The University of North Carolina at Charlotte

Email Spam Detection Using Machine Learning

Primary Paper Studied: “Phish Responder: A Hybrid Machine Learning Approach to Detect Phishing and Spam Emails.” by Molly Dewis and Thiago Viana. The primary paper was published in 2022 and is available at https://doi.org/10.3390/asi5040073

Daniel Head

ITCS 5156: Applied Machine Learning

Dr. Minwoo “Jake” Lee

October 12, 2022

Github Link: https://github.com/DanielHead96/ITSC5156_Applied_Machine_Learning

Introduction

Problem Statement

According to Dixon, billions of spam emails are sent and received each day. While many of these may be filtered out by existing filters, machine learning presents a new and possibly better method of classifying spam emails and preventing spam from reaching a person's email inbox. Aside from cluttering up their email inbox, the spam could be a phishing attempt where the sender is trying to get valuable information such as payment information, passwords, or by attempting access to the recipient's device by having them download a file. In all of these cases, spam is harmful. I believe that researching further methods to prevent this by exploring innovative machine learning techniques that may prove effective is important to prevent the harm that can be done by spam emails.

Motivation and Challenges

Spam and phishing attacks can come in many forms, but based on the large number of emails being sent it is clear that email is a large source of them. I think that preventing this from occurring, when possible, would be ideal. I also thought it would be interesting for me to work on because I have been receiving, what seems to be, a very large amount of spam emails recently. I started to think that perhaps there is research going on to help prevent these spam emails, and given that this is a machine learning course, I thought it would be interesting to research machine learning approaches to detecting spam emails. After researching the topic, I was impressed to find some papers had accuracies of 98% or higher in detecting spam emails. Spam emails are prevalent, and it is clear that machine learning methods can achieve high accuracy in detecting these spam emails.

The largest challenge that I foresaw was working with a large enough set of data to achieve good results with a model I create but also to be able to train the model with the resources I had available. The primary paper I chose, Phish Responder: A Hybrid Machine Learning Approach to Detect Phishing and Spam Emails, actually used multiple datasets when they created their model. I chose one dataset from Kaggle (Nitisha), and ran into RAM issues when attempting to use the full dataset. Another challenge I ran into was that I had worse accuracy than expected when I tried to replicate the model that was created in the primary paper I chose. Their model had an input layer, an output layer, and 7 other layers (2 LSTM, 5 dense) however, I was only able to achieve a maximum accuracy of around 75% with this model[1]. After modifying it slightly, I was able to achieve a much higher accuracy. I believe that the challenge of having resources to properly train a model is common to many machine learning projects, as is having to adjust models to the amount of data you are using.

My Approach

As mentioned previously, I initially attempted to replicate the methods done in my primary paper chosen. To do this, I used TF-IDF and created the same type of model as they had, consisting of 9 total layers. I was achieving a lower accuracy than I hoped for in my initial testing, so I modified the model to have 1 input layer, 1 output layer, and 4 other layers (2 LSTM, 2 dense). I did this because I suspected that, with using less data, a model with fewer layers may be better. I also eliminated stop words, converted all numbers to their word equivalent (converting 0 to zero, 75 to seventy-five, etc), and eliminated punctuation and symbols such as /, #, etc. One of the largest reasons I did this was in the hopes of cleaning up and reducing the number of unique words in my data (one of the parts that was often crashing was

[1] Later during the project, it was found that the low accuracy was due to the model not being trained long enough. This was not realized until after most of the paper was completed and the project was considered done, so is discussed in the conclusion section of the paper.

the TF-IDF step), however, this was also done to simply clean the data. Dewis and Viana did not mention doing some of these steps, but I found that they improved model accuracy during my own testing. Finally, LSTM models were created to compare and test hyperparameters. The LSTM models were used to compare the Adam, AdaMax, NAdam, and RMSProp optimizers and batch sizes of 10, 32, 64, and 128. In addition to the LSTM models, a Gaussian Naive Bayes, Perceptron, Random Forest, and SGD model were created to compare against as well. After testing, an LSTM model was trained with the best hyperparameters found for 100 epochs.

Background

Survey of Related Works

I reviewed two related papers, both of which focused on spam detection in text form. However, one focused on emails, while the other focused on text messages. I found both to be interesting for different reasons, and despite one being focused on text messages, I think both were relevant. In the end, text messages are a different medium of communication than email, but are still textual data. “A Study of Machine Learning Classifiers for spam detection” by Shrawan Kumar Trivedi was a helpful resource for a general overview of classifiers and how they perform for spam detection, as well as an overview of how quickly they can be trained. “Deep Convolutional Forest: a Dynamic Deep Ensemble Approach for Spam Detection in Text” by Mai A. Shaaban, Yasser F. Hassan, and Shawkat K. Guirguis was very interesting because they created a new model that hoped to overcome some of the limitations that deep learning posed. Their goals included creating a new model, extracting features dynamically with convolutional and pooling layers, and achieving high accuracy in classifying spam vs ham messages. In both related works, their testing was able to achieve high accuracy and show that machine learning may be a good approach for spam detection.

Summary of Approaches Surveyed

The deep convolutional forest network presented by Mai A. Shaaban, Yasser F. Hassan, and Shawkat K. Guirguis had a few key components that made it stand out against previously done research, including automatic feature selection and being able to change the complexity dynamically based on the increase in the model's performance. The authors believed that one of the gaps in the existing literature that their solution addressed was that “no domain expertise is required to carry out the classification process” (8) in their solution. Because of the way their model handles feature selection, I think that this statement carries weight. They compared their method against a few other models including Naive Bayes, Random Forest, CNN, and LSTM models. In their testing, the deep convolutional forest performed with the highest accuracy. All in all, I think that their method was interesting and resulted in fairly good results.

Shrawan Kumar Trivedi’s paper, A Study of Machine Learning Classifiers for Spam Detection, focused more on the accuracy and training of various machine learning classifiers including Naive Bayes, SVM, and the differences AdaBoost can cause for Naive Bayes and Bayesian models. I believe one of the biggest pros of the methods used in this paper is that the training times were much faster than using deep learning models such as LSTM, CNN, or the deep convolutional forest method mentioned before. I noticed this when I was training models for my project as well as during class labs, but these models are much much faster than training most of the machine learning models we experimented with. For pre-processing, they removed tags (such as HTML tags), removed stop words, and performed lemmatization. Finally, when comparing models, they focused on the false positive rate and F-score the most. Their theory was that it was important to prevent spam emails from reaching a user, but more important to prevent ham emails from being filtered out as this could cause issues for the user. I found this paper interesting because it compared a variety of models for email spam detection.

The primary paper I chose, Phish Responder: A Hybrid Machine Learning Approach to Detect Phishing and Spam Emails by Molly Dewis and Thiago Viana, focused on using deep learning for email spam detection. They tackled numerical and text data differently, using MLP and LSTM models respectively. In both cases, they used the relu activation function for the input layer and the first hidden layer, and then sigmoid for all other layers including the output layer. Their approach included testing multiple optimizers and batch sizes between 10 and 64. For pre-processing, they chose to only use tokenization and TF-IDF. They found that other methods, such as lemmatization, did not affect model accuracy in their testing. They then compared their model with accuracies from existing research, where they found their model to have performed well in comparison to existing research.

Relation to My Approach

The work with the strongest relation to my approach is the primary paper, particularly the LSTM model they used. I initially attempted to recreate their model, however, through testing I found it more beneficial to include a few techniques from other works. Namely, removing stop words as well as converting numbers to their character equivalent. One of the main reasons I did this was actually to attempt to possibly slightly cut down on the number of words in the data since I was running out of RAM. In hindsight, perhaps lemmatization would have been good to do as well, particularly since the related works also mentioned that. Similar to all the works I reviewed, I also chose to compare the models I created to other models and existing research.

Method

The key part of the method I used is the LSTM layers. LSTM models are a type of RNN model, but particularly useful because they can be used to remember important data for the long term. They have a few components to allow this behavior to work well, including a forget gate, input gate, and output gate. Using these, they can be used to determine what information isn't relevant or important and forget it, determine what information to replace it with, and of course output that information. A general diagram and the equations that an LSTM cell uses can be seen in Figures 1 through 7, all of which have been taken or adapted from Olah, Christopher. Essentially, each cell determines what information should be forgotten, what information should be learned, and then determines what should be output. It can keep information stored for a long term which is helpful because the body text of an email can be long and keywords may need to be remembered over a large sequence of words.

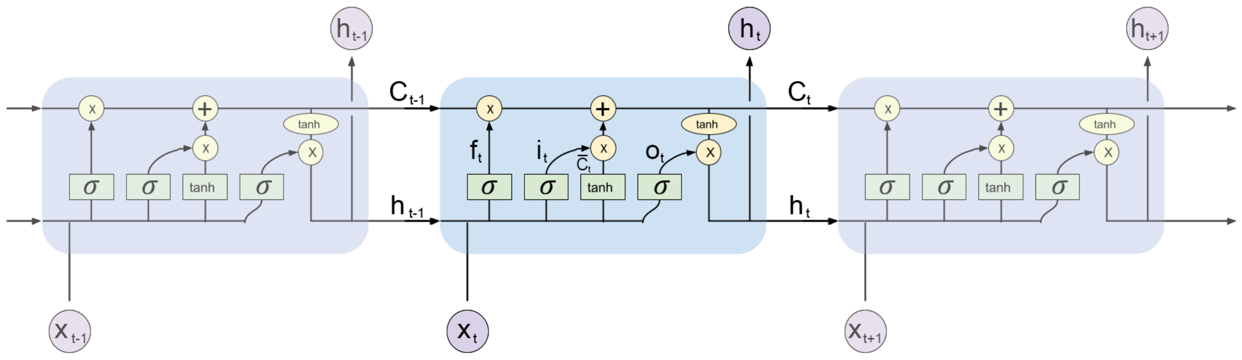

Figure 1. A diagram of LSTM layers. Adapted from Olah, Christopher.

$$f_t = \sigma\left(W_f \bullet \left[h_{t-1}, x_t\right] + b_f\right)$$

Figure 2. This equation is for the forget gate, where an LSTM cell determines what information to keep or forget. From Olah, Christopher.

$$i_t = \sigma\left(W_i \bullet \left[h_{t-1}, x_t\right] + b_i\right)$$

Figure 3. This equation is for the input gate, which decides what values will be updated by the LSTM cell. From Olah, Christopher.

$$\overline{C}_t = tanh\left(W_c \bullet \left[h_{t-1}, x_t\right] + b_c\right)$$

Figure 4. This equation creates an array of candidate values that can be used by the LSTM cell. From Olah, Christopher.

$$C_t = f_t * C_{t-1} + i_i * \overline{C}_t$$

Figure 5. This equation is where the old state is updated using the data from the forget and input gates. From Olah, Christopher.

$$\sigma_t = \sigma\left(W_o\left[h_{t=1}, x_t\right] + b_o\right)$$

Figure 6. This equation determines what part of the cell state will be outputted. From Olah, Christopher.

$$h_t = o_t * tanh\left(C_t\right)$$

Figure 7. This equation creates the final output, outputting only what is desired to be outputted and using tanh to output -1 or 1. From Olah, Christopher.

For my project, I experimented with two main diagrams. The first of which came from Dewis and Viana and can be seen in Figure 8. In my testing, I mostly had better results with a slightly less dense model, but the key features remained the same. This model can be seen in Figure 9. I used binary cross-entropy for the loss function, as did Dewis and Viana, and Shaaban et. al. The best optimizer I found in testing was NAdam and the best batch size I found was 32, so these were used. Dewis and Viana found that these were the best options in their testing as well. So, the best model that I found consisted of 1 input layer, 4 hidden layers, and 1 output layer and used a binary cross-entropy loss function, the NAdam optimizer, and a batch size of 32.

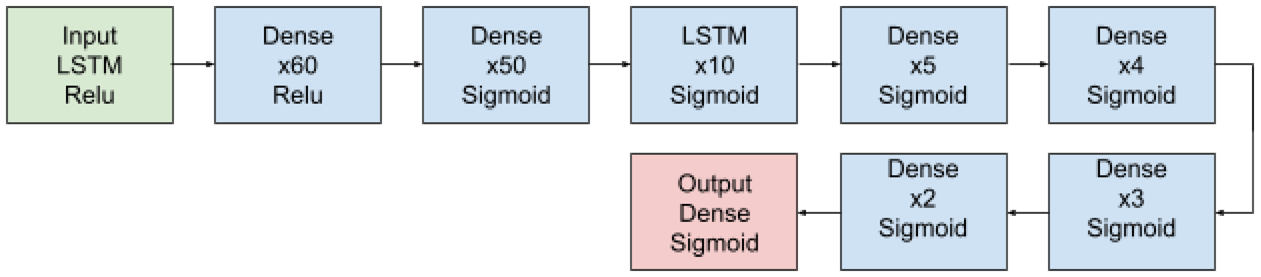

Figure 8. The model used by Dewis and Viana (p. 12).

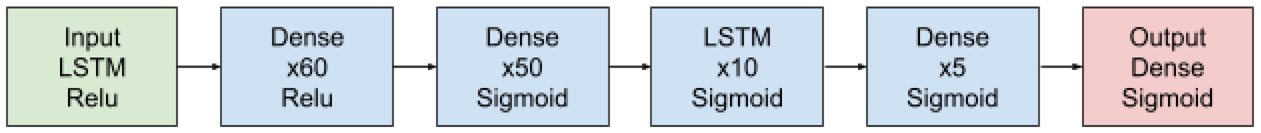

Figure 9. The best model I found during my testing, which was slightly adapted from Dewis and Viana.

Experiments

I experimented with the models and, initially, I was getting fairly poor results with the model Dewis and Viana used. I then trained a few variations of the model for 25 epochs each but still achieved much lower accuracy than they achieved, only around 74-75%. I hypothesized that this may be due to the fact that I had cut down on the testing data I was using. The dataset I chose provided 3 csv files of data, however, I ran into RAM issues when I attempted to combine and use all 3 files of data with the TF-IDF vectorizer. Because of this, I chose to only use one of the files for the models. After experimenting, I found that removing a few layers gave me the best results. I then decided to use that model and test various optimizers and batch sizes. I split my data into 80% training data and 20% testing data. For training, I used 10% of the training data as validation data and kept the testing data separate. I trained 16 models for 50 epochs each to compare the Adam, AdaMax, NAdam, and RMSProp optimizers with batch sizes of 10, 32, 64, and 128. All of the models use binary cross-entropy as the loss function. They were all compared using the accuracy and loss scores on the testing data. In my initial testing, I found that NAdam and a batch size of 32 gave me the best results with an accuracy of 99.06% (see Figure 10). I was interested in both the best accuracy and best loss, so Figures 10, 11, and 12 can be used to view the results of the training of those two best models. Both models had a macro f1-score of 0.99. I then used the settings from the model with the best accuracy to train a model for 100 epochs.

Model	**Optimizer**	**Batch Size**	**Test Accuracy**	**Test Loss**
Best Accuracy	NAdam	32	99.06%	0.057
Best Loss	AdaMax	64	98.96%	0.054

Figure 10. Results showing the models with the highest accuracy and lowest loss when the Adam, AdaMax, NAdam, and RMSProp optimizers as well as batch sizes of 10, 32, 64, and 128 were compared.

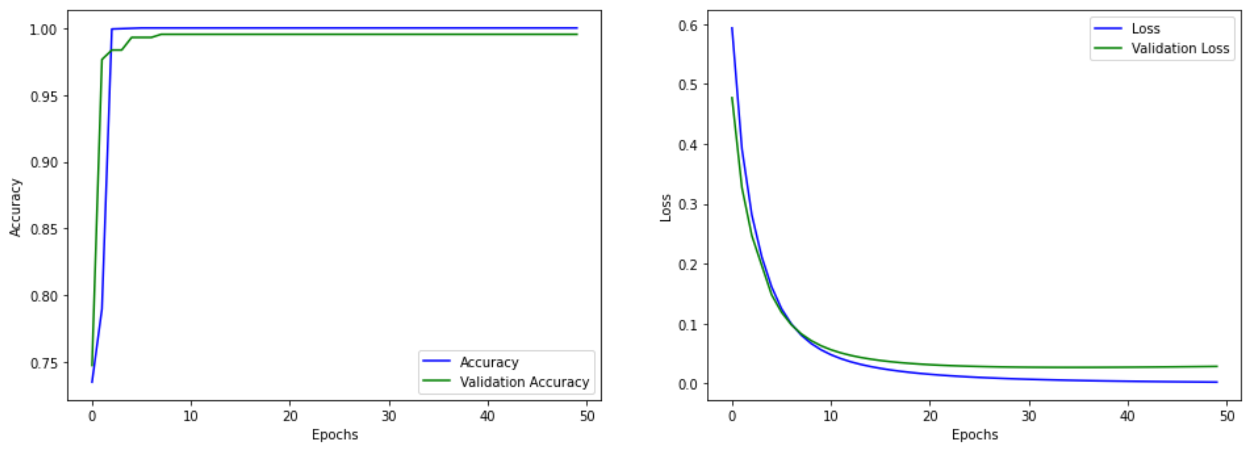

Figure 11. Results for the training and validation accuracy and loss for the model with the best accuracy, using the NAdam optimizer and a batch size of 32.

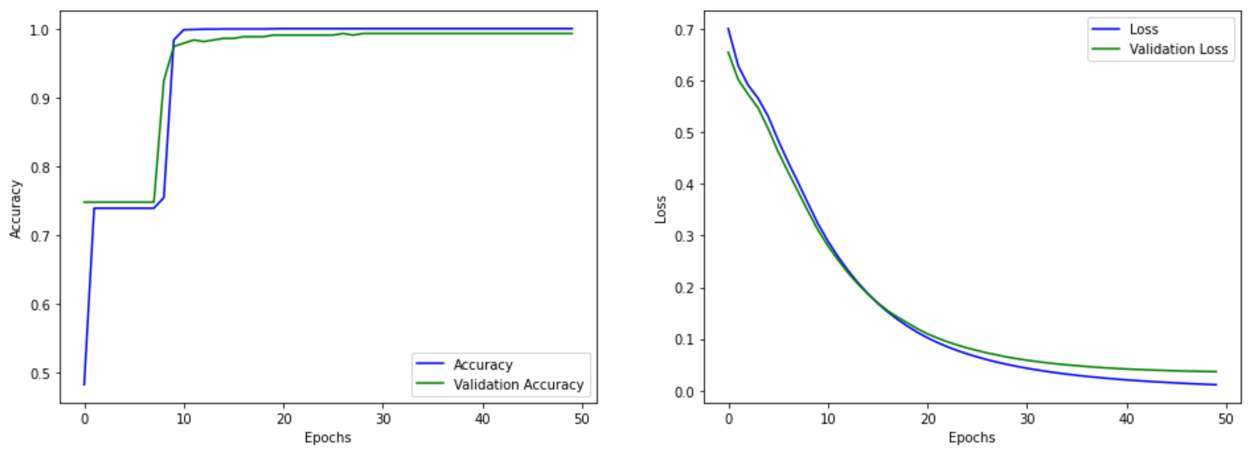

Figure 12. Results for the training and validation accuracy and loss for the model with the lowest loss, using the AdaMax optimizer and a batch size of 64.

In addition to testing to find the best parameters for the LSTM model, testing was also done so that the model could be compared to other types of models. These models were SGD, Perceptron, Gaussian Naive Bayes, and Random Forest. For the most part, the default hyperparameter values were used for these models. The Random Forest model was altered slightly and used 256 as the value for max leaf nodes. This was just slightly better than the accuracy using default parameters. The results for these models are shown in Figure 13. The best model found out of this group was the Perceptron model (highlighted green). However, the best accuracy out of all the models tested during this phase was the LSTM model with the NAdam optimizer and a batch size of 32.

Model	**Test Accuracy**	**Macro F1-Score**
SGD	97.83%	0.97
Perceptron	97.92%	0.97
Gaussian Naive Bayes	92.92%	0.90
Random Forest	95.93%	0.94

Figure 13. Results showing the models with the highest accuracy and lowest loss when the SGD, Perceptron, Gaussian Naive Bayes, and Random Forest were compared.

Test Results

The final model that I planned to train as my best model used an NAdam optimizer, a batch size of 32, the binary cross-entropy loss function, and 100 epochs. Once it finished training, it achieved an accuracy of 98.87% on the test data, with a macro f1-score of 0.99. The os was 0.0875. I was surprised to see this because I expected the accuracy to increase or stay the same, not decrease. Figure 13 shows the accuracy and loss for this model over time. Dewis and Viana found that 100 epochs were optimal, however, it seems that fewer epochs may be optimal for this model with slightly fewer layers. Supporting this, by comparing Figures 11 and 14, it appears that there are no major changes in training or validation accuracy after 50 epochs, however, validation loss slowly begins to increase. So, in summary, training the model that was found to have the best accuracy over 50 epochs for 100 epochs actually led to a decrease in accuracy on the test data. However, that simply leads me to believe that fewer epochs may be optimal and that high accuracy can be found in less time by training the model for fewer epochs.

Dewis and Viana reported an accuracy of 99% for their LSTM model (p. 14), Shaaban et. al reported accuracy of 98.38% for their deep convolutional forest model (p. 8), and the best model found by Trivedi was the SVM model which achieved a 93.3% accuracy (p. 179). In comparison with the results reported by other authors as well as other models created during this project, it can be seen that the accuracy of 98.87% is very good. The accuracy of the model in the earlier testing phase was higher, at 99.06%. Considering that the LSTM models I experimented with for this project did well as did Dewis and Viana's, I believe that LSTM models are very promising for detecting spam emails.

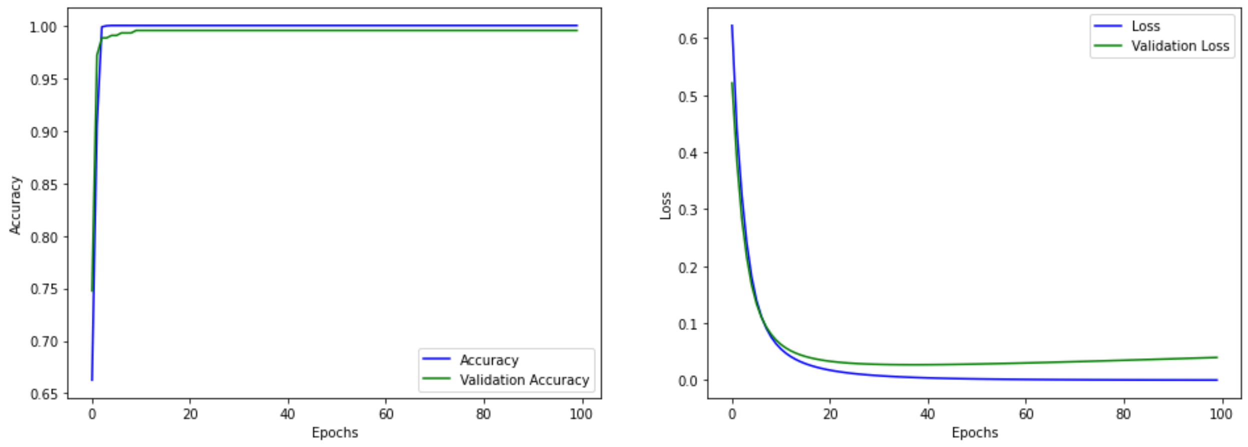

Figure 14. Results for the training and validation accuracy and loss using the NAdam optimizer, a batch size of 32, and the binary cross-entropy loss function over 100 epochs.

Conclusion

The journey of creating this project began with low accuracies and struggles working with Tensorflow to create the models. However, thanks to our class labs, the documentation, Stack Overflow, and trial and error, I was able to achieve an accuracy that I was happy with. I was not expecting to be able to create a model with an accuracy as high as ~99%, and it made me quite happy to do so. Some of the issues I experienced were expanding the input data properly, cleaning the data correctly, removing stop words, and running out of RAM while training models. I did not realize that Tensorflow's TF-IDF Vectorizor allowed you to easily remove stop words using it, for instance, and that would've been much easier than removing them in a lambda function. Another struggle I had was that, while I learned a great deal from our class labs, some of the labs focusing on the text data were actually after I had started working on the project and I struggled a little bit with getting things we did in our labs working properly for my project.

After I completed the project notebook and finished typing most of the report, I decided to train some of the earlier models with more epochs (the models with more layers, like Dewis and Viana) and one of the models achieved an accuracy of 99.15%. This was better than the other models, though it was within 0.10% of the model I previously found with the best accuracy so I don't believe it is significantly better. It just required more training time to see good results than I used for my initial testing. Since I had already typed most of the paper and it felt a little late to redo the best models I found, I decided not to rewrite the testing section of the paper, but did want to include it in the discussion section. If I were to do this again, I would start out with longer training times right away to test models better. I did not consider the fact that, in addition to the best optimizer and batch size, the best number of epochs is also important to find.

In summary, I learned a lot about working with text data including the importance of preparing the data properly through tokenization and methods such as TF-IDF, I learned about how LSTM models work, and I learned about the importance of testing models long enough in order to see their full potential and find the optimal number of epochs to train with. I also learned that training models can take a large amount of memory - thankfully, Google Colab was very helpful for that, even though it also had limitations. All in all though, I believe that I learned enough that I could tackle a similar problem and have an idea of how to begin and what testing to do to find a good solution. Machine learning is a very interesting topic to me and it was very exciting to learn more about it during class and during the course of this project.

My Contributions

A large part of my work was based on the LSTM model created by Dewis and Viana. Initially, I hoped that I could create something similar to their model in order to achieve good results. One thing I did not expect was how difficult it could be to go from the model architecture they presented to actually implementing it as a model. Much of the work I ended up doing was troubleshooting errors and getting things to work the way I wanted them to work, and most of my time was spent on training models with slight modifications to see what works well and what does not. Dewis and Viana did not mention some of the steps I did, such as converting numbers to their word equivalents (75 to seventy-five, for example), and I experimented a lot with minor changes to the model but most changes did not significantly change the accuracy achieved. I also attempted experimenting with the learning rate of the optimizer, but that did not seem to make a significant difference either. To get things to work well for me, I had to limit the data I was using to train and test the models, research solutions for problems I had (such as expanding the input data for use with the LSTM layers), and simply had to spend time experimenting with models.

Works Cited

Dewis, Molly, and Thiago Viana. “Phish Responder: A Hybrid Machine Learning Approach to Detect Phishing and Spam Emails.” MDPI, Applied System Innovation, 28 July 2022, https://doi.org/10.3390/asi5040073.

Nitisha. “Email Spam Dataset.” *Kaggle*, Nitisha, 30 Oct. 2020, https://www.kaggle.com/datasets/nitishabharathi/email-spam-dataset?resource=download

Olah, Christopher. “Understanding LSTM Networks.” Understanding LSTM Networks -- Colah's Blog, 27 Aug. 2015, https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

S. Dixon. “Global Average Daily Spam Volume 2021.” Statista, 28 Apr. 2022, https://www.statista.com/statistics/1270424/daily-spam-volume-global/.

Shaaban, Mai A., et al. “Deep Convolutional Forest: A Dynamic Deep Ensemble Approach for Spam Detection in Text ” SpringerLink, Complex & Intelligent Systems, 27 July 2022, https://doi.org/10.1007/s40747-022-00741-6.

Trivedi, Shrawan Kumar. “A Study of Machine Learning Classifiers for Spam Detection” IEEE Xplore, IEEE 4th International Symposium on Computational and Business Intelligence, 2016, https://ieeexplore.ieee.org/document/7743279.