Robust Detection of Comment Spam Using Entropy Rate

Alex Kantchelian
UC Berkeley
akant@eecs.berkeley.edu

Justin Ma
UC Berkeley
jtma@eecs.berkeley.edu

Ling Huang
Intel Labs
ling.huang@intel.com

Sadia Afroz
Drexel University, Philadelphia
sadia.afroz@drexel.edu

Anthony D. Joseph
UC Berkeley
adj@eecs.berkeley.edu

J. D. Tygar
UC Berkeley
tygar@eecs.berkeley.edu

ABSTRACT

In this work, we design a method for blog comment spam detection using the assumption that spam is any kind of uninformative content. To measure the "informativeness" of a set of blog comments, we construct a language and tokenization independent metric which we call *content complexity*, providing a normalized answer to the informal question "how much information does this text contain?" We leverage this metric to create a small set of features well-adjusted to comment spam detection by computing the content complexity over groupings of messages sharing the same author, the same sender IP, the same included links, etc.

We evaluate our method against an exact set of tens of millions of comments collected over a four months period and containing a variety of websites, including blogs and news sites. The data was provided to us with an initial spam labeling from an industry competitive source. Nevertheless the initial spam labeling had unknown performance characteristics. To train a logistic regression on this dataset using our features, we derive a simple mislabeling tolerant logistic regression algorithm based on expectation-maximization, which we show generally outperforms the plain version in precision-recall space.

By using a parsimonious hand-labeling strategy, we show that our method can operate at an arbitrary high precision level, and that it significantly dominates, both in terms of precision and recall, the original labeling, despite being trained on it alone.

The content complexity metric, the use of a noise-tolerant logistic regression and the evaluation methodology are thus the three central contributions with this work.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - spam; K.4.2 [**Computers and Society**]: Social Issues—Abuse and crime involving computers

AISec'12, October 19, 2012, Raleigh, North Carolina, USA.

Keywords

Spam filtering, Comment spam, Content complexity, Noisy label, Logistic regression

1. INTRODUCTION

Online social media have become indispensable, and a large part of their success is that they are platforms for hosting user-generated content. An important example of how users contribute value to a social media site is the inclusion of comment threads in online articles of various kinds (news, personal blogs, etc). Through comments, users have an opportunity to share their insights with one another. However, the medium presents a unique opportunity for abuse by criminals and other miscreants. Abusers use comment threads to post content containing links to spam and malware sites, as well as content that itself is considered as spam by users. If left unaddressed, abuse reduces the value of a social media site by reducing the proportion of legitimate comments, thereby leaving users less satisfied with their experience.

Many approaches were proposed for detecting email spam [16, 19, 22, 23]. However, most of them are not directly applicable to detecting spam in social media. Spam in social media is different from email spam in several ways. The majority of spam email messages are generated by dumb botnets using certain predefined templates [15]. Large volumes of them contain similar content, format and target URLs [24], and display strong temporal regularity and correlation in their sending time [15]. These properties make it relative easy to develop effective approaches to filter out email spam. Unlike email spam messages, social media spam messages, for example, blog comment spam messages, are usually short and carefully crafted by humans, for which even human experts have hard times to differentiate from legitimate ones. We also observe little temporal regularity in the posting of blog comment spam. These differences require us to develop different detectors to filter out blog comment spam. (Although we use the term "blog comment spam," our test set also included items such as newspaper articles.)

Social media spam also takes the advantage of the open nature of the blog comment space. Anonymous communication is allowed in most social media which is not possible in case of email. A single spam message can potentially reach as many viewers as users of the social media. These spam messages usually target search engines to increase the pagerank of the advertised page as well as users.

Because of the scale and complexity of the data involved in detecting social media spam, approaches based on machine learning (ML) offer promising solutions since they scale beyond what a human expert can achieve. However, practitioners who are interested in applying ML to filtering spam face three fundamental

challenges: First, developing features that provide a strong signal for the classifier. Second, constructing an algorithmic approach that can make effective use of those features. Third (often overlooked), evaluation. To be useful, the evaluation must both measure meaningful characteristics and be conducted in a setting as realistic as possible. This means using a sample set which accurately describes the real data. Given a limited amount of hand-labeling resources, this is usually a hard task.

Our approach. While it is possible to define blog comment spam as *any kind of undesirable content* in the fashion of personal email spam, such a single-user centric view is problematic in the context of an open medium where everybody is both entitled to contribute to the media and to access it. In particular, it is possible to find instances of comments where two legitimate users might disagree on whether the message is acceptable, as we show in the evaluation section. Moreover, the context of the surrounding website plays a crucial role in evaluating the undesirability of the comment [10].

As the basis for our approach, we have decided to define spam as content that is uninformative in the information-theoretic sense. Intuitively, if the comments generated by a particular user are highly redundant, then there is a high likelihood that these messages will appear as undesirable to the other users of the social media. The *content complexity* metric we develop quantifies this redundancy in an uniform manner for variable length texts. Once constructed, we can almost immediately obtain meaningful features for blog comment spam detection using this metric.

In addition to the content complexity features, we introduce a latent variable model that can tolerate noisy labels. For practitioners who want to use machine learning methods, acquiring enough labeled training data is one of the biggest challenges. Sometimes they may gain access to feeds or services that provide machine-labeled data that can be noisy (i.e., with the ground-truth label for the data being different from the label provided). In an adversarial context, this noise-tolerance is important because it may make our approach robust to mislabeled outliers deliberately introduced. As such, we adapt our model training to be tolerant of noisy labels. We use a latent variable model to handle noisy labels and enhance the detection accuracy of the content complexity features.

Our contributions. Our approach provides contributions on three fronts:

1. We introduce *content complexity* features for characterizing the IP addresses, usernames, and embedded URL hostnames associated with each comment;

2. We adopt a latent variable model for training a classifier with complexity features (with non-linear expansion) that can *tolerate noisy labels* in our data sets;

3. We conduct a rigorous evaluation of our method, leading to semi-normalized precision-recall curves which we believe are more telling than both receiver operating characteristic curves or their associated single-dimension area under the curve metric.

Organization. The rest of this paper is organized as follows. Section 2 introduces the content complexity metric and explains the construction of our feature vectors for comment spam detection. Section 3 describes the latent variable model we use for classification and tolerating noisy labels. Section 4 describes the data set and the methods we use for the evaluations in Section 5. We finish the paper with a discussion of related work in Section 6 and discuss future work in Section 7.

2. CONTENT COMPLEXITY

We now turn to the construction of the content complexity metric upon which we build features for blog comment spam detection.

Intuitively, content complexity describes the amount of redundancy in the content associated with a string. While remaining language-agnostic, it is normalized both for natural language and string length. In everything that follows and without loss of generality, $x \in \{0;1\}^*$ is a binary string and $|x|$ designates its length, C is a lossless data compression algorithm.

The basis for the content complexity metric is the compression ratio $|C(x)|/|x|$. However, we expect the compression ratio of a string to be sensitive to the original string length (e.g., we expect the complexity of a long essay to be lower than the complexity of a short paragraph). Because we want to be able to compare complexity values between strings of different lengths, we introduce a normalization factor $h(n)$, a parametrized function which models the expected compression ratio of strings of length n. The addition of $h(n)$ allows the content complexity of a string to be calculated independently of its length, and we discuss how we fit the parameters of h in Section 2.2.

2.1 Modeling the Compression Ratio

Let $r_C(x) = |C(x)|/|x|$ be the compression ratio (or the complexity rate) of string x under compressor C. Let $x_{1:n}$ denote the first n bytes subsequence of x. For any well behaved x and compression algorithm C, we use the following function $h(n)$ to approximate $r_C(x_{1:n})$:

$$h(n) = \alpha + A \log n / n^{\gamma} + B/n, \qquad (1)$$
$$\alpha, A, B > 0, \quad 0 < \gamma < 1,$$

where α, A, B, γ are constants which depend on the probability source emitting x alone.

The term B/n represents the fixed-size overhead of off-the-shelf compression algorithms, and is an extension to Schurmann's model to help model the expected compression ratio for small-to-medium values of n [18]. For example, compressing a zero-length string with Lempel-Ziv-Markov chain algorithm (LZMA) [13] produces a 15 byte output. The term is asymptotically negligible for larger values of n.

The first two terms $\alpha + A \log n/n^{\gamma}$ describe a power-law convergence to a fixed value α which can be interpreted as an upper bound on the entropy rate of x.

Finally, while it is possible to give a precise mathematical statement for what we mean by a "well behaved" sequence by requiring stationarity and ergodicity of the stochastic process, we consider in this work general sequences from natural language for which such strong assumptions seem unwarranted. Furthermore, the model is essentially a postulate which is only backed up *a posteriori* by the following experimental evidence.

2.2 Natural Language Complexity Rates

For the rest of this work, we fix the compression function to be LZMA[1]. For each of the six languages presented in Figure 1, we randomly select a dozen plain text UTF-8 e-books from Project Gutenberg. We minimally preprocess each file by discarding its header and footer data which contain various copyright information in English, irrespective of the actual document language. Then, for each text, we compute the sequence of compression ratios on initial subsequences of increasing size and plot the resulting graph.

[1]One reason for this choice is that LZMA can be set up to use a very large compression block size - or equivalently a very large dictionary size - which makes it able to capture long-range dependencies in its input.

We finally superimpose the predicted compression ratio by the above model, where the four parameters have been obtained by a standard Levenberg [8] Marquardt [9] optimization on the non-linear least squares formulation. The initial parameters values are $(\alpha, A, B, \gamma) = (0, 1, 1, 0.5)$, but the final result is independent of these values on a large domain. We fit the model using the data of all six languages so that we obtain a single set of parameters.

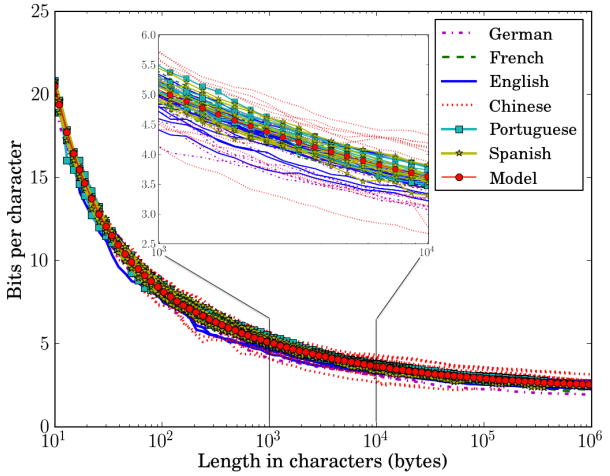


Figure 1: LZMA compression ratios $|C(x)|/|x|$ for prefixes of ebooks in 6 languages, with a single fitted model ($R^2 = 0.995$). Each line represents one ebook. Notice the lower compressibility of Chinese.

The optimal parameters derived from our dataset are:

$$(\alpha, A, B, \gamma) = (2.23, 7.13, 120, 0.419) \quad (2)$$

Notice that the extrapolated LZMA entropy rate on infinite length sequences is $\alpha = 2.23$ bits per byte. This is compatible with although much higher than Shannon's experiments [20] suggesting an average entropy rate of 1.3 bits per character for English texts. Remarkably, LZMA's stable behavior is well captured by our simple model. While European languages are particularly well described, the Asian language samples of our dataset exhibit a much higher variance.

We can finally define the content complexity metric of a string x as $Q(x) = |C(x)|/|x| - h(|x|)$, where $h(n)$ is defined in Eqn (1), and its parameters are previously computed by Eqn (2). The content complexity $Q(x)$ is a real number. Intuitively, it represents the intrinsic "informativeness" of the string x independent of its length. A low value of $Q(x)$ means that x contains very little new information and has a lot of redundancy. A high value of $Q(x)$ means that x contains a lot of new information and has little redundancy.[2]

2.3 From Complexity to Detection Features

One important question to answer is whether the natural language as encountered in blog comments will be as well described by the same model h as for ebooks. Figure 2 gives evidence of a positive answer. Each point on this plot represents a username for which all the contributions are concatenated together in a single string and the compression ratio is subsequently computed. The predicted natural language complexity h is represented in solid black. The model curve still describes fairly accurately the bulk of the distribution, except for the fact that the entropy rate of blog comments seems slightly lower than the one of ebooks. More interestingly, we observe a relatively clear separation between ham and spam on this graph.

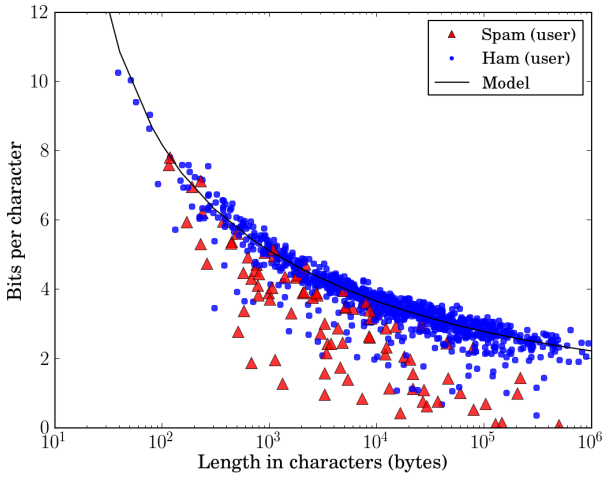


Figure 2: A random sampling of a thousand usernames with two associated comments or more. For each user, all her comments are concatenated and the compression ratio is computed. The predicted compression ratio h learned on the ebooks is indicated in solid black. The labels are propagated from the comments to the usernames by a simple zero-tolerance rule: a user is labeled as spam if and only if one of her comments is labeled as spam. Notice the occurring separation between ham and spam.

The steps for generating our features are as follow. First, we minimally normalize the messages by removing periodic runs of characters, for a period of at most 4 chars. By convention, we only keep the first two runs. For example, `ahahahah` becomes `ahah`, `ooooh` becomes `ooh` but `ahAHaHaHAHAh` remains unchanged. Such periodic runs artificially lower the content complexity metric and are a non negligible source of false positives when left unaddressed.

Second, we choose an aggregation field between messages. In this work, we form the following four groups:

1. messages which share the same username,
2. messages which share the same host name in any linked URL,
3. messages which are posted to the same permalink,
4. messages which come from the same IP address, within a given time period.

The reason why we aggregate on multiple dimensions and not just on a per username basis is that a consequent portion of the comments of our dataset are anonymous, and that a spammer can easily generate a new user account for every new comment, leading to singleton groupings. This evasion strategy is made more difficult by aggregating and scoring across IP and including URL host names.

For the URL extraction, we use a relatively naive regular expression which matches top-level domains. To avoid IP aliasing, the

[2]Note that our definition of "informative" does not exactly correspond to the colloquial use of the term. Nonetheless, as we discuss below, our definition is highly effective at identifying commercial and other spam.

Figure 3: Graphical model for our learning approach

aggregation by IP is parametrized by a timing parameter Δt, such that if the time delta between two posts coming from the same IP address is smaller than Δt, the messages are grouped together. The full IP clustering is obtained by transitively applying the property: two messages a, b are in the same IP grouping if and only if there exists a sequence of messages staring at a and finishing at b sharing the same IP and such that the consecutive time deltas are smaller than Δt. Δt is taken to be 3 hours.

A further cleaning-up step we take is removing duplicate messages within groups. For our dataset, we do this by relying on the presence of an `update` flag. We only keep the latest version of an updated comment.

Finally, we compute the content complexity metric for all the groups of two messages or more, and we back-propagate the results to the messages themselves. In the case of the host name grouping, when a message contains several linked URLs and thus belongs to several host name groupings, we back-propagate the lowest content complexity metric among all these groups. If a grouping resulted in a singleton, we assign the normal content complexity zero to the corresponding feature.

We call this set of features F_C, for *complexity* features. Thus, there are exactly four F_C features per message. We also define F_{LGS} for *log-group-size* which as their name suggests give the logarithm of the grouping size (four features per message again), and F_{dG} for *is-defined-group* which are four binary features indicating whether the associated grouping resulted in a singleton or not.

3. LATENT VARIABLE MODEL

We now turn to the design of the classifier. As our base component, we use a logistic regression (LR), which accommodates well to unknown dependencies between features. Besides, classification is intuitively monotonic in the complexity features: a low content complexity score is highly indicative of bulk and thus spam traffic.

More formally, our approach of classifying social media comments trains a model using a dataset of n points: $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, \mathbf{x}_n, y_n)\}$. For comment i in the data set, $\mathbf{x}_i$ is the vector representing its associated features and $y_i \in \{0, 1\}$ represents the label (our convention is 0 for ham, 1 for spam).

Furthermore, the label y_i might be a noisy representation of the example's *ground truth* label $g_i \in \{0, 1\}$. For example, a spam comment could be mislabeled as benign in the training set (label $y_i = 0$) when its ground truth label is $g_i = 1$. The goal of classification is to train a model so that given a feature vector $\mathbf{x}_i$ we can predict g_i with high accuracy. Because the ground truth label is not observed directly (only $\mathbf{x}$ and the noisy y are observed), we model g as a latent variable and derive a latent version of LR similar to the approach proposed in [17].

3.1 Model Description

Here we describe the probabilistic framework for the latent model, which is illustrated in Figure 3. The variable X represents the distribution of messages in the feature space, G represents ground truth labels, and Y represents the noisy labels. We assume both X and Y are visible and G is hidden. The conditional probabilities which define the model are $P(G|X)$ (for inferring the ground truth label given a data point) and $P(Y|G)$ (for modeling the noise in the dataset's labels).

We parametrize the conditional probability $P(G|X)$ using the usual logistic regression model as follows:

$$P(G = g|X = \mathbf{x}) = \sigma(\mathbf{x}^\top \mathbf{w})^g \sigma(-\mathbf{x}^\top \mathbf{w})^{(1-g)}, \tag{3}$$

where $\sigma(z) = [1 + e^{-z}]^{-1}$. We mention but notationally omit that we append a dummy column vector of all 1's to handle bias.

We define the noise model $P(Y|G)$ as a mixture of Bernoulli distributions which we express as

$$P(Y{=}y|G{=}g) = \alpha^{gy}(1-\alpha)^{g(1-y)}(1-\beta)^{(1-g)y}\beta^{(1-g)(1-y)}.$$

Because y and g are either 0 or 1, the exponents act as indicator functions that help select a probability based on whether the ground truth label and data label match. For example, α is the probability that a spam blog comment is labeled correctly ($g = 1$ and $y = 1$), $(1 - \alpha)$ is the probability that a spam blog comment is labeled incorrectly ($g = 1$ and $y = 0$), $(1 - \beta)$ is the probability that a ham blog comment is labeled incorrectly ($g = 0$ and $y = 1$), and β is the probability that a ham blog comment is labeled correctly ($g = 0$ and $y = 0$).

3.2 Learning

The parameters we have to learn for our model are the classification weight vector $\mathbf{w}$, and the noise parameters α, β. We use an EM algorithm to maximize the model's log-likelihood $L(\mathbf{w}, \alpha, \beta) = \sum_i L_i(\mathbf{w}, \alpha, \beta)$, where each data point's log likelihood is

$$\begin{aligned} L_i(\mathbf{w}, \alpha, \beta) \triangleq\ & g_i y_i \log \alpha + g_i(1-y_i)\log(1-\alpha) + \\ & (1-g_i)y_i \log(1-\beta) + \\ & (1-g_i)(1-y_i)\log\beta + \\ & g_i \log\sigma(\mathbf{x}_i^\top \mathbf{w}) + (1-g_i)\log\sigma(-\mathbf{x}_i^\top \mathbf{w}). \end{aligned} \tag{4}$$

For the E-step, we want to compute the expected value for each hidden ground truth label g_i in our dataset using existing estimates for conditional probabilities in our model. Because g_i is a binary value, we have $\mathbb{E}[g_i|\mathbf{x}_i, y_i] = P(g_i{=}1|\mathbf{x}_i, y_i)$, which we calculate as follows:

$$P(g_i{=}1|\mathbf{x}_i, y_i) = \frac{P(g_i{=}1|\mathbf{x}_i)P(y_i|g_i{=}1)}{\sum_{g\in\{0,1\}} P(g|\mathbf{x}_i)P(y_i|g)}. \tag{5}$$

Then, we assign $\hat{g}_i \leftarrow \mathbb{E}[g_i|\mathbf{x}_i, y_i]$, substitute $\hat{g}_i$ for g_i in Equation 4, and then proceed to the M-step.

In the M-step, we reassign the model parameters $\mathbf{w}$, α, and β to maximize the log-likelihood. Using the logistic regression model and the L-BFGS optimization routine [14], we reassign the parameter vector $\mathbf{w}$. The noise parameters α and β are reassigned as follows:

$$\alpha \leftarrow \frac{\sum_i \hat{g}_i y_i}{\sum_i \hat{g}_i}, \tag{6}$$

$$\beta \leftarrow \frac{\sum_i (1-\hat{g}_i)(1-y_i)}{\sum_i (1-\hat{g}_i)}. \tag{7}$$

A good initialization is a key to a successful run of the EM. Experimentally, we found that initializing $\mathbf{w}$ to be the result of the plain LR on the dataset and setting $\alpha = \beta = 0.5$ provides the best results. Hence, we use this initialization strategy in what follows.

Finally, we stop the EM loop as soon as the relative L_1 difference of two consecutive $\mathbf{w}$ parameter vectors is within 1%, i.e.,

$$\frac{|\mathbf{w}^{(i)} - \mathbf{w}^{(i-1)}|}{|\mathbf{w}^{(i-1)}|} \leq 0.01,$$

or if we exceed 300 iterations.

3.3 Feature Expansion

To capture potential non-linear effects among features, we use polynomial basis functions over those features. Specifically, if $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_k)$ is our initial feature vector of k features, we expand it to all the terms in the expansion of the quadratic polynomial $(1+\mathbf{z}_1+\ldots+\mathbf{z}_k)^2$. We define $\text{expan}(\mathbf{z}, 2)$ as the mapping from the original data vector $\mathbf{z}$ to the quadratic polynomial expansion.

Formally:

$$\text{expan}(\mathbf{z}, 2) = (1) \cup (\mathbf{z}_i)_i \cup (\mathbf{z}_i \mathbf{z}_j)_{i \leq j}.$$

Expanding a feature vector of size k results in a new vector of size $\Theta(k^2)$, thus we can only use this strategy when the initial number of features is small.

Kernel logistic regression can be also used to handle non-linear effects among features. However, it usually results in more complex models and requires more computation budget to train a model and make predictions than the linear method. We leave it as future work to explore the feasibility of applying kernel logistic regression to our problem.

4. EVALUATION METHOD

In this section, we explain our evaluation method. We start by describing our dataset and motivate a sampling strategy for efficient hand-labeling. We conclude by explaining our actual labeling process.

4.1 Dataset

The dataset we use for our evaluations comes from a provider that aggregates comment threads from a variety of social media platforms, such as personal, political or business oriented blogs, news websites, various entertainment websites, etc. The collection time period is four months between December 2011 and March 2012. Non-English languages are present at a significant level: we often encounter comments in Spanish, Portuguese, Chinese, Japanese or French. The comment data also comes with machine-generated labels for each comment which contain some error. The provider is a well reputed source which implement cutting-edge statistical filtering techniques.

In practice, the dataset resides in a Hadoop cluster where each record has the form:

```
timestamp, user_id, end_user_ip,
article_permalink, content, spam_label
```

For a rigorous evaluation of our algorithms, we divide this dataset into two equal length time periods of two months each. The first time period will always serve as the training set (set A), while the second will be used for scoring only (set B). In particular, when computing the groupings for the content complexity metrics, we never aggregate together two messages coming from distinct subsets: future information is not available when training, and past information is not available when scoring.

A summary of the split dataset is presented in Table 4.1. For equal time periods, the scoring dataset is significantly larger than the training dataset. We explain this fact by the rapid gain in popularity and expansion of the associated web service. Note that the anti-aliasing process for grouping by IP can produce several distinct groups sharing the same IP, but at different times, as described in 2.3.

Characteristic	Training set A	Scoring set B
Time period	12/01/2011 01/31/2012	02/01/2012 03/31/2012
Number of comments	25m (420k/day)	40m (660k/day)
Number of distinct user ids	990k	1.5m
Distinct anti-aliased IP addresses	3.9m	6.5m
Number of distinct article permalinks	1.4m	2.3m
Distinct hosts from linked URLs	99k	150k
Labeled spam	1.8%	3.6%

Table 1: Learning and training set characteristics.

4.2 An Unbiased Sampling Strategy

While we do have the provider's labels for each of sets A and B, we can only use them for training. Indeed, we know and we observe in the following evaluations that these labels display a non negligible false positive rate. Since our goal is to compare our methods both against themselves and the provider, we must establish a form of ground truth by manually labeling carefully chosen samples from B.

Also, notice that we still want to use the provider's labeling for training, as a way to minimize the hand-labeling labor and test the latent LR in the wild.

While remaining unbiased, our sampling strategy is driven by two stringent constraints:

- the high cost of obtaining ground truth labels and
- the scarcity of positive (spam) instances in the data.

Because labeling blog comments requires looking up contextual information (e.g., the actual blog post, surrounding comments, user profile and user past activity, translating the contents in English when necessary), it is a rather time consuming task. Under such circumstances, a naive uniform sampling over B is inefficient.

The problem is exacerbated because only a small fraction of the comments are actually spam. In practice, this means that naive uniform sampling tends to have a strong bias towards extracting ham comments.

The approach we choose relies on the observation that if one is just interested in the precision score, i.e., the proportion of spam among all the flagged instances, then restricting the sampling to be uniform only over the flagged instances is sufficient. This is expressed by the following basic result.

Let $l : B \to \{0; 1\}$ the ground truth labeling (0=ham, 1=spam). Let τ be a detection threshold and f a classifier, e.g., any function mapping a feature vector to a real number in the 0-1 range. For notational convenience, we define $f_\tau(x) = \mathbb{I}_{f(x)>\tau}$ to be our decision function. τ-level precision is defined as:

$$p_\tau = \frac{\sum_x f_\tau(x) l(x)}{\sum_x f_\tau(x)}.$$

Let $B^+ = \{x \in B, l(x) = 1\}$ be the subset of spam messages and $\chi_\tau^+ = \{x \in B, f_\tau(x) = 1\}$ be the flagged users at detection level τ.

PROPOSITION 4.1. *Let $\tilde{\chi}_\tau^+ \subset \chi_\tau^+$ a uniformly selected subset of size n. The following is an unbiased estimator of p_τ:*

$$\tilde{p}_\tau = \frac{\sum_{x \in \tilde{\chi}_\tau^+} l(x)}{n}$$

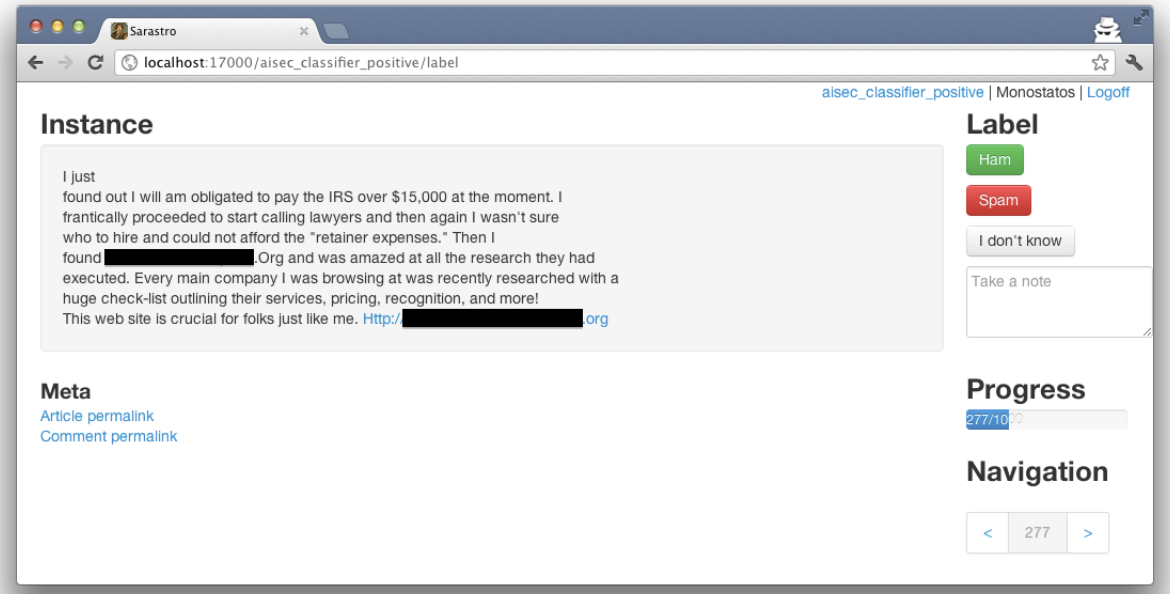


Figure 4: Labeler Monostatos has already labeled 277 over 1000 instances in task *aisec_classifier_positive*.

and

$$\operatorname{Var} \tilde{p}_\tau = \frac{|\chi_\tau^+| - n}{n(|\chi_\tau^+| - 1)}(p_\tau - p_\tau^2) \leq \frac{1}{4n}.$$

Proof: see Appendix.

If we had a single classifier f to evaluate, we could start by finding a detection threshold τ_0 such that $\frac{|\chi_{\tau_0}^+|}{|B|} = 5.66\%$, meaning that the classifier flags 5.66% of the dataset at level τ_0. This is justified by the fact that the provider's labeling which serves as a baseline comparison is such that $\frac{|\chi^+|}{|B|} = 3.62\%$, so that we need at least the same proportion of flagged instances for comparison. Adding a reasonable safety margin gets us to 5.66%.

From there, we uniformly sample n instances $\tilde{\chi}_{\tau_0}^+ \subset \chi_{\tau_0}^+$ to obtain the base evaluation sample for f. Notice that for the same classifier f and $\tau > \tau_0$, any subset $\{x \in \tilde{\chi}_{\tau_0}^+, f(x) > \tau\}$ is also a uniform sample in χ_τ^+, albeit of a smaller size.

Besides the fact that such sampling does not directly yield the recall or equivalently the false negative rate of the classifier, a major disadvantage is that the sampling is completely dependent on the classifier. A sampling that is uniform for one classifier has no reason to be uniform for another one. Thus, one practical issue is that it is difficult to sample and start labeling before the classifier is defined.

Concerning the recall issue, we notice that it is sufficient to multiply the estimated precision $\tilde{p}_\tau$ by the volume of flagged instances $|\chi_\tau^+|$ to obtain an unnormalized measure which is directly proportional to the recall of the algorithm. As $|\chi_\tau^+|$ is exactly known, the uncertainty on the unnormalized recall is simply $\sigma(\tilde{p}_\tau)|\chi_\tau^+|$. To obtain a dimensionless number for the evaluations, we use $\tilde{p}_\tau \frac{|\chi_\tau^+|}{|B|}$ as our unnormalized recall measure.

Finally, we build our evaluation dataset using the following strategy. Fix a uniform sampling rate $0 < r < 1$ and a flagged volume $0 < v < 1$. Let $f^1, \ldots, f^k$ be k classifiers we are interested in evaluating. For each classifier i, compute by binary search the minimal detection threshold τ_0^i such that $\{x \in B | f^i(x) > \tau_0^i\}/|B| \approx v$. Uniformly sample with rate r in subset:

$$\{x \in B | f^1(x) > \tau_0^1\} \cup \cdots \cup \{x \in B | f^k(x) > \tau_0^k\}.$$

The resulting sample provides by construction an unbiased evaluation sample for each classifier w.r.t. the $\tilde{p}$ measure, provided no classifier i is operated at detection thresholds lower than τ_0^i. In practice, we choose $r = 0.06\%$ and $v = 5.66\%$, which corresponds to a minimum of $rv|B| = 1358$ samples to label.

4.3 Labeling Process

Once constructed, we must hand-label the sampled set. To this end, we wrote a simple Python WSGI web application managing simultaneous labelers and datasets. Figure 4 shows the labeling screen the labeler is presented after logging in and selecting a task. In the labeling screen, the possible actions for the user are: (1) assign one of the three labels {*ham*, *spam*, *I don't know*} along with an optional short note on the instance; (2) browse back or forward to correct a label; (3) look at the comment in its context.

The *spam* label is assigned to a comment when at least one of the following is true.

1. Comment links or refers to a commercial service (most of the time luxury, beauty, pharmaceutical, dating, or financial offers) and appears to be completely unrelated to the comments thread. No URL needs to be present as the spammer can include it in the account profile, or simply ask the users to search for a particular term.

2. Comment is a generic "thank you" or "nice blog", with the intent of boosting the user account reputation (when available, we examine user's history).

Each comment gets three potentially conflicting labels from the three labelers, who used much more or even completely different information for labeling than what the algorithm uses. With three labelers and three labels, an instance label can only be one of the following. It is unanimous when the labelers choose the same label, a majority when exactly two labelers choose the same label, and conflicting when all three labelers choose a different label. Table 4.3 summarizes the labelers disagreement on all the evaluated instances. We briefly tried to resolve conflicts, but quickly backtracked as the process is very time consuming. Instead, we used the majority label when one was available, and we treated conflicts and *I don't know* labels as *spam*. We also experimented with turning the uncertain labels to *ham* or simply discarding them from the dataset at the risk of breaking the uniformity of the sample. None of the policies noticeably changed the results and the conclusions.

Total number of labeled comments	2349
Unanimity	1575 (67%)
Majority	728 (31%)
Conflicted	46 (2%)
I don't know	28 (1%)

Table 2: The final evaluation sample characteristics. Most of the comments result in an unanimous label while a tiny fraction result in intra-labelers conflicts.

5. EVALUATION

We evaluate the approach of using our latent variable model (Section 3) with content complexity features (Section 2) associated with social media comments from our data set (Section 4). Over the course of the evaluation, we want to answer the following questions: How effective are content complexity features for classification? Does the noise-tolerant latent variable model provide an improvement over standard logistic regression (which is not noise-tolerant)? And which combinations of complexity features provide the most accurate classification?

Figure 5 shows the precision-recall curves for the classification algorithms, comparing Plain LR to Latent LR using four different combinations of features. In all figures, the precision and recall of the data provider's labels is shown for reference (it is a dot because the labels are discreet "ham/spam" labels). The scaling of the x- and y-axis of all figures is the same. The filled area around the curve is an upper bound of a standard deviation above and below the expected precision and recall for a given threshold, as given by proposition 4.1. Notice the large precision variance in the low recall region, as only a handful instances are labeled there (low n). We stress the fact that we use the same labeled set for all the different algorithms and features, meaning that the relative position of the curves is by itself significant.

A few high-level trends emerge when looking at the plots. First is that in all cases, all our classifiers that use complexity features outperform the labels from the reference data provider. This is a notable result because it means that it is possible to use a training set labeled by another algorithm to train a classifier that can outperform the original algorithm. This observation also makes sense because the complexity features help model a key characteristic of spam: the repetitive, uninformative nature of the posted content associated with a user/IP/embedded hostname.

Second, from Figures 5(a), 5(c), and 5(d), we see that for most feature sets the Latent LR outperform the Plain LR (i.e., having a higher precision for a given recall, or vice versa). Thus, a noise-tolerant algorithm like Latent LR can provide an improvement over Plain LR, especially in a data set that could contain noisy labels. The exception is Figure 5(b), where the performance of Plain and Latent LR are indistinguishable. This happens in Figure 5(b) because in a quadratic expansion with a higher number of dimensions, the adverse affects of mislabeling are less noticeable because the decision boundary itself is more complex (i.e., more nonlinear with respect to the original, unexpanded features). Still, because Latent LR performs better than Plain LR in most cases (and in the worst case performs at least as well as Plain LR), it is safe to use the Latent LR in practice.

But given the Latent LR algorithm, which of the feature set combinations are most appropriate for deploying in practice? As we can see from Figure 6, the performance of the different feature sets are relatively close overall. A slight edge may be awarded to the $F_C \cup F_{LGS}$ feature set because it has good performance in both the high-precision region and high-recall regions of the precision-recall space (the parts of the graph where unnormalized recall is ≤ 0.75 and ≥ 1.75, respectively). By contrast, other feature sets may favor higher recall at the expense of lower precision (like $(F_C \cup F_{LGS})^2$), while yet others may favor high precision at the expense of recall (like the remaining feature sets). For example, at a recall of 0.5, $F_C \cup F_{LGS}$ scores a precision of over 97%, whereas the quadratic expansion $(F_C \cup F_{LGS})^2$ only achieves 90% precision. And at a recall of 1.8, $F_C \cup F_{LGS}$ achieves a precision of 69%, whereas the precision for F_C and $F_C \cup F_{dG}$ is 65%.

On a computational complexity standpoint, there are three phases with distinct performances: features extraction, learning, and scoring. The scoring phase consists solely of an inner product of size the number of features and a scalar comparison, thus its running time is negligible. Depending on the number of features used, training a plain logistic regression on the whole dataset takes from a few minutes to an hour on a 2 GHz Intel Core i7. On the same machine, training a latent logistic regression takes between half an hour to several hours. The feature extraction phase is the most computationally expensive and is done on a Hadoop grid with 8 mappers and 6 reducers. Grouping comments (map and sort phase) and compressing groups (reduce phase) takes a few hours, thus computing the 4 core groupings takes about half a day.

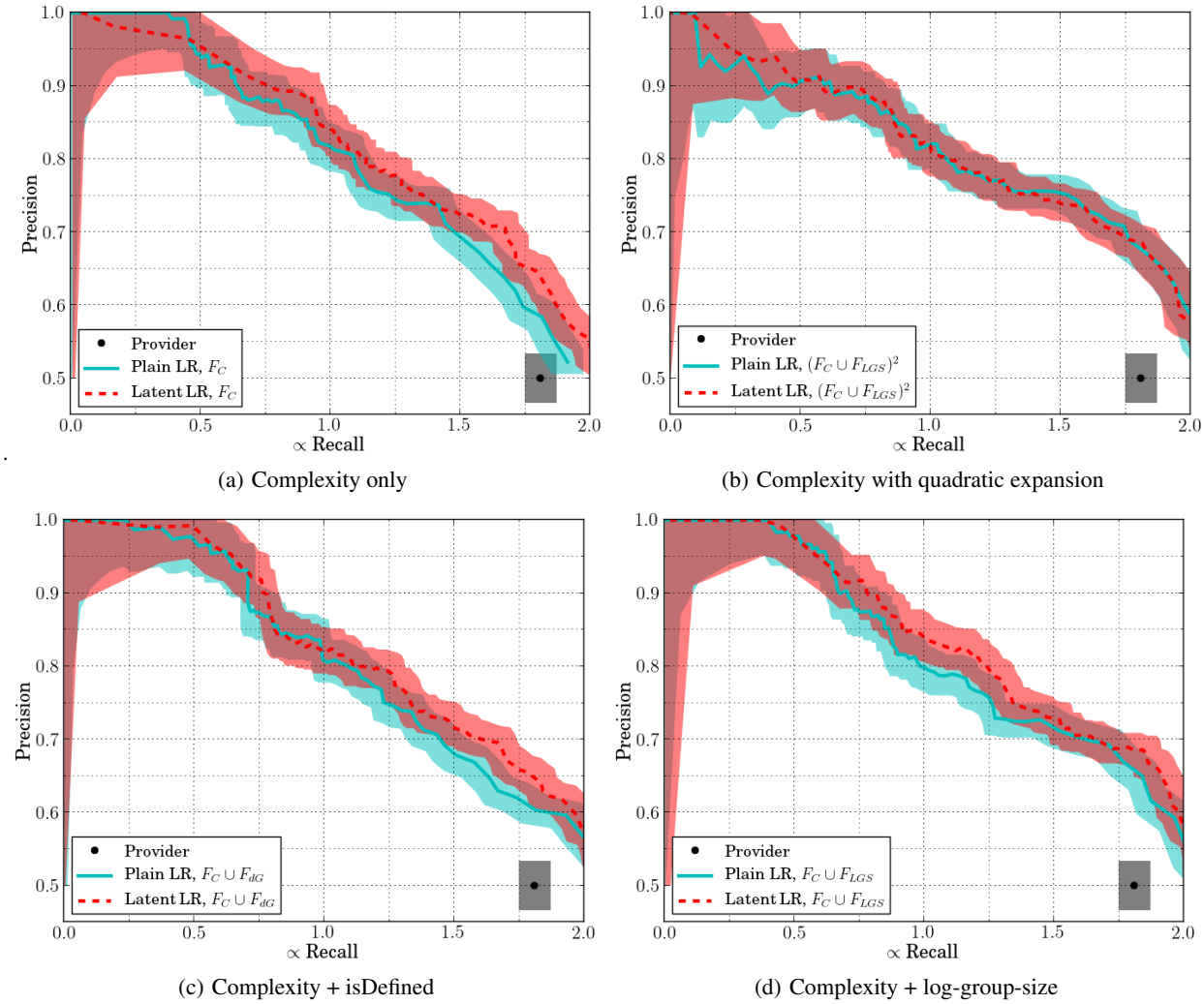
Overall, because of the performance of the complexity features combined with the latent variable model, they provide a promising complementary to existing comment spam detection techniques.

Finally, we mention that there are two arbitrarily fixed parameters we did not evaluate. The first is the Δt parameter we use for IP anti-aliasing. Everything else considered equal, decreasing this parameter breaks down groupings into smaller ones, until all IP groupings are singletons. On the contrary, increasing it will merge all groups sharing the same IP together, thus loosing the anti-aliasing benefit. Thus, there is arguably an optimal value for the parameter, which we did not try to evaluate. An adversary might also want to arrange for posting her messages with a temporal rate smaller than Δt^{-1}, thus it also acts as an implicit upper bound on the spam rate of a given IP.

In the same fashion, the second parameter which evaluation we did not take is the time window on which we compute the features. This parameter is equal to two months in our evaluation, and presents a behavior similar to the one of Δt when varied.

6. RELATED WORK

User-generated content (e.g., comments and reviews) is widely available on blog and online shopping websites, and online social networks. Mishne and Glance analyze various aspects of Web blog comments in their 2006 study, which (among other findings) showed that blog comments could be as crucial as the original blog



(a) Complexity only

(b) Complexity with quadratic expansion

(c) Complexity + isDefined

(d) Complexity + log-group-size

Figure 5: Precision-recall plots for Plain LR and Latent LR using different combinations of features, with standard deviations for both axis.

post in providing original, searchable content to the article [11]. Brennan et al. use metadata, for example user activity, reputation, comment time, and posts' contents to predict the community ratings of the Slashdot comments [3].

Ranking and rating help to promote high-quality comments and demote low-quality ones. Using a large corpus of Digg stories and comments, Hsu et al. collect a set of features related to user, content, and popularity to train a Support vector Machine (SVM) to rank comments according to quality [6]. They find that the tandem of user and content based features are among the most effective ones. Their measure of content complexity is different from ours because they compute the entropy of a single comment message, whereas the study in this paper computes messages over a set of comments grouped. Chen et al. show that the quality of a comment is almost uncorrelated to the ratings of comment, and propose a latent factor model to predict comment quality based on its content, author reputation, agreement in opinions, etc [4]. Mishra and Rastogi apply semi-supervised learning techniques to address the user bias issue in comment ratings using information from user-comment graph [12].

Using data compression for spam filtering is not new. Bratko et al. propose adaptive data compression models for email spam filtering [2]. They train the classifier by building two compression models from the training corpus, one from spam examples and one from legitimate examples, and then predict the label of new instances using a minimum description length principle [1].

Mishne et al. study the feasibility of using unigram language models for detecting off-topic link spam blog comments [10]. They use Kullback-Leibler divergence between language model of the blog post and that of the comment to predict whether a comment is link spam.

Shin et al. study comment spam on a research blog using various content specific and host-based features [21]. Their approach showed significant performance but was limited to only one particular blog whereas our data are more diverse including a wide range of personal and commercial sites.

Gao et al. quantify and characterize spam campaigns on online social networks using a dataset of "wall" messages between Facebook users [5]. Their approach first groups together wall posts that share either the same URL or strong textual similarity using

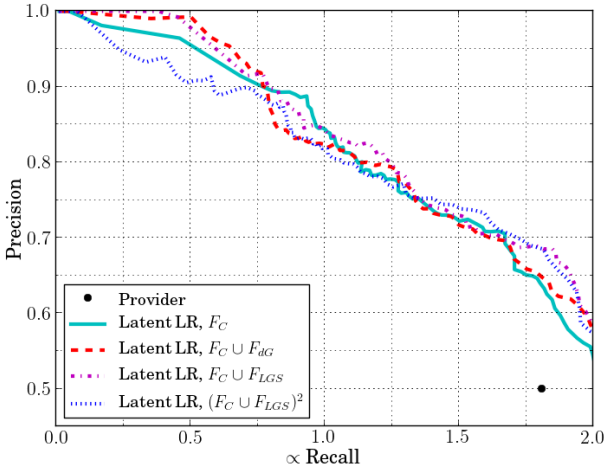


Figure 6: A side-by-side comparison of the precision-recall performance for different feature sets using Latent LR.

a graph-based clustering technique, and then apply threshold filters based on the wall post sending rate of user accounts and time correlation within each subgraph to distinguish potentially malicious clusters from benign ones.

Lee et al. propose a honeypot-based approach for uncovering social spammers in online social network [7]. They deploy social honeypots for harvesting deceptive spam profiles from social networking communities, and create spam classifier using machine learning methods (e.g., SVM) based on a variety features derived from user demographics (e.g., age, gender, location), user contributed content (e.g., blog posts, comments, tweets), user activity features (e.g., posting rate, tweet frequency), and user connections (e.g., number of friends, followers, following).

Email spam problem has been extensively studied in recent years. One category of approaches are based on machine learning techniques. Sculley and Wachman show that Support Vector Machine (SVM) classifiers produce state-of-the-art performance for online content-based detection of spam on the web (e.g., email, comments, splogs) [19]. To relieve the burden of labeling large-scale data, Xu et al. propose the use of active semi-supervised learning method for the training spam classifier [23]. Their approach can leverage both unlabeled emails and asynchronous human feedback to build high performance classifiers with small amount of labeled data. Instead of using content-based information, Ramachandran et al. develop a spam filtering system to classify email senders based on their sending behavior [16]. Xie et al. characterize spamming botnets using information from both spam payload (e.g., embedded URLs) and spam server traffic, and develop a signature-based framework to detect botnet-based spam emails and botnet membership [22].

7. CONCLUSION AND FUTURE WORK

In this paper we described a robust approach of detecting spam in social media. Our approach uses content complexity of comments with a latent logistic regression classifier. This approach is a first step in detecting comment spam from noisy and missing labels using language-agnostic features. Our evaluation shows the approach is general and robust, and provides new insights for industry to further improve their techniques for fighting comment spam.

Moving forward, a diverse of research directions can be pursued to improve the performance of comment spam detection. We highlight a few below.

Alternative definition of spam. Our algorithm is designed based on the informativeness of a comment, and only uninformative comments are considered as spam. The algorithm misclassifies in cases where this assumption is not true (e.g., short messages saying "wow," "nice," or "thanks," messages containing only URLs). A more broad definition of spam, along with the classifiers derived from it, could provide bigger coverage on spam cases and further improve spam detection performance.

More features. In addition to the content-complexity based features, there are a variety of other features we would like to incorporate into our framework to further improve its detection accuracy. Among them, the containment of spam trigger words and URLs has been shown to be very discriminative [22], as well as the posting behavior of users [16], user reputation [4] and user-comment relationship graph [12].

Online update and operation. In production, spam detectors are usually deployed and operated in an online setting. When a new example arrives, features need to be extracted from the data and fed to the classifier. The classifier makes a prediction, is told if its prediction is correct, and updates its model accordingly. While our classifier itself can be easily adapted for online setting, the feature sets are very intertwined: when a new data point is assigned to groups with several messages in it, the complete calculation of the content complexity measure has to be re-triggered and back-propagated to all affected messages. This could lead to an expensive online update step for several data points. We leave it as a future work to develop an efficient (incremental) approach for extracting content complexity features from comment data.

Human in the loop. There are both passive and active ways to incorporate human experts in the predict-feedback-update loop for improving and adapting the spam classifier. In a passive way, experts may examine (some of) the classification results (e.g., those comments classified as spam) to identify the mistakes that the classifier has and re-train the classifier using the misclassified examples. In an active setting, the classifier can use an active learning approach to identify the weakest classification region of itself and ask human expert to collect more labeled samples to improve its confidence in that region.

Adversary-awareness. Knowing what features that a spam classifier is built on, an adversary may modify its comments and its behavior of comments posting to evade the spam classifier. We should develop learning algorithms that are robust to adversaries, using features that are resistant to adversarial modification, and adopt an online approach that can continuously adapt the classifier to the adversarial changes.

Distributed learning framework. In this social network era, users are generating enormous content. With massive data being continuously generated, it is imperative to develop a distributed learning framework to speed up the (online) training, testing and updating process for spam classifier.

Acknowledgement

We thank Impermium Corporation for the data as well as invaluable suggestions to improve this work. We also thank anonymous reviewers for their helpful comments on the paper. We are grateful to the Intel Science and Technology Center for Secure Computing, DARPA (grant N10AP20014), the National Science Foundation (through the TRUST Science and Technology Center), and the US Department of State (DRL) for supporting this work in part.

The opinion in this paper are those of the authors and do not necessary reflect the opinion of any of the funding sponsors.

8. REFERENCES

[1] A. R. Barron, J. Rissanen, and B. Yu. The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2760, 1998.

[2] A. Bratko, G. V. Cormack, B. Filipic, T. R. Lynam, and B. Zupan. Spam filtering using statistical data compression models. *Journal of Machine Learning Research*, 7:2673–2698, Dec 2006.

[3] M. Brennan, S. Wrazien, and R. Greenstadt. Learning to extract quality discourse in online communities. In *Workshops at the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[4] B.-C. Chen, J. Guo, B. Tseng, and J. Yang. User reputation in a comment rating environment. In *Proceedings of KDD'11*, 2011.

[5] H. Gao, J. Hu, C. Wilson, Z. Li, Y. Chen, and B. Zhao. Detecting and characterizing social spam campaigns. In *Proceedings of IMC*, 2010.

[6] C.-F. Hsu, E. Khabiri, and J. Caverlee. Ranking Comments on the Social Web. In *Proceedings of the IEEE International Conference on Computational Science and Engineering (CSE)*, pages 90–97, 2009.

[7] K. Lee, J. Caverlee, and S. Webb. Uncovering social spammers: social honeypots+ machine learning. In *Proceedings of SIGIR*, 2010.

[8] K. Levenberg. A method for the solution of certain non-linear problems in least squares. In *Quarterly of Applied Mathematics*, volume 2, pages 164–168, 1944.

[9] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. In *SIAM Journal on Applied Mathematics*, volume 11, pages 430–441, 1963.

[10] G. Mishne, D. Carmel, and R. Lempel. Blocking Blog Spam with Language Model Disagreement. In *Proceedings of the International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, 2005.

[11] G. Mishne and N. Glance. Leave a Reply: An Analysis of Weblog Comments. In *Proceedings of the International World Wide Web Conference (WWW)*, 2006.

[12] A. Mishra and R. Rastogi. Semi-supervised correction of biased comment ratings. In *Proceedings of WWW'12*, 2012.

[13] I. Pavlov. LZMA SDK (software development kit), 2007.

[14] P. L. R. H. Byrd and J. Nocedal. A limited memory algorithm for bound constrained optimization. In *SIAM Journal on Scientific and Statistical Computing*, volume 16, pages 1190–1208, 1995.

[15] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. In *Proceedings of Sigcomm*, 2006.

[16] A. Ramachandran, N. Feamster, and S. Vempala. Filtering spam with behavioral blacklisting. In *Proceedings of CCS'07*, 2007.

[17] V. C. Raykar, S. Yu, L. H. Zhao, G. H. Valadez, C. Florin, L. Bogoni, and L. Moy. Learning from crowds. *Journal of Machine Learning Research*, 11:1297–1322, April 2010.

[18] T. Schurmann and P. Grassberger. Entropy estimation of symbol sequence. *Chaos (Woodbury, N.Y.)*, 6(3):414–427, Sept. 1996.

[19] D. Sculley and G. M. Wachman. Relaxed online svms for spam filtering. In *Proceedings of SIGIR'07*, 2007.

[20] C. Shannon. Prediction and Entropy of Printed English. *Bell System Technical Journal*, 30(1):50–64, 1951.

[21] Y. Shin, M. Gupta, and S. Myers. Prevalence and mitigation of forum spamming. In *INFOCOM, 2011 Proceedings IEEE*, pages 2309–2317. IEEE, 2011.

[22] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming botnets: Signatures and characteristics. In *Proceedings of SIGCOMM 08*, 2008.

[23] J.-M. Xu, G. Fumera, F. Roli, and Z.-H. Zhou. Training spamassassin with active semi-supervised learning. In *Proceedings of the 6th Conference on Email and Anti-Spam (CEAS'09)*, 2009.

[24] L. Zhuang, J. Dunagan, D. R. Simon, H. J. Wang, I. Osipkov, G. Hulten, and J. Tygar. Characterizing botnets from email spam records. In *Proceedings of LEET*, 2008.

9. APPENDIX

This is a formal proof of proposition 4.1.

$$\begin{aligned}
\mathbb{E}[\tilde{p}_\tau] &= \binom{|\chi_\tau^+|}{n}^{-1} \sum_{|\tilde{\chi}_\tau^+|=n} \frac{\sum_{x,x\in\tilde{\chi}_\tau^+} l(x)}{n} \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n} \sum_{|\tilde{\chi}_\tau^+|=n} \sum_{x\in B} l(x) 1_{x\in\tilde{\chi}_\tau^+} \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n} \sum_{x\in B} l(x) \sum_{|\tilde{\chi}_\tau^+|=n} 1_{x\in\tilde{\chi}_\tau^+} \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n} \sum_{x\in B} l(x) \binom{|\chi_\tau^+|-1}{n-1} 1_{x\in\chi_\tau^+} \\
&= \frac{1}{n} \sum_{x\in B} l(x) 1_{x\in\chi_\tau^+} \frac{n}{|\chi_\tau^+|} = \frac{1}{|\chi_\tau^+|} \sum_{x\in B} l(x) 1_{x\in\chi_\tau^+} \\
&= p_\tau
\end{aligned}$$

proving that $\tilde{p}_r$ is unbiased.

$$\begin{aligned}
\mathbb{E}[\tilde{p}_\tau^2] &= \binom{|\chi_\tau^+|}{n}^{-1} \sum_{|\tilde{\chi}_\tau^+|=n} \left(\frac{\sum_{x,x\in\tilde{\chi}_\tau^+} l(x)}{n} \right)^2 \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n^2} \sum_{|\tilde{\chi}_\tau^+|=n} \left(\sum_{x\in B} l(x) 1_{x\in\tilde{\chi}_\tau^+} \right)^2 \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n^2} \sum_{|\tilde{\chi}_\tau^+|=n} \sum_{x,y\in B} l(x) l(y) 1_{x\in\tilde{\chi}_\tau^+} 1_{y\in\tilde{\chi}_\tau^+} \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n^2} \sum_{x,y\in B} l(x) l(y) \sum_{|\tilde{\chi}_\tau^+|=n} 1_{x\in\tilde{\chi}_\tau^+} 1_{y\in\tilde{\chi}_\tau^+} \\
&= \binom{|\chi_\tau^+|}{n}^{-1} \frac{1}{n^2} \sum_{x,y\in B} l(x) l(y) \Big(\binom{|\chi_\tau^+|-1}{n-1} 1_{x,y\in\chi_\tau^+} 1_{x=y} \\
&\quad + \binom{|\chi_\tau^+|-2}{n-2} 1_{x,y\in\chi_\tau^+} 1_{x\neq y} \Big) \\
&= \frac{1}{n|\chi_\tau^+|} \sum_{x,y\in B} l(x) l(y) 1_{x,y\in\chi_\tau^+} \left(1_{x=y} + \frac{n-1}{|\chi_\tau^+|-1} 1_{x\neq y} \right) \\
&= \frac{1}{n|\chi_\tau^+|} \Big(\frac{n-1}{|\chi_\tau^+|-1} \sum_{x,y\in B} l(x) l(y) 1_{x,y\in\chi_\tau^+} \\
&\quad + \frac{|\chi_\tau^+|-n}{|\chi_\tau^+|-1} \sum_{x\in B} l(x) 1_{x\in\chi_\tau^+} \Big) \\
&= \frac{1}{n(|\chi_\tau^+|-1)} \left[|\chi_\tau^+|(n-1)p_\tau^2 + (|\chi_\tau^+|-n)p_\tau \right]
\end{aligned}$$

Hence:

$$\begin{aligned}
\operatorname{Var} \tilde{p}_\tau &= \mathbb{E}[\tilde{p}_\tau^2] - \mathbb{E}[\tilde{p}_\tau]^2 = \mathbb{E}[\tilde{p}_\tau^2] - p_\tau^2 \\
&= \frac{|\chi_\tau^+|-n}{n(|\chi_\tau^+|-1)} (p_\tau - p_\tau^2)
\end{aligned}$$