Pattern Matching Techniques to Identify Syntactic Variations of Tags in Folksonomies

F. Echarte, J.J. Astrain, A. Córdoba, J. Villadangos

Dpt. de Ingeniería Matemática e Informática
Universidad Pública de Navarra
Campus de Arrosadía. 31006 Pamplona (Spain)
patxi@eslomas.com, {josej.astrain, alberto.cordoba, jesusv}@unavarra.es

Abstract. Folksonomies offer an easy method to organize information in the current Web. This fact and their collaborative features have derived in an extensive involvement in many Social Web projects. However they present important drawbacks regarding their limited exploring and searching capabilities, in contrast with other methods as taxonomies, thesauruses and ontologies. One of these drawbacks is an effect of its flexibility for tagging, producing frequently multiple syntactic variations of a same tag. In this paper we study the application of two classical pattern matching techniques, Levenshtein distance for the imperfect string matching and Hamming distance for the perfect string matching, to identify syntactic variations of tags.

Keywords: Folksonomies, Semantic Web, Annotations, Pattern Matching

1 Introduction

Folksonomies [1] are based in the assignation of text tags to different resources, such as photos, web pages, documents, etc. Users use these tags to annotate resources defining collaboratively the meaning of the annotated resources, and the used tags. New search and exploration approaches are possible with Folksnomies, based on the use of the tags [2,3]. Users can search for tags, or use navigation systems such as clouds of words, to locate resources tagged by other users and to find information. Though folksonomies have a great success in current web, mainly due to their simplicity of use, they have also important disadvantages. The fact of users creating tags and assigning them freely to resources produces the inexistence of any structure among these tags. As folksomonies become larger, more problems appear regarding the use of synonyms, syntactic tag variations and different granularity levels [4]. All these problems make more and more difficult the exploration and retrieval of information [5,6] decreasing the quality of folksonomies Thus, the reduction of syntactic tag variations aids to improve the quality of folksonomies.

There exist different types of syntactic variations of tags: typographical misspellings in the annotation process (*semanticweb*/*semnticwev*/*zemantcweb*);

grammatical number (singular or plural) of the same word (*semanticweb/ semanticwebs*); separators (*semantic-web*/*semanticweb*); or a combination of them (*semntic-web*/*smanticweb*, *semntic-webs*, etc.). The existence of these variations causes the classification of the resources under different tags, when they should be classified under just one. This fact makes more confusing the clouds of words, the location of information and the navigation on the folksonomy. However, by identifying all of them as variations of the same label "*semantic web"* and grouping them under the same tag, a user can access this tag obtaining all the information concerning the resources associated with it and its syntactic variations.

In this paper we focus on the application of pattern matching techniques to identify syntactic tag variations. As contribution of this paper, we propose the utilization of pattern matching techniques to identify syntactic variations of tags. We study two classical pattern matching techniques as Levenshtein [7] and Hamming [8] distances on a large real dataset, evaluating how these techniques perform the identification of both variations of known tags and new (non-existing) tags.

We present the results of syntactical variations identification achieved for each distance considered. Only in [9] there is a related work, where a pre-filtering of the tags is performed before applying an algorithm for tag clustering. This is used to minimize the effects of syntactic variations and to increase the quality of tag clustering. Authors identify similar tags using the Levenshtein similarity metric to determine morphological variations, although over a reduced experimental data set. Another way to represent these variations is presented in [4] where an ontology with three properties associated to tags (*prefLabel*, *altLabel* and *hiddenLabel*) is used.

The use of pattern matching techniques designed to automatically recognize syntactic variations of tags provides mechanisms to improve the quality of folksonomies. Approximate string matching techniques allow dealing with the problem introduced by syntactic variations on folksonomies. The problem consists on the comparison of a candidate input string called α, maybe containing errors, and a pattern string ω in order to transform α in ω [10]. The Hamming distance [8] between two strings of equal length indicates the number of positions for which the corresponding symbols are different. It measures the minimum number of substitutions required to transform the candidate string into the pattern string. The Hamming distance can be shown as the number of errors that transform one string into the other. The Hamming distance is commonly used to perform perfect matching between pairs of strings but in our case we deal with pairs of strings with different length strings by padding white spaces at the end of the shortest string. However, for these situations and for those cases where not just substitutions but also insertions or deletions have to be expected, a more sophisticated metric like the Levenshtein distance is more appropriate.

Probably, the most relevant contribution in the field of imperfect string matching is the Generalized Levenshtein Distance [7]. The main drawback of these techniques is the limited number of errors that can be considered due to the finite number of rules introduced in the grammar to model edit operations.

The rest of the paper is organized as follows: section 2 describes the experimental scenario and section 3 presents and analyzes the experimental results obtained; finally, conclusions and references end the paper.

2 Workbench

This section describes the experimental scenario we have used to evaluate Levenshtein and Hamming distances, paying special attention to the datasets and the methodology followed. This workbench is available on the web[1].

Datasets – We have collected data from the social web CiteULike (which contains bibliographic cites) in order to evaluate our proposal, collecting a total number of 2,290,740 annotations. Each annotation consists on a tag assigned by a user to a resource, at a given date.

After a first analysis of the resulting data set, we can appreciate the existence of two tags with a significantly larger number of annotations than the rest, which correspond to some automatic procedure. We eliminate them. The resulting data set has the following characteristics: (1) 2,038,172 annotations (one record per user-tag-resource), (2) 494,206 resources, (3) 21,480 users, and (4) 151,522 tags.

In order to evaluate Levenshtein and Hamming distances, we have created two data sets: one with the aim of checking the correct identification of variations (DS1) and another (DS2) to check the recognition of new tags.

DS1 is obtained from the 10,000 most often used tags. These tags are used in 1,557,198 annotations, representing the 76.4 % of the total amount of annotations. DS1 consists of a set of tuples <*pattern tag-candidate tag*>: *pattern* is one of the 10,000 related tags, and *candidate* is a syntactic variation of the *pattern*. These variations are created automatically. These syntactic variations consider different cases: (i) the singular or plural, (ii) simulation of a typographical error, (iii) simulation of transposition of adjacent symbols, (iv) removal and replacement of separators, and (v) the own pattern tag in order to verify that the used distances recognize the correct pattern when both tags fit. In the creation process, if a syntactic variation of a pattern tag *t* fits another pattern tag *t'*, the candidate tags obtained from *t'* are addressed to *t* and *t'* is deleted. After the whole process, DS1 contains 8,806 different pattern tags and 39,255 tuples (*pattern, candidate*) to check. DS2 contains 5,000 tags not included as pattern tags in DS1. These tags are used in 122,394 annotations representing the 6% of the total amount of annotations. We create a dictionary with the 8,806 pattern tags contained in DS1. This dictionary is used to perform the Levenshtein and Hamming distances over DS1 and DS2 datasets. We use a trie structure to store the dictionary when computing the distances.

Methodology - The identification of syntactic variations of tags becomes useful whenever: (1) the pattern matching techniques used ensure a high recognition rate of tags, which are variations of an existing one; and (2) identify, with a high degree of success, new tags that do not fit any existing one. The goal is to maximize the number of syntactic tag variations identified without conditioning the recognition of new tags. In our experimental scenario, the goal is to maximize the number of correctly identified tuples <*pattern, candidate*> on DS1; and to maximize the number of tags identified as new tags on DS2.

To perform this evaluations a discriminator is used. This discriminator consists of the dictionary created from DS1 and a distance (Hamming or Levenshtein). The discriminator accepts as input one tag and checks it against the tags defined in the

[1] http://www.eslomas.com/index.php/publicaciones/tagspatternmatching

dictionary. As output the discriminator provides two values, (1) the most similar tag in the dictionary to the provided one, and (2) the distance value. We denote in the following by *candidate* the input at the discriminator, and by *pattern'* its output as depicted in Fig. 4.

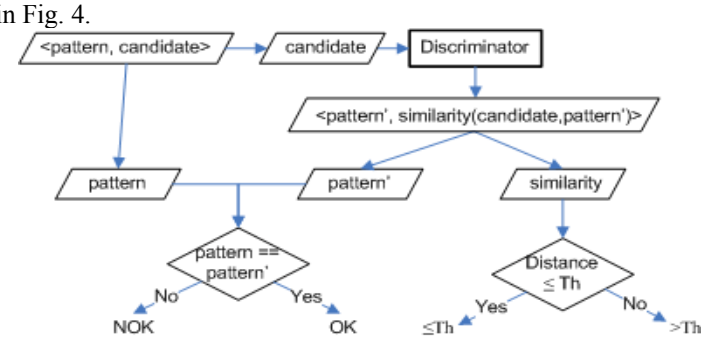

Fig. 1. Methodology schema for DS1 data set evaluation.

We take the *candidate* tag from each tuple in DS1 and we apply Levenshtein and Hamming distances over the dictionary at the discriminator. As output we get a corresponding *pattern'* and its distance to the *candidate*. Note that the algorithm could select a pattern tag (*pattern'*) different to the correct pattern (the associated to the *candidate* tag in DS1). We denote by *OK* the case when the tag selected by the algorithm (*pattern'*) is the *pattern* associated to *candidate* tag in DS1. That is, the algorithm selects the expected pattern tag. We use *NOK* to describe the case when *pattern'* and *pattern* do not fit.

A threshold level, called *Th*, determines the accuracy of the discriminator. The *candidate* tag is classified as a syntactic variation of the *pattern'* tag if the discriminator provides a distance less than or equal to this threshold (see Fig.4). Thus, for example *NOK & ≤TH* indicates that *candidate* and *pattern'* tags match with a low distance between them, but *candidate* derives from *pattern* tag (in DS1) which is different to *pattern'*. Dealing with the problem of new tags identification, when the distance values obtained for the tags contained in DS2 (*candidates*) are greater than *Th*, tags are considered as new pattern tags (*New*). In other case tags will be considered as syntactic variations (*Not New*) of the *pattern'* provided by the discriminator.

3 Experimental Results

Table 1 shows the results of processing data sets DS1 and DS2 with the Levenshtein and Hamming distances, using the dictionary with the 8,806 distinct tags of DS1. Results on DS1 are shown in four different columns, representing if the correct pattern has been identified for each candidate string (*OK*) or not (*NOK*), and if the distance is less than or equal to a determined threshold (*≤Th*). Threshold values for each distance have been set to 1.0 experimentally, so only one edit operation by string is considered. Table 1 shows that Hamming distance is more restrictive at the recognition of pattern variations in DS1 (*OK & ≤Th*) and that it does not overcome

distance threshold in more cases than the Levenshtein distance. This shows that Hamming distance trends to identify less variations and therefore to identify them as new tags. It corresponds with the results on DS2, where we can appreciate a greater identification ratio than the obtained for the Levenshtein distance.

Table 1. Process results on datasets DS1 and DS2.

	DS1				DS2	
	NOK & >Th	NOK & ≤Th	OK & >Th	OK & ≤Th	> TH (New)	≤TH (Not New)
Hamming	3,095	2,152	8,127	25,881	4,162	838
Levenshtein	1,641	2,516	7,767	27,331	4,016	984

A breakdown of these results based on the different variation types is presented in Table 2. The variation labelled as *Self,* represents that both pattern and candidate are the same strings, so the distance is null and both Hamming and Levenshtein distances should identify them always correctly. Table 2 shows that both distances identify correctly the pattern and candidate strings when they match (*Self / OK & ≤TH*).

Table 2. Breakdown of the results of Hamming and Levenshtein distances by variation type.

	NOK & >Th		NOK & ≤Th		OK & >Th		OK & ≤Th	
	Ham.	Lev.	Ham.	Lev.	Ham.	Lev.	Ham.	Lev.
Self	0	0	0	0	0	0	8,788	8,788
Plural/Singular	233	253	523	617	928	896	7,291	7,209
Typo error	65	21	669	762	24	60	8,195	8,110
Transposition	1,022	1,358	934	1,108	6,807	6,280	6	23
Separators	1,775	9	26	29	368	531	1,601	3,201
Total (#)	3,095	1,641	2,152	2,516	8,127	7,767	25,881	27,331
Total (%)	7.88%	4.18%	5.48%	6.41%	20.70%	19.79%	65.93%	69.62%

Regarding the failures, both distances provide similar results for the other syntactical variations considered, except for substitution/deletion of separator, where Levenshtein-distance performs better.

Transposition variations imply the existence of two edit operations, so a threshold value of 1.0 avoids the identification of this kind of variation. The greater number of results corresponds to *OK&>Th*, representing that they are able to identify correctly the original pattern, but with a high distance.

One way to deal with this situation is increasing the threshold, however this implies the acceptance of a higher number of edit operations, and this would produce an increase in the incorrect identification of variations among short tags. For example, a threshold value of 2.0, would allow to identify as variations many of the transposition cases, but would affect negatively in the identification of new tags on DS2 for both distances. In the case of Hamming distances, identifications as new tags (*>Th*) decreases from 4,162 to 3,281 and in the case of Levenshtein, from 4,016 to 2,957. Another way is to consider transposition operations as a unique and atomic edit operation, assigning a unitary cost as occurs in [11].

Both distances provide good results identifying variations originated by typographic errors in which one letter has been replaced by another, providing a distance lower or equal than the threshold value. Results differ when separators are changed by other characters. Hamming distance is negatively affected when separators inside tags are deleted. In addition, both perform well in case of plural/singular syntactic variations. Both distances also provide similar results for more complex plurals, as (*library* and *libraries*). In these cases the distance values are greater than the threshold value.

Regarding the lengths of candidate strings, Fig. 2 shows the identification errors between *candidate* and *pattern* strings (*pattern'* ≠ *pattern*), attending to the candidates string length. It shows that in both cases (Hamming and Levenshtein) the greater number of errors happens for tags with lengths of 3 and 4 characters. The reason is that any variation in short length strings can produce the resultant strings will be more similar to other patterns distinct to the original. It can be also seen that Hamming distance has lightly less identification errors for lengths in the range between 4 and 7, and worse results from this point.

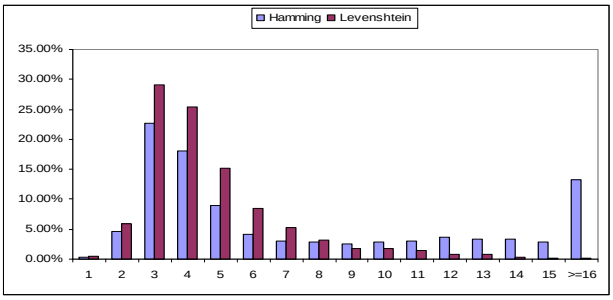

Fig. 2. Failures of Hamming and Levenshtein distances on DS1 per candidate string length

Regarding identification errors for string lengths between 4 and 7, Levenshtein distance get lightly worse results because it is not so much affected by the deletion or insertion of new characters as Hamming distance. For instance, the candidate string *vace*, generated from pattern *face* with a typographic error changing the first character. Hamming distance gets the correct original pattern *face*, but Levenshtein distance gets *ace* as the nearest pattern.

However, Hamming distances starts to increase the number of incorrect identifications in comparison with Levenshtein distance, from the pattern length 8. The reason is due to the fact that from these length is more frequent the use of separators in tags, and Hamming is more affected by the elimination of characters in the middle of patterns. So for instance, for the candidate string *websearch*, generated from the pattern *web_search*, Levenshtein identifies correctly the original pattern, but Hamming distance gets as results the pattern *substance*.

With the objective of ignoring strings with lengths less than or equal to 3, information about the amount of tags in the initial dataset of CiteULike has been obtained.

Based on these data, trying to identify variations only for lengths greater than 3, the distances would be used for a 95.37% of the tags in the initial CiteULike data set,

(91.07% of the total number of annotations). Table 3 summarizes the results obtained for DS1 and DS2 data sets, and ignoring tags with length less than or equal to 3.

Table 3. Process results on DS1 and DS2 ignoring lenghts <=3

	DS1				DS2	
	NOK & >Th	NOK & ≤Th	OK & >Th	OK & ≤Th	>Th (New)	≤ Th (Not New)
Hamming	2,966	587	8,111	24,431	4,171	418
Levenshtein	1,539	871	7,751	25,934	4,076	513

It shows that the number of incorrectly identified patterns with distance less than or equal to the threshold (*NOK&≤Th*) has been reduced: from 2,152 to 587 (72.73%) in the case of Hamming distance, and from 2,516 to 871 (65.38%) in the case of Levenshtein. Regarding results on DS2, it shows that both distances keep the number of correctly tags identified as new, while they reduce the number of tags identified as known tags (*Not New*): from 838 to 418 in the case of Hamming distance and from 984 to 513 in the case of Levenshtein. This shows that both distances were detecting tags with lengths less than or equal to 3 as variations of some known tag. Table 4 shows a breakdown of the results on DS1 based on the different variations.

Table 4. Results of Hamming and Levenshtein distances by variation type ignoring lengths ≤3.

	NOK & >Th		NOK & ≤ Th		OK & >Th		OK & ≤ Th	
	Ham.	Lev.	Ham.	Lev.	Ham.	Lev.	Ham.	Lev.
Self	0	0	0	0	0	0	8,088	8,088
Plural/Singular	172	201	180	220	916	884	6,827	6,790
Typo error	54	13	175	248	24	59	7,918	7,851
Transposition	970	1,318	216	381	6,805	6,279	1	14
Separators	1,770	7	16	22	366	529	1,597	3,191
Total (#)	2,966	1,539	587	871	8,111	7,751	24,431	25,934
Total (%)	8.22%	4.26%	1.63%	2.41%	22.47%	21.47%	67.69%	71.85%

It can be seen that the number of identification errors has been reduced in the three types of variations where we get more errors previously: plurals/singulars, typographic errors and transpositions. The greater reduction in associated to transposition variations: from 934 errors to 216 in the case of Hamming distance and from 1,108 to 281 in the case of Levenshtein. The reason is that any transposition in short patterns has high probability to generate another different pattern with less distance.

6 Conclusions

In this work we have analyzed the performance of two pattern matching techniques to identify syntactic variations in folksonomies. We have performed the analysis over a

large dataset in two different ways: identifying (i) pattern-candidate combinations and (ii) new tags. Experiments show that both techniques provide similar results for some syntactic variation types as typographic errors and simple plurals/singulars, but Levenshtein gets significantly better results than Hamming identifying variations based in the insertion/deletion of characters. However both techniques do not perform as well as desired when identifying variations based in the transposition of adjacent characters or some kind of singulars/plurals (*library/libraries*). Moreover, both techniques improve their results ignoring candidate tags with lengths less than four.

These results show that this techniques can be used to identify syntactic variations of tags, though they should be adapted to perform better with some variation types as plurals/singulars and transpositions of adjacent characters.

Acknowledgements

Research supported by the Spanish Research Council TIN2006-14738-C02-02.

References

[1] Vander Wal, T.: Folksonomy, http://vanderwal.net/folksonomy.html
[2] Millen, D.R., Feinberg, J.: Using Social Tagging to Improve Social Navigation. In: Workshop on the Social Navigation and Community based Adaptation Technologies, Dublin, Ireland, June (2006)
[3] Golder, S.A., Huberman, B.A.: The Structure of Collaborative Tagging Systems. Journal of Information Science, vol. 32, no.2, pp. 198-208 (2005)
[6] Gruber, T.: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, vol. 5, no.2, pp. 199-220 (1993)
[4] Echarte, F., Astrain, J.J., Córdoba, A., Villadangos, J.: Ontology of Folksonomy: A New Modeling Method. In: Semantic Authoring, Annotation and Knowledge Markup (SAAKM 2007), Whistler, British Columbia, Canada October 28-31, 2007
[5] Mathes, A.: Folksonomies - Cooperative Classification and Communication Through Shared Metadata. Computer Mediated Communication, Dec (2004)
[6] Guy, M., Tonkin, E.: Folksonomies - Tidying up Tags? DLib Magazine, 12, vol. 1 (2006)
[7] Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals, Soviet Physics Doklady, vol. 10, no.8, pp. 707–710 (1966)
[8] Hamming, R.W.: Error Detecting and Error Correcting Codes, Bell System Technical Journal, vol. 26, num 2, pp. 147-160 (1950)
[9] Specia, L., Motta, E.: Integrating Folksonomies with the Semantic Web. In: European Semantic Web Conference (ESWC 2007), The Semantic Web: Research and Applications. LNCS 4519, pp. 503—517, Springer. Heidelberg (2007)
[10] Navarro, G.: A Guided Tour to Approximate String Matching, ACM Computing Surveys, vol. 33, no.1, pp. 31–88 (2001)
[11] Oommen, B.J. Loke, R.K.S.: Pattern recognition of strings with substitutions, insertions, deletions, and generalized transpositions, Pattern Recognition, vol. 30, no. 5, pp. 789–800 (1997)