

# SpamBuster

**Anthony J. Bustamante, Anurodh Acharya, Krishna Paudel**  
**CSS-576 (Winter-2023)**

## INTRODUCTION

SpamBuster is a lightweight, reliable, and easy-to-use security tool that scans your emails to identify spam. This tool is powered by a machine learning model behind the scenes which is hosted in the Google Cloud. This tool automatically scans the incoming emails as well as provides you with an option to manually scan the existing emails or folders. This tool comes in the form of an Outlook plug-in that can be used right from your Outlook Explorer window.

## DESIGN

There are three main components in the SpamBuster Solution: Outlook Add-In, ML Model, and Cloud Function

### A. SpamBuster Outlook Add-In

This is an Outlook plugin also known as Outlook Add-In built with C# on a .NET Framework. This application is the GUI interface for our Spam Ham Classifier project.

Features of the Add-In:

1. Automatically scans the newly received emails in any of the Outlook folders.
2. Provides On-Demand scans of the Outlook folders for potential spam.
3. Moves the Spam to the Junk Folder.
4. Interacts with Google Cloud Function through an API for prediction.
5. Asynchronous programming to make the Outlook program interactive during the operation.
6. Logs all the scanning activity into the folder: %LocalAppData%\SpamBusterLogs\

The SpamBuster Outlook Add-In was developed using Visual Studio Office Developer Tools on .NET Framework with C#. This solution includes

#### 1. ThisAddIn.cs

This class is responsible for starting the Add-In activities when Outlook starts. It sets the log4net configuration and initiates the logger. It registers a listener event in case a new email is received when the Outlook application is running. When a new mail arrives, the listener calls a method that captures the mail and sends it for further processing to predict the email category as spam or ham. If predicted as spam, it will move the email to the Junk Folder of the user's email account.

## **2. SpamBusterRibbon.cs**

This class is responsible for generating the SpamBuster UI which is a button in the Outlook Explorer Ribbon. At any given point, when this button is clicked, it will ask the user to pick a folder to scan. This is typically useful when you want to scan your email On-Demand or when you are installing the add-in for the first time and would like to scan the existing emails.

When an email is being scanned, this class has a method that collects the properties of the email like headers, email body, email subject, etc. to the `getPrediction()` method of `SpamBusterModel.cs` class for further processing. Once the email category is predicted, the email will be appropriately handled i.e., it will be disposed of to the Junk folder if it is found to be spam.

## **3. ProgressBar.cs**

This class is responsible for tracking the progress when a scan request is submitted. It shows the total number of emails it is scanning and the current progress. One can click "Cancel" to stop the scanning in the mid of the operation. It will also show Warning messages when you try to submit more than one operation while one is in progress.

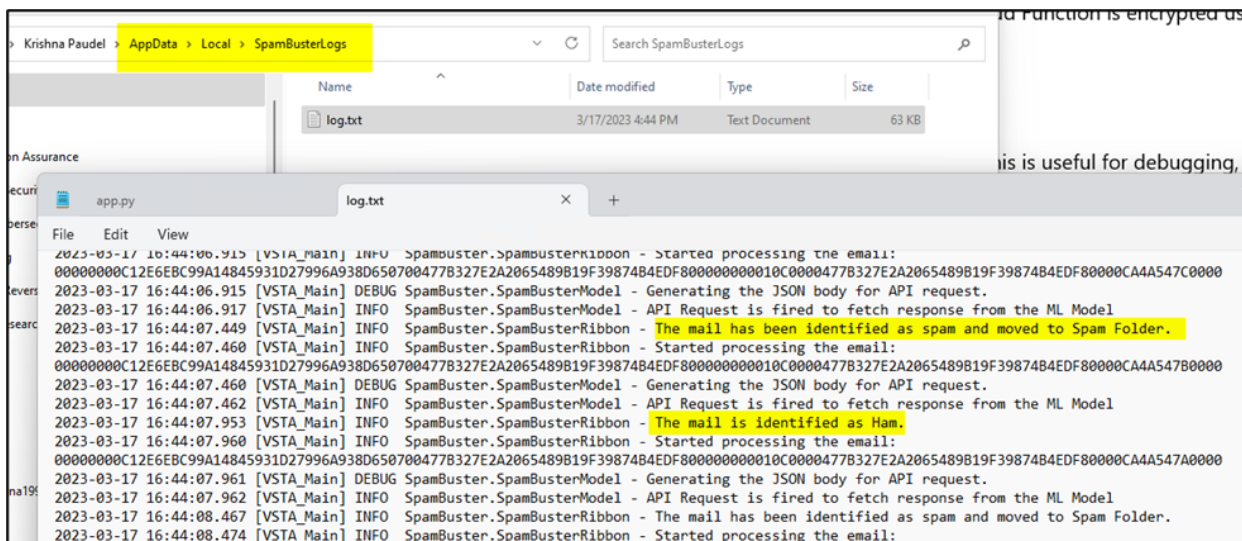
## **4. SpamBusterModel.cs**

This class includes the most important methods of the whole solution. It extracts useful authentication headers like SPF, DKIM, and DMARC flags that are used as raw features for prediction. Once all the features are collected, it sends this information to a SpamBuster Cloud Function through an API call that uses HTTPS POST Method. The API call submits the input to the function, which processes and cleans it before submitting it to the ML model for prediction. Once predicted, the result is sent back as a response to the HTTPS POST request. The communication between the Outlook Add-In and the API in Google Cloud Function is encrypted using TLS 1.2, hence all the data being transferred are encrypted and safe.

## 5. log4net config

Apache Log4Net is a logging framework that can be used to capture logs. This is useful for debugging, logging warnings and errors, and any information during the program execution. It is very easy to use and flexible to configure various useful options with a simple change in its configuration. In SpamBuster, the log4Net collects logs of various natures like INFO, WARN, ERROR, etc. along with a timestamp and relevant message.

The log files will be stored at %LOCALAPPDATA%\SpamBusterLogs\

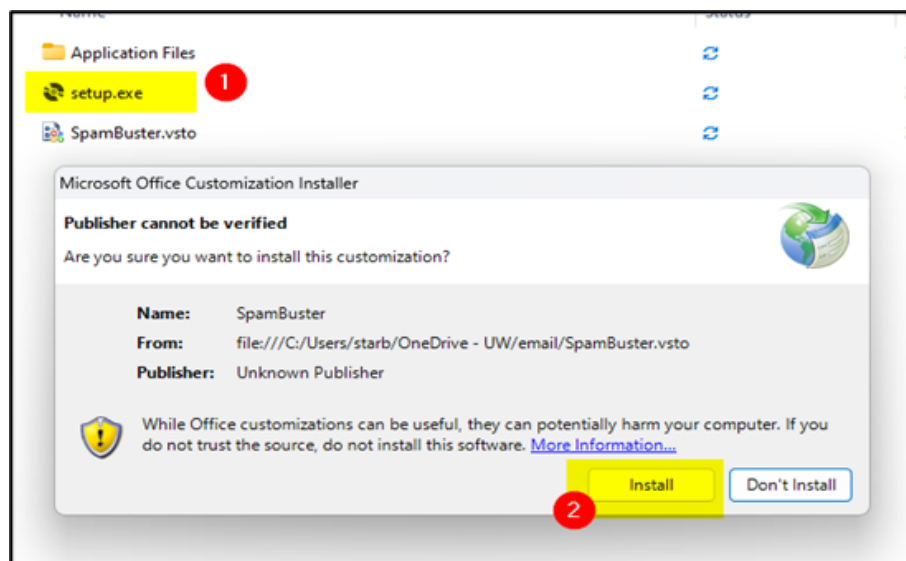


## How to use the Outlook Add-In?

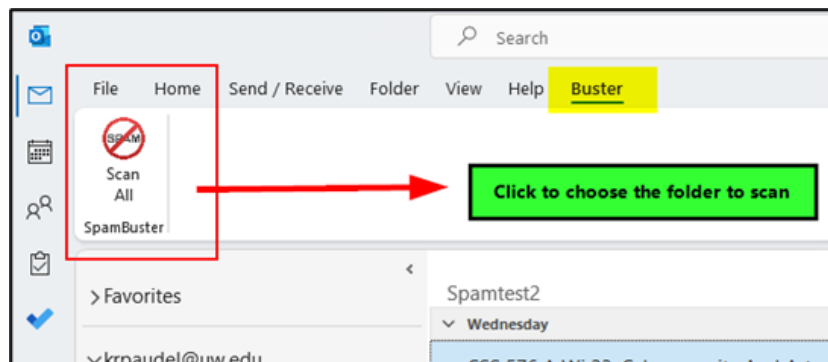
**DISCLAIMER:** Before you proceed with running the Outlook Add-In, please understand that if an email is predicted as SPAM, it will be moved from its original location to the Junk/Spam folder. The ML model is not 100% accurate as it is trained on the sampled dataset, so please be cautious while running it. We recommend running it in the folder of emails that are not important. Or you can also move back emails from the Spam/Junk folder if it's miscategorized. However, please be aware that most mailboxes retain spam for 30 days only and you may lose the data.

We have built a Click Once windows application that will install the plugin into your Outlook application.

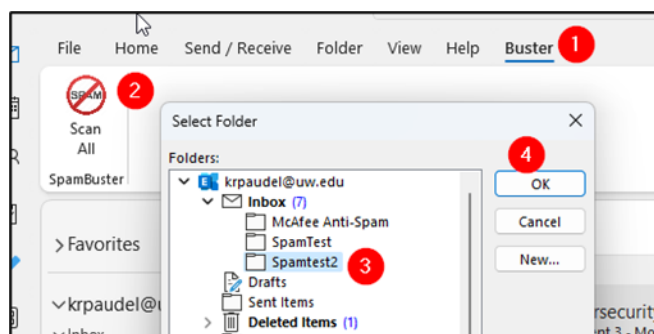
- Extract the compressed file SpamBuster-Outlook-Addin.zip to a local path.
- Double-click on the setup.exe file and Click Install
- Restart the Outlook application.



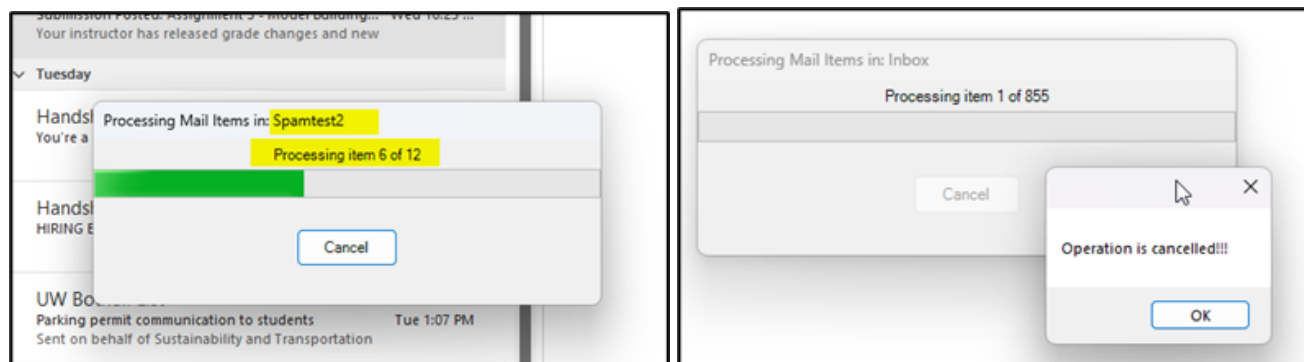
- Once installed, locate the Buster menu in the Outlook Explorer Ribbon. Click on Scan All.



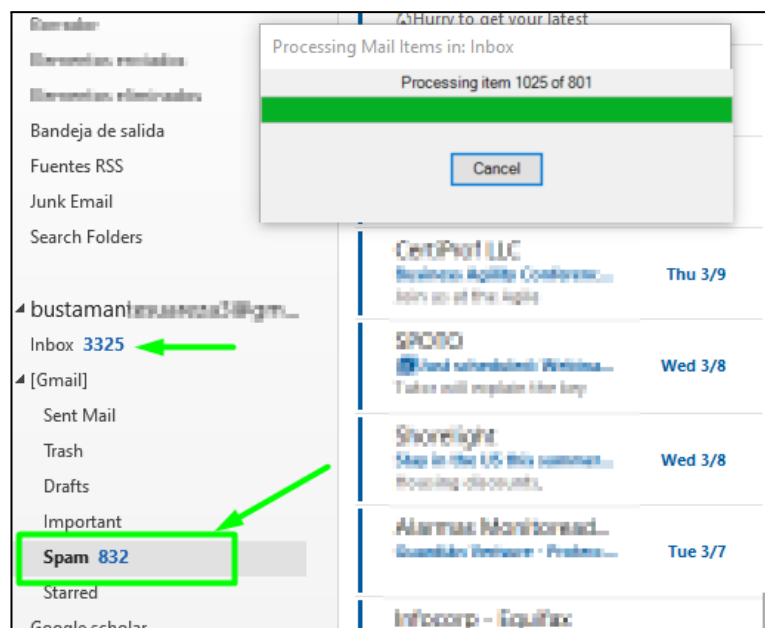
- Pick the folder you want to scan the emails from.



- Monitor the progress or cancel it during the mid of the operation.



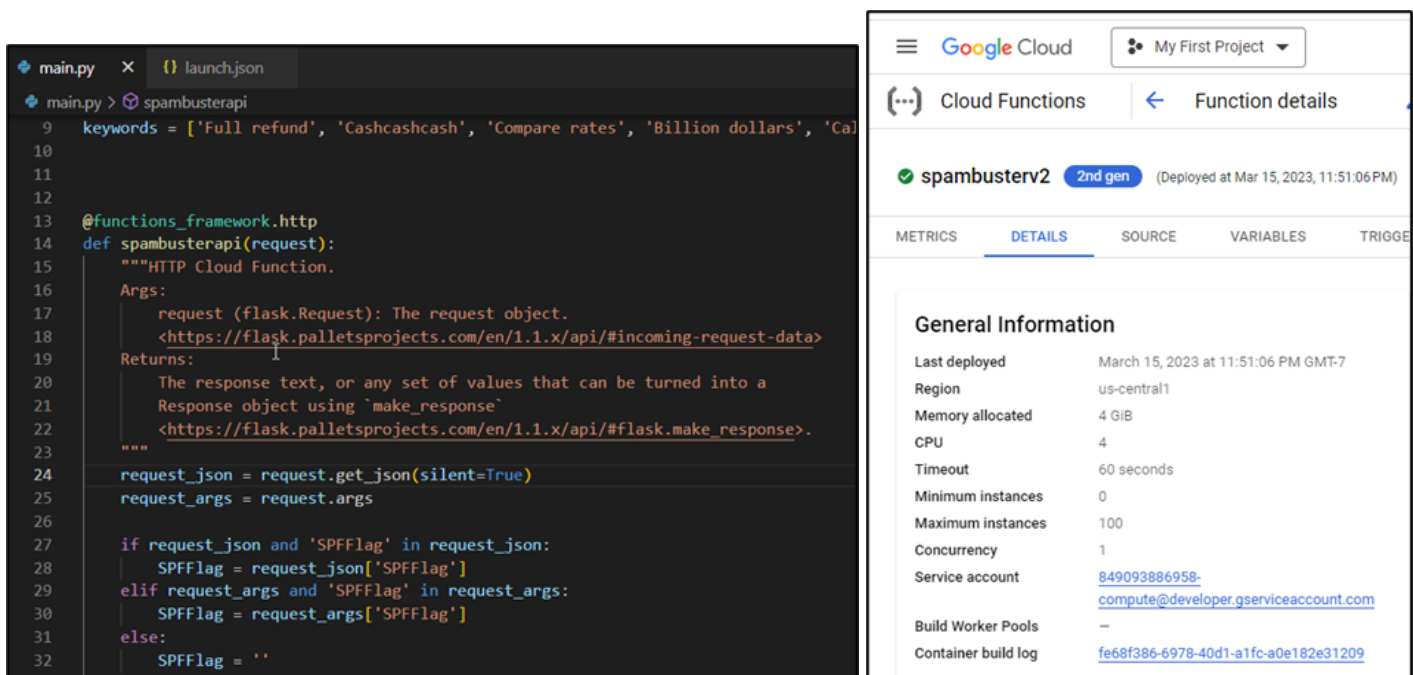
- You will notice that the mail identified as spam will be moved to the Junk or Spam folder.



## B. SpamBuster Cloud Function

The SpamBuster ML Function is a flask application deployed to the Google Cloud Function with a Python Environment as an API endpoint. This function accepts a HTTPS POST/GET Request through the API with the raw input features for the prediction of Ham or Spam emails. The input values are then parsed and preprocessed using the tokenizer and encoders pickled while training the model. Once the input values are encoded and tokenized, they are passed to the ML model which is restored using its pickle file. The model is then used to predict the email category.

The reason behind selecting GCP over Azure and AWS was the flexibility of scaling up the SKU of the function while using the free trial version. We have used 4 GB of Disk size and 2 vCPU to run the function on Google Cloud that serves as an API endpoint.



The image shows two side-by-side screenshots. The left screenshot is a code editor displaying the `main.py` file. The code defines a Flask API endpoint `spambusterapi` that takes a request and returns a response based on the 'SPFFlag' in the request JSON or args. The right screenshot is the Google Cloud 'Function details' page for a function named 'spambusterv2'. It shows deployment details: last deployed on March 15, 2023, at 11:51:06 PM GMT-7, in the 'us-central1' region, with 4 GB of memory allocated, 4 CPUs, and a 60-second timeout. The service account is `849093886958-compute@developer.gserviceaccount.com`.

```

9 keywords = ['Full refund', 'Cashcashcash', 'Compare rates', 'Billion dollars', 'Ca
10
11
12
13 @functions_framework.http
14 def spambusterapi(request):
15     """HTTP Cloud Function.
16
17     Args:
18         request (flask.Request): The request object.
19         <https://flask.palletsprojects.com/en/1.1.x/api/#incoming-request-data>
20     Returns:
21         The response text, or any set of values that can be turned into a
22         Response object using 'make_response'
23         <https://flask.palletsprojects.com/en/1.1.x/api/#flask.make_response>.
24     """
25     request_json = request.get_json(silent=True)
26     request_args = request.args
27
28     if request_json and 'SPFFlag' in request_json:
29         SPFFlag = request_json['SPFFlag']
30     elif request_args and 'SPFFlag' in request_args:
31         SPFFlag = request_args['SPFFlag']
32     else:
33         SPFFlag = ''

```

General Information	
Last deployed	March 15, 2023 at 11:51:06 PM GMT-7
Region	us-central1
Memory allocated	4 GiB
CPU	4
Timeout	60 seconds
Minimum instances	0
Maximum instances	100
Concurrency	1
Service account	<a href="#">849093886958-compute@developer.gserviceaccount.com</a>
Build Worker Pools	—
Container build log	<a href="#">fe68f386-6978-40d1-a1fc-a0e182e31209</a>

In short, the cloud function does the following in order:

1. Triggers when an HTTPS Request is received.
2. Parses the input sent in the JSON body
3. Cleans and pre-processes the input strings
4. Tokenizes the texts and encodes them using tokenizer and encoder pickle files.

Performance Metrics

We have captured Fiddler trace to determine the round trip time of the API Request and Response. The average round trip time taken for the request-response is approximately 1 second.

Progress Telerik Fiddler Classic

File Edit Rules Tools View Help

WinConfig Replay Go Stream Decode Keep: All sessions outlook:7052 Find Save Browse

#	Result	Protocol	Host	URL
1	500	HTTPS	outlook.office365.com	/EWS/Exchange
2	200	HTTP	Tunnel to	spambusterv2-afya...
3	200	HTTPS	spambusterv2-afya...	/
4	200	HTTPS	spambusterv2-afya...	/
5	200	HTTPS	spambusterv2-afya...	/
8	200	HTTPS	spambusterv2-afya...	/
9	200	HTTPS	spambusterv2-afya...	/
10	200	HTTPS	spambusterv2-afya...	/
11	200	HTTP	Tunnel to	outlook.office365.com
12	200	HTTPS	spambusterv2-afya...	/
13	201	HTTPS	outlook.office.com	/api/v2.0/Me
14	500	HTTPS	outlook.office365.com	/EWS/Exchange
15	401	HTTPS	outlook.office.com	/api/v2.0/Me
16	201	HTTPS	outlook.office.com	/api/v2.0/Me
17	200	HTTPS	spambusterv2-afya...	/
18	201	HTTPS	outlook.office.com	/api/v2.0/Me
19	200	HTTPS	spambusterv2-afya...	/
20	201	HTTPS	outlook.office.com	/api/v2.0/Me
21	200	HTTPS	spambusterv2-afya...	/
22	200	HTTPS	outlook.office365.com	/api/emsmd

Get Started Statistics Inspectors AutoResponder Composer FO

Request Count: 1  
Bytes Sent: 912 (headers:221; body:691)  
Bytes Received: 386 (headers:375; body:11)

ACTUAL PERFORMANCE

This traffic was captured on Thursday, March 16, 2023.

ClientConnected: 21:41:47.057  
ClientBeginRequest: 21:41:47.153  
GotRequestHeaders: 21:41:47.153  
ClientDoneRequest: 21:41:47.520  
Determine Gateway: 0ms  
DNS Lookup: 0ms  
TCP/IP Connect: 0ms  
HTTPS Handshake: 0ms  
ServerConnected: 21:41:47.082  
FiddlerBeginRequest: 21:41:47.520  
ServerGotRequest: 21:41:47.520  
ServerBeginResponse: 21:41:47.538  
GotResponseHeaders: 21:41:47.662  
ServerDoneResponse: 21:41:47.662  
ClientBeginResponse: 21:41:47.662  
ClientDoneResponse: 21:41:47.662

Overall Elapsed: 0:00:00.508

RESPONSE BYTES (by Content-Type)

~headers~: 375  
text/html: 11

Some of the other performance metrics on the Cloud Function (API) is attached below:



From the above snapshots, we can see that the performance counters like Memory, CPU utilization is pretty when we were load testing it by sending the 200+ request in sequence with no delay. The response is served within less than a second or in few milliseconds making the API solution pretty feasible with the deployed ML model.

### C. SpamBuster ML Model

#### Introduction -- What problem are you trying to solve and what was your basic approach.

People receive a lot of emails and most of the emails received are either spam or phishing emails. We implemented a solution to detect the received emails as either ham or spam(unwanted email). This will help the users to separate out the most essential emails that they receive and this will also help prevent phishing-spoofing-related attacks. Also, many users want to separate out the important emails and the emails which are not useful. Thus they look for a feature or an application that can do this with high accuracy so that



they don't have to take the extra burden of manually checking an email and classifying them as spam or an important email.

We downloaded our own emails that were categorized as spam in our Gmail(from assignment 1). Further, we processed our data using various techniques and created a clean dataset. In order to reduce bias, we also downloaded the Enron dataset from the internet, preprocessed it to match the format with our previous dataset, and then merged it. We used Neural Networks to test our model and also Logistic classifier.

For integrating the spam/ham classification in Outlook, we created an Outlook plugin that queries the API with the incoming email details and then receives a response which indicates that whether the received email is spam or ham. Based on this, this plugin will sort the email in the appropriate folder in Microsoft Outlook.

Overall, this approach involved the use of machine learning techniques to develop an effective spam/ham classifier and further integrate it with email software in order to provide a good user experience.

Something to note in the next questions is that we used results like SPF, DKIM, or DMARC for this project because the concept of SPAM is too subjective, e.g. some advertisements can be spam for some people but not for others, therefore we have not used a specific standard as we focus more on preventing phishing and spoofing emails, although our project is also very good at predicting the most common "spam" 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?**

**Describe how and where you collected your data.**

We collected emails from the inbox and spam folder present in the email clients. For this, we wrote a python code and subscribed to several websites so as to receive a high number of spam emails (Assignment 1). Further, we also used Enron dataset from the internet and merged it with our own dataset. We created python scripts, and PowerShell scripts to parse the raw data and extract useful information that can be employed as raw features.

**Why is this data good for your particular security application?**

Our data consists of two different datasets, one of which includes our own emails and the other is the Enron dataset. The emails extracted from our own dataset are useful in training our model with the new data that

contains spam emails that represent the latest trend. On the other hand, the Enron dataset will provide us with data consisting of spam/phishing techniques used in the early days. So this gives us the ability to handle every time of spam/phishing emails and hence it is considered to be good for our security application.

### **Reflect on any limitations in the process such as bias that may have been introduced.**

One limitation of this process is the fact that many email clients will categorize promotional emails as spam emails. Hence while we collected this data from the same email client, we might have trained our model to detect these promotional emails as spam emails.

Further, since we looked at the SPF, DKIM, and DMARC records to categorize the incoming email as either spam or ham, there is a possibility that some spam emails, even though they aren't legitimate, have been trained to be classified as ham.

The majority of our training data included promotions that are not exactly spam but several users tend to mark them as spam. This imbalance in the data distribution could have introduced bias, however, we got rid of most of such cases to reduce the bias as much as possible.

### **How did you balance class sizes?**

Even though we had a significantly high number of non-spam emails as compared to spam emails, we took both types of emails in a similar ratio so as to balance the classes between spam and ham. Initially, we had only a handful number of spam emails, however, we wrote a python code to subscribe to several websites (Assignment 1) so as to receive more spam emails. This helped us to create a dataset with a similar ratio.

The second dataset (Enron dataset) that we used was already evenly balanced i.e. 16k and 14k spam.

**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 with preprocessed data, note that here.**

### **How did you go about deciding on features.**

We used domain knowledge to decide on the essential features. We observed that many of the spam/phishing emails seem to have failed either SPF, DKIM, or DMARC test. We decided to use this knowledge and trained our model. Further, we observed that spam emails tend to have certain keywords such as "Full refund "Free money",

etc. We built a keywords list that could also be compared with the email's subject and body to categorize the incoming email as spam(unwanted email) or ham.

### If you used feature reduction, what did you do?

We did not use dimensional reduction(feature reduction) for our dataset because we performed a comparative analysis of the response time and found it to be much faster in responding compared to the one with dimensional reduction.

To recollect the dataset, we used around 20k emails collected in assignment 1 and +30k from the external dataset(Enron Raw Dataset), we wrote a scraping and parser program to extract all the data from .mbox/.eml files containing the emails.

### What steps were necessary to collect those features from the data.

There were several steps involved in collecting the features from the data. We first downloaded the emails from our Inbox and Spam folder from the email client. We kept the spam and ham folders separately before processing. Then we wrote a python code to go through every spam email and look for several features. For every email to be processed, we first checked if the email has an 'Authentication-Results' present in the email. If that is present, we extracted the SPF record, DKIM record, and DMARC using various regular expressions. We added a label(ham/spam) in the dataset based on the folder we were reading from. Further, we extracted the subject portion of the dataset and appended that as a feature in the column of the dataset.

Overall the final dataset that we utilized is observed in the following table:

Subject	SPF	DKIM	DMARC	Word pattern	Label
EE Agrmts - URGENT	neutral	neutral	pass	Non-spam patterns	ham
RE: Contracts	neutral	pass	pass	Non-spam patterns	ham
FW: EWS A&A PRC reps	pass	pass	none	Non-spam patterns	ham
Re: Confidential	pass	pass	none	Non-spam patterns	ham
Loose your Fat in 9 Days	neutral	temperror	temperror	Has spam patterns	Unwanted email
Loose your Fat in 9 Days	temperror	permerror	fail	Has spam patterns	Unwanted email
Love pills - low price	permerror	permerror	none	Has spam patterns	Unwanted email
Loose your Fat in 9 Days	neutral	permerror	none	Has spam patterns	Unwanted email
Better pricing means more savings to you	neutral	neutral	fail	Has spam patterns	Unwanted email
Loose your Fat in 9 Days	temperror	fail	quarantine	Has spam patterns	Unwanted email
Viagra - 75% OFF	neutral	temperror	temperror	Has spam patterns	Unwanted email

We merged our collected data with the Enron dataset from the internet.

**If you augmented with preprocessed data, note that here.**

Further, we did not augment the dataset as both Enron and our dataset were captured in raw format.

**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 created a Python script that performs pre-processing steps on a spam email dataset. The dataset was loaded using the Pandas library and split into training and testing sets using the `train_test_split` function from the scikit-learn library.

To preprocess the data, we created three functions. The first function, `label_encoding`, encodes categorical features in the dataset using the `LabelEncoder` class from the scikit-learn library. This function takes as input the training and testing datasets, as well as the name of the categorical column to be encoded. The function first fits the `LabelEncoder` on the training data, and then applies the same encoding to the testing data. The function then returns the transformed training and testing datasets.

The second function, `onehot_encoding`, performs one-hot encoding on the categorical features in the dataset using the `OneHotEncoder` class from the scikit-learn library. This function takes the training and testing datasets, as well as a list of columns to be one-hot encoded. The function first fits the `OneHotEncoder` on the training data, and then applies the same encoding to the testing data. The function then returns the transformed training and testing datasets.

The third function, `count_vectorizer`, creates a bag-of-words representation of the text data using the `CountVectorizer` class from the scikit-learn library. This function takes as input the training and testing datasets, and applies the `CountVectorizer` on the text column of the dataset. The function returns the transformed training and testing datasets.

**What model(s) did you use.**

After the pre-processing steps, the pre-processed datasets are used to train a logistic regression model using the `LogisticRegression` class from the scikit-learn library. The trained model is then used to make predictions on the testing data, and the performance of the model is evaluated using the confusion matrix, precision, recall, F1-score and accuracy metrics from the scikit-learn library.

## How did you decide on hyperparameters?

Furthermore, we aimed to find the best hyperparameters for a logistic regression model that would give us the highest accuracy and precision scores. To accomplish this, we employed a GridSearchCV object, which allowed us to search through a range of hyperparameters and evaluate their performance using 5-fold cross-validation. Also, we used **Precision** and **Accuracy** to choose the best hyperparameters possible, in our model our labels are tagged as 1=ham and 0=spam, because we are looking for the best Precision value, that would indicate that a high proportion of messages predicted to be ham are actually ham. In other words, the classifier is not incorrectly labeling spam messages as ham very often. This is an important metric for a spam classifier because it is generally more important to avoid false positives (i.e., avoid labeling a ham message as spam) than false negatives (i.e., avoid labeling a spam message as ham).

In the end, we did not see any substantial improvement with different hyperparameters because, with all of them, we got high Precision and Accuracy(around 0.99), hence, we use the default hyperparameters set up by sklearn.

Something very important to keep in mind is that our objective was besides doing an accurate model also to do a fast model because, for this project, the model is hosted in the cloud

Note: for the results, we tagged every "HAM" email in our model as "1", and every "unwanted email/spam" as "0".

## What were the results?

### Results:

#### A. Precision, Recall, F1 Score, and Accuracy:

**Precision:** 0.9829678935003915

**Recall:** 0.9962301587301587

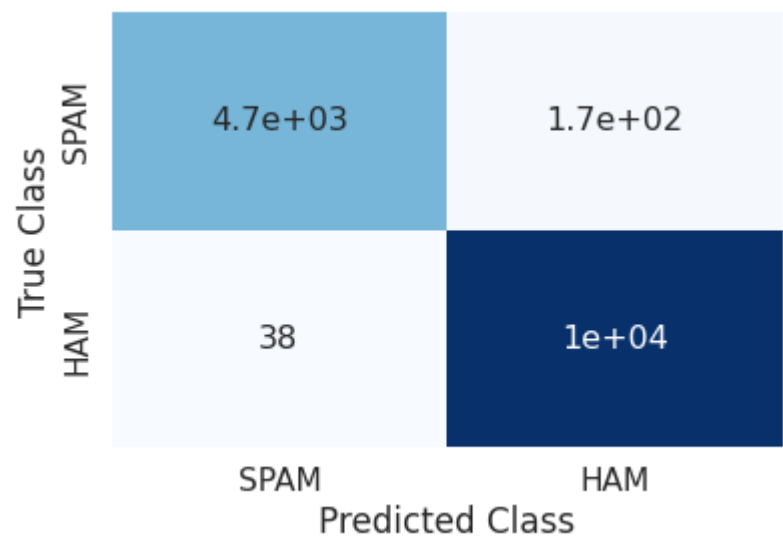
**The F1 score is:** 0.9895545920378399

**The accuracy:** 0.9858666666666667

The Precision of 0.982 means that out of all the predictions the model made for a particular class, 98.2% were actually correct. The Recall of 0.996 indicates that out of all the actual instances of that class in the data, the model was able to correctly identify 99.6% of them. The F1 score of 0.989 provides a harmonic mean of precision and recall. Finally, the accuracy of 0.986 suggests that the model was able to predict the correct class

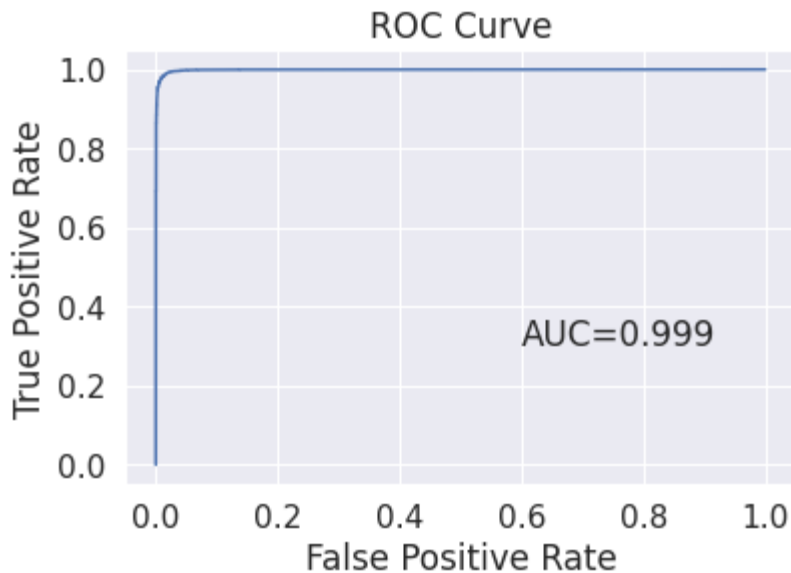
of 98.6% of the total instances. These results indicate that the model has a high level of accuracy and ability to correctly identify both positive and negative classes.

**B. Confusion Matrix:**



Overall, the confusion matrix shows that the model has a high number of true positives and true negatives, indicating good performance in identifying positive and negative cases. However, there is still room for improvement in the model's performance due to the false positives and false negatives shown to be as low as possible.

**C. ROC Curve and AUC:**



For AUC and ROC we can see that the AUC is 0.999, which is very close to the optimal value of 1. This means that the model has a high predictive power and is capable of distinguishing between positive and negative classes with a high degree of accuracy.

Furthermore, the ROC curve is also close to 1, which means that the model has a high true positive rate and a low false positive rate. This indicates that the model is very good at correctly classifying positive cases and is less likely to misclassify negative cases.

Overall, since our model has an AUC of 0.999 and a ROC curve close to 1, we can interpret that our model is performing very well and is highly accurate in its predictions.

**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.**

**Did you write your application as a part of another system or a standalone application.**

We created an outlook plugin, a standalone solution that on receiving any new emails, parses the received email, preprocesses the email, and tests it to check whether the email is a ham or spam. Once it tests and finds the result, it will send the spam/malicious email to junk/spam folder.

## **Performance Metrics**

The speed at which the result was produced was quite impressive. It was able to predict and produce the result of spam/ham in less than a second which is very good in terms of computational performance. Users will not wait for email filters to run for long periods of time before accessing their emails so it was important for us to reduce the time required to verify the category of the email. Further, we tested our plugin and deployed it to test in a real-world scenario. Interestingly, it was able to classify the email as ham or spam with an accuracy of over 90% which showed us that it was very good with its classification. The model was able to predict approximately 100 emails per second.

### **A. Precision, Recall, F1 Score, and Accuracy:**

**Precision:** 0.9829678935003915

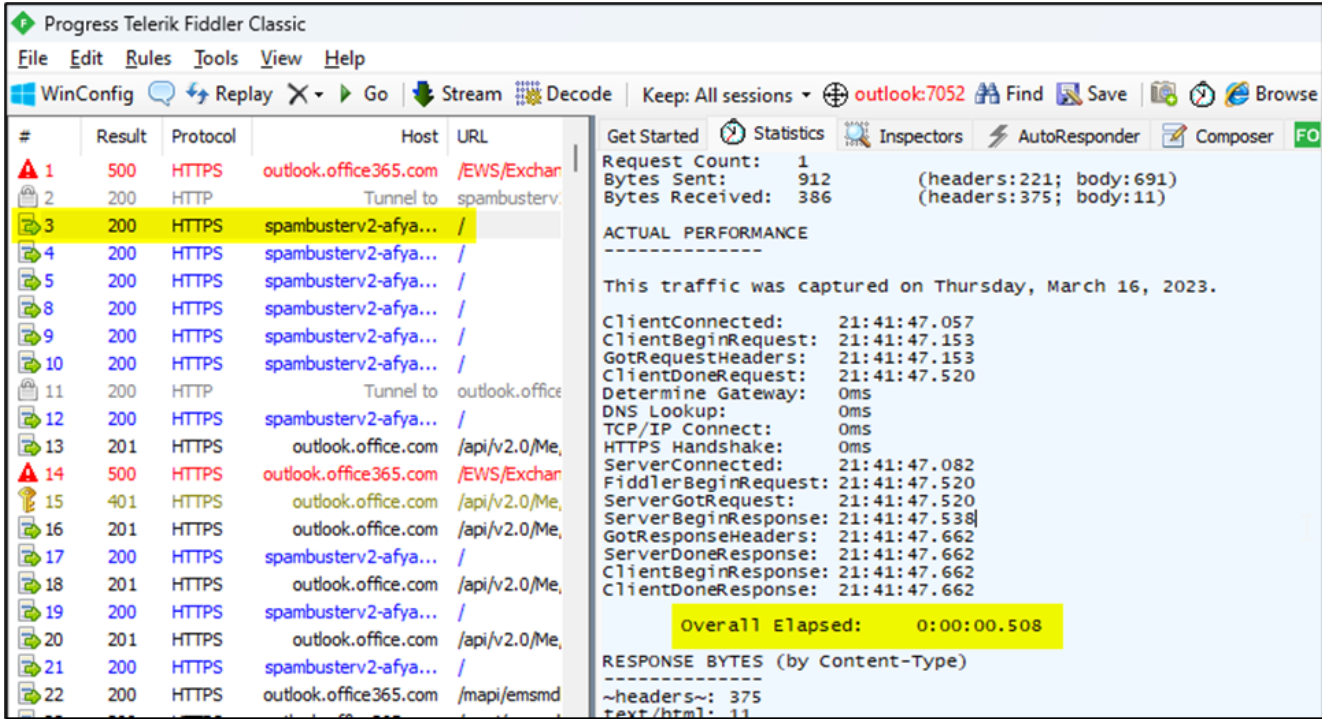
**Recall:** 0.9962301587301587

**The F1 score is:** 0.9895545920378399

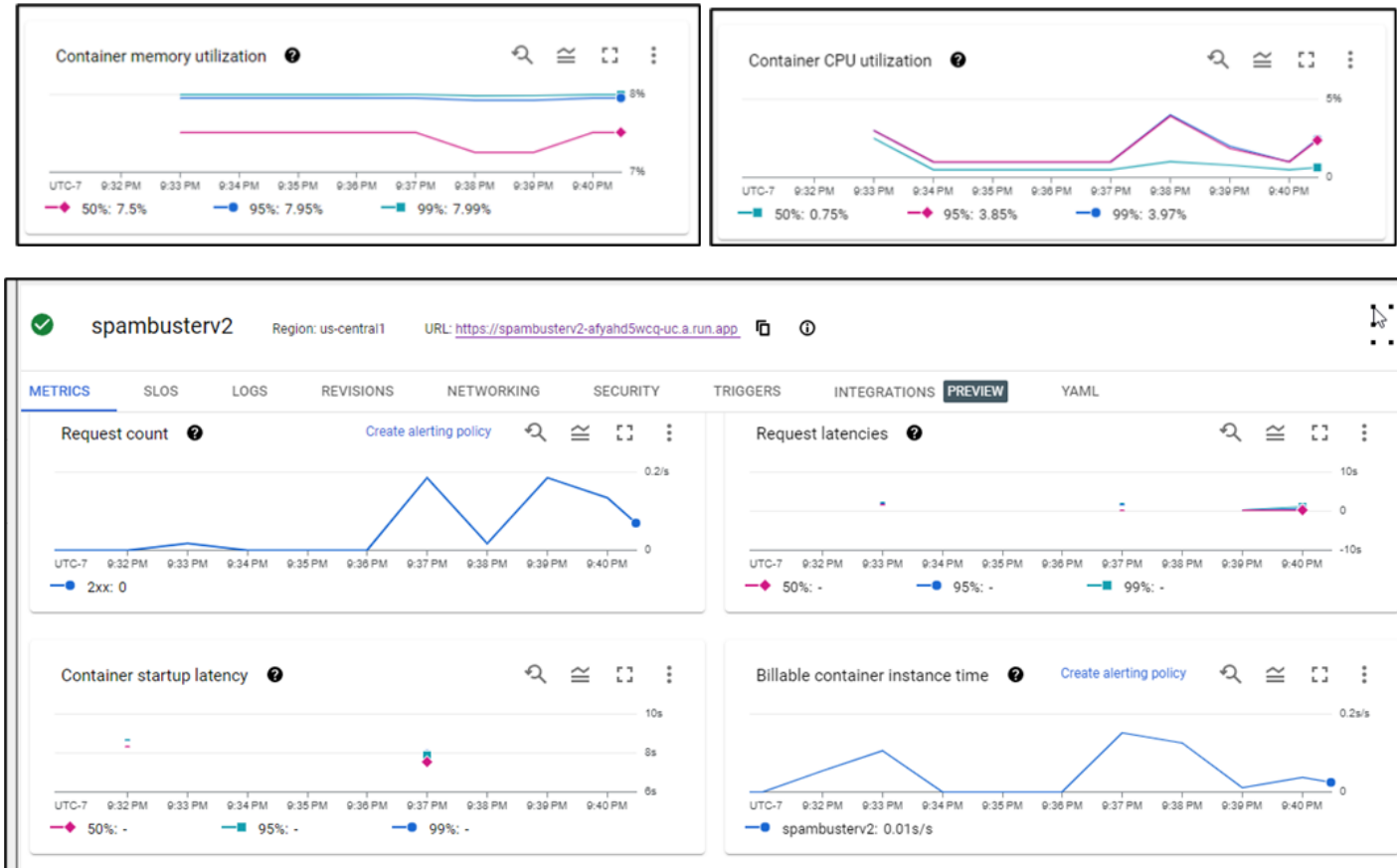
**The accuracy:** 0.9858666666666667

We have captured Fiddler trace to determine the round trip time of the API Request and Response. The average round trip time taken for the request-response is approximately 0.5 seconds.





Some of the other performance metrics on the Cloud Function (API) is attached below:



From the above snapshots, we can see that the performance counters like Memory, and CPU utilization is pretty when we were load testing it by sending the 200+ request in sequence with no delay. The response is served within less than a second or in a few milliseconds making the API solution pretty feasible with the deployed ML model.

## **Usability**

There were several approaches in deploying the model and building solutions to classify the emails. But it was important to consider the usability and effectiveness of this application before deciding on the approach. Our plugin appears on top of outlook where users can simply choose to scan a particular folder(eg. inbox) of the email client. In less than a second, it will process each email and predict the category of the email and put it in its respective folder. Further, for every new incoming email, it will scan and send it to a particular folder depending on the result of the testing of the email. Further, in order to make sure that even non-technical people can easily use email, we tested our application with several people who are not tech-savvy. We received their feedback and concluded that this application is easy to get started with, easy to use, and also provides timely prediction.

## **Limitations**

Some of the limitations include the fact that there are some instances in which the email records(SPF, DKIM, DMARC), even if they pass, tend to contain spam/malicious emails which could make the model predict the email as ham. If the email does not contain mostly observed spam words and passes all the email records, the model might predict the email as ham even if it could contain malware as an attachment. Another limitation could be, even though our email prediction model is fast to respond to, it might not be fast enough to process millions of messages in a day.

## **ART - Adversarial Robustness Toolbox:**

Some “evasion” adversarial attacks that can be used against our logistic binary text classifier that are available in ART are:

**Fast Gradient Sign Method (FGSM):** This is a simple and effective attack that perturbs the input data by adding a small magnitude of the gradient of the loss function with respect to the input.

**Projected Gradient Descent (PGD):** This is a stronger attack than FGSM that iteratively perturbs the input data by taking small steps in the direction of the gradient of the loss function while constraining the perturbations to lie within a certain range.

**DeepFool:** This is an iterative attack that aims to find the minimum adversarial perturbation required to misclassify an input sample, by iteratively linearizing the decision boundary and computing the distance between the input and the boundary.

**Carlini-Wagner (CW) attack:** This is a powerful optimization-based attack that minimizes the distance between the adversarial example and the original input, subject to a constraint that the adversarial example should be misclassified.

To defend against these attacks, you can use various techniques, such as adversarial training, input preprocessing, and defensive distillation, which are also supported by ART.

From all the previously mentioned attacks we used **FGSM** attack to evaluate our model:

For this experiment, we did not use all the dataset that we used to construct our original model, but just half of it(25K email samples) for performance issues, we did generate the model again and we also got quite high results for Precision, Recall, F1 Score, and Accuracy(close to 0.99). In the following code, you can see that we took our “ann” model and rename it as “clf” to avoid any confusion:

```

from art.attacks.evasion import FastGradientMethod
from art.estimators.classification import SklearnClassifier
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Train a logistic regression classifier on the training data
clf = ann
# Wrap the logistic regression classifier with ART's SklearnClassifier
classifier = SklearnClassifier(model=clf, clip_values=(0, 1))
# Define the FGSM attack with ART
epsilon = 0.1
fgsm = FastGradientMethod(estimator=classifier, eps=epsilon, targeted=False)
# Generate adversarial examples for all samples in x_test
adv_vec_list = []
for sample_vec in x_test:
    adv_vec = fgsm.generate(x=np.expand_dims(sample_vec, axis=0))
    adv_vec_list.append(adv_vec)

adv_vec_all = np.vstack(adv_vec_list)
# Evaluate the model's accuracy on the original and adversarial examples
pred_label = clf.predict(x_test) #Prediction of classes for the original test dataset
adv_pred_label = clf.predict(adv_vec_all) #Prediction of classes for the test dataset crafted with ART

```

We used logistic regression to train a classifier on a dataset and then implemented ART's SklearnClassifier to wrap the logistic regression model. We then used Fast Gradient Sign Method (FGSM) to generate adversarial examples for all samples in the test data. FGSM is an attack technique that computes the gradient of the loss function with respect to the input data and perturbs it in the direction that maximizes the loss, thus creating an adversarial example.

To generate the adversarial examples, we set an epsilon value of 0.1 for FGSM, which controls the strength of the attack. We then iterated over each sample in the test data and generated an adversarial example for it using the FGSM attack. The resulting adversarial examples were stored in a list and then converted to a 2D matrix for further analysis.

Finally, we evaluated the performance of our model on both the original and adversarial examples using accuracy, precision, recall, and F1 score metrics. We calculated the prediction of classes for both the original and adversarial test datasets using our trained logistic regression model. Overall, our results showed that the accuracy, precision, recall, and F1 score of our model were lower on the adversarial examples than on the original ones. This demonstrates the vulnerability of machine learning models to adversarial attacks and highlights the importance of developing robust models that can withstand such attacks.

**ART's Result:** From the following figure we can see that FGSM varied just a little bit the testing data to fool our logistic classifier.

Original Dataset for testing:

```
*****
[[0 0 0 ... 1.0 0.0 0.0]
 [0 0 0 ... 0.0 0.0 0.0]
 [0 0 0 ... 1.0 0.0 0.0]
 ...
 [0 0 0 ... 1.0 0.0 0.0]
 [0 0 0 ... 1.0 0.0 0.0]
 [0 0 0 ... 1.0 0.0 0.0]]
```

Dataset crafted with ART

```
*****
[[0.0 0.0 0.0 ... 0.9 0.0 0.1]
 [0.0 0.0 0.0 ... 0.0 0.0 0.1]
 [0.0 0.0 0.0 ... 0.9 0.0 0.1]
 ...
 [0.0 0.0 0.0 ... 0.9 0.0 0.1]
 [0.0 0.0 0.0 ... 0.9 0.0 0.1]
 [0.0 0.0 0.0 ... 0.9 0.0 0.1]]
```

Furthermore, our results went from being close to 0.99 to less than 0.01, which shows how impactful ART can be against machine learning.

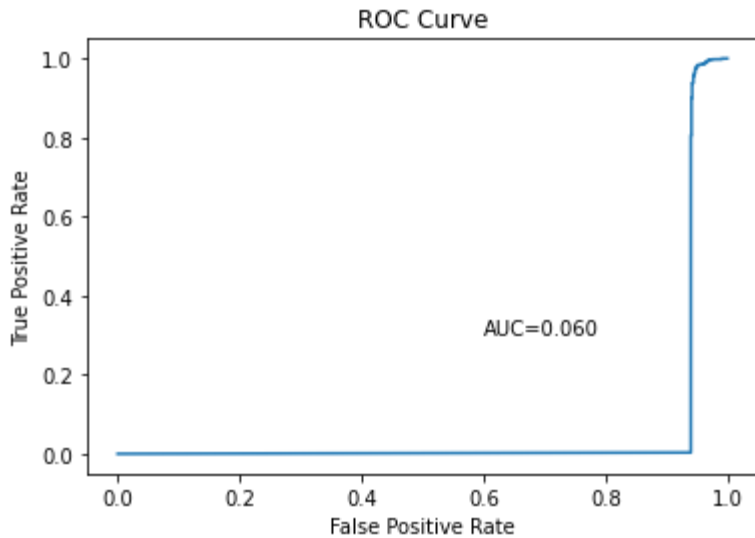
Original Results:

```
*****
Precision on original examples: 0.9790184921763869
Recall on original examples: 0.996741491672701
F1 score on original examples: 0.9878005023322569
Accuracy on original examples: 0.9818666666666667
```

Adversial Results:

```
*****
Precision on adversarial examples: 0.009594882729211088
Recall on adversarial examples: 0.0032585083272990588
F1 score on adversarial examples: 0.0048648648648648655
Accuracy on adversarial examples: 0.018133333333333335
```

Finally, we can see that the ROC curve after ART is disastrous, and the AUC is now equal to 0.060 from being previously 0.999, which one more time shows how FGSM(ART) can impact on our models negatively.



**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.**

**What take-aways do you have from the project.**

We learned a lot during the course of completing this project. In the classroom, we theoretically understood the concepts involving every aspect of building a secure ML application but this project gave us hands-on practice on various steps involved in developing machine learning-based applications which include data collection, preprocessing, deciding on the model, model building, and model deploying. While trying to deploy the model, both integrated within the plugin as well as in the cloud, we learned various cloud features, creating API, creating plugins, and so on. We think we are now more comfortable with ML than we were 3 months ago.

**If you ran into performance problems with your application, what did you do to address them.**

We took the Enron dataset from the internet to initially classify the emails as spam or ham. The performance was satisfactory while testing the emails from the same dataset. However, on testing newer emails from 2023, it performed relatively poorly and the performance was not satisfactory. In order to make it more relevant to the current context, we downloaded several spam emails from our email clients and merged it with the Enron dataset. Our expectation was that with the Enron dataset, it will cover spam/phishing emails that were prevalent during the early days. When merged with the latest emails, it would provide better results for all types of emails.

As expected, the performance of the model was significantly improved. Also, we spent approximately 8 hours deploying to the appropriate cloud as most of the working solutions were not free.

While initially deploying the model in the cloud, it took more than 5 seconds to receive the response of the request sent to the API endpoint. This caused significant lag which would decrease the usability of our model. We then decided to scale up the cloud infrastructure by increasing the CPU count. This helped us reduce the response time to up to 200 ms per request.

### **What we would do differently if we started again.**

While progressing in this project, we understood that there were several other features that would be important for us in classifying the email as either ham or spam. One of them includes the presence of the attachment file and the type of attachment file. We would definitely include this feature as a part of our dataset if we did start again.

Further, due to a lack of higher computational power, we could not train all of our collected data which would take days to complete. However, if we started again, we would look at purchasing cloud resources and deploying our model with high computation power so as to better prepare the model that provides us with higher metric values.

### **What would you do in the future?**

Our application provides easy-to-use functionality to the users where they can easily specify the emails to scan and also get new emails to be scanned automatically. Now, further, in the future, we will provide several functionalities such as providing an email alert or a message to the user if the email is considered to be spam. It could result in a lot of alerts so we would integrate this functionality, especially for malicious phishing emails which contain potential malware and which our model predicted with a confidence of more than 90%. We can make it more error-proof than it is today. We can upgrade the API authentication to use KeyVaults instead of making it public which it is as of today. We can add more options to the GUI of the Outlook Add-In that includes many customizable features like exceptions, exclusions, disable, run once a day, and other configurations.

Additionally, we will extend the features and provide users with the ability to run different types of scans on emails. For eg, users can run a full scan that will go through each email, extract each of the records(SPF, DKIM, DMARC), and provide the output, or a quick scan that will quickly go through the low-hanging fruits such as attachment type and subject line and provide with the output of whether the email is a ham or a spam.

**Address how building the full project affected your thinking from when you were only performing the work for individual stages during your earlier assignments.**

While performing the work for individual stages, we were focused on only one aspect of the product and even if performing certain actions would make it difficult in later stages, we used to proceed forward with its implementation. On working on a full-fledged project, each stage of an application made us think not only about the current stage but also how it would affect the later stages in product development.

Also, since we built the entire final project, it was essential for us to take a deep look at the usability of the product. There are several ways of implementing this but we decided on the deployment that would help the users easily get started with our product and easily consume the result. So this project helped us to think from not only technical points of view but also the user's perspective on using this application.

## **CONCLUSION**

The tool is extremely useful for someone who does not want to get distracted by petty spam emails that we receive almost every day from the internet. We can make further improvements in the machine learning model, the GUI, and the infrastructure in the cloud that supports API. While working on this project, we learned to build an end-to-end product involving machine learning and cybersecurity so as to not only develop a highly accurate model but also be highly secure.

For ART, we realized that these types of attacks can be very impactful for our AI models, thus, we should always consider training our model with the adversarial dataset to add robustness to our models. Besides, we should also consider other types of solutions such as model ensembling, regularization, etc.