Kevin Huang, William Selke
CSS 576
Final Project Report
03/17/2023

Spam Detection Application Report

Introduction

What problem are you trying to solve and what was your basic approach?

Spam emails are a persistent problem that we all experience. In today's digital world, it is extremely difficult to go through day-to-day activities without having an email address. Through using emails, it is inevitable that one would come across spam emails at some point. Spam emails not only consume valuable time and resources, but they also pose a security risk as they may contain malware, phishing scams, or other fraudulent content. Because of this, our goal is to explore the use of machine learning for spam detection and build an application that can aid in spam filtering.

Our approach to building the spam filtering application involves several steps. The first step is collecting a large dataset of spam and ham (legitimate) emails. Then, the dataset is preprocessed to identify and extract relevant features for distinguishing between spam and ham emails. After that, machine learning models are trained using the preprocessed data. The next step is to evaluate the performance of the models and also optimize it for spam detection. Finally, the model is deployed in the spam filtering application where it can be used to help classify incoming emails as either spam or ham emails.

Data Collection

Describe how and where you collected your data. Why is this data good for your particular security application? Reflect on any limitations in the process such as bias that may have been introduced. How did you balance class sizes?

Data was collected from a variety of sources: the SpamAssassin archive, the untroubled spam archive, and our own personal emails. We downloaded the spam, easy ham, and hard ham datasets from the SpamAssassin archive through https://spamassassin.apache.org/old/publiccorpus/. The data was good for spam detection because it provided both ham emails and the target, spam emails. Also, the 2022 dataset from untroubled was downloaded through http://untroubled.org/spam/. This dataset was useful because it contained spam emails from recent years. The tactics of spam email senders are constantly changing so it is beneficial to have spam emails that represent the current methods. In addition, we collected ham emails from our personal emails to use as part of our data. This

dataset was beneficial because it provided a better representation of current ham emails. All of these datasets are not preprocessed so we are able to extract the features that we want.

Although we gathered a sufficient amount of data to use for model training, we have identified some limitations to the datasets that we have chosen. The first limitation is the age of the datasets from SpamAssassin. All of the data is from the early 2000's so are exposing our model to potential bias of email structures and content from that time period. The second limitation is that we do not know the source of the emails. We do not know what methods were used to obtain these emails and whether or not it is representative of spam emails from all industries. For example, the emails might have been only from the financial industry and doesn't include other fields such as medical or retail shopping. This can bias our model to the type of data provided by the untroubled and SpamAssassin archives. In addition, we have introduced bias by including our own emails. Our emails contained content from school, shopping, and investment/banking. These are only a few categories of the total types of emails a user can receive.

After we have compiled the emails from the sources described above, we noticed that a majority of our data were spam emails. We decided that we need to rebalance the class sizes of ham and spam emails to improve our model training. The method we chose to use was SMOTE from the Imbalanced-learn library. This technique balances the classes by creating synthetic data from the minority class. After performing this function, we had a dataset that had an equal amount of spam versus ham emails.

Data Preprocessing

How did you go about deciding on features? If you used feature reduction, what did you do? What steps were necessary to collect those features from the data. If you augmented it with preprocessed data, note that here.

Our data preprocessing can be broken down into extraction/normalization of the data and feature processing.

For the first portion extraction, the raw header files are extracted into a dictionary. The data was extracted from the email headers, first using commonly noticed features that humans used, like the top level domain of the sender and the subject and the message itself. In addition more features were added that are usually not shown to the users such as, the precedence field.

Here are the list of features that were extracted from the header

1. Top Level Domain (of sender)
2. Precedence
3. Subject and Message Punctuation Count (number of punctuations in subject title
4. Subject and Message Symbol Count (number of symbols in subject title)
5. Subject and Message Money Sign Count

6. Subject and Message Misspelled Word Count
7. Subject and Message Capitalized Word Count
8. Subject and Message Word Count
9. Subject and Message Sentence Count
10. Large Sums (number of large numbers > 1000) in the message
11. Number of Emails in the Message
12. Subject and Message word frequency

To normalize the data for the model, those that are integers are normalized to a predetermined lower and upper limit (the determination was made for analyzing the data to see the typical distribution of each feature). For features which are strings however, they are one hot encoded, such as the type of level domain. And the word frequency distribution for both the message and the subject.

After the preprocessing of the data, we ended up with more than 12,000 columns. In an attempt to reduce the features down further we had two approaches for feature reduction. Firstly since most of these features came from the one hot encoding of word frequency, we performed a count in which we kept the top 20 words that appeared in the spam but not ham and vice versa. This was the first step of our feature reduction in which only the words which may be helpful in our identifications are kept. By the end of this process the number of features had been skimmed down to 145 features.

Secondly, we ran the model through a white box “Intreprete” library which create a white box model and trained the data on it. After which it then gives us rankings on the top words. Unsure of how many features would be useful however, we created multiple datasets, with the top 10, top 20, top 30 features and so forth. All of these datasets were trained and the optimized ones were located with gridsearch in the model optimization part.

Model Training

What model(s) did you use. How did you decide on hyperparameters? How did you decide on which metrics to optimize for your specific application. What were the results?

We decided to train a neural network model and a decision tree model and choose the best performing model out of the two. The metric that we wanted to optimize for our spam filtering application was precision. Precision is calculated by dividing the number of true positives by the sum of the true positives plus the false positives. This metric is useful for when you want to minimize false positives. This case applies to our application because we want to minimize the probability that an email is flagged as spam when it is actually ham. False positives are detrimental to our application because if an important email gets filtered and classified as spam, the user will likely want to use the application. Because of this, we want to optimize precision.

The neural network which we trained had 4 hidden layers which its size however was randomized each time to perform a grid search. The neural network, in the end, had quite a

good performance at 98% accuracy. We used the PyTorch Library to implement this neural network and the RELU activation function for all layers.

For the decision tree model, we decided on the hyperparameters through grid search. We first found the rough optimal depth using all default hyperparameters and varying the depth from 1 to 100. We found that the depth that gave us the best precision was 14. Then, we had multiple versions of our dataset with different numbers of features (10 features, 20 features, 30 features, etc.) and tested and found that the best performing model was when we used all of the 145 features that we extracted. Lastly, we did hyperparameter tuning using grid search and testing the depth from 8 to 18 and trying the 'best' and 'random' splitters. We found that the best hyperparameters were 16 depth and 'best' as the splitter. The precision that we got for our testing data was 97.44%. Printing out the confusion matrix, we saw 2001 true positives, 1988 true negatives, 63 false negatives, and 42 false positives. It was great to see only 42 false positives since this was what we wanted to minimize.

For our model, we decided to go with the decision tree model for its computational time, precision results, and explainability. We found that the computational performance for the decision tree model was better than the neural network model. The neural network model took longer to train and make predictions. We wanted our application to perform classifications quickly to improve usability for the users. We found that the precision results were similar so there is no performance loss for using a simpler model. Another factor to use the decision tree model was to promote explainability. We were able to print out a visual representation of the model to display how the model makes its decisions. This visibility into our application promotes trust and user confidence.

Application Development

Did you write your application as a part of another system or a standalone application. Assess its performance in terms of classification and computational performance. How do these results affect the usability. What are the limitations (eg, would the spam filter be able to run on a mail server that processed millions of messages a day?). If ART has attacks that can be launched on your system, assess how effective they are. I do not expect you to convert an attack designed for image recognition into network log anomaly detection -- only do this if ART has a directly applicable attack.

We created our application as a standalone application. A single user would login to their Gmail account and then the application would download all their emails in their inbox and classify them as either ham or spam emails using the decision tree model that we trained. The application then displays a count and list of all the spam emails.

Unfortunately, since the model does take a while to pull emails from the server and run the function, our GUI and models are runned on a separate thread to ensure the responsiveness of the GUI. After the model runs the results are then communicated back to the GUI via the queue

library. The model would place the number of spam detected along with the subject and the sender of the spam emails to display to the user.

This application was meant for individual users and would not be able to function efficiently on a mail server that processed millions of messages a day. It requires a single user to log in to their Gmail account in order to download their emails from their inbox. Although the whole application does not fit in a commercial environment, the model that we made may be able to integrate with mail servers.

An attack from ART that can be launched on our system is documented in the attack_decision_tree.ipynb notebook in ART. If an attacker has knowledge of the learned tree structure, they would be able to perform evasion attacks on our system. They only need to change a subset of the features so that it meets the threshold to be misclassified. This attack is very effective on our system so we need to implement measures so that the learned tree structure is not exposed to attackers.

Conclusions and Final Reflections

What take-aways do you have from the project. If you ran into performance problems with your application, what did you do to address them. What would you do differently if you started it again? What would you do in the future? Address how building the full project affected your thinking from when you were only performing the work for individual stages during your earlier assignments.

The main takeaway we have from the project is that there is more to building an AI-enhanced application than just training a machine learning model. All stages of the development process are important, from the data collection, to preprocessing the data, to selecting the model, and finally building the application. How well you do a stage will affect the results of the later stages. Also, it is important to consider the usability of the application. It does not matter how effective your application is if users do not trust it or if it's too complex to use.

If we were to restart building this application, we would want to collect more data ourselves. A majority of our ham and spam emails came from third party collected datasets. We do not have any control or insight into where and how they collected the emails. By collecting all or a majority of the emails ourselves, we can better understand what biases there are in the data. This will in turn provide us more explainability and better model training.

One thing that we can do in the future is to train more robust machine learning classifiers. We decided to go with a decision tree model for our application but some possible models that would make our application more robust would be ensemble or random forest models. By combining multiple models together, it can improve the overall performance by reducing the impact of errors or biases in individual models.

An additional thing that we would like to do is allow the application to explain its decision essentially. One of the main reasons we went with a decision tree was its explainability, so we would like to be able to levy this to improve our user experience.

Building a full application made us more careful when considering our methodology for each stage than when we were doing the stages individually in the assignments. For the data collection stage, we put a lot of emphasis on gathering sufficient data and making sure that we had a balance of data for each class. For the preprocessing stage, we spent more time looking at what features to collect and did additional analysis on the correlations for better feature selection. In the model training stage, we thought carefully about what model to use and decided on the decision tree model to provide explainability to promote trust from the users.