THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

The Transferability of Adversarial Samples within Machine Learning

Sheryll Martutartus
Spring 2022

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Patrick McDaniel
William L. Weiss Chair in Information and Communications Technology
and Distinguished Professor Thesis Supervisor

John Hannan
Associate Department Head of Computer Science and Engineering
Honors Adviser

* Electronic approvals are on file (not signatures)

Abstract

Spam detectors used across email providers are largely based on machine learning algorithms. While such detectors are used to segregate the spam emails from the real ”ham” emails, as with any machine learning model, there is the possibility for misclassification due to model inaccuracy. However, attacks on machine learning models are becoming more prevalent, creating input to ”trick” the model and result in purposeful misclassification. In the sense of spam detection, these attacks could be used to have a genuine email sent to the junk folder, or even more detrimental, a spam email bypassing the junk folder and sent to the inbox. In this paper, we explore two types of machine learning methods and their effectiveness in classifying emails. We also experiment with different attacks to generate adversarial samples and trick each of the model. Our findings demonstrate that Naive Bayes models and Neural Networks classify emails with a high accuracy, and we found that an attack generated specifically for Neural Networks, Projected Gradient Decent, has a greater impact on tricking both models versus an attack made for the Bayesian model such as ham-word injection. Lastly, this study interprets what these types of models and tranferability of attacks on these models means in application for the security of email use.

Table of Contents

List of Figures **iii**

List of Tables **iv**

Acknowledgements **v**

1 Introduction **1**

2 Background **4**
2.1 Machine Learning Models 5
2.2 Adversarial Attacks on Machine Learning Models 7

3 Experiment **9**

4 Results **12**
4.1 Naive Bayes Results 13
4.2 Neural Network Results 14
4.3 Cross Model Attack Results 16

5 Discussion **18**

6 Conclusion **20**

Bibliography **22**

List of Figures

2.1 A multilayer neural network illustrating the forward feed of inputs and computation of backpropagation. 6
2.2 Summary of how Projected Gradient Descent perturbs an input sample. 8

4.1 Naive Bayes Model's accuracy decreasing as the number of times the "ham word" is injected into the email. 13
4.2 The most common words found in ham emails (top) and spam emails (bottom). . . 15
4.3 Word Cloud representation of the spam (left) and ham (right) word occurrences. . . 15

List of Tables

4.1 Confusion matrix for predicted outputs of Naive Bayes model. 13
4.2 Confusion matrix for predicted outputs of ham-word injection attack on Naive Bayes model. 14
4.3 Confusion matrix for predicted outputs of Neural Network model. 16
4.4 Confusion matrix for predicted outputs of PGD attack on Neural Network model. . 16
4.5 Confusion matrix for predicted outputs of ham-word injection attack applied to the Neural Network model. 16
4.6 Confusion matrix for predicted outputs of PGD attack applied to the Bayesian model. 17

Acknowledgements

I would like to thank Patrick McDaniel for giving me the great opportunity of being a part of the SIIS lab. He challenged me to think outside-the-box and encouraged me to explore different realms of security. Knowing I have his support makes me want to be a better student and a better computer scientist. I would like to thank Ryan Sheatsley for guiding me throughout my research and for sharing his knowledge of machine learning. He has taught me so much, and I could not have done my research without his help. I thank him a ton for putting up with my *many* Slack messages/spam. I would also like to thank Blaine Hoak for being a great role model. Ever since I became a part of the SIIS lab, I always looked up to her and admired her intelligence and work ethic. Especially for woman in STEM, it is great to have a role model such as Blaine to build one's confidence in this type of field. She is always willing to take time out of her day to help, and her positivity and passion for computer science is contagious. Overall, I would like to thank everyone in the SIIS lab. Whether they were a part of my research or not, I could not have asked for a better group of people to share my love of computer science and security with. Just walking into the lab, I am instantly put in a better mood because of how friendly and supportive everyone is. Because of their level of intelligence and success that they have achieved, they are always a reminder to myself of why I chose computer science and push me to challenge myself more and explore more possibilities. Lastly, I would like to thank coffee in general. My hours of coding and debugging could not have been possible if I wasn't loaded up on caffeine.

Chapter 1

Introduction

With the rapid growth of technology throughout the 21st century, electronic mail (email) has become a major means of communication. Whether it is for career purposes or for personal use, a majority of people have an email address, or even multiple addresses, these days. Given how pervasive it is and how easy it is to get a hold of one's email address, it is now more important than ever to make sure our inboxes are secure and that they prevent any unwanted messages. However, many of these unsolicited emails, or spam mail, continue to show up in our inboxes. Spam mail, as opposed to legitimate emails referred to as "ham mail," has become a large security concern especially when it made up about 48% of all emails sent in 2020 [3]. Though many people think of spam mail as annoying emails sent to try and sell someone a cheap product or service, they have evolved to be much more deceptive and dangerous to its recipients. Attackers can create spam messages as a decoy to trick recipients into downloading malicious software or releasing their personal information. The dangerous part is that they appear to be like any other ordinary email. Malware, malicious software, can be dangerous to any device especially large corporations, and yet about "94% of malware is delivered via email" [3].

Given that spam mail has evolved to appear more like legitimate emails, it can be difficult to distinguish the spam mail from the ham mail. Popular email providers such as Gmail and Outlook have devised a means for filtrating the spam emails and dumping them into the spam or junk folder separate from the inbox. With the growth of AI, email providers have developed and incorporated machine learning models into their spam filters. They analyze thousands of emails, both spam and ham, in order to build upon their current models and increase the accuracy in which they classify an email. And as spam messages continue to evolve, these filters must also too as they continuously create new rules in which to classify emails; email providers even take into consideration which emails users manually send to the junk folder [7]. The most common types of machine learning models used in spam detectors are Naïve Bayes, Support Vector Machines (SVMs), and Neural Networks [7]. For our experiment, we will focus on only two, Naive Bayes and Neural Networks. The Naive Bayes model is typically the first model thought of for spam detection because of its incorporation of conditional probabilities in its predictions, which is why it is often used in prior studies on spam detection [17]; there is even a machine learning model of Naive Bayes specifically made for spam detection called SpamBayes. We also chose to use the Neural Network model in our study because of its reputation in accurate predictions and because popular email providers such as Gmail and Outlook utilize those types of models in their system [7]. While there are multiple means by which they could dissect an email (IP address of sender, header info, blacklist, attachments, etc.), content-based filtering is most widely used as input to the models as it analyzes the word distribution of the email in terms of words used, frequency, capitalization, spacing, punctuation, etc [17][16].

Despite these major tech companies and their advanced machine learning applications, the inaccuracy of AI pose a large security threat to email users if they receive missclassified emails. Machine learning models, though tend to have a high accuracy, can also make mistakes in classification, and in terms of email filtration, could mean that a legitimate email is sent to the spam folder, or worse, a spam message bypasses the spam folder and ends up in the inbox; this instance of classification error is where we are most concerned with as it poses the great security threat. Aside from scientific errors, there are also ways in which on can purposefully trick the model into misclassification. These types of attacks on machine learning models are called adversarial samples in which attackers perturb input data to create a purposeful misclassification [12]. Attackers create these adversarial samples by sending loads of emails and analyzing how the model deals

with them; they mainly focusing on which ones get classified as spam vs. ham and use this to their advantage to construct emails that will bypass the spam filter. In doing so, it will gradually skew the spam filter and degrade the accuracy of the machine learning model, letting more and more spam emails and altering the rules of classification and word distribution [7]. The most common machine learning models used for spam classification, Naïve Bayes and Neural Networks, can fall victim to adversarial attacks. Since Naïve Bayes is a generative model, "trained to maximize the likelihood of the training data" [17], they are susceptible to attacks given their "poor representation power. . . [and] single (component) density to represent spam" [18]. Since the Bayesian model's learning process focuses on the probability of an event/feature occurring, it is easy for an attacker to make slight changes to the input and change the conditional probabilities of a feature, resulting in mislabeling of the sample [18]. Similarly, Neural Networks are also susceptible to adversarial attacks. Neural Networks are not based on probability like the Bayesian model is. Their learning focuses on updating the weights of the model based on the amount of loss gathered through its predictions. Since Neural Networks are gradient-based models, this makes it easy for attackers to apply small perturbations to the input in order to affect the model's gradient and change the weight of the features [4]. With this in mind, it is possible for attackers to exploit the models' weaknesses and create adversarial spam emails to bypass the filter.

Given these possible attacks, we aim to explore how adversarial emails impact different machine learning models and their email classification. Focusing on the Naïve Bayes model and Neural Network, we use these attacks to create adversarial samples and exploit the models. We then aim to explore if there exists a transferability between the models in which adversarial emails generated by one model can be used to trick the other model. Given that these two models are operate and "learn" through different means, there is a possibility that the generated samples can be used to trick either both models or that they only operate one-way. More so, we aim to discover which features of the input the models focus on and which aspect(s) of their learning methods play an important role in their classification. It is believed that the adversarial emails generated by a Naïve Bayes attack can be used to trick the Neural Network, but the adversarial emails generated by the Neural Network will not impact the accuracy of the Bayesian model; therefore, only operating in a one-way attack. With the attack created specifically for the Bayesian model, we aim to inject the word most commonly found in daily emails multiple times throughout each email sample in hopes of throwing off the classifier and having spam emails bypass the spam detector. Because this word injection will change the probability of word occurrences in emails, it should prove to be effective in tricking the Bayesian model, and in terms of the Neural Network, we believe that increasing a word count ratio within an email would throw off the weights of the network, creating a misclassification. Conversely, while a Neural Network solely operates through the calculation of its gradient and updating of weights to reduce loss, an attack such as Projected Gradient Descent (PGD) can be effective in updating the weights toward the direction of greater loss [4]. Because Naïve Bayes focuses on word count and contents of the email, it is believed that a PGD attack, one operating on the changing of gradients, would not be effective in tricking the Bayesian model, and therefore, having the transfer of adversarial samples created only successfully tricking one model but not the other. Overall, the results of this experiment will explore the transferability of adversarial samples and lead way into the discussion of these models sharing the same latent spaces.

Chapter 2

Background

2.1 Machine Learning Models

Machine learning in general is a subset of Artifical Intelligence in which we utilize an abundance of data to feed through algorithms and make predictions about future data trends. These algorithms analyze aspects and features of the data we input to find any patterns or trends; this is called training the model. While there are many different types of models, ranging from classification models to regression models, they all aim to learn about the patterns in the data in order to make successful predictions on future inputs. From there, we analyze the success rate of their predictions and use this loss to better improve upon the model, therefore, having it "learn".

When it comes to building spam detectors, much research has been brought to analyze which models are most effective in their spam classification. [19] created their own spam filter through the use of an Artificial Neural Network (ANN) and compared the results of the ANN to that of a Naïve Bayes model on the data. They discovered that the ANN resulted in a better accuracy classification (93%) than their Naïve Bayes model (88%); however, because of the complexity of a Neural Network, they also found that the ANN took longer to run than the Naïve Bayes model. [14] also built a spam filter using a Neural Network and found that their results were more accurate than when using other models. [6] compared the use of Naïve Bayes models with Decision Trees to see which performed better in spam classification; they found that while Decision Trees' predictions were more accurate than that of Naïve Bayes, the Bayesian model proved to be more robust and effective when used on other datasets.

Of all the machine learning models, we chose to use two commonly utilized models in spam detection, Naïve Bayes Model and Neural Networks.

The Naïve Bayes Model is based on Bayes' theorem and the theorem of total probability. From this, we get the conditional probability of an event occurring given other events in the set. In relation to machine learning, the Bayes model describes the joint probability over features X_i and a class C, assuming every feature is independent from each other given the class it belongs to [17]:

$$P(X, C) = P(C) \prod_{i=1}^{n} P(X_i|C)$$

Here P(C) represents the probability of a class occurring, while $P(X_i|C)$ is the conditional probability of feature X_i occurring given that it belongs to class C. This method of conditional probability can be applied to spam filtration to determine the probability that an email is classified as spam (or ham) given its multiple observed features [16]:

$$P(S|W) = \frac{P(W|S) * P(S)}{P(W|S) * P(S) + P(W|H) * P(H)}$$

In this equation, $P(S|W)$ is the probability that an email is spam given that it contains a certain word W. It is found by taking the joint probability of multiple words (our features in this case) to their association with spam and ham emails.

While much of prior research on spam detection utilizes the content of the email such as which words are used and how often, there are also studies in which they examine the email structure and header information in consideration for its classification. [22] took an approach to analyze the headers and syslogs of emails in order to determine spam classification, while [10] used a hybrid

approach to incorporate both the sender's IP address and the email content as input into their models. They believe that as the machine learning models evolve to learn which words trigger spam detection, spamming behaviors will evolve as well. [9] agrees that word count distribution within an email, or as they refer to it as Bag-Of-Words, has its weaknesses, and so they attempt a hybrid approach in which they not only analyze the word count, but the positioning of words throughout the email and the comprehensiveness of the message. Such research could lead us to creating models with hybrid inputs of analyzing both the email structure (header, etc.) and its content to test its efficiency in predicting emails, given that spam continues to evolve over the years.

While spam filters can be built to analyze multiple features such as sender IP address, email header info, attachments, etc., our focus will be on the word distribution within the email. By gathering a count of each word within an email, we can use Bayes' theorem to build a relationship between words and their classification. For example, if a specific word is found more often in spam emails, then we can use this knowledge in the future to see that if an email contains this word, there is a high chance it is spam. In order to get the word count from each email sample, we utilize scikit-learn's feature, CountVectorizer. This function takes in a text document, or an email, and outputs a matrix of token counts corresponding to each word in the text document. We will be using this matrix of word counts as input to both our Naïve Bayes model and Neural Network.

Neural Networks are widely used within machine learning for their complexity and accuracy in predictions [12]. For training purposes, the data is fed through layers of neurons, and through the forward feed, they perform computations based on the weights and biases of the neurons and the chosen activation function. The weighted value resulting from all these computations is used in determining the classification of the object. Depending on how accurate this classification is, we need to learn the gradient of the network through the given loss function, propagate backwards through the neural network, and update each neuron's weight to improve the network's accuracy [12]. With more iterations of this process and more data inputted to the network, we can settle on weights that provide the greatest accuracy for each sample.

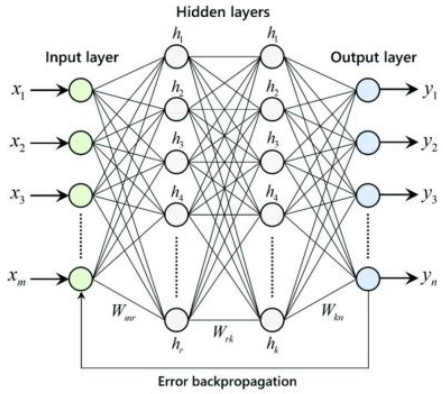

Figure 2.1: A multilayer neural network illustrating the forward feed of inputs and computation of backpropagation.

2.2 Adversarial Attacks on Machine Learning Models

Adversarial attacks can be applied to machine learning models to degrade the accuracy of the output. In terms of the Bayesian Model, the main type of attack is done through data poisoning. The attacker will generate multiple emails and exploit the structure of the spam filter by sending in large quantities of emails to observe their classification. They will use this information to observe the patterns of the constructed emails and their correspondence to the classification. Essentially, they begin to learn which words are "spammy" or most likely found in spam emails [18].

Within data poisoning, there are several different types of attacks based on how much knowledge the attacker has of the spam filter and its user. In a passive attack, the attacker knows little to nothing about the word classification and has to make an "educated guesses regarding which words are 'good' and which are 'bad'" [17]. One type of passive attack is called a dictionary attack in which a set of words is chosen from the dictionary and inserted into the email to observe its classification that was tested out by [17]. However, their results have shown that this attack has been known to make the email "more spammy" given the insertion of random words [17]. Conversely, an active attack is when the attack has some knowledge of the spam filter and the word distribution and can send messages to discover more of the word list.

Previous research on spam models such as [16], [18], and [17] have also used Naive Bayes models and have found successful ways in which to perturb the input and attack the model. [16] focused on three techniques to attack the Bayesian model: synonym replacement, spam word spacing, and ham word injection. They found that replacing certain words of the email with their synonyms had little influence in changing the model's classification. With spam word spacing, they took the commonly found spam words within an email and added spaces in between each letter; they found this technique to bypass the spam filter about 60% of the time. In their last attack technique, ham-word injection, they found out that by injecting commonly found ham words into spam emails, they could shift the classification of the email; they also found that by replacing abbreviated words such as "U" and "UR" with the correct spelling, it also shifts the spam email over to a ham classification. Inspired by their attack techniques, we will be using a ham-word injection attack in which we take a list of words most commonly associated with legitimate (ham) emails and insert them into the email samples during training of the model. As opposed to using spam-word injection, ham-word injection is aimed to have spam emails bypass the spam filter by making them appear to be more similar to legit emails.

In terms of attacks on Neural Networks, much research has gone into examining the altercation of the model's gradient to shift it in a direction of high loss. Attacks on Neural Networks are commonly studied through images because while pixel perturbation cannot be seen to the human eye, it can be used to trick the model [12]. [1] has also studied the effect of Neural Network attacks, specifically focusing on PGD and FGSM and its effect on a MNIST dataset. [5] explores multiple types of attacks on Neural Networks such as L-BFGS, FGSM, JSMA, and Deepfool, describes the differences in the way each method attacks the network, and then compares the results of each attack to discover which one(s) are more effective. Finally, [4] invites us to explore the transferability of black-box attacks on Deep Neural Networks and have shown that they just as effective results as a white-box attack would on a MNIST dataset. This inspires us to further investigate how such attacks on Neural Networks (and other types of machine learning models) can be transferred across different types of classifying models.

For the Neural Network, it will require an attack on the model's gradient. While there are multiple types of attacks that can be applied to Neural Networks because of the accessibility to the model's gradient, we will be focusing on the Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD). Gradient descent attacks focus on altering the gradient of the model to "minimize the output function and moving towards the steepest descent" [20]. FGSM is one of the first methods used for generating adversarial examples, and it is one of the simplest as it just takes in the gradient information of the neural network computed through the loss function and adds a perturbation to the sample input to push the model towards the direction of higher loss in its classification [21].

$$X_{Adversarial} = X + \epsilon.sign(\nabla_X J(X, Y))$$

From this equation, we take the gradient of the loss function, and since we are only concerned with the direction and not magnitude, we take the sign of the gradient. We then multiply this direction by epsilon, a small number, which creates a small perturbation in the direction of greatest loss. This perturbation is added to the original input to create the new adversarial sample, which will then serve as input to the model.

PGD

1. Pick an initial point $\mathbf{x}_0 \in \mathcal{Q}$
2. Loop until stopping condition is met:
   - 2.1 Descent direction: pick the descent direction as $-\nabla f(\mathbf{x}_k)$
   - 2.2 Stepsize: pick a step size α_k
   - 2.3 Update: $\mathbf{y}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$
   - 2.4 Projection: $\mathbf{x}_{k+1} = \underset{\mathbf{x} \in \mathcal{Q}}{\operatorname{argmin}} \frac{1}{2}\|\mathbf{x} - \mathbf{y}_{k+1}\|_2^2$

Figure 2.2: Summary of how Projected Gradient Descent perturbs an input sample.

Projected Gradient Descent is similar to FGSM in how it takes the gradient of the model and uses it as the perturbation to each sample; however, PGD is just an iterative form of FGSM, essentially performing the same calculations of FGSM multiple times. Again, PGD aims to find a perturbation that maximizes the loss of the model through its gradient while keeping the size of it small enough so it is not noticeable to the model [15]. In each iteration, PGD takes a step towards the direction of greatest loss, and then projects this perturbation back onto the input samples; this process is repeated until it converges and finds a gradient that maximizes the loss.

Chapter 3

Experiment

To test the effectiveness of our two machine learning models, we used a subset of the TREC 2007 public spam corpus [2]. Much of prior research on spam detection have also utilized this spam corpus such as [18]. Despite the data set consisting of emails from 2007, not much has changed since within the structure of an email, but rather with the appearance of spam mail. A graph from Statistica of the email traffic from 2007 to 2019 has shown that spam hit its peak in 2008, consisting of about 93% of all email traffic and declining since. This could imply that our spam detectors have become "smarter" or better at detecting the spam mail [13]. This could also be due to the fact that email providers started to incorporate AI into their detectors in about the early 2010s. [8].

The total dataset consists of 75,419 email messages gathered by the National Institute of Standards and Technology (NIST) in which 25,220 are labeled ham and 50,199 are labeled spam, yet we will be using a sample of 4,000 emails, split evenly with 2,000 being spam and 2,000 ham. From these emails, we used the Python library, CountVectorizer, to get a matrix of each word and their occurrences in the emails. Specifically for the Neural Network model, we used the CountVectorizer to create an array of the top 50 words commonly found in spam emails and top 50 words of those found in ham emails. Then we used this list of top 100 words and compared against each email to see how many times each of these 100 words appeared in each email. This created a vector of 100 features for each email, and after normalization to scale the counts between zero and one, we used this data as input for the model. The word count for each classification would be used to determine which words were more associated with spam vs. ham emails and used later for generating the adversarial attacks.

We then split the data at random with about 80% used for training and about 20% used for testing purposes; the 80/20 split is a commonly used practice in machine learning when a testing set is not explicitly provided as it divides the data to effectively train the model, while also preventing the model from overfitting [11] . Each model, Naïve Bayes and Neural Network, was trained on the word count matrix for each email and its corresponding classification. After trained, the model was then tested on the verification data set, and the accuracy of the model was computed based on the number of emails correctly classified out of the total number of emails. Not only is computing the accuracy of the model important, but in the case of spam detection, it is important to delve deeper into what kind of error and misclassification was made. For that, we analyzed the confusion matrix of the output, focusing on what type of errors we received. A false negative would tell us that a legitimate email was misclassified as a spam email. More importantly, we are concerned with false positives in which a spam email bypasses the filter and is classified as ham.

After predicting with adequate accuracy, the Naïve Bayes model predicting samples at about a 96% accuracy and the Neural Network at about a 93% accuracy, we turned our efforts to generating adversarial samples to attack the models. For the Naïve Bayes model, we used ham-word injection. To do this, we analyzed all the ham emails to discover which word occurred the most. For this attack, we wanted to explore how injecting the topmost-occurring word multiple times into each email would have an impact on the model's loss. We discovered that the most commonly used word in ham emails is "the." After considering the fact that we might receive common words such as "the," "a," "you," etc., in our results, we decided to utilize the CountVectorizer's feature of "stopWords" to prevent such words from being considered in our count. However, after several rounds of eliminating these stopWords, we discovered that a non-stop word produces similar results in the ham word-injection attack as the word "the" would. After discovering what the most commonly found word in ham emails was, we experimented with how injecting this word n-times

would affect the model's accuracy with 'n' ranging from 10 to 5,000.

For the Neural Network, we utilized PGD to create our adversarial samples. Passing in our feature vector as input, we then applied PGD to each sample which made a small perturbation to each sample in the direction of the gradient that increases the loss. Once these adversarial samples are created, we passed them into the Neural Network and analyzed the model's loss. In order to improve the impact that the attack had and increase the loss of the model, we experimented with the degree of perturbation that was applied to PGD, choosing values from 0.01 to 0.3.

After training each model and generating its adversarial samples, we get to the main focus of the experiment of seeing if the adversarial samples generated on one model can be effectively applied to the other. The adversarial samples from each model's attack are saved and then used as input for the opposite model. Similar to before, we will train each model on the non-perturbed data, and then test the accuracy and loss of each adversarial sample to determine its effectiveness in tricking its classification. We can determine if the attack was successful if the loss significantly increases, aiming for a prediction accuracy of about 20% or less.

Chapter 4

Results

4.1 Naive Bayes Results

After running all 4,000 email samples through the CountVectorizer to get a vector of each email's word count, we ran it through scikitlearn MultinomialNB to train the model. Then we used the testing samples to test the accuracy of the model and received a testing accuracy of 96%. By looking at the confusion matrix shown below, we can further see how the predictions measure up to the label out of the 800 testing samples. Given that this model was highly successful in classifying the emails as spam or ham, the true positives (actual spam predicted as spam) and true negatives (actual ham predicted as ham) consists of a higher ratio than false positives (spam predicted as ham) and false negatives (ham predicted as spam).

	Predicted Spam	**Predicted Ham**
Actual Spam	392	1
Actual Ham	28	379

Table 4.1: Confusion matrix for predicted outputs of Naive Bayes model.

After we trained and tested the Naive Bayes model to achieve maximal accuracy, we then performed the ham-word injection to perturb our email samples. From the start, we found that the "top ham word," the word most commonly found in all ham emails, was "the." Given that this is a commonly found word in general, we wanted to narrow down our results in hopes of finding a word that is more specific to just the ham emails. After calculating the top-most ham word multiple times, we found that for the first several rounds, the word came out to be one that is commonly found in English sentences such as "the", "a", "you", "for", "as", etc. If it resulted in one of these sentence structure words, we would add it to our "stopWord" list and continue testing until we found a "non-stop word" of "http." However, after running the ham-word injection attack with either word, "http" or "the", we found that the stopWords did not affect the testing accuracy. What did affect the accuracy was how many times the word was injected or repeated within a single email:

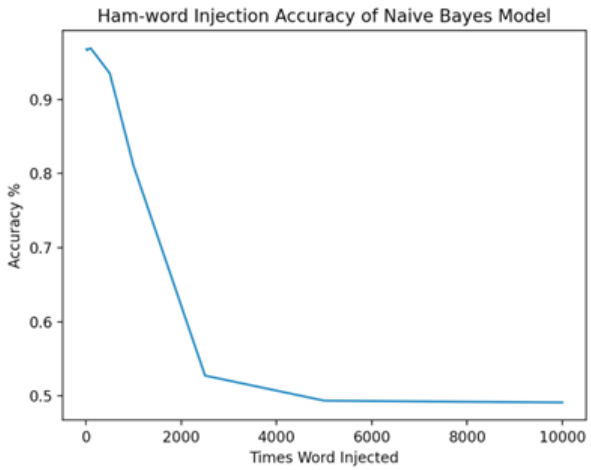

Figure 4.1: Naive Bayes Model's accuracy decreasing as the number of times the "ham word" is injected into the email.

As shown above is a plot that illustrates how the accuracy of the Naïve Bayes model was affected depending on how many times the top ham word was injected into it. We can see that injecting it 10 times barely affected the accuracy, while injecting it at least 5,000 times decreased the accuracy to about 50% to where it plateaued at that point. In another attempt, we tried to find how the top n-ham-words, each injected into the emails once, would affect the accuracy of the model; similarly, we found that it produced similar accuracy results as we needed the top 5,000 words injected once to receive a 50% accuracy. Realistically, if a single word was injected over multiple times within one email, the user would be able to see that it is obviously a spam email. Because of that, we were hoping that just by injecting it under 20 times would greatly affect the accuracy; however, our results show that the attack was not successful until we injected the word at least 5,000 times. So while we would be able to trick the machine learning model, an adversarial email generated by these means would be obvious spam to the user. Future research could involve exploring this technique such that the alteration of the email would deceive both the model and the user.

We can see through the confusion matrix of the ham-word injection output how the predictions vary from the original data ran through the Bayesian model. Since this attack caused the predictions to be correct only 50% of the time, we can see a greater difference in emails falsely classified. Specifically, we see more ham emails falsely classified as spam than spam emails classified as ham; this however, goes against our attempt to have spam emails classified as ham by injecting more ham-associated words into them. Therefore, these results show that ham-word injection can make emails ”more spammy” as we inject a single word multiple times. We even see that the amount of emails in total predicted as ham is only 7, while the majority of emails are classified as spam. A further exploration would be to see if injecting common spam words would have spam-classified emails predicted as ham.

	Predicted Spam	**Predicted Ham**
Actual Spam	393	0
Actual Ham	400	7

Table 4.2: Confusion matrix for predicted outputs of ham-word injection attack on Naive Bayes model.

4.2 Neural Network Results

Before training the Neural Network, we needed to find the most commonly found words in the ham emails and in the spam emails. As shown below is 50 of the most commonly used words in ham emails and in spam emails. Then we use Word Cloud for each of the spam and ham email sets to illustrate which words occurred most frequently.

```
(Pdb) hamWords
['http', 'can', 'will', 'new', 'please', 'list', 'all', 'has', 'read', 'more', 'mailing', 'use', 'news',
'what', 'would', 'which', 'april', 'one', 'there', 'about', 'apr', 'here', 'how', 'some', 'return', 'so',
 'provide', 'any', 'when', 'get', 'like', 'posting', 'guide', 'click', 'reproducible', 'they', 'am', 'no'
, 'may', 'data', 'should', 'up', 'just', 'using', 'other', 'file', 'out', 'than', 'also', 'don']
[(Pdb) spamWords
 ['http', 'all', 'le', 'transaction', 'pas', 'can', 'more', 'en', 'internet', 'will', 'they', 'pour', 'no', '
 group', 'nous', 'si', 'institution', 'tout', 'people', 'one', 'like', 'us', 'would', 'money', 'up', 'said',
 'their', 'what', 'cid', 'about', 'see', 'out', 'now', 'dear', 'thanks', 'only', 'men', 'had', 'most', 'has',
  'hours', 'every', 'high', 'when', 'number', 'business', 'during', 'problem', 'last', 'account']
```

Figure 4.2: The most common words found in ham emails (top) and spam emails (bottom).

Figure 4.3: Word Cloud representation of the spam (left) and ham (right) word occurrences.

We used this list of 100 words to get a count of each word occurrence per email, which generated a vector that, after normalization, was used as input to the Neural Network. Using train_test_split from scikitlearn, we split our training and testing data 80-20. We experimented with the number of layers and number of neurons in each layer, first starting out with 10 hidden layers and number of neurons decreasing by 10 in each layer. We found that this structure was ineffective and producing a low accuracy, so we then divided the neurons per layer by multiples of 100. We then discovered the optimal number of hidden layers is 3, consisting of 75, 50, and 25 neurons, to produce a maximal accuracy. Next we experimented with the learning rate, testing values of 0.1, 0.01, 0.05, and 0.001; out of all of these values, we found that 0.001 produced the highest accuracy. Finally, we experimented with the number of epochs to train the model, testing the accuracy with epoch values of 10, 15, 20, and 25. We found that with 15 epochs, we received the highest accuracy of them all to receive an average accuracy rate of about 93%. Similarly to the Naive Bayes model, the accuracy of the Neural Network model on the data alone is fairly high, and because of this, we can see in the confusion matrix below, that the number of true positives and negatives are higher than the false positives and negatives. However, since the accuracy of the Neural Network is a little bit smaller than that of the Naive Bayes model, we can see how there are a slight more false positives and negatives outputted from the Neural Network.

	Predicted Spam	**Predicted Ham**
Actual Spam	384	35
Actual Ham	25	356

Table 4.3: Confusion matrix for predicted outputs of Neural Network model.

After training the model on the original data, we perturbed the testing samples using Projected Gradient Descent using an alpha value of 0.2 for 15 iterations. With this attack, we were able to trick the model and received a testing accuracy of about 15%. Since the PGD attack produced a low accuracy prediction, we can see that many of the emails were misclassified in the confusion matrix below. In a way, we can view the PGD confusion matrix as a mirror to the one of the Neural Network on the original data as the majority of the emails were falsely predicted, which therefore, proves that the attack on the samples were successful.

	Predicted Spam	**Predicted Ham**
Actual Spam	62	324
Actual Ham	355	59

Table 4.4: Confusion matrix for predicted outputs of PGD attack on Neural Network model.

4.3 Cross Model Attack Results

The main focus of the experiment was to see how adversarial samples created for one model would affect the accuracy predicted by the opposite model. Using the same top-most ham word of "http" and injecting it 5,000 into each email, we then used the word count vector of 1,000 email samples (500 ham and 500 spam) as inputs into the Neural Network. At first, we only perturbed the spam emails in hopes that injecting ham words would falsely classify the spam emails as ham; however, this only increased the accuracy as it easily allowed the model to correctly identify spam emails through the use of "http." Because of this, we decided to perturb all emails, which then only reduced the accuracy about 10% from about 90% to about 80%. Since the ham-word injection attack was not very successful when applied to the Neural Network, the confusion matrix shown below still has a greater ratio of emails correctly classified. We can see, however, similar to the ham-word injection applied to the Bayesian model, that it shifted the predictions more towards spam than ham.

	Predicted Spam	**Predicted Ham**
Actual Spam	491	9
Actual Ham	205	295

Table 4.5: Confusion matrix for predicted outputs of ham-word injection attack applied to the Neural Network model.

Lastly, we trained the Naïve Bayes model using the original email data, and then used PGD to perturb the email testing set using the same testing parameters as mentioned previously. Using

the perturbed emails, we tested the accuracy of the model, and it predicted the emails correctly at about 16%, showing that the PGD perturbations were successful in tricking the Naïve Bayes model as well as the Neural Network. The PGD attack applied to either model proved to be effective in increasing the accuracy of the predictions. The confusion matrix of the PGD attack on the Bayesian model, shown below, is similar to the PGD attack on the Neural Network with a majority of the emails being falsely classified.

	Predicted Spam	**Predicted Ham**
Actual Spam	94	305
Actual Ham	350	51

Table 4.6: Confusion matrix for predicted outputs of PGD attack applied to the Bayesian model.

Chapter 5

Discussion

We have considered two different machine learning models and how they would respond to attacks created specifically for one type of model. A Naïve Bayesian makes its predictions based on the probability a certain feature would be found within a class. Opposingly, a Neural Network learns through multiple rounds of feeding the data within the network and back propagating to update the weights and minimize the overall loss.

We found that the ham-injection attack proved only mildly effective for tricking the Bayesian model, bringing the prediction accuracy to about 50%; however, in order to achieve this level of loss, we had to inject the email with the ham word an excessive amount of times (5000), which is unrealistic as the user would be able to see the repetition of this word and instantly know it is spam, despite it bypassing the spam detector. Similarly, the ham-injection attack proved ineffective for the Neural Network as it only decreased the overall accuracy to about 80%, about 10% less than the model's prediction on the original data. Despite "http" being a word commonly found in both spam and ham emails (listed as part of the input vector), even when the attack was performed on a different commonly found word such as "the" or "not", it proved to return the same accuracy, showing that the Neural Network is unaffected by a ham-word injection attack.

However, we found that PGD is largely effective in tricking a model, no matter which type of model it is. Running PGD to create samples to then run through the Neural Network and the Bayesian model produce similar accuracy results of about 15%. This is because PGD focuses on changing the input, the vector of word counts, by small amounts that will shift it towards the direction of greatest loss, and no matter what model we run these samples through, they will always trick the model into misclassification. While ham-word injection is ineffective on the Neural Network, PGD samples prove effective in tricking the Bayesian model as well, despite PGD altering the samples according to the gradient and the Bayesian model relying on statistics and probability.

Through our results, we can see that our original prediction of the experiment was true in the sense that one of these attacks can operate on both models. We originally believed that the ham-word injection attack could operate on both models, which proved incorrect as it produced only a slight loss in the Neural Network. It turns out that the PGD attack was able to produce adversarial samples that could greatly impact the loss of both model's predictions. These results lead way into analyzing the transferability of PGD in attacking other models besides those solely structured around gradient changes such as a Neural Network. We can also further analyze non-gradient based attacks, such as ham-word injection and other alterations of the email's contents, to see how we can improve on its attack efficiency on not only the Bayesian model but also other models such as a Neural Network.

Chapter 6

Conclusion

Through the construction of a Naïve Bayes model and Neural Network, we were able create attacks that perturbed samples and trick the models. We found PGD to be more effective in tricking both the Neural Network and the Bayesian model than the ham-word injection attack, despite the injection attack being creating specifically for the Bayesian model. As the number of times we injected the word into the emails increased, the accuracy decreased until it plateaued at about 50% with the word repeating about 5,000 times. PGD applied a small perturbation to each input feature and proved effective in tricking both models, despite one model being gradient-based and the other being statistically based. If both attacks would have proved effective for each model, then we could have concluded that the latency space of each model was similar; however, since this is not the case, we can conclude that PGD makes perturbations to the word count vectors that shift the predictions towards a greater loss. Given that Bayesian models and Neural Networks are structured and operate differently, we can also infer that the transferability of these attacks could work across other machine learning models as well. We can continue our research area by applying these attacks to other classification models as well as constructing new attacks to the emails; other attacks could consider other textual features such as entire email word counts, spacing, punctuation/symbol count, letter case, or they could use other parts of the email (header, IP address, etc.) to determine the classification of the email.

As for implications towards spam detectors, we can conclude that either the Bayesian model or Neural Network would be equally effective in classifying an email and moving it to the junk folder. However, the Bayesian model would be more effective in protecting against text alteration attacks as seen by the ham-word injection attack, yet an attack such as that would also be visually detected by the user as being spam. PGD applied to email samples could effectively have a spam email bypass the junk folder; one way to protect against this would be to better train the model on more samples as well as train the model on adversarial crafted samples. Overall, while either model proves effective in classifying emails based on their word content, both models prove susceptible to a gradient-ascent attack. This kind of an attack could be detrimental to the spam detector as it could have a spam email bypass the junk folder and placed in the inbox, and while this attack may produce results in the email not noticeable to the human eye, it could place the user at harm and increase a threat in their security and personal information by accidentally opening a spam email.

Bibliography

[1] Adversarial robustness- theory and practice. https://adversarial-ml-tutorial.org/.

[2] Trec 2007 public corpus. https://plg.uwaterloo.ca/cgi-bin/cgiwrap/gvcormac/foo07, 2007.

[3] Spam statistics. https://99firms.com/blog/spam-statistics/gref, July 2021.

[4] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Practical black-box attacks on deep neural networks using efficient query mechanisms. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

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

[6] J. Clark, I. Koprinska, and J. Poon. A neural network based approach to automated e-mail classification. In *Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)*, pages 702–705, 2003.

[7] Emmanuel Gbenga Dada, Joseph Stephen Bassi, Haruna Chiroma, Shafi'i Muhammad Abdulhamid, Adebayo Olusola Adetunmbi, and Opeyemi Emmanuel Ajibuwa. Machine learning for email spam filtering: review, approaches and open research problems. *Heliyon*, 5(6):e01802, 2019.

[8] Caitlin Dawson. The origin of spam and other online intrusions. https://viterbischool.usc.edu/news/2019/07/the-origin-of-spam-and-other-online-intrusions/, July 2019.

[9] Yanlei Diao, Hongjun Lu, and Dekai Wu. A comparative study of classification based personal e-mail filtering. In *Proceedings of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, PADKK '00, page 408–419, Berlin, Heidelberg, 2000. Springer-Verlag.

[10] Samira Douzi, Feda Alshahwan, Mouad Lemoudden, and Bouabid Ouahidi. Hybrid email spam detection model using artificial intelligence. *International Journal of Machine Learning and Computing*, 10:316–322, 02 2020.

[11] Afshin Gholamy, Vladik Kreinovich, and Olga Kosheleva. Why 70/30 or 80/20 relation between training and testing sets: A pedagogical explanation. 2018.

[12] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014.

[13] Joseph Johnson. Global spam volume as percentage of total e-mail traffic from 2007 to 2019. https://www.statista.com/statistics/420400/spam-email-traffic-share-annual/, Jan 2021.

[14] Alexey S. Katasev, Lilia Yu. Emaletdinova, and Dina V. Kataseva. Neural network spam filtering technology. In *2018 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, pages 1–5, 2018.

[15] Oscar Knagg. Know your enemy. https://towardsdatascience.com/know-your-enemy-7f7c5038bdf3, Jan 2019.

[16] Bhargav Kuchipudi, Ravi Nannapaneni, and Qi Liao. Adversarial machine learning for spam filters. 2020.

[17] Daniel Lowd and Christopher Meek. Good word attacks on statistical spam filters. 2005.

[18] David Miller, Xinyi Hu, Zhen Xiang, and George Kesidis. A mixture model based defense for data poisoning attacks against naive bayes spam filters. 10 2018.

[19] Amit Sharma, Sudesh Kumar, and Mohammed Aslam. A comparative study between naive bayes and neural network (mlp) classifier for spam email detection. 2014.

[20] Arif Siddiqi. Adversarial security attacks and perturbations on machine learning and deep learning methods. *ArXiv*, abs/1907.07291, 2019.

[21] Ken Tsui. Perhaps the simplest introduction of adversarial examples ever. https://towardsdatascience.com/perhaps-the-simplest-introduction-of-adversarial-examples-ever-c0839a759b8d, Aug 2018.

[22] Chih-Hung Wu. Behavior-based spam detection using a hybrid method of rule-based techniques and neural networks. *Expert Systems with Applications*, 36(3, Part 1):4321–4330, 2009.

Sheryll Martutartus

EDUCATION

The Pennsylvania State University: Schreyer Honors College
Bachelor of Science in Computer Science
Minor in Cyber Security, Mathematics

TECHNICAL EXPERIENCE

Computer Science and Engineering Research
Systems and Internet Infrastructure Security Lab Fall 2020-Present

- Focused on developing security strategies with machine learning to determine the adverse effects it has on various models.
- Analyze and clean through datasets that will result in the most relevant, efficient output.
- Explore relevance of adversarial examples within various types of machine learning models to assess the impact it has on the result's accuracy.
- Developed attack strategies within machine learning to generate adversarial examples.

PPL Electric Utilities
Threat and Vulnerability Management Intern Summer 2021

- Implemented the development process of using a vulnerability manager (Qualys) to assess CIS benchmarks as a testing ground for network devices.
- Performed security assessment scans on various devices to assess compliance with current policies.
- Utilized a vulnerability management API to create a script that retrieves asset criticality and compared the results to current priority assignments within the database.

Lockheed Martin
Cyber Certifications Intern Summer 2020

- Performed cyber security scans of Navy programs to understand the vulnerabilities associated with each system.
- Analyzed scan results and mitigated the vulnerabilities to create Risk Management Framework packages.
- Developed a tool using Excel Macros and VBA that verifies that the scan policy settings and scan results are present and correct in the vulnerability report prior to customer delivery.