Combining Winnow and Orthogonal Sparse Bigrams for Incremental Spam Filtering

Christian Siefkes[1], Fidelis Assis[2],
Shalendra Chhabra[3], and William S. Yerazunis[4]

[1] Berlin-Brandenburg Graduate School in Distributed Information Systems[⋆]
Database and Information Systems Group, Freie Universität Berlin
Berlin, Germany
christian@siefkes.net

[2] Empresa Brasileira de Telecomunicações – Embratel
Rio de Janeiro, RJ, Brazil
fidelis@embratel.net.br

[3] Computer Science and Engineering
University of California, Riverside
California, USA
schhabra@cs.ucr.edu

[4] Mitsubishi Electric Research Laboratories
Cambridge, MA, USA
wsy@merl.com

Abstract. Spam filtering is a text categorization task that has attracted significant attention due to the increasingly huge amounts of junk email on the Internet. While current best-practice systems use Naive Bayes filtering and other probabilistic methods, we propose using a statistical, but non-probabilistic classifier based on the *Winnow* algorithm. The feature space considered by most current methods is either limited in expressivity or imposes a large computational cost. We introduce *orthogonal sparse bigrams (OSB)* as a feature combination technique that overcomes both these weaknesses. By combining Winnow and OSB with refined preprocessing and tokenization techniques we are able to reach an accuracy of 99.68% on a difficult test corpus, compared to 98.88% previously reported by the *CRM114* classifier on the same test corpus.

Keywords: Classification, Text Classification, Spam Filtering, Email, Incremental Learning, Online Learning, Feature Generation, Feature Representation, Winnow, Bigrams, Orthogonal Sparse Bigrams.

1 Introduction

Spam filtering can be viewed as a classic example of a text categorization task with a strong practical application. While keyword, fingerprint, whitelist/blacklist, and heuristic–based filters such as SpamAssassin [11] have been successfully

[⋆] The work of this author is supported by the German Research Society (DFG grant no. GRK 316).

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 [6] and implemented in *SpamBayes* [12]; others such as the *CRM114 Discriminator* can be modeled by a Markov Random Field [15]. Other approaches such as *Maximum Entropy Modeling* [16] 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.

As a statistical, but non-probabilistic alternative we examine the incremental *Winnow* algorithm. Our experiments show that Winnow reduces the error rate by 75% compared to Naive Bayes and by more than 50% compared to *CRM114*.

The feature space considered by most current methods is limited to individual tokens (unigrams) or bigrams. The *sparse binary polynomial hashing (SBPH)* technique (cf. Sec. 4.1) introduced by *CRM114* is more expressive but imposes a large runtime and memory overhead. We propose *orthogonal sparse bigrams (OSB)* as an alternative that retains the expressivity of SBPH, but avoids most of the cost. Experimentally OSB leads to equal or slightly better filtering than SBPH. We also analyze the preprocessing and tokenization steps and find that further improvements are possible here.

In the next section we present the Winnow algorithm. The following two sections are dedicated to feature generation and combination. In Section 5 we detail our experimental results. Finally we discuss related methods and future work.

2 The Winnow Classification Algorithm

The Winnow algorithm introduced by [7] is a statistical, but not a probabilistic algorithm, i.e. it does not directly calculate probabilities for classes. Instead it calculates a *score* for each class[1].

Our variant of Winnow is suitable for both binary (two-class) and multi-class (three or more classes) classification. It keeps an n-dimensional weight vector $w^c = (w_1^c, w_2^c, \ldots, w_n^c)$ for each class c, where w_i^c is the weight of the ith feature. The algorithm returns 1 for a class iff the summed weights of all active features (called the score Ω) surpass a predefined threshold θ:

$$\Omega = \sum_{j=1}^{n_a} w_j^c > \theta.$$

Otherwise ($\Omega \leq \theta$) the algorithm returns 0. $n_a \leq n$ is the number of active (present) features in the instance to classify.

The goal of the algorithm is to learn a linear separator over the feature space that returns 1 for the true class of each instance and 0 for all other classes on this instance. The initial weight of each feature is 1.0. The weights of a class

[1] There are ways to convert the scores calculated by Winnow into confidence estimates, but these are not discussed here since they are not of direct relevance for this paper.

are updated whenever the value returned for this class is wrong. If 0 is returned instead of 1, the weights of all active features are increased by multiplying them with a *promotion factor* α, $\alpha > 1$: $w_j^c \leftarrow \alpha \times w_j^c$. If 1 is returned instead of 0, the active weights are multiplied with a *demotion factor* β, $0 < \beta < 1$: $w_j^c \leftarrow \beta \times w_j^c$.

In text classification, the number of features depends on the length of the text, so it varies enormously from instance to instance. Thus instead of using a fixed threshold we set the threshold to the number n_a of features that are active in the given instance: $\theta = n_a$. Thus initial scores are equal to θ since the initial weight of each feature is 1.0.

In multi-label classification, where an instance can belong to several classes at once, the algorithm would predict all classes whose score is higher than the threshold. But for the task at hand, there is exactly one correct class for each instance, thus we employ a *winner-takes-all* approach where the class with the highest score is predicted.

This means that there are situations where the algorithm will be trained even though it did not make a mistake. This happens whenever the scores of both classes[2] are at the same side of the threshold and the score of the true class is higher than the other one – in this case the prediction of Winnow will be correct but it will still promote/demote the weights of the class that was at the wrong side of the threshold.

The complexity of processing an instance depends only on the number of active features n_a, not on the number of all features n_t. Similar to *SNoW* [1], a sparse architecture is used where features are allocated whenever the need to promote/demote them arises for the first time. In sparse Winnow, the number of instances required to learn a linear separator (if exists) depends linearly on the number of relevant features n_r and only logarithmically on the number of active features, i.e. it scales with $O(n_r \log n_a)$ (cf. [8, Sec. 2]).

Winnow is a non-parametric approach; it does not assume a particular probabilistic model underlying the training data. Winnow is a linear separator in the Perceptron sense, but by providing a feature space that itself allows conjunction and disjunction, complex non-linear features may be recognized by the composite feature-extractor + Winnow system.

2.1 Thick Threshold

In our implementation of Winnow, we use a *thick threshold* for learning (cf. [4, Sec. 4.2]). Training instances are re-trained even if the classification was correct if the determined score was near the threshold. Two additional thresholds θ^+ and θ^- with $\theta^- < \theta < \theta^+$ are defined and each instance whose score falls in the range $[\theta^-, \theta^+]$ is considered a mistake. In this way, a large margin classifier will be trained that is more robust when classifying borderline instances.

2.2 Feature Pruning

The feature combination methods discussed in Section 4 generate enormous numbers of features. To keep the feature space tractable, features are stored in an

[2] Resp. two or more classes in other tasks involving more than two classes.

LRU (least recently used) cache. The feature store is limited to a configurable number of elements; whenever it is full, the least recently seen feature is deleted. When a deleted feature is encountered again, it will be considered as a new feature whose weights are still at their default values.

3 Feature Generation

3.1 Preprocessing

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 [6, 16]. We did compare three types of email-specific preprocessing.

– Preprocessing via *mimedecode*, a utility for decoding typical mail encodings (Base64, Quoted-Printable etc.)
– Preprocessing via Jaakko Hyvatti's *normalizemime* [9]. This program converts the character set to UTF-8, decoding Base64, Quoted-Printable and URL encoding and adding warn 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 mail including large blocks of Base64 data in the encoded form.

Except for the comparison of these alternatives, all experiments were performed on *normalizemime*-preprocessed mails.

3.2 Tokenization

Tokenization is the first stage in the classification pipeline; it involves breaking the text stream into tokens ("words"), usually by means of a regular expression. We tested four different tokenization schemas:

P (Plain): Tokens contain any sequences of printable characters; they are separated by non-printable characters (whitespace and control characters).
C (CRM114): The current default pattern of *CRM114* – tokens start with a printable character; followed by any number of alphanumeric characters + dashes, dots, commas and colons; optionally ended by a printable character.
S (Simplified): A modification of the CRM114 pattern that excludes dots, commas and colons from the middle of the pattern. With this pattern, domain names and mail addresses will be split at dots, so the classifier can recognize a domain even if subdomains vary.
X (XML/HTML+header-aware): A modification of the **S** schema that allows matching typical XML/HTML markup[3], mail headers (terminated by

[3] Start/end/empty tags: `<tag>` `</tag>` `<br/>`; Doctype declarations: `<!DOCTYPE`; processing instructions: `<?xml-stylesheet`; entity + character references: `&mdash;`; attributes terminated by "="; attribute values surrounded by quotes.

Table 1. Tokenization Patterns

Name	Regular Expression
P	`[^\p{Z}\p{C}]+`
C	`[^\p{Z}\p{C}][-.,:\p{L}\p{M}\p{N}]*[^\p{Z}\p{C}]?`
S	`[^\p{Z}\p{C}][-\p{L}\p{M}\p{N}]*[^\p{Z}\p{C}]?`
X	`[^\p{Z}\p{C}][/!?#]?[-\p{L}\p{M}\p{N}]*(?:["'=;]|/?>|:/*)?`

":"), and protocols such as "http://" in a token. Punctuation marks such as "." and "," are not allowed at the end of tokens, so normal words will be recognized no matter where in a sentence they occur without being "contaminated" by trailing punctuation.

The **X** schema was used for all tests unless explicitly stated otherwise. The actual tokenization schemas are defined as the regular expressions given in Table 1. These patterns use Unicode categories – `[^\p{Z}\p{C}]` means everything except whitespace and control chars (POSIX class `[:graph:]`); `\p{L}\p{M}\p{N}` collectively match all alphanumerical characters (`[:alnum:]` in POSIX).

4 Feature Combination

4.1 Sparse Binary Polynomial Hashing

Sparse binary polynomial hashing (SBPH) is a feature combination technique introduced by the *CRM114 Discriminator* [3, 14]. SBPH slides a window of length N over the tokenized text. For each window position, all of the possible in-order combinations of the N tokens are generated; those combinations that contain at least the newest element of the window are retained. For a window of length N, this generates 2^{N-1} features. Each of these joint features can be mapped to one of the odd binary numbers from 1 to $2^N - 1$ where original features at "1" positions are visible while original features at "0" positions are hidden and marked as skipped.

It should be noted that the features generated by SBPH are not linearly independent and that even a compact representation of the feature stream generated by SBPH may be significantly longer than the original text.

4.2 Orthogonal Sparse Bigrams

Since the expressivity of SBPH is sufficient for many applications, we now consider if it is possible to use a smaller feature set and thereby increase speed and decrease memory requirements. For this, we consider only word pairs containing a common word inside the window, and requiring the newest member of the window to be one of the two words in the pair. The idea behind this approach is to gain speed by working only with an *orthogonal* feature set inside the window, rather that the prolific and probably redundant features generated by SBPH.

Instead of all odd numbers, only those with two bits “1” in their binary representations are used: $2^n + 1$, for $n = 1$ to $N - 1$. With this restriction, only $N - 1$ combinations with exactly two words are produced. We call them *orthogonal sparse bigrams (OSB)* – “sparse” because most combinations have skipped words; only the first one is a conventional bigram.

With a sequence of five words, $w_1, \ldots, w_5$, OSB produces four combined features:

$$\begin{array}{rrrrr} & & & w4 & w5 \\ & & w3 & <\!skip\!> & w5 \\ & w2 & <\!skip\!> & <\!skip\!> & w5 \\ w1 & <\!skip\!> & <\!skip\!> & <\!skip\!> & w5 \end{array}$$

Because of the reduced number of combined features, $N - 1$ in OSB versus 2^{N-1} in SBPH, text classification with OSB can be considerably faster than with SBPH. Table 2 shows an example of the features generated by SBPH and OSB side by side.

Table 2. Features Generated by SBPH and OSB

Number		**SBPH**					**OSB**				
1	(1)					today?					
3	(11)				lucky	today?				lucky	today?
5	(101)			feel	<*skip*>	today?			feel	<*skip*>	today?
7	(111)			feel	lucky	today?					
9	(1001)		you	<*skip*>	<*skip*>	today?		you	<*skip*>	<*skip*>	today?
11	(1011)		you	<*skip*>	lucky	today?					
13	(1101)		you	feel	<*skip*>	today?					
15	(1111)		you	feel	lucky	today?					
17	(10001)	Do	<*skip*>	<*skip*>	<*skip*>	today?	Do	<*skip*>	<*skip*>	<*skip*>	today?
19	(10011)	Do	<*skip*>	<*skip*>	lucky	today?					
21	(10101)	Do	<*skip*>	feel	<*skip*>	today?					
23	(10111)	Do	<*skip*>	feel	lucky	today?					
25	(11001)	Do	you	<*skip*>	<*skip*>	today?					
27	(11011)	Do	you	<*skip*>	lucky	today?					
29	(11101)	Do	you	feel	<*skip*>	today?					
31	(11111)	Do	you	feel	lucky	today?					

Note that the *orthogonal sparse bigrams* form an almost complete basis set – by “ORing” features in the OSB set, any feature in the SBPH feature set can be obtained, except for the unigram (the single-word feature). However, there is no such redundancy in the OSB feature set; it is not possible to obtain any OSB feature by adding, ORing, or subtracting any other pairs of other OSB features; all of the OSB features are unique and not redundant.

Since the first term, unigram w_5, cannot be obtained by ORing OSB features it seems reasonable to add it as an extra feature. However the experiments reported in Section 5.4 show that adding unigrams does *not* increase accuracy; in fact, it sometimes decreased accuracy.

5 Experimental Results

5.1 Testing Procedure

In order to test our multiple hypotheses, we used a standardized spam/nonspam test corpus from SpamAssassin [11]. This test corpus is extraordinarily difficult to classify, even for humans. It consists of 1397 spam messages, 250 hard nonspams, and 2500 easy nonspams, 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 [15].

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[4]. 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 *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[5]. The average number of errors per test run is given in parenthesis.

5.2 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. The **S** tokenization was used for the tests in the section.

Best performance was found with Winnow using 1.23 as promotion factor, 0.83 as demotion factor, and a threshold thickness of 5%[6]. These parameter values turned out to be best for both OSB and SBPH – the results reported in Tables 3 and 4 are for OSB.

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

[5] For SBPH-5 it was about two hours which it not surprising since SBPH-5 generates four times as many features as OSB-5.

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

Table 3. Promotion and Demotion Factors

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 4. Threshold Thickness

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 5. Comparison of SBPH and OSB with Different Feature Storage Sizes

	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)

5.3 Feature Store Size and Comparison with SBPH

Table 5 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.

5.4 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 6, showing OSB without unigrams peaking at 0.32% error rate, while adding unigrams pushes the error rate up to 0.38%.

5.5 Window Sizes

The results of varying window size as a system parameter are shown in Table 7. Again, we note that the optimal combination for the test set uses a window size

Table 6. Utility of Single Tokens (Unigrams)

	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 7. Sliding Window Size

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)

of five tokens (our default setting, yielding a 0.32% error rate), with both shorter and longer windows producing worse error rates.

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%).

5.6 Preprocessing and Tokenization

Results with *normalizemime* were generally better than the other two options, reducing the error rate by up to 25% (Table 8). Accuracy on raw and *mime-decoded* mails was roughly comparable.

Table 8. Preprocessing

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)**

The **S** tokenization schema initially learns more slowly (the overall error rate is somewhat higher) but is finally just as good as the **X** schema (Table 9). **P** and **C** both result in lower accuracy, even though they initially learn quickly.

Table 9. Tokenization Schemas

Schema	**X**	**S**	**C**	**P**
Last 500	**0.32% (1.6)**	**0.32% (1.6)**	0.44% (2.2)	0.42% (2.1)
Last 1000	**0.33% (3.3)**	**0.33% (3.3)**	0.39% (3.9)	0.38% (3.8)
All	1.24% (51.4)	1.32% (54.7)	1.28% (52.9)	**1.23% (51.1)**

Table 10. Comparison With Naive Bayes and 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)**

5.7 Comparison with CRM114 and Naive Bayes

The results for *CRM114* and Naive Bayes on the last 500 mails are the best results reported in [15] for incremental (single-pass) training. For a fair comparison, these tests were all run using the **C** tokenization schema 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 (Table 10).

5.8 Speed of Learning

The learning rate for the Winnow classifier combined with the OSB feature generator is shown in Fig. 1. Note that the rightmost column shows the incremental error rate on new messages. After having classified 1000 messages, Winnow+OSB achieves error rates below 1% on new mails.

6 Related Work

Cohen and Singer [2] use a Winnow-like multiplicative weight update algorithm called “sleeping experts” with a feature combination technique called “sparse phrases” which seems to be essentially equivalent to SBPH[7]. Bigrams and n-grams are a classical technique; SBPH has been introduced in [14] and “sparse phrases” in [2]. We propose orthogonal sparse bigrams as a minimalistic alternative that is new, to the best of our knowledge.

An LRU mechanism for feature set pruning has been employed by the first author in [10]. We suppose that others have done the same since the idea seems to suggest itself; but currently we are not aware of such usage.

[7] Thanks to an anonymous reviewer for pointing us to this work.

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)

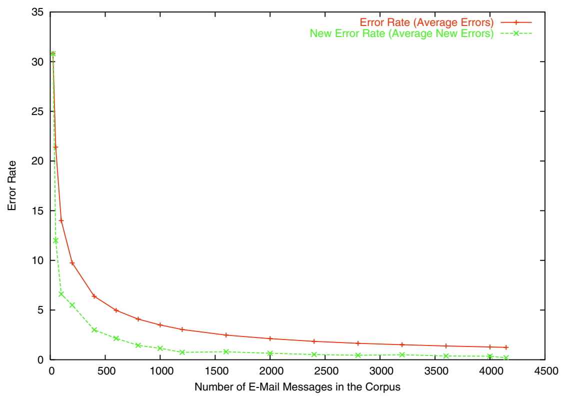

Fig. 1. Learning Curve for the best setting (Winnow$_{1.23,0.83,5\%}$ with 1,600,000 features, OSB-5, **X** tokenization)

7 Conclusion and Future Work

We have introduced *orthogonal sparse bigrams (OSB)* as a new feature combination technique for text classification that combines a high expressivity with relatively low computational load. By combining OSB with the *Winnow* algorithm we more than halved the error rate compared to a state-of-the-art spam filter, while still retaining the property of *incrementality*. By refining the preprocessing and tokenization steps we were able to further reduce the error rate by 30% [8].

In this study we have measured the accuracy without taking the different costs of misclassifications into account (it can be tolerated to let a few spam mails through, but it is bad to classify a regular email as spam). This could be addressed by using a cost metric as discussed in [5]. Winnow could be biased in favor of classifying a borderline mail as nonspam by multiplying the spam score by a factor < 1 (e.g. 99%) when classifying.

Currently our Winnow implementation supports only binary features; how often a feature (sparse bigram) appears in a text is not taken into account. We plan to address this by introducing a *strength* for each feature (cf. [4, Sec. 4.3]).

Also of interest is the difference in performance between the LRU (least-recently-used) pruning algorithm used here and the random-discard algorithm used in *CRM114* [15]. When the random-discard algorithm in *CRM114* triggered, it almost always resulted in a decrease in accuracy; here we found that an LRU algorithm could act to provide an *increase* in accuracy. Analysis and determination of the magnitude of this effect will be a concern in future work.

Acknowledgments

We thank the anonymous reviewers for their helpful comments and suggestions.

[8] Our algorithm is freely available as part of the *TiEs* system [13].

References

1. A. J. Carlson, C. M. Cumby, N. D. Rizzolo, J. L. Rosen, and D. Roth. SNoW user manual. Version: January, 2004. Technical report, UIUC, 2004.
2. W. W. Cohen and Y. Singer. Context-sensitive learning methods for text categorization. *ACM Transactions on Information Systems*, 17(2):141–173, 1999.
3. CRM114: The controllable regex mutilator. http://crm114.sourceforge.net/.
4. I. Dagan, Y. Karov, and D. Roth. Mistake-driven learning in text categorization. In *EMNLP-97*, 1997.
5. J. M. Gómez Hidalgo, E. Puertas Sanz, and M. J. Maña López. Evaluating cost-sensitive unsolicited bulk email categorization. In *JADT-02*, Madrid, ES, 2002.
6. P. Graham. Better Bayesian filtering. In *MIT Spam Conference*, 2003.
7. N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.
8. M. Munoz, V. Punyakanok, D. Roth, and D. Zimak. A learning approach to shallow parsing. Technical Report UIUCDCS-R-99-2087, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1999.
9. normalizemime v2004-02-04. http://hyvatti.iki.fi/ jaakko/spam/.
10. C. Siefkes. A toolkit for caching and prefetching in the context of Web application platforms. Diplomarbeit, TU Berlin, 2002.
11. SpamAssassin. http://www.spamassassin.org/.
12. SpamBayes. http://spambayes.sourceforge.net/.
13. Trainable Incremental Extraction System. http://www.inf.fu-berlin.de/inst/ag-db/software/ties/.
14. W. S. Yerazunis. Sparse binary polynomial hashing and the CRM114 discriminator. In *2003 Spam Conference*, Cambridge, MA, 2003. MIT.
15. W. S. Yerazunis. The spam-filtering accuracy plateau at 99.9% accuracy and how to get past it. In *2004 Spam Conference*, Cambridge, MA, 2004. MIT.
16. L. Zhang and T. Yao. Filtering junk mail with a maximum entropy model. In *20th International Conference on Computer Processing of Oriental Languages*, 2003.