Proceedings of the Twenty-Ninth International
Florida Artificial Intelligence Research Society Conference

Evaluating Methods for Distinguishing between Human-Readable Text and Garbled Text

Jette Henderson[1], Daniel Frazee[2], Nick Siegel[2], Cheryl Martin[2], Alexander Liu[2]

[1]Institute for Computational Engineering and Sciences, The University of Texas at Austin, 201 East 24th Street, C0200, Austin, TX 78712
jette@ices.utexas.edu

[2]Applied Research Lab, The University of Texas at Austin, 10000 Burnet Rd, Austin, TX 78758
{dfrazee, nsiegel, cmartin, aliu}@arlut.utexas.edu

Abstract

In some cybersecurity applications, it is useful to differentiate between human-readable text and garbled text (e.g., encoded or encrypted text). Automated methods are necessary for performing this task on large volumes of data. Which method is best is an open question that depends on the specific problem context. In this paper, we explore this open question via empirical tests of many automated categorization methods for differentiating human-readable versus garbled text under a variety of conditions (e.g., different class priors, different problem contexts, concept drift, etc.). The results indicate that the best approaches tend to be either variants of naïve Bayes or classifiers that use low-dimensional, structural features. The results also indicate that concept drift is one of the most problematic issues when classifying garbled text.

Introduction

In a variety of cyberattacks, attackers will use automatically generated text that is not human-readable in places where human-interpretable and/or human-generated text is typically expected. We will use the term "garbled" to refer to text that is encoded, encrypted, or otherwise not easily interpreted by a human. For example, an analyst may have no notion of what the garbled text "gzmxltp" means but would be able to understand the text "Mozilla 5.0." Attackers often use garbled text to conceal command and control (C&C) communications. In past attacks, they have obfuscated C&C communications from an infected host using XOR encryption and variants of base-64 encoding and have placed these commands in HTTP request fields such as the User-Agent. To further mask communications, an attacker may also combine an encrypted or encoded message with a conventional User-Agent string. In both of these cases, a human can easily distinguish between the garbled text and the typical human-readable text. However, the number of texts that must be examined for cybersecurity monitoring is too large for manual processing. An automated method for distinguishing between human-readable text and garbled text is therefore necessary to handle extremely large volumes of data.

In this paper, we compare several machine learning approaches for distinguishing human-readable text from garbled text. Additionally, since our goal is to identify garbled text, we take the convention of referring to this text as the *positive* class and human-readable text as the *negative* class.

We use two different cybersecurity problem contexts to evaluate the machine learning methods. The first corresponds to known methods for C&C communication in HTTP header information. The second corresponds to embedding garbled text into tweets.

For each problem, we consider two types of garbled observations. Each type comprises the positive class in a separate set of experiments. The first type of garbled observation consists of completely garbled text, while the second type of observation is a mixture of garbled text and human-readable text. We find that these ways of generating the positive class lead to quite different performances from the classifiers.

Related Work

The problem of differentiating human-readable text from garbled text can be studied as one of two types of categorization problems, one based on the content of the text, and the other on its structure. In the content categorization problem, n-grams extracted from the words make up the feature set. On the other hand, in the structural categorization problem, the structural properties of the text, such as

number of spaces in the text or the length of the shortest word, comprise the features.

Using n-gram features, the categorization problem we pose would be approached as a content-based text categorization problem. In the text categorization community, some researchers have shown support vector machines to be robust text classifiers that are less susceptible to overfitting, and others have had substantial success using simple multinomial naïve Bayes to differentiate spam from normal text (Sebastiani 2002; Joachims 1998). Of particular interest to us is the work of Freeman (2013), who uses a high-dimensional text categorization approach for differentiating between usernames created by humans and those generated by spammers in a social network. In this case, the feature set consists of n-grams of letters of the first and last names of the usernames, and the most successful classifier is a variant of multinomial naive Bayes. Freeman found that n-grams that did not appear in the training set but did appear in the test set were more likely to be generated by spammers. To take the missing test n-grams into account, he modifies multinomial naïve Bayes—a technique we will use in our experiments.

In contrast to high-dimensional classifiers that use the actual text as features, Kartaltepe et al. (2010) use J48, Weka's implementation of the C4.5 decision tree, with a low-dimensional set of features based on the structure of the text to distinguish between human-readable text and garbled text, of which Nazario (2009) found examples on Twitter. The tweets Nazario uncovered were completely encoded text, and in decoding them, Nazario found bit.ly[1] links. There is strong evidence that this behavior is the result of C&C activity. Kartaltepe et al. recreated the behavior of this C&C activity and then trained a decision tree to detect it. They found that J48 performs very well when the negative class consists of human-readable tweets and the positive class consists of completely garbled text. However, they report performance degradation when observations from the positive class are a mixture of human-readable text and garbled text.

In this paper, we test the approaches used by Freeman and Kartaltepe against other competing classifiers (support vector machines and logistic regression).

Features and Tested Classifiers

In this section, we describe the feature sets and classifiers used in our experimentation. Table 1 summarizes all of these classifiers and the feature sets used by each.

[1] bit.ly is a service that maps urls to shorter urls so they can be used in social media that has length requirements.

Feature Extraction

We generate two types of features—high-dimensional n-gram features and low-dimensional structural features.

To generate n-grams for a given string observation s, we extract all possible consecutive substrings of length n from s. We experimented with several different values for n but found, as did Freeman, that 4-grams result in good performance without too much overfitting.

For the structural features of each observation *s*, we record the length of the longest substring within *s*, the length of the shortest string within *s*, and the number of spaces in *s*. In this case, substrings are divided by punctuation or whitespace.

Classifiers Using n-gram Features

We use a total of five classifiers with n-gram-based features. The first and second, MNB-LS and MNB-LS-test, are the standard multinomial naïve Bayes classifiers wherein the only difference between them is that Laplace smoothing is calculated based on both the training and test sets in the latter classifier, but based only on the training set of the former classifier. The smoothing across all features (training and test) for MNB-LS-test means we do not throw away features in the test set that were not present in the training set. In practice, the choice between MNB-LS and MNB-LS-test is dependent on whether the test set is available when training the classifier, which would never be the case in a real-time environment. The comparison allows us to assess the extent to which changes in attack characteristics over time might affect classifier performance. The third classifier is a support vector machine (SVM) with a linear kernel, for which we use a validation set to tune the trade-off between empirical error and margin. Both MNB-LS and SVM are standard baselines in text classification that tend to perform well on standard text categorization problems.

The fourth and fifth classifiers that we test using n-gram-based features are modifications of multinomial naïve Bayes used in Freeman (2013), which give weight to features in the test set that are not present in the training set. Freeman's motivation in making these modifications is the idea that a previously unseen n-gram in the test data is most likely to be in the positive class, so it is important to encode this idea in a feature.

Multinomial naïve Bayes with recursive missing n *-gram Probability Estimation*, referred to herein as MNB-R, is the method Freeman used most successfully to detect fake user names in social networks. To implement MNB-R, we generate a probability for a missing n-gram by replacing it with the recursive product of the conditional probabilities of its constituent $(n-1)$-grams.

We call the second method used by Freeman *multinomial naïve Bayes with missing n-gram feature estimation from a validation set* (MNB-V). This method uses a validation set to estimate a probability for every missing n-gram

Classifier Name		Input Feature Types	
Full	Abbreviated	Textual	Structural
Multinomial Naïve Bayes with Laplace Smoothing	MNB-LS	✓	
Multinomial Naïve Bayes with Laplace Smoothing over the Test Set	MNB-LS-test	✓	
Multinomial naïve Bayes with recursive missing n-gram Probability Estimation	MNB-R	✓	
Multinomial naïve Bayes with missing n-gram feature estimation from validation set	MNB-V	✓	
Support Vector Machine	SVM	✓	
J48	J48		✓
Logistic Regression	LogReg		✓
Filtered Logistic Regression	F-LogReg	✓	✓

Table 1: Classifier Names, Abbreviated Names, and the Types of Features Used by Each

in the test set. To implement this estimate, we add a new feature to the trained classifier, called the "missing feature". We generate the missing feature by splitting the validation set into two sets and then counting, by class, the number of times n-grams appear in the complement of the intersection of the sets. We assign this count to the "missing feature".

Classifiers Using Structural Features

We empirically test both the J48 decision tree and logistic regression (LogReg) when trained on low-dimensional structural features.

Classifiers Using a Combination of n-gram and Structural Features

In preliminary experimentation, we found that only one method using both n-grams and structural features performed well. The approach is a logistic regression classifier where features are selected using forward selection and information gain. In practice, we found that using the top five features resulted in good performance. We refer to this classifier as filtered logistic regression (F-LogReg).

Experimental Set-Up

We use the same general method to construct test, training, and validation data sets for the User-Agent (HTTP headers) and Twitter (tweets) use cases. The test, training, and validation sets consist of observations from two classes. Human-readable text belongs to the negative class, and data that we generate based on knowledge of the use cases are in the positive class.

User-Agent Data

For the negative class, we extract the User-Agent field of HTTP headers from a sample of HTTP traffic in 2015. The entire data set consists of 46,689 unique normal User-Agents. Here, we assume that the set of collected User-Agents does not contain any garbled text.

We generate four sets of positive observations for four separate experiments. In the first data set, which we will call the *Completely Garbled Set*, the positive class consists of strings to which we have applied XOR encryption and base-64 encoding.

The observations in the remaining three garbled data sets consist of a legitimate User-Agent concatenated with an XOR-encrypted, base-64-encoded string. The variation in these data sets reflects the idea that defenders may gather training and test sets at different time points, and between these time points attackers may change the set of User-Agents to which they concatenate garbled text. Attackers could do any of the following between the time the defenders collect the training set and the time they collect the test set:

1. *Mixed User-Agent, Same Set*: Keep the same User-Agents when concatenating garbled text
2. *Mixed User-Agent, Half-Same Set*: Replace some of the User-Agents with new User-Agents
3. *Mixed User-Agent, Different Sets*: Replace all of the User-Agents with new User-Agents

We generate data in the positive class corresponding to the situations listed above by changing how we draw User-Agents for observations in the training and test sets. Let $S_{training}$ be the set of User-Agents from which we draw User-Agents for the positive class in the training set, and similarly let S_{test} be the set of User-Agents from which we draw User-Agents for the positive class in the test set. In User-Agent, Same Set, attackers do not change their User-Agents, so $S_{training} = S_{test}$. In the Mixed User-Agent, Half-Same Set, attackers replace some of the User-Agents with new ones, so we draw User-Agents from

$$S_{test} = \{randomly\ drawn\ subset\ of\ S_{training}\} \cup \{User - Agents \notin S_{training}\}$$

where the first and second sets are both half the size of $S_{training}$. In Mixed User-Agent, Different Sets, attackers start using a completely new set of User-Agents, so we draw User-Agents from S_{test} such that $S_{training} \cap S_{test} = \emptyset$.

Generation of these different datasets is motivated by the idea that the set of User-Agents is continuously changing, evidenced as follows: We built a whitelist detector based on the database of User-Agents from user-agent.org, which was updated most recently in 2011, and we found that thousands of legitimate User-Agents on our network were not listed on user-agent.org. The three data-generation methods for mixed human-readable and garbled text address this idea of drift between the training and test sets.

For each of these four positive classes of data set, we generate 46,689 unique observations, which is equal to the number of observations we collected from actual HTTP requests.

Twitter Data

For Twitter data, the entire negative class consists of a subset of 40,000 tweets sampled from 1.5 million tweets scraped from Twitter.

As in the User-Agent scenario, we generate two types of positive observations for tweets. For the first set, we generate *Completely Garbled* strings. For the second set, we generate positive observations that are a mixture of natural language and garbled text by sampling without replacement from a set of actual tweets and concatenating them with garbled strings while maintaining the 140-character limit of tweets. We call this set of positive examples *Tweets Combined with Garbled Text*. In contrast to the User-Agent scenario, we make each human-readable portion of the positive observations unique by sampling without replacement from the entire set of tweets.

Construction of Test, Training, and Validation

To construct non-overlapping training and test sets for our experiment, we draw samples without replacement such that the combined size of training and test sets is 20,000 observations. We vary the ratio of positive and negative observations in our experiments. To construct the validation set, we perform stratified random sampling of 10% of the training set.

Empirical Execution and Metrics

We executed ten runs of the experiments using each dataset. To quantify performance, we use F1-measure, which is a standard metric used in various machine learning communities (including the text mining community). We also calculated the variance but do not report it in the results tables because it was small in all cases (generally less than 10^{-2}).

Results and Discussion

We present the results of our experiments in Tables 2, 3, and 4. To reduce clutter, we mostly present results for the case where 40% of the data corresponds to the positive class; as we will show in a later subsection, we found these results to be the most representative set of results.

Table 2 shows the F1-measure for the simplest formulation of the classification problem, where the positive class is Completely Garbled text and the negative class is human-readable text. Table 3 shows results for the case where the positive class is a mixture of human-readable tweets and garbled text. Table 4 shows effects of concept drift and will be discussed in a later section.

In general, most variants of naïve Bayes and approaches that use structural features (J48 and LogReg) perform the best. Neither SVMs, F-LogReg, nor MNB-LS perform particularly well overall. We believe that this is due to the presence of n-gram features that occur in the test set but not the training set, a hypothesis we examine in more depth in the next subsection. Note that this problem of features present in the test set but not the training set cannot occur for classifiers that use only structural features. Thus, we believe the classifiers that use only structural features (J48 and LogReg) to be more robust than the classifiers that use n-gram features.

Analysis of the Effect of Differences between Training Set/Test Set Features

In this subsection, we explore the effect of n-gram features that are present in the test set but not in the training set.

In an operational setting, the detector may be applied to test sets that are not available during training. In this situation, it is not possible to apply Laplace smoothing across both the training and test sets (as does MNB-LS-test). In addition, in a properly encrypted string, the distribution of characters is close to uniform. If there are n_a characters in our alphabet, the probability of encountering a string of

Dataset	MNB-LS-test	MNB-LS	MNB-V	MNB-R	J48	LogReg	F-LogReg	SVM
User-Agent	0.994	0.381	0.984	0.998	0.963	0.986	0.571	0.418
Twitter	0.999	0.373	0.994	1.000	0.981	0.978	0.414	0.987

Table 2: F1-measure of Normal versus Completely Garbled Text

Dataset	MNB-LS-test	MNB-LS	MNB-V	MNB-R	J48	LogReg	F-LogReg	SVM
Twitter	0.012	0.011	0.628	0.949	0.951	0.951	0.599	0.238

Table 3: F1-measure of Tweets and Tweets Combined with Garbled Text

Dataset	MNB-LS-test	MNB-LS	MNB-V	MNB-R	J48	LogReg	F-LogReg	SVM
Same Set	0.989	0.970	0.978	0.993	0.971	0.898	0.571	0.965
Half-Same Set	0.558	0.517	0.598	0.683	0.397	0.398	0.571	0.500
Different Sets	0.172	0.105	0.168	0.222	0.095	0.139	0.571	0.062

Table 4: F1-measure of Normal and Mixed Garbled Text for User-Agents

% Positive Class in Training Set	MNB-LS-test	MNB-LS
0.1	0.904	0.103
0.2	0.972	0.224
0.3	0.988	0.310
0.4	0.994	0.381
0.5	0.997	0.996
0.6	0.998	0.998
0.7	0.999	0.998
0.8	0.999	0.999
0.9	0.224	0.999

Table 5: Effect of Class Prior on Two Variants of Naïve Bayes

length n_c is $(n_c^{n_a})^{-1}$. For example, there are 10^{62} possible strings of length 10 comprising the standard lower- and upper-case English alphanumeric characters. This implies that the probability of an encrypted, encoded n-gram appearing in both the training and test sets is extremely unlikely.

The variants of naïve Bayes used by Freeman (i.e., MNB-V and MNB-R) both account for features in the test set that do not appear in the training set, thus avoiding many of the problems that arise due to the low probability of n-grams appearing in both the test and training sets. However, approaches such as SVMs, F-LogReg, and MNB-LS are all susceptible to this issue.

For example, this low probability of a garbled string is problematic in MNB-LS because if the n-grams in a test observation are all missing from the training set, the model will classify the observation based primarily on the priors (i.e., the more dominant class in the training set). We can demonstrate this empirically by specifically creating training and test sets with varying percentages of observations in the positive class.

Let:

- R be the set we are currently constructing (where R can be the training or test set)
- N be the total number of observations in the set
- N_n be the number of observations drawn from the normal data without replacement
- N_m be the number of observations drawn from the entire set of positive observations without replacement

Draw floor($h_n \cdot N_n$) observations from the entire set of negative observations, and draw ceiling($h_m \cdot N_n$) observations from the entire set of positive observations.

As shown in Table 5 (where we compare the performance of MNB-LS versus MNB-LS-test for differentiating User-Agents from completely garbled text in datasets with different positive class priors), when the proportion of the positive class is low, the trained MNB-LS classifier will classify instances with missing features as normal, resulting in poor F1-measure. Once the majority class prior corresponds to the positive class, the MNB-LS classifier will classify instances where all n-gram features are missing from the training set as positive, resulting in high F1-measure.

Thus, approaches that do not account for n-grams that occur in the test set but not in the training set will tend to underperform compared to approaches that avoid this problem.

Analysis of the Effects of Concept Drift

In the experiments where we vary the sets from which test User-Agents are generated (Table 4), we see the effect of concept drift on classifier performance. If we use the same set of User-Agents to generate observations in the positive class (i.e., "Same Set" row of Table 4) then the F1-measure is quite high. Here, the classifier picks up on the features of human-readable portions of the observations in the positive class in the training set and uses these to classify most of the observations in the test set correctly. Note that even MNB-LS does well in the "Same Set" situation. However, as the sets of User-Agents used to generate observations in the positive class drift from one another (i.e., "Half-Same Set" and "Different Set" rows of Table 4), the performance of all classifiers degrades or remains consistently poor. This indicates that concept drift is a particularly difficult problem for the classifiers in our experiments when detecting garbled text.

Conclusion

We have evaluated how well various classifiers differentiate between human-readable text and garbled text. In general, a variant of multinomial naïve Bayes using high-dimensional n-gram features tends to perform the best, although classifiers using low-dimensional structural features are also quite competitive. We examined classifier performance under a variety of conditions including differing class priors, different types of garbled text, and concept drift. The most difficult problem in terms of performance appears to be concept drift. In the presence of concept drift, none of the tested approaches performs particularly well. In contrast, either a variant of multinomial naïve Bayes or a classifier using low-dimensional structural features was able to perform well in the presence of differing class priors and different types of garbled text.

References

Freeman, D. M. 2013. Using Naive Bayes to Detect Spammy Names in Social Networks. In Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security, 3–12. New York, NY, USA: ACM.

Joachims, T. 1998. Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In Proceedings of the 10th European Conference on Machine Learning, 137–42. London, UK.

Kartaltepe, E. J., Morales, J. A., Xu, S., and Sandhu, R. 2010. Social Network-Based Botnet Command-and-Control: Emerging Threats and Countermeasures. In *Applied Cryptography and Network Security*, 511–28: Springer.

Nazario, J. 2009. Twitter-Based Botnet Command Channel. https://asert.arbornetworks.com/twitter-based-botnet-command-channel/ (accessed January 14, 2015).

Sebastiani, F. 2002. "Machine Learning in Automated Text Categorization." In *ACM Computing Surveys* 34(1), 1–47..