Experimental Assessment of a Threshold Selection Algorithm for Tuning Classifiers in the Field of Hierarchical Text Categorization

Andrea Addis, Giuliano Armano, and Eloisa Vargiu

Department of Electrical and Electronic Engineering
University of Cagliari
Piazza d'Armi, Cagliari, Italy
{addis, armano, vargiu}@diee.unica.it

Abstract

Text Categorization is the task of assigning predefined categories to text documents. It can provide conceptual views of document collections and has many important applications in the real world. Nowadays, most of the research on text categorization has focused on mapping text documents to a set of categories among which structural relationships hold. Without loss of generality, let us assume that a classifier entrusted with recognizing documents of a given category outputs a degree of membership, usually a value in [0,1]. The behavior of any such classifier typically depends on an acceptance threshold, which turns the degree of membership into a dichotomous decision. In principle, the problem of finding the best acceptance thresholds for a set of classifiers related by taxonomic relationships is a difficult problem. Hence, any proposal aimed at finding suboptimal solutions to this problem may have great importance, especially in the field of hierarchical text categorization. In this paper, we make an experimental assessment of a greedy threshold selection algorithm aimed at finding a suboptimal combination of thresholds in a hierarchical text categorization setting. The quadratic complexity of the algorithm makes it easier to find good suboptimal solutions even for large taxonomies. Experimental results, performed on Reuters data collections, show that the proposed approach is able to find suboptimal solutions with small computational complexity.

1 Introduction

The new information era has widely changed our lives thanks to a great deal of new possibilities to create content (i.e., knowledge) and share it all over the world throughout new and widespread communication systems. Human beings have always known the importance of organizing entities or notions in hierarchies, according to the "divide et impera" paradigm. In the last few decades, with the advent of modern information systems, this concept has been widely employed to partition items and concepts into smaller parts, each being effectively and efficiently managed. The contribution of this

Proceedings of the 17th International RCRA workshop (RCRA 2010): Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion Bologna, Italy, June 10–11, 2010

approach is particularly evident in the Web 2.0, where the main and most crucial knowledge bases exploit it in order to organize large collections of web pages[1], articles[2] or emails[3] in hierarchies of topics. This organization allows to focus on a specific level of details ignoring specialization at lower levels and generalization at upper levels. In this scenario, the main goal of automatic categorization systems is to deal with reference taxonomies in an effective and efficient way –the corresponding research subfield being Hierarchical Text Categorization (HTC).

Nowadays there is a great amount of contributions in HTC focusing, in particular, on how to build training sets, selecting features, and learning methods. In our opinion, a further important issue is concerned with the fact that, in real-world scenarios, data is typically characterized by imbalance. In fact, relevant and irrelevant documents (i.e., positive and negative examples, respectively) are typically imbalanced, turning classifiers trained with the same percent of positive and negative examples into inadequate tools. According to [8], a hierarchical approach can give benefits to the systems involved with the above scenarios.

Without loss of generality, let us assume that a classifier for a category $c_i \in C$ usually consists of a function $CSV_i : D \rightarrow [0, 1]$. Given a document d_j, $CSV_i(d_j)$ returns a categorization status value for it that represents the evidence for the fact $d_j \in c_i$. The behavior of c_i depends on an acceptance threshold θ_i, such that $CSV_i(d_j) \geq \theta_i$ is interpreted as $d_j \in c_i$, and conversely $CSV_i(d_j) < \theta_i$ is interpreted as $d_j \notin c_i$. In HTC, this is transposed into a multidimensional space, i.e., the definition of a vector of thresholds θ.

In principle, how to find the best acceptance thresholds for a set of classifiers related by taxonomic relationships is a difficult problem. As for TC, there are various policies for determining the acceptance threshold, the most important distinction being whether it is analytically or experimentally derived [11]. The former approach is possible only in presence of a theoretical result that indicates how to compute the threshold that maximizes the expected value of an effectiveness function [10]. The latter consists of testing different values for the acceptance threshold on a validation set and choosing the value which maximizes effectiveness [12]. As for HTC, let us recall here the threshold selection algorithms proposed in [7] and [5].

In this paper, we perform an experimental assessment of a greedy threshold selection algorithm aimed at finding a suboptimal combination of thresholds in the context of Progressive Filtering (PF), the hierarchical text categorization setting proposed in [2]. Experimental results, performed on Reuters data collections, show that the proposed approach is able to find suboptimal

[1]see e.g., Google Directory (http://www.google.com/dirhp) and the DMOZ project (http://www.dmoz.org)

[2]see e.g., Wikipedia (http://www.wikipedia.org) and Reuters (http://www.reuters.com/)

[3]see e.g., Thunderbird 3 (http://www.mozillamessaging.com/thunderbird/)

solutions while maintaining a quadratic complexity, which makes it easier to find good suboptimal solutions even for large taxonomies [3].

The rest of the paper is organized as follows: in section 2, we briefly recall the PF approach. In section 3 we describe TSA, putting into evidence the theoretical background that allowed to build the algorithm and its computational benefits. Experiments and results are illustrated in section 4. Conclusions discussed in section 5 end the paper.

2 Progressive Filtering

A way to implement Progressive Filtering (PF) consists of unfolding the given taxonomy into pipelines of classifiers, as depicted in Figure 1. Each node of the pipeline is a binary classifier able to recognize whether or not an input belongs to the corresponding class (i.e., to the corresponding node of the taxonomy).

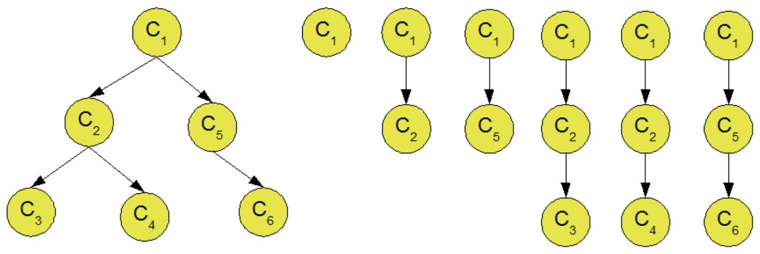


Figure 1: A taxonomy and its corresponding pipelines.

In principle, PF could be applied to classify any kind of item: images, audios, videos, textual documents, and so on. In this paper we are interested in using PF for hierarchical text categorization, so that textual documents (documents for short hereinafter) have been considered.

Given a taxonomy, where each node represents a classifier entrusted with recognizing all corresponding positive inputs (i.e., interesting documents), each input traverses the taxonomy as a "token", starting from the root. If the current classifier recognizes the token as positive, it passes it on to all its children (if any), and so on. A typical result consists of activating one or more branches within the taxonomy, in which the corresponding classifiers have been activated by the given token. Let us note that partitioning the taxonomy in pipelines gives rise to a set of new classifiers, each represented by a pipeline. For instance, the taxonomy depicted in Figure 1 gives rise to six corresponding pipelines.

3 The Proposed Threshold Selection Algorithm

3.1 Motivations

As we know from classical text categorization, given a set of documents D and a set of labels C, a function $CSV_i : D \rightarrow [0, 1]$ exists for each $c_i \in C$. The behavior of c_i is controlled by a threshold θ_i, responsible for relaxing or restricting the acceptance rate of the corresponding classifier. Let us recall that, given $d \in D$, $CSV_i(d) \geq \theta_i$ is interpreted as a decision to categorize d under c_i, whereas $CSV_i(d) < \theta_i$ is interpreted as a decision not to categorize d under c_i.

In PF, let us still assume that CSV_i exists with the same semantics adopted in the classical case. Considering a pipeline π, composed by n classifiers, the acceptance policy strictly depends on the vector of thresholds $\theta_\pi = \langle \theta_1, \theta_2, \cdots, \theta n \rangle$ that embodies the thresholds of all classifiers in π. In order to categorize d under π, the following constraint must be satisfied: for $k = 1..n$, $CSV_i(d) \geq \theta_k$. On the contrary, d is not categorized under c_i in the event that a classifier in π rejects it. In so doing, a classifier may have different behaviors, depending on which pipeline it is embedded. As a consequence, each pipeline can be considered in isolation from the others. For instance, given $\pi_1 = \langle C_1, C_2, C_3 \rangle$ and $\pi_2 = \langle C_1, C_2, C_4 \rangle$, the classifier C_1 is not compelled to have the same threshold in π_1 and in π_2 (the same holds for C_2).

Actually, the proposed approach performs a sort of "flattening" though *preserving* the information about the hierarchical relationships embedded in a pipeline. For instance, the pipeline $\langle C_1, C_2, C_3 \rangle$ actually represents the classifier C_3, although the information about the existing subsumption relationships are preserved (i.e., $C_3 \prec C_2 \prec C_1$, where "$\prec$" denotes the usual covering).

In PF, given an utility function[4], we are interested in finding an effective (and computationally "light") way to reach a sub-optimum in the task of determining the best vector of thresholds. To this end, for each pipeline π a sub-optimal combination of thresholds is searched for, considering the actual ratio between positive and negative examples, which in turn depends on the given scenario. Unfortunately, finding the best acceptance thresholds is a difficult task. In fact, exhaustively trying each possible combination of thresholds (brute-force approach) is unfeasible, the number of thresholds being infinite. However, the brute-force approach can be approximated by defining a granularity step that requires to assess only a finite number of points in a range $[0, 1]$ in which the thresholds are permitted to vary with step δ. This "relaxed" brute force algorithm (RBF for short) for calibrating thresholds would be very helpful in the task of finding a sub-optimal solution,

[4]Different utility functions (e.g., precision, recall, F_β, user-defined) can be adopted, depending on the constraint imposed by the underlying scenario.

although *still* too heavy from a computational point of view. Thus, in this paper we propose a novel Threshold Selection Algorithm (namely, TSA) devised to deal with this problem, which maintains the capability of finding a near-optimum solution characterized by a low time complexity.

3.2 The Proposed Algorithm

Utility functions typically adopted in TC, and therefore in HTC, are nearly-convex with respect to the acceptance threshold. In Figure 2 three typical trends of utility functions are depicted (precision, recall, and F_1).

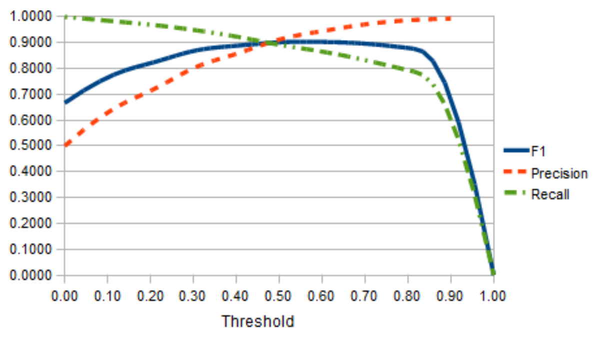


Figure 2: Example of utility functions.

Let us note that setting the threshold of a classifier to 0, no matter which utility function is adopted, forces the classifier to reach its maximum error in terms of false positives (FP). On the other hand, setting the threshold to 1 forces the classifier to reach its maximum error in terms of false negatives (FN).

Due to the shape of the utility function and to its dependence on FP and FN, it becomes feasible to search its maximum around a restricted range (i.e., a subrange of $[0, 1]$) towards a defined direction. In particular, bearing in mind that the lower the threshold the less restrictive the classifier, we can build a greedy bottom-up algorithm for selecting decision threshold that relies on two functions:

- *repair* ($\mathcal{R}$), which operates on a classifier C by increasing or decreasing its threshold –i.e., $\mathcal{R}(up, C)$ and, $\mathcal{R}(down, C)$, respectively– until the selected utility function reaches and maintains a local maximum.
- *calibrate* ($\mathcal{C}$), which operates going downwards from the given classifier to its offspring by repeatedly calling $\mathcal{R}$. It is intrinsically recursive and at each step it calls $\mathcal{R}$ to calibrate the current classifier.

Given a pipeline $\pi = \langle C_1, C_2, \cdots, C_L \rangle$, where L is its depth, *TSA* is then defined as follows (all thresholds are initially set to zero):

$$TSA(\pi) := for\, k = L \;\; downto\, 1\, do\, \mathcal{C}(up, C_k) \tag{1}$$

which indicates that $\mathcal{C}$ is applied to each node of the pipeline, starting from the leaf ($k = L$).

Under the assumption that p is a structure that contains all information about a pipeline, including the corresponding vector of thresholds and the utility function to be optimized, the pseudo-code of TSA is:

```
function TSA(p:pipeline):
  for k:=1 to p.length
    do p.thresholds[i] = 0
  for k:=p.length downto 1
    do Calibrate(up,p,k)
  return p.thresholds
end TSA
```

The *Calibrate* function is defined as follows:

$$\begin{aligned} &\mathcal{C}(up, C_k) := \mathcal{R}(up, C_k),\;\; k = L \\ &\mathcal{C}(up, C_k) := \mathcal{R}(up, C_k) + \mathcal{C}(down, C_{k+1}),\;\; k < L \end{aligned} \tag{2}$$

and

$$\begin{aligned} &\mathcal{C}(down, C_k) := \mathcal{R}(down, C_k),\;\; k = L \\ &\mathcal{C}(down, C_k) := \mathcal{R}(down, C_k) + \mathcal{C}(up, C_{k+1}),\;\; k < L \end{aligned} \tag{3}$$

where the $+$ operator actually denotes a sequence operator, meaning that in the formula $a + b$ action a is performed *before* action b. In pseudo-code:

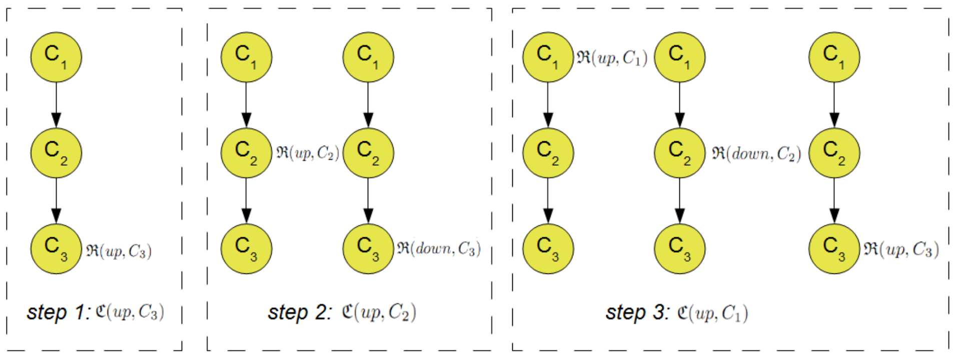
```
function Calibrate(dir:{up,down}, p:pipeline, level:integer):
  Repair(dir,p,level)
  if level < p.length then Calibrate(toggle(dir),p,level+1)
end Calibrate
```

where *toggle* is a function that reverses the current direction, and *Repair* is defined as:

```
function Repair(dir:{up,down}, p:pipeline, level:integer):
  delta := (dir = up) ? p.delta : -p.delta
  best_threshold := p.thresholds[level]
  max_uf := p.utility_function()
  uf := max_uf
  while uf >= max_uf * 0.8 and p.thresholds[level] in [0,1]
    do p.thresholds[level] := p.thresholds[level] + delta
       uf := p.utility_function()
       if uf < max_uf then continue
       max_uf := uf
```

```
        best_threshold := p.thresholds[level]
  p.thresholds[level] := best_threshold
end Repair
```

The factor 0.8 is used to limit the impact of local minimums during the search.



$\mathfrak{R}(up, C_3)$
step 1: $\mathfrak{C}(up, C_3)$
$\mathfrak{R}(up, C_2)$
$\mathfrak{R}(down, C_3)$
step 2: $\mathfrak{C}(up, C_2)$
$\mathfrak{R}(up, C_1)$
$\mathfrak{R}(down, C_2)$
$\mathfrak{R}(up, C_3)$
step 3: $\mathfrak{C}(up, C_1)$

Figure 3: Unfolding the threshold-selection procedure for a pipeline composed by three classifiers.

To better illustrate the approach, let us consider the unfolding reported in Figure 3, which corresponds to $\pi = \langle C_1, C_2, C_3 \rangle$:

step 1

$$\mathcal{C}(up, C_3) = \mathcal{R}(up, C_3)$$

step 2

$$\begin{aligned} \mathcal{C}(up, C_2) &= \mathcal{R}(up, C_2) + \mathcal{C}(down, C_3) \\ &= \mathcal{R}(up, C_2) + \mathcal{R}(down, C_3) \end{aligned}$$

step 3

$$\begin{aligned} \mathcal{C}(up, C_1) &= \mathcal{R}(up, C_1) + \mathcal{C}(down, C_2) \\ &= \mathcal{R}(up, C_1) + \mathcal{R}(down, C_2) + \mathcal{C}(up, C_3) \\ &= \mathcal{R}(up, C_1) + \mathcal{R}(down, C_2) + \mathcal{R}(up, C_3) \end{aligned}$$

Once calculated the sub-optimal combination of thresholds for a given imbalance, the pipelines are ready to be used in the corresponding scenario. Let us note, here, that this combination depends on both the adopted dataset and the actual input imbalance. In fact, as noted in [10], different goals for the system lead to different optimal behaviors.

3.3 Computational Complexity

Searching for a sub-optimal combination of thresholds in a pipeline can be actually viewed as the problem of finding a maximum in a utility function F that depends on the corresponding thresholds θ:

$$\theta^* = \underset{\theta}{argmax}\ \ F(\theta) \tag{4}$$

Unfortunately, the above task is characterized by a high time complexity, being actually a problem of meta-learning (i.e., a learning problem whose *instances* are in fact learning problems themselves). To calculate the computational complexity, let us define a granularity step that requires to visit only a finite number of points in a range $[\rho_{min}, \rho_{max}]$, $0 \leq \rho_{min} < \rho_{max} \leq 1$, in which the thresholds could vary with step δ. Therefore, $p = \lfloor \delta^{-1} \cdot (\rho_{max} - \rho_{min}) \rfloor$ being the maximum number of points to be checked for each classifier in a pipeline and L being the length of the pipeline, the average time of TSA, $T(TSA)$, is proportional to $(L + L^2) \cdot p \cdot (\rho_{max} - \rho_{min})$, which implies that TSA has complexity $O(L^2)$. Hence, the time complexity is quadratic with the number of classifiers embedded by a pipeline. As already noted, a comparison between TSA and the brute-force approach is unfeasible, the generic element of the threshold vector being a real number. Nevertheless, experimental results show that the effectiveness of TSA is almost identical to the one obtained by running RBF, in which only p points are checked for each classifier in a pipeline. Note that the average time of RBF, $T(RBF)$, is proportional to p^L, which in turns implies that its computational complexity is $O(p^L)$.

To show the drastic complexity reduction brought by the TSA algorithm, let us consider a pipeline composed of 4 classifiers (i.e., $L = 4$), and $p = 100$. In this case the orders of magnitude of $T(RBF)$ and $T(TSA)$ are 10^8 whereas 10^3, respectively.

It is also important to note that, due to the enormous computational time that can be reached, the RBF approach should be applied in practice only by setting p to a value much lower than the one applicable with TSA. For instance, with a ratio of 10^{-2}, i.e., $p_{RBF} = 20$ and $p_{TSA} = 2000$, $T(RBF) \propto 160,000$ and $T(TSA) \propto 20,000$. In other words, TSA is still 8 times faster than RBF (even considering $\rho_{max} = 1$ and $\rho_{min} = 0$), while ensuring a better result due to the higher granularity.

4 Experimental Results

The Reuters Corpus Volume I (RCV1-v2) [9] has been chosen as benchmark dataset. In this corpus, stories are coded into four hierarchical groups: Corporate/Industrial (CCAT), Economics (ECAT), Government/Social (GCAT), and Markets (MCAT). Although the complete list consists of 126 categories, only part of them have been used in our hierarchical approach. The total number of codes actually assigned to the data is 93, whereas the overall number of documents is about 803,000, each document belonging to at least one category and, on average, to 3.8 categories. To calculate the time complexity of TSA with respect to RBF, we selected the 24 pipelines of depth 4 that end with a leaf node.

Experiments have been performed on a SUN Workstation with two Opteron 280, 2Ghz+ and 8Gb Ram. To perform experiments we customized the X.MAS [1] architecture, a generic multiagent architecture built upon JADE [4] and devised to make it easier the implementation of information retrieval/filtering applications.

Experiments have been carried out by using classifiers based on the *wk*-NN technology [6], which do not require specific training and are very robust with respect to noisy data. As for document representation, we adopted the bag of words approach, a typical method for representing texts in which each word from a vocabulary corresponds to a feature and a document to a feature vector. First, all non-informative words such as prepositions, conjunctions, pronouns and very common verbs are disregarded by using a stop-word list. Subsequently, the most common morphological and inflexional suffixes are removed by adopting a standard stemming algorithm. After having determined the overall sets of features, their values are computed for each document resorting to the well-known TF-IDF method. To reduce the high dimensionality of the feature space, we locally select the features that represent a node by adopting the information gain method. During the training activity, each classifier has been trained with a balanced data set of 1000 documents, characterized by 200 (TF-IDF) features selected in accordance with their information gain.

Experiments, performed on a balanced dataset of 2000 documents for each class, have been focused on calculating the performance improvement of TSA vs. RBF. A step $\delta_{TSA} = 5 \times 10^{-4}$ (hence, $p_{TSA} = 2 \times 10^3$) has been adopted to increment thresholds in TSA; whereas a step $\delta_{RBF} = 0.05$ (hence, $p_{RBF} = 20$) has been adopted for the RBF approach[5]. Table 1 illustrates the results comparing the time spent, in seconds and hours, by RBF and TSA. Each row of the table corresponds to the time spent to perform a calibrate step so that the last row corresponds to the total elapsed time. The last column clearly shows that, the ratio between the time spent by the two algorithms is approximately constant. Table 1 shows that, as expected, TSA is faster than RBF notwithstanding the imposed granularity step. To show that the overall performances of PF are not worsened by the adoption of TSA vs. RBF, we performed further experiments aimed at comparing the performance obtained by applying TSA vs. the performance obtained by

[5]the motivation of this choice has been discussed in the previous section

Table 1: Time comparison between TSA and RBF.

	RBF		TSA		
Step	*time* (s)	*time* (h)	*time* (s)	*time* (h)	*ratio*
*Step*1 (*Level*4)	10333	2.87	382	0.11	27.02
*Step*2 (*Level*3)	97907	27.20	3497	0.97	28
*Step*3 (*Level*2)	213724	59.37	7610	2.11	28.08
*Step*4 (*Level*1)	328372	91.21	11634	3.23	28.22

Table 2: F_1 in presence of input imbalance.

Input imbalance	$F_1(RBF)$	$F_1(TSA)$
2^{-1}	0.830	0.835
2^{-2}	0.722	0.733
2^{-3}	0.619	0.632
2^{-4}	0.497	0.515
2^{-5}	0.404	0.428
2^{-6}	0.323	0.345
2^{-7}	0.245	0.273

applying RBF. In particular, experiments have been performed by assessing the behavior of PF in terms of F_1 in presence of different ratios of positive examples vs. negative examples –i.e., from 2^{-1} to 2^{-7}. Results, summarized in Table 2, show that the performance of TSA is always better than the one obtained with RBF. This is due to the fact that, as previously pointed out, TSA worked with a higher granularity (i.e., $p_{RBF}/p_{TSA} = 20/2000 = 10^{-2}$).

Taking into account these results, we can also establish the average range of search in which the TSA found a maximum of the utility function with $p_{RBF} = 20$, $p_{TSA} = 2000$, and $L = 4$: $T_{RBF} = 8 \cdot \rho \cdot T_{TSA} = 28 \cdot T_{TSA}$ –i.e., $\rho = 0.29$. This further interesting result shows that TSA succeeds in finding the sub-optimum within less that the 30% of the search range. For obvious reasons, $\rho_{max} - \rho_{min}$ decreases while climbing a pipeline from the leaf to the root.

5 Conclusions

In this paper, after proposing TSA, a threshold selection algorithm for hierarchical text categorization, we made an experimental assessment aimed at finding a suboptimal combination of thresholds in a hierarchical text categorization setting. The quadratic complexity of the algorithm makes it easier to find good suboptimal solutions even for large taxonomies. Experimental results, performed on Reuters data collections, show that the proposed approach is able to outperform a relaxed version of the brute force approach. Furthermore, TSA is also very effective in dealing with the input imbalance that typically occurs in real-world scenarios.

References

[1] A. Addis, G. Armano, and E. Vargiu. From a generic multiagent architecture to multiagent information retrieval systems. In *AT2AI-6, Sixth International Workshop, From Agent Theory to Agent Implementation*, pages 3–9, 2008.

[2] A. Addis, G. Armano, and E. Vargiu. Using progressive filtering to deal with information overload. In *7th International Workshop on Text-based Information Retrieval*, 2010.

[3] A. Addis, G. Armano, and E. Vargiu. Using the progressive filtering approach to deal with input imbalance in large-scale taxonomies. In *Large-Scale Hierarchical Classification Workshop*, 2010.

[4] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. John Wiley and Sons, 2007.

[5] M. Ceci and D. Malerba. Classifying web documents in a hierarchy of categories: a comprehensive study. *Journal of Intelligent Information Systems*, 28(1):37–78, 2007.

[6] W. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10:57–78, 1993.

[7] S. D'Alessio, K. Murray, and R. Schiaffino. The effect of using hierarchical classifiers in text categorization. In *Proceedings of of the 6th International Conference on Recherche dInformation Assiste par Ordinateur (RIAO)*, pages 302–313, 2000.

[8] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas. Handling imbalanced datasets: a review. *GESTS International Transactions on Computer Science and Engineering*, 30:25–36, 2006.

[9] D. Lewis, Y. Yang, T. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

[10] D. D. Lewis. Evaluating and optimizing autonomous text classification systems. In *SIGIR '95: Proceedings of the 18th annual international*

ACM SIGIR conference on Research and development in information retrieval, pages 246–254, New York, NY, USA, 1995. ACM.

[11] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys (CSUR)*, 34(1):1–55, 2002.

[12] Y. Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1/2):69–90, 1999.