Using tree-grammars for training set expansion in page classification

Stefano Baldi Simone Marinai Giovanni Soda

DSI - University of Florence - Italy
Email: marinai@dsi.unifi.it

Abstract

In this paper we describe a method for the expansion of training sets made by XY trees representing page layout. This approach is appropriate when dealing with page classification based on MXY tree page representations. The basic idea is the use of tree grammars to model the variations in the tree which are caused by segmentation algorithms. A set of general grammatical rules are defined and used to expand the training set. Pages are classified with a $k-nn$ approach where the distance between pages is computed by means of tree-edit distance.

1. Introduction

Document image classification has a large number of applications such as document organization, retrieval, routing, and understanding. An efficient initial classification can be achieved by representing the layout with XY-trees [9], and their extension dealing with ruling lines (MXY-tree [2]). In XY-trees the root brings information about the entire page and each child contains a portion of the image related to its father. Every portion is recursively obtained by XY-cuts. An XY-cut is a horizontal or vertical cut following blank spaces (or thin lines in MXY-tree), which extends from side to side of the image.

MXY trees have been recently used for page classification by using a vectorial representation of the tree that is classified by means of artificial neural networks [3]. MXY-tree descriptions of document images have been used as well for the retrieval of relevant pages in the image domain [4].

Unfortunately, the segmentation algorithms which build the XY-tree, do not produce similar trees starting from similar pages (e.g. Fig. 1). In some cases the related trees are very different each other, and this will give rise to unexpected difficulties when trying to compare these trees.

When working in the domain of trainable classifiers a basic assumption is the availability of a large enough training set so as to be able to generalize differences introduced by segmentation algorithms and obtain a correct classification. For practical uses it is frequently too expensive to produce large hand-labeled training sets. Moreover, in some application domains large training sets are simply unavailable regardless of the effort required for groundtruthing.

In this paper we propose a solution to this problem that is based on the introduction of a set of tree-grammar rules which are used by an expansion algorithm to enlarge an initial set of XY-trees. A larger training set is obtained which contains both the initial samples (*natural samples*) and the *artificial* ones. This set will be the new learning set. This approach is somehow similar to the distortion of graph models proposed in [8] in order to model real world distortions in attributed graphs. Training set expansion has been recently applied also in the domain of handwritten character recognition [1].

The paper is organized as follows: in Section 2 we describe the proposed method for training set expansion, that in turn contains a description of the proposed tree grammar (Sec. 2.1) and a discussion of the expansion algorithm (Sec. 2.2). The use of tree-edit distance for page classification is analyzed in Section 3, and the related classes are analyzed in Section 3.1. The experimental results are reported in Section 4, whereas a final discussion is drawn in Section 5.

2. Training set expansion

Page classification is based on two main steps: an off-line training set expansion and an on-line page classification. In the off-line step new trees are added to the training set by modifying the labeled ones to simulate actual distortions occurring in real segmentations. The distortions are modeled with an appropriate tree grammar. In the second step an unknown tree is classified by comparing it with the trees in the expanded training set. To purpose, in this paper we use a $k-nn$ classification approach where the distance with documents in the training set is computed with the tree-edit distance.

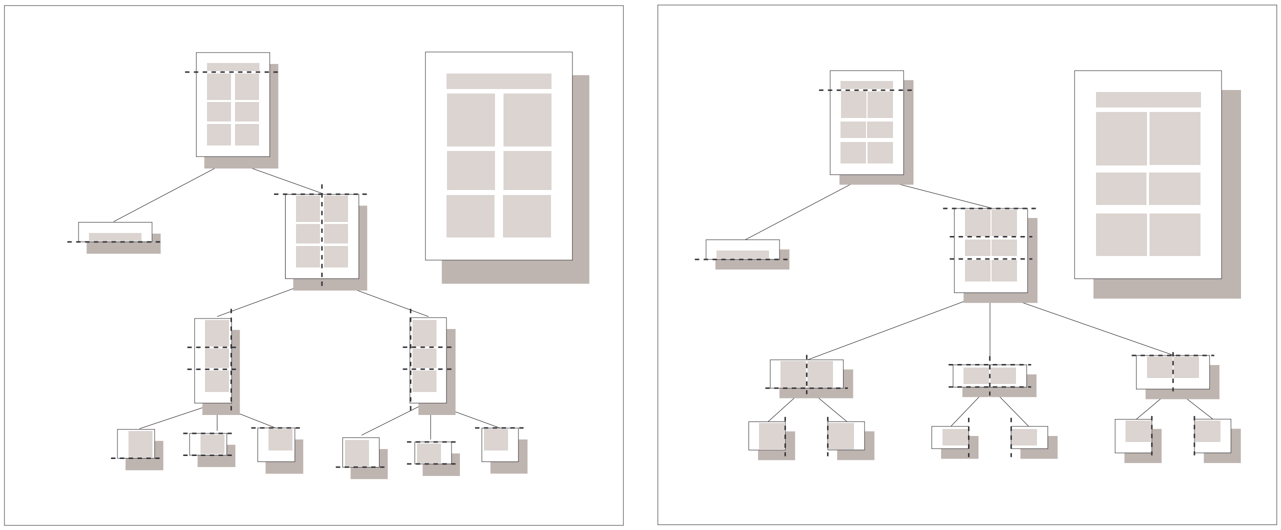

Figure 1. MXY trees obtained from similar pages with different spacings between regions.

2.1. MXY tree grammar

Tree grammars [7] are similar to string grammars except that the basic objects are trees instead of strings. More precisely a **tree grammar** $G = (\mathcal{S}, \mathcal{N}, \mathcal{T}, \mathcal{P})$ is defined by a starting symbol $\mathcal{S}$ ($\mathcal{S} \in \mathcal{N}$), a set $\mathcal{N}$ of nonterminal symbols, a set $\mathcal{T}$ of terminal symbols, a set $\mathcal{P}$ of production rules of the form $\alpha \rightarrow \beta$ where α contains at least one nonterminal. In the following we will refer to α as left hand side (LHS) member and to β as right hand side (RHS) one. Generally speaking the LHS member detects what objects the rule has to be applied to, and RHS describes how to build the related output.

It is well known that each labeled tree can be represented as a string by using a pre-fix notation where the label of a node *precedes*, in the related string, the list of the sub-strings which represent the sub-trees originated by the node's children (see Fig. 2).

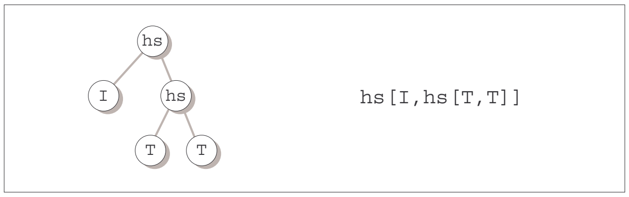

Figure 2. Example of prefix notation.

This property becomes very useful when defining rules with a pre-fix notation. Similarly to string grammars we also allow the use of *wildcards* as star-mark (*) or plus-mark (+) with the meaning of "zero or more repetitions of" and "one or more repetitions of" a tree, respectively.

In this way we describe the *structures* of the trees we want to detect or build. In addition, we are interested to work with labeled trees, where a label describes a region, like the type of XY-cut for internal nodes (horizontal or vertical cut along spaces or lines) and the region content for leaves (image, text block, or ruling line). The labels can contain additional information such as the block size and the number of children.

Several packages exist for the definition of tree grammar and subsequent language generation (e.g. [5, 6]), however the language generation algorithms are not appropriate for our problem. Instead we developed our own system in order to integrate the grammar definition with the expansion algorithm described in Section 2.2. Our main task in designing the expansion program is to allow an user to describe the rules in the most flexible way. Several kinds of tree grammars can be described as *regular* or *expansive* ones [7], and the user can define (in a specific JAVA compiled file) *logical predicates* or *alteration functions* to be used into rules.

A *logical predicate* (Table 1) works in the *LHS* member of a rule, it checks a local property of one tree (related to arrangement, number and label of nodes) and returns a boolean value. The *LHS*-conditions are satisfied only when all the contained predicates are satisfied. One example of predicate is the exclamation-mark (!) that works as a negation. Let us suppose it precedes a label identifying an image in the LHS member; in this case the rule will be applied to all the trees that do not contain an image in the related node. On the contrary, an *alteration function* (Table 1) works in the *RHS* member of a rule, and allows to modify the tree structure. The user can change the type of a set of nodes (blocks of text with images for example), or their disposition.

We use the following set of labels for describing the meaning of nodes. `hs` (`vs`): cut along a *horizontal* (*vertical*) space. `hl` (`vl`): cut along a *horizontal* (*vertical*) line. `T` : text-block (leaf). `I` : image-block (leaf). `hL` (`vL`): hori-

Predicate	Meaning
Ldw $value$	Node level lower than $value$
Lup $value$	Node level greater than $value$

Function	Meaning
b+(trees)	The sub-trees listed in trees are added as right-brothers to the current node.
b-(trees)	The sub-trees listed in trees are added as left-brothers to the current node.
+(trees)	The sub-trees listed in trees are added as right-children to the current node.
/(tree)	The tree starting in the current node is substituted with the tree described in tree .
rnd(tree)	The sub-tree described in tree is added to the children's list of the current node in a random point.

Table 1. Some predicates and alteration functions.

zontal (vertical) line (leaf). x : identifies any kind of block. Rules are defined with a pre-fix notation. For instance, one rule containing hl[I,hs[T,T]] in the LHS will be applied to trees like that in Figure 2. By using wildcards it is possible to define more general rules. For instance, the expression hl[I,hs[T*]] still detects the tree described in Figure 2 as well as trees with a different number of leaves in the right branch.

Let us now describe some typical examples of rules. Rule (1) adds an image-leaf in the right branch.

$$\texttt{hl[I,hs[T,T]]} \rightarrow \texttt{hl[I,hs[T,I,T]]} \quad (1)$$

Rule (2) shows how predicates and alteration functions work. " Ldw0 " is a logical predicate which is true when the related node belongs to the zero level of the tree (i.e. it is the root). " ! " is a logical predicate which is a negation. " rcb " is an alteration function which changes the order of the children of the corresponding node (recombination). The effect of this rule (2) is to modify a tree without an image in the first branch and with two text-block leaves on the other one, into a tree with an image in the first block and the two text-blocks inverted each other.

$$\texttt{hl(Ldw0)[!I,hs[T,T]]} \rightarrow \texttt{hl[I,hs(rcb)[T,T]]} \quad (2)$$

2.2. Language generation

After defining an appropriate set of rules, it is possible to expand the training set. In other words we can generate the language defined by the grammar. To avoid an excessive distortion in generated trees, we assign a threshold to each tree and a cost to the deformation made by each rule. A given rule matching the LHS with one tree will be applied when the application cost ($Cost$) is below the current *Threshold*. This approach is described in Algorithm 1, where t is the tree to expand, r is the expansion rule, and (T) is the result of the possible application of r to t. *ApplicationCostOf* checks the *LHS*-conditions and computes the application cost of r to t (if *LHS*-conditions do not hold it returns an overflow value) by considering the cost assigned to the rule. *ThrOf* associates to each tree the related threshold, and *Apply* is a function that returns a modified tree.

Algorithm 1 Expand(t, r; T)

```
  Cost = ApplicationCostOf($t$, $r$)
  **if** Cost $<$ ThrOf($t$) **then** T $\leftarrow$ Apply($t$, $r$) return **true**
  **else return false**
```

When expanding a training set we start with an initial working set (*T_Set*) containing only hand-labeled trees (the *Natural_Set*), then we try to apply to each tree $t \in T\_Set$ each rule r in the rule set, by the *Expand* algorithm. The modified tree T is added to the set *T_Set* and the $Threshold$ is updated (see Algorithm 2).

Algorithm 2 ExpandTreeSet

```
  T_Set $\leftarrow$ Natural_Set
  foreach $t \in$ T_Set do :
    foreach $r \in$ Rules_Set do :
      if (Expand($t$, $r$; $T$) ) then T_Set $\leftarrow$ T_Set $\cup$ $T$
      ThrOf($T$) $\leftarrow$ ThrOf($t$) - ApplicationCostOf($t$, $r$)
```

Note that an expanded tree can be expanded again (as a natural one). However, its threshold will be lower than the original one, so it will probably generate a lower number of expansions. This process is repeated until no more rule is applicable or it is too much expensive regarding the remaining threshold. The scheme in Figure 3 shows an example of hierarchy of the language terms, it looks like a tree in the root of which there is one natural tree. We will call this hierarchy a *dictionary*. Every tree of a dictionary will be assigned to the class of the natural tree in the root. During language generation a dictionary is build for each natural tree, and the corresponding trees are added to the expanded training set.

3. Tree classification

MXY tree representations for page layout classification have been used in [3] in conjunction with a vectorial representation of trees and MLP-based classifiers. In this paper we check the effectiveness of the proposed expansion method with a classification approach that is more appropriate for incremental learning. Basically, we compare with a tree-edit distance the unknown tree with trees in the training set and the class is found with a $k-nn$ mechanism. The classification cost of this approach is quite expensive, however the principal aim is to demonstrate the advantages

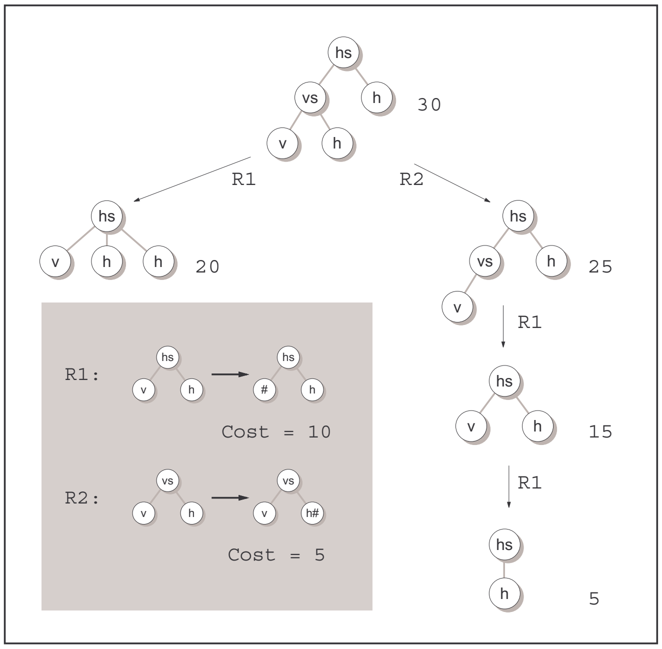

Figure 3. Example of hierarchy of generated terms.

that can be obtained when expanding the training set (see Section 4).

Similarly to string edit distance, the tree edit distance is a method for evaluating the distance between labeled trees by counting the number of edit operations (with an associated cost) needed to transform one tree into another. We can define the distance between two trees as the cost of the minimum-cost set of operations that are required to transform the first tree into the second one. Zhang [10] proposed an efficient algorithm to compute the tree edit distance. Using tree edit distance we can build a $K - nn$ classifier were each tree is ascribed in the most common class among the K trees of the training set that are nearest (in the sense of the Zhang's distance) to the unknown tree.

Using such a classifier, we can classify a tree using an expanded training set or a not-expanded (*natural*) one and evaluate the differences among the classifications obtained. If the rules work properly the classifications on the expanded training set will be better classifications on natural-set. First of all we have to identify some useful expansion rules. Remaining in the most general conditions, we'll look for a different rules-set for each class, since different classes have different peculiarities to be considered.

3.1. Classes

In a classification problem the training set contains a different number of samples for each class. Some class is highly populated and some other is not. This is related to problems of practical nature. For instance, in a book there are less pages containing titles or illustrations than full-text pages. We are obviously interested in the expansions of low-populated classes instead of the highly-populated ones. In our experiments we consider seven different classes shortly described in Table 2.

Class name	Description of pages in the class
Image	An Image with or without caption
ImageText2	An Image on two columns text
Issue2	Start of an issue
SecE2	End-of-section page
SecM2	Section mark page
Text2	Text on two columns (no images)
Text2Image	Image and two text-columns

Table 2. Main features of classes.

Text2 and *ImageText2* are the most populated classes, we did not write expansion-rules for these ones. For the other classes we designed *a priori* some appropriate rules following common sense. For instance, in classes with images one appropriate rule allows us to add a text block (*caption*) below an image when it does not appear (see Figure 4). In classes with two-column text it is useful to provide rules splitting one text-block in two or more blocks in order to simulate the presence of sub-paragraphs in the text.

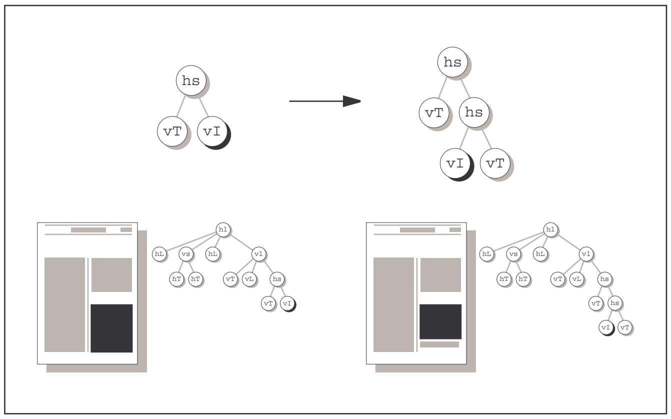

Figure 4. Example of a rule used to add a caption to a figure.

4. Experimental results

In this section we describe the experiments carried out on a set of labeled pages contained in two books of the 19th Century (books of the 19th Century are particularly interesting for Digital Libraries since they are copyright-free),

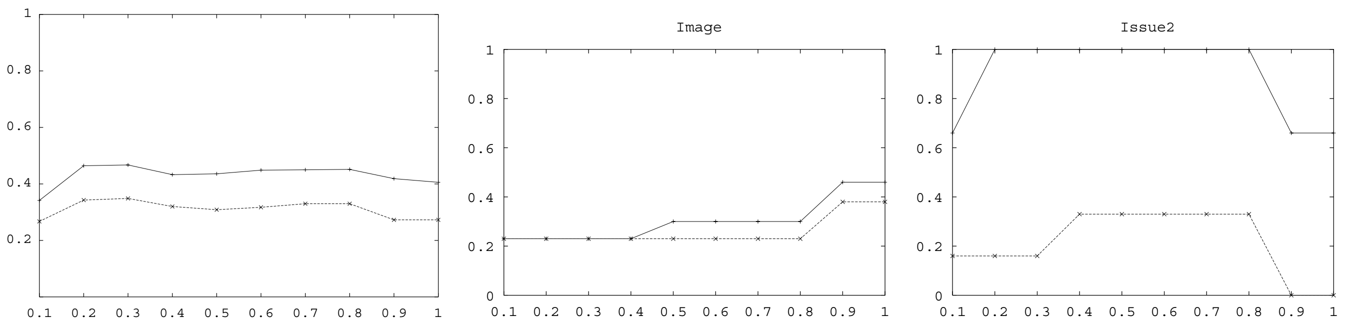

Figure 5. Left: average classification error (for the whole test set) when varying the percentage of pages in the natural set. Center and right: classification error in class Image and Issue2, respectively.

downloaded from the web site of the *National Library of France*. Each book contains roughly 650 pages belonging to classes described in Table 2. An average of five rules have been designed for each class. With these experiments we want to answer to two questions. First, verify whether the generated pages are correct samples of the corresponding class. Second, evaluate the smallest size of the expanded set of pages.

We classified all the pages of a book (*test-set*) with a *training-set* built starting from natural pages from the other book only. The test-set is fixed, whereas we varied the number of natural pages from 10% to 100% of pages in the second book. Figure 5 reports the average relative classification error when growing the size of the natural set. The two lines report the error obtained classifying with a training set of natural pages only (continuous line) and a training set of natural pages and expanded ones (dotted line). In the figure we report also the relative classification error for two typical classes.

We can observe that the average error on the whole set of classes is always the lowest when using the expanded set regardless of the size of the natural set used for training. In some classes the error gets best just when the percentage arises up $0.3 - 0.4$ (classes *Image*, *Text2Image*).

5. Conclusions

The experimental results confirm us that the artificially generated pages are representative of the corresponding class. Moreover, we can actually extimate that when using a set of pages containing less than half of a book, we can get a good generalization from rules. Of course these results are related to the database and the set of rules we have used, but anyway they validate the effectiveness of the proposed approach.

The rule set is easy to redefine, so changes in the database can be described with the grammar. However currently every change in the rule set is committed to user. We are working on a system that automatically improves the performance of the rule set by acting on its composition excluding unuseful rules. This is made by computing the contribution of the application of each rule to the overall classification error on a training set. On the other hand a full-automatic learning of rules seems to be difficult for the high-level nature of the rules.

References

[1] J. Cano, J. C. Pérez-Cortes, J. Arlandis, and R. Llobet. Training set expansion in handwritten character recognition. In *Proc. SSPR/SPR 2002*, pages 548–556, 2002.

[2] F. Cesarini, M. Gori, S. Marinai, and G. Soda. Structured document segmentation and representation by the modified X-Y tree. In *Proc. Fifth ICDAR*, pages 563–566, 1999.

[3] F. Cesarini, M. Lastri, S. Marinai, and G. Soda. Encoding of modified X-Y trees for document classification. In *Proc. Sixth ICDAR*, pages 1131–1136, 2001.

[4] F. Cesarini, S. Marinai, and G. Soda. Retrieval by layout similarity of documents represented with MXY. In *Document Analysis Systems V*, pages 353–364, 2002.

[5] F. Drews. *The TREEBAG Manual*. Department of Computer Science, Ume University, S-90187 Ume Sweden.

[6] C. Ermel and T. Schultzke. *The AGG Enviroment: A Short Manual*. TU Berlin.

[7] R. C. Gonzalez and M. C. Thomason. *Syntattic Pattern Recognition - an introduction*. Addison-Wesley, 1978.

[8] B. T. Messmer and H. Bunke. Error-correcting graph isomorphism using decision trees. *IJPRAI*, 12(6):721–742, 1998.

[9] G. Nagy and S. Seth. Hierarchical rappresentation of optical scanned documents. *Proc. of ICPR*, pages 347–349, 1984.

[10] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Computing*, 18(6):1245–1262, December 1989.