

Feature Weighting for Improved Classifier Robustness

Aleksander Kolcz
Microsoft Live Labs
One Microsoft Way
Redmond, WA USA
ark@microsoft.com

Choon Hui Teo
Australian National University & NICTA
Canberra, ACT 0200
Australia
choonhui.teo@anu.edu.au

ABSTRACT

There are often discrepancies between the learning sample and the evaluation environment, be it natural or adversarial. It is therefore desirable that classifiers are robust, i.e., not very sensitive to changes in data distribution. In this paper, we introduce a new methodology to measure the lower bound of classifier robustness under adversarial attack and show that simple averaged classifiers can improve classifier robustness significantly. In addition, we propose a new feature reweighting technique that ameliorates the performance and robustness of standard classifiers at at most twice the computational cost. We verify our claims in content based email spam classification experiments on some public and private datasets.

1. INTRODUCTION

In many practical applications of machine learning, there will be some discrepancy of between the nature of the data used to build a classification model and the nature of the data the model is evaluated on. This may cause the learned models underperform, sometimes very dramatically so [10]. Depending on the circumstances and causes of this discrepancy, as well as its magnitude, researchers have been addressing this problem under different guises. And so classifiers are desired to be robust to small changes of the data and they should not be overfitting the particulars of the training set.

Often the data distribution changes occur naturally over time. For example, the topicality of Web news stories or blog posts evolves as new/seasonal events come into play. Similarly, emails or Instant Messages will reflect various seasonal trends and reflect current events. Importantly, changes in content distribution may also exhibit an adversarial angle, whereby attackers seek to compromise the effectiveness of a classifier (e.g., as in filtering email spam). While it is desired that classifiers continually adjust to such changes in content distribution, this is not always possible, e.g., due to delays in feedback or other practical system concerns. It is therefore desired that classifiers created with the initial training data *degrade gracefully* as the distribution of the evaluation data diverges from the distribution of the original training data.

One of the reasons behind the lack of robustness is feature under or overtraining. Thus a learner may overempha-

size the apparently highly relevant features while ignoring ones that are less informative, even if the latter ones collectively carry equivalent information about the classification problem. In this work we explore an alternative method based of feature weighting. Motivated by the success of term weighting techniques in text classification (where they tend to outperform feature selection [9]) we investigate to what an extent document representation has an impact on classification accuracy in email filtering domain.

In section 2 we review common document representation techniques. We then describe our feature weighting approach in Section 3 and the related approaches in Section 4. The experimental results and analyses are presented in Section 5. Finally, we conclude our work and describe some potential future works in Section 6.

2. DOCUMENT REPRESENTATION IN TEXT CLASSIFICATION

2.1 Supervised and Unsupervised Term Weighting

In text classification problems, a document is commonly represented by a finite high dimensional feature vector where the features can be, for example, the frequencies of all possible n -grams, words, and phrases observed in the document. Due to the nature of natural language, sensible documents usually result in feature vectors with high sparsity, i.e., any single document contains only a very small fraction of all possible words. For small learning samples, large dimensionality of feature space may lead to overfitting and traditionally researchers have been applying some form of feature selection prior to building a model. Recent results indicate, however, that hard removal of features caused by feature selection is sometimes undesirable and better results can be obtained by simply downweighting the less important features [13, 9]. Feature weighting has a long history in Information Retrieval, where various forms of TFxIDF schemes have been proposed to better capture the relevance of a document to a search query. In such formulations, for each unique feature occurring in a document, its weight is proportional to a function of the frequency of the term in the current document and inversely proportional to the overall frequency of that term in the training corpus, e.g.,

$$\text{TFxIDF}_i = tf_i \cdot \ln \left(\frac{N}{df_i} \right) \quad (1)$$

where tf_i is the number of occurrences of term i in the current document, df_i is the number of documents con-

taining term i and N is the total number of training documents. Eqn. (1) represents just one of many different variants of TFxIDF weighting. IR-inspired weighting schemes have been widely adopted in text classification. They can be thought of as unsupervised, since the weight for a term depends only on the current document and the overall statistics of the training collection. The supervised schemes, on the other hand, take advantage of the fact that in classification problems, the class label can provide additional information regarding the usefulness of a term to the problem at hand [9]. Nevertheless, so far no single “best” weighting scheme emerged, although several good choices for supervised and unsupervised methods have been suggested.

2.2 Document Length Normalization

In IR it has also been common to normalize document feature vectors to unit L2 norm. Since documents can vary significantly in length, unnormalized similarity-based ranking would give preference to longer documents. Under the L2 normalization scheme, the cosine-similarity measure used in vector space models is equivalent to a dot product between feature vectors. This type of transformation has also been adopted in text classification as it tends to improve the performance measures. Other types of length normalization have also been considered for text classifiers, since learners are not necessarily tied to the cosine similarity. For example, length normalization according to the L1 norm has shown to be improve over the L2 scheme for some learners [13]. The combination of term weighting and document length normalization can make a prominent difference for some classifiers, such as Naive Bayes [14, 13].

2.3 Document Representation and Classification Robustness

Feature weighting is typically expected to assign higher weights to more important/informative features, as estimated from the training set. But, in principle, it could be arbitrary and reflect, for example, some form of prior knowledge. In this work we consider the question to what an extent inverse importance weighting makes sense in the context of improving classifier robustness. Our intuition is as follows: the weights assigned to features by a learning algorithm correspond to the importance of these features from that classifier’s perspective. In a supervised term-weighting scheme, the model weights assigned by a classifier using no term weighting can provide term weights for another classifier of the same type. Thus to the extent that the classifier is under or overemphasizing some features, this is reflected in how the weights are distributed. If we assume that features with higher weight magnitudes could be hampering robustness, downweighting them in favor of features with lower weights appears to be a natural countermeasure. Conversely, increasing the weights of features de-emphasized by a classifier might force it to use them more often. All in all, one can envision a two-stage classifier induction process where the initial classifier is used to derive feature weights, which are then transformed and the final model is induced over the transformed feature weighting.

3. PROBLEM STATEMENT

We assume that a given training set $D = \{(x_i, y_i)\}_{i=1}^m \subset \mathbb{R}^d \times \{\pm 1\}$ is drawn from an unknown distribution, where x is a feature vector and y the corresponding class label.

As motivated by the discussion in Section 2.2, we further assume that x be normalized to unit length in L2 norm. The classifier of interest is a linear decision function

$$f(x) = \text{sign}(w^T x + b) \quad (2)$$

parameterized by a weight vector $w \in \mathbb{R}^d$ and a bias term $b \in \mathbb{R}$ obtained via standard modeling techniques such as the Support Vector Machines (SVM) or regularized logistic regression (LR).¹

Our goal is to train a *robust* classifier such that its evaluation performance (e.g., area under ROC curve), while maintained at reasonable level, degrades at a lower rate than a standard classifier as the underlying distribution of the evaluation data gradually diverges away from that of training data. In our problem setting, the changes in the distribution of evaluation data can be characterized by the increment/decrement of feature values. When binary feature values are used, this boils down to addition/deletion of features. It is then natural to expect that the weight vector of a robust classifier should be distributed as evenly as possible.

3.1 Feature Reweighting

Our approach to overcoming feature under and overtraining is by reweighting the features inversely proportional to their corresponding importances. These importances are derived from the weight vector w of an initial classifier trained on the training set D . For example, we may rescale i -th training feature vector x_i into another \bar{x}_i as follows:

$$\bar{x}_{ij} := x_{ij}/s(w_j), \quad j = 1, \dots, d \quad (3)$$

where w is the weight vector, $s: \mathbb{R} \mapsto (0, \infty)$ is some positive monotonically increasing function (e.g. $s(u) = \log(e + |u|)$), and x_{ij} is the j -th feature of x_i . Clearly, the length of the rescaled feature vectors \bar{x} may be arbitrary long, which is not desirable. To remedy this side effect of feature reweighting, one could either renormalize \bar{x} or the inverse of scaling vector $s(w)$ (such that $\|1/s(w)\| = 1$). The latter renormalization scheme guarantees that $\|\bar{x}\| \leq 1$ by the Cauchy-Schwarz inequality provided the original feature vector x is already normalized. With the rescaled training set $\bar{D} = \{(\bar{x}_i, y_i)\}_{i=1}^m$ we train a final classifier and obtain the model weight vector \bar{w} .

In fact, the feature reweighting scheme can be seen as a particular case of regularized risk minimization (RRM) [18] with a quadratic form regularizer characterized by the scaling $s(w)$: Let S be a diagonal matrix with entries $S_{ii} = s(w_i)$ and $S_{ij} = 0$ for $i \neq j$. We can rewrite the feature weighting stated in Eqn. (3) as $S^{-1}x$. The two-stage training cast under the RRM framework is then a convex optimization problem:

$$\min_w \frac{\lambda}{2} w^T w + \frac{1}{m} \sum_{i=1}^m l(w^T (S^{-1}x), y) \quad (4)$$

$$\equiv \min_{\bar{w}} \frac{\lambda}{2} \bar{w}^T S S \bar{w} + \frac{1}{m} \sum_{i=1}^m l(\bar{w}^T S S^{-1} x, y) \quad (5)$$

where $\lambda > 0$ is regularization constant, and l is a convex loss function such as the hinge loss

$$l(f, y) = \max(0, 1 - yf) \quad (6)$$

¹In practice, the weight vector w is optimized while the bias term b is tuned on validation set.

Method	Complexity	References
FB	$O(Kmd)$	[15]
PLR	$O(Kmd)$	[3]
CWL	$O(md)$	[7]
FDROP	$O(md \log d)$	[10, 16]
FMICO	$O(md)$	[6]
Feature Reweighting	$O(md)$	Section 3.1

Table 1: Per iteration time complexities of various iterative robust classifier training algorithms; note that these methods can be run in an online fashion. m is the size of training data, d is the number of features, and K is the number of base models.

for SVMs or the logistic loss

$$l(f, y) = \log(1 + \exp(-yf)) \quad (7)$$

for regularized logistic regression. Since S is strictly positive its inverse S^{-1} exists and hence SS^{-1} is an identity matrix. This implies that the original feature vectors need not be reweighted under this framework.

4. RELATED WORK

The potential of machine learning algorithms to overfit certain features has been known for a long time, even in anecdotal form. One of the tasks of data preparation and cleaning, for example, is to prevent “perfect” artifact features (e.g., inventory tags correlated with the class label) entering the data used for training. Apart from such human errors, however, it is fairly common that the data used to induce a classifier does not truly reflect the statistics of the domain being modeled. As a result, the apparent high usefulness of certain features or combinations thereof may not be reflected in practice. In the sequel, we describe some of the state of the art classifier training algorithms which aim at improving classifier robustness or overcoming the differences in training and evaluation data.

4.1 Feature Bagging

In [15] this problem was described as feature undertraining. I.e., certain features are ignored or receive relatively low importance in favor of other more informative features. Given this high reliance on the highly informative features, the classifier is going to perform poorly if these features are not present during testing or if their prevalence is much lower than expected. The authors proposed *feature bagging* (FB), i.e., building a probabilistic model as an arithmetic or geometric mean of several base models with each focusing on a possibly overlapping subset of the original feature set. One possible reason behind the success of this method is that by prior knowledge or chance (due to randomness) in feature subset selection, highly indicative features are assigned to bags different from that of the less indicative features. Therefore, weights of the less indicative features will not be overwhelmed by the highly indicative ones during the modelling process.

This method is computationally disadvantageous compared to conventional training algorithm as it involves training of multiple base classifiers. Furthermore, the optimal number-of base classifiers and feature subset size are not known in advance, hence one may need to spend extra for tuning such hyper-parameters.

4.2 Partitioned Logistic Regression

The authors of [3] introduced the *partitioned logistic regression* (PLR) which can be seen as a specialization of the feature bagging approach with stronger assumptions that feature bags are independent conditioned on the class label and are non-overlapping. The base models are trained independently similar to the feature bagging. Unlike feature bagging, the trained model parameters are concatenated into one as a result of the non-overlapping feature bags assumption.

4.3 Confidence-Weighted Learning

Arguably, the problem of feature undertraining has also motivated the development of *confidence-weighted learning* (CWL) in [7]. The authors observed that rare features in the training data are bound to be undertrained. So this learning algorithm updates the weights for those features more aggressively than the frequent features. In particular, this approach models the uncertainty of feature weights by maintaining a normal distribution over the weight vector of a linear classifier. The mean represents the “average” weight vector, whereas the standard deviation captures the “uncertainty” and correlation of the feature weights. It is this uncertainty measure that determines how aggressively the weight for each feature should be updated.

The algorithm is initialized with user-provided parameters (i.e., the mean and the covariance matrix). In an online fashion, a new training instance is presented and the parameters are updated such that the Kullback-Leibler divergence between the current (Gaussian) distribution and the new distribution is minimized, while having a probability of correctly classifying the training instance larger than a user-specified threshold. The algorithm requires only $O(md)$ operations for one epoch of training when the covariance matrix is restricted to a diagonal matrix; the setting has been proven to be very efficient and effective in the NLP tasks presented in [7]. In addition, this approach can be parallelized by splitting the original training data set into several non-overlapping data sets and the final weight vector can be taken as the average of those trained in parallel in a principled way.

4.4 Feature Noise Injection

Feature noise injection is an approach to alleviate the problem of feature overtraining by introducing artificial feature noise during model induction. The types of feature noise varies according to the nature of the problem domains. For example, in the cases where features are measurements from a sensor networks, failure in sensor nodes would results in come feature values being zero. In other cases, it may be that the feature values are corrupted e.g., imprecise measurements due to environmental conditions.

In [10] the authors proposed to account for model sensitivity to feature deletion during model induction in a worst case fashion. In the proposed algorithm (FDROP), every training feature vector is subject to at most N feature deletions such that the training loss entailed by the resulting SVM model is as high as possible. The original algorithm involves a quadratic programming problem with $O(md)$ variables, which hinders the applicability of the algorithm to high dimensional problems such as spam filtering or text classification. [16] extended this algorithm to a more general feature noise setting and introduced an iterative algorithm

which solves the same problem in $O(\epsilon^{-1}md \log d)$ time and requires only $O(\epsilon^{-1}d)$ additional space, where ϵ (usually in the order of 10^{-3}) is the difference between the optimal and the final objective values.

We also note that in [6], the authors proposed a similar method (FMICO) which allows one to assign importance to features *a priori* to countermeasure feature over or under-training. In addition, the authors suggested the use of L_∞ norm regularization to produce a denser weight vector such that the classifier could withstand feature deletion or corruption better at evaluation time. The authors introduced a linear programming based algorithm, which has $O(m)$ variables and does not scale to large training sets, and an online algorithm which is much more efficient and was shown to be performing equally well.

4.5 Sample Selection Bias Correction

Another line of related work is *sample selection bias correction* whereby a correcting weight is assigned to each training instance such that the reweighted training data resembles those drawn from the underlying distribution of evaluation data. The state of the art methods such as those proposed in [11, 2] infer the correcting weight without the need of explicit density estimation. Note that this approach is different from the previously discussed approaches as the evaluation data is assume to be present and accessible during the model induction. In this sense, sample bias correction is positioned closer to domain adaptation.

5. EXPERIMENTS

Spam filtering is an important application where classifier robustness is of particular importance. The content of legitimate email messages (i.e., *ham*) naturally changes over time, following seasonality patterns, recent events, etc.. Additionally, spammers constantly adjust the content of their unsolicited messages so as to avoid detection. As a result, it is difficult to obtain a large labeled sample of emails that will reflect the true distribution of ham and spam, which a deployed filter will actually encounter. Also, due the adversarial nature of spam, spammers will attack the weaknesses of the particular deployed filter, which are not fully apparent until the filter becomes active. One cannot therefore perform a fully realistic lab experiment, although useful approximations are possible.

Commonly, researchers use a time split sample of emails captured from the same source (e.g., user of a particular institutions, an ISP, etc.). The earlier part of the data is used for training, while the later one is used for evaluation. This type of a setup allows one to measure the robustness against the natural variability of email. It may, however, lead to an overestimation of the spam detection performance, since any adversarial changes in the spam distribution was in response to filters active in the system where the data was collected and not in response to the filters being considered as part of the experimental procedure. This experimental setup should therefore be seen as providing an upper bound estimate of the actual filter performance.

To better assess the classifier robustness when faced with an adversarial attack, we also consider an alternative evaluation setup, one where the test instances are modified such as to make them more difficult to be classified correctly by a particular model. When restricted to the case of linear classifiers this is accomplished by removing the *K most* in-

formative features of spam class for each spam instance or by injecting each spam instance with *K less* informative features of the ham class not observed in the instance.² When performed for each spam instance of the evaluation set, this leads to an estimate of the worst possible degradation of classifier performance when a spammer is restricted to *K* feature additions/deletions per instance. Given that spammers have only limited knowledge of the underlying filter and not all spam received by a system is adversarial, this is likely to underestimate the filter effectiveness, but it provides a lower bound estimate of the practical performance.

In addition to adversarial attacks, we will also evaluate the classifier robustness on time split evaluation sets. In this realistic setting, the robustness of a classifier refers to the ability to withstand the (slight) changes in the evaluation environment over time.

5.1 Datasets

In the experiments, we used the spam datasets from TREC 2005 [5], TREC 2006 (English set) [4], ECML/PKDD 2006 Discovery Challenge (Task A evaluation set), and Hotmail. The first three datasets are publicly available while the last one is proprietary. Details of the datasets are summarized in Table 2.

Dataset	d	#tr	#va	#ev	Ref
TREC05	208,844	24,582	6,145	61,455	[5]
TREC06	133,495	10,086	2,521	25,241	[4]
ECML06	206,908	3,200	800	7,500	[1]
Hotmail	2,644,921	688,500	76,500	150,000	[13]

Table 2: Details of datasets used in experiments. Second column indicates the number of features. Third through fifth columns indicate the numbers of training, validation, and evaluation instances, respectively. The last column indicates the references for the datasets.

5.2 Methods

In training content based spam classifier, one normally resort to linear classifier induction methods such as LR and SVM. These methods will serve as the baselines in our experiments. In addition, we used the following ‘robustness-oriented’ methods:

- **reweight.** This is our proposed methods described in Section 3.1. It involves two training phases and a reweighting operation on the dataset. The procedure is easily implemented on top of existing linear classifier training software such as the LIBLINEAR [8]. See Algorithm 1 for pseudo code.
- **avg.** This is a variant of feature bagging (Section 4.1) whereby the final classifier is an average of K base classifiers, each trained on (possibly overlapping) randomly selected feature subsets. In the experiments of this paper, we set K to 10, and the subset size to 50% of the full set size. See Algorithm 2 for pseudo code.
- **fscale.** This is a variant of worst case feature noise injection algorithm (Section 4.4) that relaxes the constraint on the maximum number of features subject

²In reality, spammers have more knowledge about spam features but little about the ham features. Therefore, this adversarial attack simulation is more realistic.

to noise injection. The training algorithm scales the features of each feature vector optionally in order to maximize the loss function value (cf. (6) and (7)). See Algorithm 3 for pseudo code.

The three abovementioned methods can be further specialized to the cases of LR and SVM. Therefore, we denote the specializations of these methods with suffixes ‘-LR’ and ‘-SVM’ for LR and SVM, respectively.

Algorithm 1 Training of reweight-LR or reweight-SVM

- 1: **input:** Training set $D = \{(x_i, y_i)\}_{i=1}^m \subset \mathbb{R}^d \times \{\pm 1\}$
- 2: Get initial weight vector w by training a LR/SVM on D
- 3: Compute scaling vector $s_j = \log(e + |w_j|)$, $j = 1, \dots, d$
- 4: Let $\bar{D} := \{(\bar{x}_i, \bar{y}_i)\}_{i=1}^m$ be a new dataset such that:

$$\bar{x}_{ij} = x_{ij}/s_j, j = 1, \dots, d \quad \text{and} \quad \bar{y}_i = y_i \quad (8)$$

- 5: Get final weight vector \bar{w} by training a LR/SVM on \bar{D}
 - 6: **return:** \bar{w}
-

Algorithm 2 Training of avg-LR or avg-SVM

- 1: **input:** Training set $D = \{(x_i, y_i)\}_{i=1}^m \subset \mathbb{R}^d \times \{\pm 1\}$,
Number of base models K , Feature subset size F
- 2: **for** $k = 1$ to K **do**
- 3: Let I be an index set of F randomly selected integers from the set $\{1, \dots, d\}$ without replacement
- 4: Let $\bar{D} := \{(\bar{x}_i, \bar{y}_i)\}_{i=1}^m$ be a new dataset such that:

$$\bar{x}_{ij} = \begin{cases} x_{ij} & j \in I \\ 0 & j \notin I \end{cases}, j = 1, \dots, d \quad \text{and} \quad \bar{y}_i = y_i \quad (9)$$

- 5: Get weight vector w_k by training a LR/SVM on \bar{D}
 - 6: **end for**
 - 7: Let $w_{\text{avg}} = \frac{1}{K} \sum_{k=1}^K w_k$
 - 8: **return:** w_{avg}
-

5.3 Experimental Setup

With the datasets and methods described in previous sections, we trained each methods with tuning of various hyperparameters such as (1) artificial bias feature³, (2) regularization constant, and (3) scaling function s for **reweight**, and **fscale**. The best model for each method was determined by the highest area under ROC curve (AUC) at 10% false positive rate [13] on validation set. With the best models, we carried out classifier performance evaluation in the subsequent sections.

5.4 Results and Analysis

5.4.1 Simulated Adversarial Attack

In this section, we assess the robustness of classifiers trained with abovementioned methods in the setting of adversarial attack. As mentioned in previous section, our simulated adversarial attack is a rather realistic lower bound of the true measure of classifier robustness. The evaluation proceeds with a given number of adversarial attack K and alternates between feature *deletion* and feature *insertion* until

³Artificial bias feature refers to the additional constant appended to each feature vector. This simplifies the optimization procedure considerably [12]

Algorithm 3 Training of fscale-LR or fscale-SVM

- 1: **input:** Training set $D = \{(x_i, y_i)\}_{i=1}^m \subset \mathbb{R}^d \times \{\pm 1\}$
 - 2: Get initial weight vector w by training a LR/SVM on D
 - 3: Compute scaling vector $s_i = \log(e + |w_i|)$
 - 4: **while** not converged **do**
 - 5: Get current weight vector \bar{w}
 - 6: **for** each instance $(x, y) \in D$ **do**
 - 7: **for** $i = 1$ to d **do**
 - 8: //Minimize the value of $yw^T x$ by scaling each
 - 9: //feature x_i of x by 1 or $1/s_i$
 - 10: Let $u^* = \arg \min_{u \in \{1, s_i\}} y_i w_i x_i / u$
 - 11: Let $\bar{x}_i = x_i u^*$
 - 12: **end for**
 - 13: **end for**
 - 14: Let \bar{D} be a version of D with x replaced by \bar{x}
 - 15: Compute loss function value l and gradient g of LR/SVM on \bar{D}
 - 16: Pass l and g to iterative convex solver such as BMRM [17] to complete current iteration
 - 17: **end while**
 - 18: **return:** \bar{w}
-

the number of attacks is reached. For feature deletion, it models the Bad Word Obfuscation (BWO) trick⁴ by setting the most spammy feature (i.e. the one with the largest positive weight) off. Likewise, the feature insertion part models the Good Word Insertion (GWI) trick⁵ by setting a hammy feature on. Since spammers have less knowledge about the global feature set, we further restrict the *good* feature to be set on in each instance must not be better than the best feature in the instance. See Algorithm 4 for detailed pseudo code for this evaluation procedure.

We carried out extensive experiments and found that, almost in all cases, the SVM variants of the methods did not perform as good as their LR counterparts. For ease of interpretation, we present here only the results for LR variants of the methods.

The right column of Figure 1 shows the AUC evaluated at 10% false positive rate as a function of number of adversarial steps performed. Since what we are after in this experiment is the relative trends of classifier performance degradation among the methods as adversary gets intense, statistical significance is not a very appropriate criterion in this case.

In general, **avg-LR** performs the best on the original evaluation set and exhibits strongest robustness among the methods as the number of adversarial steps increases except on the Hotmail dataset where its performance degrades faster than others. **reweight-LR** consistently outperforms the baseline LR, albeit worse than **avg-LR** in most of the cases. Although the performance of **fscale-LR** is the worst during the first few steps of adversarial attack, it has the slowest rate of performance degradation on all datasets used in this experiment.

The left column of Figure 1 shows the proportion of the

⁴I.e., known bad words are obfuscated in some way such that these words do not appear in the feature set.

⁵I.e., insertion of random words or those with educated guess to make spam email looks more like a ham.

absolute values of top K feature weights:

$$\text{proportion} = \frac{\text{Sum of K largest absolute values of } w}{\text{Sum of all absolute values of } w} \quad (10)$$

This essentially illustrates the evenness of the distribution of weights among the features, i.e., flatter line implies more evenly distributed weight vector. From the figure, we see that evenly distributed weight vectors such as those of **avg-LR** and **reweight-LR** are more likely to result in stronger robustness. (Note that the distributions of feature weight of **avg-LR**, **reweight-LR**, and **LR** are not much more even than that of **fscale-LR** as the difference is only around 0.4%.)

Algorithm 4 Classifier performance evaluation under simulated adversarial attack

```

1: input: weight vector  $w$ , # of attacks  $K$ , test set
    $D = \{(x_i, y_i)\}_{i=1}^m \subset \{0, 1\}^d \times \{\pm 1\}$  with binary valued
   features
2: for  $k = 1$  to  $K$  do
3:   for each spam instance  $(x, y) \in D$  do
4:     if  $k$  is odd then
5:       //Delete feature with largest positive weight
6:       //(i.e., most spammy feature)
7:       Let  $j = \arg \max_q \{w_q x_q\}$ 
8:       Set feature  $x_j$  to 0 if  $w_j x_j > 0$ 
9:     else
10:      //Insert a non-existing feature which has
11:      //negative weight larger than the smallest
12:      //negative weight of existing feature
13:      Let  $v = \min_q \{w_q x_q\}$ 
14:      Let  $j = \arg \min_q \{w_q \mid 0 > w_q \geq v \text{ and } x_q = 0\}$ 
15:      Set feature  $x_j$  to 1 if  $j \neq \emptyset$ 
16:     end if
17:   end for
18:   Evaluate performance on L2-normalized  $D$ 
19: end for

```

5.5 Non-stationary Evaluation Set

In this section, we aim at measuring the robustness of classifier from the point of view of concept drift which occurs naturally in the email environment. In other words, we assess the performance of classifiers on a time split evaluation set provided the training set is *fixed* and is in close proximity of the whole evaluation set. This setting is fundamentally different from that of *online filter evaluation* as in the latter case, classifiers are re-trained very often and hence less susceptible to concept drift⁶.

In fact, the concept drift setting we emphasize here is a very common problems faced by real world content based email spam classifiers. By studying classifier robustness in this sense, we are able to characterize the effective lifetime of a spam classifier before it becomes obsolete (and requires re-training with newer training set).

The ECML06 dataset was excluded from this experiment as it does not satisfy the assumption of the performance evaluation that the training and evaluation sets are ordered in chronological order. On the remaining three datasets, we partition each of the corresponding evaluation sets into 10 roughly equal-sized subsets while preserving the time order of the instances. We evaluated the AUC at 10% false positive

⁶I.e., the evaluation environment diverges further away from the training environment.

rate on each subset and plot the results in Figure 2. From the figure, we see that the performance of the methods **avg-LR** and **reweight-LR** are the best and outdo the baseline **LR** and **fscale** on some subsets by wide margin.

6. CONCLUSIONS & FUTURE WORKS

We have seen in the experiments that the performance and robustness of simple techniques such as **reweight-LR** and **avg-LR** surpass that of standard method such as **LR**. The computational overhead for **avg-LR** is negligible as the training of K base classifiers can be parallelized over a cluster of machines. This parallelized training is feasible especially in large organizations such as email service providers.

Since the simulated adversarial attack performance evaluation is computationally cheap and it provides a sequence of classifier performance measures, it can well be considered as an alternative criterion for use in the cross validation phase. I.e., we say classifier A is better than classifier B when $(p_{A,1}, \dots, p_{A,k}) \geq_\delta (p_{B,1}, \dots, p_{B,k})$, where $p_{C,i}$ is some monotonically decreasing performance measure of classifier C after i steps, and “ $s \geq_\delta t$ ” refers to “ $s = t$ ” when $|s - t|$ is less than some small positive value δ , and to “ $s \geq t$ ” otherwise. It is important to note that this criterion is more stable and less sensitive to numerical rounding errors in performance measure values.

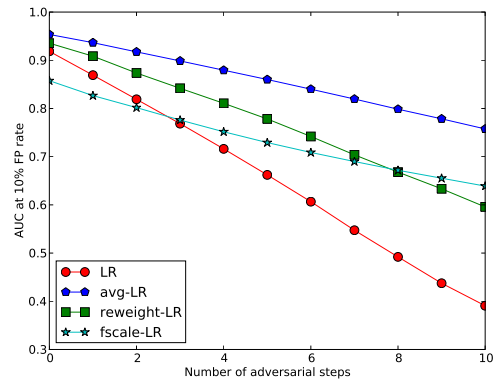
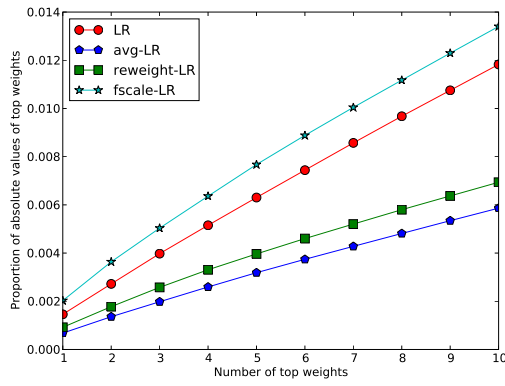
In the near future, we plan to study the sensitivity of the performance of **avg** and **reweight** to the number of base classifiers and feature subset size, and initial training set size, respectively. It will be interesting and insightful to study how **fscale** manages to achieve slow degradation in the cases of intense adversarial attack.

Acknowledgement

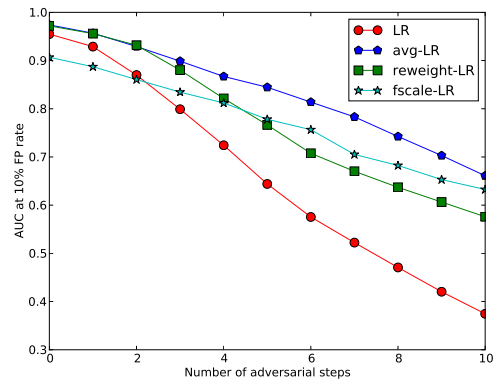
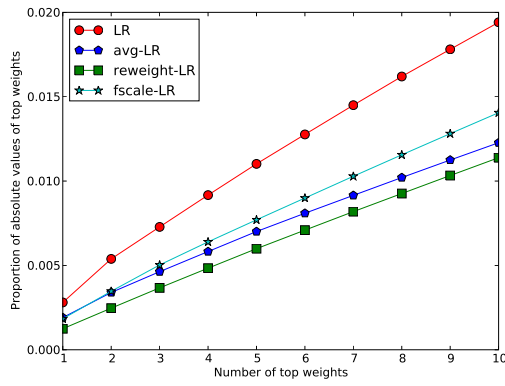
The authors would like to thank the reviewers for comments and suggestions that help improve the quality of this paper greatly.

7. REFERENCES

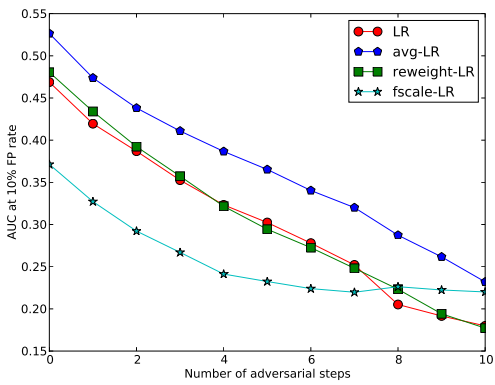
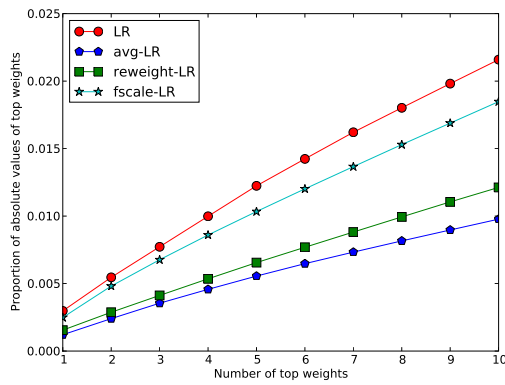
- [1] S. Bickel. Discovery challenge 2006 overview. In *Proceedings of ECML/PKDD Discovery Challenge Workshop*, 2006.
- [2] S. Bickel, M. Brückner, and T. Scheffer. Discriminative learning for differing training and test distributions. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 81–88, New York, NY, USA, 2007. ACM.
- [3] M.-W. Chang, W.-T. Yih, and C. Meek. Partitioned logistic regression for spam filtering. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–105, New York, NY, USA, 2008. ACM.
- [4] G. Cormack. TREC 2006 spam track overview. In *Fifteenth Text REtrieval Conference (TREC-2006)*, NIST, Gaithersburg, MD, 2006.
- [5] G. Cormack and T. Lynam. TREC 2005 spam track overview. In *Fourteenth Text REtrieval Conference (TREC-2005)*, NIST, Gaithersburg, MD, 2005.
- [6] O. Dekel and O. Shamir. Learning to classify with missing and corrupted features. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 216–223, New York, NY, USA, 2008. ACM.



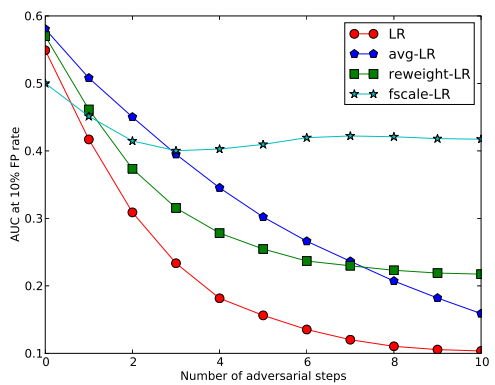
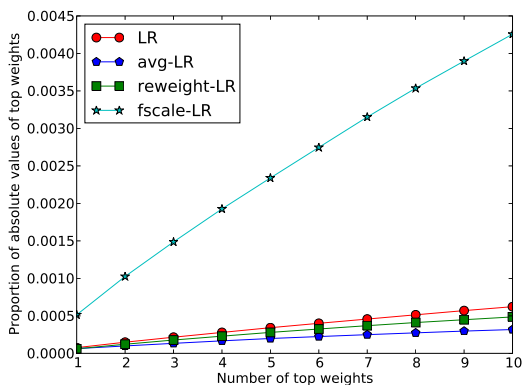
(a) TREC05



(b) TREC06



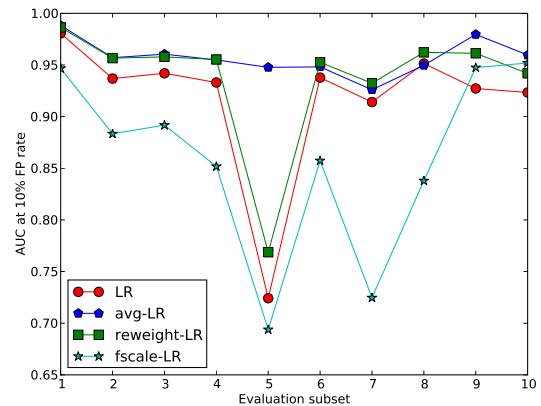
(c) ECML06



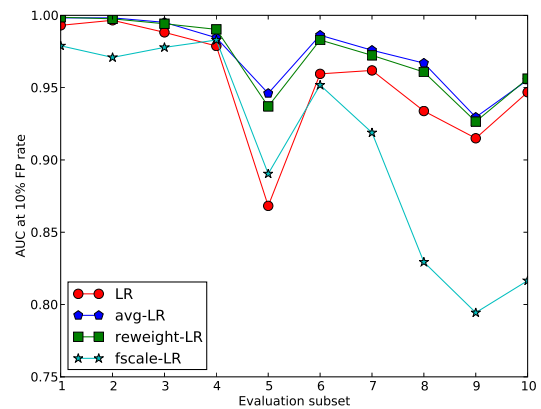
(d) Hotmail

Figure 1: Left column: Proportion of top 1 to 10 weights of best classifiers on various datasets. Right column: Performance of best classifiers under 0 to 10 adversarial attack steps on various datasets.

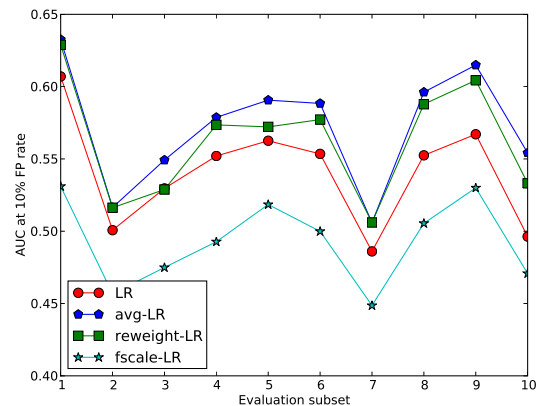
- [7] M. Dredze, K. Crammer, and F. Pereira. Confidence-weighted linear classification. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 264–271, New York, NY, USA, 2008. ACM.
- [8] R.-E. Fan, J.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, Aug. 2008.
- [9] G. Forman. BNS feature scaling: an improved representation over tf-idf for svm text classification. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 263–270, New York, NY, USA, 2008. ACM.
- [10] A. Globerson and S. Roweis. Nightmare at test time: robust learning by feature deletion. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 353–360, New York, NY, USA, 2006. ACM.
- [11] J. Huang, A. J. Smola, A. Gretton, K. M. Borgwardt, and B. Schölkopf. Correcting sample selection bias by unlabeled data. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 601–608. MIT Press, Cambridge, MA, 2007.
- [12] S. S. Keerthi and D. DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. *J. Mach. Learn. Res.*, 6:341–361, 2005.
- [13] A. Kolcz and W.-T. Yih. Raising the baseline for high-precision text classifiers. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 400–409, New York, NY, USA, 2007. ACM.
- [14] J. D. M. Rennie, L. Shih, J. Teevan, and D. R. Karger. Tackling the poor assumptions of naive bayes text classifiers. In *ICML '03: Proceedings of the 20th international conference on Machine learning*, pages 616–623, 2003.
- [15] C. Sutton, M. Sindelar, and A. McCallum. Feature bagging: Preventing weight undertraining in structured discriminative learning. Technical report, in *Structured Discriminative Learning*. In: CIIR, 2005.
- [16] C. H. Teo, A. Globerson, S. Roweis, and A. J. Smola. Convex learning with invariances. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1489–1496. MIT Press, Cambridge, MA, 2008.
- [17] C. H. Teo, Q. Le, A. J. Smola, and S. Vishwanathan. A scalable modular convex solver for regularized risk minimization. In *Proc. ACM Conf. Knowledge Discovery and Data Mining (KDD)*. ACM, 2007.
- [18] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, New York, 1995.



(a) TREC05



(b) TREC06



(c) Hotmail

Figure 2: AUC at 10% false positive rate of best classifiers (trained on fixed training sets) on different evaluation subsets.