16 Text Classification Experiments

16.1 Introduction

In a classification-based approach such as ours, the employed classification algorithm is of core importance for the system. When used in our information extraction setup (cf. Chap. 10), the performance of the classifier is only one of several factors influencing the results. To get a better impression of the performance of the classifier and to to find out whether it is competitive with other state-of-the-art classifiers, it therefore makes sense to evaluate the classifier on a less complex task where the results will depend primarily on the used classification algorithm.

Test classification is a classical test case for classification algorithms where this is the case. Among the possible application areas of text classification we have chosen *spam filtering* as a particularly interesting test case. Spam filtering is a highly competitive task which has attracted a lot of recent research—if the classifier can compete in this area, we can reasonable expect it to the competitive in other areas as well. Also, spam filtering is a task which is usually modeled and evaluated in an *incremental training* setup (cf. Sec. 3.3 and the test setups described below), since there is a stream of e-mail messages which must be filtered in the order in which they arrive and the classification model should be continuously adapted from the feedback of the user.

We will use this test scenario to check whether our choice of a default classification algorithm (Winnow coupled with a feature combination technique, cf. Chap. 11) appears to be a good one and to find out whether and which feature combination technique we should use: Is the usage of such techniques helpful? If yes, can we afford to use the OSB (Sec. 11.2.2) technique which is faster and generates less features then SBPH (Sec. 11.2.1) or should the latter technique be used to get better results?

Additionally, we will use this test case to optimize parameters for Winnow (the promotion factor α, the demotion factor β, and the threshold thickness) and the combination techniques (the window size N). We will use the optimized parameter values for the information extraction experiments that will follow, based on the assumption that these parameters should be sufficiently stable across different tasks. While we did some informal tests that make this assumption appear reasonable, we did not formally test this, so it is possible that the IE results we will report in the following chapters could be further improved by re-optimizing these parameters—we will leave this as future work, since parameter optimization tests are not among our goals (as stated in Sec. 7.4).

16.2 Text Classification Setup for Spam Filtering

Spam filtering can be viewed as a classic example of a text classification task with a strong practical application. While keyword, fingerprint, whitelist/blacklist, and heuristic–based filters such as SpamAssassin [SpA] have been successfully deployed, these filters have experienced a decrease in accuracy as spammers introduce specific countermeasures. The current best-of-breed anti-spam filters are all probabilistic systems. Most of them are based on Naive Bayes as described by Graham [Gra03] and implemented in *SpamBayes* [SpB]; others such as the *CRM114 Discriminator* can be modeled by a Markov Random Field [Yer04]. Other approaches such as *Maximum Entropy Modeling* [Zha03] lack a property that is important for spam filtering—they are not *incremental*, they cannot adapt their classification model in a single pass over the data.

For our spam filtering experiments, we have tested Winnow in an incremental setup as a statistical, but non-probabilistic alternative. The feature space considered by most current spam filters is limited to individual tokens (unigrams) or bigrams. We overcome this limitation by combining Winnow with one of the feature combination techniques SBPH or OSB.

For these text-classification experiments, we did not perform language-specific preprocessing techniques such as word stemming, stop word removal, or case folding, since other researchers found that such techniques tend to hurt spam-filtering accuracy [Gra03, Zha03]. We did compare three types of mail-specific preprocessing.

- Preprocessing via *mimedecode*, a utility for decoding typical mail encodings (Base64, Quoted-Printable etc.)
- Preprocessing via Jaakko Hyvatti's *normalizemime* [nor] which was specifically developed for the use with spam filters. This program converts the character set to UTF-8, decoding Base64, Quoted-Printable and URL encoding and adding warning tokens in case of encoding errors. It also appends a copy of HTML/XML message bodies with most tags removed, decodes HTML entities and limits the size of attached binary files.
- No preprocessing. Use the raw mails including large blocks of Base64 data in the encoded form.

By default, we used *normalizemime* for the experiments reported in the next section and no preprocessing (just the raw mails) for participation in the *TREC Spam Filtering Challenge* (Sec. 16.4).

16.3 Experimental Results on the SpamAssassin Corpus

16.3.1 Testing Procedure

For evaluating the spam filtering performance we have used a standardized spam/non-spam test corpus from SpamAssassin [SpA]. It consists of 1397 spam messages, 250 hard non-spam and 2500 easy non-spam messages, for a total of 4147 messages. These

4147 messages were "shuffled" into ten different standard sequences; results were averaged over these ten runs. We re-used the corpus and the standard sequences from [Yer04].

Each test run begins with initializing all memory in the learning system to zero. Then the learning system was presented with each member of a standard sequence, in the order specified for that standard sequence, and required to classify the message. After each classification the true class of the message was revealed and the classifier had the possibility to update its prediction model accordingly prior to classifying the next message.[1] The training system then moved on to the next message in the standard sequence. The final 500 messages of each standard sequence were the *test set* used for final accuracy evaluation; we also report results on an extended test set containing the last 1000 messages of each run and on all (4147) messages. Systems were permitted to train on any messages, including those in the test set, *after* classifying them; at no time a system ever had the opportunity to learn on a message before predicting the class of this message. For evaluation we calculated the

$$\textit{error rate } E = \frac{\textit{number of misclassifications}}{\textit{number of all classifications}};$$

occasionally we mention the *accuracy* $A = 1 - E$.

This process was repeated for each of the ten standard sequences. Each complete set of ten standard sequences (41470 messages) required approximately 25–30 minutes of processor time on a 1266 MHz Pentium III for OSB_5.[2] The average number of errors per test run is given in parenthesis.

16.3.2 Parameter Tuning

We used a slightly different setup for tuning the Winnow parameters since it would have been unfair to tune the parameters on the test set. The last 500 messages of each run were reserved as test set for evaluation, while the preceding 1000 messages were used as *development set* for determining the best parameter values.

Among the tested parameter settings, best performance was reached with Winnow using 1.23 as promotion factor, 0.83 as demotion factor, and a threshold thickness of 5%.[3] These parameter values turned out to be best for both OSB and SBPH—the results reported in Tables 16.1 and 16.2 are for OSB.

Promotion	1.35	1.25	1.25	1.23	1.2	1.1
Demotion	0.8	0.8	0.83	0.83	0.83	0.9
Test Set	0.44% (2.2)	0.36% (1.8)	0.44% (2.2)	**0.32% (1.6)**	0.44% (2.2)	0.48% (2.4)
Devel. Set	0.52% (5.2)	0.51% (5.1)	0.52% (5.2)	**0.49% (4.9)**	0.51% (5.1)	0.62% (6.2)
All	**1.26% (52.4)**	1.31% (54.3)	1.33% (55.1)	1.32% (54.7)	1.34% (55.4)	1.50% (62.2)

Table 16.1: Promotion and Demotion Factors

[1] In actual usage training will not be quite as incremental since mail is read in batches.

[2] For SBPH_5 it was about two hours which is as expected since SBPH_5 generates four times as many features as OSB_5.

[3] In either direction, i.e., $\theta^- = 0.95\,\theta$, $\theta^+ = 1.05\,\theta$.

Threshold Thickness	0%	5%	10%
Test Set	0.68% (3.4)	**0.32% (1.6)**	0.44% (2.2)
Development Set	0.88% (8.8)	**0.49% (4.9)**	0.56% (5.6)
All	1.77% (73.5)	**1.32% (54.7)**	1.38% (57.1)

Table 16.2: Threshold Thickness

16.3.3 Feature Store Size and Comparison with SBPH

Table 16.3 compares orthogonal sparse bigrams and SBPH for different sizes of the feature store. OSB reached best results with 600,000 features (with an error rate of 0.32%), while SBPH peaked at 1,600,000 features (with a slightly higher error rate of 0.36%). Further increasing the number of features permitted in the store negatively affects accuracy. This indicates that the LRU pruning mechanism is efficient at discarding irrelevant features that are mostly noise.

	OSB				
Store Size	400000	500000	600000	700000	800000
Last 500	0.36% (1.8)	0.38% (1.9)	**0.32% (1.6)**	0.44% (2.2)	0.44% (2.2)
Last 1000	0.37% (3.7)	0.37% (3.7)	**0.33% (3.3)**	0.37% (3.7)	0.37% (3.7)
All	1.26% (52.3)	1.29% (53.4)	**1.24% (51.4)**	1.26% (52.2)	1.27% (52.5)
	SBPH				
Store Size	1400000	1600000	1800000	2097152 (2^{21})	2400000
Last 500	0.38% (1.9)	**0.36% (1.8)**	0.42% (2.1)	0.44% (2.2)	0.42% (2.1)
Last 1000	0.37% (3.7)	**0.34% (3.4)**	0.38% (3.8)	0.39% (3.9)	0.38% (3.8)
All	1.35% (55.8)	**1.28% (53.1)**	1.30% (54)	1.30% (54)	1.31% (54.2)

Table 16.3: Comparison of SBPH and OSB with Different Feature Storage Sizes

16.3.4 Unigram Inclusion

The inclusion of individual tokens (unigrams) in addition to orthogonal sparse bigrams does not generally increase accuracy, as can be seen in Table 16.4, showing OSB without unigrams peaking at 0.32% error rate, while adding unigrams pushes the error rate up to 0.38%.

	OSB only	**OSB + Unigrams**	
Store Size	600000	600000	750000
Last 500	**0.32% (1.6)**	0.38% (1.9)	0.42% (2.1)
Last 1000	**0.33% (3.3)**	**0.33% (3.3)**	0.36% (3.6)
All	1.24% (51.4)	**1.22% (50.6)**	1.24% (51.4)

Table 16.4: Utility of Single Tokens (Unigrams)

16.3.5 Window Sizes

The results of varying window size as a system parameter are shown in Table 16.5. Again, we note that the optimal combination for the test set uses a window size of five tokens (our default setting, yielding a 0.32% error rate), with both shorter and longer windows producing worse error rates.

Window Size	**Unigrams**	**2 (Bigrams)**	**3**	**4**	**5**	**6**	**7**
Store Size	All (ca.55000)	150000	300000	450000	600000	750000	900000
Last 500	0.46% (2.3)	0.48% (2.4)	0.42% (2.1)	0.44% (2.2)	**0.32% (1.6)**	0.38% (1.9)	0.42% (2.1)
Last 1000	0.50% (5)	0.43% (4.3)	0.39% (3.9)	0.40% (4)	**0.33% (3.3)**	0.38% (3.8)	0.37% (3.7)
All	1.43% (59.2)	1.23% (51.2)	1.24% (51.4)	1.26% (52.2)	1.24% (51.4)	1.28% (53)	**1.22% (50.8)**
Store Size		All (ca.220000)	All (ca.500000)	600000		900000	1050000
Last 500		0.48% (2.4)	0.42% (2.1)	0.42% (2.1)		0.40% (2)	0.46% (2.3)
Last 1000		0.43% (4.3)	0.38% (3.8)	0.38% (3.8)		0.38% (3.8)	0.40% (4)
All		1.24% (51.3)	**1.22% (50.6)**	1.25% (51.8)		1.27% (52.5)	1.25% (51.7)

Table 16.5: Sliding Window Size

This "U" curve is not unexpected on an information-theoretic basis. English text has a typical entropy of around 1–1.5 bits per character and around five characters per word. If we assume that a text contains mainly letters, digits, and some punctuation symbols, most characters can be represented in six bits, yielding a word content of 30 bits. Therefore, at one bit per character, English text becomes uncorrelated at a window length of six words or longer, and features obtained at these window lengths are not significant.

These results also show that using OSB_5 is significantly better then using only single tokens (error rate of 0.46%) or conventional bigrams (0.48%).

16.3.6 Preprocessing

Results with *normalizemime* were generally better than the other two options, reducing the error rate by up to 25% (Table 16.6). Accuracy on raw and *mimedecoded* mails was roughly comparable.

Preprocessing	**none**	**mimedecode**	**normalizemime**
Last 500	0.42% (2.1)	0.46% (2.3)	**0.32% (1.6)**
Last 1000	0.37% (3.7)	0.35% (3.5)	**0.33% (3.3)**
All	1.27% (52.5)	1.26% (52.1)	**1.24% (51.4)**

Table 16.6: Preprocessing

16.3.7 Comparison with CRM114 and Naive Bayes

The results for *CRM114* and Naive Bayes on the last 500 mails are the best results reported in [Yer04] for incremental (single-pass) training. For a fair comparison, these tests were all run using the same tokenization schema as CRM114 on raw mails without preprocessing. The best reported CRM114 weighting model is based on empirically derived weightings and is a rough approximation of a Markov Random Field. This model reduces to a Naive Bayes Model when the window size is set to 1. To avoid the different pruning mechanisms (CRM114 uses a random-discard algorithm) from distorting the comparison, we disabled LRU pruning for Winnow and also reran the CRM114 tests using all features. Our results (Table 16.7) show a reduction in the error rate by 75% compared to Naive Bayes and by more than 50% compared to CRM114.

	Naive Bayes	**CRM114**	**CRM114**	**Winnow+OSB**
Store Size	All	1048577 ($2^{20}+1$)	All	All
Last 500	1.84% (9.2)	1.12% (5.6)	1.16% (5.8)	**0.46% (2.3)**
All	3.44% (142.8)	2.71% (112.5)	2.73% (113.2)	**1.30% (53.9)**

Table 16.7: Comparison with Naive Bayes and CRM114

16.3.8 Speed of Learning

The learning rate for the Winnow classifier combined with the OSB feature generator is shown in Fig. 16.1. Note that the rightmost column shows the incremental error rate on new messages. We can see that the classifier learns very fast—after having classified 1000 messages, Winnow+OSB achieves error rates below 1% on new mails.

There are some additional issues that are of interest for text classification and spam filtering, such as suitable schemas for tokenizing text in such a way that provides specifically interesting features to the classifier. We will not treat these issues here since are of limited interest in the context of this work—discussions and evaluation results can be found in [Sie04b].

16.4 TREC Spam Filtering Challenge

With the filter setup described above, we participated in the 2005 Spam Filtering Task of the renowned *Text REtrieval Conference (TREC)*. The 2005 Spam Filtering Task was to perform a ham (= non-spam) vs. spam classification on several e-mail corpora. The task prescribes the same incremental training trained regimen as used for the experiments reported above: the classifier has to classify each message as it comes in, returning a "spamminess score"[4]; *after* classification, the true class (*spam* or *ham*) of the message is revealed and the classifier can update its prediction model prior to classifying the next message. Different from the test setup used above, the TREC spam corpora are ordered by date and are processed in this order. Thus there is only a single run over each corpus.

Many users of spam filters will consider the cost of misclassifying (and thus losing) a good (ham) message higher than the cost of misclassifying (and thus having to read) a spam message. Because of this, the tasks organizers decided to measure error rates

[4] A real number with higher numbers indicating higher likelihoods that a message is spam; we returned the probability estimated by the classifier for the message being in the *spam* class.

Mails	**Error Rate** (Avg. Errors)		**New Error Rate** (Avg. New Errors)	
25	30.80%	(7.7)	30.80%	(7.7)
50	21.40%	(10.7)	12.00%	(3)
100	14.00%	(14)	6.60%	(3.3)
200	9.75%	(19.5)	5.50%	(5.5)
400	6.38%	(25.5)	3.00%	(6)
600	4.97%	(29.8)	2.15%	(4.3)
800	4.09%	(32.7)	1.45%	(2.9)
1000	3.50%	(35)	1.15%	(2.3)
1200	3.04%	(36.5)	0.75%	(1.5)
1600	2.48%	(39.7)	0.80%	(3.2)
2000	2.12%	(42.3)	0.65%	(2.6)
2400	1.85%	(44.4)	0.53%	(2.1)
2800	1.65%	(46.2)	0.45%	(1.8)
3200	1.51%	(48.2)	0.50%	(2)
3600	1.38%	(49.7)	0.38%	(1.5)
4000	1.28%	(51.1)	0.35%	(1.4)
4147	1.24%	(51.4)	0.20%	(0.3)

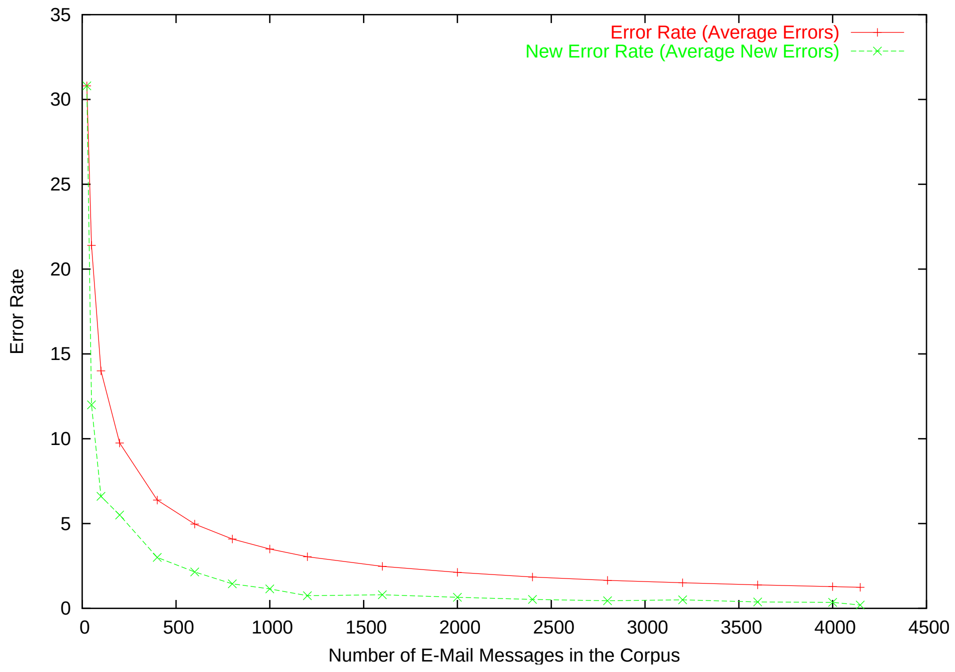

Figure 16.1: Learning Curve for the best setting ($\text{Winnow}_{1.23,0.83,5\%}$, 600,000 features, OSB_5)

for both types of mail separately instead of measuring accuracy: the *ham misclassification rate (hm)* is the fraction of all ham messages classified as spam; the *spam misclassification rate (sm)* is the fraction of all spam messages classified as ham.

The fact that filters report a "spamminess score" s in addition to a binary spam-or-ham judgment makes it possible to introduce an adjustable threshold t and to judge each message as spam iff $s > t$. The threshold t can be adjusted to reflect user preferences regarding misclassification costs. Based on s and t, we can compute hm as a function of sm (the value of hm when t is adjusted to achieve a specific sm), and vice versa. This function can be graphically represented as a *Receiver Operating Characteristic (ROC)* curve (also called recall-fallout curve). The area under this curve measures the general effectiveness of filters over all values of t. Analogously to hm and sm, which measure failure instead of success (smaller values are better), the area *above* the ROC curve ($1 - ROCA$) has been used as general evaluation metric in the TREC task.

As an alternative metric that combines hm and sm into a single figure without using a varying threshold, they task organizers chose the *logistic average misclassification rate*

$$lam = \text{logit}^{-1}\left(\frac{\text{logit}(hm) + \text{logit}(sm)}{2}\right)$$

where $\text{logit}(x) = \log\left(\frac{x}{1-x}\right)$. Thus, lam is the geometric mean of the *odds* ($\frac{p}{1-p}$, instead of the probability p) of ham and spam misclassification. A logit scale is used instead of a linear scale since current spam filters generally reach very good results which are all very near to 0.0 (when measuring failure) or 1.0 (when measuring success).

The task comprises spam/ham classification over four corpora, one of them public (made available to task participants during the task and to the general public afterwards) and the other three private (only available to the task organizers, due to privacy concerns). Together, the four corpora contain 318,482 messages—113,129 spam mails and 205,253 ham mails. In addition to reporting results on the individual corpora, the organizers also published aggregate results combining the results of all corpora "as if they were one" (i.e., the weighted average) to "provide a composite view of the performance on all corpora" [Cor05, Sec. 4.2].

44 filters submitted by 12 different groups participated in the task. The aggregate results of our filter, called *crmSPAM2* by the task organizers[5], are the best all 44 filters regarding lam (0.62%); regarding ROC (0.115%), they are beaten only by the filter submitted by the *Jozef Stefan Institute (IJS)* (in its four configurations, *ijsSPAM1*…*ijsSPAM4*—cf. [Cor05, Tables 5+6]).

The ROC curve for the best classifiers is shown in Fig. 16.2. It can be seen that our classifier (*crmSPAM2*) is somewhat better than the best *IJS* classifier for the medium range of the curve, for a ham misclassification rate from ca. 0.07% to ca.

[5] This name reflects the fact that for participation we had formed a group with the developers of the *CRM114 Discriminator* [CRM] since, after learning of the results reported above (Sec. 16.3), the CRM114 developers decided to integrate our algorithm into their framework and a C re-implementation of our Winnow+OSB classifier is now available as an alternative classifier in CRM114. However, for participation in the TREC task, we used the Java implementation created for this thesis and described in Chap. 11, not the re-implementation in C.

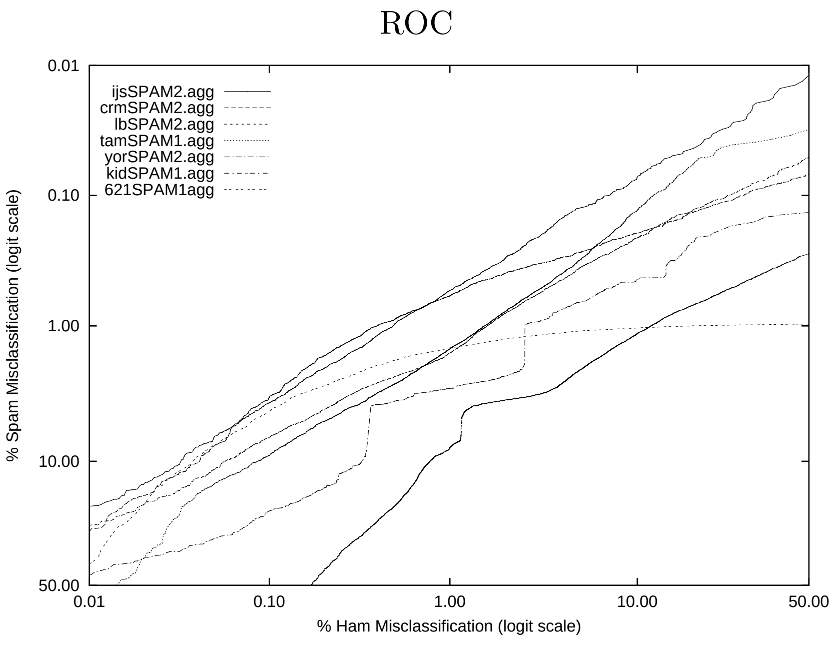

Figure 16.2: ROC curve for the best filters (Source: [Cor05, Fig. 2])

0.7%; while for the outer ranges of the curve it falls off compared to the *IJS* classifier (and to some other classifiers as well). This is probably caused by the fact that the adjustable threshold t is moved for determining the ROC curve, while the Winnow training regimen (described in Sec. 11.1) is based on a fixed balanced threshold.

The training regimen is made somewhat more robust due to our using *thick threshold* training (cf. Sec. 11.1.1), i.e., training not only errors (assuming a balanced internal decision threshold) but also "near misses"—messages where the classification was correct but the scores/probabilities of the two classes are near to each other. This still works with a shifted external decision threshold, but only as long as the external threshold remains safely within the "thick threshold" area. For more extreme external thresholds, this training regimen is no longer adequate—the training process will ignore some of the instances that have been misclassified (when applying the shifted threshold) and thus *should* have been trained.

Adjusting the internal training threshold together with the external threshold should thus remove or reduce the falloff that can be seen at the ends of the curve. However, for the TREC evaluation this was not possible, since all data is based on a single run of the classifier interpreted with varying external thresholds.

More details on the TREC spam filtering experiments and results can found in the track overview paper [Cor05] and in the paper describing our contributions [Ass05].

16.5 Concluding Remarks

The results reached for spam filtering indicate that our chosen classification algorithm, *Winnow* in combination with the newly introduced *OSB* feature combination technique, is highly competitive with the best state-of-the-art classifiers. We have found that *OSB* is a good choice of a feature combination technique and there is no need to use the slower *SBPH* technique instead. Among the tested parameters settings, we found that setting the promotion factor α to 1.23, the demotion factor β to 0.83, and the threshold thickness to 5% and using an OSB window size of 5 yielded best results. As announced above, we will use these parameters settings for the IE experiments in the following chapters.

Spam filtering as the task of separating texts that contain potentially relevant information (text that the user wishes to read = non-spam) from texts that are unwanted and do not contain relevant information (spam) can also be seen as a part of the *text filtering* task which we have identified in Section 3.1 as the first step of a comprehensive IE algorithm. Text classification performance is potentially also relevant in other ways for such a comprehensive IE system: if there are different target schemas (cf. Sec. 9.1) for different text texts, determining the target schema to use for each text could be modeled as a multi-class text classification task among the different text types; the *extraction of implicit information* such as the topic area of a seminar is another task that could potentially be handled by text classification (by classifying among a list of predefined values enumerated by the target schema). We have not tested such scenarios since they are not the focus of our work (which is the classical IE task of identifying and extracting explicitly stated pieces of information, cf. Sec. 7.1.1) and due to the lack of adequate test corpora, but we should keep them in mind so as not to under-estimate the role of text classification.