Tokenizing Micro-Blogging Messages using a Text Classification Approach

Gustavo Laboreiro
LIACC — Faculdade de Engenharia da Faculdade do Porto — DEI
Rua Dr. Roberto Frias, s/n
4200-465 Porto PORTUGAL
gustavo.laboreiro@fe.up.pt

Luís Sarmento
Labs SAPO and LIACC — Faculdade de Engenharia da Faculdade do Porto — DEI
Rua Dr. Roberto Frias, s/n
4200-465 Porto PORTUGAL
las@co.sapo.pt

Jorge Teixeira
Labs SAPO and LIACC — Faculdade de Engenharia da Faculdade do Porto — DEI
Rua Dr. Roberto Frias, s/n
4200-465 Porto PORTUGAL
jft@fe.up.pt

Eugénio Oliveira
LIACC — Faculdade de Engenharia da Faculdade do Porto — DEI
Rua Dr. Roberto Frias, s/n
4200-465 Porto PORTUGAL
eco@fe.up.pt

ABSTRACT

The automatic processing of microblogging messages may be problematic, even in the case of very elementary operations such as tokenization. The problems arise from the use of non-standard language, including media-specific words (e.g. "2day", "gr8", "tl;dr", "loool"), *emoticons* (e.g. "(ò_ó)", "(=ˆ-ˆ=)"), non-standard letter casing (e.g. "dr. Fred") and unusual punctuation (e.g. ".... ..", "!??!!!?", ",,,"). Additionally, spelling errors are abundant (e.g. "I;m"), and we can frequently find more than one language (with different tokenization requirements) in the same short message. For being efficient in such environment, manually-developed rule-based tokenizer systems have to deal with many conditions and exceptions, which makes them difficult to build and maintain. We present a text classification approach for tokenizing *Twitter* messages, which address complex cases successfully and which is relatively simple to set up and maintain. For that, we created a corpus consisting of 2500 manually tokenized *Twitter* messages — a task that is simple for human annotators — and we trained an SVM classifier for separating tokens at certain discontinuity characters. For comparison, we created a baseline rule-based system designed specifically for dealing with typical problematic situations. Results show that we can achieve F-measures of 96% with the classification-based approach, much above the performance obtained by the baseline rule-based tokenizer (85%). Also, subsequent analysis allowed us to identify typical tokenization errors, which we show that can be partially solved by adding some additional descriptive examples to the training corpus and re-training the classifier.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Linguistic processing*

General Terms

Design, Experimentation, Measurements

Keywords

user-generated content, tokenization, corpus, microblogging, twitter, text pre-processing

1. INTRODUCTION

Despite its potential value for *sociological studies* and *marketing intelligence* [8], microblogging contents (e.g. *Twitter* messages and *Facebook* status messages) remain a challenging environment for automatic text processing technologies. Some of these challenges arise from the fact that micro-blogging messages, as most User-Generated Content (UGC), are populated with a significant number of *misspelled* or *unknown* words and *media-specific* vocabulary (e.g. "lol"). Some work has already been done in developing text *pre-processing* methods for homogenizing UGC, and transform it into text that is more amenable to traditional Natural Language Processing procedures, such as *contextual spell-checking* [7] and name normalization [2, 4].

In contrast with more mature user-generated media (such as web logs), the microblogging environment has some additional characteristics that further complicate automatic text processing. First, since messages are limited to 140–200 characters, text economy becomes crucial, and users tend to produce highly condensed messages, skipping characters whenever possible (including white space), ignoring proper standard casing rules, and creating non-standard abbreviations (such as "altern8") — similar to SMS text [1]. Second, the quasi-instant nature of micro-blogging diffusion promotes conversations between micro-bloggers. Thus, messages tend to have a strong presence of *oral markers*, such as *emoticons* and non-standard or missing punctuation. Finally, due to the almost ubiquitous access to a network, users can micro-blog from any place using mobile devices, which, for usability reasons, tend to

Table 1: Examples of tokenization challenges.

1	"the people at home scream at the **phone.and** complain when i **scream,like** 'ahhhhhhh'. **af,start** start start"
2	"Band in the style of Morchiba I just discovered: Flunk. Very good. Thank you **last.fm**!"
3	"i didn't seee dougie **O.o** should I go to the doctor ??"
4	"@FirstnameLastname Really ? ? the first time I saw I was in **Frankfurt-GERMANY**, and this is the first time I saw Madri this way ,cuvered in snow.. kiss"
5	"@Berrador Benfica is not to blame that Vitor Pereira is irresponsible..But it's bad for everyone. Notice the appointing of Duarte Gomes to the **fcp-scp**"
6	"Be a bone marrow **donor-Show** up in norteshopping until friday and unite for this cause."
7	"What is worse, **coca-cola** without gas or beer without alcohol?"
8	"@aneite can i then schedule a **check-up** ? (cc @first-lastname)"
9	"Today's theories: **a)** if its in april, a thousand waters, then have rain until april **b)** if rain wets, then whoever walks in the rain gets wet -.-' "
10	"Have a good week, Guys **:o)**"
11	"The movie's impressive numbers RT **@Firstname-Lastname**: Ubuntu Linux is the key to Avatar's success **http://bit.ly/7fefRf #ubuntu #linux #avatar"**

constrain how users write messages and (not) correct them in case of spelling mistakes.

We focus on a fundamental text pre-processing task: *tokenization* (or symbol segmentation) as the initial step for processing UGC. For instance, in regular text — such as in printed media — the slash ("/") has a restricted number of use cases, and from a tokenization point-of-view it is unambiguous. In the microblogging context, a slash may also be part of a URL or a smiley. In fact, this kind of problem is also found in some specialized fields, such as the biomedical domain [13], where some common symbols (such as "/" and "-") may perform a function *within* the token, (e.g. "Arp2/3" is a protein name, and "3-Amino-1,2,4-triazole" is an organic compound). As their interpretation becomes more context-dependent, specialized tokenizer tools become essential.

Tokenization of microblogging messages thus may be quite problematic. The most common problem is users simply skipping white space, so the tokenizer has to determine if those spaces should exist. Table 1 has some illustrative examples. Message 1 shows the typical situation of the missing white space. While these situations could be fixed using simple rules, messages 2 and 3 exemplify other frequent situations where such a simple approach would not work. Messages 4, 5 and 6 show examples where the tokenizer should divide the emphasized text in three tokens: "Frankfurt - GERMANY", "fcp - scp" and "donor - Show". By contrast, in messages 7 and 8, the emphasized text should be considered as a single token. Cases involving parentheses also provide good examples of how tokenization may be complex. Usually, parentheses make up their own token. However, when used in enumerations and smileys, the parentheses are part of the larger token (as in messages 9 and 10). Finally, there are also many microblogging-specific exceptions to traditional tokenization operations, such as *usernames* (@username), *hash tags* (#hash_tag) and URLs that must be kept intact (example 11 illustrates these situations).

It would be possible to use an existing tokenizer and add new rules to cover these additional situations. One such tokenizer is *UPenn Treebank*'s tokenizer[1], which fails to correctly tokenize many of the above examples — for instance, it has no understanding of URLs or smileys, resulting in tokens such as "http : //bit.ly/123456" and ": -)". However, and contrary to *traditional* text, microblogging content is very irregular (with each author creating their individual writing style), and the token universe is *open*, due to the creative use of letters, numbers and symbols (i.e. characters other than letters, numbers and white space).

For the above reasons, many tokens are not in a dictionary. We cannot rely on compiling a list of possible emoticons, for example, and new ones continue to appear. Some can be quite elaborate, for instance: "8<:-)", "orz" (a kneeling man), "<(-'.'-)>", "(ò_ó)" or "(=^-^=)". The "orz" example in particular shows that simple heuristics are insufficient to identify all emoticons, as they can pass as an unknown word[2]. A more frequent example is the "XD" smiley. Without a compiled list or an adequate pattern to match them against, emoticons are difficult to accommodate in a rule system developed manually. This is not a small setback, as emoticons play a fundamental role in the interpretation of UGC-like messages, such as sentiment analysis[10] or opinion mining tasks.

We propose a *text-classification* approach to the tokenization of microblogging messages. We train a classifier for deciding whether a white space character should be introduced *before* or *after* certain symbols, such as *punctuation*, *quotation marks*, *brackets*, etc. Compound words (such as "airfield") will never be separated.

This tokenizer is only concerned with the separation of tokens. We assume that other tasks normally associated with text pre-processing, such as the identification of compound multiword expressions, character transposition, text normalization, and error correction are to be delegated to other modules further down the processing pipeline.

For training the classifier we use a set of manually tokenized *Twitter* messages, which can be easily obtained, as the annotation task itself is relatively simple for humans. The key point is that adding more (and more diverse) training examples is an easier way to improve the performance than adding new rules[9].

In this paper we address the following questions:

- Can a classification-based approach to tokenization of UGC clearly outperform a tokenizer based on rules specifically crafted for this purpose? If so, how much training data is required for achieving such performance?

- Which features — both in diversity and amount of contextual information — does the classifier need to obtain optimal tokenization performance?

Our work is mostly done on Portuguese text, and that influences some of the decisions made when defining token borders. However, our methods and results are valid when considering most western languages with little or no adaptation.

The remainder of the paper is organized as follows: in Section 2 we introduce some work that relates to tokenization or text pre-processing. In Section 3 we will describe the method used to tokenize UGC. In Section 4 we explain the rules for the processing of the corpus and why we made such choices. In Section 5 the experimental setup is described, explaining how we reached the results, that are then presented in Section 6. We analyze the most significant errors in Section 7, and then conclude our paper with a summary of the results and directions for further work.

[1]http://www.cis.upenn.edu/ treebank/tokenizer.sed
[2]"Orz" is also the name of an alien race in the fictional universe of Star Control 2.

2. RELATED WORK

Tomanek et al. study the tokenization problem on biomedical texts [13]. The notation used to write *biomedical entity names*, *abbreviations*, *chemical formulas* and *bibliographic references* conflicts with the general heuristics and rules employed for tokenizing common text. This problem becomes quite complex, since biomedical authors tend to adopt different, and sometimes inconsistent, notations. The method proposed consists in using Conditional Random Fields (CRF) as text classifier to perform two tasks: (i) *token boundary detection* and (ii) *sentence boundary annotation*. For obtaining the corpus, the author gathered a number of abstracts from works in the biomedical fields, and extended it with a list of problematic entities. The set of features used for classification includes, for example, the unit itself, its size, the canonical form, the orthographical features, whether it is a known abbreviation and the presence of white space on the left or right of the analyzed point. After training the classifier they achieved a tokenization accuracy of 96.7% (with 45% of false positives).

Tomanek's successful method in biomedical texts cannot be reproduced when applied to UGC due to the different nature of the texts. Scientific articles obey general editorial rules. They impose a certain standard writing style, assure that the author is consistent in the language used (usually English), and that the text is easy to read. In UGC there is no review process, leading to many non-uniform messages (incorrect capitalization, for instance) with higher noise levels (non-standard punctuation, like ",,,"), multiple languages in the same short message (difficult to recognize due to misspellings and the small amount of text for statistical analysis), and as stated before, the open nature of the lexicon makes the "problematic entity list" strategy inviable.

Takeuchi et al. propose a method for solving the problem of identifying *sentence boundaries* in transcripts of *speech data* [11]. The difficulty arises from the complete lack of punctuation in the transcripts, and also from word error rates of about 40%. The method, named *Head and Tail*, is a probabilistic framework that uses the information about typical 3-grams placed in the beginning (Heads) or in the end of sentences (Tails) to decide if there is a sentence boundary or not. The authors compare their method with a maximum entropy tagger trained on manually annotated corpora with boundary information. They also define a baseline that uses only the information about silence in the transcripts to decide if there is a boundary. Results show that the method proposed outperforms both the maximum entropy tagger and the baseline. Moreover, results improve further when the Head and Tail method is configured to use silence information and additional heuristics to remove potentially incorrect boundaries.

We believe that the tokenization task we are addressing is, in certain ways, similar to the word segmentation problem in Chinese [12]. Although tokenization of UGC is expected to be simpler than the word segmentation in Chinese, they both require making use of contextual information for deciding whether two symbols should be separated or not. Additionally, and contrary to word segmentation in Chinese, tokenization approaches in microblogging environments cannot make a solid use of dictionary information, since a very large number of "words" are not lexicalized. Our tests estimate that nearly 15.5% of tokens containing only alphabetic characters are not in a standard lexicon[3].

The most successful approaches to the tokenization problem use machine learning techniques. For example, Tang et al. compare the performance of three classification algorithms, namely (i) Support Vector Machines (SVM), (ii) CRF and (iii) Large Margin Methods, for achieving tokenization [12]. Results show that text classification is capable of achieving relatively high F-measures (95%). This led us to believe that machine learning approaches can also be successful in tokenizing microblogging messages. On the other hand, the task of *word chunking*, i.e. grouping words together to form lexical or syntactical compounds, can be seen as special type of tokenization where each character is used as a "token". Again, machine learning approaches have proven successful. For example, Kudo and Matsumoto used several SVMs with weighted voting [6]. Their results also show that accuracy is higher, regardless of the voting weights, compared with any single representation system.

[3]The Portuguese GNU Aspell at http://aspell.net/

3. A METHOD FOR TOKENIZING MICROBLOGGING MESSAGES

A token is an atomic symbol used in text processing. Its definition is very dependent on what role it will play in the following processing pipeline. For instance, when tokenizing text for machine translation, it may be preferable to identify frequent expressions as tokens and not just single words. This "flexibility" is reflected in different results when comparing different tokenizers [3], that may not be simply interchanged.

Our intent is to perform information extraction on microblogging messages. For that it is fundamental that we first correct the frequent misspellings made by the users, and then normalize the inconsistent text. If we opt for a "traditional" tokenization process, we may split tokens that are difficult to piece back together at a later time. (E.g. "alternate" typed as "altern8", or "I'm" typed as "I;m".)

We define that each token is an element that can *later* be considered a unit of processing, for example: a word, a punctuation block (e.g. "!?", ","), a URL, or a microblogging-specific mark such as a user reference (e.g. @PeterP).

Tokenization is achieved by adding a space character in special *decision points* of the microblogging message. These points surround all non-alphanumeric characters (excluding the white space). In the following example, we use the symbol "||" to indicate decision points where white space should be inserted, and mark with a "|" those that should be left intact (i.e. white space is already present or adding a space character would break the token).

> "Oh||,| if I was in Sampa||.|.|.| I would certainly go||!| |"||Beat it||"| Flash Mob||,| 25|/|01||,| at 14h||,| in Anhangabaú Valley||.| |(||via |@|terciors||)"

The tokenization method that we propose can thus be stated as a *binary classification problem*: can we train a model to determine if a certain decision point should have a space character inserted (or not)?

At this point, the question of an adequate training corpus arises. In Section 5.3 we describe how we annotated microblogging messages to create such a data source.

3.1 Extracting features for classification

The first step in our method is defining the features that we use. These features are based on the properties of each *character* that composes the message. Therefore we can say that we work at *character level*. Every character is described by at least one feature.

All features used are binary (they are either present or absent), and describe different aspects of the characters (e.g. nature, function and purpose). No effort is made to reduce the number of features — it is up to the classification algorithm to identify the relevant features. In total we created 31 distinct features. Some examples follow:

- Character nature: alphanumeric, alphabetic, numeric, punctuation, symbol, space, etc.
- Type of alphabetic characters: upper case, upper case without accent, letter with accent, vowel, etc.
- Type of symbol and punctuation: bar, dash, monetary symbol, closing symbol (i.e.“(”, “>”, “}”, “]”), opening quotation (e.g. “"”, “«”), accent (e.g. “^”, “~”), arithmetic operator, smiley nose, etc.

The window of characters for which we generate features extends from the relevant decision point in both directions. This means that when using a 10 character window, the classifier can base its prediction on the features of the previous 10 characters, and the next 10 characters. The size of the window is one of the parameters of the process. One should take into account that using all characters in the message would greatly increase the feature space, and thus the computational requirements. Even worse, it could also worsen the tokenizer's results. This could happen as the classification algorithm may infer incorrect patterns due to insufficient number and variety of examples covering the extended context.

The relative position of the character and the name of the feature are encoded into the features themselves. For example, “char_is_alphanumeric_pre-3” means that an alphanumeric character is found 3 characters to the left of the decision point.

Both positional and character information are required to address certain vocabulary-specific cases that are usually problematic in tokenization, like frequent abbreviations (for example, “Dr.” and “etc.”) and URLs (“http://”). The relevant feature in these cases is the *literal* feature. It represents each individual character, and provides the information “this is a “D” character”, or “this is a “$” character”.

Each character can be described by a number of features that is close to 180 (one for each literal character, plus the 31 category features we defined). Even with a small feature window size, the dimension of the *feature space* is very large.

4. TOKENIZATION GUIDELINES

The rules that govern our tokenization will have a big impact in later processing tasks. Subtle decisions at this stage can lead to very different results down the NLP pipeline [3].

A set of annotation guidelines was produced to ensure a good level of consistency during the manual annotation of the training corpus. These are summarized below to describe the corpus creation effort — the problems found and the decisions made. As a general rule, to avoid compromising the subsequent processing phases, it is essential not to “break” some “malformed” constructions, and lose the opportunity to correct them in later stages. Hence, we focus on what the author intended rather than what they actually typed.

4.1 Word tokens

Word tokens are defined in a loose way, due to the nature of the messages. For example, “2day” is treated as an equivalent to “today”, and both are equal *word* tokens. *Twitter* usernames (@twisted_logic_), hash tags (#Leitao2010) and hyphenated compound words (“e-mail”) are also considered regular words/tokens. It is common to use the dash as a replacement for a word in certain circumstances. For example, with “Egypt–Nigeria” the author means “Egypt *versus* Nigeria”, or “from Egypt *to* Nigeria”, and therefore it should be tokenized as “Egypt – Nigeria”. Also, acronyms are traditionally written with a dot, as in “U.K.”, and need to be recognized as one unit. In case of abbreviations, the dot is also considered part of the word.

Another situation is the contraction used in informal oral speech, where the apostrophe is used to indicate that one syllable is unpronounced. This is seen in English, as in “li**'**l” or “lil**'**” meaning “li**tt**le”, or in Portuguese where “**'**tava” stands for “**es**tava”. Once again, the token is considered as atomic, since the apostrophe is meaningless alone in such a situation. However, the apostrophe is also used to combine two words, as in “I'm”. In this situations, it should be tokenized as two separate tokens (in the example: “I 'm”).

Special care is taken with words that are typed incorrectly, as we expect them to be a frequent occurrence. The general rule is treating the string as if it was correctly written. A common error is the user pressing the key next to the correct one. For instance, “I**;**m” instead of “I**'**m” should be split as two tokens; while “im**[**ossible” instead of “im**p**ossible” should be kept as-is. Another problem with keyboards is when they are misconfigured (set to an incorrect layout). A user expecting to type “for**ç**a” in a Portuguese layout would produce “for**;**a” if the keyboard would be set to US.

Other difficult situations are deliberately created by the user. For instance, when they cannot find the symbol they want on their keyboard. One example is writing the “©” symbol as “(c)”. Typing accented letters in some foreign keyboards is a more complicated problem. The apostrophe is a known way of signaling acute or grave accents in these situations; for example, typing “ate**'**” meaning “at**é**”. One final situation happens when a user changes one or more letters in their text to obfuscate it, as a way to avoid censure (e.g. “sh*****t”). Some of the occurrences described here are not very frequent, but they illustrate the diversity of problems being addressed.

4.2 Numeric tokens

We decided to separate the numbers from the units they represent, as some people write “1€” and others opt for “1 €” or even “€1”. Dates and times, that possess a more complex representation, are preserved as a single token whenever possible.

Another problem occurs when dealing with ordinals, that joins numbers and letters or symbols. For example, in English UGC “1st” is typically typed as “1st”, while in Portuguese it is written as “1^{o}” or “1^{a}”, and it is often found written as “1.^{o}”, “1.^{a}”, “1o” or “1a”. No space character should be inserted in any of these situations, as it represents a single word, and information could be lost (e.g. interpreting “st” as a short for “street” or “saint”).

4.3 Punctuation and other symbols

When dealing with punctuation, end-of-sentence symbols (“.”, “. . . ”, “!” and “?”) are kept grouped as a single token. The other punctuation characters and symbols create their own token, except in the rare instance where they are duplicated. In this way, “--,” is transformed into “-- ,”, that is, the dashes are kept grouped, and the coma is another atom. This rule is used for all non-punctuation symbols as well.

There are a few exceptions to the rules described above: URLs and emoticons/typographical symbols (e.g. “=>”) are not broken-up, as they would otherwise loose their meaning. The same applies to enumerators such as “a) b) c)”.

5. EXPERIMENTAL SET-UP

Our experiment is based on the comparison of the performance of a classification-based approach with that produced by a regular expression based method that will be used as a *baseline*. This rule-based method is inspired by the UPenn Treebank tokenizer, but follows our rules of tokenization previously defined in Section 4, and consequently is better suited for UGC than UPenn's.

Table 2: Tests generated from an example message for both scenarios.

Tokenized	`@PeterP Yes ;-) ...`
S_{all}	`@PeterP Yes;-)...`
S_{one}	`@PeterP Yes;-) ...`
	`@PeterP Yes ;-)...`

5.1 The tokenization methods

We opted for using SVM [5] as our classification algorithm, which were found to be extremely robust in text classification tasks, and can deal with the *feature space* we have (in the order of 180^N, where N is the number of characters in the feature window). In addition, they are also binary classifiers, as the decision is always between inserting or not inserting white space. We used an "off-the-shelf" implementation of the SVM classifier (SVMLight[4]).

To determine the best kernel to use with the SVM, simple preliminary tests were executed. The second degree polynomial function outperformed the default linear kernel by a small but significative amount, and therefore is used. However, an exhaustive search for optimum parameters for the SVM falls outside the purpose of this work, and the remainder of the parameters were left at their default values.

We wish to investigate how this classification approach behaves according to two variables: (i) the dimension of the corpus and (ii) the size of the feature window.

Our *baseline* algorithm works by trying to isolate tokens that belong to one of a series of categories, each one defined by a regular expression:

1. URL, identified by the "http://" or "www.";
2. a block of alphanumeric words starting with a letter (like a regular word);
3. a date, as "2010-07-08" or "25/12/2009";
4. a time, such as "4:15" or "18h15m";
5. a number, for example, "-12,345.6" or "2º";
6. one of about 25 popular smileys and variants;
7. a block of punctuation characters;
8. a user reference ("@john");
9. a hash tag ("#true");
10. a block of unexpected symbols.

5.2 Evaluation Scenarios and Measures

Both methods (classification and rule-based) are evaluated under two scenarios, that we consider complementary. In the first scenario, S_{all}, we remove *all* white space adjacent to each decision point, joining some tokens together (but not sequences of words). In the second scenario, S_{one}, the test messages have been correctly tokenized except for *one* white space that is removed. We generate one test for each white space symbol excluded in this scenario.

We will use the tokenized message "@PeterP Yes ;-) ..." as an example. It has 8 *decision points*: "@|PeterP Yes |;|-|)| |.|.|.". Only 2 of the 3 spaces will be removed, as one is not adjacent to a *decision point*. The test messages produced from this example text are shown in Table 2.

When evaluating the tokenization results, for each decision point, a *True Positive* (TP) is a space character that is correctly inserted, and a *True Negative* (TN) is a space character that is correctly *not* inserted. A space character inserted in the wrong place is considered a *False Positive* (FP), while a missing one is a *False Negative* (FN).

[4]http://svmlight.joachims.org/

The performance can thus be determined using Precision (P), Recall (R), F1 and Accuracy (A) measures calculated as

$$P = \frac{TP}{TP+FP} \quad R = \frac{TP}{TP+FN}$$

$$F1 = \frac{2PR}{P+R} \quad A = \frac{TP+TN}{TP+TN+FP+FN}$$

These values are all obtained using a 5-fold cross validation process.

We can see that both scenarios address different aspects of the problem. In the first scenario all spaces around *decision points* have been removed, so the tokenizer has to put them back. This scenario tests the *Recall* of the tokenizer. The chance of introducing incorrect white space is relatively low, and adding back spaces is important. The scenario S_{one}, where only one space next to a *decision point* was removed, tests the *Precision*. Any other white space added is incorrect, as there is *exactly one correct classification*. The situations typically found in UGC lie somewhere in between these two scenarios.

5.3 The corpus creation

The data used in the classification consists of a manually annotated corpus with 2500 messages from *Twitter* users in Portugal, collected using the SAPO[5] broker service. These messages were randomly selected from 6,200,327 posts collected between 2010-01-12 and 2010-07-08, from a total of 93,701 distinct user accounts. The vast majority of the messages are in Portuguese, but there are traces of other languages. To split the workload, these messages were manually tokenized by 5 annotators using the rules stated in Section 4, at an average rate of 3 messages per minute. To ensure a good level of consistency, the entire set of examples was later reviewed by the first author.

6. RESULTS AND ANALYSIS

Figure 1 illustrates the performance (F1) of the SVM classifier in both scenarios (S_{all} and S_{one}) as a function of the dimension of the training corpus. We used a feature window of size 10 in this experiment. The rule-based system simply ignores its training data. We can see that with just 100 training examples the SVM classifier already outperforms the rule-based tokenizer significantly, as expected. Adding more examples consistently improves the SVM's results. The rule-based approach would only provide better results if we added new rules — hence its almost stagnant F1.

It can also be observed from Figure 1 that the results for scenario S_{all} are always higher than those obtained for S_{one}. To illustrate the relative complexity of each scenario we tried a trivial tokenization approach — always inserting a space at every opportunity. This oversimplified method obtained F1 values of 0.672 and 0.273 for S_{all} and S_{one} respectively.

In Figure 2, we can observe the results of varying the feature window's size. Scenario S_{all} is less influenced by the size variation of this window than scenario S_{one}. It can also be seen that the window of size 10 used in the previous experiment is *not* optimal, as the highest performance lies between 3 and 6 characters. Window sizes larger than 6 characters (to the left *and* the right of the *decision points*) show a decrease in performance. This can be justified by the sparsity of the features being generated, taking into account the size of the training set.

Significant values for feature window sizes considering F1 and accuracy are shown in Table 3. These include the highest F1 values as well as the performance obtained when using a 10 characters

[5]http://www.sapo.pt/

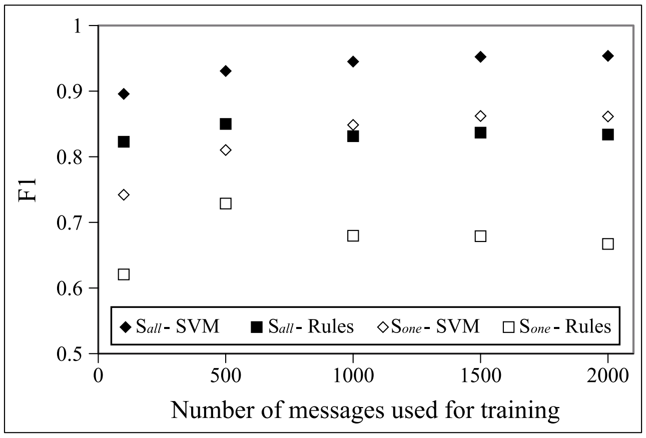

Figure 1: F1 vs training corpus size for SVM classifier for S_{all} and S_{one}, and analogous test for the baseline rule-based system.

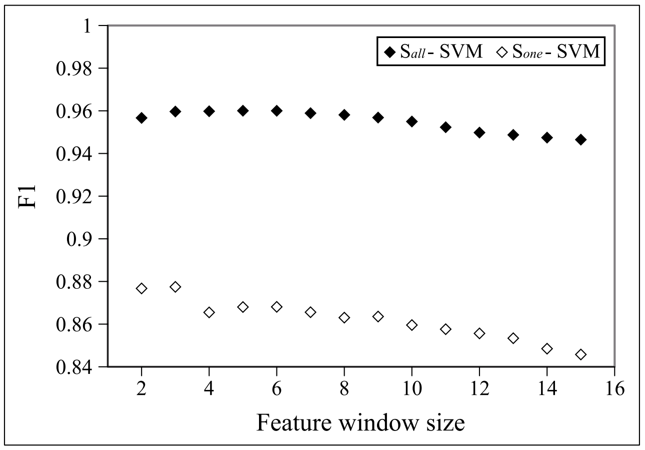

Figure 2: F1 vs feature window size in characters for SVM classifier, trained using 2000 messages for S_{all} and S_{one}.

feature window used to evaluate the effect of corpus size. It is not easy to compare our results with related work, since most tokenization projects (reviewed in Section 2) have different goals, and do not deal specifically with user-generated content. Even so, we claim that the results that we obtain in our task are at the same level as those presented in other works, such as 96.7% accuracy in the tokenization of biomedical texts [13] and 95% F-measures in the tokenization of texts in Chinese [12].

7. ERROR ANALYSIS

We will describe the most significant errors made by the classification-based tokenizer when trained with 2000 messages (500 testing messages), with a feature window of size 3, as it provided the best general results.

Table 4 shows the characters most frequently associated with tokenization errors. As expected, the most problematic characters are those that should be tokenized in different ways, depending on their context. Those challenges have been presented in the discussion of Table 1 and presented in Section 4, so we were not surprised with this result. Every character in the table can be tokenized "alone" in some situations, and as part of a larger token in other contexts —

Table 3: Feature window sizes for S_{all} and S_{one}, by F1 and accuracy values

Window	S_{all}		S_{one}	
size	F1	A	F1	A
3	0.9597	0.9622	**0.8775**	**0.7817**
5	**0.9600**	**0.9626**	0.8680	0.7668
10	0.9550	0.9583	0.8596	0.7537

Table 4: The most problematic characters for both scenarios when processing 2500 messages, the errors found, total errors and the number of messages containing the character.

	S_{all}		S_{one}		Total	Messages w/
Char	FP	FN	FP	FN	errors	occurrences
.	160	96	812	53	1121	1587
-	123	77	586	43	829	477
’	55	7	342	1	405	86
:	52	19	255	24	350	1101
/	11	62	40	59	172	725
?	22	24	101	18	165	445
(	12	6	79	9	106	172
)	8	25	43	24	100	327

for example, when part of an emoticon (even if some characters are less frequently used in this way).

We can see that there is no direct correlation between the frequency of the characters and their tokenization error rate. For example, the comma is not even present in the table, while 748 out of our 2500 messages (28.7%) have at least one comma. This can be justified by the small number of functions associated with the comma. At the same time, the apostrophe is only present in 17% of the messages, and is processed incorrectly more often than correctly. This, however, is not true with the other characters in the table.

One of the most surprising error situations arises from the use of "." as a mistyping of "-", that was more frequent than we expected. The other unexpected result comes from the difficulty of processing the apostrophe correctly. Part of the reason derives from the many languages found in the messages. The better known cases such as the "'s" in English, the "l'" in French and the "d'" in Portuguese and Spanish require more examples than those available to infer the correct classification pattern. In addition, the use of the apostrophe as a replacement for an accent or part of an emoticon is also more frequent than we anticipated. This situation also reinforces the need for more examples that address uses of this character.

Even though the values in Table 4 are biased towards the *False Positives*, overall values in scenario S_{all} were balanced.

7.1 Augmenting training data based on errors

All errors that we detected seem solvable by adding more and better examples. To calculate the cost of improving our system's results, we performed an additional experiment. We created a new test set by selecting 500 random messages from our 6 million *Twitter* messages collection. Using a model trained with all 2500 messages in our previous corpus, we noticed 110 of the test messages were incorrectly tokenized. We then augmented our corpus with 400 new messages — 50 new messages containing each of the eight problematic characters in Table 4. The new model, trained with the now 2900 messages corpus, failed in only 89 messages out of the previous 500. That is a reduction of almost 20% in the number of messages tokenized incorrectly.

With little more than 2 hours of work from a non-expert (as the only requirement is for them to understand and follow the tokenization rules stated in Section 4), the classification system improved by a significant amount.

8. CONCLUSION AND FUTURE WORK

We showed that the problem of tokenization of UGC, although fundamental for subsequent language processing work, cannot be considered a simple task. The usual UGC (and microblogging in particular) way of expression can be problematic for "traditional" methods of text processing. In tokenization this is particularly evident. The difficulty arises from the lack of editorial rules and few accepted standards, meaning that ambiguity is not always avoidable. In particular, no character (other than white space) can be said to always mark the start or end of a token.

We created a corpus of *Twitter* messages that were tokenized manually following our rules for UGC tokenization. We then compared the results obtained by a classification-based and a rule-based tokenizer across two scenarios, that draw a typical upper and lower limit of performance for most microblogging usage.

In our experiments, we achieve F1 values of 0.96 for scenario S_{all} (that tests Recall) and 0.88 for scenario S_{one} (that focuses in Precision), for our classification-based tokenizer, when trained with 2000 examples. Our results can be said in-line with comparable systems in the literature.

We have also shown that text classification methods can successfully tackle this problem, and significantly better than rule-based classifiers. Not only do they achieve better performance, (based on F1 values), but also their performance can be improved significantly by simply adding to the training set more examples regarding the problematic situations. The text classification approach we propose, is also better in terms of development costs, since the bulk of the work (i.e., the annotation) can be distributed among a group of non-experts.

As future work, we intend to experiment with the SVM kernels and parameters, looking for an optimal classification configuration. We also wish to compare the results of the SVM with other classification algorithms, such as Conditional Random Fields (CRF). In addition, we also consider extending our feature set to include, for instance, information about keyboard layouts (e.g. which keys are next to the one that produces this character).

Looking even further ahead, producing language and/or user specific models could result in even better results, as more specific patterns (such as preferences for certain emoticons) can be identified.

9. ACKNOWLEDGMENTS

This work was supported in part by grant SFRH/BD/23590/2005 from FCT Portugal and grant SAPO/BI/UP/Sylvester from SAPO / Portugal Telecom. We would like to thank Claudio Valente (from SAPO) for his expertise in dealing with character encoding issues. We also wish to thank several anonymous reviewers for their useful comments, and to our tokenizers for their valuable help.

10. REFERENCES

[1] S. Acharyya, S. Negi, L. V. Subramaniam, and S. Roy. Unsupervised learning of multilingual short message service (sms) dialect from noisy examples. In *AND '08: Proceedings of the second workshop on Analytics for noisy unstructured text data*, pages 67–74, New York, NY, USA, 2008. ACM.

[2] R. Ananthanarayanan, V. Chenthamarakshan, P. M. Deshpande, and R. Krishnapuram. Rule based synonyms for entity extraction from noisy text. In *AND '08: Proceedings of the second workshop on Analytics for noisy unstructured text data*, pages 31–38, New York, NY, USA, 2008. ACM.

[3] B. Habert, G. Adda, M. Adda-Decker, P. B. de Marëuil, S. Ferrari, O. Ferret, G. Illouz, and P. Paroubek. Towards tokenization evaluation. In A. Rubio, N. Gallardo, R. Castro, and A. Tejada, editors, *Proceedings First International Conference on Language Resources and Evaluation*, volume I, pages 427–431, Granada, may 1998.

[4] V. Jijkoun, M. A. Khalid, M. Marx, and M. de Rijke. Named entity normalization in user generated content. In *AND '08: Proceedings of the second workshop on Analytics for noisy unstructured text data*, pages 23–30, New York, NY, USA, 2008. ACM.

[5] T. Joachims. Text categorization with support vector machines: learning with many relevant features. In C. Nédellec and C. Rouveirol, editors, *Proceedings of ECML-98, 10th European Conference on Machine Learning*, number 1398 in Lecture Notes in Computer Science, pages 137–142, Chemnitz, DE, 1998. Springer Verlag, Heidelberg, DE.

[6] T. Kudo and Y. Matsumoto. Chunking with support vector machines. In *NAACL '01: Second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies 2001*, pages 1–8, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[7] K. Kukich. Techniques for automatically correcting words in text. In *ACM Comput. Surv.*, pages 377–439, New York, NY, USA, 1992. ACM.

[8] M. Mathioudakis and N. Koudas. Efficient identification of starters and followers in social media. In *EDBT '09: Proceedings of the 12th International Conference on Extending Database Technology*, pages 708–719, New York, NY, USA, 2009. ACM.

[9] G. Ngai and D. Yarowsky. Rule writing or annotation: Cost-efficient resource usage for base noun phrase chunking. In *In Proceedings of ACL'02*, pages 117–125, 2000.

[10] J. Read. Using emoticons to reduce dependency in machine learning techniques for sentiment classification. In *ACL '05: Proceedings of the ACL Student Research Workshop*, pages 43–48, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

[11] H. Takeuchi, L. V. Subramaniam, S. Roy, D. Punjani, and T. Nasukawa. Sentence boundary detection in conversational speech transcripts using noisily labeled examples. *Int. J. Doc. Anal. Recognit.*, 10(3):147–155, 2007.

[12] B. Tang, X. Wang, and X. Wang. Chinese word segmentation based on large margin methods. *International Journal of Asian Language Processing*, 19(2):55–68, 2009.

[13] K. Tomanek, J. Wermter, and U. Hahn. Sentence and token splitting based on conditional random fields. In *PACLING 2007 – Proceedings of the 10th Conference of the Pacific Association for Computational Linguistics*, pages 49–57. Melbourne, Australia, September 19-21, 2007. Melbourne: Pacific Association for Computational Linguistics, 2007.