Ensembles of Sparse Multinomial Classifiers for Scalable Text Classification

Antti Puurula and Albert Bifet

Department of Computer Science, The University of Waikato, Private Bag 3105, Hamilton 3240, New Zealand

Abstract. Machine learning techniques face new challenges in scalability to large-scale tasks. Many of the existing algorithms are unable to scale to potentially millions of features and structured classes encountered in web-scale datasets such as Wikipedia. The third Large Scale Hierarchical Text Classification evaluation (LSHTC3) evaluated systems for multi-label hierarchical categorization of Wikipedia articles. In this paper we present a broad overview of our system in the evaluation, performing among the top systems in the challenge. We describe the several new modeling ideas we used that make text classification systems both more effective and scalable. These are: reduction of inference time complexity for probabilistic classifiers using inverted indices, classifier modifications optimized with direct search algorithms, ensembles of diverse multi-label classifiers and a novel feature-regression based method for scalable ensemble combination.

Keywords: LSHTC3, scalability, Multinomial Naive Bayes, sparse representation, big data, ensemble methods, direct search optimization, random search, Feature-Weighted Linear Regression

1 Introduction

Large-scale hierarchical classification of text has become a contemporary topic over the last years due to web-scale processing of structured data, including web services such as Wikipedia and DMOZ. The third Large Scale Hierarchical Text Classification (LSHTC3) evaluation organized in 2012 had its first evaluation track dedicated to classification of unlabeled Wikipedia articles into the category structure of Wikipedia. This paper presents a broad overview of the contribution of one of the top-performing teams that participated in the challenge.

Comparing to the systems used in the earlier evaluation LSHTC2 [1, 2, 3], our approach brings a number of novel modeling ideas into the task:

1. Sparse inference.
   We use an exact classification algorithm utilizing the sparsity in text data, resulting in complexity lower than commonly assumed possible and depending on the data and model sparsities. This enables us better scaling to large problems, while maintaining effective performance.

2. Direct search optimization of classifier modifications.
   We use direct optimization with parallelized random search algorithms to develop and optimize our classifiers. This effectively merges the procedures of classifier development and optimization, providing classifiers modified for any measure of performance on a development dataset.
3. Ensemble of multi-label extension methods.
   We use extension methods well-known in multi-label classification to approach the hierarchical classification problem. In addition we use a KNN-based classifier as done by the leading contributions in LSHTC2. This provides us an ensemble of classifiers that are very diverse and well suited for model combination.
4. Scalable ensemble combination.
   We use both voting and improvements of this to combine the ensemble of classifier outputs, including the use of feature-based regression for modeling optimal classifier combinations for each instance. We present an ensemble combination method that scales freely to any size of problem, as it works on classifier outputs, independent of the original problem complexity.

The presented methods are both state-of-the-art in terms of effectiveness and truly scalable in terms of efficiency. In terms of effectiveness, the best of the classifiers in our ensemble would have come sixth in the LSHTC3 evaluation on its own, while the ensemble combination presented here would have come third. In terms of efficiency, our base-classifiers train on average using a couple of minutes with a single processor, while the ensemble combination models train in some seconds. The results presented here are somewhat better than our final submission to LSHTC3, due to an error in the ensemble combination that was not identified until after the evaluation deadline.

The rest of the paper proceeds as follow. Section 2 describes the used feature modeling, sparse inference with multinomial models and use of optimized modifications. Section 3 presents the scalable ensemble combination method. Section 4 shows experiment results on the LSHTC3 Wikipedia classification task, and Section 5 concludes the paper with a discussion.

2 Sparse Multinomial Classifiers

2.1 Multi-label Text Classification for LSHTC3

Multi-label classification of text involves categorization of documents to a set of labels, given a document. The set of labels is often expressed as a binary vector $\boldsymbol{l} = [l_1, ..., l_M]$ of label indicators, taking the value $l_m = 1$ when the label m assigned to a document and $l_m = 0$ otherwise. Since there are 2^M possible combinations of labels, in practice multi-label classification is approached by transforming the problem to a more tractable one. This is often done by extension methods applied to existing binary and multi-class classifiers [4, 5]. In some cases additional hierarhy information is available to constrain the allowed

labelsets. The classifiers we utilized are generic multi-label classifiers, not taking the LSHTC3 hierarchy information into account.

The common vector space model represents documents as vectors of features $\boldsymbol{w}$, of length $|\boldsymbol{w}| = N$. Most often each feature n gives a non-negative and normalized count w_n of a word in the document. The datasets in LSHTC3 evaluation consisted of a few passages of a Wikipedia article, available in both preprocessed word vector and original text forms. In addition to the preprocessed features (L3) we attempted other preprocessing configurations, keeping two variants that showed promise, named O1 and O2. Both used regular expression filtering, lowercasing and stopwording. O2 additionally used stemming with Porter-stemmer and short word removal.

Feature normalizations such as TF-IDF transforms are commonly used, as this substantially improves classification accuracy [6]. The basic form of TF-IDF that we later improve takes the form:

$$w_n = \log[1 + \frac{w_n^u}{s(\boldsymbol{w}^u)}] \log[\frac{D}{D_n}], \tag{1}$$

where w_n^u is the unmodified word count, D the number of training documents, D_n the number of training documents where $w_n^u > 0$ and $s(\boldsymbol{w}^u)$ is the number of non-zero values in $\boldsymbol{w}$. Using $s(\boldsymbol{w}^u)$ is also known as "$L0$-norm" and has been shown to be more consistent than $L1$ or $L2$-norm for text normalization [7].

2.2 Sparse Inference with Multinomial Naive Bayes

The classifiers we use build on the Multinomial Naive Bayes model for text classification. This is a generative model of the joint distribution $p(\boldsymbol{w}, m)$, defined for single label classification: $\sum_m l_m = 1$. Generative Bayes models consider the joint distribution $p(\boldsymbol{w}, m)$ to factorize $p(\boldsymbol{w}, m) = p(m)p_m(\boldsymbol{w})$, where $p(m)$ is a called prior model and each $p_m(\boldsymbol{w})$ the conditional models. MNB further parameterizes $p_m(\boldsymbol{w})$ with a Multinomial distribution[1], so that $p_m(\boldsymbol{w}) \propto \prod_{n=1}^{N} p_m(n)^{w_n}$. In summary, MNB takes the form:

$$p(\boldsymbol{w}, m) = p(m)p_m(\boldsymbol{w}) \propto p(m) \prod_{n=1}^{N} p_m(n)^{w_n}, \tag{2}$$

where the prior $p(m)$ is Categorical and the conditional $p_m(\boldsymbol{w})$ is Multinomial.

The time complexity of classification with Multinomial Naive Bayes is commonly considered to be $O(s(\boldsymbol{w})M)$, where $s(\boldsymbol{w})$ is the number of non-zero values

[1] In a strict sense the Multinomial distribution is not defined over non-negative real numbers. This is omitted in most discussion on MNB, as reformulation for fractional counts is not straightforward. For the purposes of classification, a sufficient correction would be multiplying all w_n by a large value to integers and correcting the prior probabilities $p(m)$ to take this into account. This model is defined over integers, but gives rank-equivalent results to using fractional counts.

in $\boldsymbol{w}$. We show in a separate publication [8] that this is not exactly the case, as the complexity in terms of the number of classes M can be reduced. This complexity reduction applies to linear classifiers, as well as the multi-label extensions used here. We briefly outline this algorithm in the following.

The textbook complexity $O(s(\boldsymbol{w})M)$ comes from computing a score for each label m by multiplying together the probabilities $p_m(n)$ for each $n : w_n > 0$. However, most probabilities $p_m(n)$ come from a smoothing distribution shared by the labels. With Jelinek-Mercer smoothing $p_m(n)$ decomposes into $p_m(n) = (1 - a)p^u_m(n) + ap^s(n)$, where $p^u_m(n)$ is the unsmoothed Multinomial, $p^s(n)$ is label-independent Multinomial for smoothing, and a the smoothing coefficient. With this sparse representation, it suffices to compute $p^s(n)$ for each $n : w_n > 0$ according to the smoothing distribution and then update these according to the non-zero parameters $p^u_m(n) > 0$. An inverted index can be used to find for each word $w_n > 0$ the relevant parameters, resulting in a reduced worst case time complexity $O(s(\boldsymbol{w}) + \sum_{n:w_n>0} \sum_{m:p^u_m(w_n)>0} 1)$. The resulting inference algorithm can be viewed as a document-at-a-time ranking algorithm [9] applied to classification with probabilistic models.

2.3 Modifications and Direct Search Optimization

The basic form of MNB is seldom used in practice. Smoothing is an example of a modification that is mandatory in real uses. Combinations of modifications can be used to substantially improve effectiveness of MNB, such as feature transforms and several smoothing methods. We use combinations of modifications, optimized on a held-out development sets for the performance measure of interest. In our earlier experiments with a number of datasets this approach lead to roughly over 20% more accurate classifiers compared to unoptimized MNBs. We list in the following the modifications we use, the resulting meta-parameters $\boldsymbol{a}$ and the used direct search optimization algorithm.

The modifications and required meta-parameters are:

1. Feature transforms. The TF-IDF in Equation 1 can be generalized
   (a) a_1 length_scale. TF-factor is replaced by: $\log[1 + \frac{w^u_n}{s(\boldsymbol{w}^u)^{a_1}}]/s(\boldsymbol{w}^u)^{1-a_1}$. Case $a_1 = 1$ normalizes length after log-transformation, $a_1 = 0$ before
   (b) a_2 idf_lift. IDF-factor is replaced by: $\log[\max(1, a_2 + \frac{D}{D_n})]$. Notable cases are Robertson-Walker $a_2 = 0$ and unsmoothed Croft-Harper $a_2 = -1$

2. Parameter pruning. Parameters $p^u_m(w_n)$ can be pruned during training
   (a) a_3 prune_count_insert. On each update parameters below a_3
   (b) a_4 prune_count_table. Before normalization parameters below a_4
   (c) a_5 min_count. Words with document count D_n below a_5
   (d) a_6 prune_labelset_count. Labelsets with document count below a_6

3. Parameter smoothing. Conditional and prior models can be smoothed

(a) a_7 cond_unif_weight. $p_m(n) = (1 - a_7)p^u_m(n) + a_7U$, U is Uniform
(b) a_8 cond_bg_weight. $p_m(n)$ is smoothed twice: $p_m(n) = (1-a_7-a_8)p^u_m(n)+ a_7U + a_8p^s(n)$, where $p^s(n)$ is a label-independent Multinomial
(c) a_9 cond_scale. $p^u_m(n)$ can be scaled before normalization: $p^u_m(n)^{a_9}$, with an effect similar to absolute discounting
(d) a_{10} prior_unif_weight. $p(\boldsymbol{l}) = (1 - a_{10})p^u(\boldsymbol{l}) + a_{10}U$, U is Uniform
(e) a_{11} prior_scale. Effect of prior can be scaled: $p(\boldsymbol{l})^{a_{11}}$

4. Inference pruning. Pruning of hypotheses can be used during classification
   (a) a_{12} eval_prune. Labels m can be ranked and sorted using the inverted index, and evaluations stopped if the mean log-probability of evaluated labels falls below a_{12} from the maximum found so far

The classifiers in the experiments used parameters useful for that type of classifier. For example, most classifiers do not benefit from using several of the pruning criteria a_3-a_6. In addition some of the extensions used one or two additional meta-parameters, such as the value of k for KNN classification. For each selected meta-parameter and classifier a maximum and minimum of permissible values were defined, as well as starting values for each optimization.

Direct search optimization [10, 11] with a random search algorithm [12, 13] was used to find the optimal meta-parameter configuration for each classifier, using accuracy measures on development data as the function value $f(\boldsymbol{a})$ to maximize. The used random search is a type of hill-climbing algorithm, where points $\boldsymbol{a}$ are generated by sampling points around the function maxima from previous iterations. With our classifiers each iteration generated 40 points in parallel, each point training and evaluating a classifier on a development set using the meta-parameter vector. In model development some tens to hundreds of iterations were used per classifier and optimizations were restarted a few times to reconfigure the meta-parameters.

The algorithm is described in a separate publication [8], but as an overview a couple of heuristics distinguish it from Steepest Ascent Hill-Climbing [11]. These are: use of multiple best points, generating points with a Bernoulli-Lognormal distribution, mirrored directions and adaptive stepsizes. Instead of a single point, we keep all new points sharing the current maximum value, and generate points uniformly from the current set of best points. New points are sampled by adding steps to current points, by generating step directions from a uniform Bernoulli distribution and step sizes from a Lognormal distribution for each parameter. Better sampling coverage is done by generating the steps in pairs of mirrored directions [13]. The stepsizes are also adapted, so that iterations improving the function value cause stepsizes to be increased by a small value, otherwise stepsizes are decreased per iteration.

2.4 Multi-label Extension Methods for Multinomial Naive Bayes

MNB is defined for single-label classification. A number of different extensions can be used to extend it for multi-label classification. We use five extensions:

1. Label Powerset (LP) [4] transforms a multi-label problem into a multi-class problem by mapping each unique labelset seen in training to a class.
2. Binary Relevance (BR) [5] transforms a multi-label problem into M independent binary classification problems. A label threshold parameter a_{13} can be used to improve combination of results, so that labels with scores deviating more than a_{13} from the maximum score found are pruned from the result.
3. Multi-label Mixture Model (MLMM) [14, 15, 8] works like Label Powerset, but decomposes the labelset-conditional probabilities $p_{\boldsymbol{l}}(n)$ into uniform mixtures of label-conditional probabilities $p_{\boldsymbol{l}}(n) = \sum_m p_m(n) / \sum l_m$. Classification is done by iteratively adding the component improving the probability the most, halting if the improvement was less than an iteration pruning parameter a_{14}.
4. Reverse Naive Bayes Label Powerset (RLP) [16] works like Label Powerset, but $p_m(n)$ are normalized to $p(m|n)$ instead of $p(n|m)$. This provides a linear classifier with a slightly different parameterization to MNB.
5. KNN-combination (MNB-KNN) estimates a MNB model for each document. In classification a ranked retrieval is first done, then from the top a_{15} documents the majority labelset is voted as the classification result.

3 Scalable Ensemble Combination

Ensemble methods have had a profound influence on machine learning over the last decade. In the context of scalable classification, many ensemble methods such as model averaging can be inconvenient to use, as base-classifiers will not efficiently return comparable scores over the full space of outputs. We describe next how Feature-Weighted Linear Regression (FWLR) [17] can be effectively adapted for scalable combination of classifier outputs, using a couple of novel techniques for generating reference weights and meta-features.

We start from the baseline of majority voting (vote) and describe incremental improvements to this method, named in parenthesis. The vote-method takes the output labelvector $\boldsymbol{g}_j = \boldsymbol{l}$ from the vector of outputs $\boldsymbol{G}$ and chooses the most common output as the combined result $\boldsymbol{l}' = \text{argmax}_{\boldsymbol{l}} \sum \boldsymbol{g}_{j:\boldsymbol{g}_j=\boldsymbol{l}} 1$. The baseline voting method resolves any ties from split decisions randomly. With a small number of base-classifiers, ties are common and resolving these with a heuristic can improve voting considerably.

For better tie resolution, we introduce *oracle weight estimation* for each classifier. We can use labeled data and the classifier outputs to generate reference weights, by classifying an input and then distributing weight uniformly to the classifiers giving the most accurate decision for a given accuracy measure. This provides for each document an approximate oracle reference of classifier weights. By summing the reference weights over a development dataset and normalizing $\sum_j v_j = 1$, we get weights v_j that can be used as tie-brakers (+tie_brakes). We can also use the weights directly for weighted voting (+weight_votes) $\boldsymbol{l}' = \text{argmax}_{\boldsymbol{l}} \sum \boldsymbol{g}_{j:\boldsymbol{g}_j=\boldsymbol{l}} v_j$.

One basic improvement for ensemble combination is greedy pruning of the base-classifiers, by iteratively removing a classifier from the ensemble until the

development score doesn't improve. With weighted voting this can still be performed efficiently (+selection).

Using the oracle reference weights, we can model weights v_{ij} dependent on a meta-feature vector $\boldsymbol{f}_j$, similarly to FWLR. This provides us a major improvement for combining the outputs: $\boldsymbol{l}'$: $\boldsymbol{l}' = \text{argmax}_{\boldsymbol{l}} \sum \boldsymbol{g}_{j:\boldsymbol{g}_j=\boldsymbol{l}} \sum_i v_{ij} f_{ij}$ (+FWLR_weights). Aside from the use for classification, one important difference to FWLR is that in our case the regression problems estimating the regressor weights $\boldsymbol{v}_j$ are considered independent. This simplifies estimation and enables approaching the regression problem with any models for regression designed with the Weka toolkit [18]. It also enables using a separate feature vector $\boldsymbol{f}_j$ for each classifier, reducing the number of features passed to each of the J independent regression problems.

The meta-feature vectors $\boldsymbol{f}_j$ can use any features related to the classification situation. In contrast to earlier work, we use the classifier outputs themselves to compute meta-features. The generated meta-features for each classifier are:

1. vote_count. The number of classifiers voting for the same output
2. mode_intersect. Number of shared labels with the most voted labelset
3. mode_jaccard. Jaccard similarity with the most voted labelset
4. j_intersect. $J-1$ features. Number of shared labels with output $\boldsymbol{g}_j$
5. j_jaccard. $J-1$ features. Jaccard similarity with output $\boldsymbol{g}_j$

The benefit of working solely on classifier outputs is that the ensemble combination is fully independent of the original classification task complexity. The number of classes and features that the base-classifiers work on is irrelevant for the model combination. Instead, the features seen by the ensemble model are the meta-features generated by the classifiers. The ensemble output variables are the vote weights, one per base-classifier. This makes the ensemble combination complexity depend solely on the number of classifiers, resulting in a fully scalable method.

4 Results

4.1 Ensemble Configuration

Figure 1 summarizes the processing stages used for the ensemble system. The LSHTC3 evaluation dataset consists of 456886 Wikipedia articles for training and 81262 for evaluation through an online oracle system. We further partitioned the training dataset to enable held-out optimization for both the base-classifiers and the ensemble. This randomly split the original training dataset to two sets, comb_train with 434042 documents and comb_dev with 22844 documents. For the base-classifiers we further split comb_train into 10 folds with 429042 documents per fold for training set and 5000 for development set each.

Based on earlier research, we started development using the given preprocessing (L3) and the Label Powerset (LP) classifier on training fold 0. We filled the other folds 9 folds with the other classifiers, attempting configurations that

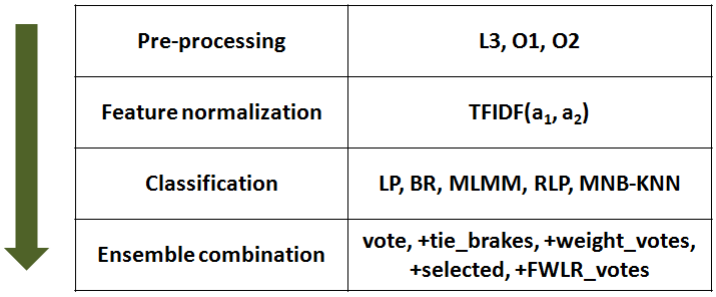

Fig. 1. Processing stages and options for the emsemble combination

showed both good performance and were different from the LP method with L3 features.

We first developed all classifiers for Micro-averaged F-score (miFscore) using optimizations with a few restarts for each type of classifier and preprocessing. As the LSHTC3 leaderboard was primarily sorted according to "Accuracy"[2], we decided to optimize the classifiers again for this measure (meanJaccard), resulting in 20 classifiers for ensemble combination. We used roughly ten days of processing time per each classifier on a i7-2600 CPU processor for developing the first 10 classifiers, 5 more days to optimize for meanJaccard. However, it is not possible to clearly separate classifier development from optimization, when classifier modifications are searched during optimization. Running 8 classifiers in parallel, mean training times for the final 20 classifiers were 99.1 s per classifier, mean classification times were 1.1 s per document on the evaluation set (L3_eval).

Table 1 summarizes the classifiers and shows the meanJaccard scores on comb_dev and L3_eval, as well as the weights v_j used in the vote tie-braking and vote-weighting improvements for ensemble combination. The oracle reference method for computing v_j used meanJaccard as the measure to maximize, distributing weight for each document in comb_dev uniformly among the classifiers with the highest Jaccard similarity to the reference labelset.

4.2 Ensemble Results

We combined the 20 base-classifier outputs using the described ensemble combination methods. Classifier selection removed 5 base-classifiers in the order "9, 19, 16, 17, 5". Interestingly, while none of the top-three individual classifiers was removed, 5 of the top 10 classifiers were, while none of the worst 10 classifiers was removed. This suggests that the diversity of base-classifiers is as important as having individual high-performers, as even simple mean estimation of weights

[2] Although the "Accuracy" naming is common in multi-label classification research, technically this is the mean of Jaccard scores, not accuracy as generally defined.

Table 1. Base-classifier meanJaccard scores and mean combination weights v_j

			miFscore(0-9)			meanJaccard(10-19)		
Classifiers	Features	Model	comb_dev	L3_eval	v_j	comb_dev	L3_eval	v_j
0, 10	L3	LP	0.405	0.384	0.042	0.408	0.388	0.044
1, 11	O1	BR	0.387	0.369	0.075	0.392	0.376	0.093
2, 12	O1	MLMM	0.397	0.387	0.057	0.397	0.387	0.056
3, 13	O2	MNB-KNN	0.381	0.363	0.050	0.379	0.360	0.049
4, 14	O1	MNB-KNN	0.387	0.369	0.048	0.390	0.373	0.050
5, 15	O1	LP	0.404	0.388	0.041	0.407	0.389	0.040
6, 16	O1	RLP	0.410	0.390	0.040	0.407	0.386	0.041
7, 17	O2	LP	0.404	0.382	0.044	0.406	0.387	0.043
8, 18	L3	MLMM	0.384	0.364	0.054	0.384	0.365	0.052
9, 19	L3	RLP	0.404	0.380	0.040	0.406	0.382	0.041

can utilize the weaker classifiers. This was seen in the unselected vote weights $\boldsymbol{v}_j$ in Table 1, where many of the weaker classifiers are given high voting weights.

For the selected 15 base-classifiers, FWLR_votes generated 256 meta-features from the outputs, passing 32 meta-features to each regression problem for the prediction of the base-classifier weight. Training the 15 regression models from base-classifier outputs were done with the Weka LinearRegression model, with ridge parameter $-R = 50$. This took 9 s on the comb_dev dataset, excluding the time for generation of the base-classifier outputs and meta-features.

Table 2 shows the meanJaccard results from ensemble combination and the differences in scores after each improvement. For reference the best single classifier and a gold-standard oracle combination method is shown, to show how much a perfect combination would achieve. For the comb_dev result of FWLR_votes, 10-fold cross-validation was used for independent parameter estimates.

Table 2. Ensemble methods and meanJaccard scores.

Combination	comb_dev	L3_eval	Δ comb_dev	Δ L3_eval
best classifier	0.4099	0.3896	-	-
vote	0.4268	0.4066	0.0169	0.0169
+tie_brakes	0.4283	0.4085	0.0184	0.0188
+weight_votes	0.4307	0.4113	0.0208	0.0217
+selected	0.4346	0.4150	0.0247	0.0254
+FWLR_votes	0.4447	0.4264	0.0348	0.0367
oracle	0.5849	-	0.1750	-

Comparing the base_classifier results to +FWLR_votes, a difference of 3.67% absolute meanJaccard score is seen from the use of ensemble combination. This is a 9.4% relative improvement in classification performance. Voting on its own improves the score by 1.69% absolute and the basic improvements up to model

selection increase this to 2.54%. The results are consistent on both development and evaluation sets, suggesting that the comb_dev dataset was more than sufficient in size to prevent overfitting.

5 Discussion

This paper presented a high-level description of the system and methods used in our submission to the LSHTC3 evaluation. Compared to the last year's systems, we have brought several new modeling ideas into the task that make large-scale text classification both more efficient and effective.

We used inference based on inverted indices to reduce time complexity of probabilistic classifiers, resulting essentially in probabilistic classifiers with the scalability of modern information retrieval systems. The classifiers were developed and optimized using parallel direct search algorithms in regards to the actual evaluation measure used in LSHTC3. An ensemble of classifiers was constructed, generating diversity of classifiers using different multi-label extension methods, feature preprocessing, different training data folds and optimization. Finally, the ensemble of classifiers was combined with a novel scalable ensemble method.

There are a number of possibilities for both extension of the overall system and the individual modeling ideas. For extending the system, larger ensembles could be constructed, as the individual classifiers are efficiently trained with minimal resource use. New types of feature preprocessing, normalization, model modification and multi-label extension could be introduced. Substantial improvements could come from more sophisticated ensemble combination methods, as the presented work has only scratched the surface of possibilities with ensemble modeling. Non-linear ensemble techniques could be combined with the regression models and better meta-features could be introduced.

In terms of performance, the final ensemble combination would have been a little more than one percent behind the leading LSHTC3 contribution, with meanJaccard score of 0.4381. Considering the effort put into improving our system, it will be interesting to learn what techniques were introduced by the other teams to achieve their scores. Most likely our system would have benefited from the use of hierarchy information of the labels, as smoothing of parameter estimates with the hierarchy information has been a prominent topic in the last years.

6 Acknowledgments

We would like to thank Quan Sun for suggesting the use of classifier selection.

References

[1] Wang, X.l., Zhao, H., Lu, B.L.: Enhance top-down method with meta-classification for very large-scale hierarchical classification. In: Proceedings of

5th International Joint Conference on Natural Language Processing, Chiang Mai, Thailand, Asian Federation of Natural Language Processing (November 2011) 1089–1097

[2] Grimal, C., Bisson, G.: Using a co-similarity approach on a large scale text categorization task. In: Seconde conférence sur les Modèles et l'Analyse des Réseaux : Approches Mathématiques et Informatique. (2011)

[3] Han, X., Liu, J., Shen, Z., Miao, C.: An optimized k-nearest neighbor algorithm for large scale hierarchical text classication. In: Proceedings of the Joint ECML/PKDD PASCAL Workshop on Large-Scale Hierarchical Classification. (September 2011) 2–12

[4] Boutell, M.R., Luo, J., Shen, X., Brown, C.M.: Learning multi-label scene classification. Pattern Recognition **37**(9) (2004) 1757 – 1771

[5] Godbole, S., Sarawagi, S.: Discriminative methods for multi-labeled classification. (2004) 22–30

[6] Rennie, J.D., Shih, L., Teevan, J., Karger, D.R.: Tackling the poor assumptions of naive bayes text classifiers. In: ICML'03. (2003) 616–623

[7] Singhal, A., Buckley, C., Mitra, M.: Pivoted document length normalization. In: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval. SIGIR '96, New York, NY, USA, ACM (1996) 21–29

[8] Puurula, A.: Scalable text classification with sparse generative modeling. In: Proceedings of the 12th Pacific Rim International Conference on Artificial Intelligence. Lecture notes in Artificial Intelligence, Springer (September 2012)

[9] Turtle, H., Flood, J.: Query evaluation: Strategies and optimizations. Information Processing & Management **31**(6) (1995) 831 – 850

[10] Powell, M.J.D.: Direct search algorithms for optimization calculations. Acta Numerica **7** (1998) 287–336

[11] Luke, S.: Essentials of Metaheuristics. Version 1.2 edn. Lulu (2009) Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[12] Li, J., Rhinehart, R.R.: Heuristic random optimization. Computers & Chemical Engineering **22**(3) (1998) 427 – 444

[13] Brunato, M., Battiti, R.: Rash: A self-adaptive random search method. In Cotta, C., Sevaux, M., Srensen, K., eds.: Adaptive and Multilevel Metaheuristics. Volume 136 of Studies in Computational Intelligence. Springer (2008) 95–117

[14] McCallum, A.: Multi-label text classification with a mixture model trained by EM. In: Proceedings of the AAAI' 99 Workshop on Text Learning. (1999)

[15] Ueda, N., Saito, K.: Parametric mixture models for multi-labeled text. In: In Advances in Neural Information Processing Systems 15, MIT Press (2002) 721–728

[16] Juan, A., Ney, H.: Reversing and smoothing the multinomial naive bayes text classifer. In: In Proceedings of the 2nd Int. Workshop on Pattern Recognition in Information Systems (PRIS 2002. (2002) 200–212

[17] Sill, J., Takács, G., Mackey, L., Lin, D.: Feature-weighted linear stacking. CoRR **abs/0911.0460** (2009)

[18] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. SIGKDD Explor. Newsl. **11**(1) (November 2009) 10–18