Experimentally Studying Progressive Filtering in Presence of Input Imbalance

Andrea Addis, Giuliano Armano, and Eloisa Vargiu

Dept. of Electrical and Electronic Engineering, University of Cagliari, Cagliari, Italy
{addis,armano,vargiu}@diee.unica.it
http://iasc.diee.unica.it

Abstract. Progressively Filtering (PF) is a simple categorization technique framed within the local classifier per node approach. In PF, each classifier is entrusted with deciding whether the input in hand can be forwarded or not to its children. A simple way to implement PF consists of unfolding the given taxonomy into pipelines of classifiers. In so doing, each node of the pipeline is a binary classifier able to recognize whether or not an input belongs to the corresponding class. In this chapter, we illustrate and discuss the results obtained by assessing the PF technique, used to perform text categorization. Experiments, on the Reuters Corpus (RCV1- v2) dataset, are focused on the ability of PF to deal with input imbalance. In particular, the baseline is: (i) comparing the results to those calculated resorting to the corresponding flat approach; (ii) calculating the improvement of performance while augmenting the pipeline depth; and (iii) measuring the performance in terms of generalization- / specialization- / misclassification-error and unknown-ratio. Experimental results show that, for the adopted dataset, PF is able to counteract great imbalances between negative and positive examples. We also present and discuss further experiments aimed at assessing TSA, the greedy threshold selection algorithm adopted to perform PF, against a relaxed brute-force algorithm and the most relevant state-of-the-art algorithms.

Keywords: Progressive filtering, Threshold selection, Hierarchical text categorization, Input imbalance.

1 Introduction

According to the "divide et impera" philosophy, the main advantage of the hierarchical perspective is that the problem is partitioned into smaller subproblems, each being effectively and efficiently managed. Therefore, it is not surprising that in the Web 2.0 age people organize large collections of web pages, articles or emails in hierarchies of topics or arrange a large body of knowledge in ontologies. Such 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.

In this chapter, we are aimed at assessing the effectiveness of the "Progressive Filtering" (PF) technique, applied to text categorization in presence of input imbalance. In its simplest setting, PF decomposes a given rooted taxonomy into pipelines, one for

A. Fred et al. (Eds.): IC3K 2010, CCIS 272, pp. 56–71, 2013.

each path that exists between the root and any given node of the taxonomy, so that each pipeline can be tuned in isolation. To this end, a Threshold Selection Algorithm (TSA) has been devised, aimed at finding a sub-optimal combination of thresholds for each pipeline. The chapter extends and revises our previous work [2]. The main extensions consist of: (i) a more detailed presentation of TSA; and (ii) further experiments aimed at assessing the effectiveness of TSA with respect to three state-of-the-art threshold selection algorithms.

The chapter is organized as follows. In Section 2, we give a brief survey of relevant work on: (i) hierarchical text categorization; (ii) threshold selection strategies; and (iii) input imbalance. Section 3 concentrates on PF and TSA. In Section 4, we present and discuss experimental results. Section 5 ends the chapter with conclusions.

2 Background

The most relevant issues that help to clarify the contextual setting of the chapter are: (i) the work done on HTC, (ii) the work done on thresholding selection, and (iii) the work done on the input imbalance.

2.1 Hierarchical Text Categorization

In recent years several researchers have investigated the use of hierarchies for text categorization.

Until the mid-1990s researchers mostly ignored the hierarchical structure of categories that occur in several domains. In 1997, Koller and Sahami [17] carry out the first proper study about HTC on the Reuters-22173 collection. Documents are classified according to the given hierarchy by filtering them through the best-matching first-level class and then sending them to the appropriate second level. This approach shows that hierarchical models perform well when a small number of features per class is used, as no advantages were found using the hierarchical model for large numbers of features. Mc Callum et al. [23] propose a method based on naïve Bayes. The authors compare two techniques: (i) exploring all possible paths in the given hierarchy and (ii) greedily selecting at most two branches according to their probability, as done in [17]. Results show that the latter is more error prone while computationally more efficient. Mladenic and Grobelink [24] use the hierarchical structure to decompose a problem into a set of subproblems, corresponding to categories (i.e., the nodes of the hierarchy). For each subproblem, a naïve Bayes classifier is generated, considering examples belonging to the given category, including all examples classified in its subtrees. The classification applies to all nodes in parallel; a document is passed down to a category only if the posterior probability for that category is higher than a user-defined threshold. D'Alessio et al. [12] propose a system in which, for a given category, the classification is based on a weighted sum of feature occurrences that should be greater than the category threshold. Both single and multiple classifications are possible for each document to be tested. The classification of a document proceeds top-down possibly through multiple paths. An innovative contribution of this work is the possibility of restructuring a given hierarchy or building a new one from scratch. Dumas and Chen [13] use the hierarchical

structure for two purposes: (i) training several SVMs, one for each intermediate node and (ii) classifying documents by combining scores from SVMs at different levels. The sets of positive and negative examples are built considering documents that belong to categories at the same level, and different feature sets are built, one for each category. Several combination rules have also been assessed. In the work of Ruiz and Srinivasan [26], a variant of the Hierarchical Mixture of Experts model is used. A hierarchical classifier combining several neural networks is also proposed in [30]. Gaussier et al. [15] propose a hierarchical generative model for textual data, i.e., a model for hierarchical clustering and categorization of co-occurrence data, focused on documents organization. In [25], a kernel-based approach for hierarchical text classification in a multi-label context is presented. The work demonstrates that the use of the dependency structure of microlabels (i.e., unions of partial paths in the tree) in a Markovian Network framework leads to improved prediction accuracy on deep hierarchies. Optimization is made feasible by utilizing decomposition of the original problem and making incremental conditional gradient search in the subproblems. Ceci and Malerba [9] present a comprehensive study on hierarchical classification of Web documents. They extend a previous work [8] considering: (i) hierarchical feature selection mechanisms; (ii) a naïve Bayes algorithm aimed at avoiding problems related to different document lengths; (iii) the validation of their framework for a probabilistic SVM-based classifier; and (iv) an automated threshold selection algorithm. More recently, in [14], the authors propose a multi-label hierarchical text categorization algorithm consisting of a hierarchical variant of ADABOOST.MH, a well-known member of the family of "boosting" learning algorithms. Bennet and Nguyen [6] study the problem of the error propagation under the assumption that the "higher" the node is in the hierarchy, the worse the mistake is. The authors also study the problem of dealing with increasingly complex decision surfaces. Brank et al. [7] deal with the problem of classifying textual documents into a topical hierarchy of categories. They construct a coding matrix gradually, one column at a time, each new column being defined in a way that the corresponding binary classifier attempts to correct the most common mistakes of the current ensemble of binary classifiers. The goal is to achieve good performance while keeping reasonably low the number of binary classifiers.

2.2 Threshold Selection Strategies

In TC, the three most commonly used thresholding strategies are RCut, PCut, and SCut [35]. For each document, RCut sorts categories by score and selects the *t* top-ranking categories, with $t \geq 1$ (however, as noted in [28], RCut is not a strict thresholding policy). For each category C_j, PCut sorts the documents by score and sets the threshold of C_j so that the number of documents accepted by C_j corresponds to the number of documents assigned to C_j. For each category C_j, SCut scores a validation set of documents and tunes the threshold over the local pool of scores, until a suboptimal, though satisfactory, performance of the classifier is obtained for C_j.

Few threshold selection algorithms have been proposed for HTC [12] [27] [9]. The algorithm proposed by D'Alessio et al. [12] tunes the thresholds by considering categories in a top-down fashion. For each category C_j, the search space is visited by incrementing the corresponding threshold with steps of 0.1. For each threshold value,

the number of True Positives (TP) and False Positives (FP), i.e., the number of documents that would be correctly and incorrectly placed in C_j, is calculated. The utility function, i.e., the goodness measure that must be maximized for each threshold, is $TP - FP$ (in the event that the same value of the utility function occurs multiple times, the lowest threshold with that value is selected). Ruiz [27] selects thresholds that optimize the F_1 values for the categories, using the whole training set to identify the (sub)optimal thresholds. His expert-based system takes a binary decision at each expert gate and then optimizes the thresholds using only examples that reach leaf nodes. This task is performed by grouping experts into levels and finding the thresholds that maximize F_1. The best results are selected upon trials performed with each combination of thresholds from the vector $[0.005, 0.01, 0.05, 0.10]$ for level 1 and $[0.005, 0.01, 0.05, 0.10, 0.15, \ldots, 0.95]$ for levels 2, 3, 4. The algorithm proposed by Ceci and Malerba [9] is based on a recursive bottom-up threshold determination. The algorithm takes as input a category C and the set of thresholds already computed for some siblings of C and their descendants. It returns the union of the input set with the set of thresholds computed for all descendants of C. In particular, if C' is a direct subcategory of C, the threshold associated to C' is determined by examining the sorted list of classification scores and by selecting the middle point between two values in the list, to minimize the expected error. The error function is estimated on the basis of the distance between two nodes in a tree structure (TD), the distance being computed as the sum of the weights of all edges of the unique path connecting the two categories in the hierarchy (a unit weight is associated to each edge).

2.3 Input Imbalance

High imbalance occurs in real-world domains where the decision system is aimed at detecting rare but important cases [18]. Imbalanced datasets exist in many real-world domains, such as spotting unreliable telecommunication customers, detection of oil spills in satellite radar images, learning word pronunciations, text classification, detection of fraudulent telephone calls, information retrieval and filtering tasks, and so on [32] [34]. Japkowicz [16] contributed to study the class imbalance problem in the context of binary classification, the author studied the problem related to domains in which one class is represented by a large number of examples whereas the other is represented by only a few.

A number of solutions to the class imbalance problem have been proposed both at the data- and algorithmic-level [20] [10] [19]. Data-level solutions include many different forms of resampling such as random oversampling with replacement, random undersampling, directed oversampling, directed undersampling, oversampling with informed generation of new samples, and combinations of the above techniques. To counteract the class imbalance, algorithmic-level solutions include adjusting the costs of the various classes, adjusting the decision threshold, and adopting recognition-based, rather than discrimination-based, learning. Hybrid approaches have also been used to deal with the class imbalance problem.

3 Progressive Filtering

Progressive Filtering (PF) is a simple categorization technique framed within the local classifier per node approach, which admits only binary decisions. In PF, each classifier is entrusted with deciding whether the input in hand can be forwarded or not to its children. The first proposals in which sequential boolean decisions are applied in combination with local classifiers per node can be found in [12], [13], and [29]. In [31], the idea of mirroring the taxonomy structure through binary classifiers is clearly highlighted; the authors call this technique "binarized structured label learning".

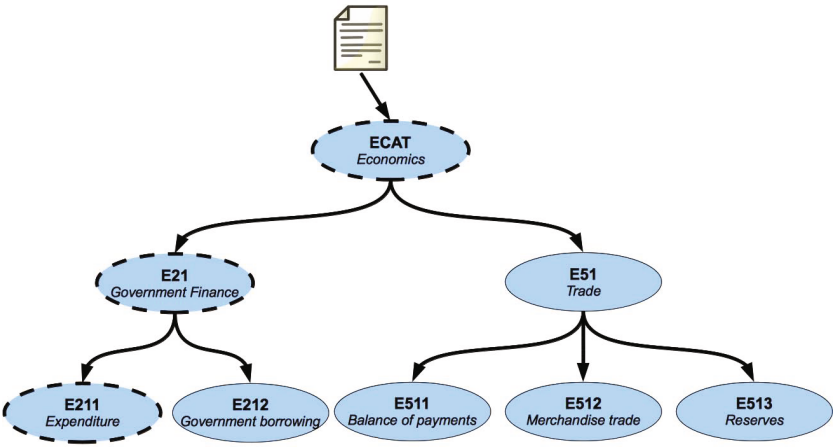

Fig. 1. An example of PF (highlighted with bold-dashed lines)

In PF, 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 is passed 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 accepted the token. Figure 1 gives an example of how PF works. A theoretical study of the approach is beyond the scope of this chapter, the interested reader could refer to [4] for further details.

3.1 The Approach

A simple way to implement PF consists of unfolding the given taxonomy into pipelines of classifiers, as depicted in Figure 2. 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). Partitioning the taxonomy in pipelines gives rise to a set of new classifiers, each represented by a pipeline.

Finally, let us note that the implementation of PF described in this chapter performs a sort of "flattening" though *preserving* the information about the hierarchical relationships embedded in a pipeline. For instance, the pipeline $\langle ECAT, E21, E211 \rangle$ actually

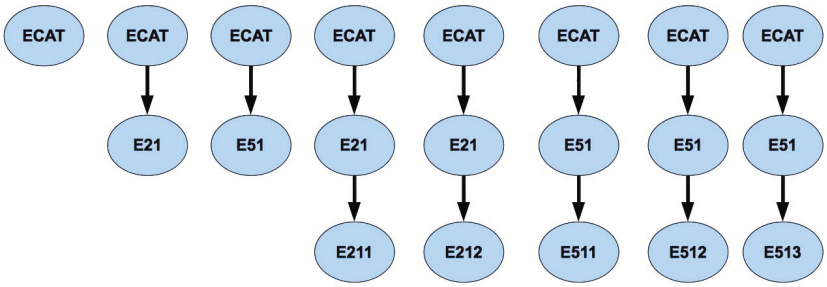

Fig. 2. The pipelines corresponding to the taxonomy in Figure 1

represents the classifier $E211$, although the information about the existing subsumption relationships are preserved (i.e., $E211 \prec E21 \prec ECAT$, where "$\prec$" denotes the usual covering relation).

3.2 The Adopted Threshold Selection Algorithm

According to 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, with $d \in D$, $CSV_i(d) \geq \theta_i$ 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, we assume that CSV_i exists, with the same semantics adopted by the classical setting. Considering a pipeline π, composed by n classifiers, the acceptance policy strictly depends on the vector of thresholds $\theta = \langle \theta_1, \theta_2, \ldots, \theta n \rangle$ that embodies the thresholds of all classifiers in π. In order to categorize d under π, the following constraint must be satisfied: for $k = 1 \ldots n$, $CSV_i(d) \geq \theta_k$. On the contrary, d is not categorized under c_i in the event that a classifier in π rejects it. Let us point out that we allow different behaviors for a classifier, depending on which pipeline it is embedded in. As a consequence, each pipeline can be considered in isolation from the others. For instance, given $\pi_1 = \langle ECAT, E21, E211 \rangle$ and $\pi_2 = \langle ECAT, E21, E212 \rangle$, the classifier $ECAT$ is not compelled to have the same threshold in π_1 and in π_2 (the same holds for $E21$).

In PF, given a utility function[1], 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. 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 virtually infinite. However, the brute-force approach can be approximated by defining a granularity step that requires to check only a finite number of points in the range $[0, 1]$, in which the thresholds are permitted to vary with step δ. Although potentially useful, this "relaxed" brute force

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

algorithm for calibrating thresholds (RBF for short) is still too heavy from a computational point of view. On the contrary, the threshold selection algorithm described in this chapter is characterized by low time complexity while maintaining the capability of finding near-optimum solutions.

Utility functions typically adopted in TC and in HTC are nearly-convex with respect to the acceptance threshold. Due to the shape of the utility function and to its dependence on false positives and false negatives, it becomes feasible to search its maximum around a subrange of $[0, 1]$. Bearing in mind that the lower the threshold the less restrictive is the classifier, we propose a greedy bottom-up algorithm for selecting decision threshold that relies on two functions [3]:

- *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, calls $\mathcal{R}$ to calibrate the current classifier.

Given a pipeline $\pi = \langle C_1, C_2, \ldots, C_L \rangle$, *TSA* is defined as follows (all thresholds are initially set to 0):

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

which asserts 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:

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

and

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

where ";" denotes a sequence operator, meaning that in "*a;b*" action a is performed *before* action b.

The pseudo-code of *Calibrate* is:

```
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 (from *up* to *down* and vice versa). The reason why the direction of threshold optimization changes at each call of *Calibrate* (and hence of *Repair*) lies in the fact that increasing the threshold θ_{k-1} is expected to forward less FP to C_k, which allows to decrease θ_k. Conversely, decreasing the threshold θ_{k-1} is expected to forward more FP to C_k, which must react by increasing θ_k.

The pseudo-code of *Repair* is:

```
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 * p.sf and p.thresholds[level] in [0,1]
    do 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 scale factor (*p.sf*) is used to limit the impact of local minima during the search, depending on the adopted utility function (e.g., a typical value of *p.sf* for F_1 is 0.8).

It is worth pointing out that, as also noted in [21], the sub-optimal combination of thresholds depends on the adopted dataset, hence the sub-optimal combination of thresholds need to be recalculated for each dataset.

Searching for a sub-optimal combination of thresholds in a pipeline π is characterized by high time complexity. In particular, two sources of intractability hold: (i) the optimization problem that involves the thresholds and (ii) the need of retraining classifiers after modifying thresholds. In this work, we concentrate on the former issue while deciding not to retrain the classifiers. In any case, it is clear that the task of optimizing thresholds requires a solution that is computationally light. To calculate the computational complexity of TSA, 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 vary with step δ. As a consequence, $p = \lfloor \delta^{-1} \cdot (\rho_{max} - \rho_{min}) \rfloor$ is the maximum number of points to be checked for each classifier in a pipeline. For a pipeline π of length L, the expected running time for TSA, say $T_{TSA}(\pi)$, is proportional to $(L + L^2) \cdot p \cdot (\rho_{max} - \rho_{min})$. This implies that TSA has complexity $O(L^2)$,

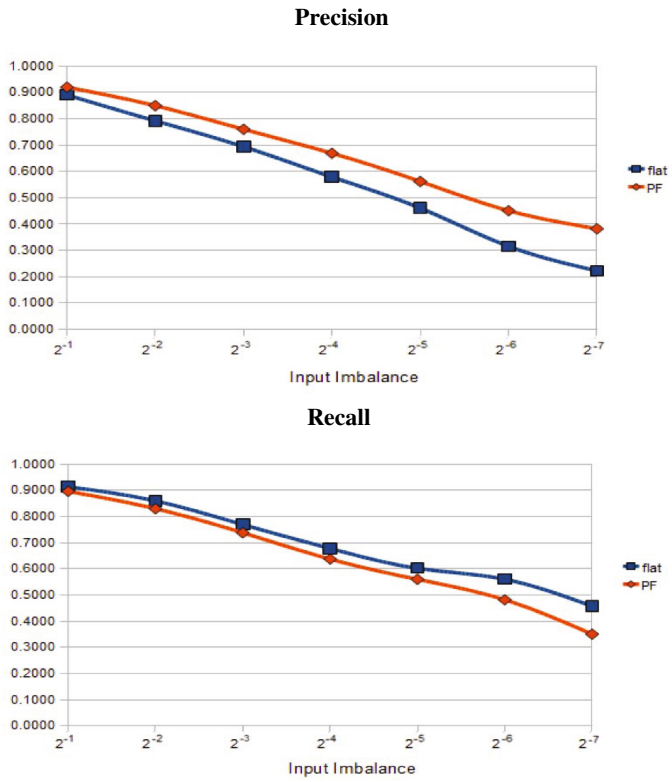

Fig. 3. Comparison of precision and recall between PF and flat classification

i.e., quadratic with the number of classifiers embedded by a pipeline. A comparison between TSA and the brute-force approach is unfeasible, as the elements of the threshold vector are real numbers. However, a comparison between TSA and RBF is feasible, although RBF is still computationally heavy. Assuming that p points are checked for each classifier in a pipeline, the expected running time for RBF, $T_{RBF}(\pi)$, is proportional to p^L, which implies that its computational complexity is $O(p^L)$.

4 Experiments and Results

The Reuters Corpus Volume I (RCV1-v2) [22] has been chosen as benchmark dataset. In this corpus, stories are coded into four hierarchical groups (a fragment of the taxonomy is reported in Figure 1: Corporate/Industrial (CCAT), Economics (ECAT), Government/Social (GCAT), and Markets (MCAT). Although the complete list consists of 126 categories, only some of them have been used to test 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 belongs to at least one category and, on average, to 3.8 categories.

Reuters dataset has been chosen because it allows us to perform experiments with pipelines up to level 4 while maintaining a substantial number of documents along the pipeline. To perform experiments, we used, about 2,000 documents per each leaf category. We considered the 24 pipelines of depth 4 yielding a total of about 48,000 documents.

Experiments have been performed on a SUN Workstation with two Opteron 280, 2Ghz+ and 8Gb Ram. The system used to perform benchmarks has been implemented using X.MAS [1], a generic multiagent architecture built upon JADE [5] 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 [11], 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. After determining the overall sets of features, their values are computed for each document resorting to the well-known TFIDF method. To reduce the high dimensionality of the feature space, we select the features that represent a node by adopting the information gain method.

Experiments have been performed to validate the proposed approach with respect to the impact of PF in the input imbalance. To this end, three series of experiments have been performed: first, performances calculated resorting to PF have been compared with those calculated by resorting to the corresponding flat approach. Then, PF has been tested to assess the improvement of performances while augmenting the pipeline depth. Finally, performances have been calculated in terms of generalization- / specialization- / misclassification-error and unknown-ratio.

Furthermore, to show that the overall performances of PF are not worsened by the adoption of TSA, we performed further experiments, on a balanced dataset of 2000 documents for each class, focused on (i) comparing the running time and F_1 of TSA vs. RBF, and on (ii) comparing TSA with the selected state-of-the-art algorithms, i.e., those proposed by D'Alessio et al. [12], by Ruiz [27], and by Ceci and Malerba [9].

4.1 Experimenting Progressive Filtering in Presence of Input Imbalance

The main issue being investigated is the effectiveness of PF with respect to flat classification. In order to make a fair comparison, the same classification system has been adopted, i.e., a classifier based on the *wk*-NN technology [11]. The motivation for the adoption of this particular technique stems from the fact that it does not require specific training and is very robust with respect to noisy data. In fact, as demonstrate in [33] *wk*-NN-based approaches can reduce the error rate due to robustness against outliers.

During the training activity, first, each classifier is trained with a balanced data set of 1000 documents by using 200 (TFIDF) features selected in accordance with their information gain. For any given node, the training set contains documents taken from the corresponding subtree and documents of the sibling subtrees –as positive and negative examples, respectively. Then, the best thresholds are selected. Both the thresholds of the pipelines and of the flat classifiers have been chosen by adopting F_1 as utility function[2]. As for pipelines, we used a step δ of 10^{-4} for TSA.

[2] The utility function can be adopted depending on the constraint imposed by the given scenario. For instance, F_1 is suitable if one wants to give equal importance to precision and recall.

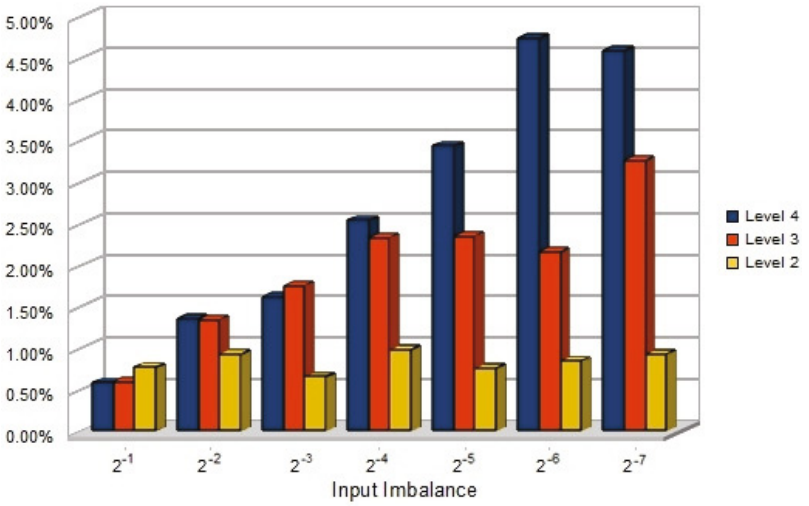

Fig. 4. Performance improvement

Experiments have been performed by assessing the behavior of the proposed hierarchical approach in presence of different ratios of positive examples versus negative examples, i.e., from 2^{-1} to 2^{-7}. We considered only pipelines that end with a leaf node of the taxonomy. Accordingly, for the flat approach, we considered only classifiers that correspond to a leaf.

PF vs Flat Classification. Figure 3 shows macro-averaging of precision and recall. Precision and recall have been calculated for both the flat classifiers and the pipelines by varying the input imbalance. As pointed out by experimental results (for precision), the distributed solution based on pipelines has reported better results than those obtained with the flat model. On the contrary, results on recall are worse than those obtained with the flat model.

Improving Performance along the Pipeline. Figure 4 shows the performance improvements in terms of F_1 of the proposed approach with respect to the flat one. The improvement has been calculated in percentage with the formula $(F_1(pipeline) - F_1(flat)) \times 100$. Experimental results –having the adopted taxonomies a maximum depth of five– show that PF performs always better than the flat approach.

Hierarchical Metrics. Figure 5 depicts the results obtained varying the imbalance. Analyzing the results, it is easy to note that the generalization-error and the misclassification-error grow with the imbalance, whereas the specialization-error and the unknown-ratio decrease. As for the generalization-error, it depends on the overall number of false negatives (FNs), the greater the imbalance the greater the amount of FNs. Hence, the generalization-error increases with the imbalance. In presence of input imbalance, the trend of the generalization-error is similar to the trend of the recall measure. As for the specialization-error, it depends on the overall number of false positives (FPs), the greater the imbalance, the lower the amount of FPs. Hence, the specialization-error

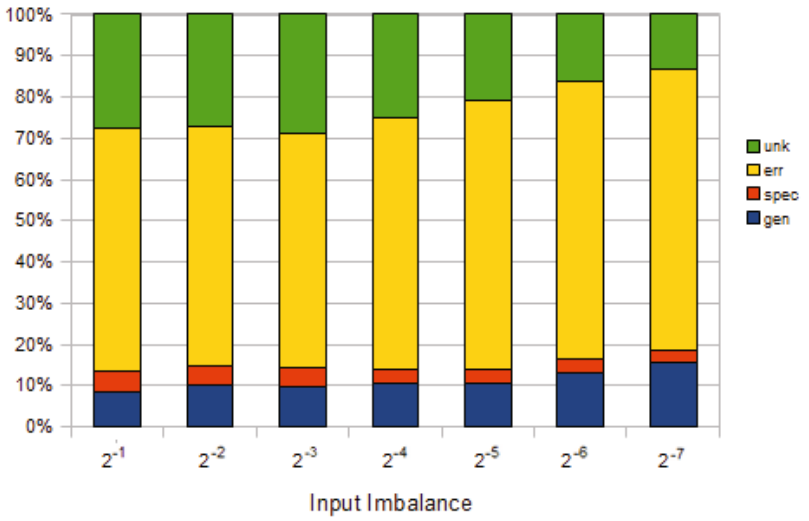

Fig. 5. Hierarchical measures

decreases with the imbalance. In presence of input imbalance, the trend of the specialization-error is similar to the trend of the precision. As final remark, let us note that different utility functions can be adopted, depending on which aspect or unwanted effect one wants to improve or mitigate. For instance, recall could be adopted as utility function while trying to reduce the number of FNs. This is due to the fact that recall optimization is biased against FNs (with the typical unwanted effect of increasing FPs). In this way, the unknown-ratio (which depends on FNs) decreases, while the misclassification-error (which depends on FPs) increases.

4.2 Comparative Experiments of Threshold Selection Strategies

TSA vs. RBF. Experiments performed to compare TSA with RBF have been carried out calculating the time (in milliseconds) required to set the optimal vector of thresholds for both algorithms, i.e., the one that reaches the optimal value in term of F_1, the selected utility function. Different values of δ (i.e., 0.1, 0.05, 0.01) have been adopted to increment thresholds during the search. Each pair of rows in Table 1 reports the comparison in terms of the time spent to complete each calibrate step ($t_{lev4} \ldots t_{lev1}$), together with the corresponding F_1. Results clearly show that the cumulative running time for RBF tends to rapidly become intractable[3], whereas the values of F_1 are comparable.

TSA vs. State-of-the-art Algorithms. As already pointed out, to compare TSA we considered the algorithms proposed by D'Alessio et al. [12], by Ruiz [27], and by Ceci and Malerba [9]. We used $\delta = 10^{-3}$ for TSA. Let us note that Ruiz uses the same threshold value for level 3 and level 4, whereas we let its algorithm to search on the entire space of thresholds. In so doing, the results in terms of utility functions cannot be

[3] Note that 1.9E+8 millisecond are about 54.6 hours.

Table 1. Comparisons between TSA and RBF (in milliseconds), averaged on pipelines with $L = 4$

Algorithm	$\mathbf{t_{lev4}}$	$\mathbf{t_{lev3}}$	$\mathbf{t_{lev2}}$	$\mathbf{t_{lev1}}$	$\mathbf{F_1}$
experiments with $\delta = 0.1$, $(p = 10)$					
TSA	33	81	131	194	0.8943
RBF	23	282	3,394	43,845	0.8952
experiments with $\delta = 0.05$, $(p = 20)$					
TSA	50	120	179	266	0.8953
RBF	35	737	17,860	405,913	0.8958
experiments with $\delta = 0.01$, $(p = 100)$					
TSA	261	656	1,018	1,625	0.8926
RBF	198	17,633	3.1E+6	1.96E+8	0.9077

Table 2. Comparisons between TSA and three state-of-the-art algorithms (UF stands for Utility Function)

UF: F1	F1	TP-FP	TD	Time (s)
TSA	**0.9080**	814.80	532.24	1.74
Ceci & Malerba	**0.0927**	801.36	545.44	0.65
Ruiz	**0.8809**	766.72	695.32	29.39
D'Alessio et al.	**0.9075**	812.88	546.16	14.42
UF: TP-FP	F1	TP-FP	TD	Time (s)
TSA	0.9050	**813.36**	521.48	1.2
Ceci & Malerba	0.9015	**802.48**	500.88	1.14
Ruiz	0.8770	**764.08**	675.20	24.4
D'Alessio at al.	0.9065	**812.88**	537.48	11.77
UF: TD	F1	TP-FP	TD	Time (s)
TSA	0.8270	704.40	**403.76**	1.48
Ceci & Malerba	0.8202	694.96	**404.96**	0.62
Ruiz	0.7807	654.72	**597.32**	26.31
D'Alessio et al.	0.8107	684.78	**415.60**	13.06

worse than those calculated by means of the original algorithm. However, the running time is an order of magnitude greater than the original algorithm.

For each algorithm, we performed three sets of experiments in which a different utility function has been adopted. The baseline of our experiments is a comparison among the four algorithms. In particular, we used: $F1$, according to the metric adopted in a previous work on PF [2] and in [27]; $TP - FP$, according to the metric adopted in [12]; and TD, according to the metric adopted in [9].

As reported in Table 2, for each experimental setting, we calculated the performance and the time spent for each selected metric. Table 2 summarizes the results. For each experimental setting, the most relevant results (highlighted in bold in the table) correspond to the metric used as utility function. As shown, TSA always performs better in terms of $F1$, $TP - FP$, and TD. As for the running time, the algorithm proposed by Ceci and Malerba shows the best performance.

5 Conclusions

In this chapter, we made experiments on PF to investigate how the ratio between positive and negative examples affects the performances of a classifier system and how these performances can be improved by adopting PF instead of a classical flat approach. Results show that the proposed approach is able to deal with high imbalance between negative and positive examples.

Acknowledgements. This research was partly sponsored by the Autonomous Region of Sardinia (RAS), through a grant financed with the "Sardinia POR FSE 2007-2013" funds and provided according to the L.R. $7/2007$ "Promotion of the Scientific Research and of the Technological Innovation in Sardinia".

References

1. Addis, A., Armano, G., Vargiu, E.: From a generic multiagent architecture to multiagent information retrieval systems. In: AT2AI-6, Sixth International Workshop, From Agent Theory to Agent Implementation, pp. 3–9 (2008)
2. Addis, A., Armano, G., Vargiu, E.: Assessing progressive filtering to perform hierarchical text categorization in presence of input imbalance. In: Proceedings of International Conference on Knowledge Discovery and Information Retrieval, KDIR 2010 (2010)
3. Addis, A., Armano, G., Vargiu, E.: A Comparative Experimental Assessment of a Threshold Selection Algorithm in Hierarchical Text Categorization. In: Clough, P., Foley, C., Gurrin, C., Jones, G.J.F., Kraaij, W., Lee, H., Mudoch, V. (eds.) ECIR 2011. LNCS, vol. 6611, pp. 32–42. Springer, Heidelberg (2011)
4. Armano, G.: On the progressive filtering approach to hierarchical text categorization. Tech. rep., DIEE - University of Cagliari (2009)
5. Bellifemine, F., Caire, G., Greenwood, D. (eds.): Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology). John Wiley and Sons (2007)
6. Bennett, P.N., Nguyen, N.: Refined experts: improving classification in large taxonomies. In: SIGIR 2009: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, pp. 11–18. ACM, New York (2009)
7. Brank, J., Mladenic, D., Grobelnik, M.: Large-scale hierarchical text classification using svm and coding matrices. In: Large-Scale Hierarchical Classification Workshop (2010)
8. Ceci, M., Malerba, D.: Hierarchical Classification of HTML Documents with WebClassII. In: Sebastiani, F. (ed.) ECIR 2003. LNCS, vol. 2633, pp. 57–72. Springer, Heidelberg (2003)
9. Ceci, M., Malerba, D.: Classifying web documents in a hierarchy of categories: a comprehensive study. Journal of Intelligent Information Systems 28(1), 37–78 (2007)
10. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: SMOTE: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research 16, 321–357 (2002)
11. Cost, R.S., Salzberg, S.: A weighted nearest neighbor algorithm for learning with symbolic features. Machine Learning 10, 57–78 (1993)
12. D'Alessio, S., Murray, K., Schiaffino, R.: The effect of using hierarchical classifiers in text categorization. In: Proceedings of of the 6th International Conference on Recherche dInformation Assiste par Ordinateur (RIAO), pp. 302–313 (2000)
13. Dumais, S.T., Chen, H.: Hierarchical classification of Web content. In: Belkin, N.J., Ingwersen, P., Leong, M.K. (eds.) Proceedings of 23rd ACM International Conference on Research and Development in Information Retrieval, SIGIR 2000, pp. 256–263. ACM Press, New York (2000)

14. Esuli, A., Fagni, T., Sebastiani, F.: Boosting multi-label hierarchical text categorization. Inf. Retr. 11(4), 287–313 (2008)
15. Gaussier, É., Goutte, C., Popat, K., Chen, F.: A Hierarchical Model for Clustering and Categorising Documents. In: Crestani, F., Girolami, M., van Rijsbergen, C.J.K. (eds.) ECIR 2002. LNCS, vol. 2291, pp. 229–247. Springer, Heidelberg (2002)
16. Japkowicz, N.: Learning from imbalanced data sets: a comparison of various strategies. In: AAAI Workshop on Learning from Imbalanced Data Sets (2000)
17. Koller, D., Sahami, M.: Hierarchically classifying documents using very few words. In: Fisher, D.H. (ed.) Proceedings of 14th International Conference on Machine Learning, ICML 1997, pp. 170–178. Morgan Kaufmann Publishers, San Francisco (1997)
18. Kotsiantis, S., Kanellopoulos, D., Pintelas, P.: Handling imbalanced datasets: a review. GESTS International Transactions on Computer Science and Engineering 30, 25–36 (2006)
19. Kotsiantis, S., Pintelas, P.: Mixture of expert agents for handling imbalanced data sets. Ann Math. Comput. Teleinformatics 1, 46–55 (2003)
20. Kubat, M., Matwin, S.: Addressing the curse of imbalanced training sets: One-sided selection. In: Proceedings of the Fourteenth International Conference on Machine Learning, pp. 179–186. Morgan Kaufmann (1997)
21. Lewis, D.D.: Evaluating and optimizing autonomous text classification systems. In: SIGIR 1995: Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 246–254. ACM, New York (1995)
22. Lewis, D.D., Yang, Y., Rose, T., Li, F.: RCV1: A new benchmark collection for text categorization research. Journal of Machine Learning Research 5, 361–397 (2004)
23. McCallum, A.K., Rosenfeld, R., Mitchell, T.M., Ng, A.Y.: Improving text classification by shrinkage in a hierarchy of classes. In: Shavlik, J.W. (ed.) Proceedings of 15th International Conference on Machine Learning, ICML 1998, pp. 359–367. Morgan Kaufmann Publishers, San Francisco (1998)
24. Mladenic, D., Grobelnik, M.: Feature selection for classification based on text hierarchy. In: Text and the Web, Conference on Automated Learning and Discovery CONALD 1998 (1998)
25. Rousu, J., Saunders, C., Szedmak, S., Shawe-Taylor, J.: Learning hierarchical multi-category text classification models. In: ICML 2005: Proceedings of the 22nd International Conference on Machine Learning, pp. 744–751. ACM, New York (2005)
26. Ruiz, M.E., Srinivasan, P.: Hierarchical text categorization using neural networks. Information Retrieval 5(1), 87–118 (2002)
27. Ruiz, M.E.: Combining machine learning and hierarchical structures for text categorization. Ph.D. thesis, supervisor-Srinivasan, Padmini (2001)
28. Sebastiani, F.: Machine learning in automated text categorization. ACM Computing Surveys (CSUR) 34(1), 1–55 (2002)
29. Sun, A., Lim, E.: Hierarchical text classification and evaluation. In: ICDM 2001: Proceedings of the 2001 IEEE International Conference on Data Mining, pp. 521–528. IEEE Computer Society, Washington, DC, USA (2001)
30. Weigend, A.S., Wiener, E.D., Pedersen, J.O.: Exploiting hierarchy in text categorization. Information Retrieval 1(3), 193–216 (1999)
31. Wu, F., Zhang, J., Honavar, V.: Learning Classifiers using Hierarchically Structured Class Taxonomies. In: Zucker, J.-D., Saitta, L. (eds.) SARA 2005. LNCS (LNAI), vol. 3607, pp. 313–320. Springer, Heidelberg (2005)
32. Wu, G., Chang, E.Y.: Class-boundary alignment for imbalanced dataset learning. In: ICML 2003 Workshop on Learning from Imbalanced Data Sets, pp. 49–56 (2003)

33. Takigawa, Y., Hotta, S., Kiyasu, S., Miyahara, S.: Pattern classification using weighted average patterns of categorical k-nearest neighbors. In: Proceedings of the 1th International Workshop on Camera-Based Document Analysis and Recognition, pp. 111–118 (2005)
34. Yan, R., Liu, Y., Jin, R., Hauptmann, A.: On predicting rare classes with svm ensembles in scene classification. In: Proceedings of 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP 2003), vol. 3, pp. III-21–III-4 (April 2003)
35. Yang, Y.: An evaluation of statistical approaches to text categorization. Information Retrieval 1(1/2), 69–90 (1999)