Keith Dyer

COSC 4397: Security Analytics

Authorship Analysis for Intrusion Detection using Machine Learning and NLP

**Introduction**

Intrusion detection is one of the most studied security problems, with many papers and researcher’s contributing new methods yearly. Despite this some reports estimate that as much as 50%-80% of the world’s spam is sent from zombie computers, or computer that have been taken control of through an security intrusion (Spring, 2005). One approach that hasn’t been applied as extensively is trying to identify if the outgoing traffic from a computer fits a normal range for that user based on the NLP of the sent messages. Two approaches to this task are building language models such as HMM with semantic analysis, and another is using NLP techniques to build a feature set, and then using machine learning to classify intrusive behavior. In this paper I will discuss and outline how I went about testing this hypothesis for the machine learning approach.

**Dataset**

Originally I wanted to use a collection of email data for the training and testing sets, but after some experimenting with the Enron email datasets I found them to have two problems.

Firstly, the dataset was rather sparse on a user by user basis. Using a frequently cited dataset provided on August 21, 2009 at <http://www.cs.cmu.edu/~einat/datasets.html> I was able to collect 700 email messages. From these I wrote a script to separate them into files by author, using the first “From:” token in the email as the indication of the sender, and their email as their named key. After running this code I found that, when excluding the primary Enron announcement email, the user with the most sent emails in the dataset had only 7 emails to his name. The dataset I need for intrusion detection needs to be large and exhaustive, so that small amounts of intrusive messages can be inserted into it. If the dataset was 8 emails with 7 legit and 1 intrusive it wouldn’t be very conclusive.

Secondly, the emails are raw text are rather poorly formed. There were many nested, forwarded emails which were difficult to parse and remove such that we were only analyzing the characters typed in the keyboard by the user sending the email. In addition, there was lots of non-human written text in the emails such as ID numbers and header information, this again proved a parsing challenge.

My first solution to this problem was to use twitter feeds as a different source of human generated input. From open twitter datasets it is possible to collect thousands of tweets from individual people. I found that this would work, but the character limit and frequent use of hashtags may not correlate to a good dataset. The final solution I settled on was to utilize the social networking site reddit. Reddit is a popular forum board with hundreds of millions of users, many of them with more than 5 years of public history of comments and posts with many thousands of entries. Reddit doesn’t have character limits, yet still has blends of formal and informal dialog depending on which board you post to. In order to get this dataset I wrote a python scrapper to fetch the X most recent comments posted by user Y to reddit using the websites API. For my experiments I fetched 1000 posts from my own reddit user account, and paired it with 100 posts from another random users reddit account. From this base line I pruned 2 entries from the set of 1000 due to formatting errors and was left with a dataset of 1098, in a near 10 to 1 ratio of legitimate posts and hypothetical intrusive posts. The data was fairly rich, containing 7,313 unique words with the following distribution of data.

|  |  |
| --- | --- |
| Metric | Value |
| Average number of words/entry | 43 |
| Shortest entry | 1 |
| Longest entry | 646 |
| Standard Deviation of words/entry | 53.538 |
| Number of Unique Words: | 7313 |

Table Dataset Metrics. The code for metric extraction is DatasetRichness.py

In order to recollect this data, or collect another set of data the data generation python files are in the attached folder. To extract a set of user comments from reddit use the following command line argument:

Python DatasetCollector.py <reddit account name> <number of comments to collect>

These comments will be stored in an xml file with the users name.xml as the file. The users account name must be a valid reddit username. For the dataset used in this paper I used the following code:

Python DatasetCollector.py caedin8 1000

Next to generate the set of 100 intrusive samples I ran the following:

Python DatasetCollector.py superbug 100

Superbug is the username of a random person I found on reddit. I didn’t look at any of his data before querying to eliminate selection bias. I verified that his account had more than 100 posts and then collected it. I put both files in the TrainingData folder. Next, to generate the complete data set to read into WEKA I ran the python dataset generator file pointing to the folder. The DatasetGenerator.py file takes two command line arguments, the first is the path to a folder holding the data, the second is the name of the output file. It will take every item out of the folder and extract them out of the XML format (output from the scrapper) and put them in a single CSV format containing two columns, entry and username. This CSV file can be read into WEKA in order to do preprocessing. For this dataset I ran the following:

Python DatasetGenerator.py TrainingData Dataset1.csv

Using this process many datasets could be generated from Reddit users, in any combination of users and entries by simply calling these programs.

**Pre-Processing**

Now that we have a CSV file with user names as classes and entries as strings I decided to apply some NLP techniques to the data. Luckily the WEKA tool has some built in tools for NLP use. In order to import the CSV file from above into WEKA, go into the GUI and click open file, and then navigate to the file. You will likely receive an error message, instead of clicking OK click “Use Converter” you will need to specify that the first column is a string variable by type by typing in a 1 in the field for “String attributes”. After this press OK and the program should read the data in without a problem. I wanted to apply NLP techniques that we discussed in the class on the data set first, so I chose to apply the StringToWordVector filter in WEKA. In the configuration for this filter I chose to apply the filter to only the first attribute, and then I set “lowerCaseTokens” to true. Next I applied the IteratedLovinsStemmer, the Rainbow stopwords removal algorithm, and the alphabetic tokenizer. The stemmer and stopword removal algorithm are discussed in detail in the WEKA docs, and are based off of research papers in those areas of study. The alphabetic tokenizer creates tokens from the sentences by extracting all contiguous sequences of alphabetic tokens. In other words, every character that isn’t “a-z” is treated as a delimiter. Note: The dataset collectors remove all punctuation from the data to remove the issue of conjunctions. Lastly, I set the outputWordCounts to true, and wordsToKeep to a high value such that no words are lost. Click apply to create the new data set (Found in the submission as Dataset1\_FeatureExtracted.arff). I call this data dataset I. Next I took this data, and decided to run it through a feature selection algorithm. I used two different methods for feature selection. In the first method I used a statistical significance test to find the most likely tokens, this reduced the token count from 5000+ to 324. The other feature selection method was to run a GreedyStepwise search on subsets of the data set using WEKAs tools. This selection resulted in a dataset 59 features. Lastly I repeated the above processes to generate a set of data from the original CSV file, using the same stemmer and stopword removal, but instead generating all unigrams to trigrams of the words as features. I then reduced this dataset using the same statistical feature selection method as above. For reference please see the following papers for the feature selection methods used:

* M. A. Hall (1998). Correlation-based Feature Subset Selection for Machine Learning. Hamilton, New Zealand.
* Amir Ahmad, Lipika Dey (2004). A feature selection technique for classificatory analysis.

|  |  |  |  |
| --- | --- | --- | --- |
| Dataset | File in Submission Folder | Number of Features | Construction Method |
| Dataset I | Dataset1\_FeatureExtracted.arff | 3322 | - Alphabetic  - No Selection |
| Dataset II | Dataset1\_StatisticalFeatureSelection.arff | 324 | - Alphabetic  - Statistical |
| Dataset III | Dataset1\_SubsetFeatureSelection.arff | 59 | - Alphabet  - CfsSubsetEval |
| Dataset IV | Dataset1\_NGrams\_SFE.arff | 4431 | -NGrams(1-3)  - Statistical |

Table Breakdown of datasets used and their construction methods. Under Construction Methods the first point is the tokenizer used, and the second is the feature selection method used.

**Results**

From these four data sets I conducted experiments using the WEKA GUI for many different machine learning algorithms. For all of the following results I report 10-fold cross-validation test results using Accuracy and F-scores as metrics for determining how effective a classifier was in detecting the intrusion. From the collection of experiments I chose to report the results of the following, because they stood out as have the highest F-score for the detection of the intrusive messages, while maintaining a high accuracy rate with low false positives: Logistic Regression, NaiveBayes, NaiveBayesMultinomial, Support Vector Machines trained with Stochastic Gradient Descent, and a Voted Perceptron Neural Network system. All of these classifiers and their functionality are described in detail in the WEKA documentation and are included in the basic WEKA 3.7 package download, so I won’t go into details of how the algorithms work.

For the base data set with no feature selection, the NaiveBayesMultinomial (NBM) performed slightly better than the SVM trained using the SGD. The NBM had an accuracy of 91.62% and an F-measure of 0.954 (Legit class) and an F-measure of 0.549 (Intrusive class). These results were not great, but were the best of the set of attempts using no feature selection. I also tried an SVM trained used SGD. This resulted in 91.71% accuracy, but the F-measure of the intrusive class was only 0.428. The F-score of the legitimate class was 0.955. I tried various other learning methods such as JRip, and PART rule based classification methods, as well as the J48 decision tree, and a random forest with 100 trees and all of these methods had F-measures below the NBM and the SVM. I tried polynomial kernals on the SVM training with orders 1, 2, and 3 and found the best results at order 1, leading me to believe the data is linearly separable. (Up to the results we achieved with the SVM).

**Summary of Results: Dataset I**

|  |  |  |  |
| --- | --- | --- | --- |
| Notable Methods | Accuracy | F-Measure (Legitimate) | F-Measure (Intrusive) |
| SVM with SGD | 91.71% | 0.955 | 0.428 |
| NaiveBayes Multinomial | 91.62% | 0.954 | 0.549 |

|  |  |
| --- | --- |
| Notable Methods | Confusion Matrix |
| SVM with SGD | |  |  | | --- | --- | | 973 | 25 | | 66 | 34 | |
| NaiveBayes Multinomial | |  |  | | --- | --- | | 950 | 48 | | 44 | 56 | |

For the second data set, the two classifiers with above average performance were the NBM again, along with a voted perceptron system. In general the second data set performed the best of all four data sets, across most classifier methods. For the voted perceptron system 369 perceptrons were trained over 10 iterations. The WEKA default of 1 for the exponent and kMAX parameters were used. The Voted Perceptron system on the statistically pruned feature set had an accuracy of 94.26% with only 4 false-positives on the legitimate user’s comments. Of the 100 intrusive comments 41 out of 59 were captured in the 10-fold CV tests. The F-measures were 0.969 for legitimate comments, and 0.566 for intrusive comments. The NaiveBayes Multinomial performed well on the reduced feature set, increasing its accuracy to 96.72% and had 0 misclassifications of the legitimate data. Of the intrusive data set the classifier predicted 64 of the 100 entries correctly. The F-score was 0.982 for legitimate data, and 0.780 for intrusive data. This classifier, feature selection combination had the best results of all the methods tried by me during the project.

**Summary of Results: Dataset II**

|  |  |  |  |
| --- | --- | --- | --- |
| Notable Methods | Accuracy | F-Measure (Legitimate) | F-Measure (Intrusive) |
| Voted Perceptron | 94.26% | 0.969 | 0.566 |
| NaiveBayes Multinomial | 96.72% | 0.982 | 0.780 |

|  |  |
| --- | --- |
| Notable Methods | Confusion Matrix |
| Voted Perceptron | |  |  | | --- | --- | | 994 | 4 | | 59 | 41 | |
| NaiveBayes Multinomial | |  |  | | --- | --- | | 998 | 0 | | 36 | 64 | |

Using the best subset feature selection method with a greedy stepwise search algorithm we obtained the third data set. This dataset performed in general better than no feature selection at all, but not as well as the statistical feature selection. I’ll present the NBM results to compare, as it performed in the top for the two preceding sets, but using logistic regression yielded the best set of results for this data set.

**Summary of Results: Dataset III**

|  |  |  |  |
| --- | --- | --- | --- |
| Notable Methods | Accuracy | F-Measure (Legitimate) | F-Measure (Intrusive) |
| Logistic Regression | 94.44% | 0.970 | 0.573 |
| NaiveBayes Multinomial | 93.90% | 0.967 | 0.504 |

|  |  |
| --- | --- |
| Notable Methods | Confusion Matrix |
| Logistic Regression | |  |  | | --- | --- | | 996 | 2 | | 59 | 41 | |
| NaiveBayes Multinomial | |  |  | | --- | --- | | 997 | 1 | | 66 | 34 | |

For the fourth data set I wanted to try not just word tokens but the ngram models we discussed in class for NLP. I was able to use a build in WEKA filter to split the data into features of unigram to trigram values. This resulted in many hundreds of thousands of values which I was able to use statistical feature selection on to yield the final dataset. There is no ngrams dataset with the subset evaluation feature selection method because the program was taking too long to find the optimal subset of over one hundred thousand features. From this data set the NaiveBayes Multinomial performed similar to dataset II with the addition of 2 more of the 100 intrusive entries are caught, with no change in legitimate entries. The rest of the classifiers performed at or below their values for unigram models.

**Summary of Results: Dataset IV**

|  |  |  |  |
| --- | --- | --- | --- |
| Notable Methods | Accuracy | F-Measure (Legitimate) | F-Measure (Intrusive) |
| NaiveBayes Multinomial | 96.90% | 0.983 | 0.795 |

|  |  |
| --- | --- |
| Notable Methods | Confusion Matrix |
| NaiveBayes Multinomial | |  |  | | --- | --- | | 998 | 0 | | 34 | 66 | |

**Conclusion of Machine Learning**

By applying the NLP methods of stemming, stopword removal, and tokenization we were able to extract a set of features that allowed a machine learning program to detect intrusive behavior in the traffic submit by a user of a computer. In the best of all cases we used a NaiveBayes Multinomial classifier paired with a statistical feature selection algorithm to get a 96.9% accurate classifier, with no false positives and catching 66 of 100 intrusive messages. This system may not be powerful enough in isolation to detect intrusive activity on a network, but it may be a successful tool worth adding to the toolbox of a security professional trying to eliminate intruders and zombies in the organization he is protecting.

**A Natural Language Approach**

The machine learning method was able to fairly successfully classify the authors based on the preceding work, but the machine learning approach leaves out semantic meaning from the phrases, and thus might be further improved upon by more sophisticated natural language methods. In order to test this hypothesis I conducted an experiment using Authorship profiling using the common N-grams (CNG) distance metric. I evaluated this method using three feature sets: words, parts of speech, and synsets from the WordNet library. For each comment in the data set the distance from the normal authorship profile was calculated using the CNG approach and then these distance metrics were used to train a classifier to find and detect the intrusive messages.

**The Process and Model**

The model I used was loosely based off of this diagram:

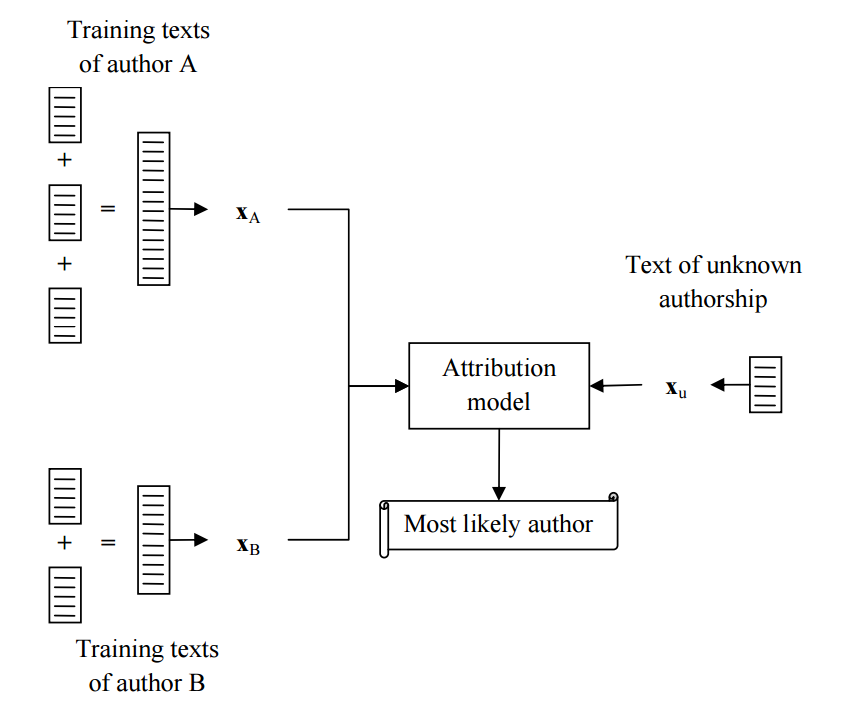


Figure Model of Authorship Profiling

The change made to this model was that in my data set there was no training for a second B author. This is because in a real system employed at an organization you won’t be able to build a data set of intrusive messages to train from. The idea is to find outliers that are so far from the normal that they become detected, not classify them as an intruder already seen before.

A more thorough model of the entire process I actually used is given in the image below. This model shows the data collection from reddit, the authorship profiling phase, the distance calculation phase, and finally the output to WEKA for building the classifier.

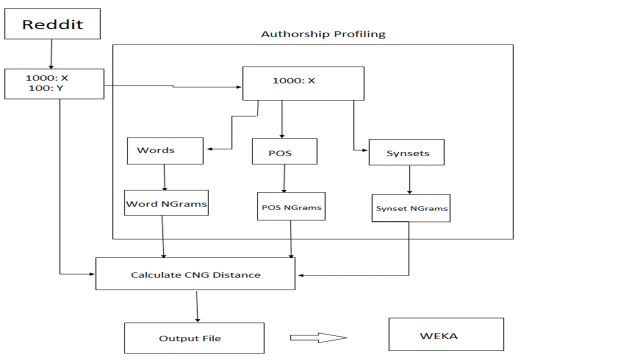


Figure Final model for building the NLP intrusive detection classifier

**Common N-Grams Distance Metric**

The Common N-Grams distance metric was first outlined in Keselj et al, in PACLING 2003: N-gram-based author profiles for authorship attribution. They use the following formula to calculate the distance between any two documents x and y. I used this same metric to train my classifiers using the NLP techniques. I chose to use the top 5000 N-grams from the training corpus of 1000 legitimate author comments.

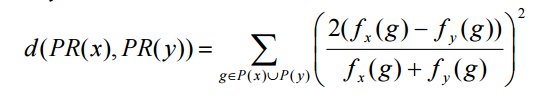


Figure Formula for CNG Distance Metric

**N-grams Comparisons**

I tried three methods of N-grams for the three features words, parts of speech, and synsets. I created a unigram, bigram, and trigram dataset for each of these features and then ran information gain feature selection to rank them in order of importance. From this I found that the Synsets were the most valuable, followed by the words, and lastly followed by the parts of speech. This relationship held true across all three N-gram data sets. Additionally, I found that the trigram data set was better able to classify the data into intrusive and normal comments than either of the other two data sets.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Unigram** | **Bigram** | **Trigram** |
| CNG-WSD | 0.0531 | 0.1207 | 0.1566 |
| CNG-words | 0.0466 | 0.0709 | 0.0839 |
| CNG-POS | 0.0372 | 0.0421 | 0.0458 |

Figure Information Gain Metrics

**NLP Results**

To evaluate the success of the NLP method I ran a 10 fold cross validation on the same original data as the machine learning methods in the previous part of this paper. I experimented with different model n-gram sizes, and different combinations of features. My best data set was the trigram model with all three CNG distances included. I found that using logistic regression classifier on this set of data was fairly accurate, and on par with the average machine learning methods employed in the first part.

|  |  |  |  |
| --- | --- | --- | --- |
| **Notable Methods** | **Accuracy** | **F-Measure (Legitimate)** | **F-Measure (Intrusive)** |
| **Logistic Regression** | 92.81% | 0.961 | 0.470 |

Figure Best Classifier for NLP with CNG Distances

**Further Work**

These experiments were interested to run, and led to some decent results. The problems with this method are that the user and the intruder have no syntactical information extracted from the setting. It is viable to assume that an intruder will behave in a specific manner, and not just in a different manner. Therefore, there perhaps is much room for improvement of this system by including syntactical information from libraries such as WordNet and utilizing HMMs for language analysis. Also sentiment analysis and verb usage may play a telling role in revealing the intruder. The Intruder will likely have an agenda which might be revealed by these NLP methods. There might be better ways to combine the NLP and machine learning methods to build a better classifier. The Synset methods from WordNet appeared to be very useful in the NLP model, and perhaps incorporating that into a machine learning approach with large bags of binary features may outperform both of these techniques.