Enhancing Spam Email Classifier Robustness Against Textual Adversarial Attack

Nishant Dash[1], Francisco Haas[1], Siddharth Kalra[1], GaHyun Yoon[1]
[1] University of Michigan, Ann Arbor
{ndash, haasf, skalra, gahyuny}@umich.edu

1 Introduction

As machine learning and natural language processing (NLP) have become more widespread in real-life applications, their robustness has been called into question. To build resilience against faulty and malicious inputs, adversarial machine learning focuses on understanding how models can be deceived or manipulated by specially designed algorithms (Carlini and Wagner, 2017; Goodfellow et al., 2015; Papernot et al., 2015). The data generated by these algorithms are known as adversarial examples. In NLP, these adversarial examples are generated by perturbing the input text with a variety of techniques, including masking words and characters (Tanner, 2024; Zang et al., 2020; Goyal et al., 2023).

In an attempt to leverage these generated examples for improved performance, adversarial training is the process of explicitly training a model on adversarial examples (Liu et al., 2020; Goodfellow et al., 2015; Madry et al., 2019; Kurakin et al., 2016). Whether increasing model robustness to attacks or reducing test error on clean inputs, including adversarial examples in the training set has been shown to be successful (Goodfellow et al., 2015; Huang et al., 2015).

Our goal is to explore the efficacy of different types of adversarial attacks on a variety of models. We also explored how models can become more robust to adversarial attacks through adversarial training. We will do this by first creating models to perform spam detection on clean data and then see how they perform on adversarial examples. Finally, we will include some adversarial examples in the training set, retrain the models, and compare the performance of the retrained models to the original models on the adversarial test data.

2 Related Work

In recent years, there has been an increasing effort in adversarial attack and defense. Many researchers have explored various types of attacks with different goal functions, constraints, transformations, and search methods (Alzantot et al., 2018; Jin et al., 2019; Kuleshov et al., 2021; Li et al., 2018; Wang et al., 2019).

The field took a great step when researchers published a unified framework that facilitated the development of an adversarial attack and defense pipeline: `TextAttack` (Morris et al., 2020). The framework supports adversarial evaluation with 16 attack methods and streamlines the process of benchmarking, development, data augmentation, training, and fine-tuning. It is flexible regarding model compatibility, as it works with machine learning frameworks like TensorFlow, Scikit-Learn, Keras, etc. It also helped to develop new attack models, such as BAE, which was integrated into the framework shortly after its release (Garg and Ramakrishnan, 2020a).

The following year, `OpenAttack` (Zeng et al., 2021) was introduced, complementing `TextAttack` with sentence-level attacks and parallel processing for better attack efficiency (Zeng et al., 2020). Furthermore, the new framework offered more freedom in the design of attack models and in the optimization of adversarial attacks. In a recent survey, both frameworks evaluated different training, regularization, and defense mechanisms for adversarial attacks in NLP (Goyal et al., 2023). The survey presents a taxonomy of adversarial defense, metrics to evaluate the model's performance after the defense method, and challenges in defending advanced deep neural networks in NLP. Keeping these challenges in mind, we will utilize the aforementioned frameworks and metrics to improve the robustness of the spam email detection

model.

3 Data Processing

In this project, we are working with the Enron email dataset, a dataset which contains emails from the Enron Energy Company obtained by the Federal Energy Regulatory Commission during their investigation. We had previously found a cleaned version of the dataset but through exploratory data analysis we realized the dataset only contained 6 unique spam data points. We then decided to get the original raw data and clean and process it ourselves.

We are using the 2006 version of the dataset, hosted by the Athens University of Economics and Business (Metsis et al., 2006). To build the cleaned dataset, we download and extract each of the spam and ham tar files. We had to disable SSL, which is generally unsafe, but required to download data from the source. We then proceed to parse the SMTP messages for the subject and body of each email, and extract text with Beautiful Soup 4 when the email contains HTML.

After removing duplicates and NA values, our final cleaned dataset contains 18429 Ham data points and 17466 Spam data points. Each data point contains a subject, email content, and binary label indicating spam or ham. Examples of emails can be found in Table 1.

These data are suitable for our task as they are binary-labeled data. After cleaning the data, they are suitable for classification by machine learning models and attack by adversarial example generation pipelines.

Subject	Content	Label
Re: Congratulations	”Thanks. Vince J Kaminski@ECT 01/11/2000 08:01 AM To: Richard Shapiro/HOU/EES@ cc: Subject: Congratulations Rick, I have just looked at the memo regarding promotions. Congratulations - well deserved. Vince”	**0**
Status	”Hi, Did you recieve my email from last week? I’m happy to tell you that you are a p. p roved for a home l o a n with 4.1 Your tracking number is # 99 51 05 You must visit the link below in 24 hrs to confirm your details. rrtzmvidi Best Regards, Michael Beltran Senior Account Officer”	**1**

Table 1: Example emails from the Enron dataset

4 Methodology

A high-level outline of our methodology is to generate adversarial examples and retrain models on those examples to defend against these adversarial inputs with a greater accuracy.

4.1 Using `OpenAttack`

To begin our testing process, we needed to integrate our dataset to work with the `OpenAttack` framework to successfully generate adversarial examples. `OpenAttack`’s framework proved challenging to work with at first, due to it being written for research rather than large scale production, but eventually we were able to start obtaining some concrete data. Like with the data, we had to disable SSL and use a deprecated version of Python and NLTK in order to use some of the attacks.

4.2 Generating Adversarial Examples

After experimenting with the `OpenAttack` framework, we developed a Python script that takes a data point and attempts to alter the input slightly to create an adversarial example close enough to the original, but sufficiently malicious to cause the probabilities of its classification to change when rerun through the classifier model, particularly to classified as the opposite of the input’s original label. While developing this script, we started testing multiple inputs on our processed dataset.
Soon after the training process started, we realized that some of the input text was too long for some of the attacker types, taking more than 30 minutes to generate one adversarial example. Because we only wanted to gauge which adversarial attacker

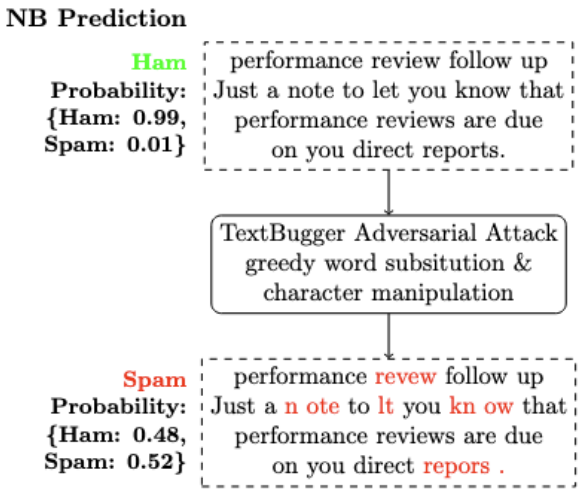

Figure 1: An Example of successful attack using TextBugger: When adversarial input was inputted to the Multinomial Naive Bayes Classifier, the classification result flipped from Ham to Spam.

would be the most effective, we tried most attackers offered by `OpenAttack` library on a smaller dataset. To create this smaller dataset, we only chose emails whose subject and content lengths were less than the median of the dataset (median subject length: 30; median content length: 949). This operation reduced the size of the data set by about 65%, resulting in 12,222 entries. We implemented 80/20 train/test split, randomly chose 50 data points, combined the subject and content into one text string, and retrieved TF-IDF (Term Frequency Inverse Document Frequency) representation of that string. We fed these data points into three models we will discuss in 4.4 to experiment with various attacks.

4.3 Utilizing More Attacks

Once we had developed this initial script, we began testing different attacks that `OpenAttack` offered. Through this process, we checked how many inputs from a dataset of 50 data points were successfully attacked. Our definition of a successful attack was evaluated based on whether the model misclassified the adversarial example. We also measured the run-time of the different attacks to see the efficiency of these adversarial attack generations and help us discern which attacks might work better than others. It should be noted that `OpenAttack` had other metrics that could be useful for evaluation, such as Jaccard similarity at a word or character level, and many others that are accessible in the framework.

4.4 Adversarial Training

Adversarial training is the process of leveraging adversarial examples to make a model more robust (Yoo and Qi, 2021). The process consists of generating adversarial examples and then fine-tuning a model on those misclassified examples.

To test the efficacy of adversarial training, we generate an initial test set of adversarial examples that are misclassified by the pre-trained model. Then, after each epoch of adversarial training, we test the accuracy of the model on the previously misclassified test set.

As we show in 5.2, adversarial training does result in a more robust model. However, there are some limitations. The main limitation of adversarial training is computational efficiency. Because adversarial training leverages the current state of the model and examples must be generated one by one, it is largely not parallelizable. As a result, adversarial training processes, such as the experiment discussed in 5.2, can take on the order of an entire week to run on a subset of data and a few epochs. Additionally, adversarial training may not reduce generalization. For example, adversarially training a model may result in reduced accuracy on clean data. When training on adversarial examples, a model may focus on defending against specific perturbations, rather than on features used to classify clean data.

5 Result

5.1 Adversarial Attacking

Of the 14 attackers offered through the `OpenAttack` API, we implemented 10 different attacks on three different vanilla models: Multinomial Naive Bayes, Support Vector Machine, and Random Forest, and two complex models: LSTM and BERT, a transformer. The four attackers that we did not implement were due to source code error (SCPN), time constraints (BERT, Genetic), and lack of embedding in these models (FD). The success rate and runtime of each attack for the different models are shown in Table 2, Table 3, Table 4, Table 5, and Table 6.

Across the three vanilla models, the Generative Adversarial Network (GAN) model that generates adversarial text by encoder-decoder architecture (Zhao et al., 2018) consistently performed the best, as it always had a success rate greater than 50%. This GAN model architecture is on the complex side compared to other attackers, as it is a black-

box model at sentence-level. Despite its complexity, GAN had a shorter runtime compared to most attack types, which makes the attacker appealing for our task that needs to be able to handle longer text. One caveat is that GAN often produced text that was not intelligible.

Other attackers that performed well were TextBugger (Li et al., 2019) and PWWS (Ren et al., 2019). These attackers access more of the model decision process than GAN as they directly access the score on the generated adversarial examples. While their success rate significantly decreased on Random Forest model as noted in Table 4, these attacks performed very well with the other vanilla models.

Given the results above, we hypothesized that the attackers GAN and TextBugger would perform similarly on more complex models such as LSTM and transformer. The success rate of these attackers aligned with our result from vanilla models as noted in Table 5 and Table 6. While the TextBugger attackers had significantly longer runtime on transformer, the adversarial examples output by the GAN attacker was incoherent and unrealistic; for instance, when a stereotypical work email was attacked, the resulting example that caused misclassification was primarily composed of stopwords and punctuation as such: " and and to to to to a kind of , of that the film of of , be to the , , , the 're n't more of , of the". Therefore, we believe that the TextBugger attacker is the most suited for the spam classification task.

Many of the attacks showed little to no success on the vanilla models. The worst attack in both success rate and runtime was BAE (Garg and Ramakrishnan, 2020b). Since the attack utilizes BERT embedding, we presume the low success rates are due to the lack of embedding-based models in this small experiment. We believe that other low performing models might have similar reasons behind their failure as well.

5.2 Adversarial Training

Due to the large amount of time required for adversarial example generation and training, our results are on a subset of the data (1000 clean test examples, 5000 training). We used the simple Multinomial Naive Bayes as it performed well on binary classification and TextBugger attack as discussed in 5.1.

After training our model on the full training

	Success Rate (%)	Runtime (sec)
BAE	6	2496.24
DeepWordBug	14	13.08
GAN	56	3.33
HotFlip	14	26.39
PSO	10	52.90
PWWS	40	14.48
TextBugger	64	62.26
TextFooler	32	78.28
UAT	6	0.30
VIPER	26	9.11

Table 2: Adversarial Attack Success Rates and Runtime on Multinomial Naive Bayes Model

	Success Rate (%)	Runtime (sec)
BAE	0	2699.56
DeepWordBug	4	43.70
GAN	56	4.11
HotFlip	2	126.64
PSO	2	147.41
PWWS	40	113.23
TextBugger	46	220.36
TextFooler	28	130.33
UAT	0	0.88
VIPER	38	22.42

Table 3: Adversarial Attack Success Rates and Runtime on Support Vector Machine Model

	Success Rate (%)	Runtime (sec)
BAE	2	676.87
DeepWordBug	12	67.30
GAN	54	5.33
HotFlip	14	145.80
PSO	4	223.40
PWWS	18	69.53
TextBugger	22	354.61
TextFooler	14	119.22
UAT	2	1.53
VIPER	44	41.20

Table 4: Adversarial Attack Success Rates and Runtime on Random Forest Model

	Success Rate (%)	Runtime (sec)
GAN	50	24.24
TextBugger	66	38.45

Table 5: Adversarial Attack Success Rates and Runtime on LSTM Model

	Success Rate (%)	Runtime (sec)
GAN	52	3.72
TextBugger	46	724.38

Table 6: Adversarial Attack Success Rates and Runtime on BERT Transformer Model

set, the test set is generated by putting the test set through TextBugger and saving each successful result where the pre-trained model misclassified the adversarial example. For a test set of size 1000 and around 4 seconds to generate each example, this process takes around 1 hour.

Once the test set is generated, we start training. For each example in the original training set, if TextBugger is able to generate an adversarial example that example is then fed back into the model to fine-tune. Table 7 shows results of adversarial training for 9 epochs. As expected, the test accuracy goes up as we train for more epochs, suggesting adversarial training results in a model learning how to defend against previous adversarial attacks. Furthermore, the success rate on adversarial example generation on the training set decreases as we train the model, showing the model becomes more robust as we train on adversarial examples, so much so that TextBugger fails to generate an adversarial example a majority of the time.

Epoch	Test Accuracy (%)	Success Rate (%)
Pre-trained	00.0	53.3
1	90.4	43.1
2	91.4	38.5
3	91.8	35.9
4	92.6	33.4
5	93.2	32.2
6	93.6	31.2
7	94.1	30.9
8	94.5	30.0
9	94.7	29.8

Table 7: Test Accuracy and Training Generation Success Rate for Adversarial Training over 9 Epochs

6 Future Work

6.1 Attacking with Different Attack Types and Improving Runtime

Further experiments could include using the other four attackers that we did not use in 5.1. One could fix the source code dependency error for the SCPN attacker and run our scripts on Great Lakes so that they can experiment with computationally heavy BERT and Genetic attacks.

6.2 Attacking a Bigger Dataset

After assessing the results of the attacks on a smaller dataset, we chose a subset of attackers that performed the best. Rather than attacking just a subset of the data, we could attack the entire dataset and get a more accurate evaluation on each attack using various evaluation methods built in `OpenAttack` framework, such as accuracy under attack, success rate, average perturbed word, average number of words per input, etc.

6.3 Further Adversarial Training

Due to the excessive time required to train adversarially, the current adversarial training results were based on only 6000 total points (1000 test, 5000 train) and still took around a week to run for just 9 epochs.

Future results could consist of adversarial training for a large portion of the dataset, and for more epochs. Specifically, we could take advantage of the over 35,000 total data points included in the dataset. To do so, we could test on 7,000 points and adversarially train on the remaining approximately 29,000.

References

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani B. Srivastava, and Kai-Wei Chang. 2018. Generating natural language adversarial examples. *CoRR*, abs/1804.07998.

Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57.

Siddhant Garg and Goutham Ramakrishnan. 2020a. BAE: bert-based adversarial examples for text classification. *CoRR*, abs/2004.01970.

Siddhant Garg and Goutham Ramakrishnan. 2020b. Bae: Bert-based adversarial examples for text classification. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and harnessing adversarial examples.

Shreya Goyal, Sumanth Doddapaneni, Mitesh M. Khapra, and Balaraman Ravindran. 2023. A survey of adversarial defences and robustness in nlp.

Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. 2015. Learning with a strong adversary. *CoRR*, abs/1511.03034.

Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. 2019. Is BERT really robust? natural language attack on text classification and entailment. *CoRR*, abs/1907.11932.

Volodymyr Kuleshov, Evgenii Nikishin, Shantanu Thakoor, Tingfung Lau, and Stefano Ermon. 2021. Quantifying and understanding adversarial examples in discrete input spaces. *CoRR*, abs/2112.06276.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2016. Adversarial machine learning at scale. *CoRR*, abs/1611.01236.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2018. Textbugger: Generating adversarial text against real-world applications. *CoRR*, abs/1812.05271.

Jinfeng Li, Shouling Ji, Tianyu Du, Bo Li, and Ting Wang. 2019. Textbugger: Generating adversarial text against real-world applications. In *Proceedings 2019 Network and Distributed System Security Symposium*, NDSS 2019. Internet Society.

Xiaodong Liu, Hao Cheng, Pengcheng He, Weizhu Chen, Yu Wang, Hoifung Poon, and Jianfeng Gao. 2020. Adversarial training for large neural language models.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2019. Towards deep learning models resistant to adversarial attacks.

V. Metsis, I. Androutsopoulos, and G. Paliouras. 2006. Enron spam data.

John X. Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp.

Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2015. The limitations of deep learning in adversarial settings. *CoRR*, abs/1511.07528.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. 2019. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1085–1097, Florence, Italy. Association for Computational Linguistics.

Chris Tanner. 2024. Lecture 18: Adversarial nlp.

Xiaosen Wang, Hao Jin, and Kun He. 2019. Natural language adversarial attacks and defenses in word level. *CoRR*, abs/1909.06723.

Jin Yong Yoo and Yanjun Qi. 2021. Towards improving adversarial training of NLP models. *CoRR*, abs/2109.00544.

Yuan Zang, Fanchao Qi, Chenghao Yang, Zhiyuan Liu, Meng Zhang, Qun Liu, and Maosong Sun. 2020. Word-level textual adversarial attacking as combinatorial optimization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6066–6080, Online. Association for Computational Linguistics.

Guoyang Zeng, Fanchao Qi, Qianrui Zhou, Tingji Zhang, Bairu Hou, Yuan Zang, Zhiyuan Liu, and Maosong Sun. 2020. Openattack: An open-source textual adversarial attack toolkit. *CoRR*, abs/2009.09191.

Guoyang Zeng, Fanchao Qi, Qianrui Zhou, Tingji Zhang, Bairu Hou, Yuan Zang, Zhiyuan Liu, and Maosong Sun. 2021. Openattack: An open-source textual adversarial attack toolkit. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 363–371.

Zhengli Zhao, Dheeru Dua, and Sameer Singh. 2018. Generating natural adversarial examples.