SymSpell and LSTM based Spell-Checkers for Tamil

Selvakumar Murugan, Tamil Arasan Bakthavatchalam, Malaikannan Sankarasubbu

Saama Technologies AI Research Lab

{selvakumar.murugan, t.arasan, malaikannan.sankarasubbu}@saama.com

Nov 15, 2020

Abstract

Spell checker is an invisible but indispensable component of a very large chunk of computer users' lives. It comes with most softwares that manipulate text like office suites that include from simple note-taking to fully versatile word processors and search engines. Spell checkers tokenize text and verify whether the text contains spelling and grammatical errors. Existing spell checkers, though work for most european languages, do not account for linguistic features of the Indian languages like Tamil. In addition, the issues with encoding representation of Tamil under unicode adds even more complexity on how the Tamil text should be handled for applications like spell checking. We implement and test three different spell checkers for Tamil namely *bloom-filter, symspell, LSTM* based spell checkers. *Symspell* is very fast for validation and lookup suggestions. *LSTM* implementation though not accurate enough for day to day use, is an interesting line of work that remains unexplored.

Introduction

Tamil is one of the ancient and classical languages of the Dravidian family which dates back to 3rd century B.C. Tamil is a very rich and complex language in terms of literature, dialects, rich set of graphemes and vocabulary enabled by linguistic features like agglutinative morphology and grammar. This richness comes with its costs. Representation of Tamil script in computers is riddled with issues for instance unicode encoding of Tamil does not correspond to its natural script and how it composes *uyir-mei* characters. This adds even more complexity on how the Tamil text should be handled for applications like spell checking.

Spelling and grammar checking is a significant component of most web applications and search engines. Spell checking is a subdomain of natural language processing (NLP). The definition of spell checking varies widely ranging from trivial token correction implemented by simple lookup tables to sophisticated analyses including formal/informal tone detection, sentiment analysis, suggesting semantically similar alternatives, paraphrase detection for plagiarism checking, all of them require intelligent algorithms.

Spell-checker predicts the word user intended to write down by using a combination of clever data structures and algorithms in conjunction with established dictionaries, for finding the correct spelling and suggesting similar words based on distance metrics such as Levenshtein edit distance.

There exist a wide variety of spell checkers such as ispell, aspell, hunspell[1] which work well for English and other European languages. Though there are plugins to support east asian languages like Tamil they do not capture the linguistic features of the language, which greatly influences the different types of errors. In this work, we take inspiration from spell checker implementations for European languages and implement three methods for Tamil. We describe the common source of errors for both English and Tamil, to contrast and illustrate the influence of intrinsic nature underneath each language.

Ideally a spell-checker should let the user write down thoughts and ideas without the constant cognitive load of verifying every word and every sentence. The tool must be non intrusive such that the process of translating thoughts into its textual form is not hindered by spelling and grammatical mistakes very much similar to how a paper notebook maintains its silence throughout.

Sources of Errors

Common types of mistakes include ***swapped characters:*** *sawp → swap;* ***character case:*** *LOWER → lower;* ***double occurrence:*** *twwo → two, thethe → the;* ***missing characters:*** *mising → missing.* There are many mechanisms from which the errors manifest, the following list is not exhaustive.

Language Independent Errors

Keyboard layouts: Muscle memory developed on a particular keyboard layout will hinder speed when switching to a new keyboard layout. In the case of multilingual documents, this manifests very vividly due to frequent switching between two or more input methods.

Homophones: Similar sounding alphabets are used. *அகளவிரித்து → அகலவிரித்து, அனுகப்பட → அணுகப்பட, ஆளுனரால் அனுப்பி → ஆளுநரால் அனுப்பி, காலத்திர்க்கானவை → காலத்திற்கானவை*

Language Dependent Errors

Input engine issues: some input systems follow the scheme: *uyirmei + pulli → mei* and some follow scheme: *mei + uyir → uyirmei*

Apart from mechanistic and phonetics sources of errors above, the grammatical features of the language can give rise to a suite of errors in writing. The fundamental units of languages and its text differ greatly and are influenced by cultural history.

Even the very fundamental unit of a language, the alphabet performs different functions in English and Tamil. Graphemes in English and the alphabet in Tamil are not exactly the same. The definition and expression of the unit *word* also differs wildly. Words in English are mostly delimited both visually and grammatically whitespace except in a few special cases for instance, *New York* is considered a single word.

Tamil is an agglutinative language in which the words can be formed by attaching suffixes to the root word, e.g: காலை > காலையில் > காலையிலிருந்து > காலையிலிருந்தே > காலையிலிருந்தேயவன்.

Agglutination have brings in alphabets that are not in the combined words which are a common source of error in writing Tamil, e.g: இணைத்து பார்த்தேன் → இணைத்துப் பார்த்தேன்; இணைந்துப் படிப்பதற்கும் → இணைந்து படிப்பதற்கும். Notice that depending upon the grammatical context, the alphabet ப் appears or vanishes.

Some are not strict grammatical rules but guidelines to elucidate a cleaner context e.g: கிராமப் புறங்களில் இருந்து → கிராமப்புறங்களிலிருந்து.

These different features of language and writing tools creates an exponentially large space of errors rendering spell checking a non-trivial problem. The following section describes the methods we implemented and tested.

Methods

In this work we implemented three different methods to realize spell checking for Tamil. The first two methods bloom-filter and symspell considers space delimited consecutive stream of alphabets as the single word unit. The following sub-section describes the dictionary corpus that forms the backbone of these two methods. The LSTM[2] based implementation however employs wordpiece[3] based tokenization using Byte Pair Encoding(BPE). This is to avoid the vocabulary explosion problem created by the agglutinative nature of Tamil. We chose BPE because of its versatility to adapt to the corpus instead of relying heavily on strict rules. On a character level, we do not directly use unicode code points but instead use individual alphabets that respect Tamil script. It is important to note here that there are other tokenization methods that more grounded on grammar like morphology based tokenization [4 soumya]

Corpus of Correct Words

We gathered collections of words from [5] and Tamil etymological dictionary[6]. The entire dictionary for the spellchecking was built by merging the collected corpora. The corpora are merged and the resulting dictionary acts as the source of truth for the bloom-filter and symspell algorithms.

Corpus	Count
all_tamil_nouns/all_nouns.txt	181185
tharavukkanam/tamil-etymological-dict	119754
merged	**249056**

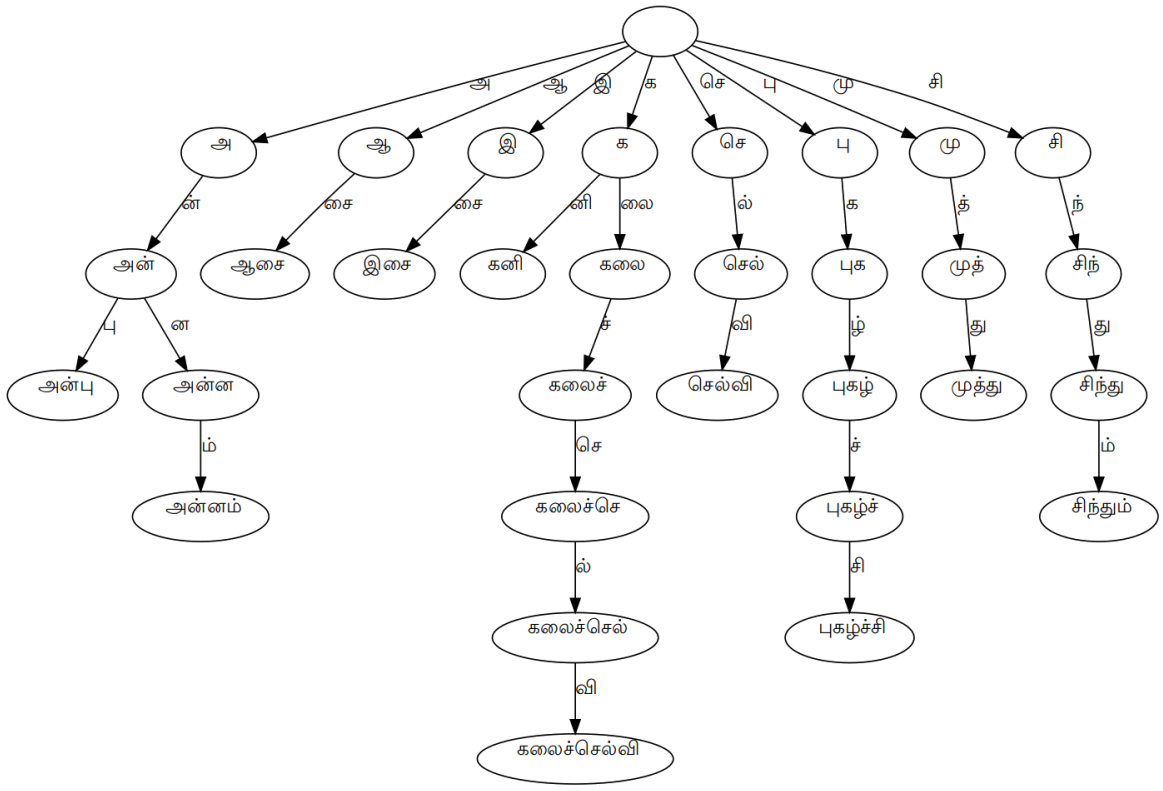

Fig-1: Trie built for the Tamil words: அன்பு , அன்னம் , ஆசை, இசை, கனி, கலைச்செல்வி, செல்வி, புகழ், புகழ்ச்சி, முத்து, சிந்து, சிந்தும். Notice that the செல்வி in கலைச்செல்வி is completely different from standalone செல்வி

Bloom Filter

Very minimal function of a spell checker is to flag from a stream of words which ones are incorrectly spelled. A simple lookup for the presence of the word in the dictionary is sufficient. But as the dictionary gets larger and larger we need an algorithm that can make this lookup faster. The bloom filter[7] is the simplest algorithm to implement that.

There are two issues with the bloom filter. First, the bloom filter can only check whether the word is correct or not. It does not have mechanisms to come up with a set of suggestions. Second, once a word is added to bloom-filter it cannot be removed.

Though the bloom filter can not be considered a spell checker as per widespread definition but it can very well act as a first line of validation in eliminating correctly spelled words, and let the more clever spell checking algorithms work only on incorrectly spelled words filtered out.

SymSpell

Most spell checking algorithms that employ a form of trie data structure for both validating whether a word is misspelled and also lookup suggestions, i,e closest correctly spelled words from the dictionary by a well defined metric. Usually the metric is Levenshtein distance. The trie is a tree data structure built by parsing the entire dictionary branching out as determined by the Levenshtein distance. *Fig-1* shows an example of the trie built for the words, *அன்பு , அன்னம் , ஆசை, இசை, கனி, கலைச்செல்வி, செல்வி, புகழ், புகழ்ச்சி, முத்து, சிந்து, சிந்தும்.*

Instead of looking from a dictionary of correct words to figure out the closest match for a misspelled word, SymSpell[8] algorithm flips the problem around. It builds a map of misspelled words to a probable list of correct words for up to a predetermined edit distance. This makes it extremely fast for lookup but requires more memory. In simple words, SymSpell precomputes probable misspellings of the entire dictionary beforehand to make the lookup much faster. *Fig-2* shows an instance of symspell edits dictionary with *edit distance = 2,* built for the same set of words from the trie example as in *Fig-1*

LSTM

We extended the intuition gained from the symspell approach to sentence level in addition to employing a machine learning method for sequence modelling called LSTM. We built a corpus of corrupted sentences and effectively transformed the spell checking problem into a translation problem. In order to satisfy the data requirement by LSTM, we exploited the news corpus *tamiltext-7M.txt*[9] built for language modelling. The sentences from the tamiltext-7M.txt corpus are corrupted on alphabet level at random positions to generate sentences with errors while retaining the original sentence as the ground truth. This corpus can also be exploited for building an extensive dictionary of surface forms of tamil lexicon, though we have not used it in such a fashion. *Fig-3* shows the sentence and word length distribution in the tamiltext-7M.txt corpus.

இச	இசை	செல்வ	செல்வி	முத்ை	முத்து
இசை	இசை	செல்வி	செல்வி	மு்து	முத்து
இை	இசை	செல்ி	செல்வி	லச்ச	கலைச்செல்வி
கன	கனி	செ்வி	செல்வி	ிந்து	சிந்து, சிந்தும்
கனி	கனி	சை	இசை	ைகழ்	புகழ்
கலச்ச	கலைச்செல்வி	னி	கனி	ைகழ்ச	புகழ்ச்சி
கலைசச	கலைச்செல்வி	பகழ்	புகழ்	ைத்து	முத்து
கலைச்	கலைச்செல்வி	பகழ்ச	புகழ்ச்சி	ெல்வி	செல்வி
கலைச்ச	கலைச்செல்வி	புகழ	புகழ்	அனனம்	அன்னம்
கலை்ச	கலைச்செல்வி	புகழச	புகழ்ச்சி	அனபு	அன்பு
கி	கனி	புகழ்	புகழ், புகழ்ச்சி	அன்னம	அன்னம்
கைச்ச	கலைச்செல்வி	புகழ்ச	புகழ்ச்சி	அன்னம்	அன்னம்
சந்து	சிந்து, சிந்தும்	புக்	புகழ்	அன்னம்	அன்னம்
சல்வி	செல்வி	புக்ச	புகழ்ச்சி	அன்ன்	அன்னம்
சிநது	சிந்து, சிந்தும்	புழ்	புகழ்	அன்ப	அன்பு
சிந்த	சிந்து, சிந்தும்	புழ்ச	புகழ்ச்சி	அன்பு	அன்பு
சிந்து	சிந்து, சிந்தும்	மத்து	முத்து	அன்ம்	அன்னம்
சிந்ை	சிந்து, சிந்தும்	முதது	முத்து	அன்ை	அன்பு
சிது	சிந்து, சிந்தும்	முத்த	முத்து	அ னம்	அன்னம்
செலவி	செல்வி	முத்து	முத்து	அ பு	அன்பு
ஆச	ஆசை	கலைச்ச	கலைச்செல்வி	ன்பு	அன்பு
ஆசை	ஆசை	சிந்து	சிந்து, சிந்தும்	புகழ்	புகழ்
ைஆ	ஆசை	செல்வி	செல்வி	புகழ்ச	புகழ்ச்சி
இசை	இசை	சை	ஆசை	முத்து	முத்து
கனி	கனி	ன்னம்	அன்னம்		

Fig-2: Sample edit lookup table built by symspell with edit distance = 2 for *அன்பு , அன்னம் , ஆசை, இசை, கனி, கலைச்செல்வி, செல்வி, புகழ், புகழ்ச்சி, முத்து, சிந்து, சிந்தும்.*

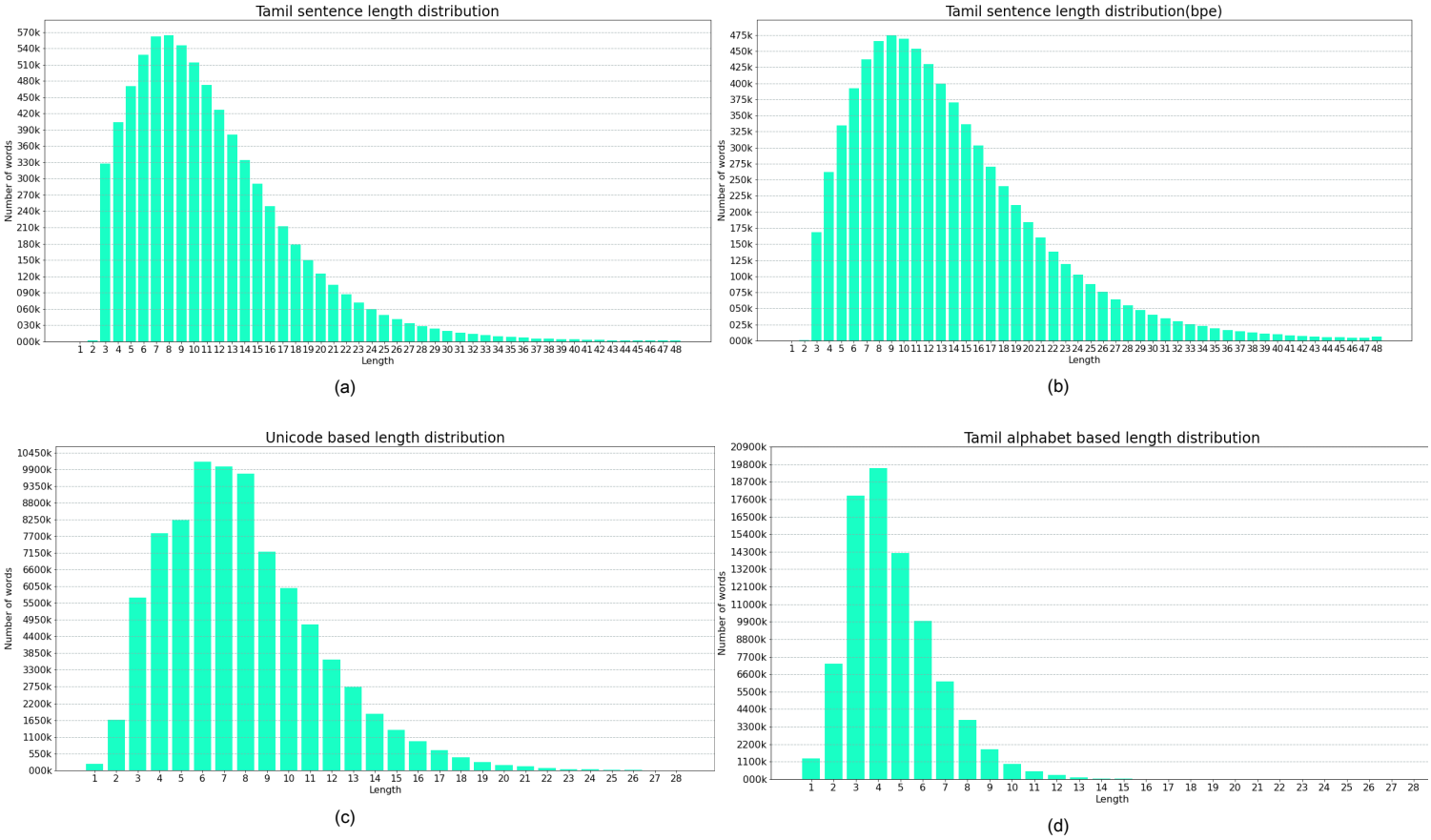

Fig-3: Distribution of sentence and word lengths in the corpus *tamiltext-7M.txt*. (a) The space delimited sentence length(number of words in a sentence) distribution. (b) BPE tokenized sentence length distribution. (c) word length distribution based on unicode encoding. (d) word length distribution based on natural Tamil alphabets.

-- Code Listing --

```
BuildCorruptedSamples(alphabets, dictionary, text, abberation_count):
  tokens = split_by_whitespace(text)
  for i in range(0, length(tokens)):
    if abberation_count:
      if random_float() > 0.5:
        abberation_count = aberration_count - 1
        random_index = random_integer(0, length(tokens[i]))
        tokens[i][random_index] = random_choice(alphabets)
```

```
    return join_with_whitespace(tokens)
```

Fig-4: pseudocode for generating parallel corpus for LSTM training

Training the character level LSTM with the parallel corpus of *corrupted* and *correctly* spelled sentences we achieve a score of 0.40 in exact match. Even though this score is too small to be useful in practical application, this line of algorithms remain unexplored for spell checking in Tamil.

The LSTM was trained with tamiltext-7M.txt dataset with train test *split ratio = 0.8* with *embedding_dim = 200, hidden_dim = 50* for *1000 epochs.* The optimizer used was vanilla SGD with l*earning_rate = 0.001 and momentum = 0. 1*

Discussion

We discussed the implementation of three different spell checking algorithms for Tamil language. Bloom-filter is very fast but offers limited usability. It cannot offer suggestions to the misspelled words. Symspell is also very fast at both flagging misspelled words and generating suggestions for the word under scrutiny. However it requires a large amount of memory and is not dynamic, i.e the symspell can not suggest new words that are in edit greater than what it was generated with. The memory requirement grows geometrically with edit distance. LSTM implementation is an interesting approach that at present, is not ready for real-time use. Though newer neural network architectures like transformers[10] can be used, it is still an open question whether the benefits will outweigh the computational complexity that they demand.

Runtime environment for the spell checker also varies wildly from end user devices like mobiles phones, laptops to services that reside in servers over the cloud. Even mobile devices come with a wide range of memory and cpu compute capacity. The spell checker implementations need to be stripped down or can be beefed up depending on where they are deployed. Symspell can be tweaked to run on mobile devices with limited memory with a wide range of suggestions at the risk of slowing down other system services. LSTM based spell checking in mobile platforms is considerably slow and requires more engineering on model tweaking, but is well suited to run on cloud environments.

It is important to mention that it is useful to have the spelling and grammar checking tools explain why it came with the suggestion in grammatical terms. Existing spell-checkers do this by building a set of rules with patterns of errors and their corrections, described mainly in XML format by linguistic experts. However machine learning methods like LSTM can learn these

rules implicitly but do not produce naturally interpretable output. On the other hand combining machine learning methods and rule based systems that can extract rules from what the LSTM has learned, we bypass the tedious initial bootstrapping of hand crafted rules. The machine learning system can be broken down into specific components like named entity recognizers and part of speech taggers and their outputs can be used to inform each other in an incremental fashion for practical use.

Conclusion

Spelling and grammar checking is effectively an AGI problem and so even though it is extremely complicated to create a completely automated spell checker, it is rewarding to attempt at making one. The existing spell checkers employ a suit of different heuristics to perform the function, under different environments. Tools like Grammarly work extremely well for English. Its existing infrastructure provides itself with access to a huge amount of data to learn from and improve its algorithms everyday. Though this work is merely a baby step in that direction, we look forward to developing such a platform for Tamil with the collective effort from the Tamil speaking community.

References

1. László Németh, Kevin Hendricks, *https://github.com/hunspell/hunspell*
2. Sepp Hochreiter and Jürgen Schmidhuber, *Long Short-Term memory*. Neural computation, 1997.
3. Yonghui Wu, et al., *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*, 2016
4. Sowmya S Sundaram, *Morphological Processing for தமிழ் — The Unsupervised Way,* https://medium.com/analytics-vidhya/morphological-processing-for-தமிழ்-the-unsupervised-way-68afebc388c4
5. T. Shrinivasan, et al., *all_nouns.txt*, https://github.com/KaniyamFoundation/all_tamil_nouns
6. *Tamil Etymological Dictionary*, https://github.com/vanangamudi/tharavukkanam
7. Malaikannan Sankarasubbu, *Bloom-filter*, https://github.com/malaikannan/TamilSpellChecker
8. Wolf Garbe, *Python implementation of SymSpell algorith*m, https://github.com/mammothb/symspellpy
9. Selvakumar Murugan, *tamiltext-7M.txt*, https://github.com/vanangamudi/tharavukkanam/tree/master/tamil-etymological-dict
10. Ashish Vaswani, et al., *Attention Is All You Need*, https://arxiv.org/abs/1706.03762