively easy and efficient. The evaluation process will involve primarily generating non-standard text from the various semiotic classes and calculating the number of correct and incorrect normalizations to obtain an accuracy score.

5.3. RQ3: How do we make it easy to answer such questions?

We present an accompanying website to our tool which was designed with intuitiveness in mind. Through the website, a native speaker with limited programming or technical skills will be able to input the non-standard to standard mapping of various words such as digits, currency, tens, mathematical symbols and more, by filling in a spreadsheet in the form of a comma-separated value (CSV) file. The back-end of the website will then automatically parse the spreadsheet, allowing the user to read and listen to the normalization of the non-standard text and provide any feedback or corrections.

5.4. RQ4: How do we address variability?

Given that our system hosts over 600 unique languages on a public server for public use, the data we are collecting is vulnerable to malicious or erroneous input. To prevent this from happening, we will store multiple tables from various languages and we will take the most common mapping for each item. This method will also solve the issue of dialectal variation that may arise from the variety of input collected from the native speakers. Taking the most common mapping for each item can also be used to resolve any text normalization ambiguities within a language.

5.5. RQ5: How do we extend this to a full front-end?

There are still plenty of non-standard text that may appear in a corpus which do not include numerical values. These could be alphabetical sequences which are unseen but pronounceable, letter sequences such as acronyms and organization names (e.g. CIA, WHO) whose letters are to be read individually, or shortened form/abbreviations which are commonly used (e.g. dept $\rightarrow$ department). In the future, we would like to extend our system to be able to capture these nuances from multiple languages using the same data collection process.

6. Conclusions

Given the variability of the textual representations of the semiotic classes in various languages, text normalization continue to prove to be an inherently difficult task. In this work, we propose a method to easily collect sufficient information through an intuitive user interface from native speakers, particularly native speakers of low-resource languages, to obtain a generalizable mapping from a non-standard text to its standardized form, allowing it to be pronounced correctly by a voice synthesizer.

7. References

[1] P. Ebden and R. Sproat, "The kestrel tts text normalization system," *Natural Language Engineering*, vol. 21, no. 3, pp. 333–353, 2015.

[2] R. Sproat and N. Jaitly, "Rnn approaches to text normalization: A challenge," *arXiv preprint arXiv:1611.00068*, 2016.

[3] H. Zhang, R. Sproat, A. H. Ng, F. Stahlberg, X. Peng, K. Gorman, and B. Roark, "Neural models of text normalization for speech applications," *Computational Linguistics*, vol. 45, no. 2, pp. 293–337, 2019.

[4] R. Sproat, A. W. Black, S. Chen, S. Kumar, M. Ostendorf, and C. Richards, "Normalization of non-standard words," *Computer speech & language*, vol. 15, no. 3, pp. 287–333, 2001.

[5] P. Taylor, *Text-to-speech synthesis*. Cambridge university press, 2009.

[6] S. Ritchie, E. Mahon, K. Heiligenstein, N. Bampounis, D. van Esch, C. Schallhart, J. Mortensen, and B. Brard, "Data-driven parametric text normalization: Rapidly scaling finite-state transduction verbalizers to new languages," in *Proceedings of the 1st Joint Workshop on Spoken Language Technologies for Under-resourced languages (SLTU) and Collaboration and Computing for Under-Resourced Languages (CCURL)*, 2020, pp. 218–225.

[7] S. Ritchie, R. Sproat, K. Gorman, D. van Esch, C. Schallhart, N. Bampounis, B. Brard, J. F. Mortensen, M. Holt, and E. Mahon, "Unified verbalization for speech recognition & synthesis across languages." in *INTERSPEECH*, 2019, pp. 3530–3534.

[8] P. Singh, *Machine Learning Deployment as a Web Service*. Berkeley, CA: Apress, 2021, pp. 67–90.

[9] A. W. Black, "Cmu wilderness multilingual speech dataset," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 5971–5975.

[10] A. W. Black and P. K. Muthukumar, "Random forests for statistical speech synthesis," in *INTERSPEECH*, 2015.

[11] A. W. Black and K. A. Lenzo, "Flite: a small fast run-time synthesis engine," in *4th ISCA Tutorial and Research Workshop (ITRW) on Speech Synthesis*, 2001.

Appendices

A. Web application

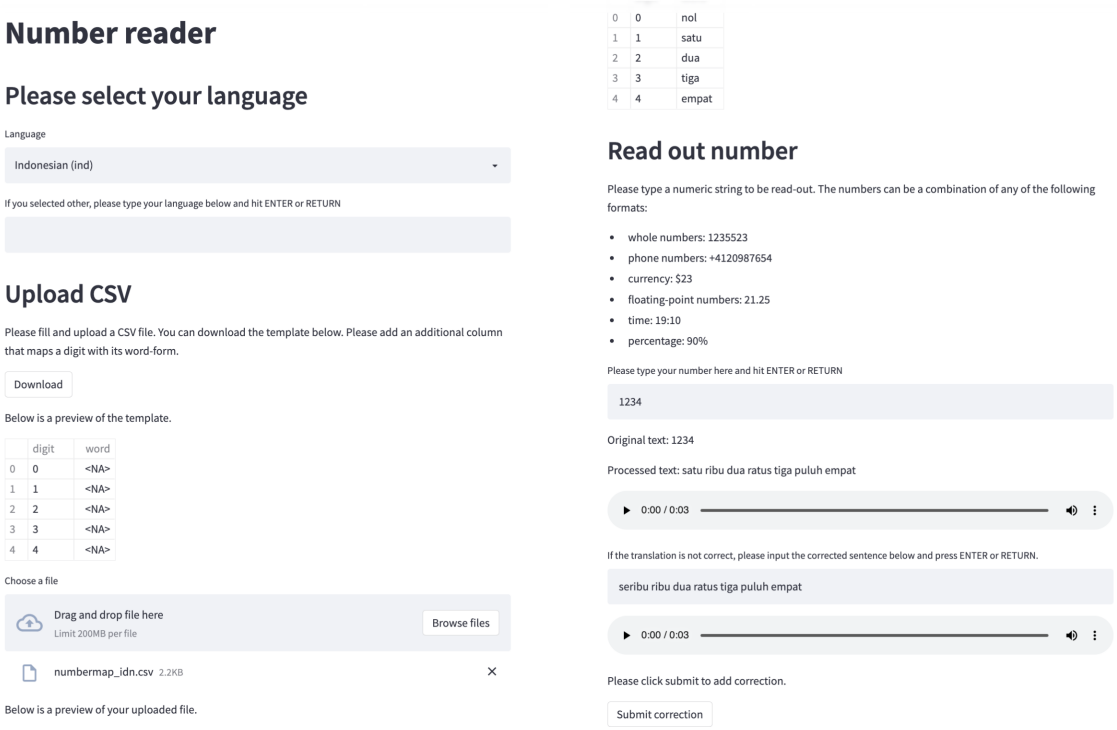

Figure 4: *Screenshot of web application used to collect text normalization data from native speakers and evaluate the existing system*