SecureNLP - Classification of Relevant Sentences to Malware

Prachi Kumar
prachikb@cs.ucla.edu

Shreyas Lakhe
shreyaslakhe@cs.ucla.edu

Abstract

With the increasing number of cyber-security threats and vulnerabilities, it has become important to identify these quickly to take the appropriate action. There is a large amount of malware related information on the web and it becomes difficult for security researchers to go through this information to find the useful content within it. There is a need to be able to extract the highly relevant sentences without having to read through the entire malware reports or text. In our project, we aim to work on classifying sentences as being relevant to malware or not. This is a part of the Semeval 2018 Task 8. We have used deep learning along with other methods for this task. We have tried different approaches and compare the results of our experiments.

1 Introduction

Detecting which sentences are relevant to malware and cyber-security using Natural Language Processing has a lot of potential to benefit security researchers. Surprisingly, not much work has been done in applying NLP to the security domain. In our project, we have tried to classify sentences as being relevant to malware or irrelevant to malware using different approaches. A sentence which is relevant to malware is a sentence describing the malware's actions or capabilities. For example, 'Once decoded, FireEye identified the payload as a poison ivy variant' will not be relevant to malware. But the sentence 'The backdoor contained versioning info which attempted to masquerade as a Google Chrome File' will be relevant to malware as it talks about the actions of the malware. We have focused on applying deep learning, and simpler approaches using Random Forest Classifiers and gradient boosting for comparison. We have used SVM and Naive Bayes classifiers as our benchmarks upon which we are trying to improve further.

2 Related Work

In 'Convolutional Neural Networks for Sentence Classification' (Kim, 2014), the author has trained a CNN with one layer of convolution using pre-trained word vectors. The paper describes how a CNN can be used for sentence classification with different experiments. They train their models on different benchmark datasets. They chose the parameters for their CNN by a grid search over models with different parameter values. We have tried the approach of using a CNN for our malware sentence classification task.

In 'Comparative Study of CNN and RNN for Natural Language Processing' (Yin et al., 2017), the authors compare using a CNN versus using a Recurrent Neural Network for different tasks in NLP. They have used different datasets and different NLP tasks. For our experiments we tried a more specific version of RNN called Long Short Term Memory(LSTM), which is better at modeling temporal sequences and long range dependencies than traditional RNNs. Quoting the paper "Which DNN type performs better in text classification task depends on how often the comprehension of global/long-range semantics is required". For the sentence classification tasks, they observe that CNNs and RNNs produce comparable results when the sentence length is short, but in longer sentences, RNN produces better results than CNN. This was also seen in our experiments, as we tried both CNN and LSTM and got a comparable performance. This is because the longest sentence in our training data was of length 10.

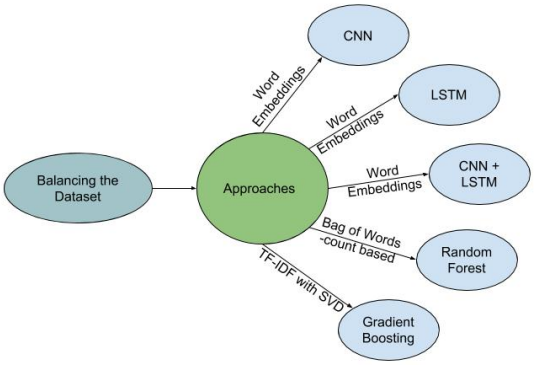

Figure 1: Summary of our Approaches

In 'A c-LSTM Neural Network for Text Classification' (Zhou et al., 2015), they have combined a CNN and LSTM for text classification. We have tried this approach for our task also.

Naive Bayes approach and SVM Classifiers have been used for this Semeval task but with an F-score below 0.7 (Lim et al., 2017). In particular, they got an F-score of 0.6 using SVM, and an F-score of 0.63 using a Naive Bayes classifier. In our project, we have tried to improve upon this further and have managed to get a better F-score.

3 Our Different Approaches

We have used trained word embeddings of length 100 for each of the LSTM, CNN and c-LSTM based approaches . For representing the sentences, we have used a *maxlen* of 100, with sentences lesser than *maxlen* padded with 0s at the end and sentences with length greater than *maxlen* truncated. A summary of our approaches is shown in the Figure 1.

3.1 LSTM

We conducted experiments by testing various architectures for LSTM. The best performance was obtained with a three layer LSTM with 256, 128 and 16 units in the layers respectively and a dropout rate of 0.2. At the end of the network we added a NN layer with 1 unit and sigmoid activation for classification. We have used one hot encoding to encode the words in the sentences along with an embedding layer in the network. We have used binary cross-entropy loss.

3.2 CNN

We tested the performance of various configurations of CNNs. We found out the results by trying combinations of sets of one and two convolutional layers and varying the number of convolutional units from [64, 128, 256], each set followed by a maxpooling layer. We got the best results with an architecture made of two sets of two convolutional layers with 128 filters and a filter size of 5 followed by a maxpooling layer of size 5. At end of these convolutional layers we added a dropout layer with rate 0.5. Finally, we added two NN layers, one with size 128 and relu activation and another with size 1 and sigmoid activation for classification. It gave us an F-score: 0.73, Precision: 0.74, Recall: 0.73 and 81%accuracy.

3.3 CNN + LSTM (c-LSTM)

We used the parameters that gave the best performance on the dataset. We have 2 convolutional layers with 128 filters and a window size of 5, and 2 LSTM layers with 128 and 32 units respectively. We have used binary cross-entropy loss.

3.4 Random Forest Classifier

We have used a bag of words approach, with Count Vectorizer to get feature vectors of the sentences, where it creates a vector of token counts. We have used a Random Forest Classifier with 200 trees.

3.5 TFIDF + SVD + XGBoost

We did text feature extraction from the malware text reports using Term Frequency - Inverse Document Frequency (Tf-Idf) method. After that we used Singular Value Decomposition to reduce the number of features to 100 in the dataset. On these set of features, we applied a gradient boosting method called XGBoost to get the final classifier.

4 Experiments

In this section, we describe the various experiments we have performed. We used Python, Keras, TensorFlow and Sci-kit learn to perform the experiments.

4.1 Dataset

We have used the training data provided by SemEval for Task8. This data consists of 65 annotated documents each containing a different number of sentences. In Figure 2, we have created

Figure 2: Word Cloud for the dataset

a word cloud depicting the frequently occurring words in the dataset. The total number of sentences in all the documents is 9411. However, around 7000 of these sentences are irrelevant to malware, and only around 2000 sentences are relevant to malware. Thus, this data is imbalanced and tends to train models that are biased towards predicting non-relevant sentences. We tried training different models based on this imbalanced data, but the models were predicting nearly every unseen sentence as being non-relevant to malware. The performance of the models was quite poor. For instance, on training a Random Forest Classifier on this imbalanced data, we got an F-score of of 0.2, precision of 0.38, recall of 0.14, and the SemEval leaderboard score of 0.09. We realized that we needed to make our training data balanced to effectively train a model.

To solve our imbalanced training dataset, we have tried two different approaches - undersampling and oversampling with SMOTE (Chawla et al., 2002). These two approaches gave a much better performance in all our models.

4.1.1 Undersampling Approach

We have tried undersampling our dataset. In this approach, we use only a balanced subset of the data for training. We reconstructed our dataset to be made of 4000 sentences, out of which 2000 are relevant and 2000 are irrelevant. Then, we split this dataset to take 75% of it for training and 25% of it for testing. This approach gave the results as seen in the Table 1. We will present an analysis on these results in Section 4.2.

4.1.2 Oversampling with SMOTE Approach

We have tried oversampling our dataset using SMOTE - Synthetic Minority Oversampling Technique(Chawla et al., 2002). In this method, synthetic instances of the minority class (in our case malware relevant sentences) are generated to create a balanced dataset. We have used SMOTE oversampling to expand our dataset size to 14,414 sentences, with half of these as relevant and half as non-relevant instances. In our case, SMOTE generates around 5000 synthetic instances of the relevant sentences using the features of the existing relevant sentences and its nearest neighbors. We split the dataset to take 67% of it for training and 33% of it for testing.

Additionally we also modified our loss function by adding class weights to it. So an instance of positive class was given higher weights than an instance of the negative class.

4.2 Performance of Models

We have used F-Score along with Precision and Recall as our evaluation parameters. We have run our models 3 times each and noted the average values of these metrics over the 3 runs. For every model, we have submitted it to the SemEval Task 8 Leaderboard and noted the score observed on the leaderboard also. The results obtained can be seen in the tables.

Model	P	R	F1	SemEval Score
Random Forest	0.59	0.62	0.61	0.14
LSTM	0.78	0.63	0.69	0.32
CNN	0.74	0.73	0.73	0.31
CNN + LSTM	0.71	0.70	0.70	0.2
TfIdf + SVD + XGBoost	0.76	0.77	0.74	0.41

Table 1: Performance of Models with Undersampling

Model	P	R	F1	SemEval Score
Random Forest	0.88	0.86	0.87	0.12
LSTM	0.91	0.74	0.82	0.20
CNN	0.86	0.80	0.82	0.2
CNN + LSTM	0.85	0.80	0.82	0.21
TfIdf + SVD + XGBoost	0.86	0.93	0.89	0.35

Table 2: Performance of Models with Oversampling

4.2.1 Precision Versus Recall Analysis

We see that in most of our models, the precision is higher than the recall. Precision = TP / (TP + FP),

Results

#	User	Entries	Date of Last Entry	Team Name	SubTask1 ▲
1	PeterP	9	10/30/17	Benchmark	0.52 (1)
2	amilaS	18	10/30/17	Benchmark	0.52 (1)
3	edwinwest	11	12/06/17	Team ENA	0.52 (2)
4	srlakhe	22	12/04/17	DataGeeks	0.41 (3)

Figure 3: Leaderboard standing

while Recall = TP / (TP + FN). So if the precision is higher than the recall, it means that in general our models have a larger number of false negatives than false positives. False negatives are sentences which should actually be relevant to malware, but are being classified as non-relevant. This is challenging because these are usually sentences which are talking about an attacker (maybe by its name) and sometimes these do not contain malware specific terms.

4.2.2 SemEval Leaderboard Score Analysis

We see that although we are getting a good performance on our unseen test data, the models do not have a good score on the Semeval Leaderboard. A possible reason for this is that our models are overfitted to the sentences in our dataset, thus performing well on our test data, but not doing well on the Semeval sentences. Again, in oversampling our data with SMOTE, we may be overfitting the model to our data further, because as is seen in the table, the SemEval scores are generally higher for models trained on undersampled data. For example, we see a Leaderboard score of 0.32 for LSTM with Undersampling, while that model performs quite poorly compared to our other models on our own test data. This suggests that this model could be generalizing better to new data because it is not an overfit.

4.2.3 Oversampling and Undersampling Analysis

The F-scores obtained for undersampling and oversampling on our unseen test data show a higher f-score for the oversampled models compared to the undersampled models. The Random Forest Classifier shows very good performance, with an F-score of 0.87, but again this model performs very poorly on the SemEval leaderboard, which may mean it is overfitted to our dataset.

4.2.4 CNN versus LSTM Performance Analysis

We can see from the results that both CNN and LSTM give almost similar performance on all the metrics. They are able to give good precision recall on the test data because they are able to extract high level complex feature representation and model long term dependencies. As the maximum length of sentence is 10 and number of training sentences is not too high with only 3000 for undersampling and around 9500 for oversampling, the CNN and LSTM have given comparatively less score on the SemEval test data, because the size of the dataset is not too high and they are over-fitting the dataset during training. Additionally they are performing better on oversampling than undersampling supporting the belief that with greater data they will generalize better.

4.3 Conclusion and Future Work

In our project, we tried out different methods and got a better F-score than the baseline approach. We are still lagging a little behind on the leaderboard, and would like to work on this in our future work.

References

Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. 2002. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Swee Kiat Lim, Aldrian Obaja Muis, Wei Lu, and Chen Hui Ong. 2017. Malwaretextdb: A database for annotated malware articles. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1557–1567.

Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. 2017. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*.

Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. 2015. A c-lstm neural network for text classification. *CoRR*, abs/1511.08630.