An Ensemble Framework for Spam Detection on Social Media Platforms

Junzhang Wang, Diwen Xue, and Karen Shi

Abstract—**As various review sites grow in popularity and begin to hold more sway in consumer preferences, spam detection has become a burgeoning field of research. While there have been various attempts to resolve the issue of spam on the open web, specifically as it relates to reviews, there does not yet exist an adaptive and robust framework out there today. We attempt to address this issue in a domain-specific manner, choosing to apply it to Yelp.com first. We believe that while certain processes do exist to filter out spam reviews for Yelp, we have a comprehensive framework that can be extended to other applications of spam detection as well. Furthermore, our framework exhibited a robust performance even when trained on small datasets, providing an approach for practitioners to conduct spam detection when the available data is inadequate. To the best of our knowledge, our framework uses the most number of extracted features and models in order to finely tune our results. In this paper, we will show how various sets of online review features add value to the final performance of our proposed framework, as well as how different machine learning models perform regarding detecting spam reviews.**

Index Terms—**Feature extraction, machine learning, predictive analytics, spam detection.**

I. Introduction

As information media comes to play more prominent roles in modern society, people's daily lives have become more and more inseparable from, as well as susceptible to, the dispersion of information on media platforms. Nowadays, the medium for communication has shifted away from more accountable resources such as physical newspapers to social media platforms where everyone has a voice. Given the major impact that information can have on not only individuals but small businesses as well, it has become of the utmost importance to ensure the integrity of the information that is being shared, liked, and commented on by individuals on their various platforms.

Spamming - a practice that is used to mislead or deceive social media users by posting harmful links, posting repeatedly to trending topics to grab attention, or posting advertisements - has always been the biggest threat to a healthy social media environment [1]. Despite efforts in the past on combating spam posts and comments online through various means, the patterns for online spam are evolving so rapidly that few spam detection algorithms are robust and adaptive enough to cope. In this project, we aim to design an adaptive and robust online spam detection framework that combines many cutting-edge techniques related to feature extraction, feature engineering, as well as machine learning algorithms - tree-based models, neural networks, statistical models, and so on. Within the framework we propose, we will implement different predictive algorithms and compare their performance on our experimental dataset. We will also show how different sets of features add values to the predictive performance of our proposed spam detection framework. Furthermore, we will also train our framework on smaller datasets to exhibit the robustness of our framework.

Manuscript received December 26, 2019; revised September 1, 2020.

The authors are with Courant Institute of Mathematical Sciences, New York University, New York, NY 10003 USA (e-mail: jw4647@nyu.edu, dx374@nyu.edu).

II. Related Works

Due to the significance of the topic, a considerable amount of research has been done in the field of online spam detection over the past years. In 2010, Wang compared the performance of four predictive models in detecting spam tweets - Naive Bayes, Neural Network, Support Vector Machine, and Decision Tree [2]. In Wang's approach, he described each tweet using user-based features - attributes that are related to the senders of tweets, as well as content-based features - attributes that are related to the text of the tweets. In 2011, Mccord and Chuah compared the performance of Random Forest, Support Vector Machine, Naive Bayes, and k-Nearest Neighbor in detecting spam tweets using similar set of features as Wang did in his design [3].

In 2016, Kaur, Singhal, and Kaur conducted a comprehensive review of the methodologies used in detecting spam on Twitter [1]. In their paper, they investigated both the attributes people used when classifying tweets and the algorithms for conducting the classifications. The paper summarized four main categories of features related to each tweet - user-based features, content-based features, hybrid-based features, and relation-based features. Regarding spam detection algorithms, they listed various types of models that people had investigated during the past - Support Vector Machine, Naive Bayes, k-Nearest Neighbor, Neural Network, Decision Tree, Random Forest, Logistic Regression, and Ensemble Models, all of which have achieved certain levels of success in past Spam Detection studies.

In 2017, Sedhai and Sun proposed a semi-supervised Twitter spam detection model that is capable of adaptive learning through batch mode - constantly retraining the model after a certain time window [4]. Their paper presents a way which enables the spam detection framework to capture new vocabulary and new spamming behaviors, thus making the framework adaptive to deal with dynamic spamming activities. In the same year, Mateen, *et al*. proposed a hybrid approach of detecting spam tweets using content-based and graph-based features [5]. Their findings indicate that combining multiple approaches in spam detection may yield better performance overall.

In [6], researchers proposed using an unsupervised machine learning algorithm in order to detect opinion spam on China's Weibo. They discussed how duplicate user replies and posts can indicate potential spam accounts. However, many non-duplicated reviews can be spam suspects too. Some researchers tried to establish a correlation between spamness (how spam-like a review is) and usefulness. But [7] shows top-ranked reviewers (whose reviews are deemed 'useful' by others) are actually less trustworthy compared to other reviewers.

More recently, many research initiatives attempted to integrate different paths to achieve best efficiency and accuracy. For example, in Extracting Product Features and Opinions from Reviews [8], researchers used sentiment analysis for labeling review text, and then associated the opinions with specific features of the products, which allows for more granular targeting of spam reviews and user accounts. Another group of researchers obtained the best accuracy with a Neural Network Classifier by combining the text-centric and reviewer-centric paths [9]. Besides metadata, recent researchers also have tried to utilize relational data by putting reviewers into a network.

In [10], a new group spam detection algorithm called GSRank was proposed. This algorithm, while yielding encouraging results, requires a labeled group spam dataset which isn't available for most datasets. Therefore, an increasing number of researches in recent years sought to identify a more general solution for spam detection problems.

In 2014, Giyanani and Desai tried to take on spam by using a block diagram that takes into account the email URL, checks it against a blacklist, the number of emails this source sends, then broken down into keywords and passed into a classifier and then finally the NLP engine, which uses N-gram modeling, Word Stemming, and Bayesian classification [11].

In [12], Shankar uses a statistical approach to NLP in order to determine whether or not bulk emails are spam. The purpose of this is to block emails that may be security concerns or have otherwise malicious intents for the user, and the system used is similarly N-gram modeling, Word Stemming, and Bayesian classification. Very similar to [11], the inputs are checked against a blacklist database, then classified and processed through the NLP engine, which is statistical in this case. This interestingly is an attempt to see if it is possible to minimize the massive overhead storage space for databases due to the threshold counter, which is a more traditional way of blocking spam.

Taking a step away from emails, in their paper, Kale, Jadhav, and Pawar focus on customer reviews [13]. Instead of focusing on the extraction, classification, and summarization as previous works have been, however, this paper focuses on finding irregular text flow, vulgar language, and a lack of relation to the content at hand before checking the similarity of that with other comments, which is a different approach than what we have traditionally seen.

Reference [14] focuses on Twitter once again, in order to determine specific spam profiles and thus their posts. They consider four main types of spammers, namely malware propagators, phishers, adult content propagators, and marketers. This eventually results in a three-step process, starting with a comparison of blacklisted URLs, feeding everything through an NLP engine, and then finally machine learning techniques. The NLP engine recognizes very specific phrases and ends with stemming, after which they switch over to the machine learning step if it is not determined to be spam.

Reference [15] once again returns to email spam filtering, in which they focus on content-based filtering rather than the meta-data of the email. Their approach is to split the email into subject and body, as a semi-structured document, and use an interpolated generative model, which involves node probabilities to add weights to the words and understand the document structure.

In our proposed framework, we will integrate and improve upon the many existing methodologies discussed above - feature extraction approaches, predictive algorithms - related to online spam detection, and propose a comprehensive, adaptive and robust online spam detection pipeline. Using the framework we proposed, a close experimental study will be conducted on detecting Yelp spam reviews.

III. Procedure for Paper Submission

Our framework is divided into two major pipelines, which consist of a feature extraction component as well as a predictive modeling component. Between those two main components, some intermediate steps - feature normalization, feature selection, and data sampling - may also be performed. Fig. 1 is the overall design diagram of our framework.

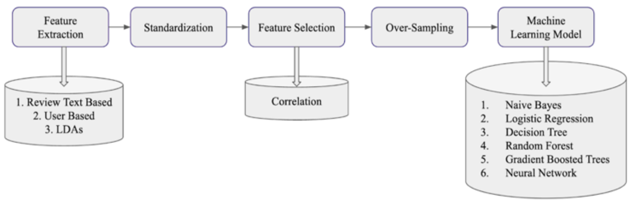


Fig. 1. Design diagram.

A. *Feature Extraction*

Our designed features are divided into two main categories:

(1). Review-Based Features - 287 features that capture attributes as well as patterns regarding the content of the online review.

(2). Reviewer-Based Features - 12 features that capture attributes as well as patterns regarding the user who posts the online review.

Review-Based Features (287)

Table I summarizes the 8 structural features in our framework that are based on the construction of the review including length, word characteristics, and punctuation.

Table II summarizes the 6 sentiment features in our framework that focus on the sentiment of the words in the review, along with a separate score for objectivity. A list of commonly used stop words will be excluded for this part of the analysis.

TABLE I: STRUCTURAL FEATURES

Feature Name	Description
Review_Length	The number of words for each review.
Number_Sentences	The number of sentences for each review.
Average_Word_Length	The average word length for each review, calculated as *Number_of_Chars / Review_Length*.
Average_Sentence_Length	The average sentence length for each review record, calculated as *Number_of_Chars / Number_Sentences*.
Percent_Digits	The percentage of digits for each review, calculated as *Number_of_Digits / Number_of_Chars*.
Percent_Capital	The percentage of capital letters for each review, calculated as *Number_of_Capital_Letters / Number_of_Chars*.
Percent_Exclaimation	The percentage of exclamation mark "!" for each review, calculated as *Number_of_Exclaimation_Marks / Number_of_Chars*.
Percent_Question	The percentage of question marks: "?" for each review, calculated as *Number_of_Question_Marks / Number_of_Chars*.

TABLE II: SENTIMENT FEATURES

Feature Name	Description
Sum_Pos_Score	The summation over all words' positive sentiment scores (determined by SentiWordNet [18], [19]) for each review. If any negation word exists, the subsequent word's sentiment score will be negated.
Sum_Neg_Score	The summation over all words' negative sentiment scores (determined by SentiWordNet [18], [19]) for each review. If any negation word exists, the subsequent word's sentiment score will be negated.
Percent_Extreme_Pos	The percentage of words with extreme positive sentiment score in the review. A threshold needs to be set to determine how positive is extremely positive (any word with positive sentiment score >= threshold is labeled as "extremely positive").
Percent_Extreme_Neg	The percentage of words with extreme negative sentiment score in the review. A threshold needs to be set to determine how negative is extremely negative (any word with negative sentiment score >= threshold is labeled as "extremely negative").
Sum_Obj_Score	The summation over all words' objective scores in the review. Objective score is calculated as 1 - (pos_score + neg_score). A word is said to *be* objective if there is no strong sentimental implication associated with that word.
Percent_Objective_Word	The percentage of words that are "objective" in the review. A threshold needs to be set on each word's objective score to *determine* if that word is considered as an "objective word" (any word with an objective score greater than threshold is an "objective word").

TABLE III: TEXTURAL FEATURES

Feature Name	Description
First_Pronoun_Count	The number of first-person pronouns, such as I, my, myself, we, us, our, ourselves, mine, etc. in the review text.
Second_Pronoun_Count	The number of second-person pronouns, such as you, yourself, your, yourselves, in the review text.
Second_First_Ratio	The ratio of *First_Pronoun_Count* and *Second_Pronoun_Count* as a real number feature. We found that spam reviews tend to use more second person pronouns than legitimate reviews do.
Pos_Tags (36)	The number of each of the 36 POS tags in the review.
Unigram (100)	Firstly, the percentage of each unigram token in spam reviews and the percentage of each unigram token in non-spam reviews are calculated. Then, the top 100 unigrams that have the most different percentages (in spam and non-spam) are taken out, and their counts in the review text are used as features. This process selects the 100 most distinctive unigrams between spam vs. non-spam contexts.
Bigram (100)	Similarly, the top 100 bigrams that have the most different percentages in spam and non-spam reviews are taken out, and their counts in the review text are used as features.
LDA_Distribution (30)	A LDA (Latent Dirichlet Allocation) topic model is fitted to the review dataset, and the topic distributions are used as features. For this project, we set *num_of_topics* to be 30. Normalized TF-IDF vectors are computed in order to carry out LDA modeling.

TABLE IV: METADATA FEATURES

Feature Name	Description
Review_Date	The date in which the review was posted, normalized as a real number feature.
Review_Time_Slides	The number of days between the review posted and the first review given to the same product. We observed that spam reviews are often posted early in order to maximize its influence.
Product_Id	A unique ID for each product in the dataset.
Rating	The numeric rating given by the review, between 1 - 5.

Table III summarizes the 269 textural features in our framework, that analyze the review text using features such as pronouns, parts-of-speech tags, and topic modeling.

Table IV summarizes the 4 metadata features in our framework, that include information about the review such as the date it was posted and the rating given.

Reviewer-Based Features (12)

Table V summarizes the 12 user features in our framework, which take into account aspects of the profile that posted the review, and include features such as the user's rating distribution, average review rating and length, maximum number of reviews a day, and other features that may indicate suspicious behavior.

B. Intermediate Feature Processing

After obtaining the numeric values for all the features through the feature extraction step, it could be beneficial to do some additional processing before feeding the samples into predictive algorithm training.

In our predictive analytics framework, we designed three intermediate steps that could be applied to further feature processing after feature extraction.

Firstly, feature normalization can be done using z-transformation (standardization) to normalize the scales of all features. Since each feature has values of different ranges. Feeding those features directly to machine learning algorithms will produce bias towards features with higher magnitudes, thus affecting the accuracy of the overall framework performance. Therefore, before moving on to other modules of our framework, standardization of all the features could be done using the formula below:

$$z = \frac{x-\mu}{\sigma} \quad (1)$$

In (1), μ is the mean value of the feature, σ is the standard deviation of the feature values. After standardization, each feature will have a mean of 0 and a standard deviation of 1.

Secondly, feature selection using correlation could also be applied to discard features that are relatively uncorrelated with the class label. In some cases, having a high sample dimension can cause the sample space to be sparse, which will hinder the training process of machine learning models due to curse of dimensionality. Therefore, feature selection can be implemented on all the features generated based on each feature's correlation with the class label:

$$r = \frac{n(\sum xy)-(\sum x)(\sum y)}{\sqrt{\left[n\sum x^2-(\sum x)^2\right]\left[n\sum y^2-(\sum y)^2\right]}} \quad (2)$$

In (2), n represents the number of samples in the dataset; x represents the feature value; y represents the class label. The higher the correlation between a feature and the class label, the more that the class label of a sample depends on that feature. Therefore, by applying correlation feature selection, the framework not only reduces the dimensionality of the sample space but also keeps only the features that are most correlated with the class label.

TABLE V: USER FEATURES

Feature Name	Description
User_ID	The identification code for the user who posted the review.
Max_User_Rating	The maximum rating score ever given by the user.
Min_User_Rating	The minimum rating score ever given by the user.
Mean_User_Rating	The mean rating score of the user's ratings.
Median_User_Rating	The median rating score of the user's ratings.
STD_User_Rating	The standard deviation of all the user's ratings.
Num_Ratings_by_User	The number of ratings ever given by the user.
Num_Extreme_Ratings_by_User	The number of extreme ratings (score 1 or 5) ever given by the user.
User_Avg_Review_Length	The average length of the user's reviews in number of words.
User_Avg_Positive_Ratio	The average positive word ratio in the user's reviews defined by the sentiment features.
User_Avg_Negative_Ratio	The average negative word ratio in the user's reviews defined by the sentiment features.
User_Max_Rating_Num_per_Day	The maximum number of ratings the user has ever made in a day.

Thirdly, due to the reason that some spam detection datasets have imbalanced distributions - the proportion of spam samples is often tiny when compared to that of non-spam samples, machine learning models may emphasize their trainings on non-spam samples and thus ignore those spam samples when comes to the final detection stage. To overcome this problem, data over-sampling could be performed and the over-sampled data could be fed into the training process of the predictive machine learning models. Fig. 2 illustrates one of the ways to conduct over-sampling, which is by multiplying the spam data in the training set. However, the downside of this approach is that it may cause the machine learning models to overfit on the training spam data and not generalize well to unseen situations.

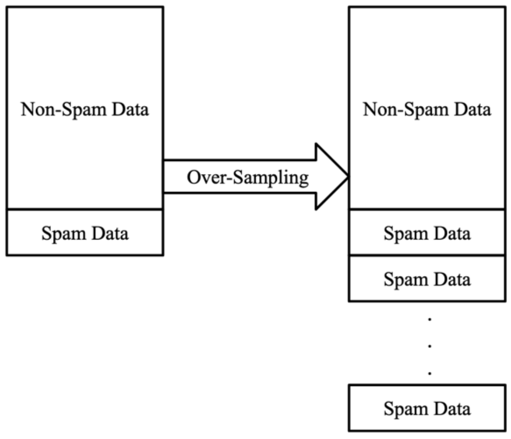


Fig. 2. Over-sampling.

Despite the fact that those intermediate steps may add great value to the ultimate performance of our proposed framework, some of them may hinder the performance of the framework as well. For example, although feature selection helps reduce the complexity of the sample space, it reduces the amount of information contained in the feature space at the same time. Over-sampling, as mentioned above, could also cause the model to overfit. Therefore, it is up to the discretion by the practitioners who implement our framework to decide whether to use any of the intermediate steps or not.

C. Predictive Modeling

After obtaining the numeric features representing each review/post following any applicable intermediate feature processing stages, we feed those values into the last stage of our predictive analytics framework to conduct spam detection. Any machine learning models or any ensemble of models can be used in this step. There is no universal rule regarding which model is the best to use when it comes to spam detection problems due to the case-dependent nature of spam detection. Users of our framework are encouraged to conduct multiple experiments to compare the performance of different models based on the characteristics of their specific problems.

IV. EXPERIMENTAL IMPLEMENTATION

Using the framework we proposed above, we conduct a close experimental study on an existing dataset. In our experiments, we process the data through feature extraction, normalize the features through standardization, and train our framework using 6 machine learning models that were shown to be useful by past studies.

A. Dataset Selection

In our experimental implementation, we focus on one specific platform (Yelp) to test our algorithms when designing our framework. We adopt the YelpZip dataset to test our framework, which originally contained 608,598 restaurant reviews. This dataset was collected by authors Shebuti Rayana and Leman Akoglu in 2015 [16]. Reviews in this dataset include product and user information, timestamp, ratings, and a review text. Yelp has a filtering algorithm in place that identifies fake/suspicious reviews. While this algorithm is not perfect, it has been found to produce accurate results, so we will treat the labels in this dataset as ground truth [17]. In the original dataset, there exist 13.22% spam reviews by 23.91% spammers.

In real-life spam detection applications, however, the amount of data available - especially the amount of labeled data available - is most often inadequate due to the cost and difficulty for analyzing online posts and reviews. In order to simulate such application cases as well as improve the robustness of our framework when applying to smaller datasets, we conduct our experiments on a randomly sampled subset of the YelpZip dataset. Our experimental dataset - the dataset that combines the rotating training and development sets as well as the test set, as shown by Fig. 6 - contains 100,000 samples in total, with 12,264 spam samples and 87,736 non-spam samples. By checking the distributions of attributes in our experimental dataset with that of the original dataset, we affirm that the data distribution for our experimental dataset aligns well with that for the original dataset.

B. Feature Extraction

The definitions and methods for extracting all the features are elaborated in Section III Framework Design. Here we describe some additional implementational details regarding our feature extraction processes.

When parsing the original text into sentences, we used nltk.sent_tokenize(). When labeling our parsed review text, we used nltk.pos_tag().

When determining the words' positive and negative sentiment scores in the review text, we used SentiWordNet as our reference [18], [19].

When setting thresholds for Percent_Extreme_Pos, Percent_Extreme_Neg, and Percent_Objective_Word, we set their thresholds to be 0.5.

Fig. 3 represents the top 100 unigrams that have the most significant differences between the percentage in spam reviews and the percentage in non-spam reviews in our experimental dataset (The size of the token in the figure reflects how big is the difference).

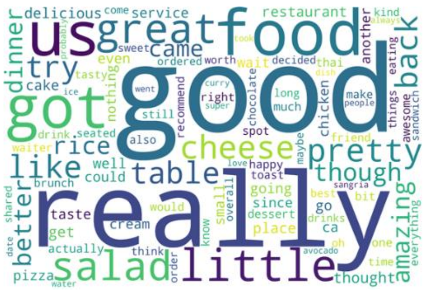


Fig. 3. Unigram Indicators.

Fig. 4 illustrates the cloud of tokens analyzed by LDA when forming LDA distributions on our experimental dataset. Those tokens are the building blocks that form the 30 LDA topic features in our experimental implementation. The key point here is that, while n-grams models only consider the top 50/100/150 words due to limited feature space, LDA takes into consideration all tokens when modeling the topics. Therefore, LDA topics distribution is far more informative than features based on top grams. In the evaluation section, we also observed that after adding LDA features into our framework, we could get a better result in terms of F-1 score.

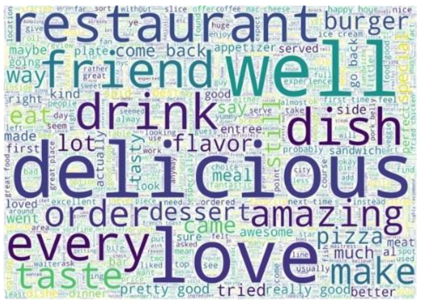
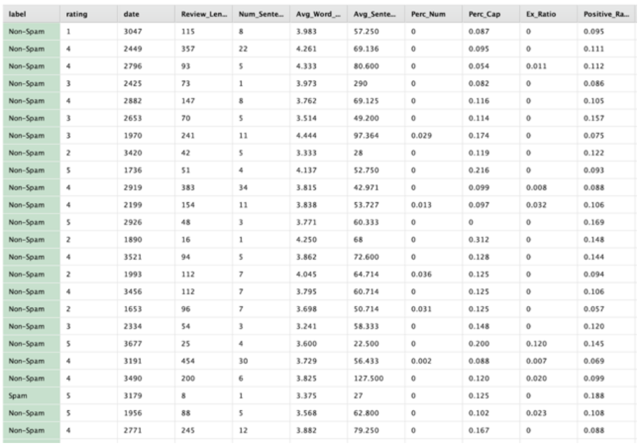


Fig. 4. LDA cloud.



label	rating	date	Review_Len...	Num_Sente...	Avg_Word_...	Avg_Sente...	Perc_Num	Perc_Cap	Ex_Ratio	Positive_Ra...
Non-Spam	1	3047	115	8	3.983	57.250	0	0.087	0	0.095
Non-Spam	4	2449	357	22	4.261	69.136	0	0.095	0	0.111
Non-Spam	4	2796	93	5	4.333	80.600	0	0.054	0.011	0.112
Non-Spam	3	2425	73	1	3.973	290	0	0.082	0	0.086
Non-Spam	4	2882	147	8	3.762	69.125	0	0.116	0	0.105
Non-Spam	3	2653	70	5	3.514	49.200	0	0.114	0	0.157
Non-Spam	3	1970	241	11	4.444	97.364	0.029	0.174	0	0.075
Non-Spam	2	3420	42	5	3.333	28	0	0.119	0	0.122
Non-Spam	5	1736	51	4	4.137	52.750	0	0.216	0	0.093
Non-Spam	4	2919	383	34	3.815	42.971	0	0.099	0.008	0.088
Non-Spam	4	2199	154	11	3.838	53.727	0.013	0.097	0.032	0.106
Non-Spam	5	2926	48	3	3.771	60.333	0	0	0	0.169
Non-Spam	2	1890	16	1	4.250	68	0	0.312	0	0.148
Non-Spam	4	3521	94	5	3.862	72.600	0	0.128	0	0.144
Non-Spam	2	1993	112	7	4.045	64.714	0.036	0.125	0	0.094
Non-Spam	4	3456	112	7	3.795	60.714	0	0.125	0	0.106
Non-Spam	2	1653	96	7	3.698	50.714	0.031	0.125	0	0.057
Non-Spam	3	2334	54	3	3.241	58.333	0	0.148	0	0.120
Non-Spam	5	3677	25	4	3.600	22.500	0	0.200	0.120	0.145
Non-Spam	4	3191	454	30	3.729	56.433	0.002	0.088	0.007	0.069
Non-Spam	4	3490	200	6	3.825	127.500	0	0.120	0.020	0.099
Spam	5	3179	8	1	3.375	27	0	0.125	0	0.188
Non-Spam	5	1956	88	5	3.568	62.800	0	0.102	0.023	0.108
Non-Spam	4	2771	245	12	3.882	79.250	0	0.167	0	0.088

Fig. 5. Snippet of extracted features.

C. Intermediate Processing

We apply only the standardization step for intermediate processing of the extracted features in our experiments. We skip the feature selection step because we would like our experimental implementation to utilize the full power of all the features we extracted from the review dataset and see how different features add values to the performance of the framework. We also do not perform any data over-sampling in our experimental implementation because our experimental dataset is not incredibly imbalanced. In this case, it would not be worth it to apply over-sampling while incurring the potential risk of model over-fitting.

D. Machine Learning Models (Predictive Modeling)

There are various machine learning models that are proven to be useful when it comes to spam detection problems. We pick six of the widely used models to carry out the final component of our framework implementation. By comparing their performances, we conclude which model works the best given our experimental dataset.

Naive Bayes

Naive Bayes, an incredibly popular text categorization method, is one of the simplest Bayesian network models [20]. To construct the classifiers for this technique, the algorithm runs off of a model that will take in a considerable amount of features, and then assume that all of these features contribute independently to a total probability for a class label. This model relies on Bayes' theorem, as illustrated:

$$p(C_k|x) = \frac{p(C_k)p(x|C_k)}{p(x)} \tag{3}$$

This provides the conditional probability of a certain instance to be classified as a specific label by taking the probability of the prior word multiplied by its likelihood divided by existing evidence. This can then be extended into repeated applications of Bayes' theorem. Finally, the chosen classifier can be determined, for a class label y = C_k, as:

$$\hat{y} = argmax_{k\in\{1,\dots,K\}} p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \tag{4}$$

Logistic Regression

Logistic Regression takes in independent variables, which do not have to be binary, then uses a logistic function to output a binary dependent variable for predicting the class label [21]. The logistic function is as follows:

$$f(x) = \frac{1}{1+e^{-x}} \tag{5}$$

The algorithm we used for our model starts by initializing a vector of weights, also known as the regression coefficients, to zero, then we train the selected sample features by calculating a new prediction vector using our version of the logistic regression function, updating the gradient vector, and then finally updating the weights (an iterative process). We minimize the logistic loss through gradient descent. Furthermore, we choose to implement methods that may increase our average training set loss, but decrease the average loss on the test set, a process known as regularization. We do this by adjusting the rate of learning for each gradient update by using a stochastic gradient descent with simulated annealing so that we replace the learning rate constant with that of a gradually diminishing variable rate.

Decision Tree

In a machine learning context, the decision tree is built by weighting a list of features with probabilities, with the ultimate desired output of a class label based on the input variables that consist of the interior nodes [22]. Each tree may be trained by splitting it further into smaller trees and recursively addressing all of the sub-trees and nodes. When the subset splitting no longer adds informational value, the recursion ends.

Gradient Boosted Trees

Generally, boosting in machine learning refers to training the model to learn and weight weak classifiers based off of their accuracy to add to a strong classifying label [23]. For a gradient boosted tree, we use a fixed-sized decision tree as the base, and minimize the loss function to update and better the quality of fit. This improves upon the logistic regression and decision tree models, as it allows for us to use our large number of features without having to input a similarly large number of parameters.

Random Forest

Building off of the various decision trees we have worked with, the Random Forest uses multiple decision trees with each one providing a class label given the input parameters, and the class label that results from the most number of trees will be the final label outputted by the random forest. The randomness comes from the selection of sample data used to train the trees, and the resulting trees are built using a random selection of features [24].

Neural Network

Commonly used in machine learning, neural networks are modeled off the human brain and include components such as neurons, which are similar to the nodes in a decision tree, connections which include probability weights, along with a propagation function that is fed the output of predecessor neurons and then converts it into an input for the next neuron [25]. Neural networks are meant to be hyper-adaptable and to be able to fine tune themselves to the task at hand through repetition and training.

E. Evaluation

Our paper aims to propose a general framework for spam detection on online platforms. As mentioned in Section III Framework Design, due to the domain-dependent nature of spam detection problems, the best configuration of our framework (the best choices regarding the features, the intermediate steps and the machine learning models) may vary case by case. Therefore, instead of proposing specific configurations of our framework in general, we mainly test and compare different framework configurations using cross-validation results, discuss the results' implications, and suggest the configuration that works the best on our experimental dataset in the next section.

As shown in Fig. 6, firstly, we split our experimental dataset into a test set (10% - 10k samples), and a training and development set (90% - 90k samples). The training and development set is then split into rotating training and development sets in cross-validation. Based on cross-validation performances, we pick a framework configuration that works best for our experimental dataset and test it using the test set. We also discuss the general adaptiveness and robustness of our framework as well as different framework configurations' pros and cons by comparing their cross-validation results in the next section - Section V Experimental Results.

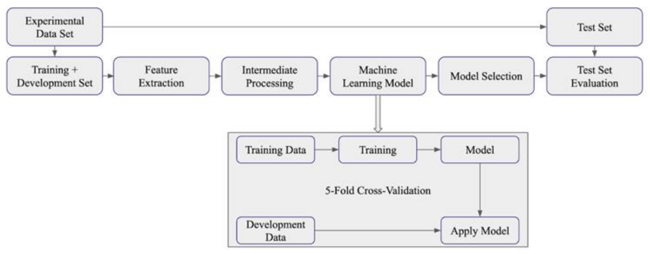


Fig. 6. Experimental implementation + evaluation.

We employ 5-fold cross-validation (rotation estimation) when evaluating our framework performances. Once we obtain the features through feature extraction and the intermediate steps, the training and development set is then randomly split into 5 equal portions with the same sizes and with the same distributions (same spam vs. non-spam ratios). As shown by Fig. 6, within each of the 5 folds, one portion of the set is treated as the development set for model validation while the others are used for training purposes. Thus, for every framework run (per every unique framework configuration), 5 folds of training-validation processes described above are performed. The 5 evaluation results are combined to form the final cross-validation evaluation result that can help estimate the overall performance of the framework configuration.

We adopt overall accuracy, and F-1 score (combines precision and recall) on spam labeled reviews as our performance evaluation metrics.

V. Experimental Results

Table 6 shows our experimental results when evaluating our framework with different learning algorithms during the cross-validation phase (on the 90k training and development samples). Moreover, for each learning algorithm we also train our framework with different combinations of features to show how our holistic approach enhances analytical performance. (Note: Purely Textual Features = Structural Features + Sentiment Features + Textural Features (without LDA Features), User-Based Features = Meta Data + Reviewer-Based Features.)

Based on the cross-validation results in Table VI, we can observe that as we include more features in our framework, the framework performance improves progressively. As shown in Table VI, among all framework configurations that we experimented on, Gradient Boosted Trees combined with all proposed features yielded the best cross-validation performance.

TABLE VI: Cross-Validation Results on 90k Training and Development Set: Framework Configurations Comparison

	Purely Textual Features		Purely Textual Features + User-Based Features		**Purely Textual Features + User-Based Features + LDA Features**	
	Accuracy	F-1 Score	Accuracy	F-1 Score	Accuracy	F-1 Score
NB	76.27%	46.64%	77.82%	48.46%	78.65%	49.23%
LR	89.39%	40.48%	92.63%	67.14%	92.62%	67.05%
DT	87.79%	30.69%	91.27%	64.79%	91.44%	65.09%
GBT	89.78%	50.19%	93.84%	73.62%	94.35%	75.88%
RF	87.79%	1.15%	88.37%	11.66%	92.65%	71.15%
NN	89.46%	44.10%	92.43%	66.93%	93.07%	69.87%

Based on the cross-validation results in Table VI, we pick the best framework configuration on our experimental dataset - ‘All Features + Gradient Boosted Trees’. After testing this configuration using the test set, we obtain the results in Table VII. When comparing our framework’s test performance with that of the state-of-the-art model we have found in [9], in which the model was tested on the same Yelp dataset, our framework outperforms their model in terms of the overall test accuracy ([9] achieves a test accuracy of 81.92%). Our test F-1 score also reaches a level that is close to that in [9] ([9] achieves a test F-1 score of 81.42%).

TABLE VII: Test Results on 10k Test Set: Evaluation on the Best Experimental Framework Configuration

	Accuracy	F-1 Score
GBT + All Features	**94.06%**	**74.72%**

To show the robustness of our framework, we re-run feature extraction, intermediate steps, and cross-validation on datasets with different sizes (all with full features this time). The three datasets we feed into the above procedures are: our entire experimental dataset (100k samples), randomly sampled 50k samples from the experimental dataset, and randomly sampled 10k samples from the experimental dataset (the class distributions within the random samples are kept). Resulting cross-validation accuracies and F-1 scores associated with each dataset can be found in Table VIII. Despite that when acting on smaller datasets, our framework evaluates to lower accuracies and F-1 scores, the performance in general does not vary significantly on datasets with different sizes. The results in Table VIII show that our framework can give decent analytical accuracy as well as F-1 scores even with limited training information.

TABLE VIII: Cross-Validation Results on 10k, 50k, 100k Sets: Robustness of Framework Performance on Small Datasets

	10k Samples		50k Samples		100k Samples	
	Accuracy	F-1 Score	Accuracy	F-1 Score	Accuracy	F-1 Score
NB	76.07%	45.07%	76.49%	47.84%	77.22%	47.65%
LR	91.04%	58.61%	92.29%	66.37%	92.62%	67.04%
DT	89.27%	35.47%	90.58%	62.53%	91.52%	65.51%
GBT	92.82%	68.40%	93.79%	73.67%	94.42%	76.46%
RF	90.41%	39.64%	92.30%	70.43%	92.64%	71.36%
NN	90.22%	54.52%	91.99%	66.95%	93.08%	70.29%

VI. Conclusion and Future Work

There are a lot of studies going on with regards to detecting opinion spam on websites that host reviews. Many of the recent techniques for detecting such opinion spam boils down to one of the two paths: review (textual) centric path and reviewer (meta-data) centric path.

In our paper, we propose a holistic framework that integrates both paths into analytics. Our framework looks at both the review text and the review metadata. For textural features, our framework attempts to find structural / semantic / sentimental patterns that are associated with spamness. On the other hand, in generating meta-data features, our framework looks for suspicious, cross-review patterns. These patterns could be temporal (such as the review_timeslide feature) or relational (such as a user’s rating history). Moreover, we compare the performance of different machine learning algorithms in problem-solving. And in the evaluation section we have shown that our framework works nearly as well as the state-of-the-art model in this field. Furthermore, our framework achieved decent accuracy and F-1 score even on very small datasets, which shows the adaptiveness and robustness of our framework under limited training data.

Our framework is also domain-independent. This means that any site that offers reviews for anything, whether that be their own products, or other businesses, can use what we have proposed to gain insight on suspicious activities in the review section of their domain. Of course, since we took a domain-independent design for our framework, our framework is almost certainly not as strong as individual algorithms that are custom-made to detect spam for specific domains. However, our method thrives in its scalability to multiple domains and its robustness under limited training data, and thus can be used by a wider audience.

In the future, we plan to incorporate more features in our framework such as network-based features that capture the relationship among multiple reviews. We also plan to try to implement more ensemble models in the predictive modeling stage to discover potentially better ways to conduct detection.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Junzhang Wang designed and implemented reviewer-based features, designed and implemented the intermediate processing steps and the machine learning modeling procedures, involved in writing and editing of the paper.

Diwen Xue designed and implemented review-based features, designed and implemented LDA modeling as well as uni-gram modeling, involved in writing and editing of the paper.

Karen Shi designed structural-based features, involved in writing and editing of the paper.

All authors had approved the final version.

Acknowledgment

The authors would like to thank those who had done previous studies in the field of spam detection. This work was inspired in part by their contributions. The authors would also like to thank Professor Adam Meyers from New York University, whose teaching in Natural Language Processing motivated this work.

References

[1] K. Prabhjot, A. Singhal, and J. Kaur, "Spam detection on Twitter: A survey," in *Proc. 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016.
[2] A. H. Wang, "Don't follow me: Spam detection in twitter," in *Proc. 2010 International Conference on Security and Cryptography*, 2010.
[3] M. Michael and M. Chuah, "Spam detection on twitter using traditional classifiers," in *Proc. International Conference on Autonomic and Trusted Computing*, Springer, Berlin, Heidelberg, 2011.
[4] S. Surendra and A. X. Sun, "Semi-supervised spam detection in Twitter stream," *IEEE Transactions on Computational Social Systems*, vol. 5, no. 1, pp. 169-175, 2017.
[5] M. Malik *et al.*, "A hybrid approach for spam detection for Twitter," in *Proc. 2017 14th International Bhurban Conference on Applied Sciences and Technology*, 2017.
[6] Z. Guo, L. Wang, Y. Wang, G. Zeng, S. Liu, and G. Melo, *Public Opinion Spamming: A Model for Content and Users on Sina Weibo*.
[7] N. Jindal and B. Liu, "Opinion spam and analysis," *WSDM*, 2008.
[8] A. Popescu and O Etzioni, "Extracting product features and opinions from reviews," in *Proc. HLT/EMNLP*, 2005, 339- 346.
[9] Z. H. Wang, Y. Z. Zhang, and T. P. Qian, *Fake Review Detection on Yelp*.
[10] A. Mukherjee, B. Liu, and N. Glance, "Spotting fake reviewer groups in consumer reviews," *WWW'2012*.
[11] R. Gyanani and M. Desai, "Spam detection using natural language processing," *IOSR Journal of Computer Engineering (IOSR-JCE)*, 2014.
[12] S. Shankar, "Advanced detection of spam and email filtering using natural language processing algorithms," *International Journal of Advance Research, Ideas and Innovations in Technology*, 2018.
[13] C. Kale, D. Jadhav, and T. Pawar. "Spam review detection using natural language processing techniques," *International Journal of Innovations in Engineering Research and Technology*, 2016.
[14] S. Chorey and R. Sawade, "Detecting spam classification on Twitter using URL analysis, natural language processing, and machine learning," *International Journal of Innovative and Emerging Research in Engineering*, 2016.
[15] B. Medlock, *Investigating Classification for Natural Language Processing Tasks*, University of Cambridge, 2008.
[16] R. Shebuti and L. Akoglu, "Collective opinion spam detection: Bridging review networks and metadata," in *Proc. the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.
[17] W. Karen, "A lie detector test for online reviewers," *Bloomberg Business Week*, 2011.
[18] E. Andrea and F. Sebastiani, "Sentiwordnet: A publicly available lexical resource for opinion mining," *LREC*, vol. 6, 2006.
[19] B. Stefano, A. Esuli, and F. Sebastiani, "Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining," *Lrec.*, vol. 10, 2010.
[20] T. H. Sun, "Spam filtering based on naive bayes classification," *Archive of Research Papers*, Babes Bolyai University, 2009.
[21] A. Nikhila, *Logistic Regression for Spam Filtering*, 2008.
[22] C. Sarit and B.Mondal, "Spam mail filtering technique using different decision tree classifiers through data mining approach-a comparative performance analysis," *International Journal of Computer Applications*, vol. 47, no. 16, 2012.
[23] J. R. He and T. Bo, "Asymmetric gradient boosting with application to spam filtering," *CEAS*, 2007.
[24] B. U. Gaikwad and P. P. Halkarnikar, "Spam e-mail detection by random forest algorithm," *Computer Science & Technology*, Department of Technology, Shivaji University, Kolhapur, Maharashtra, India, 2013.
[25] E. Andrew, *Detecting Spam with Artificial Neural Networks*, 2017.

Junzhang Wang received his bachelor's degree from New York University. His research interests include machine learning, natural language processing, and predictive analytics. He has done other research projects in areas such as anomaly detection and using A.I. algorithms to detect severe clinical cases of COVID-19.

Diwen Xue received his bachelor's degree from New York University. He is currently pursuing a PhD degree at the University of Michigan. His research interests include adversarial Machine Learning and Privacy-preserving analytics.