Real-time unsupervised classification of web documents

Anthony Sigogne, Mathieu Constant

Proceedings of the International Multiconference on Computer Science and Information Technology pp. 281–286
ISBN 978-83-60810-22-4
ISSN 1896-7094

Real-time unsupervised classification of web documents

Anthony Sigogne, Matthieu Constant
Université Paris-Est
Laboratoire d'Informatique Gaspard-Monge
5, bd Descartes, Champs-sur-Marne, 77454 Marne-la-Vallée, France
Email: {asigogne,mconstan}@univ-mlv.fr

***Abstract*—This paper adresses the problem of clustering dynamic collections of web documents. We show an iterative algorithm based on a fine-grained keyword extraction (simple, compound words and proper nouns). Each new document inserted in the collection is either assigned to an existing class containing documents of the same topic, or assigned to a new class. After each step, when necessary, classes are refined using statistical techniques. The implementation of this algorithm was successfully integrated in an application used for Information Intelligence.**

I. Introduction

INFORMATION on the web continously evolves as new documents on new topics are made available everyday. Nevertheless, standard classification algorithms are usually best-suited for static collections of documents and are not optimized for dynamic ones. Indeed, generally, each time a new document arrives, the representations of all classes have to be recalculated. This paper adresses this problem by describing a fine-grained clustering system on dynamic sets of web documents. Its implementation is driven by real industrial needs in Information Intelligence. It requires a real-time classification procedure as thousands of new documents arrive everyday. Classes (or clusters) must correspond to precise events or topics. They are not predefined in advance as news constantly change.

In this paper, we show an *iterative algorithm* based on a fine-grained keyword extractor taking compound and proper nouns into account. Each new document processed is either assigned to an existing class containing documents of the same topic, or assigned to a new class if no classes match the document topic. After each step, when necessary, the modified classes are refined using statistical techniques by [1]. Our method can be compared to [5], that was also implemented with strict time efficiency requirements.

Our method is different from standard methods using *k-means* algorithm and its extensions, that assum a fixed number (k) of classes and have to recompute classes at each step. Such approach is unappropriate to our problem. Standard agglomerative algorithms are also hardly applicable because in our system all documents are not known initially[1].

This work was partly funded by the company Xeres.

[1]Note though the experiments in [2] on the use of agglomerative algorithm for real-time clustering.

The paper is divided into 4 sections. First, we describe how a document is represented in our system, on the basis of a keyword extraction process. Then, we detail our real-time classification algorithm. Next, we show some implementation details on our system, that is, finally, evaluated.

II. Document representation

In the Space Vector Model (SVM) approach, documents are represented as vectors. Each dimension corresponds to a unique term in a collection of documents. For a given document, a dimension is the importance weight of the corresponding term in the document and generally depends on the frequency of the term. For space and speed efficiency reasons, it is often of great interest to limit the number of terms with non-zero weights. In the following, we call these terms keywords. This section is devoted to the extraction and the weigthing of keywords for each web document to be classified.

A. Keyword candidate recognition

Keywords are usually limited to stemmed or lemmatized word-tokens. More and more studies show that the use of more complex forms like named entities (e.g. proper nouns) or word bigrams (trigrams) may improve the system quality. In this paper, we propose to augment the set of standard terms with compound nouns and proper nouns. Compound nouns are recognized by dictionnary look-up and proper nouns by local grammar application. For compounds, we used Unitex large-scale morphological resources [4] available for various languages. These resources were built manually since the 90s by linguists. They also indicate the lemma of each word, so it is possible to lemmatize each compound word: e.g. complex lemma *fried patato* would stand for compound *fried patatoes*. Resources also include information on internal structures: this allows for selecting compounds with specific internal structures. Proper nouns are recognized with the use of a local grammar in the form of a finite-state graph [3], [8]. The graph roughly represents sequences of capitalized words like *John Smith*. Sequences like *Marne-la-Vallée*, *Mohammed al Cherif* are also recognized. These grammars are contextualized so that utterance *In Paris* located at the beginning of a sentence, should not be considered as a proper noun. The named entity recognized should be limited to *Paris*.

B. Keyword filtering and weigthing

In order to improve efficiency, we limited document terms to relevant keywords by filtering candidate terms with a mixed linguistic and statistical approach. First, it is well-known that the main semantic information is contained in nouns, verbs and adjectives of texts. Keywords are even often reduced to nouns. In our case, we applied the morphosyntactic tagger *TreeTagger* [7] and kept only the lemmas of the nouns in the document.

This linguistic process is not sufficient and can be combined with a statistical filtering. Usually, each term is given a weight. For each document, the filtering process then keeps only the terms with the best weights. A very well estabished weighting schema is TF.IDF [6]. For each document j, a term i is assigned a weight w_{ij}:

$$w_{ij} = TFIDF(i,j) = tf_{ij} \cdot \log \frac{N}{N_i}$$

where tf_{ij} is the frequency of term i in document j, N is the number of documents in the collection and N_i is the number of documents where term i occurs. We applied this term weighting schema by using an external collection of documents, because the entire collection to be classified is not known at the beginning of the process. We used a static collection of web documents randomly extracted from news documents on the web. The main drawback with this collection is that it is independent of the main topic of the processed documents and is limited to general language.

In order to deal with unknown words (words occurring in the document processed, but absent in the external collection), we slightly smoothed the term weight formula like in:

$$w_{ij} = tf_{ij} \cdot \log \frac{N+1}{N_i+1}$$

The term frequency tf_{ij} is defined by the number of terms i occurring in document j normalized by the size of document j (number of terms). If a term is not in the external collection, its weight cannot be null in a text where it occurs. Moreover, this enables the weight to be independent of the size of the document processed.

This weighting schema was not entirely relevant for compound nouns. Compound nouns can also be reduced to one or some of their components: e.g. *green card* can be reduced to *card*. In a text block where *green card* occurs, an occurrence of *card* is also potentially a reference to the entity *green card*. Therefore, it sounds relevant to augment the weight of such a term by the weight of its internal nouns (in the example, *card*). So the weight w_{ij} of a compound noun i including k nouns (N_1, N_2, ..., N_k) in a document j is

$$w_{ij} = TFIDF(i,j) + \sum_{k=1}^{n} TFIDF(N_k, j)$$

Note that this weighting schema is not always relevant. For instance, *bank card* cannot be reduced to *bank*. This technique would require refinement by keeping only the head noun. But this is not always accurate or/and sufficient. For instance, French word *vin rouge* (red wine) can be reduced in *rouge* (red). Some compounds cannot be reduced at all: *porte-parole* (spokesman).

The filtering process of simple and compound nouns consists in discarding any of them the weight of which is lower than a threshold. Proper nouns are not filtered. After some experimental tests, we observed that some recurrent non-relevant words (often, terms in the domain of informatics) were not removed by the filtering process. We then manually formed a stop word list including these words in order to refine the process.

For each document, the filtering process results in a vector each dimension of which corresponds to a unique keyword in the collection. Each dimension is the frequency of the corresponding keyword in the document. Although many systems used TFIDF to measure the weight of keywords, we chose to keep the frequency because the TFIDF would depend on a static external statistics independent of the topic of the subjects processed. In addition, after a strict filtering process, the frequency is a good indicator of the importance of a term.

III. Classification

The classification approach uses a single-pass algorithm. It tries to assign a class (an existing one or a new one) to each new document. If the new document has been included in an existing class, this class might need some refinements: (1) merge with others whether they have become very similar to each others or (2) split whether it has become incoherent.

A. Class projection

A class is defined by the set of documents belonging to it. It is represented in the term space by the centroid of the vectors of all its documents. The dimension c_{ij} corresponding to the importance weight of a term i in a class j is defined by:

$$c_{ij} = \frac{\sum_{k \in C_j} w_i k}{K_j}$$

where C_j is class j and K_j is the number of documents in C_j.

Like for document representation, it is of great interest to reduce the size of the term space. For instance, there might be many terms in common amongst classes because all documents are of the same general topic. These terms are useless to discriminate classes. As a consequence, they can be removed from the term space. A way to filter such terms is to use a weighting schema ressembling TFIDF, the normalized gini-index introduced in [1]. It measures the discriminating value of a term in the set of classes.

Let's assume that the collection of documents can be classified in K classes. Let p_i be a function which takes a term as an argument and returns the weight of this term in class i. It is defined as follows:

$$p_i(x) = \frac{\frac{f_i(x)}{n_i}}{\sum_{j=1}^{K} \frac{f_j(x)}{n_j}}$$

where $f_i(x)$ is the number of occurrences of term x in the documents of C_i and n_i is the total number of terms in the

documents of C_i. If a term x is considered as noise, $p_i(x)$ should be close to $1/K$ for all classes.

The normalized gini-index of term x is then defined as:

$$gini(x) = 1 - \sqrt{\sum_{i=1}^{K} p_i(x)^2}$$

For each term x, if gini(x) is lower than a fixed threshold, x is discarded from the term space of all classes.

B. Class assignation

The classification algorithm is a standard single-pass algorithm as in [5]. It classifies one document at a time and assigns it to the most similar class. The similarity is measured by the cosine between the document vector and the class centroid. For each class, if the cosine is greater than a manually-fixed threshold, the class is added to the candidate class list sorted by decreasing similarity to the document. Once all classes have been compared with the document, there exist three cases:

1) No class is candidate. This means that the topic of the new document does not match with the topics of the other classes. We therefore create a new class only composed of the document.
2) One class is candidate. We then assign the document to this class. The class centroid is then recomputed.
3) Two or more classes are candidate. A simple solution is to choose the most similar class. But, [1] showed that, in the case where the difference of similarities is small, there might exist some similarities between the classes, i.e. they share keywords. To best discriminate classes, it is therefore interesting to compare them without their keywords in common. Our selection process consists in taking the two best candidate classes (two first in the list) and applying the comparison method in [1] as it is described below. Once the best class is found, we assign it to the document to be classified and recompute the centroid.

Let S_1 and S_2 be the centroids of the two candidate classes. Filtering a class centroid S_1 (S_2) consists in nulling the dimensions corresponding to words in common between S_1 and S_2. The resulting vector is noted $S_1 - S_2$ (resp. $S_2 - S_1$).

Let's now define a dominance property of a class over another one with respect to a document T. We say that S_1 dominates S_2 if:

- the difference between the similarities of S_1 and S_2 to document T is greater than a manually-fixed threshold Th.
- if this difference is lower than Th, the similarity between $S_1 - S_2$ and T is greater than the one between $S_2 - S_1$ and T.

If S_1 dominates S_2, the class corresponding to S_1 is assigned to the document.

C. Class refinements

When a document is assigned to an existing class, this may cause two types of side-effects: (1) a class can become very similar to another one and it might be useful to merge them together; (2) a class can become incoherent and it might be useful to split it. Standard clustering algorithms like K-means takes this phenomenon into account: they iteratively refine the initial clustering until a certain stability in the cohesion of the clusters is reached.

Algorithm 1 Merging a cluster c with similar clusters in $clusterSet$ with threshold T

```
repeat
  for cluster x ∈ clusterSet do
    if similarity(x,c) > T or c ⊂ x then
      remove x from clusterSet
      c = c ∪ x
    end if
  end for
  add c in clusterSet
until clusterSets is modified
```

In order to refine our clustering after classifying a new document, we used the method described in [1]. We implemented two operations: merging and splitting classes.

1) Merge classes: When inserting a new document in an existing class, its centroid is then modified and the class might have become very similar to another class. They should be merged together. In our system, we consider that two classes should be merged if the cosine between their centroids is greater than a manually-fixed threshold. In that case, all documents of the two classes should be gathered in a single class. The merging method is iterative: the modified class c is successively compared with all classes. If they are similar enough, the class c is augmented with the documents of the other class. A class can therefore be merged with several classes. This process is repeated until the set of classes is stable.

2) Split classes: When inserting a new document in a class, the class might become less coherent and should be splitted. The coherency of a class can be calculated with the intra-class value. It is defined by the following formula:

$$intra(c) = \frac{\sum_{d_i,d_j \in c; d_i \neq d_j} cos(d_i, d_j)}{n}$$

where d_i and d_j are documents in class c and n is the number of documents in c. If this value is lower than a manually-fixed threshold (T_{ic}), the class have to be divided into several sub-classes that would be added to the global set of classes. All documents of this class has to be reassigned to various sub-class by using the method described in subsection III-B. At the initialization stage, an empty set of classes $newC$ is created. Then, we assign each document to the most similar class in $newC$ if their similarity is greater than a manually-fixed threshold T_{sim}. T_{sim} is required to be greater than the one in subsection III-B. If no classes are found for a document, it is added in a new class which is inserted in $newC$.

The detailed algorithm is given in figure 2.

Algorithm 2 Splitting a cluster c in several clusters to be inserted in a set of clusters $clusterSet$, given two threshold T_{ic} and T_{sim}

Require: T_{sim} greater than threshold used in section III-B
  if $intra(c) < T_{ic}$ **then**
    $newC = \{\}$ {empty set of clusters}
    for document $d \in c$ **do**
      $bestc \leftarrow$ findClass(d,$newC$,T_{sim})
      if $bestc$ is empty **then**
        $bestc \leftarrow \{d\}$
      else
        $bestc \leftarrow bestc \cup \{d\}$
      end if
    end for
    $clusterSet \leftarrow clusterSet \cup newC - \{c\}$
  end if

IV. Implementation

A. Overview

The clustering method described above has been integrated in a real-life application for the French company XERES devoted to Information Intelligence. This company continously receives RSS feeds of web documents on specific topics (around 10,000 documents a day) and is required to discover emergent opinions, trends about this topic. As the number of documents to be examined is very high, they need automatic tools extracting the global meaning of each document and classify them in fine-grained clusters.

Our tool takes RSS feeds and clusters of already classified documents as imput. It produces a new set of document classes. It contains three main modules all written in Python (around 1,500 lines of code) and with shell scripts:

- a module extracting keywords from a web documents (around 1,000 lines of code)
- a module classifying web documents (around 500 lines of code)
- a module for the graphical user interface written in Python Tkinter (around 500 lines of code)

The application also calls programs from the platform Unitex [4] in order to apply large-scale and fine-grained linguistic resources. There exists a global configuration file defining the values of the different parameters of the application like manually-fixed threshold defined in the previous sections. These values were determined after some tests on different collections of web documents.

B. Web document preprocessing

Web documents cannot be processed as is. They need two preprocessing stages: extraction of relevant text blocks and automatic language identification for later lexical ressources application.

1) Web page cleaning: The process of extracting relevant text blocks in web documents does not only consists in removing HTML tags, because the main textual information is often surrounded by noise. Indeed, the structure of a web page is nowadays more and more complex. Documents often include navigation menus, hyperlinks to other pages coming with a little summary of their content, advertisement, and so on. More and more tools are devoted to this task as it is shown with the organization of the shared task CLEANEVAL (http://cleaneval.sigwac.org.uk/).

Our module, using the SAX parser to easily read the document, is based on simple assumptions. First, the relevant text is a group of long sentences, to the contrary of navigation menus composed of lists of words or short sentences. Secondly, the relevant text has a minimal size in terms of words. Finally, it is formed of sentences relatively close in terms of tag distance. The more there are tags (line break, scripts, ...) between two groups of sentences, the more the second group becomes irrelevant.

For each group of contiguous sentences, if the group is relevant in terms of size and that it is not too far in terms of tag distance from the last relevant group, then it is selected as relevant. This algorithm is quite simple but we will show in the evaluation section that it works fine.

2) Language detection: Our system uses lexical resources depending on the language of the document. A module of language recognition has been implemented in order to detect it automatically. It is based on an algorithm identifying forbidden factors. Intuitively, for each language, there are some forbidden word factor never or rarely occurring. They can be extracted automatically from representative samples of documents for each language.

C. Web document processing

1) Keyword extraction task: The keyword extraction module takes a url as imput. The document is added to the existing document list. If it already exists, it is not processed to avoid useless operations. The document is then cleaned and its language is detected by the preprocessing stage described above. The module loads TF.IDF statistics and lexical resources associated with the corresponding language. By using the method given in section II, it outputs the best keywords for each type of keywords (simple nouns, compound nouns and proper nouns). This module also extracts a little summary composed of 5 key-sentences of the text. This summarization task is based on the weight of the keywords. We make the standard assumption that the more a sentence contains important keywords, the more it brings information on the text content. The weight of a sentence is the sum of the weight of its keywords.

For each processed document, the keyword extraction module produces an HTML page containing the keywords and the little summary.

2) Classification task: Given an existing set of document clusters (initially empty), a new document is either added to an appropriate cluster or assigned to a new one, by using the algorithm given in section III. Note that our program also removes out-of-date clusters, i.e. clusters that have not been modified for a certain amount of time. An HTML page

TABLE I
EXAMPLES OF CLASSES

Topic	# documents	simple nouns	proper nouns	compound nouns
earthquake in Sichuan and minute of silence in the Olympic Games	2	séisme catastrophe porteur relais	Sichuan Radio-Canada.ca Pékin Parti communiste	flamme olympique minute de silence tremblement de terre politique de réformes
Hurricane in the North of France	8	tornade sinistré urgence logement	Hautmont ministre de l'intérieur Premier ministre France info	solidarité nationale champ de ruines catastrophe naturelle dégâts naturels
Death of alpinists on the K2	3	alpiniste sommet mort mètres	Wilco Van Rooijen Van Rooijen K2 Reuters	haute montagne camp de base bloc de glace tombée de la nuit

containing all clusters is produced. Each cluster is represented by a list of references to its documents (with keywords and summary) and 12 representative keywords taken from the cluster centroid (the 4 best for each type of keywords).

The various thresholds of the program are manually fixed by users in a configuration file. Users can therefore adjust the clustering precision with respect to the topic of the document collection or to the size of the existing clusters. Determining a threshold value requires a little period of testing with a sample collection of documents belonging to the working topic.

After some tests on the program, we decided to make some practical changes to optimize computation cost. First, the merging class step with two-embedded loops has been simplified to one loop because the process tended to merge all documents in one or very few classes. Then, the splitting class process had to divide them again. Most operations were therefore redundant and drastically increased the computing cost. Secondly, in the theoretical algorithm, the gini-index for each term has to be recalculated each time the set of clusters is modified. This operation being costly, we decided to recalculate the gini-index only after 100 new documents are added in the collection. This does not affect the quality of the system because the insertion of a single document very slightly modifies statistics for the gini-index.

V. EVALUATION

Due to the specificity of the task requirements (e.g. creation of very fine-grained clusters), our system can hardly be evaluated automatically on existing collections used for shared-tasks (e.g. TDT). We therefore asked three independent human evaluators to do this task. None of them were involved in any way in the implementation of the tool. The system was only tested on French documents, except for the language detection module (English, French and Spanish).

A. Preprocessing evaluation

As preprocessing stage is not the central point of our paper, the evaluation of this part is very light and limited to few documents. First, the web document cleaning process was applied on 100 documents from different sources (60% from newspaper sites, 20% from blogs, 20% misc.). The evaluators determined that, among the documents,

- 98% have the totality of the relevant text extracted;
- 99% have all their menu items removed;
- 99% have all information on connected documents removed.

Although this evaluation is partial, it shows that the cleaning process is effective.

We also evaluated the language detection module by applying it on 300 documents, 100 for each language. The percentage of documents the language of which was correctly detected, reached 99% for French, 94% for English and 92% for Spanish. The incorrect detections come from web documents with no or very short texts, like *YouTube* ones.

B. Classification evaluation

We made two types of evaluations to measure the quality of our classification process: (1) the quality of the description of the classes (keyword list); (2) the quality of the document class assignement. The evaluation corpus was composed of 173 documents all speaking of the French political party *UMP* and divided by the system into 88 clusters. For the first evaluation, we asked the evaluators to indicate the quality of the description of each class obtained after inserting all documents. They had to assign each of them to one of the three following categories: *not or little understandable* (NU), *understandable* (U), *immediate understanding* (IU). Results are gathered in table II.

For the second evaluation, the evaluators had to indicate for each document, whether it has been assigned to a correct class by the system. The system reached a score of 85% of documents assigned the correct class. There exist two main types of errors. First, the document was inserted in a class with a low intra-class value but not low enough to be splitted during the class refinement process. Secondly, it was assigned to a singleton class that should has been merged to another one.

The results obtained are good. Nevertheless, it is very difficult to evaluate the system because the number of classes

TABLE II
QUALITY OF THE DESCRIPTION OF CLASSES (KEYWORD LIST)

Quality rate	NU	U	IU
Score	4.5%	20.5 %	75 %

is high (i.e. very fine-grained classes). In practice, the system has been tested and is successfully used in the company XERES.

VI. Conclusions and future work

In this paper, we described a real-time unsupervised classification system dealing with dynamic sets of web documents. It is based on a single-pass algorithm assigning a new document either to an existing class or to a new one. It has been shown that the classifications obtained are precise enough to be sucessfully used in the domain of Information intelligence. This tool, nevertheless, requires some improvements, especially time efficiency optimizations. The evaluation also needs larger experiments and has to be compared with other systems.

References

[1] Charu C. Aggarwal, Stephen C. Gates and Philip S. Yu "On using Partial Supervision for Text Categorization" *IEEE Transactions on Knowledge and Data Engineering* Volume 16 , Issue 2, 2004, pp. 245-255 .

[2] M. T. Elmore, J. W. Reed and T. E. Potok "Real-Time Document Cluster Analysis for Dynamic Data Sets" IPSI-Amalfi, 2005

[3] Maurice Gross "The construction of local grammars" In Roche, E., Schabes, Y. (eds.) *Finite-State Language Processing*, Cambridge, Mass., The MIT Press, 1997, pp. 329–352.

[4] Sébastien Paumier *Unitex 2.0 User Manual*, 2008

[5] Dragomir R. Radev, Vasileios Hatzivassiloglou and Kathleen R. McKeown "A description of the CIDR system as used for TDT-2" In Proceedings, *DARPA Broadcast News Workshop*, Herndon, VA, 1999

[6] Gerard Salton and M. J. McGill *Introduction to modern information retrieval*, McGraw-Hill, 1983, 448 p.

[7] Helmut Schmid "Probabilistic Part-of-Speech Tagging Using Decision Trees", *Proceedings of International Conference on New Methods in Language Processing*, 1994

[8] Max Silberztein *INTEX: An FST Toolbox*, Theoretical Computer Science, Volume 231 Issue 1, 2000, pp. 33-46,