Improving Hierarchical SVMs by Hierarchy Flattening and Lazy Classification

Hassan H. Malik

Thomson Reuters, 195 Broadway, New York, NY 10007, USA
hassan.malik@thomsonreuters.com

Abstract. Hierarchical SVMs are well-known for their superior performance on text classification problems. They are especially useful on large-scale problems where training flat multi-class SVMs is often resource-prohibitive. However, Hierarchical SVMs suffer from compounding of errors with each hierarchy level which may negatively impact their classification performance. We propose k^{th}-level hierarchy flattening as a simple pre-training step that limits the compounding of errors, while retaining the computational benefits of Hierarchical SVMs. The results of experiments performed on LSHTC challenge datasets show that this simple step improved the classification accuracies and Macro-F_1 scores achieved by Hierarchical SVMs by up to 11% and 13% respectively.
We also propose a novel two-phased lazy classification scheme that first uses Hierarchical SVMs to select a subset of candidate classes for each test instance, and then trains a separate flat classifier for each subset. The new subset classifiers are then used to make the final predictions. Our experiments indicate that this scheme is particularly effective in improving the Macro-F_1 scores.

1 Introduction

The hierarchical classification scheme serves as a practical alternative to flat classification in large-scale settings where training a flat classifier requires substantial system resources [8]. For example, the large LSHTC dataset [9] for the basic task contains over 12K classes, 380K features and 128K training instances. Using this dataset, training a flat multi-class classifier (such as Naïve Bayes) with a typical in-memory implementation requires at least 17GB of space to store the classification model on a system with 32-bit numeric storage (i.e., 380K*12K*4 bytes). If a binary classifier for each class is trained instead (i.e., using the one-vs-rest scheme), all of the 128K instances are used to train each of the 12K+ binary classifiers.

In contrast, the hierarchical scheme trains an independent classifier for each internal node in the hierarchy. These classifiers are required to deal with substantially fewer classes, and with an exception of the root-level classifier, a subset of training instances and (possibly) a subset of features. As a result, the hierarchical scheme uses substantially fewer computational resources.

However, even though classifiers at each level in the hierarchical scheme are likely to be more accurate than the flat classifiers constructed from the same training dataset [4], the hierarchical scheme does not always result in better classification quality than the flat scheme because of the compounding of errors with each hierarchy level. Existing studies [4][14][2][8] had shown mixed results and the literature remains inconclusive.

In an attempt to combine the benefits of flat and hierarchical schemes, we propose flattening the original class hierarchy to k^{th}-level prior to training hierarchical classifiers (with k as a user-defined parameter). By limiting the hierarchy depth to the k^{th}-level, this method retains some of the computational benefits of the hierarchical scheme while reducing the loss of classification quality with each hierarchy level. In our experiments (Section 4.2), an appropriately selected value of k improved the classification accuracies and Macro-F_1 scores by up to 11% and 13% respectively over baseline Hierarchical SVMs.

Godbole et al. [4] proposed a two-step method that first uses a multi-class Naïve Bayes classifier to obtain a confusion graph of classes. For each class c, the confusion graph is used to obtain the set of classes S that were confused with c on training data. A separate multi-class SVM classifier is then trained for c. This classifier only considers c and classes in S. Test instances are classified by first applying the primary Naïve Bayes classifier to identify the top-scoring class, and then applying the secondary SVM classifier associated with that class. This method resulted in substantially faster training as compared to multi-class SVMs, and also performed better than multi-class Naïve Bayes in terms of classification accuracy. However, it did not always result in better classification accuracies as compared to multi-class SVMs [4]. When applied on the LSHTC dry-run dataset for the basic task (Section 4.3), this method did not improve the classification quality over baseline SVM methods.

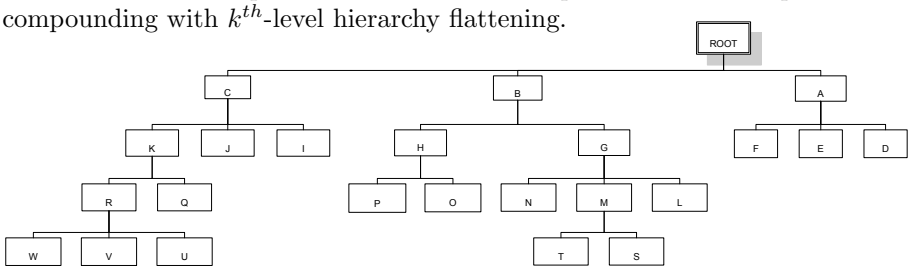
We observe that while the idea of training a secondary classifier of each class is promising because it allows these secondary classifiers to focus on a small subset of classes, it may not work well on large sparse datasets because of its reliance on training data to identify the classes used in constructing the secondary classifiers. The datasets used in [4] were filtered to exclude classes with less than 10 training instances. In contrast, sparse real-life datasets such as the datasets used in the LSHTC challenge may contain as few as two training instances for a significant percentage of classes which may not be sufficient to reliably construct confusion graphs.

Motivated by this observation, we propose a novel lazy classification scheme in this paper. Our scheme also uses primary and secondary classifiers, but does not rely on a confusion graph to identify classes used to construct the secondary classifiers. Instead of training both the primary and secondary classifiers in an eager fashion (i.e., as in [4]), our scheme defers the secondary classifier training to the classification phase. In the training phase, our scheme constructs a top-down hierarchical classifier in the usual way. To classify a test instance, our scheme first uses the hierarchical classifier to identify the most promising candidate classes for the test instance, and then trains a multi-class classifier that only considers

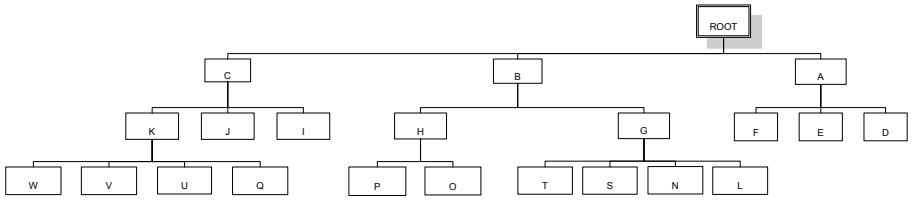
the selected classes. A parameter is used to restrict the maximum number of classes considered. The new classifier is then used to make the final prediction. Our experiments (Section 4.4) indicate that this scheme is particularly effective in improving the Macro-F_1 scores.

2 k^{th}-level Hierarchy Flattening

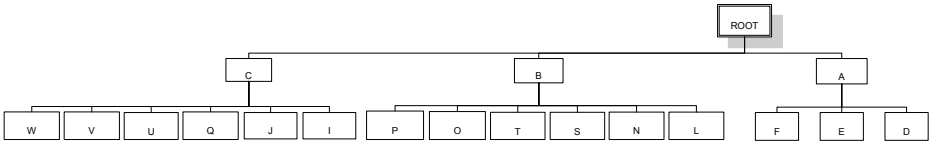
Deep hierarchies (such as The Open Directory [11]) are indeed useful for humans to efficiently navigate a large number of topics (and their associated documents) but they are not necessarily helpful to classification algorithms because of the increasing sparsity and compounding of errors with each hierarchy level, as we have discussed in the previous section. We attempt to reduce the impact of error compounding with k^{th}-level hierarchy flattening.



(a) Original hierarchy



(b) Flattened hierarchy, $k = 2$



(c) Flattened hierarchy, $k = 1$

Fig. 1: An example of hierarchy flattening.

k^{th}-level hierarchy flattening is a pre-training step that flattens the hierarchy under each node at the k^{th}-level in the original hierarchy (i.e., nodes that are at distance k from the root), where k is user-defined. If the documents are always assigned to leaf nodes (such as in the LSHTC datasets [9]), flattening may be achieved by adding all leaf nodes under the subtree of each node n at the k^{th}-level as the direct children of n, while dropping all the internal nodes in the subtree.

Figure 1 demonstrates hierarchy flattening at the first and second levels. Since the set of leaf nodes in the flattened hierarchy is exactly the same as the set of leaf nodes in the original hierarchy, hierarchy flattening does not modify the target classes for test instances in the common case, where test instances are always assigned to leaf nodes.

3 Lazy Classification Scheme

Considering a test instance t, our lazy classification scheme uses a hierarchical classifier to identify the set S of promising candidate classes for t. This is achieved by applying the top-down hierarchical classifier to t while recursively selecting up to n top-scoring classes at each level and adding all selected leaf-classes to S. Since we have used Hierarchical SVMs as the hierarchical classifier, we selected up to n top-scoring classes at each level that also triggered a positive score. A multi-class classifier is then trained with classes in S. This classifier is used to predict the final class for t. Algorithm 1 implements the lazy classification scheme. Functions "train" and "classify" corresponds to training and applying a multi-class classifier, respectively.

Algorithm 1: Classifying a test instance

```
   Input: root, t, n
   Output: predicted-class
 1 S = ∅
 2 add-candidate-classes(root, t, n, S)
 3 model = train(S)
 4 scores = apply(model, t)
 5 return the class with max score in scores
 6 function add-candidate-classes(node, t, n, S)
 7 begin
 8 |   scores = apply(node.model, t)
 9 |   P = ∅
10 |   add classes with a positive score in scores to P
11 |   sort classes in P in decreasing order of their scores
12 |   F = ∅
13 |   add top n classes in P to F
14 |   for class c ∈ F do
15 |   |   if isleaf(c) then
16 |   |   |   add c to S
17 |   |   end
18 |   |   else
19 |   |   |   add-candidate-classes(c.node, t, n, S)
20 |   |   end
21 |   end
22 end
```

Example: Considering the hierarchy in Figure 1(c), a test instance t and $n = 2$, assuming that all three classes at level-1 triggered a positive score for t, and class 'A' had the lowest score, the final set S of candidate classes may contain up to two top-scoring children of 'B' and up to two top-scoring children of 'C' that also triggered a positive score.

4 Experimental Evaluation

The evaluation was performed on the LSHTC challenge [9] datasets. The LSHTC challenge consists of four large-scale text classification tasks with partially overlapping data. The Open Directory [11] was used to construct content vectors and description vectors. Content vectors are feature vectors that were constructed by directly indexing the web pages and applying standard pre-processing, whereas description vectors are feature vectors that were constructed using the Open Directory description of web pages and categories. The "basic" task uses only the content vectors for training and testing. The "cheap" task uses description vectors for training but uses content vectors for testing. The expensive task uses both the content and description vectors for training but only uses content vectors for testing. The "full" task uses both the content and description vectors for training and testing. Two datasets are provided for each of these tasks. The "dry-run" datasets are provided with labels for test instances and contain 1,139 classes and about 6K training instances whereas the large datasets contain 12,294 classes and about 128K training instances.

The primary evaluation metrics used were accuracy, i.e., the fraction of correctly classified documents, and macro-averaged F_1 measure, i.e., the average of per-class harmonic mean of precision and recall scores without considering class sizes. Accuracy provides an idea of the overall classification performance whereas macro-averaged F_1 measure provides an idea of the classification performance on small classes. We additionally report tree-induced errors on the large datasets, i.e., the average of the number of edges in the path between each predicted label u and its corresponding actual label v.

The runtime environment consisted of a 64-bit Java Virtual Machine deployed on a dedicated 64-bit system with two 1.6GHz Intel quad-core processors. Parallelism was exploited to train and to apply classifiers, wherever possible.

4.1 Baseline Results

We first obtained the baseline classification accuracies, Macro-F_1 scores and training and test times of the flat and hierarchical variants of Naïve Bayes and SVMs on the dry-run dataset for the basic task (Figure 2). An independent multiclass classifier was trained for each hierarchy node in the hierarchical schemes, as usual. We have used the multinomial Naïve Bayes [10] as implemented in Weka [5], and used the Java version of the LibLinear [3] library for training linear SVMs in linear time [6]. The SVM regularization parameter C was automatically

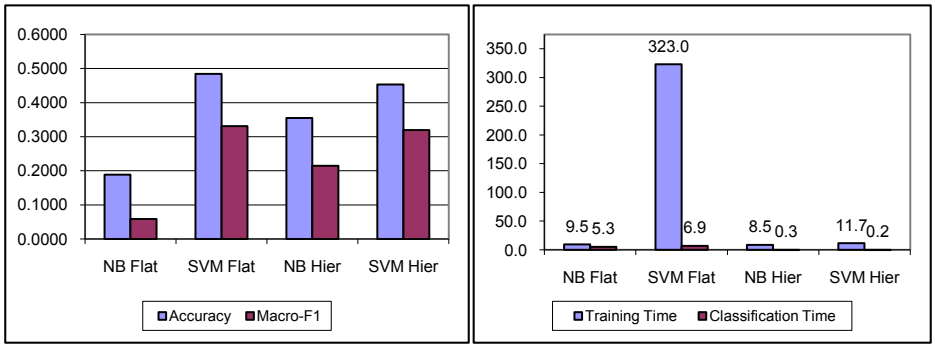


Fig. 2: Classification accuracies and Macro-F_1 scores of flat and hierarchical Naïve Bayes and SVMs on dry-run dataset for the basic task. All times are in seconds and the classification time represents the time to classify all test instances.

selected from the set $\{10^k | k = -5, ..., 2\}$ using a 4-fold cross-validation on the given training data.

Hierarchical Naïve Bayes outperformed flat Naïve Bayes, which is not surprising because Naïve Bayes is well-known to perform poorly on highly unbalanced training data because of its inherent bias that shrinks weights for classes with only a few training examples [12]. The hierarchical scheme often reduces the degree of class imbalance that each independent classifier deals with, resulting in improved classification performance. However, SVM-based methods outperformed Naïve Bayes with a significant margin. The flat multi-class SVMs resulted in slightly better classification accuracies and Macro-F_1 scores than Hierarchical SVMs but took substantially longer to train and classify (Section 1). Therefore, the remainder of our experiments focused on Hierarchical SVMs.

4.2 Hierarchy Flattening

In this section we compare the classification and runtime performance of Hierarchical SVMs using the original and flattened hierarchies. Since instances in the LSHTC datasets are always assigned to leaf nodes, the set of target classes in the flattened hierarchy remains the same as the original classification problem. Figure 3 presents these results on the large dataset for the basic task. Hierarchy flattening substantially improved the classification accuracy over the baseline and slightly improved the Macro-F_1 score but also increased the training and classification times. Flattening the LSHTC hierarchy using k=2 seems to provide the best classification quality-runtime tradeoff and we used this value for the rest of our experiments. Coincidentally, another LSHTC team [13] also explored hierarchy flattening and found the same value to be optimal on these datasets. Their method achieved better accuracies than reported here because

they flattened both the top and the bottom of the hierarchy whereas we only flattened the bottom of the hierarchy.

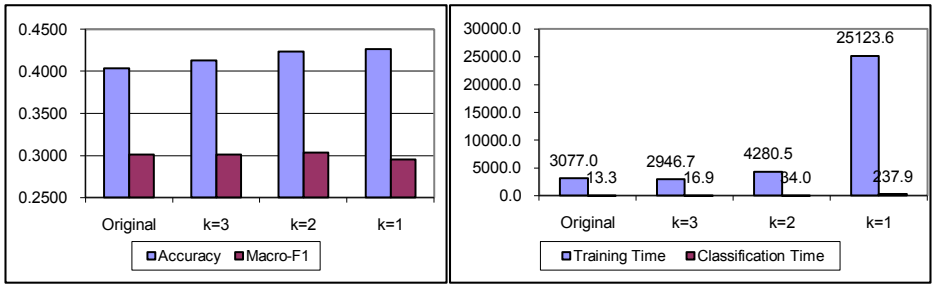


Fig. 3: Classification and runtime performance of Hierarchical SVMs on the large dataset for the basic task, using the original hierarchy and hierarchies flattened to three different levels.

Table 1 presents the hierarchy flattening results on dry-run and large datasets for the cheap, expensive and full tasks. Class descriptions were used as additional training instances for their corresponding classes (where available) for all results in this table (see Appendix A for results without class descriptions). Unlike the first task, hierarchy flattening not only improved the classification accuracies but also improved the Macro-F_1 scores. Most interestingly, hierarchy flattening improved the tree-induced errors in all cases which indicates that flattening individual subtrees within the hierarchy does not necessarily increase the chances of assigning test instances to wrong nodes that are farther from the ground-truth node in the original hierarchy.

We also observe that the classification performance on large datasets was always inferior to that on the dry-run datasets. This clearly demonstrates that the complexity of classification increases with the number of classes and the dataset size.

	Dry-run dataset				Large dataset				
	Accuracy	Macro F_1	Training time	Test time	Accuracy	Macro F_1	T. I. error	Training time	Test time
Cheap original	0.343	0.224	15.45	0.14	0.291	0.211	4.33	1354.3	9.2
Flattened	0.360	0.241	18.78	0.25	0.325	0.240	4.27	2394.0	25.1
Expensive original	0.485	0.350	14.03	0.19	0.412	0.314	3.46	2705.3	13.0
Flattened	0.491	0.358	16.22	0.34	0.434	0.322	3.41	4197.6	34.3
Full original	0.516	0.381	13.94	0.20	0.451	0.346	3.16	2644.7	13.4
Flattened	0.527	0.390	16.70	0.34	0.471	0.358	3.11	4238.0	34.8

Table 1: Classification and runtime performance of Hierarchical SVMs on datasets for the cheap, expensive and full tasks using the original hierarchy and the flattened hierarchy (using $k = 2$). Class descriptions were added as training instances, where available. T. I. error = Tree-induced Error.

4.3 GraphSVM

In this section we evaluate the GraphSVM algorithm [4] on the dry-run dataset for the basic task. The originally-proposed GraphSVM uses flat Naïve Bayes as the primary classifier to obtain the confusion graph, and then uses multi-class SVMs to construct secondary classifiers. In addition to the original algorithm, we also experimented with two variants of Graph SVM. The first variant used Hierarchical Naïve Bayes as the primary classifier and used multi-class SVMs to construct secondary classifiers whereas the second variant used Hierarchical SVMs as the primary classifier and used multi-class SVMs to construct secondary classifiers. The threshold parameter [4] was tuned for all GraphSVM variants and the best values were used for the results reported in this section.

	Accuracy	Macro F_1	Training time	Test time
NB Flat	0.188	0.058	9.5	5.32
NB Hierarchical	0.357	0.215	5.6	0.22
SVM Flat	0.484	0.331	322.96	6.89
SVM Hierarchical	0.476	0.331	12.2	0.62
Graph SVM original	0.406	0.248	398.98	15.93
Graph SVM with Hierarchical NB + MCSVM	0.425	0.284	67.35	2.36
Graph SVM with Hierarchical SVMs + MCSVM	0.478	0.331	57.3	1.27

Table 2: Classification and runtime performance of GraphSVM and its variants on the dry-run dataset of the basic task. All hierarchical methods used a hierarchy flattened to the second level.

Tables 2 presents the results of this experiment. We observe that the original GraphSVM yielded better classification quality than flat and hierarchical Naïve Bayes but it could not compete with SVM-based methods. This finding is consistent with the results reported in [4]. To our surprise, it was also the slowest method to train on this dataset which contradicts with the results in [4]. However, we found that because of the poor quality of the primary classifier (i.e., flat Naïve Bayes), the average number of classes each class was confused with was quite high and as a result, it took substantially longer to train the secondary SVM classifiers. Using better primary classifiers (such as the two GraphSVM variants that we have used) substantially reduces the training times.

The first GraphSVM variant that uses Hierarchical Naïve Bayes as the primary classifier performed better than the original GraphSVM but significantly worse than baseline SVM-based methods. The second variant that uses Hierarchical SVMs as the primary classifier resulted in classification quality that is comparable to baseline SVM-based methods. As we have discussed in Section 1, GraphSVM is unlikely to outperform the baseline SVM-based methods on large sparse datasets because it relies on the training data to identify the classes used in constructing the secondary classifiers. From Table 2, we also observe that flat methods (or methods that use a flat classifier as the primary classifier) are substantially more expensive than hierarchical classifiers to classify test instances, because of the large number of comparisons involved.

4.4 Lazy Classification

	Dry-run dataset				Large dataset				
	Accuracy	Macro F_1	Training time	Test time	Accuracy	Macro F_1	T. I. error	Training time	Test time
Basic	0.477	0.331	12.20	0.62	0.424	0.304	3.45	4245.1	34.0
Lazy, n=2	0.476	0.341	13.37	13.96	0.416	0.318	3.47	4245.1	3986.1
Lazy, n=4	0.476	0.348	13.37	27.90	0.417	0.327	3.46	4245.1	9561.4
Lazy, n=8	0.476	0.348	13.37	40.07	0.413	0.327	3.49	4245.1	19514.8
Lazy, n=20	0.479	0.350	13.37	45.73	0.414	0.331	3.50	4245.1	45910.5
Cheap	0.360	0.241	18.78	0.25	0.325	0.240	4.27	2394.0	25.1
Lazy, n=2	0.351	0.247	18.78	22.32	0.311	0.236	4.36	2394.0	2889.3
Lazy, n=4	0.355	0.256	18.78	92.40	0.310	0.255	4.35	2394.0	8618.0
Lazy, n=8	0.349	0.256	18.78	224.49	0.304	0.259	4.38	2394.0	20668.7
Expensive	0.491	0.358	16.22	0.34	0.434	0.322	3.41	4197.6	34.3
Lazy, n=2	0.491	0.364	16.22	16.70	0.423	0.330	3.45	4197.6	3441.4
Lazy, n=4	0.493	0.373	16.22	33.56	0.423	0.336	3.46	4197.6	9754.2
Lazy, n=8	0.494	0.369	16.22	52.48	0.423	0.340	3.48	4197.6	25355.2
Full	0.527	0.390	16.70	0.34	0.471	0.358	3.11	4238.0	34.8
Lazy, n=2	0.528	0.396	16.70	14.55	0.461	0.365	3.15	4238.0	3371.9
Lazy, n=4	0.526	0.400	16.70	31.54	0.460	0.370	3.16	4238.0	8814.5
Lazy, n=8	0.529	0.402	16.70	48.15	0.459	0.371	3.18	4238.0	19040.0

Table 3: Classification and runtime performance of the lazy classification scheme for several values of n on all LSHTC datasets. Lazy classification was applied to a hierarchy flattened to the second level. The baseline results (first row for each task) used the same hierarchy without lazy post-processing.

In this section we analyze the performance of our lazy classification scheme. As shown in Table 3, the lazy scheme generally improved the Macro-F_1 scores but negatively impacted the classification accuracies which indicates that the lazy scheme favors rare classes over common classes. However, it still achieved accuracies that are better than or comparable to the accuracies obtained on the original hierarchy (Figure 3 and Table 1). Since the lazy scheme dynamically trains a classifier for each test instance, it took longer to classify test instances. A real-life system may easily avoid most of these costs by caching classifiers that were trained for selected class subsets.

5 Related Work

Because of the unprecedented on-going growth in the sizes of real-life text databases, especially on the web, large-scale hierarchical text classification has gained considerable attention in recent years. However, as others have also noted [1][8], the classification quality achieved by the current methods is far from satisfactory and there is a strong need for methods that can classify large collections of documents in a reasonable amount of time with high classification quality. We highlight two recently proposed methods here.

Xue et al. [14] proposed a deep classification approach for large-scale hierarchical text classification. This approach consists of two stages that resemble stages in our lazy classification scheme. In the first stage, a category search algorithm is used to obtain a set of category candidates for a given test document. The hierarchy is then pruned to keep only the selected categories. The second stage uses the pruned hierarchy to classify the test document. The category search algorithm uses either a document-based strategy or a category-based strategy. The document-based strategy selects candidate categories by first computing cosine similarities between the test document and all training documents, and then selecting the categories assigned to the top-N most similar training documents. The category-based strategy follows a similar strategy, but instead computes the similarity between the test document and all categories. Both of these strategies are significantly more expensive than candidate category selection in out lazy classification scheme which uses a hierarchical classifier for this purpose, and only deals with a small subset of the classes and training instances.

In the refined experts method, Bennett et al. [1] reduced the error propagation by biasing the training distribution with a method called refinement, which uses the predicted behavior to change the training distribution and learns models that filters out errors. They further improved the classification quality by propagating-up first-guess "expert" information in a bottom-up fashion before making the final top-down choice. The expert information consists of a set of meta-features that contains membership predictions from lower nodes, which augments the current representation of a document. Since "refinement" is primarily useful on multi-label data, we were unable to evaluate it on the LSHTC datasets. However, we have evaluated the "experts" method on these datasets and it did not result in any noticeable improvement in the classification quality.

6 Conclusions and Future Work

In this paper, we proposed k^{th}-level hierarchy flattening, a simple yet effective pre-training step that improves Hierarchical SVMs by limiting the compounding of errors with each hierarchy level. On the LSHTC challenge datasets, k^{th}-level hierarchy flattening improved the classification accuracies and Macro-F_1 scores achieved by Hierarchical SVMs by up to 11% and 13% respectively.

We also demonstrated that GraphSVM may perform poorly on large sparse datasets because of its reliance on the training data to identify classes used in constructing secondary classifiers. We address this shortcoming with a novel two-phased lazy classification scheme that first uses Hierarchical SVMs to select a subset of candidate classes for each test instance, and then trains a separate flat classifier for each subset. The new subset classifiers are then used to make the final predictions. Our experiments indicate that this scheme is particularly effective in situations where all classes are considered equally important, regardless of their usage frequencies.

In the future, we plan to investigate more effective ways of flattening and simplifying hierarchies. We also plan to evaluate our lazy classification scheme

on additional flat and hierarchical datasets and with other classification algorithms. Finally, we plan to investigate more sophisticated ways of utilizing class description vectors.

Acknowledgments. We would like to thank the LSHTC organizers for setting up a well-defined and interesting challenge and for supporting us throughout the competition. We would also like to thank the anonymous reviewers for their constructive and detailed comments.

References

1. Bennett, P. N., Nguyen, N.: Refined experts: improving classification in large taxonomies. In Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval. 11–18 (2009)
2. Dumais, S., Chen, H.: Hierarchical classification of Web content. In Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval. 256–263 (2000)
3. Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., Lin, C.-J: LIBLINEAR: a library for large linear classification. Journal of Machine Learning Research, 9. 1871–1874 (2008)
4. Godbole, S., Sarawagi, S., Chakrabarti, S.: Scaling multi-class support vector machines using inter-class confusion. In Proceedings of the International Conference on Knowledge Discovery and Data Mining. 513–518 (2002)
5. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA Data Mining Software: An Update. ACM SIGKDD Explorations, 11 (1), 2009.
6. Joachims, T.: Training linear SVMs in linear time. In Proceedings of the International Conference on Knowledge Discovery and Data Mining. 217–226 (2006).
7. Lewis, D.D., Yang, Y., Rose, T., Li, F.: RCV1: a new benchmark collection for text categorization. Journal of Machine Learning Research, 5. 361–397 (2004).
8. Liu, T.-Y., Yang, Y., Wan, H., Zeng, H.-G., Chen, Z.,Ma, W.-Y.: Support vector machines classification with a very large-scale taxonomy. ACM SIGKDD Explorations, 7 (1). 36–43 (2005)
9. LSHTC Challenge: The Pascal Challenge on Large Scale Hierarchical Text classification. http://lshtc.iit.demokritos.gr/
10. McCallum, A., Nigam, K.: A comparison of event models for naïve bayes text classification. In Proceedings of AAAI-98 Workshop on Learning for Text Categorization, 1998.
11. ODP: The Open Directory Project. http://dmoz.org/
12. Rennie, J.D. Shih, L., Teevan, J., and Karger, D.: Tackling the poor assumptions of Naive Bayes text classifiers. In Proceedings of the 20th International Conference on Machine Learning (ICML). (2003).
13. Wang, X.-L., Lu, B.-L.,: Improved Hierarchical SVMs for Large-scale Hierarchical Text Classification challenge. Large scale hierarchical text classification challenge short paper. (2009)
14. Xue, G.-R., Xing, D., Yang, Q., Yu, Y.: Deep classification in large-scale text hierarchies. In Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval. 619–626 (2008)

A Using Class Descriptions as Training Instances

The datasets for three LSHTC tasks are provided with class descriptions for a subset of classes. We used these class descriptions as additional training instances. Table 4 presents the classification performance and training times of Hierarchical SVMs when applied to a hierarchy flattened to the second level, with and without these additional training instances. Class descriptions improved the classification accuracies, Macro-F_1 scores and tree-induced scores for all tasks with only a small impact on training times.

	Dry-run dataset			Large dataset			
	Accuracy	Macro F_1	Training time	Accuracy	Macro F_1	T. I. error	Training time
Cheap	0.334	0.213	16.8	0.319	0.228	4.30	2051.0
With class description	0.360	0.241	18.8	0.325	0.240	4.27	2394.0
Expensive	0.479	0.340	13.9	0.428	0.314	3.44	4071.4
With class description	0.491	0.358	16.2	0.434	0.322	3.41	4197.6
Full	0.511	0.368	13.6	0.462	0.340	3.15	3996.4
With class description	0.527	0.390	16.7	0.471	0.358	3.11	4238.0

Table 4: Classification performance and training times of Hierarchical SVMs with and without class descriptions for three LSHTC tasks.