Noname manuscript No.
(will be inserted by the editor)

Graph Regularization Methods for Web Spam Detection

Jacob Abernethy · Olivier Chapelle · Carlos Castillo

the date of receipt and acceptance should be inserted later

Abstract We present an algorithm, WITCH, that learns to detect spam hosts or pages on the Web. Unlike most other approaches, it *simultaneously* exploits the structure of the Web graph as well as page contents and features. The method is efficient, scalable, and provides state-of-the-art accuracy on a standard Web spam benchmark.

1 Introduction

Adversarial Information Retrieval [15] studies how to perform information retrieval tasks, such as searching or ranking, in collections in which some objects have been maliciously manipulated. The most prevalent form of such manipulation is spam, a problem that pervades most electronic communications.

Web spam manifests itself as web content generated deliberately for the purpose of triggering unjustifiably favorable relevance or importance of some Web page or pages [18]. It has been identified as one of the main challenges Web search engines need to address [21], as it not only deteriorates the quality of search results, but also weakens the trust between the user and the search engine provider, and wastes a significant amount of computational resources in the search engine.

It has been observed that spam and non-spam pages exhibit different statistical properties [16], and this difference can be exploited for building automatic classifiers. In fact, a number of machine learning approaches to Web spam detection have been shown effective [26,28,25,23].

From a machine learning perspective, the spam detection task differs from a typical classification task since not only do we have standard features available for every

Jacob Abernethy
University of California, Berkeley
E-mail: jake@cs.berkeley.edu

Olivier Chapelle
Yahoo! Research, Santa Clara
E-mail: chap@yahoo-inc.com

Carlos Castillo
Yahoo! Research, Barcelona
E-mail: chato@yahoo-inc.com

page/host, but we are also given a directed hyperlink structure on our data as well. A hyperlink often reflects some degree of similarity [14,31] among pages. Complex patterns can be observed in the hyperlinks; for instance, in the particular case of spam it has been observed that non-spam hosts rarely link to spam hosts, even though spam hosts will regularly link to non-spam hosts.

The techniques proposed to date for exploiting link-information in web spam classification fit within roughly three categories. One group of techniques analyzes the topological relationship (e.g.: distance, co-citation, etc.) between the Web pages and a set of pages for which labels are known [19,6,24,32,30,22]. Another option is to extract link-based metrics for each node and use these as features in any standard classification algorithm [3]. Finally, it has been shown that the link-based information can be used to refine the results of a base classifier by re-labelling using propagation through the hyperlink graph, or a stacked classifier [11,17].

In this paper we present a learning algorithm that we call WITCH, for Webspam Identification Through Content and Hyperlinks, that directly uses the hyperlink structure during the learning process in addition to page features. Specifically, we learn a linear classifier on a feature space using an SVM-like objective function. The hyperlink data is exploited by way of *graph regularization*, which produces a predictor with the goal that predicted values will vary smoothly between linked pages. Our results suggest that this method of SVM with graph regularization is highly effective at detecting Web spam, outperforming all other state-of-the-art methods that we have implemented.

The primary contributions of this work are as follows:

– We propose a novel approach for Web spam classification using a graph-regularized classifier.
– We demonstrate the effectiveness of this approach in a standard reference collection task.
– We show that our method performs well even with little training data.

Ours is, to the best of our knowledge, the first technique for spam detection that simultaneously uses features and the hyperlink graph directly for training. Note that these features can be a combination of any type of features such as content-based and link-based features.

The rest of this paper is organized as follows. Section 2 describes the learning schema we propose. Section 3 discusses various design choices such as the graph regularization function. Finally, Section 4 compares the performances of our algorithm with several state-of-the-art spam detection techniques.

2 Learning schema

For the remainder of this paper, we will discuss classification of *hosts* as spam or non-spam. A host is a group of Web pages sharing the same "host" component in their URLs. All techniques can be similarly applied to individual pages as well. Assume we are given the following:

– a set of l labeled examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l) \in \mathbb{R}^d \times \{-1, +1\}$, where $\mathbf{x}_i$ denotes the d-dimensional feature vector associated with the i-th host and y_i is its label: $+1$ for spam and -1 for non-spam;
– a set of u unlabeled examples, $\mathbf{x}_{l+1}, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, with $n = l + u$; and

– a weighted directed graph whose nodes are $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Let E be the sets of pairs (i, j) whenever node i is connected to node j, and let $a_{ij} \geq 0$ be the weight of the link from $\mathbf{x}_i$ to $\mathbf{x}_j$. (The weights can be constructed in various ways, which we discuss in Section 3.)

We define the *hinge function*, $[x]_+ \triangleq \max(0, x)$, for any real value $x \in \mathbb{R}$. For convenience, we will often write X for our data matrix, where row i is $\mathbf{x}_i$. Similarly, the vector Y is the corresponding column vector of labels, $Y_i \triangleq y_i$.

2.1 Learning with Graph Regularization

Suppose we want to learn a linear classifier $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$. A familiar approach is to train a linear *Support Vector Machine* (SVM) [29]. In this case, $\mathbf{w}$ is found as the minimizer of the following objective function:

$$\Omega(\mathbf{w}) \;=\; \frac{1}{l} \sum_{i=1}^{l} [1 - y_i(\mathbf{w} \cdot \mathbf{x}_i)]_+^p + \lambda \mathbf{w} \cdot \mathbf{w}, \tag{1}$$

where λ is a regularization parameter and p is either 1 (hinge loss) or 2 (quadratic loss). The above objective function captures the necessary trade-off between fitness and complexity, for we would to choose $\mathbf{w}$ to correctly classify our training data while maintaining a large margin. Here we use the hinge function to represent the loss on the training data, but any convex loss function $R(\cdot, \cdot)$ may be used. Throughout the remainder of the paper, we employ the squared hinge loss,

$$R(t, y) \triangleq [1 - ty]_+^2,$$

which is convenient as it provides a differentiable objective which can easily be optimized by any primal optimization algorithm. The second term of (1), $\mathbf{w} \cdot \mathbf{w}$, represents the inverse size of the margin and is often referred to as the *regularization* term.

For the special case of classification tasks on the Web, one has the additional advantage of the hyperlinks between nodes. Hyperlinks can be represented as a directed graph with edge set E. Of course, hyperlinks are not placed at random, and it has been shown empirically that they imply some degree of similarity between the source and the target node of the hyperlink [20, 14]. Based on this observation, it is natural to add an additional regularizer to the objective function:

$$\Omega(\mathbf{w}) \;=\; \frac{1}{l} \sum_{i=1}^{l} R(\mathbf{w} \cdot \mathbf{x}_i, y_i) + \lambda \mathbf{w} \cdot \mathbf{w} + \gamma \sum_{(i,j) \in E} a_{ij} \Phi(\mathbf{w} \cdot \mathbf{x}_i, \mathbf{w} \cdot \mathbf{x}_j), \tag{2}$$

where a_{ij} is a weight associated with the link from node i to node j. The first two terms correspond to a standard linear SVM described above. The third term enforces the desired graph regularization described above. The function Φ represents any distortion measure, and is chosen according to the problem at hand.

The objective function (2) was first introduced in [5] but in that paper the graph is not given at hand and is constructed from the examples. It was also used for web page classification in [31]. In both cases, Φ was chosen to be $\Phi(u, v) \triangleq (u - v)^2$. This particular metric, “squared distance”, encodes a prior knowledge that two neighbors should have similar predicted values. This case has been well studied [4] and the associated regularizer can be rewritten in terms of the $n \times n$ *Graph Laplacian* matrix L, defined by

$$L_{ij} \triangleq \begin{cases} -a_{ij} & i \neq j \\ \sum_{k=1}^{n} a_{ik} & i = j. \end{cases} \tag{3}$$

We may now write

$$\sum_{(i,j)\in E} a_{ij} \Phi(\mathbf{w} \cdot \mathbf{x}_i, \mathbf{w} \cdot \mathbf{x}_j) = \mathbf{w}^\top X^\top L X \mathbf{w}. \tag{4}$$

It is important to note that squared distance is symmetric on the input. One novelty of our proposed method is that we utilize *asymmetric* graph metrics that are tuned to the particular task of Web spam classification. With spam, hyperlink direction is of great importance since we do not expect genuine hosts to link to spam hosts even when links in the opposite direction are quite common. This has been empirically confirmed in [11,17]. Thus, we will also consider "positive distance squared" as a distortion measure, that is where

$$\Phi(u, v) = \max(0, v - u)^2.$$

This choice of Φ penalizes any solution for which a node i points to a node j with higher spaminess. By construction, spam nodes are labeled 1 and non-spam nodes -1. Thus, a higher value of $\mathbf{w} \cdot \mathbf{x}_i$ indicates a higher predicted spaminess. This choice of Φ does not give a simple representation of the regularization in terms of the Graph Laplacian, yet we can still efficiently optimize (2) as explained in section 2.3.

In Section 3.2 we provide a much more detailed discussion of the choice of graph regularizer and its effect on the performance.

2.2 Additional Slack Variables

When a variety of useful features are available for each node in our graph, the above optimization may be sufficient to predict whether a host is genuine or not. However, in many cases such features are not available, or are simply not useful for the task at hand. In this case, a simple linear classifier $\mathbf{w} \cdot \mathbf{x}$ on the provided feature space may be inadequate.

This problem can be overcome by introducing a parameter z_i for every node i, and learning a classifier of the form $f(\mathbf{x}_i) = \mathbf{w} \cdot \mathbf{x}_i + z_i$. This extra term can be seen as an additional slack variable that gives more freedom to the learned classifier. The introduction of an addition slack variable per node was also proposed in [31]. As also suggested in [31], however, it is necessary to regularize the vector $\mathbf{z} = [z_i]_{i=1}^{n}$ appropriately.

Our new objective becomes:

$$\Omega(\mathbf{w}, \mathbf{z}) = \frac{1}{l} \sum_{i=1}^{l} R(\mathbf{w} \cdot \mathbf{x}_i + z_i, y_i) + \lambda_1 \mathbf{w} \cdot \mathbf{w} + \lambda_2 \mathbf{z} \cdot \mathbf{z} + \gamma \sum_{(i,j)\in E} a_{ij} \Phi(\mathbf{w} \cdot \mathbf{x}_i + z_i, \mathbf{w} \cdot \mathbf{x}_j + z_j). \tag{5}$$

Here we introduce two regularization parameters λ_1 and λ_2 for controlling the values of both $\mathbf{w}$ and $\mathbf{z}$.

The latter objective function is the basis for WITCH, which we now summarize in Algorithm 1.

```
Algorithm 1 WITCH
  Params: $\lambda_1, \lambda_2, \gamma$, convex function $\Phi(\cdot,\cdot)$
  Input: labeled training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)$
  Input: unlabeled set $\mathbf{x}_{l+1}, \ldots, \mathbf{x}_n$
  Input: hyperlink graph $E$ with edge weights $\{a_{ij}\}_{(i,j)\in E}$
  Compute:
      $\mathbf{w}, \mathbf{z} \leftarrow \arg\min_{\mathbf{w},\mathbf{z}} \Omega(\mathbf{w}, \mathbf{z})$,
  with $\Omega$ defined in (5).
  Predict: label node $i$ as $\mathrm{sign}(\mathbf{w}\cdot\mathbf{x}_i + z_i)$.
```

2.3 Optimization

The unconstrained objective function (5) can be efficiently minimized in the primal using the simple techniques described for instance in [12]. This is one of the main differences with [31] which proposes a dual algorithm for solving a similar problem. But as pointed out in [12], primal training is much faster than its dual version in the case of linear classifiers. Below, we present two different primal methods for training.

Conjugate Gradient. Since the objective function is convex and differentiable, one can simply use nonlinear conjugate gradient [27] to optimize it. This is a standard and very efficient method for nonlinear optimization and it only requires the computation of the gradient. For $R(t, y) = (1 - ty)_+^2$ and $\Phi(u, v) = \max(0, v - u)^2$, the gradient with respect to $\mathbf{w}$ is given by:

$$\frac{1}{2}\frac{\partial\Omega}{\partial\mathbf{w}} = \frac{1}{\ell}\sum_{\{i:\ y_i s_i<1\}} y_i\mathbf{x}_i(y_i s_i - 1) + \lambda_1\mathbf{w} + \gamma \sum_{\substack{(i,j)\in E\\ s_i<s_j}} a_{ij}(s_j - s_i)(\mathbf{x}_j - \mathbf{x}_i), \quad (6)$$

where we define the *score* $s_i \triangleq \mathbf{w}\cdot\mathbf{x}_i + z_i$. The gradient with respect to $\mathbf{z}$ is similar. Note that evaluating (6) requires $O(nd)$ operations to compute the predictions s_i and $O(\ell d + |E|d)$ operations to compute both sums. Assuming that the number of conjugate gradient iterations is independent of the size of the training set, which is generally true in practice, the total complexity is $O(nd)$ assuming that $|E| = O(n)$. In terms of memory requirement, it depends on whether or not the graph can fit into memory. In both cases, we have to store z_i and $\mathbf{w}\cdot\mathbf{x}_i$ which are $2n$ numbers. If possible, it is better to load the graph in memory, but if it is too large, it can be read from disk. The number of times it has to be read is equal to the number of conjugate gradient iterations which, in our experiments, was typically on the order of 100. Hence, this algorithm can scale up to very large graphs including, potentially, the entire web graph.

Alternating Optimization. Let us rewrite the objective function (5) as a function of the the spam scores $s_i = \mathbf{w}\cdot\mathbf{x}_i + z_i$:

$$\Omega(\mathbf{w}, \mathbf{s}) = \frac{1}{l}\sum_{i=1}^{l} R(s_i, y_i) + \lambda_1\mathbf{w}\cdot\mathbf{w} + \lambda_2\sum_{i=1}^{n}(s_i - \mathbf{w}\cdot\mathbf{x}_i)^2 + \gamma\sum_{(i,j)\in E} a_{ij}\Phi(s_i, s_j). \quad (7)$$

We propose to minimize (7) by alternating optimization on $\mathbf{w}$ and $\mathbf{s}$, the advantage being that each of the steps can be performed using standard algorithms:

1. Optimize $\mathbf{w}$ (fixed $\mathbf{s}$):
   Ignoring the term independent of $\mathbf{w}$, one has to minimize
   $$\lambda_1 \mathbf{w} \cdot \mathbf{w} + \lambda_2 \sum_{i=1}^{n} (s_i - \mathbf{w} \cdot \mathbf{x}_i)^2.$$
   This is standard regularized linear regression on the entire dataset where the targets are the s_i. Note that it would be straightforward to extend this step to nonlinear architectures: any regression algorithm can be used here.
2. Optimize $\mathbf{s}$ (fixed $\mathbf{w}$):
   For the sake of simplicity, let us consider the case where the loss functions are quadratic: $R(t,y) = (t-y)^2$ and $\Phi(u,v) = (u-v)^2$. The derivative of (7) with respect to s_i is
   $$\frac{1}{2}\frac{\partial \Omega(\mathbf{w},\mathbf{s})}{\partial s_i} = \frac{1}{l} b_i (s_i - y_i) + \lambda_2 (s_i - \mathbf{w} \cdot \mathbf{x}_i) + \gamma \sum_j (a_{ij} + a_{ji})(s_i - s_j),$$
   where $b_i = 1$ for labeled nodes (i.e. $i \leq l$) and 0 otherwise. Note that in the last term the sum is over all j, but a_{ij} is defined to be 0 if there is no link from i to j. At the optimum, the derivative is 0 and
   $$s_i = \frac{\frac{1}{l} b_i y_i + \lambda_2 \mathbf{w} \cdot \mathbf{x}_i + \gamma \sum_j (a_{ij} + a_{ji}) s_j}{\frac{1}{l} b_i + \lambda_2 + \gamma \sum_j (a_{ij} + a_{ji})}.$$
   In other words, the optimal s_i is a weighted average of y_i, $\mathbf{w} \cdot \mathbf{x}_i$ and all spam scores s_j for all neighbors j of i. Note that when λ_2 and γ go to 0, we recover that for a labeled node, $s_i \approx y_i$. So the fix-point solution is of the following form:
   $$\mathbf{s} = \mathbf{v} + M\mathbf{s}, \tag{8}$$
   with
   $$v_i = \frac{\frac{1}{l} b_i y_i + \lambda_2 \mathbf{w} \cdot \mathbf{x}_i}{\frac{1}{l} b_i + \lambda_2 + \gamma \sum_j (a_{ij} + a_{ji})} \quad \text{and} \quad M_{ij} = \frac{\gamma (a_{ij} + a_{ji})}{\frac{1}{l} b_i + \lambda_2 + \gamma \sum_j (a_{ij} + a_{ji})}. \tag{9}$$
   As for PageRank and other standard propagation algorithms, equation (8) can be solved by iterating from an arbitrary starting point $\mathbf{s}^0$:
   $$\mathbf{s}^{t+1} = \mathbf{v} + M\mathbf{s}^t. \tag{10}$$
   The fact that $\max_i \sum_j M_{ij} < 1$ ensures via the Perron-Frobenius theorem that the largest eigenvalue of M is less than 1 and that this iterative procedure will converge.

The overall algorithm uses only standard techniques (regression and graph propagation) and is summarized in Algorithm 2. It is also related to the method of "co-training" [7]. We have indeed two complementary "views", one corresponding to the features as well as another to the graph, and at each iteration the training within one view is leveraged for the other. More precisely, the feature-based classifier training can benefit from an enriched training set containing all the unlabeled pages along with "pseudo-labels" predicted from the graph regularization. Conversely, the graph regularization (a.k.a. the label propagation mechanism) does not only rely on the labeled dataset

Algorithm 2 Minimization of (7) based on alternate optimization.

```
Input and parameters are as in Algorithm 1
**Compute:** initial classifier $f$ based on labeled data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)$.
**Set:** $s_i \leftarrow 0$.
**repeat**
  **Compute:** output of the classifier $f(\mathbf{x}_i)$ for all the pages (labeled and unlabeled).
  **Compute:** $v_i$ as in (9) (simply replace $\mathbf{w} \cdot \mathbf{x}_i$ by $f(\mathbf{x}_i)$).
  **repeat**
    $\mathbf{s} \leftarrow \mathbf{v} + M\mathbf{s}$.
  **until** Convergence
  **Compute:** regression solution $f$ on $(\mathbf{x}_1, s_1), \ldots, (\mathbf{x}_n, s_n)$
**until** Convergence
```

as in standard propagation algorithm such as [32,22], but also takes into account the labels predicted by the feature-based classifier.

Finally note that, for computational reasons, training the feature based classifier on the entire set $(\mathbf{x}_1, s_1), \ldots, (\mathbf{x}_n, s_n)$ might not be feasible. In the same spirit as in co-training, one might restrict the training set to the set of examples for which the pseudo-labels s_i are estimated with enough confidence.

The alternate optimization method may be probably slower to converge than the direct conjugate gradient descent described above. On the other hand, it easier to implement and more scalable as operations on the features and the graph are performed separately. For the particular dataset employed in the present paper, the corpus was quite small we have therefore only implemented the conjugate gradient method.

Note on Retraining. Before proceeding to the next Section, we emphasize one algorithmic difficulty of using graph regularization for the task of web spam prediction. Clearly, the Web graph is not static; pages appear and disappear constantly. One sees that, for any of the proposed optimization methods, we must retrain entirely for each modification in the graph, since node labels are heavily dependent on their neighbors. Strictly speaking, this is true, yet the story is not quite as bad as it seems. Minor additions to the graph are unlikely to make significant changes to the solution of $\mathbf{w}$, a fixed-dimensional vector that was already trained on a large amount of data. The addition of a new node i, however, will require learning a new slack variable z_i. But this value can be chosen to optimize (5) with all other neighboring node scores fixed, a very fast computation. The resulting joint solution will be nearly optimal, and full updates can be performed occasionally from here with a "warm start".

3 Design Choices

We note that the choice of edge weights and the choice of graph regularizer contribute substantially to algorithmic performance. In this section we present several choices and test the behavior of our method with respect to each. The specific details of the experimental setting are deferred to Section 4.

3.1 Graph Weights

For each pair of nodes i, j, we are provided with the number of links $n_{i,j}$ from i to j (indicating how many links exist from a page in host i to a page in host j). Since our algorithm requires a weighted graph on the set of nodes, we must decide how to choose the weights $\{a_{i,j}\}$. There are several natural choices at hand:

1. Absolute weights: $a_{i,j} \triangleq n_{i,j}$
2. Binary weights: $a_{i,j} \triangleq \mathbf{1}[n_{i,j} > 0]$
3. Square-root weights: $a_{i,j} \triangleq \sqrt{n_{i,j}}$
4. Logarithmic weights: $a_{i,j} \triangleq \log(1 + n_{i,j})$

We tested these choices in the benchmark collection for Web spam described in detail in Section 4. Among these, logarithmic weights tended to give the best performance in terms of AUC (Area Under the ROC Curve). In a different setting, logarithmic weights were also shown to be effective for propagating trust in [30]. Square-root weights had similar performance. Absolute weights were generally a poor choice, as hosts with a vast number of outgoing or incoming links were over-penalized by the regularization. The performance results[1] with different weighting schemes are shown in Table 1.

Table 1 Performance For Different Graph Weights

Weighting method	$a_{i,j}$	**AUC**
Absolute	$n_{i,j}$	0.9524
Binary	$\mathbf{1}[n_{i,j} > 0]$	0.9587
Square root	$\sqrt{n_{i,j}}$	0.9632
Logarithmic	$\log(1 + n_{i,j})$	**0.9646**

3.2 The Graph Regularizer Φ

Since we expect a host's "spaminess" (or similarly, "authenticity") to be preserved locally within the web, the graph regularization function $\Phi(\cdot, \cdot)$ ought to encode how we enforce this locality in our predictions. If node i links to node j, then $\Phi(f_i, f_j)$ should measure how "unnatural" the (f_i, f_j) link pair is.

We have already defined two possible regularization functions:

$$\Phi_{sqr}(f_i, f_j) \triangleq (f_i - f_j)^2$$
$$\Phi^{+}_{sqr}(f_i, f_j) \triangleq \max(0, f_j - f_i)^2 = [f_j - f_i]^2_+$$

The first of these penalizes the square of any deviation between the predicted values of i and j. Note that this penalization is independent of the direction of the link, so a nonspam node will be penalized if it receives a link from a spam node. Naturally we assume that nodes have control only over their out-links and typically not their in-links.

The second function only penalizes the predicted spam scores when the node creating the link has a lower predicted spam value than the link's destination. By employing

[1] These values do not represent the final algorithmic performance since we did not use any model selection to select hyperparameters. As here our goal is only to compare different graph weights, we simply checked performance across a range of parameter choices and, for each weighting scheme, we recorded the best test-set performance. Algorithmic results with model selection can be found in Table 3.

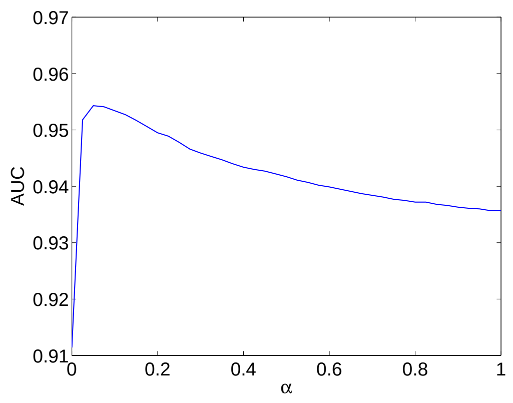

Fig. 1 Performance as a function of the regularization parameter α. When $\alpha = 0$, only the nonspam→spam links are penalized. When $\alpha = 1$ any deviation in the predicted spam value between linked hosts incurs a cost.

Φ^{+}_{sqr}, we are inherently assuming that, while it is perfectly consistent for node spaminess to decrease from the originator of a link to its destination, it is less likely that spaminess would increase.

For the task at hand, the latter choice would appear to be most appropriate. In such a directed network, we certainly expect good nodes to point to good nodes and likewise bad to point to bad. Furthermore, since bad nodes do not want to appear bad and thus can freely insert links to fool a search engine, it is perfectly natural to observe bad nodes pointing to good. On the other hand, we tend to expect that good nodes have no obvious incentive to link to bad nodes, and thus we expect to only rarely observe good-to-bad pairs.

We have also observed this empirically: among the labeled nodes within the web collection described in Section 4, only 1.8% of the out-links from non-spam hosts point to spam hosts, while 14.7% of out-links from spam hosts point non-spam hosts.

Interestingly, we have found that the best choice of regularization is neither of the above but rather a *mixture of the two*. For any $\alpha \in [0, 1]$, define:

$$\begin{aligned}\Phi_{\alpha}(a,b) &\triangleq \alpha\Phi_{sqr}(a,b) + (1-\alpha)\Phi_{sqr+}(a,b) \\ &= \begin{cases}(a-b)^2 & \text{when } a \geq b \\ \alpha(a-b)^2 & \text{when } a < b.\end{cases}\end{aligned}$$

We found this mixed regularization to be extremely effective, and surprisingly a great improvement over either using only Φ_{sqr} or only Φ^{+}_{sqr}. In Figure 1, we plot performance as measured by AUC, as a function of the choice of α. The left side is performance using Φ^{+}_{sqr}, while the right side is using Φ_{sqr}.

The improvement from the mixed regularization appears to be due to the following observation. Relying solely on Φ^{+}_{sqr} to regularize correctly *fails* on nodes that have only a handful of incoming links from spam and/or outgoing links to nonspam hosts, such as the one described in Figure 2. For these nodes, any spaminess score will be unregularized.

For such nodes, whose incoming links are all bad and outgoing links are all nonspam, to obtain a more useful spaminess score, we must rely more on incoming and outgoing

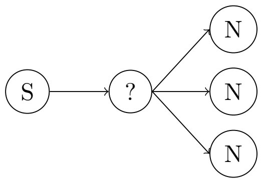

Fig. 2 According to the directed graph regularization Φ^{+}_{sqr}, the node in the middle can have any value. But empirical observations suggest that the node in the middle is more likely to be non-spam than spam. A small amount of undirected regularization can fix this problem.

links regardless of their relative spam values. Thus a small amount of Φ_{sqr} aids in dealing with such special cases. As we see from the plot, $\alpha = 0.1$ is roughly the optimal value, thus 10% of "undirected regularization" is all that we need, and performs much better than purely directed graph regularization Φ^{+}_{sqr}.

For the remainder of the results in the paper, we report performance results using $\Phi_{0.1}$, i.e. where $\alpha = 0.1$, as the regularization function. The value of α can be tuned more carefully, but performance is relatively stable around this value.

4 Experimental Evaluation

We now proceed to compare experimentally the performance of WITCH with several state-of-the-art methods.

4.1 Dataset

We experimented with the `WEBSPAM-UK2006` spam collection [10] for all the results reported in this section. This public collection is the same used in the Web Spam Challenge Tracks I and II [2], which were recent competitions to test Web Spam Detection methods.

The challenge dataset is a graph of 11,402 hosts in the `.uk` domain. Out of these, 7,473 were labeled. The hyperlink graph is represented as a list of 730,774 triples $(\text{node}_i, \text{node}_j, \#\text{links})$ which specify the number of links from host i to host j.

We used a set of 236 features proposed for the challenge including: content-based features such as average word length, number of words in the title, link-based features such as PageRank, number of neighbors, and others proposed in [3] and [26]. Given the variance in scale of these different data types, we chose to normalize the features by replacing each feature value x_{ij} with the fraction of instances having a value of less than x_{ij} for feature j. Due to missing page contents, a subset of available features was missing from 2,458 of the hosts. None of these hosts were part of the test set, but some of them were part of the labeled training set. We have discarded the labels of these hosts and ended up using a training set of 4,363 hosts, which is slightly smaller than the original one containing 5,622 labels. We still kept the hosts with missing features (as unlabeled) and set these feature values to 0.

The training/testing split was fixed and is the same that was used in the Web Spam Challenge Track I (aside from subtracting roughly 1,300 labels from the training set). For all the results we report, the algorithm worked in a transductive[2] setting: the complete graph was given as input, including features for all nodes, with a subset of labelled nodes were given for training. We also cite performance for smaller training sets, which were obtained by sub-sampling the training set. Performance was always measured on the same test set of 1,851 hosts.

WITCH requires the choice of three hyperparameters, $\lambda_1, \lambda_2, \gamma$. We maintained a hold-out set consisting of a random 20% sample of the training data. On a $7 \times 7 \times 7$ grid of parameters, we trained WITCH for each combination and chose the triple $(\lambda_1, \lambda_2, \gamma)$ that returned the best test performance on these data. The final results we report in Table 3 were obtained after validation.

4.2 Optimization Methods

We note that for all experimental results reported herein, for our proposed class of algorithms, the optimization technique we employed was the conjugate gradient method, described in Section 2.3. The optimization procedure also implemented a backtracking step, as described in Algorithm 9.2 in [9], with parameters $\alpha = 0.2, \beta = 0.5$. The final optimization was terminated once the decrement in the objective function dropped below a threshold of 10^{-6}.

4.3 Performance Metric

To measure the performance of each method, we chose the metric Area Under the ROC curve (AUC). AUC provides a natural measure of accuracy of a predicted ranking, and requires only that the algorithm outputs an ordering of the test set.

Others have utilized the F-Measure, which combines precision and recall, to determine performance. The drawback of F-Measure is that it requires a fixed choice of classification threshold, and the result can be very sensitive to the choice of threshold. Also, because the training and test sets have not been labeled by the same people, the fraction of spam hosts in them is different. This means that the threshold learned by various classifiers can be very different from the one which would yield the best F-Measure on the test set.

Finally, for practical applications on Web search, a real-valued “spaminess” prediction is more useful than a binary prediction, given that the spaminess will be combined with other features to give the final ranking of the search results. Also, its particular weight may be different on different search contexts.

4.4 Comparison with Variant Algorithms

Recall that WITCH takes advantage of three primary tools in training: host features, slack variables, and regularization along the hyperlink graph. Each of these elements plays a different role in the algorithm and contributes to performance at varying levels. To see the relative importance of each, we now consider alternative approaches that involve various subsets of the proposed techniques.

[2] Transduction is a learning paradigm where the test set is known at training time [29].

1. **Only Features.** We train a linear classifier on the given feature space with no graph regularization. The algorithm is effectively a Support Vector Machine except that we use squared hinge-loss. The final label on node i is given by $\mathbf{w} \cdot \mathbf{x}_i$, where $\mathbf{w}$ is the minimum of the objective

$$\Omega(\mathbf{w}) \;=\; \frac{1}{l}\sum_{i=1}^{l}[1 - y_i \mathbf{w} \cdot \mathbf{x}_i]_+^2 + \lambda \mathbf{w} \cdot \mathbf{w}.$$

2. **Features + Graph Regularization (GR).** We train a linear classifier on the provided feature space but we additionally regularize according to the hyperlink graph structure. As above, we label node i with $\mathbf{w} \cdot \mathbf{x}_i$, where $\mathbf{w}$ minimizes the objective

$$\Omega(\mathbf{w}) \;=\; \frac{1}{l}\sum_{i=1}^{l}[1 - y_i \mathbf{w} \cdot \mathbf{x}_i]_+^2 + \lambda \mathbf{w} \cdot \mathbf{w} + \gamma \sum_{(i,j)\in E} a_{ij} \Phi_{0.1}(\mathbf{w} \cdot \mathbf{x}_i, \mathbf{w} \cdot \mathbf{x}_j).$$

3. **Slack Variables + GR.** We ignore features and directly learn the label using graph regularization. Here, we learn node i's label directly as z_i, where the vector $\mathbf{z}$ is found by minimizing

$$\Omega(\mathbf{z}) \;=\; \frac{1}{l}\sum_{i=1}^{l}[1 - y_i z_i]_+^2 + \lambda \mathbf{z} \cdot \mathbf{z} + \gamma \sum_{(i,j)\in E} a_{ij} \Phi_{0.1}(z_i, z_j).$$

   Note that this method belongs to the same family as TrustRank [19] and Anti-Trust Rank [24] algorithms. Indeed, these algorithms do not use feature vectors and are only based on the hyperlink graph. Both algorithms also exploit the fact that normal hosts rarely link to spam hosts.
4. **Features + Slack + GR** (WITCH). We now utilize all tools available. We simultaneously train a linear classifier and slack variables, and we regularize the predicted values along the graph. The predicted label of node i is $\mathbf{w}\cdot\mathbf{x}_i + z_i$, where the vectors $\mathbf{w}$ and $\mathbf{z}$ are found by minimizing

$$\begin{aligned}\Omega(\mathbf{w}, \mathbf{z}) \;=\; & \frac{1}{l}\sum_{i=1}^{l}[1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + z_i)]_+^2 + \lambda_1 \mathbf{w} \cdot \mathbf{w} + \lambda_2 \mathbf{z} \cdot \mathbf{z} \\ & + \gamma \sum_{(i,j)\in E} a_{ij} \Phi_{0.1}(\mathbf{w} \cdot \mathbf{x}_i + z_i, \mathbf{w} \cdot \mathbf{x}_j + z_j). \qquad (11)\end{aligned}$$

We observe that, technically, this list is not exhaustive. However, we see that upon closer inspection the remaining combinations do not produce any new methods. In particular, the following Lemma shows that using slack variables without graph regularization is equivalent to method 1 above but with a larger value of λ. Given that we have optimized λ by cross-validation in setting 1, this implies that we cannot get a better result in setting 4 when $\gamma = 0$ than in setting 1. In short, "Features + Slack" is not in fact an alternative when the squared hinge loss is used. This is stated explicitly in the following lemma, whose proof we postpone to the appendix.

Lemma 1 *The solution of* (11) *when we set* $\gamma = 0$ *is the same as that obtained in objective 1 when we set* $\lambda = \lambda_1 \left(\frac{1}{l\lambda_2} + 1\right)$.

In Table 3 we report performance for each of the above four methods. The most stunning result appears to be the effect of the slack variables, which provided a significant boost in performance. Training only the slack variables and ignoring features entirely significantly outperforms solely feature-based methods. In addition, utilizing the features in combination with slack provided another reasonable benefit.

To see how performance depends on subset size, in Figure 3 we also compare each of the above algorithms for seven different training-set sizes. To obtain these AUC values, we trained a classifier on a large parameter grid for each of the $7 \times 4 = 28$ subset-size$\times$method pairs. After finding the best parameter settings for each subset-size$\times$method, we retrained 10 classifiers with different training sub-samples of the same size and took the median AUC performance of these. This was to reduce variance which was particularly high for the smaller subsets.

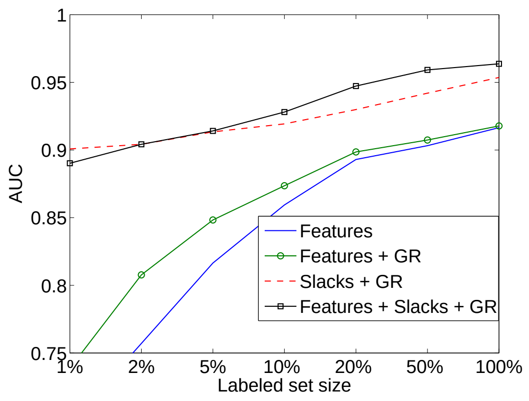

Fig. 3 Performance of WITCH, and variants, as described in Section 4.4. Experiments have been repeated 10 times with random choices of the subsets. The median result is plotted.

The improvement coming from the slack variables can be interpreted as an underfitting problem: there apparently exists no **w** which results in a good spam detector. We can think of three main reasons for this. First, probably the class of linear functions is too restrictive for this classification problem, particularly when the features may have been adversarially generated with the purpose of appearing authentic. Second, the labeling task is quite subjective and there is often disagreement among human editors [10], and thus the labels are quite noisy. Third, the feature set may not be rich enough. The additional slack variables introduce enough degrees of freedom to accurately model a spam classifier under these circumstances.

4.5 Comparison with Web Spam Challenge Results

The Track I of the Web Spam Challenge, in which we did not take part, ended in April of 2007. The best performance in terms of AUC[3] for this track was obtained by Gordon

[3] This track in fact had several winners, as the competition consider F-Measure as well as AUC for a performance metric.

Cormack with 0.956. The method he implemented used a stack of ten classifiers [2]; we note, however, that this participant used some specially designed classifiers that utilized features not provided by the competition.

On the same dataset, WITCH outperforms all submissions to the first track of the challenge, obtaining an AUC performance of 0.963[4]. Surprisingly, we achieved this result despite using a smaller training set in our experiments (see section 4.1).

The challenge included a Track II that ended in July 2007, and for this track we submitted predictions using the methods discussed herein. WITCH obtained the highest AUC against the 10 other submissions [1]. The data in Track II was generated from the data in the first track, but a different numeric feature set was provided, a new training/test set split was made, and no external information (such as the web page contents or the address of the web host) was available.

4.6 Comparison with Stacked Graphical Learning

Stacked graphical learning is a meta-learning scheme described recently by Cohen and Kou [13]. It uses a base learning scheme $\mathcal{C}$ to derive initial predictions for all the objects in the dataset. Then it generates a set of extra features for each object, by combining the predictions for the related objects in the graph by an aggregate function (in our case, we averaged the prediction for all the linked hosts disregarding direction). Finally, it adds this extra feature to the input of $\mathcal{C}$, and runs the algorithm again to get new predictions for the data.

This learning scheme was shown to be effective when applied to the Web Spam Detection task [11]. Results reported there were obtained by cross-validation on the training set, and here we repeat the same experiments using the training/testing split proposed in the Web Spam Challenge.

The algorithm begins by training a base classifier, bagging of ten C4.5 decision trees. We then apply two iterations of stacked graphical learning (more iterations do not improve the performance). The final results are shown in Table 2 and correspond to the classifier in [11]. We also provide results setting the base classifier to be a standard SVM, and this achieved somewhat better performance than a decision tree classifier.

Table 2 Results with Stacked Graphical Learning

Classifier	**AUC**
Decision trees	0.900
1-step s.g.l.	0.934
2-step s.g.l.	0.935
SVM	0.923
1-step s.g.l.	0.946
5-step s.g.l.	0.953

As reported previously, stacked graphical learning significantly improves the performance of both the bagged decision tree algorithm and the standard SVM, yet still underperforms the techniques we propose. We believe the gap is likely due to the "local" nature of SGL, whereas WITCH utilizes the entire graph in training.

[4] The hyperparameters $\lambda_1, \lambda_2, \gamma$ were chosen on a validation set as we describe in Section 4.1.

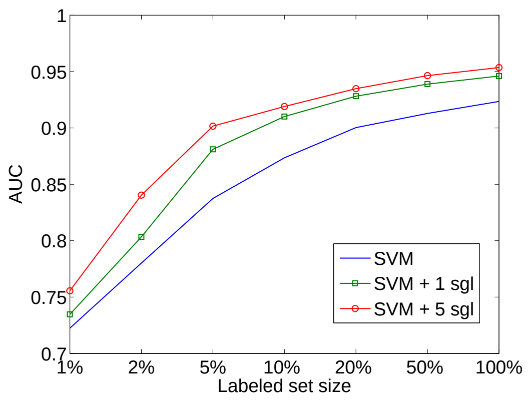

Fig. 4 Results of stacked SVM for different sizes of the training set. Experiments have been repeated 10 times with randomly chosen subsets of the training set. The median of these 10 results is plotted.

Finally, Figure 4 shows the AUC obtained by an SVM with stacked graphical learning for different training set sizes.

4.7 Comparison with Link-Based Methods

We compared our algorithm to Transductive Link Spam Detection, proposed in [32], which uses only hyperlinks and not content-based features. This methods outperforms other well-known graph-based methods based on label propagation such as TrustRank [19] and Anti-Trust Rank [24].

A technical requirement of this algorithm is that the hyperlink graph must be strongly connected, which is generally not the case in the real world. To handle this problem we create a dummy node with bidirectional links to every node in the original graph. In order that this additional node does not contribute substantially to the final solution, we set the weight associated of each new edge to a small value, 10^{-6}. The algorithm also requires the choice of a parameter α, which we selected on a validation set. Results are presented in Figure 5.

4.8 Summary of Experimental Results

Table 3 summarizes the performances of each method discussed in this paper. The particular design choices used to obtain these results are as follows:

1. The function Φ was the mixed regularizer described at the end of Section 2. As mentioned, the best parameter choice here was $\alpha = 0.1$, used througout.
2. The various hyperparameters were chosen by doing a grid search tested on a hold-out set. This is described in greater detail at the end of Section 4.1.
3. All optimizations were done using the conjugate gradient method, described in Section 4.2.

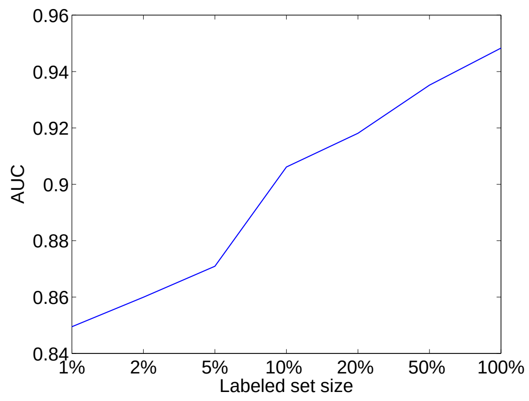

Fig. 5 AUC obtained by the method described in [32] for various sizes of the training set, taking the median of the same samples as in Figure 4.

We emphasize that, in terms of a AUC, an improvement from 0.95 to 0.96 is quite significant and can be interpreted roughly as a 20% reduction in ranking error. The results in Table 3 suggest that this boost is indeed due to the incorporation of both the hyperlink structure as well as the page contents. This observation agrees with a natural intuition, namely that it is better to leverage both types of data to accurately judge spaminess.

Table 3 Results for all methods with two different sizes of the training set. The first group of methods was discussed in Section 4 and the second group of methods, which we propose, are described in Section 3.

Training Algorithm	**AUC 10%**	**AUC 100%**
SVM + stacked g.l.	0.919	0.953
Link based (no features)	0.906	0.948
Challenge winner	–	0.956
Only Features	0.859	0.917
Features + GR	0.874	0.917
Slack + GR	0.919	0.954
WITCH (Feat. + Slack + GR)	**0.928**	**0.963**

5 Conclusions

In this paper we have presented a novel algorithm, WITCH, for the task of detecting Web spam. We have compared WITCH to several proposed algorithms and we have found that it outperforms all such techniques. Finally, WITCH obtained the highest AUC performance score on an independent Web spam detection challenge.

We attribute these positive results to a few key observations. First, best results are achieved when both content features *and* the hyperlink structure are used. Second, simply training a graph-regularized linear predictor is insufficient, as the addition of slack variables provides a very significant improvement. Third, one needs to choose the right graph regularization function, as we have observed that penalizing both spam→nonspam links and nonspam→spam links is important, yet the tradeoff should be much heavier on the latter. Lastly, we have observed that the form of the graph weights contributes significantly to performance, where using the logarithm of the number of links worked best.

For any machine learning technique to have practical impact in a Web search setting, it must also be effective when the training set is small, since labeling is expensive compared to the number of hosts on the web. To the best of our knowledge, this is the first time this issue has been addressed in a Web spam detection system. Our result show that our technique works better than a classifier trained using stacked graphical learning, and in particular when there is little training data available.

Scalability is another issue we have addressed and we argue in Section 2.3 that this method can scale up to very large graphs. To further improve speed, a training based on stochastic gradient descent [8] would be yet another alternative.

For future work, the method we have presented could be applied to other tasks on the Web, such as topical classification [31].

References

1. Graph Labeling Workshop. http://graphlab.lip6.fr/, 2007.
2. Web Spam Challenge. http://webspam.lip6.fr/, 2007.
3. L. Becchetti, C. Castillo, D. Donato, S. Leonardi, and R. Baeza-Yates. Link-based characterization and detection of Web Spam. In *Second International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, Seattle, USA, August 2006.
4. M. Belkin, I. Matveeva, and P. Niyogi. Regularization and semi-supervised learning on large graphs. *Lecture Notes in Computer Science*, 3120:624–638, 2004.
5. M. Belkin, P. Niyogi, and V. Sindhwani. On manifold regularization. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2005.
6. A. Benczúr, K. Csalogány, and T. Sarlós. Link-based similarity search to fight web spam. In *Adversarial Information Retrieval on the Web (AIRWEB)*, Seattle, Washington, USA, 2006.
7. A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, pages 92–100, 1998.
8. L. Bottou. Stochastic learning. In *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, pages 146–168. Springer Verlag, 2004.
9. S. P. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
10. C. Castillo, D. Donato, L. Becchetti, P. Boldi, S. Leonardi, M. Santini, and S. Vigna. A reference collection for web spam. *SIGIR Forum*, 40(2):11–24, December 2006.
11. C. Castillo, D. Donato, A. Gionis, V. Murdock, and F. Silvestri. Know your neighbors: Web spam detection using the web topology. In *Proceedings of SIGIR*, Amsterdam, Netherlands, July 2007. ACM.
12. O. Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19(5), 2006.
13. W. W. Cohen and Z. Kou. Stacked graphical learning: approximating learning in markov random fields using very short inhomogeneous markov chains. Technical report, 2006.
14. B. D. Davison. Topical locality in the web. In *Proceedings of the 23rd annual international ACM SIGIR conference on research and development in information retrieval*, pages 272–279, Athens, Greece, 2000. ACM Press.
15. D. Fetterly. Adversarial information retrieval: The manipulation of web content. *ACM Computing Reviews*, July 2007.
16. D. Fetterly, M. Manasse, and M. Najork. Spam, damn spam, and statistics: Using statistical analysis to locate spam web pages. In *Proceedings of the seventh workshop on the Web and databases (WebDB)*, pages 1–6, Paris, France, June 2004.

17. Q. Gan and T. Suel. Improving web spam classifiers using link structure. In *AIRWeb '07: Proceedings of the 3rd international workshop on Adversarial information retrieval on the web*, pages 17–20, New York, NY, USA, 2007. ACM.
18. Z. Gyöngyi and H. Garcia-Molina. Web spam taxonomy. In *First International Workshop on Adversarial Information Retrieval on the Web*, pages 39–47, Chiba, Japan, 2005.
19. Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating Web spam with TrustRank. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 576–587, Toronto, Canada, August 2004. Morgan Kaufmann.
20. S. W. Haas and E. S. Grams. Page and link classifications: connecting diverse resources. In *DL '98: Proceedings of the third ACM conference on Digital libraries*, pages 99–107, New York, NY, USA, 1998. ACM Press.
21. M. R. Henzinger, R. Motwani, and C. Silverstein. Challenges in web search engines. *SIGIR Forum*, 36(2):11–22, 2002.
22. A. Joshi, R. Kumar, B. Reed, and A. Tomkins. Anchor-based proximity measures. In *WWW*, pages 1131–1132, 2007.
23. P. Kolari, A. Java, T. Finin, T. Oates, and A. Joshi. Detecting spam blogs: A machine learning approach. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Boston, MA, USA, July 2006.
24. V. Krishnan and R. Raj. Web spam detection with anti-trust rank. In *ACM SIGIR workshop on Adversarial Information Retrieval on the Web*, 2006.
25. G. Mishne, D. Carmel, and R. Lempel. Blocking blog spam with language model disagreement. In *Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, Chiba, Japan, May 2005.
26. A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting spam web pages through content analysis. In *Proceedings of the World Wide Web conference*, pages 83–92, Edinburgh, Scotland, May 2006.
27. J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, 1994.
28. T. Urvoy, T. Lavergne, and P. Filoche. Tracking web spam with hidden style similarity. In *Second International Workshop on Adversarial Information Retrieval on the Web*, Seattle, Washington, USA, August 2006.
29. V. Vapnik. *Statistical Learning Theory.* John Wiley & Sons Inc, 1998.
30. B. Wu, V. Goel, and B. D. Davison. Propagating trust and distrust to demote web spam. In *Workshop on Models of Trust for the Web*, Edinburgh, Scotland, May 2006.
31. T. Zhang, A. Popescul, and B. Dom. Linear prediction models with graph regularization for web-page categorization. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 821–826, New York, NY, USA, 2006. ACM Press.
32. D. Zhou, C. J. C. Burges, and T. Tao. Transductive link spam detection. In *AIRWeb '07: Proceedings of the 3rd international workshop on Adversarial information retrieval on the web*, pages 21–28, New York, NY, USA, 2007. ACM Press.

A Proof of Lemma 1

Let us first compute:

$$\min_{z_i} \frac{1}{l}[1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + z_i)]_+^2 + \lambda_2 z_i^2. \tag{12}$$

Setting the derivative with respect to z_i at 0, one finds that the minimum is reached for

$$z_i = \frac{y_i[1 - y_i(\mathbf{w}\cdot\mathbf{x}_i)]_+}{1 + l\lambda_2}.$$

Plugging this value in (12), we have:

$$\min_{z_i} \frac{1}{l}[1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + z_i)]_+^2 + \lambda_2 z_i^2 = \frac{\lambda_2}{1 + l\lambda_2}[1 - y_i(\mathbf{w}\cdot\mathbf{x}_i)]_+^2.$$

From here it is easy to see the equivalence between setting 1 and 4 when $\gamma = 0$:

$$\min_{\mathbf{z}} \frac{1}{l}\sum_{i=1}^{l}[1-y_i(\mathbf{w}\cdot\mathbf{x}_i+z_i)]_+^2+\lambda_1\mathbf{w}\cdot\mathbf{w}+\lambda_2\mathbf{z}\cdot\mathbf{z} = \frac{l\lambda_2}{1+l\lambda_2}\left(\frac{1}{l}\sum_{i=1}^{l}[1-y_i\mathbf{w}\cdot\mathbf{x}_i]_+^2+\lambda\mathbf{w}\cdot\mathbf{w}\right),$$

where λ is defined such as

$$\lambda_1 = \frac{l\lambda_2}{1+l\lambda_2}\lambda,$$

that is,

$$\lambda \triangleq \lambda_1\left(\frac{1}{l\lambda_2}+1\right).$$

More generally, instead of the squared hinge loss, let us consider a general loss function ℓ. Then it is not difficult to see that the solution of setting 4 with $\gamma = 0$ is the same as the one of setting 1 where the loss is replaced by its *infimal convolution* with the squared function $s(t) = \lambda_2 t^2$:

$$\tilde{\ell}(x) = (\ell \star s)(x) = \inf_t f(x+t) + s(t).$$

The squared hinge loss is particular case because its infimal convolution with the squared function yields another squared hinge loss.