Vietnamese full text recovery from diacritic-lacking documents with abbreviations

Tuan Do
Computer Science Department
Brandeis University
tuandn@brandeis.edu

Abstract

In the paper, the author addressed some issues related to vernacular Vietnamese used for social network commenting and personal text messaging. The paper proposed a solution to recover the full text form and experimented the system over online news text.

I. Introduction

Vietnamese orthography has officially adopted Latin scripts at the beginning of 20's century, sharing some similarity with Portuguese alphabets. However, one notable feature of Vietnamese alphabets is the employment of a set of diacritics to serve the following two purposes:

- To modify the original Latin characters, thereby creating 7 Vietnamese characters that have similar *sound* to the original Latin characters. The list of modified characters are đ (similar sound to *d* in English, while d is similar to *z* in English), ă, â (similar to *a*), ê (similar to *e*), ô, ơ (similar to *o*), and ư (similar to *u*). They will be called decorating diacritics in the remaining of the paper.
- To add tones to syllables, there are 5 diacritics used for tones: à, á, ả, ã, ạ in addtion to the default non-diacritic tone. They will be called tone diacritics in the remaining of the paper.

With the introduction of numerous typing support softwares such as Vietkey, it is easy to input Vietnamese text in computer. However, the rapid emergence of quick commenting and status updating from smart phones on personal pages in social network such as Facebook and Twitter has created a large amount of online text that does not have or severely lacks diacritics. Any effort to translate these texts into any other languages have to deal with this problem.

Another application that requires recovering text from non-diacritic forms might involve personal communication between people of different generations. While the older generation is only familiar with the standard text form, the younger generation display an inclination toward using non-diacritic text message. That actually happens a lots in message texting between me and my parents, as they always try to type in full form messages, and I have not even installed any Vietnamese input software on my cellphone.

The extensive use of diacritics in Vietnamese complicates translation of non-diacritic text. A simple calculation on a non-diacritic form *dong* give 36 different syllables and half of them are acceptable syllables (syllables that appears in some words). Other non-diacritic form might have more than that number of normal forms.

One more issue related to online text and personal message recovery is the employment of abbreviations to shorten the typing time. That phenomenon is ubiquitous in the young generation, considered a fashion and one not adapting to it is called *prehistory caveman*

There are some simple rules to abbreviate common words in personal messages, for example:

- To keep only the initial consonant and the ending consonant, while removing the vowels of functional words, including common adverbs: *đang, đương* (currently) − > *dg*, *được* (functional word, has passive effect) − > *dc*.
- To keep only the initial consonant (initialism), especially for pronouns: *tôi, tớ, tao* (I) − > *t*, *anh* (I or you) − > *a*, *em* (I or you)− > *e*.
- Combination of the two previously mentioned rules: *hôm trước* (some days ago) − > *htrc*.

One approach to solve the problem is to have the machine translation system model some non-diacritic versions of a word just like separate words. For example, a system based on IBM models could estimate the probability of both P(đang|currently) and P(dang|currently) based on the parallel corpus. The weakness of this approach is that P(dang|currently) implicitly model P(non-diacritic text|normal text). It is not really straight forward how to estimate the ratio of non-diacritic text over normal text, given that most parallel corpus are built on normal text.

Another approach is explicitly model the translation from non-diacritic text to normal text. It is not necessary to know P(non-diacritic text|normal text). Instead, one needs to know the distribution of syllables having the same non-diacritic form. This probability could be well estimated by just using an unigram model.

Currently, Google translate has a system of suggestions that could help recovery of non-diacritic text in some cases, though they do not have suggestion for abbreviations.

II. System details

II.1 Diacritic handling

This is the module dealing with diacritic removal to simulate data for testing and to recover diacritic on non-diacritic forms. The former part just requires a map between unicode character and ASCII character.

The recovering part, strictly speaking, is to recover from a non-diacritic form to all acceptable full forms. The tone marker should be placed only on top of the main vowel, and other vowels in the same syllable should not bear any tone diacritics, however character decorating diacritics could be used.

The process selects the vowel cluster in the syllable, and try to add all available decorating marker on it. All created forms will be compared with a list of diphthong and tripthong that I made myself (they should be better called 2-vowel clusters and 3-vowel clusters and I'm actually do not know any literature treatment of diphthong and tripthong in Vietnamese). Unacceptable will be discarded. Based on the same list, the main syllable is selected. Some of diphthongs have the main vowel at the end (and glide at the beginning) and some have the main vowel at the beginning. Similarly, all tripthongs has a glide at the beginning, and the position of the main vowel depends on whether the second vowel is glide or not. After that, all tone diacritics are put on the top of the main syllable. Note that all generated forms are not strictly acceptable forms in some cases, and I will come back to that later in Remaining issues part.

II.2 Language model, abbreviation and training

Hidden Markov Model (HMM) is a highly flexible and powerful method for text tagging. It is a generative model that has a chain of hidden (latent) states and their corresponding observable states. The hidden states are generated on a chaining-based by a transition probability $P(hidden\_state_i|hidden\_state_{i-1})$, therefore state at position i only depends on the state previously generated. Each observed state is generated from its corresponding hidden state by an emission probability.

In a similar manner, for this particular problem, the hidden states are the full form of words and the observed states are the non-diacritic form. Because word segmentation is incorporated into the system, the hidden state will also reflect the word boundary, using the B (Begin), I (Inside) tags.

Strictly speaking, Hidden markov model requires input to be the observed data, two probabilities will be approximated on training step. However, because the data used in the project is simulated, the emission probability is assumed to be known beforehand. Without abbreviation, the emission probability is always 1. Token abbreviation requires the user of the system to input an abbreviation files, specifying the emission probability $P(abbreviation\_form|full\_form)$, the probability residue is assigned to the default non-diacritic one.

Therefore, the system is trained in a similar manner as training a language model. That includes counting of unigrams and bigrams, and calculating a transition probability based on linear combination of them.

II.3 Decoding

Decoding employ Viterbi algorithm. Because the algorithms is straight forward, I will just briefly mention how I use Viterbi algorithm in the project. Token in the sentence is traversed, for each one generating a list of candidate diacritic forms. For each possible candidate, the maximum probability of the hidden sentence from the beginning to the current token is calculated dynamically based on stored values from previous step. An array of tracing pointers is also stored. When traversal is finished, the most likely hidden form is found by tracing back the pointers.

II.4 Evaluation

Three different evaluation metric is calculated. The overall precision is calculated based on how many token are exactly recovered both on segment tag and form, the segment precision only on segment tag, and recovering precision only on the form of token.

$$\begin{aligned} P\_overall &= \frac{Number\_of\_exact\_form\_and\_tag\_token}{Number\_of\_token} \\ P\_tag &= \frac{Number\_of\_exact\_tag\_token}{Number\_of\_token} \\ P\_form &= \frac{Number\_of\_exact\_form\_token}{Number\_of\_token} \end{aligned}$$

III. Data preparation

III.1 Main data

Three sources of data is considered at the first stage of the project: *vnexpress.net*, *vietnamnet.vn* and *vi.wikipedia.org*. Acknowledging that text in the last source might be too formal and has far different distribution with any target data of the project, the two other sites are comparable regarding the popularity among Vietnamese. Using of language in both sites shows a relatively casual register, but including data from both sites might be extraneous as they have pretty the same content everyday. The selection of *vnexpress.net* is just a personal choice.

The crawled corpus has 6000 articles, over 4 million syllables. 5300 articles are used for training, and the other 700 articles for testing (I just run test with one coefficient set for language model so I do not need a developing corpus). The testing data is stripped of segmentation marker [and] and diacritics.

III.2 Auxiliary data

I made some auxiliary files that are pertinent to Vietnamese, including a mapping between Vietnamese character with diacritics and non-diacritic character, some files that have all diphthongs and tripthongs, and one abbreviation file that has some of the most frequently contracted forms in real text.

IV. Results

Metric	Value
Overall precision	80%
Recovering precision	85%
Segmentation precision	90%

Table 1: *Precision*

V. Remaining issues

One of the biggest issue is the difference in the distribution of the crawled corpus and the target text to be diacritic-recovered. As discussed previously, data that is in Vietnamese and most accessible is news published online. Although not as formal as literature and scientific documents, the language used in news and articles are far more formal than the language that are generally found in social networks or personal text messages. There is a difficulty for word segmentation as the former employs more Sino-Vietnamese words (words that origins from Chinese), that are generally two and more syllables, while vernacular Vietnamese is highly isolating. Therefore, the distribution of word boundary in news documents and social network texts are very different.

One issue of dealing with real text extracted in casual context is the high employment of borrowed words from English. In vernacular text message and social network update/comments, it is not hard to find sentence that has more than half the tokens are actually English words. The use of a mixed Vietnamese - English language is perhaps the result of Vietnamese increasing exposure to online resources. Or perhaps, Vietnamese language lacks a productive ways to create words that have the same meaning as English words (one productive way is to use Sino-Vietnamese combination but that is generally deemed too pedagogic and ancient). Dealing with that kind of real text requires a mixed language model.

Although it is shown that the model could be extended for syllable contraction and abbreviation, one remaining issue that I hoped to solve at the beginning of the project is to build a model that is tolerant to abbreviations at a word level (with at least two syllables). To incorporate words abbreviation, the models might need to be modified. Moreover, there is necessity to look at how frequently and systematically people employ this method in real text.

Dealing with shortening form of words requires certain level of familiarity with their usage in real text. The lack of a real corpus that one can look at the way people abbreviate limit the extension of the project. In a similar manner as using of abbreviation and contraction in other languages for speed and convenience, shortening forms of words are highly unpredictable and idiosyncratic. Although mostly understood pragmatically, it is not easy to incorporate them into any language model. Therefore, to extend the project it might need a study only about the use of abbreviation in Vietnamese.

One small issue risen in diacritic processing is the lack of a standardized system in diacritic marking. For example, a syllable *hòe* could has another form *hoè*, and although the former form is the default form of Vietnamese keyboard software (Unikey), the later form is deemed to be more grammatical correct, because the the second vowel is the main vowel. There is also documents that have otherworldly way to mark diacritics, such as marking diacritics over consonants. That could be result from using of different or broken keyboard software. I have tried to included those text into Latex without success, but anyway dealing with them is hard. Removing diacritic models could not

There is also a lots of trouble to generate all the diacritic forms of a non-diacritic syllable. Though it sounds to be straight forward, lacking a dictionary of all possible form of tokens in Vietnamese, it is not easy at all. If the system naively ignore the problem and generates all the possible diacritic forms, including one unacceptable, from a non-diacritic from, the number of diacritic forms could be as large as 432. That could make the system run slower for decoding using Viterbi algorithm.

Although I have explained the way I dealt with recovering diacritic forms from non-diacritic form, it doesn't solve all the cases. Some of the diphthongs requires a final consonants to realized because they are actually homograph of another diphthong, example is *iê* requires one of ending codas *m, n, t, p, c* to realize. One more issue is word that has stop ending codas *t, p, c, ch* could only take two out of 6 tones. These issues need to be resolved in a more expressive ways, perhaps by listing all possible syllables or finals of a syllable.

There are also issues with text processing tools used in the project. Nltk library is the most frequently-used library for text processing and it works fine for English. Trying to use Nltk for Vietnamese and other languages required some *tweak* and training on that language. For example, the default Nltk sentence tokenizer Punkt could load in a language model, but there is no available Vietnamese language model as far as I was looking on the Internet. The Vietnamese segmentation tool is outdated and behave in a very unstable manner, implying that the tool might suffer from severe implementation problem. Therefore, the precision as reported only reflected how precisely the system works in corresponding to the training data. However, it is expected that with a better segmentation tool for generating training data, or with annotated data, the overall performance of the system could be higher.

VI. Future consideration

One possible extension of the system is perhaps to try getting some data from social network. One available resource could be Twitter, as far as I know, it has an API that could help developers to crawl data newly posted each day. Another resource that have more informal text and have more similar distribution to personal messages and comments are forum posts.

Tokenization over the initial corpus need to be improved if the system is applied in reality. A better tokenizer need to be trained, maybe reutilize the corpus we already obtained with some human correction.

References

[Nguyen and Phan, 2007] Cam-Tu Nguyen and Xuan-Hieu Phan (2007). "JVnSegmenter: A Java-based Vietnamese Word Segmentation Tool", http://jvnsegmenter.sourceforge.net/, 2007.