# Resource-based lexical approach to TWEET-NORM task

## *Aproximación léxica basada en recursos para la tarea TWEET-NORM*

**Juan M. Cotelo Moya**
Universidad de Sevilla
Avda. Reina Mercedes s/n.
41012 Sevilla
jcotelo@us.es

**Fermín L. Cruz**
Universidad de Sevilla
Avda. Reina Mercedes s/n.
41012 Sevilla
fcruz@us.es

**Jose A. Troyano**
Universidad de Sevilla
Avda. Reina Mercedes s/n.
41012 Sevilla
troyano@us.es

**Abstract:** This paper proposes a resource-based lexical approach for addressing the TWEET-NORM task. The proposed system exposes a simple but extensible modular architecture in which each analysis module independently proposes correction candidates for each OOV word. Each one of these analysis modules tries to address a specific problem and each one works in a very different way. The resources are used as the main component for the OOV detection system and they works as support for the validation and filtering of candidates.
**Keywords:** Twitter, resources, modular architecture, candidates

**Resumen:** Este artículo propone una aproximación léxica basada en recursos para abordar la tarea TWEET-NORM. El sistema presenta una arquitectura modular sencilla pero extensible en la cual cada módulo de análisis propone candidatos para cada palabra OOV de forma independiente. Cada uno de estos módulos de análisis intenta abordar una problemática específica y cada uno opera de forma muy distinta. Los recursos se usan como base fundamental del sistema de detección de OOVs y como apoyo para la validación y filtrado de candidatos.
**Palabras clave:** Twitter, recursos, arquitectura modular, candidatos

## *1 Introduction and objectives*

One of the most important challenges facing us today is how to process and analyze the large amount of information on the Internet, and especially social networking sites like Twitter, where millions of people daily express ideas and opinions on any topic of interest. These texts, called tweets, are characterized by having a short length (140 characters) that is too small compared with the size of traditional genres.

Consequently, users of these networks have developed a new form of expression that includes SMS-style abbreviations, lexical variants, letters repetitions, use of emoticons, etc. The result is that current NLP tools can have problems to process and understand these short and noisy texts unless they are normalized first.

The TWEET-NORM lexical normalization task proposes the automatic "cleansing" of a set amount of tweets by identifying and normalizing, abbreviations, words with repeated letters, and generally any out of the vocabulary (OOV) words, regardless of syntactic or stylistic variants.

Before doing any normalization process, we previously did a characterization of the existing phenomena, being easier trying to address the underlying causes of OOVs. To perform this characterization we used a previously collected dataset, composed of 3.1 millions of tweets related with the *2012 UEFA European Football Championship*.

The table 1 shows that characterization and provides examples for each phenomenon.The figure 1 also shows the phenomena ratio in a clearer way. It is observed that most of errors fit into 5 major categories and most of errors are associated with the fast and informal writing in Twitter, usually done from a mobile device.

The categories proposed for this task are coarser than our characterization. *Orthographic errors*, *Texting language* and *Character reptitions* fit into the *Variation* category, *Free Inflections* and correct words fit into the *Correct* category. *Other Language* and *Ascii Art* would fit into *NS/NC* category.

The system proposed in this paper is based on lexical approaches only and it is mainly

| Phenomenon | Ratio | Examples |
|---|---|---|
| Ortographic errors | 28% | sacalo → sácalo, trapirar → transpirar, ... |
| Texting Language | 22% | x2 → por dos, q → que, aro → claro, ... |
| Character repetition | 15% | siiiiiiiii → si, quiiiiieeeeroooo → quiero, ... |
| Ascii Art | 14% | ♠ ❤ oO._.Oo ... |
| Free inflections | 7% | besote, gatino, bonico, ... |
| Other errors | 7% | htt, asdafawecas, engoriles, ... |
| Other Language | 4% | flow, ftw, great, lol, ... |
| Multiple phenomena | 3% | diass → días, artooo → rato, ... |

Table 1: Characterization of error phenomena commonly found in Twitter media

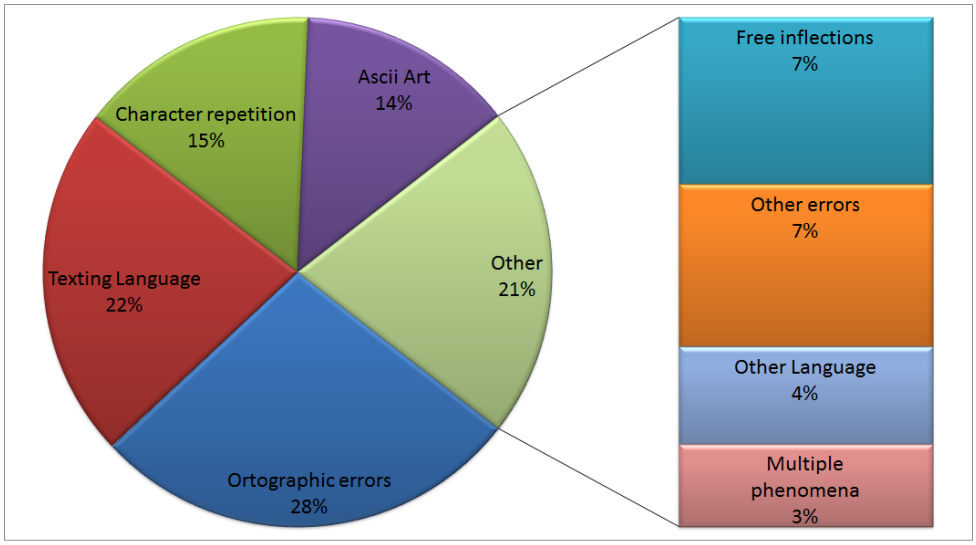


Figure 1: Ratio of characterized error phenomena commonly found in Twitter media

composed of three types of components:

- **Resources**: Lexicons and similar language resources, including resources containing specific knowledge of the media used.
- **Rules**: Rules for handling common phenomena found in this type of media as excessive character repetition, acronyms or homophonic errors.
- **Lexical distance analysis**: Traditional lexical distance analysis for handling common ortographic errors found on it.

In essence, our system works straightforwardly: it examines each word at lexical scope, determine if it is an OOV using the knowledge, generate possible correction candidates and select the best one.

## *2 Architecture and components of the system*

The architechture of our system proposed for this task is pretty straightforward. It is composed by several main components:

- Preprocessing module
- OOV/IV detection module
- OOV analyzer modules
- Candidate generator module
- Candidate scoring and selection module

The figure 2 shows all the process explained before and how the components are interconnected in a single diagram.

The preprocessing module performs the typical initial processing step done in lexical analysis, generating a stream of tokens from tweets taking into account things like hashtags and usernames, numerals, dates and preserving emoticons during the splitting.

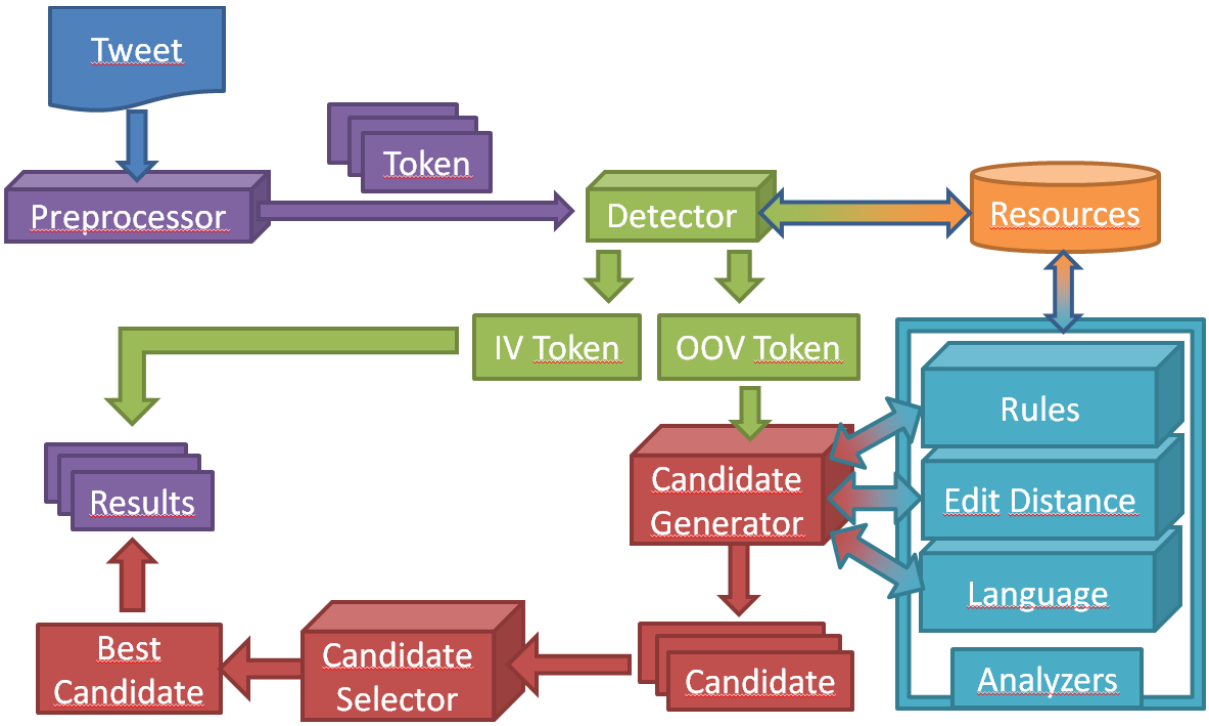


Figure 2: Architecture and processing steps of the proposed system

The detection module tries to determine if a token is an OOV or not. This module performs that detection using resources and checking if a token belongs to any resource. We used a set of lexicons, each one providing known forms used in Twitter, the Spanish language, well known emoticons or even colloquial inflections.

Given an OOV Token, an analyzer module perform some "error guessing" process and try to estimate corrections from it. The specific process varies for each analyzer. Every analyzer provides some kind of *basic scoring* providing some degree of confidence for each correction proposed. The analyzers used for this task were the following:

- **Tranformation rules**: This analyzer holds a collection of hand-crafted rules, each one representing some kind of "well defined" error and *transforms* that token into candidate of correction. It is possible to generate more than a candidate due multiple rule matching, but the number usually is limited to a few.

  These rules are intended to address phenomena that the edit distance module does not correctly address like *Character Repetition* or *Texting Language*. The table 2 shows some example rules, using Python's regular expressions.

- **Edit distance**: This module works very similar to distance-based suggestion scheme commonly found in spell checkers. The main difference is that it takes into account multiple lexicons instead a monolithic one.

- **Language**: This module tries to identify whether language the OOV token actually belongs to.

Notice that the language analyzer module does not actually perform any correction, because if the token comes from another language only has to be marked not corrected. This module uses a trigram language guessing module Python 3 implementation (Phi-Long, 2012) as backend.

The candidate generation module asks for candidates to each analyzer, performing a validation and filtering step, thus removing some incorrectly generated candidates from transformation rules according to validation rules and used language resources. Also removes duplicates.

The candidate selector module applies a normalizing scoring function from the confidence values provides for each candidates,

| Matching | Processing | Example | Phenomenon |
|---|---|---|---|
| ^[ck]n$ | con | kn, cn, → con | Texting Language |
| x([aeiouáéíóú]) | ch\1 | xaval, coxe → chaval, coche | Texting Language |
| ((\w)(\w))\1+(\2\|\3)? | \g<1> | sisisisisisi, nononononono → si, no | Character Repetition |
| ^t[qk]m+$ | te quiero mucho | tkm, tqm → te quiero mucho | Texing Language |

Table 2: Extract of the transformation rules used in our system

sorts them and selects the best one.

The system generates a token stream from the Tweet using the preprocessing module. For each token, determines if a token is an OOV or not using the detector module. If the token is an IV, no further processing is done because is a *valid form*. Otherwise, the token is an OOV, the candidate generator module creates a tentative list using the analyzers previously described. As final step, the candidate selector module selects the best candidate for correction.

## 3 Resources employed

We have used several lexicons for the detection and analyzing stages in our system. All of them are in raw text format and one entry per line.

The table 3 shows stats about all the lexicons used.

| Lexicon | Entries | Description |
|---|---|---|
| Spanish | 1250796 | Common forms from Spanish. Based on `LibreOffice` dictionaries. |
| Genre | 40 | Common forms related to Twitter. Handcrafted. |
| Emoticons | 320 | Commonly used emoticons. Handcrafted. |

Table 3: Lexicons used for our proposed system

For the transformation rule module, we crafted a ruleset of 71 rules. The syntax used in the ruleset file vaguely resembles a CSV format, being a rule per line and each line holds information about the matching and the transformation process.

The language detector module uses a trigram character language model implementation as backend and uses dictionaries as backoff just in case of insufficient data for language estimation.

## 4 Settings and evaluation

The table 4 shows performance values of the system against the provided corpus and activating different analyzer modules. It is observed that the accuracy of the system improves significantly as more modules are activated.

However, our implementation performance is hindered due high differences between the preprocessing used to build the corpus for this task and our detection and preprocessing system. Our preprocessing module leads to different sets of OOVs to be considered, often leading to outputs that differ in length respect the ones provided with the task. These discrepancies result in a high rate of what the provided test script counts as *align errors*.

| Modules | Accuracy | Align Error |
|---|---|---|
| Distance module | 0.2036 | 0.2526 |
| Rule module | 0.3307 | 0.3231 |
| Distance + Rule | 0.3905 | 0.2281 |
| Full system | 0.5053 | 0.1684 |

Table 4: System performance with different modules activated

It is worth mentioning that we used threshold of $k \leq 2$ for the edit distance module because most of correct candidates are within that threshold. Though is true that selecting a higher $k$ includes more candidates, most of newly included candidates are not a valid solution and usually they will have a low confidence score and will not be selected.

## 5 Conclusions and future work

We provide a resource-aided lexical solution for the proposed task using an extensible architecture made of independent modules. Our system has much room for improvement like adding a ngram segmenter, including context during analysis or using automatic methods for improving the candidate scoring and selection.

## *References*


Han, Bo and Timothy Baldwin. 2011. Lexical normalisation of short text messages: makn sens a #twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 368–378, Stroudsburg, PA, USA. Association for Computational Linguistics.

Han, Bo, Paul Cook, and Timothy Baldwin. 2013. Lexical normalization for social media text. *ACM Trans. Intell. Syst. Technol.*, 4(1):5:1–5:27, February.

Pennell, Deana and Yang Liu. 2011. A character-level machine translation approach for normalization of sms abbreviations. In *IJCNLP*, pages 974–982.

Phi-Long. 2012. Python 3.3+ implementation of the language guessing module made by Jacob R. Rideout for KDE.

Xue, Zhenzhen, Dawei Yin, Brian D Davison, and BD Davison. 2011. Normalizing microtext. In *Analyzing Microtext*.