Generating a Topic Hierarchy from Dialect Texts

Wim De Smet
ICRI-LIIR
K.U.Leuven, Belgium

Marie-Francine Moens
ICRI-LIIR
K.U.Leuven, Belgium

Abstract

We built a system for the automatic creation of a text-based topic hierarchy, meant to be used in a geographically defined community. This poses two main problems. First, the appearance of both standard language and a community-related dialect, demanding that dialect words should be as much as possible corrected to standard words, and second, the automatic hierarchic clustering of texts by their topic.

The problem of correcting dialect words is dealt with by performing a nearest neighbor search over a dynamic set of known words, using a set of transition rules from dialect to standard words, which are learned from a parallel corpus. We solve the clustering problem by implementing a hierarchical co-clustering algorithm that automatically generates a topic hierarchy of the collection and simultaneously groups documents and words into clusters.

1 Introduction

We developed a system that automatically creates a topic hierarchy from text documents without metadata, where the community is geographically defined to the city of Hasselt, a town in Belgium. Inhabitants of this town are invited to write about the events in town, shops, crimes, gossip, restaurants, etc. These community texts are then made accessible for the other inhabitants in the form of a *folksonomy*, a hierarchical collection of documents. However, its automatic creation poses several problems.

First, the appearance of both standard language and a community-related dialect demands that dialect words should be as much as possible corrected to standard words. Dialects are variations of a language spoken by a larger community, bound to a particular geographic area. While the official language of a country, dictated by an authority, is often regarded as superior, linguistically seen it is just one of the possible dialects. To avoid confusion, we will refer to the standardized version as ”standard language”. ”Dialect” or ”dialect language” denotes any other form that is not the standard language. In our work we corrected dialect words by performing a nearest neighbor search over a dynamic set of known words, using a set of transition rules from dialect to standard words, which are learned from a parallel corpus of standard and dialect words.

The second problem regards the automatic recognition of topical information and hierarchically clustering of the community texts based on this information. We implement a hierarchical co-clustering algorithm that automatically generates a topic hierarchy of the collection and that simultaneously groups documents and words into clusters. The co-clustering has the property of grouping related terms, which is especially valuable when processing texts that are a mixture of standard and dialect terms.

The remainder of this paper is organized as follows. In section 2, our technique to “standardize” dialect words is explained. The next section 3 describes our method to assign topic information to the text documents and to cluster the documents. We present our conclusions in section 5.

2 Handling Dialect Words

The main difference between the standard language and dialects is the consensus on spelling. As standard language is supported by authorities, it is required to have a written equivalent to enable official communications. Dialects on the other hand, usually have only a spoken component. Written versions appear however when a touch of “couleur locale” is wanted, as is probable in a city-based folksonomy. Because of the lack of consensus on spelling, dialect words are not apparent in standard lexicons. A first category of dialect words have the same origin of standard words, but are pronounced differently and their written equivalent mimics the dialect pronunciation. In other cases, completely new words appear in dialects that do not necessarily have a one-to-one translation with standard words.

Because in our clustering model we rely on the words of the texts, it is vital that words are as much as possible normalized to standard language. Noise in words, caused for example by spelling errors, deteriorate the recall of finding related documents, based on common terms. Dialect words cause similar problems.

When handling dialect words, the algorithm starts from a initial list of known words correctly spelled (dictionary list). Dialect texts are processed and upon sufficient similarity of a word with a known word, it is normalized to this standard word. In all other cases a new, previously unknown word is added to the dictionary list. We'll denote the process of correcting, normalizing or adding an unknown word as *resolving* that word.

2.1 Algorithm

We keep two different lists of information. The first is D, a list of words whose spelling is assumed correct (dictionary list). Initially, we populate this list with words from a dictionary. When new words with a correct spelling are found, they are added to this list.

The second list L, contains every word apparent in the corpus together with their corpus frequency and inverse document frequency.

When processing a text, our goal is to resolve unknown words and expand D with words we assume spelled correctly. An unknown word (i.e., $\notin D$), can belong to one of the following categories:

1. it is a standard word, but does not occur in D yet;
2. it is the misspelled version of a word in D;
3. it is a dialect word. Since dialect words do not have an official spelling, we will not distinguish between "correct" and "misspelled" dialect words.

To decide how we will resolve the unknown words from a text, we first parse the document, and iterate over every single word t, applying the following algorithm.

If t occurs in D, it needs no resolving, so we proceed to the next word. If it does not occur in D, it belongs to one of the previous categories. First, we check for categories 1 and 3. If the corpus frequency of t (the number of times t appears in the corpus) is higher than a given threshold, we rule out the possibility that it is coincidentally mistyped, and assume it is correctly spelled. We then add it to D. This is the *frequency assumption*.

If the frequency assumption does not hold, we try to find the word in D which most likely was meant by the author. To discover this, we search for words with optimal *Dialect Edit Distance* from t. Dialect Edit Distance (ded), an adaptation from the normal Edit Distance to take into account typical dialect transformations, is further explained in section 2.2. Here it suffices to state that it measures the similarity of the spelling of two words, specifically if one of them is a dialect word.

When a non-frequent, unknown word is encountered, we calculate the ded to every term in D, and assume the word associated with the smallest distance (or a random selection from several words with the same, smallest distance) is the word meant by the author. If the system should be used in an interactive environment, then the n smallest ded's can be retrieved, offering a selection to the user.

If the smallest ded from D stays below a ded-threshold, the word associated with that distance is selected as the resolution word. At this point, a word can also be added to D because of the *edit distance assumption*. If both exceed this threshold however, we assume it unlikely that a simple spelling error occured, and therefore add t to D.

2.2 Dialect Edit Distance

In case the dialect and standard word have the same roots, the dialect word has evolved to a different spelling by following a set of rules that determine the pronunciation. These rules comprise for example contractions, alterations of vowels, ... and are often (but not always) dependent on the context of letters they appear in. They are however not exclusive: the same phonetic entity can be expressed by several combinations of characters. For example, the Dutch vowel "o", would in the Hasselt dialect be written as "eu" or "ö", both resulting in the same vowel (different from the Dutch "o").

To be able to incorporate these rules, we adapt the edit-distance algorithm, as developed by Levenshtein [2]. The Levenshtein-distance calculates the minimal number of character operations, necessary to transform one word into another. In the original paper, three different operations are considered: *deleting* a character, *inserting* a character and *substituting* one character for another. With each operation adding a cost of 1, the algorithm creates a matrix to derive which order of operations generates the least total cost.

Our adaptation gives each operation a dynamic cost, depending on the type of operation and the characters involved. This allows for alterations, typical for the dialect, to cause only a small difference between a dialect word and the standard word. When comparing an unknown dialect word with a list of standard words, the word that yields the smallest dialect edit distance, is likely to be the original standard word.

2.2.1 Calculation of costs

To learn the cost of each different operation, we make use of an aligned corpus of dialect words and their standard equivalent.

After calculating the edit-distance matrix between every pair, we remove the pairs whose distance-over-word length ratio exceeds a threshold, as they are not likely to be phonetically related.

For the remaining words, the distance matrix provides the alignment where the two words concur, and which operations transform the dialect word to the standard word. For each operation O, we store its type (substitution s, insertion i or deletion d), its parameter letters (only x for insertion and deletion, x and y for substitution), and the context C(an n-letter two-sided window around the *dialect* letter). If the beginning or the ending of the word falls inside the context, then this is explicitly stored. We represent it as a following vector: $[x_{i-n}, \ldots, x_i, \ldots, x_{i+n}]$. Because the width of our context window is arbitrarely chosen, we might "overencode" information. If an operation typically occurs after one specific letter, we may loose information by storing two letters extra before, and two letters after. Therefore we also store the following subsets: $\forall v, w \in [0, n] : [x_{i-v}, \ldots, x_i, \ldots, x_{i+w}]$.

There are several options to calculate the operation's cost. This cost must lie within $[0, 1]$, and be inversely proportionate to the confidence. One way to estimate this confidence is by the formula $conf(O_C) = \frac{|O_C|}{|C|}$, $|O_c| =$ being the number of times the operation, together with context C appears in the corpus, and $|C| =$ being the number of times context C appears in the corpus alltogether. The cost-function $f(|O_C|, |C|)$ can then be expressed as $1 - conf(O_C)$.

We feel however that this relative frequency does not capture all information. For example, two operations both have a relative frequency of 50%. The first however has an $|O_C|$ of 2 and a $|C|$ of 4, while the other one has an $|O_C|$ of 20 and a $|C|$ of 40. Because operation 2 occurs 10 times more than operation 1, we wish to give it a higher confidence, and a lower cost. The cost-function we designed has for this purpose has the following characteristics. If $|O_C| = 0$, the operation did not occur in the training data, and its cost equals 1. If $|O_C| = |C|$ ($|O_C|$ cannot exceed $|C|$), it receives the minimum cost, associated with the value of $|O_C|$. This cost decreases monotonally on increasing $|C|$. For values of $|O_C|$ between 0 and $|C|$, we calculate the quadratic interpolation between 1 and the minimum cost.

We defined the monotonically decreasing function for ($|O_C| = |C|$) as

$$g(|C|) = 1 - \frac{1}{1 + \log(1 + |C|)}$$

which leads, together with the quadratic interpolation to

$$f(|O_C|, |C|) = g(|C|) * \frac{|O_C|^2}{|C|^2} - 2 * g(|C|) * \frac{|O_C|}{|C|} + 1$$

3 Topic hierarchy generation

Instead of explicitely extracting tags or keywords from documents, we try to build the topic hierarchy by grouping documents related by their topics into clusters, and create a hierarchy of these clusters. We associate representative words to each cluster. This way, the cluster hierarchy is translated into a topic hierarchy

The method we used for this is a co-clustering algorithm. In typical clustering algorithms, documents are clustered based on word similarity (i.e. how many words they have in common), and words are clustered based on their document similarity (i.e. how many documents contain the words simultaneously). The resulting clusters usually exhibit good characteristics, related words and related documents occur in the same clusters. There is no relationship however, between document clusters and word clusters, as they have been calculated independently. Co-clustering algorithms resolve this problem by alternating every refinement step between document clustering based on already found word clusters, and word clustering by already found document clusters.

An alternate hierarchical co-clustering algorithm has been developed by Dhillon in [1]. It transforms the co-clustering problem into a minimal cut problem. The algorithm is presented in figure 1, and has been borrowed from [3]. In this paper a hierarchy of forum posts are ordered into a hierarchical, topic-based clustering.

The document-by-word matrix A is generated by extracting all words from each document. Stop words are first removed from this matrix based if they have a low idf-value. Two different weight values for the matrix' entries have been tested: the $tf \cdot idf$-value of the word, and a constant value 1. Neither of these values yielded significantly better results in our implementation, so the constant value was chosen.

The resulting hierarchy should reflect the ordering of topics from high-level to low-level, where each cluster contains documents relevant to this topic, and associated words that can describe the cluster.

4 Evaluation

4.1 Dialect Edit Distance

To evaluate the dialect edit distance, we used an dictionary mapping a lexicon D of standard words with correct spelling (among which the standard words from T). The particular dialect-standard dictionary used in this evaluation was composed by a folklore organization, dedicated to the preservation of the Hasselt (and related) dialect. Not every dialect, however, will have such a group among its speakers. Therefore, we evaluated our algorithm by training it on several, smaller samples of the dictionary.

We tested the accuracy of the dialect edit distance by selecting two disjoint subsets from the corpus: *training data* and *test data*. From the training data, a set of dialect rules

1. Given the $m * n$ document-by-word matrix $\mathbf{A}$, calculate diagonal help-matrices $\mathbf{D_1}$ and $\mathbf{D_2}$, so that:

   $$\forall 1 < i \leq m : \mathbf{D_1}(i,i) = \sum_j \mathbf{A}_{i,j}$$

   $$\forall 1 < j \leq n : \mathbf{D_2}(j,j) = \sum_i \mathbf{A}_{i,j}$$

2. Compute $\mathbf{A_n} = \mathbf{D_1}^{-1/2} * \mathbf{A} * \mathbf{D_2}^{-1/2}$
3. Take the SVD of $\mathbf{A_n}$: $SVD(\mathbf{A_n}) = \mathbf{U} * \mathbf{\Lambda} * \mathbf{V}^*$
4. Determine k, the numbers of clusters by the *eigengap*: $k = \arg\max_{(m \geq i > 1)}(\lambda_{i-1} - \lambda_i)/\lambda_{i-1}$, where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m$ are the singular values of $\mathbf{A}$
5. From $\mathbf{U}$ and $\mathbf{V}$, calculate $\mathbf{U}_{[\mathbf{2},\cdots,\mathbf{l+1}]}$ and $\mathbf{V}_{[\mathbf{2},\cdots,\mathbf{l+1}]}$ respectively, by taking columns 2 to $l+1$
   where $l = \lceil \log_2 k \rceil$,
6. Compute $\mathbf{Z} = \begin{bmatrix} \mathbf{D_1}^{-1/2}\mathbf{U}_{[\mathbf{2},\cdots,\mathbf{l+1}]} \\ \mathbf{D_2}^{-1/2}\mathbf{V}_{[\mathbf{2},\cdots,\mathbf{l+1}]} \end{bmatrix}$ and normalize the rows of $\mathbf{Z}$
7. Apply k-means to cluster the rows of $\mathbf{Z}$ into k clusters
8. Check for each clusters the number of documents. If this is higher than a given treshold, construct a new document-by-word matrix formed by the documents and words in the cluster, and proceed to step 1

Figure 1. The hierarchical co-clustering algorithm for automaticaly generating a topic hierarchy

was extracted, as explained in section 2.2. Our test data consisted of 500 words, selected with two different properties. One set was sampled from the dictionary's dialect words without restriction. The second set contained only dialect words that have an edit distance of maximally 0.5 times the dialect word's length. Not every word in the dictionary is a phonetical transformation of its standard word, some words are (phonetically) unrelated. Those words can not be resolved to the standard word using the Dialect Edit Distance. Therefore, by restricting the test data to word-pairs that are relatively similar, the second test set was used to evaluate the Dialect Edit Distance's performance in se. The first test set evaluated the performance in a real life setting.

Using the acquired rules rules, for every dialect word d from the test data, we determined the word from L with the smallest dialect edit distance (or select one random in case of multiple smallest distances) and compared it to the standard word associated with d. We repeated this procedure for different sizes of the training data. The test data size remained the same. To provide comparison with a baseline approach, we repeated the experiments using the standard edit distance. Results can be seen in tables 1 and 2.

# Training data	Edit Distance	Dialect Edit Distance
500	26.4%	39.6%
1000	28.2%	43.2%
2000	27.4%	46.0%
5000	27.2%	41.2%

Table 1. Accuracy of resolving 500 dialect words without restriction, using different sizes of training sets

# Training data	Edit Distance	Dialect Edit Distance
500	40.8%	62.6%
1000	41.2%	63.6%
2000	37.8%	61.8%
5000	37.0%	64.8%

Table 2. Accuracy of resolving 500 dialect words with an edit distance smaller than 0.5 times the dialect word length, using different sizes of training sets

Some remarks on the results:

1. The Dialect Edit Distance has an average improvement of 15% accuracy over the standard Edit Distance in the real life setting, and an average improvement of 24% in case of resolvable dialect words.
2. Accuracy was based on the comparison of the single word with lowest Dialect Edit Distance. If we relax this constraint and look among n best words, we expect an improvement of these results. These n best words can be used for term expansion, as explained in section 4.2.

4.2 Topic hierarchy generation

The tested hierarchy was generated using 300 community texts from the city of Hasselt. The results from the hierarchical clustering algorithm were disappointing. The main problem consisted of the algorithm's inability to calculate a satisfying value for k, the number of clusters, when using the eigengap method. In our experiments, the 300 texts were clustered into 297 clusters in the first level, making a hierarchy impossible, and leaving only a handful of documents joined in the same cluster. When choosing a fixed value for k, it was shown that documents joined in a single cluster often share a topic, although recall was low (other related documents were in different clusters), and equally often non-related words shared a cluster.

Topic hierarchy generation is a difficult task. First of all there is the difficulty that clustering based on a bag of words often results in unsatisfactory performance. We often create sparse document vectors from the texts that have little overlap, and if there is some overlap, the overlap might be realized with words that are not salient with regard to the content of a text and which do not occur with a sufficient frequency in order to be filtered out with an inverse document frequency metric. Such a situation results in a low precision of the clusters. The problem is especially critical, if one wants to hierarchically cluster texts at various topical levels. Effective techniques for detecting words at different levels of salience is only possible when integrating natural language processing techniques that allow to recognize salience at the discourse and sentence level. Because we do not have access to part-of-speech taggers and sentence parsers for dialect texts or for texts that are a mixture of dialect and standard words, we could not implement such an approach. There is also the problem of a low recall, i.e. documents are prevented from clustering if the same content is signaled by different words. This is especially a problem if texts are a mixture of standard and dialect words. Although by using the dialect edit distance (see above), we could already normalize a part of the variations, there is still the problem of distinct words that have the same meaning. The hierarchical co-clustering algorithm was partly successful in solving this problem. However, we could make additional improvements. First of all, using the eigengap for detecting the number of clusters did not yield an accurate estimate of the number of clusters. The (relative) differences between all eigengaps were quite similar up to already a large k value. This is due to the poor overlap between the documents, making that no correlation or dependency can be found between them. We hope by increasing the number of documents and consequently the number of related terms, this method yields better results. Alternatively we could use another heuristic for estimating the natural number of clusters in the collection.

As a possible solution to the poor overlap, we propose term expansion in case of unknown words. Instead of using the unknown word, or the single resolved word, we add the n words with smallest ded to the document. This will create overlap in case of dialect words (in case of variant dialect words, or one dialect and one standard word). To compensate for the increased amount of words, we can give these expanded terms a lower value in the documents-by-words matrix. We suspect that related documents will benefit from this increased overlap in addition to words they already have in common. The similarity of non-related documents will increase also if they have similar dialect words in common, but because they will not share other terms (as they are not related), the total distance will remain limited.

5 Conclusion

We tackled the difficult problem of automatically generating a topic hierarchy from dialect texts. We were able to fairly well normalize dialect words to their equivalent terms in standard language. A standard way to create a topic hierarchy is to group related documents and extract a suitable description from each cluster. Because of the large variety in the vocabulary (not all dialect words have an equivalent term) co-clustering the documents and terms simultaneously seemed to us a suitable technique in an attempt to correlate distinct dialect and standard terms with a related meaning. The research has shown that still many research questions demand valuable solutions. Texts that are casually generated often use a living language that dynamically changes, which cannot be easily processed with current means. The importance of this kind of research will only grow as people increasingly chat, communicate and gossip through digital means.

6 Acknowledgements

The research presented in this paper has been carried out as part of the project $A4MC^3$, *Architectures for Mobile Community Content Creation*. This project is supervised by the Belgian IBBT, the Interdisciplinary Institute for BroadBand Technology, a research institute founded by the Flemish Government. We thank them for their financial support during this research.

Furthermore, we also like to thank Concentra, a Belgian publisher, for their good cooperation during the project and for providing us with test data.

References

[1] Inderjit S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Knowledge Discovery and Data Mining*, pages 269–274, 2001.

[2] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.

[3] Gu Xu and Wei-Ying Ma. Building implicit links from content for forum search. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 300–307, New York, NY, USA, 2006. ACM Press.