Weighted and unweighted transducers for tweet normalization

Mans Hulden
University of Helsinki
mans.hulden@helsinki.fi

Jerid Francom
Wake Forest University
francojc@wfu.edu

Resumen: En este artículo se presentan dos estrategias basadas en transductores de estados finitos para la normalización de tweets. La primera de ellas se basa en reglas de corrección creadas manualmente y diseñadas para capturar las erratas y abreviaturas utilizadas más comúnmente, mientras que la segunda intenta construir automáticamente un modelo de errores a partir de un corpus etiquetado (gold standard) de tweets previamente normalizados.
Palabras clave: Normalización de tweets, tranductores de estados finitos, reglas fonológicas, canal con ruido.

Abstract: We present two simple finite-state transducer based strategies for tweet normalization. One relies on hand-written correction rules designed to capture commonly occurring misspellings and abbreviations, while the other tries to automatically induce an error model from a gold standard corpus of normalized tweets.
Keywords: Tweet normalization, finite-state transducers, phonological rules, noisy channel

1 Introduction

We present two different finite-state transducer (FST) based strategies for correction and normalization of tweets. While both methods use FSTs in the recovery of normalized forms of words, the two employ different strategies for normalization. The first method is entirely rule-based; that is, the goal is that a linguist provides rules for correction and normalization of tweets based on observations of commonly recurring patterns of abbreviations or misspellings. This is accomplished by compiling correction rules into unweighted FSTs in a hierarchical pattern where simple corrections are attempted before more complex ones. The second method attempts to induce the normalization patterns from development data directly by assuming a noisy channel model. Here, an error model of character-to-character transformations is learned from the data and, combined with a language model also represented as a transducer, the most probable correction can be calculated.

2 Tools

We have employed two different toolkits in our correction strategy. The unweighted transducers for manually designed correction rules were compiled using the *foma*-toolkit[1] (Hulden, 2009). For manipulating the weighted transducers employed in the noisy channel model approach, the *Kleene*[2] (Beesley, 2009) finite-state programming language was used. Both approaches rely on a dictionary taken from the FreeLing suite (Carreras et al., 2004).

3 Correction rules as FSTs

The goal with handwritten FST correction rules is to attempt a battery of corrections to unknown input words to transform them into normalized forms found in a dictionary. The corrections are expressed as contextually conditioned phonological replacement rules, converted into transducers, and combined in a hierarchical fashion in conjunction with a dictionary. They reflect phenomena such as:

[1] http://foma.googlecode.com
[2] http://kleene-lang.org

- diacritic restoration (a → á, ny → ñ, . . .)
- commonly occurring spelling errors (v → b, d → ∅, . . .)
- repetition removal (aaa. . . → a)
- common abbreviations/errors (ke → qué/que, tqm/tkm → te_quiero_mucho)

The correction rules themselves range from the very specific to highly generic. For example, a simple rule to attempt to restore missing **d**-characters in past participles looks in the *foma* notation as follows:

```
define RULEPart [..] (->) d || [a|i] _ o (s) .#. ;
```

reflecting the idea that **d**-characters are to be inserted when preceded by **a** or **i**, and followed by **o**, and optionally **s** at the end of a word.

At the same time, a very specific rule that only addresses the high-frequency word-form **Twitter** and its various misspellings and alternative forms appears as

```
define RuleTW1 [[t|T]+ [w|u]+ Vow+ t+ Vow+/h (r)]:
               {Twitter};
```

in effect accepting such forms as **Tuiter**, **tuitah**, **twittr**, etc., and mapping all these to **Twitter**.

In total, about 20 rules were designed. Some rules count as single rules in the combination hierarchy: for example, diacritic restoration rules (a → á, e → é, etc.) are considered a single rule for the purposes of combination.

The rules themselves could be contextually conditioned outside the current word as well. This may be useful for performing corrections based on syntactic context. However, for the experiments conducted here, all rules refer only to contexts within the same word.

The rules in question are combined in a hierarchical order using a type of if-then-else logic. That is, we can, for example, first apply corrections to (1) known abbreviations. If that fails to produce a legitimate word in the dictionary, we (2) apply diacritic restoration. If that fails, we apply some common error pattern (3). Should that fail, apply both (2) and (3), etc. etc. This type of a hierarchical replacement strategy can be implemented through standard regular expression operators available in conjunction with a dictionary encoded as an automaton (Beesley and Karttunen, 2003).

4 Noisy channel FST modeling

The fundamental idea behind noisy channel modeling is to choose a w to maximize the probability $P(w|s)$, where s is the "scrambled" word and w the normalized, intended form. This quantity is proportional to $P(s|w)P(w)$, where the first factor is called the error model—the probability that a word w was perturbed to yield s—and the second factor the language model, the probability that w occurs.

Weighted FSTs provide a convenient calculus for addressing the above task, assuming we can encode the probabilities of words $P(w)$ and perturbations $P(s|w)$ as probabilistic or weighted transducers somehow. When the error model (EM), the language model (LM), and a scrambled input word s are available to us as weighted transducers, we may calculate the composition

$$W = s \circ EM \circ LM \tag{1}$$

and subsequently find the least-cost string in the output projection of W. Here, the transducer s is simply an acceptor that repeats the to-be-corrected input word with probability 1 (or cost 0, as we are operating with negative log probabilities, or weights). Since transducer composition, projection, and least-cost paths are easily calculated, what remains to be done is to construct the transducers that assign a cost (probability) to character perturbations (EM) and words (LM).

4.1 Inducing an error model

To assign probabilities to perturbations, we first align the individual characters of word-pairs in the development corpus using a Chinese Restaurant Process (CRP) (Ferguson, 1973) aligner. This is very similar to aligning the word-pairs by minimum edit distance (MED). After the alignment is performed, we work under the assumption that the character-to-character correspondences

are all independent—that is, not conditioned upon other transformations in the same word or elsewhere—and estimate the probability of each input symbol x being transformed to y as:

$$p(x \rightarrow y) = \frac{c(x:y) + \alpha}{N + |\Gamma|\alpha} \quad (2)$$

using the smoothing parameter α (set to 0.1) to avoid zero counts, N being the number of observed character pairs, and Γ our alphabet of possible symbol pairs. Note that zeroes, representing deletions and insertions are also possible.

For example, our aligner outputs alignments such as:

```
pasa_o    t__o
pasado    todo

Tambien   Adioos
También   Adió_s
```

From this, we may calculate probabilities of inserting an **o** (`_:o`), inserting a **d**, (`_:d`), repeating a **p** (`p:p`), etc., and construct a transducer representing the error model (EM) that performs such perturbations with the intended probabilities. Note that while we assume a context-independent probability for changes, there is no principled reason why complex contexts could not be accommodated as well.[3]

4.2 Calculating corrections

For the language model, LM, we have chosen to rely mainly on simple unigram frequencies for words in the FreeLing lexicon, where frequencies were collected from a Wikipedia dump. Such a model is particularly simple to produce a transducer from, but again, there is no principled reason why more complex models could not be used.

Assuming the existence of transducers labeled `$word`, representing the current word to be corrected, an error model `$EM`, and a language model `$LM`, as described above, the correction can be described as a function in the Kleene programming language as follows:

```
$^correct($word) {
  return $^shortestPath(
    $^lowerside($word _o_ $EM _o_ $LM)
  );
}
```

This reflects exactly the calculation given in (1) and the entire function is equivalent to finding argmax $P(s|w)$ $P(w)$ in the noisy channel model presented above. That is, we compose the input word with the error model and language model, extract the output projection and find the least-cost (most probable) path.

When this strategy is employed, we also need to establish a cutoff probability to choose when to simply accept a word as is. Obviously, the error model allows for any word to change into any other word, albeit at a low probability, and we do not want to allow extreme changes to produce a word in the dictionary, but rather accept the unknown word without change.

Apart from calculating corrections based on a language model and error model, we can also incorporate previously "known" corrections collected from a gold-standard set of corrected words, if such a resource is available. If we assume access to a set of already known word-word pairs that represent corrected input words, we can build another transducer that directly maps such cases to their correct form at a low cost. This can be helpful because, for example, abbreviations such as *tqm* → *te_quiero_mucho* would likely produce a high cost using the character-to-character based noisy channel corrector because of the large number of character changes required to produce the target form. However, this can be addressed by building a transducer `EX` that models such previously known word-word pairs and assigns these corrections a low cost, and the entire calculation can be performed as:

$$W = s \circ ((EM \circ LM) \cup EX) \quad (3)$$

This would then model both a lookup from a "known" list of cheap corrections as well as the noisy-channel error-model and language model corrections.

[3] For example, inserting a **d** to yield a past participle, as in **pasao**→**pasado** should reasonably be more likely than inserting a **d** in other contexts.

System	Accuracy
Rules	63.1
Noisy Channel	61.4

Table 1: Results from development set.

System	Accuracy
Rules	60.4

Table 2: Results from test set.

5 Evaluation

To evaluate the performance of the methods, we divided the gold standard corpus into five parts to be able to perform cross-validation. This is necessary, in particular, for the noisy channel model where the learning data must necessarily come from a subset of the gold corpus with normalized tweets.

The results are given in tables 1 and 2. As is seen, the manual correction rules slightly outperform the corrections produced by the weighted transducer method.

6 Discussion & Future Work

We have compared and evaluated two different approaches to tweet normalization, one being more of an expert system developed based on observation of tweets and the other a statistical one based on inducing correction rules from a gold standard corpus. As it stands, the noisy channel corrector suffers from the difficulty of constructing a reliable error model from the limited number of gold-standard corrections and the fact that the unigram language model is relatively poor. This sometimes leads to drastically incorrect conclusions. For example, the corrector would produce from the input **cansao** the output **casa** and not the desired **cansado**, since in a unigram context, the high likelihood of **casa** outweighs the cost of having to delete two characters. Such errors are frequent and could probably be addressed by both a richer language model and a more refined error model.

Apart from obvious improvements to the language model, there is also potential for combining the hand-written and data-driven approaches to yield a hybrid system.

One perhaps profitable avenue of further experimentation is to use the contextual rules developed by the linguist and learn probabilities for those rules. For example, under the current noisy channel model, the perturbation $\emptyset \rightarrow$ u (insert **u**) is learned without reference to context and deemed equally likely regardless of context. However, the hand-written rule allows that transformation primarily following a **q**, reflecting errors such as ***qiero** and ***aqi**.

Quick improvements also appear possible by investing in a good dictionary of named entities as tweets appear to contain proper names with much higher frequency than in running text in general. For the current work, entities such as movie names, companies, celebrities, etc. were not used in the development. Automatic induction of such a database from large numbers of tweets should also be considered.

References

Beesley, Kenneth R. 2009. The Kleene language for weighted finite-state programming. In *Finite-state Methods and Natural Language Processing: Post-proceedings of the 7th International Workshop FSMNLP; Edited by Jakub Piskorski, Bruce Watson and Anssi Yli-Jyrä*, volume 191, page 27. IOS Press.

Beesley, Kenneth R. and Lauri Karttunen. 2003. *Finite State Morphology.* CSLI Publications, Stanford, CA.

Carreras, Xavier, Isaac Chao, Lluis Padró, and Muntsa Padró. 2004. FreeLing: An open-source suite of language analyzers. In *LREC Proceedings.*

Ferguson, T. S. 1973. A Bayesian analysis of some nonparametric problems. *Ann. Stat.*, 1:209–230.

Hulden, Mans. 2009. Foma: a finite-state compiler and library. In *Proceedings of the 12th conference of the European Chapter of the Association for Computational Linguistics,*, pages 29–32. Association for Computational Linguistics.