Multi-Value Classification of Very Short Texts

Andreas Heß, Philipp Dopichaj and Christian Maaß

Lycos Europe GmbH, Gütersloh, Germany
{andreas.hess,philipp.dopichaj,christian.maass}@lycos-europe.com

Abstract. We introduce a new stacking-like approach for multi-value classification. We apply this classification scheme using Naive Bayes, Rocchio and kNN classifiers on the well-known Reuters dataset. We use part-of-speech tagging for stopword removal. We show that our setup performs almost as well as other approaches that use the full article text even though we only classify headlines. Finally, we apply a Rocchio classifier on a dataset from a Web 2.0 site and show that it is suitable for semi-automated labelling (often called tagging) of short texts and is faster than other approaches.

1 Introduction

Multi-value text classification is an interesting and very practical topic. In many applications, a single label only is not enough to appropriately classify documents. This is especially true in many applications on the web. As opposed to traditional documents, some texts on the web, especially on Web 2.0 sites, are very short, for example pin-board entries, comments to blog posts or captions of pictures or videos. Sometimes these texts are mere snippets, being at most one or two sentences long. Yet, in some Web 2.0 applications, labelling or tagging such short snippets does not only make sense but could be the key to success. Therefore we believe it is important to investigate how multi-value text classification algorithms perform when very short texts are classified. To test this, we classified news articles from the well-known Reuters data-set based only on the headlines and compared the results to older approaches in literature that used the full text. We applied the same algorithm to a dataset from Web 2.0 site Lycos iQ. An empirical evaluation shows that text classification algorithms perform well in both setups.

The remainder of this paper is organised as follows: first, we present a new stacking approach for multi-value classification. By comparing the performance of classifiers trained only on the short headlines of the well-known Reuters news articles benchmark to results achieved with similar classifiers using the full article text we show that classification of very short texts is possible and the loss in accuracy is acceptable. Second, we present an application of text classification for tagging short texts from a Web 2.0-site. We demonstrate that presenting suggestions to the user can greatly improve the quality of tagging.

2 Stacking for Multi-Value Classification

In contrast to a standard single-value classification where each instance is assigned exactly one class label, multi-value (also called multi-label) classification allows for assigning an arbitrary number of labels to instances. A classical example where multi-value classification makes sense is the labelling of texts that have more than one topic. Class labels for such texts could be either of different granularity or they could even be orthogonal. For example, a news article about the presidential elections in the United States could be labelled as 'politics', 'election' and 'USA'. The labels 'politics' and 'election' are based on a topic, but 'election' is a more detailed description of the content. As opposed to the other labels, 'USA' refers to location.

Although multi-value classification is a natural solution for many machine-learning problems, most algorithms can only handle single-value classification. Therefore it is common practise in multi-value classification that single-value classification algorithms are adapted by means of some combination method; see [6] for a recent survey. The most common approach is 'one-vs-all' classification: for each class label, a binary classifier that decides whether an instance is a member of this specific class is trained. This approach has a strong disadvantage: the datasets the binary classifiers are trained on are imbalanced. Consider e. g. a kNN-classifier with $k = 3$. If an instance that is to be classified is equidistant to three instances that have a single class label each and are of three different classes, all three binary classifiers would classify the new instance as a negative example. Therefore, no prediction is made, although this is most probably not correct, and a more intuitive classification would be to assign all three class labels. This argument against the one-vs-all scheme holds for other classifiers as well. Support Vector Machines are known to be sensitive towards imbalanced datasets. Godbole and Sarawagi [3] exploit the relations between classes that exist if class labels are not independent from each other by using a stacking approach to add the predictions of other binary classifiers to train another set of classifiers in a second pass. Many classification algorithms output a ranked list of predicted class labels with confidence values. Well-known algorithms in this group are the Naive Bayes, Rocchio and kNN classifiers. When using such an algorithm, another scheme for multi-value classification is thresholding: selecting class labels that are within the top n predictions or have a confidence score higher than a certain threshold. Different methods of thresholding have been discussed by Yang [9]. The *SCut*-method applies a different threshold for each class. In the *RTCut*-method, rank and confidence score are combined to a single value before thresholding is applied.

2.1 Description of our Algorithm

We propose a multi-value classification scheme which we call MVS (multi-value classification stacking) that is similar to *RTCut*: we use a classifier with confidence scores and rank, but instead of creating an artificial score, we train a binary meta-classifier for each class on the confidence score and rank as computed by the base classifier. The meta-classifiers decide whether a specific class label predicted by the base classifier should be included in the final set of predictions. In our implementation, we used JRip as implemented in WEKA [7] as meta-classifier. To train our algorithm, we first train the base classifier. Second, we train one binary meta-classifier per class label. The training examples for the meta-classifiers are created as follows: We classify each instance in the training set with the base classifier and iterate over the top n predictions. For each prediction, we check whether the predicted class label q is a true class label of the instance. If this is the case, we add the rank and confidence score of the prediction as a positive example to the training set for the meta-classifier for class q. Otherwise, we add it as a negative example. Finally, we train the meta-classifiers for each class on their respective training sets. Algorithm 2.1 illustrates the training phase of our classification scheme.

Algorithm 1 MVS: Training Phase

```
Require: $T_{1..t}$, training instances
Require: $B$, base classifier
Require: $M_{1..l}$, meta-classifiers (one per class label)
  Train $B$ on $T$
  $N_{1..t} \Leftarrow$ set of instances for meta-classifiers (initially empty)
  for $j = 1$ to $t$ do
    $C \Leftarrow$ true class labels of $T_j$
    $P \Leftarrow$ top $n$ predictions of $B$ for $T_j$
    for $l = 1$ to $n$ do
      $q \Leftarrow$ class label of $P_l$
      if $q \in C$ then
        add $P_l$ as positive example to $N_q$
      else
        add $P_l$ as negative example to $N_q$
      end if
    end for
  end for
  for $m = 1$ to $l$ do
    Train $M_m$ on $N_m$
  end for
```

The classification phase of our scheme is straightforward: first, we classify the instance using the base classifier and iterate over the top n predictions. For each prediction, we use the respective meta-classifier to determine whether the prediction is true or false. It should be noted that for some classification algorithms our MVS scheme reduces the overall complexity compared to one-vs-all. Consider for example a Rocchio classifier: When trained as a multi-class classifier, we need to compute the centroid for each class. When Rocchio is used in a one-vs-all setup, we need to compute the centroid for each *negative* class as well. Another advantage of our scheme is that it can be combined with ensemble learning. In a variety of tasks, ensembles of several classifiers have been shown to be more effective (e. g., [1]). The intention is that two or more diverse classifiers (that are assumed to have independent classification errors) are combined so that classification errors by one classifier will be compensated for by the correct predictions of another classifier. One classical ensemble learning scheme is stacking [8]: a meta-learner is trained on the output of two or more base classifiers. The basic version of our scheme can be regarded as stacking with only one base classifier. It is straightforward to extend our scheme to use more than one base classifier: the meta-classifiers simply use the output of more than one base classifier as features.

2.2 Empirical Evaluation

We decided to implement the MVS scheme with the widely used Naive Bayes, Rocchio and kNN classifiers. The Rocchio algorithm has the known disadvantage of becoming inaccurate when classes are not spheres of similar size in vector space, and it does not handle non-spherical classes (e. g. multi-modal classes that consist of more than one cluster) very well. However, Rocchio classification has been shown to work well for text classification when the texts are short and of similar length. Since a human reader is usually able to recognise the topic of a newspaper article just by looking at the headline, we experimented with the categorisation of very short texts. For stopword removal and further dimensionality reduction we used a part-of-speech tagger and selected only verbs, nouns and proper nouns for inclusion in the feature set.

We tested our algorithm on the well known Reuters-21578 collection. We used the well-known *ModApte*-split to separate training and test data. Unlabelled instances were kept. Table 1 shows the results. In preliminary experiments, we used a thresholding approach similar to *SCut* instead of MVS. These settings performed consistently worse and are not presented here. With all classifiers tested, the MVS scheme clearly outperforms the traditional one-vs-all-setup. When comparing the performance of our setup to the results presented in [2], we can conclude that classification of news articles based only on headlines is possible with only a small, acceptable loss in accuracy compared to similar classifiers trained on the full article text. The Rocchio algorithm in the MVS setting trained on the headlines even outperformed the Findsim classifier (a variation of Rocchio) trained on the full text. In general, we observe that Rocchio performs surprisingly well, which we assume is due to the fact that the text are very short and equally long, a situation were Rocchio has been shown to perform well. A stacking-approach as described above where kNN, Naive Bayes and Rocchio have been combined performs best for most classes, however, on some classes, the individual classifiers performed better, strongly affecting macro-averaged F1. We conclude that apparently the meta-classifier tended to overfit and the rules it produced are not optimal. This problem could probably be solved by validating the rules on a hold-out set.

Table 1. Performance results (F1 in percent) of different setups on the top 10 classes of the Reuters-21578 dataset. The first six classifiers were trained on the headlines only; the last three classifiers were trained on the full text and are listed for comparison, these results were reported in [2] (Naive Bayes and Rocchio) and [4] (kNN). Classifiers were (from left to right) N-1/N-2: Naive Bayes, one-vs-all/MVS; R-1/R-2: Rocchio, one-vs-all/MVS; K-1/K-2: kNN, one-vs-all/MVS; S-3: MVS with kNN, Naive Bayes and Rocchio combined; N-x: Naive Bayes, full text, one-vs-all; F-x: Findsim (similar to Rocchio), full text, one-vs-all; K-x: kNN, full text, one-vs-all.

	Headlines							Full text		
	N-1	N-2	R-1	R-2	K-1	K-2	S-3	N-x	F-x	K-x
earn	93	94	73	91	84	80	95	96	93	97
acq	81	84	72	60	65	63	71	88	65	92
money-fx	51	45	46	60	51	59	61	57	47	78
grain	50	61	57	74	68	68	76	79	68	82
crude	55	70	43	73	54	63	73	80	70	86
trade	45	60	25	69	47	56	64	64	65	77
interest	50	56	31	68	60	61	63	65	63	74
ship	38	53	31	52	34	47	57	85	49	79
wheat	37	57	46	63	40	55	58	70	69	77
corn	32	55	13	61	38	47	61	65	48	78
Mavg(10)	53	64	44	67	54	60	68	75	64	82

3 Semi-Automated Tagging

Although text classification is an old and well-researched topic in information retrieval and machine learning it has not been widely used for automatic tagging in Web-2.0 applications yet. An exception is AutoTag [5], a system that uses a k-nearest-neighbour classifier for automated tagging of blog posts. AutoTag [5] uses a search engine to locate similar blog posts. The search query is derived from the text that is to be classified using statistical query rewriting techniques. In the next step, tags from the search results are aggregated and re-ranked using information about the user. Yet, this method of predicting tags for posts has a disadvantage: re-writing the query at classification time is computationally costly. Given that many annotations are plausible and the user is involved in the classification process, it is not necessary that the algorithm predicts the exact set of true annotations. Opposed to the labelling in the Reuters benchmark, it is therefore acceptable that a classifier outputs a ranked list with suggested tags. Considering the high number of classes and the need for an incremental learning algorithm, using vector space classification algorithms such as kNN or Rocchio is a logical choice.

3.1 Experimental Setup

To evaluate our approach, we used a corpus of 116417 questions from the question-and-answer-community web site Lycos iQ that were tagged with 49836 distinct tags. We used the same setup of the Rocchio classifier as described above in section 2. Preliminary experiments showed that using Naive Bayes classification is not viable due to the high number of classes. Also, the Rocchio classifier performed faster that Naive Bayes. For comparison, we included a kNN classifier with $k = 10$ that queries the index for the ten nearest questions (postings) in the database and aggregates the tags from the results. This approach is close to AutoTag [5], but because we perform stopword removal it is not needed to rewrite the query on classification time.

3.2 Empirical Evaluation

Given the nature of the two algorithms, we expect that Rocchio classification will be faster; a factor that is very important in an interactive setting, when users are not willing to accept long response times. We measured the classification time per instance for both approaches on an Intel Core 2 machine with 1.86 GHz and 1 GB RAM. As expected, Rocchio classification was much faster then kNN. The classification time for each instance was 155 ms for kNN and 57 ms for Rocchio.

It is important to note that the tags assigned by users should not be regarded as a gold standard. Tags are not drawn from an ontology, taxonomy or controlled vocabulary, but are free text entered by users and thus prone to spelling mistakes. Also, inexperienced users tend to assign either no tags at all, only very few tags or they tag inconsistently. Given the large number of users, we also expect that users use different synonyms to denote the same concept. Due to these ambiguities and inconsistencies we expect that the accuracy of any automated approach is considerably lower than its true usefulness. In tests kNN only achieved a 26% precision for its top prediction, Rocchio reached 32% precision.

In order to circumvent the problem of noise in the test set, we distributed questionnaires and had test persons check the plausibility of tags suggested by our semi-automated approach. To reduce the workload for the test persons and because it outperformed the kNN-classifier in the automated tests, we decided to test only the Rocchio-style approach. For comparison, we also had the test persons check the precision of the user-assigned tags, since we assumed many nonsensical or inconsistent tags among them. Every test person was given one or two chunks of 100 out of a random sample of 200 questions that were either machine-tagged or hand-tagged. Every question was checked by four persons to average out disagreement about the sensibility of tags.

As expected, we could observe that there was a big disagreement among the test persons and the users who originally tagged the questions as well as between the test persons themselves. As explained above, the total 200 questions that were evaluated were split in two sets of 100 questions, yielding four different questionnaires (two for the original user-assigned tags and two for machine-annotated tags) and each chunk of 100 questions was checked by four persons. Each test person was checking at most two sets of questions. To highlight the huge difference of the several test persons, we report the individual results in the table below. For the human-annotated tags, we evaluated precision, defined as the number of useful tags divided by the total number of assigned tags. For the machine-assigned tags, we also report the fraction of questions with at least one correctly predicted tag.

For all manual tests, we evaluated the algorithms with five suggested tags only. We believe that in a real-world semi-automated setting, we cannot assume that an inexperienced user is willing to look at more than five tags. The questions that were manually tagged had mostly three tags each, some of them only two and very few questions had more than three tags.

Table 2. Evaluation on Lycos iQ dataset. Results are shown for tags *assigned* by the users and for the tags *suggested* by our system.

Test	**TP**	**TP+FP**	**avg. Prec.**
assigned tags	1535	1856	0.83
suggested tags	1866	3360	0.56

Test	**Person 1**	**Person 2**	**Person 3**	**Person 4**
Set 1, assigned tags, prec.	0.89	0.89	0.93	0.96
Set 2, assigned tags, prec.	0.52	0.73	0.73	0.87
Set 1, suggested tags, prec.	0.41	0.52	0.53	0.71
Set 2, suggested tags, prec.	0.51	0.54	0.59	0.65
Set 1, at least one correct	0.84	0.84	0.86	0.87
Set 2, at least one correct	0.87	0.87	0.91	0.91

As expected, different persons disagreed significantly on both the human-annotated and the machine-annotated tags (see table 2). It is interesting to note when looking at the second set of questions, that, although the human annotations on this set were rated worse than those from the first set, the tags suggested by our algorithm were on average rated slightly better. Since we envision a semi-automated scenario with human intervention, we see this as a confirmation that automatically suggested tags can help to improve the quality of tagging.

When looking at macro-averaged precision, it is obvious that a classification system is still not good enough for fully automated tagging. However, it is important to note that even the human-annotated questions were rated far below 100% correct by the test persons. More than half of the suggested tags were rated as useful by the test persons. We believe that this is certainly good enough for a semi-automated scenario, were users are presented a small number of tags to choose from. In absolute numbers, interestingly, the automatic classifier produced more helpful tags than were assigned by users, even compared to the number of all user-assigned tags, not just the ones perceived as helpful by the test persons. We believe that this confirms our hypothesis that users will assign more tags when they are supported by a suggestion system. However, this can only be finally answered with a user study done with a live system.

Finally, the high number of questions where at least one of the predictions by the algorithm was correct underlines our conclusion that semi-automated tagging is good enough to be implemented in a production environment. In almost nine out of ten cases there was at least one helpful tag among the suggestions.

4 Conclusion

In this paper, we have made two contributions: first, we introduced a new mode for adapting single-label classifiers to multi-label classification we called MVS. This scheme has the advantage of being more accurate and at the same time faster than the traditional one-vs-all classification and is easily extensible to using multiple base classifiers in an ensemble. Second, we introduced part-of-speech tagging as a method for stopword removal and showed that multi-value text classification is possible at acceptable accuracy even if the texts are very short. We applied this on the real-world task of tagging for Web 2.0 and have shown that it performs well enough to be used in a semi-automatic setting. In future work, we want to extend our research in various directions. Our experiments with the Reuters dataset left some important questions open. For example, we are currently ignoring relations between classes, an approach that proved successful [3]. Also, more experiments on different datasets and classifiers are needed.

Acknowledgements. The research presented in this paper was partially funded by the German Federal Ministry of Economy and Technology (BMWi) under grant number 01MQ07008. The authors are solely responsible for the contents of this work. We thank our colleagues at Lycos Europe who gave valuable feedback.

References

1. T. G. Dietterich. Ensemble methods in machine learning. In *Proc. of the First Int. Workshop on Multiple Classifier Systems*, 2000.
2. Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *CIKM '98: Proceedings of the seventh international conference on Information and knowledge management*, New York, NY, USA, 1998. ACM.
3. Shantanu Godbole and Sunita Sarawagi. Discriminative methods for multi-labeled classification. In *Proc. of the 8th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD)*, 2004.
4. Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proc. European Conf. on Machine Learning (ECML)*. Springer, 1998.
5. Gilad Mishne. Autotag: a collaborative approach to automated tag assignment for weblog posts. In *Proc. of the 15th Int. World Wide Web Conference*, New York, NY, USA, 2006. ACM Press.
6. Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13, 2007.
7. I. H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools with Java Implementations.* Morgan Kaufmann, San Francisco, 1999.
8. David H. Wolpert. Stacked generalization. *Neural Netw.*, 5(2), 1992.
9. Yiming Yang. A study of thresholding strategies for text categorization. In *Proc. of the 24th Int. ACM SIGIR Conf.*, 2001.