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

Learning To Detect Vandalism in Social Content Systems: A Study On Wikipedia

Vandalism Detection in Wikipedia

Sara Javanmardi · David. W. McDonald · Rich Caruana · Sholeh Forouzan · Cristina V. Lopes

Received: date / Accepted: date

Abstract A challenge facing user generated content systems is vandalism, edits that damage content quality. The high visibility and easy access to social networks makes them popular targets for vandals. Detecting and removing vandalism is critical for these user generated content systems. Because vandalism can take many forms, there are many different kinds of features that are potentially useful for detecting it. The complex nature of vandalism, and the large number of potential features, make vandalism detection difficult and time consuming for human editors. Machine learning techniques hold promise for developing accurate, tunable, and maintainable models that can be incorporated into vandalism detection tools. We describe a method for training classifiers for vandalism detection that yields classifiers that are more accurate on the PAN 2010 corpus than others previously developed. Moreover, because of the high turnaround in social network systems, it is important for vandalism detection tools to run in real–time, we use feature selection to find the minimal set of features consistent with high accuracy. In addition, because some features are more costly to compute than others, we use cost–sensitive feature selection to reduce the total computational cost of executing our models. In addition to the features previously used for spam detection, we introduce new features mainly based on user action histories. The user history features contribute significantly to classifier performance. The approach we use is general and can easily be applied to other user generated content systems.

Keywords Vandalism Detection · Wikipedia · UGC · Feature Selection · Machine Learning · Lasso · Cost–Sensitive Learning · User History · Random Forests

S. Javanmardi
University of California, Irvine Donald Bren Hall 5042 Irvine, CA 92697-3440
Tel.: (949) 824-2901
Fax: (949) 824-4056
E-mail: sjavanma@ics.uci.edu

1 Introduction

Wikipedia is an open, collaboratively edited encyclopedia that is a heavily used reference source on the Internet. High visibility and the simplicity of editing almost any article have made Wikipedia a popular target for vandals. Maintaining the quality of information in Wikipedia as well as many other User Generated Content (UGC) systems requires identifying and removing vandalism.

In general, vandalism is any deliberate attempt to compromise the integrity of an online source. This covers a broad range of destructive actions such as advertising, attacks on public figures, contributing misinformation, subtle distortion through equivocation or exaggeration, phishing, altering the destination of apparently legitimate links, misleading comments on a change, faulty application of markup text – among many other forms. Really, the range of types of vandalism is quite astonishing.

Many UGC systems, like Wikipedia, rely on extensive manual efforts to combat vandalism. Some automated vandal fighting tools, often in the form of semi–automated bots, are being used to alleviate this laborious task. More recently, machine learning based approaches have been proposed [23]. However, a highly accurate vandalism detection approach that can be applied on the large–scale of Wikipedia is still missing. A successful approach needs to scale well, be robust in the face of widely varying levels of user participation and high user turnover – and should be able to detect vandalism in real–time.

In this work we build upon our previous work [20] and describe the development of a low–cost and highly accurate vandalism detection model that is practical for real–time applications. While our more general focus is on UGC systems, we develop the model based on a Wikipedia corpus from the "Uncovering Plagiarism, Authorship, and Social Software Misuse (PAN)" workshop. This workshop has been developing corpora and testing algorithms head–to–head since 2007 and thus provides data and benchmarks for comparing our results. By aggregating several features used in the PAN competition and identifying a number of new features. We introduce a set of user features which account for past user activity and thereby represent something of the user reputation. The resulting classifier performs better than prior results in the PAN competition.

Further, we try to compress the model by learning from the smallest number of features possible. First, we decrease the feature set by eliminating redundant features. Then, we take into account cost of acquisition of features relative to their individual contribution to the overall performance of the classifier. All the experiments are done based on a MapReduce paradigm, which makes our approach both efficient and scalable.

This work makes two important contributions. First, while our specific application is to Wikipedia, the machine learning techniques are focused on leveraging low–cost features that can be generally applied to many forms of UGC. The overall approach is therefore very general and could provide a basis for a wide range of semi-automated vandalism detection tools. Second, one

particularly valuable set of features relate to the 'reputation' of the user. By recognizing the specific value of a reputation function for vandalism detection, we provide a general approach for understanding the cost/benefit trade–off of determining reputation. Further, by illustrating the specific value of these reputation features we raise a critical challenge for large–scale networked data; What are inexpensive techniques for determining user reputation from large complex networked data?

This paper is structured in the following way. We begin with a review of the relevant work on vandalism mitigation from both a user and a technical perspective. Through this we identify a number of persistent challenges for users as well as sets of features that are commonly used to develop technical solutions for vandalism detection. In subsequent sections we elaborate a relevant set of features, and use those features to train a classifier that is effective at predicting vandalism. We then apply *lasso* to learn a sparse low–cost model whose accuracy is comparable to the original classification model. We close the paper by considering some applications of the resulting model and the more general implications of our approach and findings.

2 Background

Vandalism detection has been a concern for Wikipedia since its inception. Vandalism in Wikipedia is defined as any addition, removal, or change of content in a deliberate attempt to compromise the integrity of Wikipedia. According to this broad definition, vandalism can include spamming, lobbying and destroying the edits of others. Wikipedia relies mostly on its human editors and administrators to fight vandalism; identifying instances of potential vandalism by reading a diff of the change and reverting changes that look to be a form of vandalism. But the scale of Wikipedia makes locating all vandalism very labor intensive. Tools such as Vandal Fighter, Huggle, and Twinkle are used to monitor recent changes to articles in real–time and revert those changes deemed vandalism [14].

In the following we describe two broad approaches to vandal fighting; (a) the user approach which relies mostly on tools to assist user detection, and (b) more automated approaches that generally rely on bot or other algorithms.

Viegas *et al.* [34] conducted some early work on the types of vandalism found in Wikipedia. They used a visualization technique called "History Flow" to see the various ways pages were edited and changed over time. In considering these changes they identified five types of vandalism: Mass Deletion, Offensive Copy, Phony Copy, Phony Redirection, and Idiosyncratic Copy. They also analyzed the time for repair of vandalism, which they termed "survival time". They found that the median survival time for some vandalism is quite short, on the order of minutes. However, they noted that the mean time is skewed long, on the order of days or weeks, depending on the type of vandalism. This means there is some vandalism that goes undetected for long periods of time.

In some follow–up work, Priedhorsky *et al.* [29], considered the impact of a piece of vandalism. That is, if some vandalism lasts on a site for days, how likely is it that a user might stumble across that and be misinformed or otherwise get a wrong impression about the quality of the entire content based on a vandalized page. They developed a model based on page viewing behaviors and vandalism persistence. Their model correlates closely with the empirical results from Viegas *et al.* [34] and further illustrates that about 11% of vandalism would last beyond 100 page views, with a small fraction of a percent lasting beyond 1000 page views.

Priedhorsky *et al.* [29] also considered the types of vandalism and how likely users were to agree on the types of vandalism. They began with the vandalism types from [34] making some small refinements and identified two additional types: Misinformation, and Partial Delete. They identified the most frequent categories of vandalism as Nonsense (Phony Copy) 53%, Offensive 28%, Misinformation 20% and Partial Delete 14%. However, they also noted that the rate of agreement among their users for some categories is somewhat low with Misinformation having the lowest level of agreement for these top four categories. This means that some of the most subtle vandalism, in the form of Misinformation, is even hard for users to agree upon.

In recent work, Geiger & Ribes [14] studied the process of editors who participate in "Recent Changes Patrolling" using Huggle. Their study raises some nice issues about the work to remove vandalism and how it is performed, illustrating how Wikipedians consider the activities of others when considering a change as potential vandalism. The study is a case study considering when an individual should be banned from the site for contributing too much content that has been deemed vandalism. In the case of the decision to ban a vandal, the effort is to understand whether the activities are intentionally designed to corrupt content and wreak havoc on the community itself. This work illustrates that there is a fair amount of vandalism which is somewhat 'routine' but that some is still difficult to detect even by people who are practiced at looking for vandalism. Another interesting insight is that most tools that support vandalism rollback do not yet categorize or provide a prediction rating for the edit being viewed. Most tools simply show the edit, the prior version, and the IP or username of the editor who made it.

A second set of approaches rely on automated bots and algorithms. This approach has generated lots of research activity, so we focus on the prior work that is most related to how we have approached the problem.

Since 2007 automated bots have been widely used to fight vandalism in Wikipedia. The most prominent of them are ClueBot and VoABotII. Like many vandalism detection tools, they use lists of regular expressions and consult databases with blocked users or IP addresses. The major drawback of these approaches is that most bots utilize static lists of obscenities and grammatical rules that are hard to maintain and, with some creativity, can be easily thwarted. A study on performance analysis of these bots in Wikipedia shows that they can detect about 30% of the instances of vandalism [32].

Several machine learning approaches have recently been proposed that would improve vandalism detection [32,28,10,18]. Comparing the performance of these approaches is difficult because of several shortcomings in early vandalism detection corpora. These early corpora were too small, they failed to account for the true distribution of vandalism among all edits, and the hand labeling of the examples had not been double–checked by different annotators. These shortcomings were resolved by the creation of a large–scale corpus for the PAN 2010 competition consisting of a week's worth of Wikipedia edits (PAN–WVC–10) [27].

The PAN 2010 corpus is comprised of 32,452 edits on 28,468 different articles. It was annotated by 753 annotators recruited from Amazon's Mechanical Turk, who cast more than 190,000 votes. Each edit in the corpus was reviewed by a minimum of three annotators. The annotator agreement was analyzed in order to determine whether each edit is a regular edit or vandalism, with 2,391 edits deemed to be vandalism. The corpus is split into a training set and a test set, which have 15,000 and 18,000 edits, respectively [23].

A survey of detecting approaches [23] shows that about 50 features were used by the 12 different teams. Features are categorized into two broad groups: (1) edit textual features; (2) edit meta information features. Edit textual features are extracted based on the text of the edit. Some features in this category are adopted from previous work on spam detection in emails or blogs. For example, "Longest character sequence" and "upper case to low case char ratio" are known to be important features for spam detection in emails [17].

Traditional spam detection systems for emails mainly rely on textual features based on the content. Most features show existence of particular words or phrases [6,22,26,31]. In some cases non–textual features are extracted from meta data. For example, whether or not a message contains attachments [6].

Edit meta information mainly contains two types of features: user features and comment features. In Wikipedia when an individual makes an edit, there is the opportunity to provide a short comment on the change. As a convention, many editors will use the comment to briefly explain the rationale for the change or what type of change is being made. Comment features are extracted based on the comment related to an edit. Most teams used comment features, but two teams extensively relied on user features. User features are extracted based on the editing pattern of the user.

Identifying an editing pattern requires that some amount of state, or history, be maintained. Some forms of UGC do not maintain history as a function of the system design, but in the case of wikis user history is a given. Two teams in the PAN competition relied on user reputation features extracted from the edit revision history. Those teams placed second and third in the competition. Other approaches rely more heavily on a model of user reputation. Adler *et.al.* [3] used WikiTrust to estimate user reputation. In their system, users gain reputation when their edits are preserved and they lose reputation when their edits are reverted or undone [4].

Javanmardi *et.al.* [19] used user reputation features based on the reputation management system they developed earlier. Compared to [4], the model is

simpler and more efficient. One reason is that it is only based on the stability of inserts. In [4], stability of deletes and reverts are also considered. A detailed comparison between these two approaches are presented in [19].

Potthast *et.al.* [23] combined the predictions submitted by the top 8 teams and developed a meta classifier based on the predictions. To learn the meta classifier, they used random forest. This improved the classification performance (ROC–AUC) of the top team significantly, from 0.91580 to 0.9569. Using a similar approach, Adler *et.al* [5] developed a meta classifier based on the predictions of three different classifiers. To evaluate the meta classifier, they merged train set and test set and reported ROC–AUC based on 10–fold cross validation. Hence, their results are not comparable with PAN competition results and our results.

In general, meta classifiers work at a macro level by aggregating results from different classifiers. However, this makes them even more complex and less practical for real–time applications. In contrast, in this study we work at the level of individual features and focus on building accurate classifiers with a minimum set of features. We show that the classification performance of the compact classifier is comparable to the meta classifier developed in [23].

3 Feature Extraction

Our feature extraction is based on the complete Wikipedia history dump, released on Jan, 2010. It turns out that 41 edits in the PAN corpus are missing from the "complete" Wikipedia dump. We use crawler4j [1] to extract the data of these missing edits. We also used additional Wikipedia SQL dumps to extract users with special access rights such as administrators and bureaucrats.

Using all these data, we extract 66 features. This feature set includes most of the features used in the PAN competition by different teams [23]. In addition, we introduce new features. Table 1 shows the features along with their definitions (note: each row might represent more than one feature).

We categorize the features into four groups. Similar to [23] we separate edit textual features and edit meta data features. Since the edit meta data features contains both user and comment features, we consider them as different groups. In addition, we add language model features as a new group. These features capture topical relevance and are estimated based the language model of the newly submitted revision and the background. The effectiveness of using these features for vandalism detection has been studied in [10, 25].

- **User Features**: We introduce 12 features for each user including statistical and aggregate features. We calculate these features by mining history revisions up to time T. For the purpose of this study, we consider T as 2009–11–18 which is the time–stamp of the earliest edit log in the PAN corpus. Hence, all features in this category are based solely on history data.
- **Textual Features**: We have 30 features in this category. Unlike previous work [23], in this work textual features are calculated not only based on the

inserted content but also based on the deleted content. To distinguish these features we use "Ins" and "Del" prefix throughout this paper. For example, *Vulgarism* shows the frequency of vulgar words. We expect insertion of vulgar words to be a signal for vandalism. Conversely, we expect deletion of such words to be a signal for legitimate edits aiming at removing vandalism.

- **Meta Data Features**: We have 22 features in this category. Most features are extracted from the comments associated with the edits. For example, we have similar textual features as in the previous category but here we extract them based on the comment. Because of descriptions we do not define them in Table 1. These features are specified by *.
  In addition to these, we introduce some new features that we extract from the automatic comments generated by Wikipedia. These comments specify which section of the article has been edited. We extract unigram, bigrams, and trigrams from these types of comments. We use feature selection on PAN train set to extract the important ones. For example, the short time interval between old and new revision might be a signal for detecting vandalism.
- **Language Model Features**:
  In this category we have 3 features which calculate the *Kullback–Leibler* distance (KLD) between two unigram language models. We calculate KLD between the previous and the new revision [10]. We introduce two more features: the KLD between the inserted content and the previous revision. Similarly, we calculate the KLD between the deleted content and the previous revision. Our intuition behind KLD features is that sometimes vandalism comes with some unexpected words so we expect to see sharp changes in the distance. Conversely, deleting unexpected words can be a signal for legitimate edits.

4 Learning Vandalism Detection Model

We consider vandalism detection as a binary classification problem. We map each edit in PAN corpus into a feature vector and learn the labels by mapping feature vectors onto {0,1}, where 0 denotes legitimate edit, and 1 vandalistic edit. To learn a classifier and tune its free parameters, we use PAN train set and keep the test set untouched for final evaluation.

We use different binary classification algorithms which have been widely applied to spam detection such as Naive Bayes [22], Logistic Regression [9], and SVM [31]. We also use random forests which has been shown to be a very effective binary classifier [8]. Table 2 shows the classification performance for 3–fold cross validation on the train set and also on a test set. We use the Java implementation of these binary classifiers from Weka [16] and tune the free parameters based on cross validation. Because we use the train set to learn some of the features, and to make our results comparable to the PAN results, we do not use the test set in any way during training. The test set is only used to measure classification performance. Here we report classification

Table 1 Feature set descriptions. Features marked with star also are extracted based on insertions and deletions.

Feature	Description
DSR, DDSR, Rep	Aggregated features representing a user's reputation
Ins Words	Total number of words inserted by a user
Del Words	Total number of words deleted by a user
Lost Words	Total number of deleted words from a user
Ins Revision	Total number of revisions a user has done insertion in
Del Revision	Total number of revisions a user has done deletion in
Ins Page	Total number of pages a user has done insertion in
Del Page	Total number of pages a user has done deletion in
User Type	User has some special rights, such an admin, a bot, or a bureaucrat
User Page	User has a user page in Wikipedia
Ins Size	Number of inserted words
Del Size	Number of deleted words
Revision Size	Size difference ratio between the old and the new revision.
Blanking	The whole article has been deleted
Internal Links	Number of links added to Wikipedia articles
External Links	Number of added external links
Word Repetitions	Length of the longest word
Char Repetitions	Length of the longest repeated char sequence
Compressibility	Compression rate of the edit differences.
Capitalization*	Ratio of upper case chars to lower case chars
Capitalization All*	Ratio of upper case chars to all chars
Digits*	Ratio of digits to all letters
Special Chars*	Ratio of non-alphanumeric chars to all chars
Diversity*	Length of all inserted lines to the (1 / number of different chars)
Inserted Words*	Average term frequency of inserted words
Vulgarism*	Frequency of vulgar words
Bias*	Frequency (impact) of biased words
Sex*	Frequency (impact) of sex related words
Spam*	Frequency (impact) of spam related words
Pronouns*	Frequency (impact) of personal pronouns
WP*	Frequency (impact) of mark up related words
Special Words*	Aggregation of vulgarism, bias, sex, spam, pronouns, and WP ratios
Time Diff	Time interval between the submission of the old and the new revision
Category	If the automatic comment contains "category"
Early Years	If the automatic comment contains "early years"
Copyedit	If the automatic comment contains "copyedit"
Personal Life	If the automatic comment contains "personal life"
Revert	If the automatic comment contains "revert"
Revision Ordinal	Ordinal of the submitted revision
Length	Length of the comment
Reverted	if the MD5 digest of new revisions is the same as one of the old ones in window size of 10
KL Distance	Kullback–Leibler distance between the old revision and the new revision
KL Distance Ins	Kullback–Leibler distance between the inserted words and the new revision
KL Distance Del	Kullback–Leibler distance between the deleted words and the new revision

Table 2 Classification performance for the binary classifiers

Algorithm	ROC–AUC (CV)	ROC–AUC (Test)
Naive Bayes	0.9482 ± 0.0057	0.9068
Logistic Regression	0.9494 ± 0.0070	0.9128
SVM (LibSVM, RBF Kernel, Gamma = 0.15, c=11)	0.9537 ± 0.0007	0.9202
Random Forest (1000 trees)	0.9739 ± 0.0024	0.9553

performance in terms of area under the ROC curve (ROC–AUC). (The ROC-AUC metric was used in the PAN 2010 evaluation. Results for area under the precision-recall curve are similar.) In all our experiments random forests outperformed the other classifiers.

4.1 Vandalism Detection Using Random Forest

Random forest is not widely used for spam detection because it is known to be expensive in terms of time and space [33]. In this section we explain how we can train random forest classifiers efficiently and gain high classification performance at the same time.

Statistical analysis of Wikipedia edits show that roughly 7% of edits are vandalistic [19], which is consistent with the vandalism ratio in PAN corpus. Given this, we need to use machine learning algorithms which are robust to imbalanced data, such as random forests. In addition, random forests are a suitable option for datasets with missing data [7]. The PAN data set is an imbalanced dataset and some features such as the user group features are sometimes missing; For 4% of users we do not have any user group information at all. This suggests that a random forest model could have benefits over other techniques that may not deal as well with imbalanced data or missing features. One advantage of the ROC-AUC metric is that it is fairly insensitive to imbalanced data.

To learn a random forest classifier, we need to tune two free parameters: the number of trees in the model and the the number of features considered to split each node. Our experiments show that classification performance is sensitive to the former but not to the latter. This result is consistent with Breiman's observation [7] on the insensitivity of random forests to the number of features considered in each split.

To tune the number of trees, we partition the train set into three folds and use 3–fold cross validation. Using three folds allowed us to keep a reasonably large number of vandalized cases in each training set (around 600). To find the optimal value for the number of trees, we need to sweep a large range of values. Hence, we need to design an efficient process for this purpose.

For each fold, we create a pool of $N = 10,000$ trees, each trained on a random sample of the training data in that fold. Then we use this pool for

creating random forests of different sizes. For example, to create a random forest with 20 trees, we randomly select 20 trees from this pool of N trees. However, since this random selection can be done in $C(N, 20)$ different ways, each combination may result in a different AUC. We repeat the random selection of trees $r = 50$ times and we report the mean and variance of the $F \times r$ results (where F is the number of folds).

The advantage of this approach is that we can calculate the mean and variance of AUC very efficiently for forests with different sizes without the need to train a huge number of trees. Otherwise, to report the mean and variance of AUC for random forests of size $k = 1$ *to* T, we would need to train $r + 2 \times r + 3 \times r + ... + T \times r = r * T(T + 1)/2$ trees for each fold, which is 10^8 trees. Using our approach we only need to train N trees per fold (in our experiments we used $N = 5 \times T$).

Figure 1 shows the mean of AUC as a function of number of trees in the model. As more trees are added to the model, mean of AUC increases and the variance decreases. The mean of AUC does not improve significantly after more than about 500 trees in the random forest but the variance continues decreasing. It should be emphasized that models with smaller variance are more stable and therefore more predictable in test environments. Although more trees may result in slightly better AUC values, we decide to set the number of trees at 1000 to have a balance between classification performance and model complexity. More complex models with more trees would require more time for prediction in a real–time application. Given this, the AUC on for 3–fold cross validation on the train set is 0.9739 ± 0.0024. The AUC value on PAN test set is **0.9553**. This result is significantly higher than the best AUC reported to the PAN competition which was 0.9218 [23].

5 Feature Selection

The classifier mentioned in the previous section makes its decision based on 66 features which fall into four logically different groups. However, computing and updating each of these features imposes significant off–line cost at run–time. For example, computing and updating features in the user group requires the tracking of all edits done by each individual. Maintaining a system that updates data for computing these features would come at a cost for the wiki. Some other features like textual features are computed after submission of a new edit and the vandalism detection system should be able to compute them in real–time.

In this section, we report the results of our experiments in finding a minimum set of features whose classification performance is almost as good as the one with all 66 features. In other words, we try to detect and eliminate redundant or unnecessary features. We consider two types of redundant or unnecessary features: (a) features that are not informative and do not help in discriminating legitimate and vandalistic content; (b) features correlated with

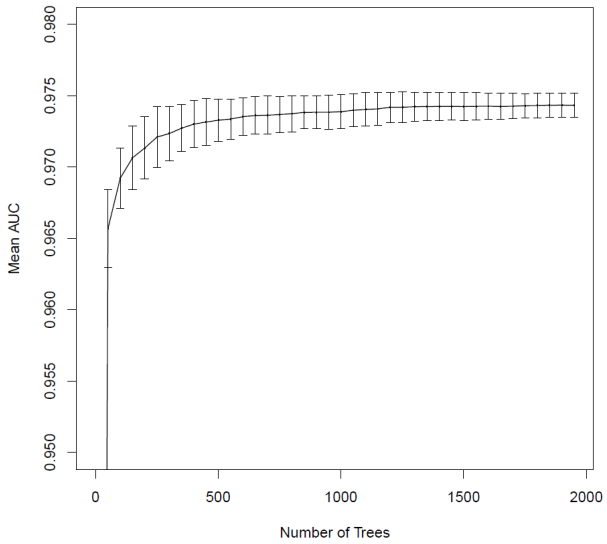

Fig. 1 The effect of number of trees on AUC mean and standard deviation

some other features so that once one of them is selected, adding others does not add any new discriminating power to the model.

In order to detect and eliminate redundant features, we perform two sets of experiments. First, in Section 5.1 we study the contribution of groups of features as a whole unit to examine if any of the four groups can be eliminated without a significant drop in AUC. Then in Section 5.2 we study the contribution of each feature individually and use the results of this analysis to eliminate redundant features.

Given the large number of experiments needed for this study, we use the Amazon MapReduce cluster to run them in parallel. In our implementation, each mapper receives specific config information and trains a classifier for that configuration. Then reducers aggregate the AUC results for different folds and report the mean AUC for the different configs.

5.1 Eliminating Groups of Features

For each group of features, we train a classifier without the features in that group. This will show us the drop in AUC when this group is ignored. Table 3 shows the results. These results indicate that removing features in user group and textual group results in large drops in AUC, while the drop in AUC for the meta data and the language model groups is much less.

Based on these results we can not infer that features in the Meta data and LM group are not informative. The only conclusion is that once we have both user and textual features in our feature set, adding meta data and language model features does not add substantial additional discriminating power to

Table 3 The drop in mean AUC when we eliminate a feature group. Results on train set are for 3–fold cross validation.

Dropped Group	Train set	Test set
User	−3.8%	−6.6%
Textual	−1.8%	−1.5%
Meta data	−0.4%	−0.3%
Language Model	−0.2%	−0.3%

Table 4 The mean AUC when we only use features in one group.

Selected Group	Train set	Test set
User	0.9399	0.9225
Textual	0.9001	0.8400
Meta data	0.7019	0.6924
LM	0.7271	0.6986
Baseline (all groups)	0.9739	0.9553

the model. Table 4 supports this interpretation: when we only use meta data or language model features the AUC value is much higher that of a random classifier (0.50). The single group of features that results in the highest AUC (0.9399) is the user group.

5.2 Eliminating Individual Features

In Section 5.1 we showed that all the four groups contain informative features. In this section, we attempt to find the smallest feature set whose AUC is comparable to the AUC of a classifier with 66 features. To this aim we do feature selection.

There are three different approaches for performing feature selection, univariate feature analysis, wrapper-based methods, and proxy methods [15, 21]. Univariate feature analysis methods such as Information Gain or Chi–Square evaluate features independently in order to estimate their importance. Because these methods evaluate the utility of each feature in isolation, they are unable to detect and benefit from correlations among the features and often yield unnecessarily large feature sets.

In wrapper-based methods one wraps a feature selection process such as forward stepwise selection or backwards stepwise elimination around the learning algorithm that will be used to train the final model so that the feature selection process can find a small set of features that works well with that learning method. Because the contribution of each feature is evaluated in the context of the other features used in the model, wrapper-based methods are very effective at eliminating features whose contribution to the model is redundant with other features already in the model, and thus often yield the smallest feature sets. The main difficulty with the wrapper approach is that it can be prohibitively expensive to wrap feature selection around expensive

learning methods such as random forests. For this reason, proxy feature selection methods are often used. In proxy methods, feature selection is performed for a simpler, more computationally tractable model class such as linear or logistic regression models, and the features found to be most important for the simpler model class is then used in the more complex, more expensive model class (in this paper random forests). Because proxy methods do not take the specific learning algorithm into account, they do not always find as compact a set of features as wrapper methods. However, in practice there usually is strong overlap between the features that are best for a simpler model and those that are effective in more complex models, so proxy methods often yield feature sets that are almost as small as those returned by wrapper methods.

Because the model class that appears to perform best on vandalism detection is computationally expensive (random forests), in this paper we consider only proxy feature selection methods. We need a feature selection algorithm that is efficient and which considers correlations between features and is able to detect and eliminate redundant features effectively. We use Lasso Logistic Regression (Least Absolute Shrinkage and Selection Operator for Logistic Regression) [13] for this purpose. This fits a regularized logistic regression model to features in such a way that the final model has a sparse solution in the feature space. Thus, the weight of redundant features in the final model would be zero. It means that we can remove these features from the model with no significant change in the classification performance. For this, we use the Logistic Regression Lasso implemented in glmnet package in R [13].

Lasso for logistic regression has a regularization parameter, λ, that is a trade off between sparsity of the model and its classification performance. Lower values for λ result in more relaxation of the regularization constraint which allows more features to have non–zero weights. The R package, glmnet [13], uses regularized maximum (binomial) likelihood to fit this regularized logistic regression model to the data. The problem can be formalized by the following:

$$\max_{(\beta_0,\beta)\in\mathbb{R}^{p+1}} [l(\beta_0,\beta) - \lambda\|\beta\|_1] \tag{1}$$

where

$$l(\beta_0,\beta) = \frac{1}{N}\sum_{i=1}^{N} y_i log(p(x_i)) + (1-y_i) log(1-p(x_i)) \tag{2}$$

and

$$p(x_i) = \frac{1}{1+exp(\beta_0 + x_i^T\beta)} \tag{3}$$

Table 5 shows the features selected for different values of λ. For $\lambda = 0.0716$ only one feature is selected. This means that according to lasso if we want to make the classification model based on only one feature, “Ins Special Words” would be our best choice. As we decrease the value of λ, more features are

Table 5 Feature Selection using Lasso. Parameter λ determines a trade off between number of selected features and AUC of the classifier. Smaller values of λ allow more features be selected and result in models with higher performance.

λ	Selected Features	AUC on PAN test set
0.0716	Ins Special Words	0.5974
0.0594	DDSR, Ins Special Words	0.8614
0.0373	DDSR, Rep, Ins Special Words	0.8965
0.0340	DDSR, Rep, User Page, Ins Digits, Ins Special Words	0.9074
0.0310	DDSR, Rep, User Page, Ins Digits, Ins Vulgarism, Ins Special Words	0.9090
0.0257	DDSR, User Page, KLDNew2OLD, Ins Digits, Ins Vulgarism, Ins Special Words	0.9197
⋮	⋮	⋮
0.0030	DDSR, Del Words, User Type, User Page, Copyedit, Personal Life, Revision Ordinal, Comment Length, KLDNew2OLD, Blanking, Ins Internal Link, Ins External Link, Longest Inserted Word, Ins Longest Char Repetitions, Ins Compressibility, Ins Capitalization, Ins Digits, Ins Special Chars, Ins Vulgarism, Ins Bias, Ins Sex, Ins Pronouns, Ins WP, Ins Special Words, Del Bias, Ins Digits, Comment Special Chars Comment Spam	0.9505
⋮	⋮	⋮

selected according to their importance. The second most important feature is "DDSR". The last column in Table 5 shows the value of the AUC on the PAN test set for classifiers trained on the selected features. As the number of selected features increases, we see a higher value for AUC but the cost of computing and updating the features would also increase.[1]

We use 3–fold cross validation on the PAN train set to pick the largest value for λ, where the drop in AUC is not statistically significant. The result was $\lambda = 0.0030$ which leads to selection of 28 features. Table 5 shows a list of the selected features. The AUC for this feature set on PAN test set is 0.9505. This feature set only includes less than half of the original features but the drop in AUC is only 0.005.

In this sparse feature set we have features from all groups. For example, "DDSR", "Del Words", "User Type", and "User Page" are selected from the user group. If we follow the Lasso path, features get added and eliminated as λ decreases. For example, "Rep" is selected as the third important feature, but because of its correlation with other selected features it is eliminated from the feature set later. Interestingly, "Rep" is computationally more expensive than "DDSR".

We have 17 features from the textual group. These features are also widely used for spam detection in other domains such as emails or blogs [17]. For example, "Ins Longest Char Repetitions" shows whether a user has inserted a long sequence of the same character which can be a good indicator of vandalism. There are also some textual features which are unique to Wikipedia. For

[1] The LASSO method does not necessarily yield a monotonically increasing set of features; it is possible that as λ is decreased some features that were in the set for larger λs might be removed from the set as other feature replace them.

example, “Del Bias” shows that the user has deleted words which represent bias and therefore is a good indicator of legitimate edit.

We have 6 features selected from the meta data group. For example, “Comment Length” or “Comment Special Chars” are selected as important features. “Personal Life” is another important feature. It shows whether the edit is made in the “Personal Life” section of a biography article. We have observed that this section of biography articles is more often vandalized and therefore this feature can be an important signal. This observation is consistent with Wikipedia statistics which shows high vandalism ratio in biography articles [2]. Given that Wikipedia automatically adds the name of an edited section to the comment associated to each edit, we can extract this feature from comments.

The only feature that is selected from the Language Model group is “KLD-New2OLD”. The goal of this feature is to detect sharp linguistic distance between the new and the previous revision which can be an important signal for vandalism detection.

6 Cost Sensitive Feature Selection

In Section 5 we focused on features selection in order to find a small subset of available features that would encapsulate all useful information for the task at hand. This was useful to eliminate redundant features and increase the interpretability of the selected features. However, in many applications, it is also useful to take into account the amount of effort required to compute the features. The importance of feature cost is ignored by many machine learning methods [12].

The general setting of cost sensitive learning is consists of designing classification models while taking into account the cost involved in the entire decision process. This includes the cost of data acquisition, the cost of labeling the training samples and the cost of making different decision errors [30]. In this work we focus on the first case where there are different costs to acquire different features. For example, to calculate value of the feature “Rep” we need to process all history revisions and track contributions of each individual user; we also need to track how other users have edited their contributions [19]. Some other features like “Comment Length” are easier to compute and only need a little light text processing.

To address this issue here we try to take into account the cost of acquiring different groups of features during the learning process. We are interested in cases where well–performing cheap features are preferred over expensive ones with slight difference in overall classification results. For this purpose, we study two different scenarios. In the first scenario features are selected based on their corresponding costs.

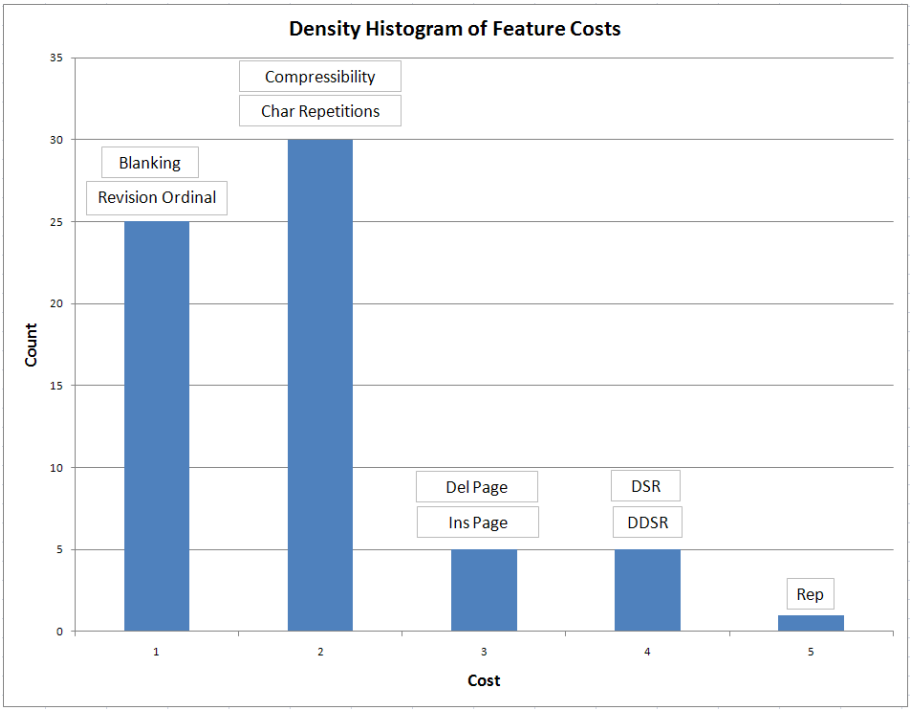

Fig. 2 Histogram of Cost of Acquisition of Features

6.1 Cost Sensitive Lasso

In this section we use regularized logistic regression based on lasso to implement cost sensitive feature selection. Unlike in 5.2 where we considered a fixed penalty λ for all features, here we assign higher penalties to the computationally more expensive features. We use glmnet package in R [13] which allows different penalties λ_j for each of the variables via a penalty scaling parameter $\gamma_j \geq 0$. If $\gamma_j > 0$, then the penalty applied to β_j is $\lambda_j = \lambda\gamma_j$. If $\gamma_j = 0$, that variable does not get penalized, and enters the model unrestricted at the first step and remains in the model. Here we set the values of each γ_j relative to the cost of acquisition of the feature; in the previous scenario γ_j was set to 1 for all the features.

Here the costs assigned to features vary between 1 and 5; 1 is for the least computationally expensive feature and 5 is for the most computationally expensive one. Figure 2 shows the histogram of cost values in our feature set along with some examples in each category.

Considering these features' costs we run lasso for logistic regression. Table 6 shows how features are selected in the lasso path along with the value of AUC. "User Page" whose cost is 1 is the first feature that is selected. As features are added to the feature set AUC tends to increase. However, there is a significant jump in AUC when "DDSR" is added to the feature set, from

Table 6 Cost Sensitive Feature Selection using Lasso. The lasso path shows how features are added incrementally and the changes in AUC.

λ	Feature Set Size	Cost	Newly Selected Features	AUC on PAN test set
0.0882	1	1	User Page	0.7129
0.0882	2		Ins Special Words	0.7581
0.0476	3	1	Revision Ordinal	0.6832
0.0447	4	2	Ins Character Repetition	0.7087
0.0396	5	1	User Type	0.7266
0.0291	6	2	Ins Vulgarism	0.7235
0.282	7	2	Ins Digits	0.8245
0.0265	8	4	DDSR	0.9065
0.0257	9	2	KL Distance	0.9299
0.242	10	2	Ins Bias	0.9305
⋮	⋮	⋮	⋮	⋮
0.0021	32	2	Del Special Words	0.9486
0.0019	33	3	Ins Revision	0.9516
⋮	⋮	⋮	⋮	⋮
0.0014	58	4	Del Words	0.9543
0.0001	59	2	KL Distance Ins	0.9553

0.8245 to 0.9065 . This feature is among the most computationally expensive features but because of its importance it has been selected in the first steps of feature selection. This result is consistent with the results in Table 5 where "DDSR" was selected as the second most important feature. In that case we assumed similar cost for all the features while here we assigned high cost to this feature. The fact that this feature is selected in early in the cost-sensitive feature selection process despite it's high cost, and the large gain in AUC, suggests the importance of this user feature.

After selecting 33 features, AUC is comparable to the AUC of a classifier with 66 features. This means that having about half of the features in the feature set the classification performance is roughly the same. However, in contrast to Section 5.2, here we are training classifiers using a similar number of features *but they are less computationally expensive features.*

6.2 Cost Sensitive Group Lasso

In this section we do cost sensitive feature selection but also take into account the data sources the features are extracted from. We categorize the features into five groups; where features in one group are captured in the same way and based on the same data source. For example, when we process text of history revisions and extract word ownerships, we have the data to calculate features like "DSR", "DDSR", and "Rep". In other words, when we pay the cost of processing text of history revisions to calculate a feature like "Rep" then the cost of calculating a new feature like "DSR" will be zero. Group Lasso [24] captures this notion and provides a way for cost sensitive feature selection.

When a feature from a group is selected, other features of that group will be selected automatically.

We acquire features in five different ways: (1) processing text of all history revisions of articles; (2) processing the text of current revision of articles; (3) processing meta data of current revision of articles; (4) processing sql dumps; and (5) crawling Wikipedia user pages. We process the text of history revisions to track contributions of users in order to acquire some features like "DSR","DDSR", and "Rep". For some other features like "Time Diff", "Category", or "Comment Length" we do not need to process the entire history dump and processing the meta data of the current revisions is enough. We use Wikipedia sql dumps to acquire features like "User Type". We also crawl Wikipedia user pages in order to calculate the value of features like "User Page".

To be able to incorporate group information into our feature selection process, we use group lasso as implemented in the package "grplasso" [24] in R. "grplasso" estimates the weights of a logistic function, considering the group information for different features. Logistic group lasso estimator β can be computed by the maximizing the following convex function:

$$\max_{(\beta_0,\beta)\in\mathbb{R}^{p+1}} [l(\beta_0,\beta) - \lambda \sum_{g=1}^{G} s(df_g)\|\beta_g\|_2] \tag{4}$$

Here, the loss function l is defined as in equation 2 and β_g is the parameter vector corresponding to the gth group of the predictors. The function $s(.)$ is used to rescale the penalty with respect to the dimensionality of the parameter vector β_g, the default value of which is set to $s(df_g) = df_g^{\frac{1}{2}}$ to ensure that the penalty term is of the order of the number of parameters df_g.

Table 7 shows the result. The first group selected is Group 5 which has one feature "User Page". Then Group 4 is added which has one feature, "User Type". Group 1 is added in the third step which has 10 features. All these three groups contain user related features. When Group 3 is added we see a significant jump in the value of AUC, about 16%. It shows the importance of user group features.

The next group which is selected is group 2 which mostly consists of textual features. Adding this group, we see some improvements in the AUC. The last selected group is Group 3 which contains meta data features. The small increase in the value of AUC shows that adding this group of features does not improve AUC significantly.

According to group lasso, user features are the most important features, confirming the results of feature selection experiments from the previous sections. Although these features are computationally expensive to acquire compared to other features, they are selected in early steps of the lasso path because of their significant contribution to classification performance. This is very interesting because in traditional spam detection systems for web content or emails there are generally no user related features. Maintaining a history of user activity in UGC systems is becoming more common. Many wikis maintain

Table 7 Cost Sensitive Feature Selection using Group Lasso

λ	Feature Set Size	Selected Group	AUC
350	1	Group 5	0.7129
330	2	Group 5+4	0.7581
300	13	Group 5+4+1	0.9230
50	45	Group 5+4+1+2	0.9519
10	66	Group 5+4+1+2+3	0.9555

history which makes computing user features possible. But that still opens the question of how to maintain user history in a way that computing user level features is fast and incremental.

7 Discussion and Conclusion

In this paper we described a machine learning approach to detect vandalism in User Generated Content (UGC) systems. Our specific example comes from Wikipedia, but the approach is general and can be applied to many forms of UGC.

Based on the validated corpus of Wikipedia edits from the PAN competition we trained and tested several binary classifiers using learning methods that have been widely used for spam detection: Naive Bayes, Logistic Regression, and SVMs. We also trained models using random forests which have not been widely used for spam detection. Interestingly the results show that the random forests significantly outperform the other three types of classifiers on this problem. An additional benefit of random forests is that they are robust to missing and unbalanced data which is a common characteristic of vandalism/spam data sets.

The common practice when training models for spam detection has been to train the models using all available features. Because our goal is to develop models that are small and fast enough to be used in a real–time vandalism detection tool, we used feature selection to eliminate redundant features so that the computational complexity of the final model is as small as possible while retaining high accuracy. Some of the features that proved to be most informative require mining user action histories. Computing the value of these features can be expensive. Because of this, we considered both traditional feature selection, as well as cost sensitive feature selection that takes into account the cost of acquisition of each feature. We compressed the learned model by estimating the contribution of different features and feature groups to the random forest model. Across the four groups of features we found that each group contained some important features, but the user features representing the history of user contributions are most important, so we could not ignore this group of features. This motivated a focus on individual features to determine which specific features (instead of groups of features) contributed most significantly to the model. Using lasso, we found a minimum set of features

whose classification performance is almost as good as the one with all 66 features. Furthermore, we did cost sensitive feature selection to force learning to prefer well–performing cheap features over more expensive features that yield only slight advantage in classification results. In combination, these techniques help us train a compact, sparse vandalism detection model that scales well and executes in real–time.

The methods we use are not specific to wikis and can be applied to other UGC systems such as blogs, twitter, or social bookmarking systems. The new features we created such as the user reputation features in form of DDSR and usage of special characters in the text were some of the most important features in the random forest model and can be easily extracted in other UGCs. Usage of special characters has been widely used for spam detection in emails or blog comments, but user features such as DDSR which measure the survivability of the content contributed by a user, generally have not been used. This notion of survivability can be translated to other domains as well. For example, for Twitter we can see how often and how fast a tweet gets re–tweeted. Similarly, in Facebook, we can look at patterns of sharing and propagation to measure survivability.

One of the main applications of the methods we develop is for end–user tools that support vandalism detection and reversion. Few of the tools currently in use predict which of the viewed edits is (or is not) vandalism. This is because most predictive models have either been not accurate enough, or have been too slow to use in real–time. The models developed in this paper are more accurate and faster than previously developed models for this task.

Another possible application of this work is as part of a change awareness tool. Most wikis support a watch list. When a user puts a specific wiki page on their personal watch list they are indicating interest in changes that are made to that page. When another user changes the page, email is sent to notify everyone who is watching that page. The amount of email might not be a problem if a user is interested in only a few pages. However, when a user is interested in many pages this can result in an unmanageable flood of email. Our model could be incorporated into the watch list mechanism to allow users to specify a vandalism threshold. That is, a user might want to set different vandalism thresholds for different pages so that they are alerted only when the prediction for a change exceeds the specific threshold they set for that page. This would allow a user to monitor a broader span of pages with reduced workload.

One open issue in our approach is the initial generation of features. In our specific application, we were aided by prior work on Wikipedia. We also identified user features based on user reputation as particularly important for detecting vandalism and suggested modifications of these for application in blog spam detection and for detecting interestingness in systems like Twitter [11]. The importance of user history features is perhaps not surprising, but raises some challenges for UGC systems. User level features require the compilation of user action histories. In some systems this is not kept by default.

One impact of our results suggest that systems which currently do not store user history should reconsider this design decision.

The value of these user level features points out a critical challenge for creating, managing, and analyzing large–scale networked data. First, social and web networks provide a broader range of user level features not considered in our model. These relational features can be considered either individually or in aggregate form. As participation in a UGC system grows the size and cost of computing those reputation (relational) features will grow. This suggests that low–cost, ego–centric and incremental approaches to calculating reputation will be valuable. Our approach provides a technique for analyzing the cost/value trade–off of maintaining user level features.

Vandalism will continue to be a challenge for all UGC systems. What constitutes vandalism will always be something that is in the eye of the beholder. As such, the community of users contributing to UGC systems will always be the final arbiters of what should stay and what should go. Machine learning–based tools provide a promising approach that is accurate, tunable, and maintainable for assessing edits in UGC systems.

Acknowledgements Authors would like to thank Prof. Robert Tibshirani and Prof. Alexander Ihler for their comments on using Lasso and cost sensitive feature selection, and also Martin Potthast for his support in making the PAN data set available to us. In addition, authors would like to thank Amazon.com for a research grant that allowed us to use their MapReduce cluster. This work has been also partially supported by NSF grant OCI-074806.

References

1. crawler4j: A fast crawler in java. URL `http://crawler4j.googlecode.com/`
2. Wikipedia biography controversy. URL `http://en.wikipedia.org/wiki/Wikipedia\biography\controversy`
3. Adler, B., de Alfaro, L., Pve, I.: Detecting wikipedia vandalism using wikitrust. Tech. rep., PAN lab report, CLEF (Conference on Multilingual and Multimodal Information Access Evaluation) (2010). URL `institute.lanl.gov/isti/issdm/papers/vandalism-adler-report.pdf`
4. Adler, B.T., de Alfaro, L.: A content-driven reputation system for the wikipedia. In: WWW '07: Proceedings of the 16th International Conference on World Wide Web, pp. 261–270. ACM, New York, NY, USA (2007)
5. anf Luca de Alfaro, B.T., M.Mola-Velasco, S., Rosso, P., West, A.G.: Wikipedia vandalism detection: Combining natural language, metadata, and reputation features. In: Proceedings of Computational Linguistics and Intelligent Text Processing(CICLing'11), pp. 266–276 (2011)
6. Androutsopoulos, I., Paliouras, G., Karkaletsis, V., Sakkis, G., Spyropoulos, C.D., Stamatopoulos, P.: Learning to filter spam e-mail: A comparison of a naive bayesian and a memory-based approach. In: Proceedings of CoRR, pp. 1–13 (2000)
7. Breiman, L.: Random forests. Machine Learning **45**(1), 5–32 (2001)
8. Caruana, R., Niculescu-Mizil, A.: An empirical comparison of supervised learning algorithms. In: Proceedings of the 23rd international conference on Machine learning, ICML '06, pp. 161–168. ACM, New York, NY, USA (2006)
9. Chang, M.w., Yih, W.t., Meek, C.: Partitioned logistic regression for spam filtering. In: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, KDD '08, pp. 97–105. ACM, New York, NY, USA (2008)

10. chi Chin, S., Srinivasan, P., Street, W.N., Eichmann, D.: Detecting wikipedia vandalism with active learning and statistical language models. In: Fourth Workshop on Information Credibility on the Web (WICOW 2010) (2010)
11. Dong, A., Zhang, R., Kolari, P., Bai, J., Diaz, F., Chang, Y., Zheng, Z., Zha, H.: Time is of the essence: improving recency ranking using twitter data. In: World Wide Web Conference Series, pp. 331–340 (2010)
12. Forman, G.: An extensive empirical study of feature selection metrics for text classification. J. Mach. Learn. Res. **3**, 1289–1305 (2003)
13. Friedman, J.H., Hastie, T., Tibshirani, R.: Regularization paths for generalized linear models via coordinate descent. Journal of Statistical Software **33**(1), 1–22 (2010). URL `http://www.jstatsoft.org/v33/i01`
14. Geiger, R.S., Ribes, D.: The work of sustaining order in wikipedia: the banning of a vandal. In: Proceedings of the 2010 ACM conference on Computer supported cooperative work, CSCW '10, pp. 117–126. ACM, New York, NY, USA (2010)
15. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–1182 (2003)
16. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update **11** (2009)
17. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer Series in Statistics. Springer New York Inc., New York, NY, USA (2001)
18. Itakura, K.Y., Clarke, C.L.A.: Using dynamic markov compression to detect vandalism in the wikipedia. In: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval, SIGIR '09, pp. 822–823. ACM, New York, NY, USA (2009)
19. Javanmardi, S., Lopes, C., Baldi, P.: Modeling user reputation in wikipedia. Journal of Statistical Analysis and Data Mining **3**(2), 126–139 (2010)
20. Javanmardi, S., McDonald, D.W., Lopes, C.V.: Vandalism detection in wikipedia: A high-performing, featurerich model and its reduction through lasso. In: WikiSym '11 Proceedings of the 2011 International Symposium on Wikis (to appear). ACM (2011)
21. Kohavi, R., John, G.: Wrappers for feature selection. Artificial Intelligence pp. 273–324 (1997)
22. K.Seewald, A.: An evaluation of naive bayes variants in content-based learning for spam filtering. Intell. Data Anal. **11**, 497–524 (2007)
23. Martin Potthast Benno Stein, T.H.: Overview of the 1st international competition on wikipedia. In: CLEF'2010 (2010)
24. Meier, L., Geer, S.V.D., Buhlmann, P., Zrich, E.T.H.: The group lasso for logistic regression. Journal of the Royal Statistical Society, Series B (2008)
25. Mishne, G., Carmel, D., Lempel, R.: Blocking blog spam with language model disagreement. In: AIRWeb, pp. 1–6 (2005)
26. Narisawa, K., Ikeda, D., Yamada, Y., Takeda, M.: Detecting blog spams using the vocabulary size of all substrings in their copies. In: In Proceedings of the 3rd Annual Workshop on Weblogging Ecosystem (2006)
27. Potthast, M.: Crowdsourcing a wikipedia vandalism corpus. In: Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval, SIGIR '10, pp. 789–790. ACM, New York, NY, USA (2010)
28. Potthast, M., Stein, B., Gerling, R.: Automatic vandalism detection in wikipedia. In: C. Macdonald, I. Ounis, V. Plachouras, I. Ruthven, R.W. White (eds.) ECIR, *Lecture Notes in Computer Science*, vol. 4956, pp. 663–668. Springer (2008)
29. Priedhorsky, R., Chen, J., Lam, S., Panciera, K., Terveen, L., Riedl, J.: Creating, destroying, and restoring value in wikipedia. In: GROUP '07: Proceedings of the 2007 International ACM Conference on Supporting group work, pp. 259–268. ACM, New York, NY, USA (2007)
30. Santos-Rodriguez, R., Garcia-Garcia, D.: Cost-sensitive feature selection based on the set covering machine. Data Mining Workshops, International Conference on **0**, 740–746 (2010). DOI http://doi.ieeecomputersociety.org/10.1109/ICDMW.2010.92
31. Sculley, D., Wachman, G.M.: Relaxed online svms for spam filtering. In: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '07, pp. 415–422. ACM, New York, NY, USA (2007)

32. Smets, K., Goethals, B., Verdonk, B.: Automatic vandalism detection in wikipedia: Towards a machine learning approach. In: Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI) Workshop on Wikipedia and Artificial Intelligence: An Evolving Synergy (WikiAI08), pp. 43–48. AAAI Press (2008)
33. Tang, Y., Krasser, S., He, Y., Yang, W., Alperovitch, D.: Support vector machines and random forests modeling for spam senders behavior analysis. In: Global Telecommunications Conference, pp. 2174–2178 (2008)
34. Viegas, F.B., Wattenberg, M., Dave, K.: Studying cooperation and conflict between authors with history flow visualizations. In: CHI '04: Proceedings of the SIGCHI Conference on Human factors in computing systems, pp. 575–582. ACM, New York, NY, USA (2004)