CSCE 5290: Natural Language Processing

Final Project

SPAM FILTERING WITH SENTIMENT ANALYSIS

Project Description:

Spam is typically defined as undesired text that is sent or received over social media platforms like Facebook, Twitter, YouTube, e-mail, etc. It is produced by spammers in an effort to divert consumer's attention from social media marketing and malware distribution, among other purposes. Spam filtering is a mechanism used to filter spam messages or emails to prevent its delivery. The volume of unsolicited email has significantly increased, necessitating the development of more robust and efficient antispam filters. Machine learning algorithms have been used to successfully identify and filter spam emails.

Here, along with the control mechanisms and datasets utilized on spam detection, we also discuss the difficulties in identifying spam.

Github repository link: https://github.com/ShishiraRudrabhatla/spam-filtering-nlp

1. Project Title: Spam Filtering using NLP

Team Members: Group # 13

No.	Name	Student Id
1	Shishira Rudrabhatla	11560633
2	Nathisha Marru	11560642
3	Abhishek Rangineni	11435098
4	Venkata Sai Rahul Kumar Katta	11505841

2. Goals and Objectives:

Motivation:

One of the quickest and most affordable modes of communication is now email.

Spam emails, on the other hand, have dramatically increased over the past several years as a result of the rise in email subscribers. Since spammers are constantly looking for ways to get around existing filters, new filters must be created to stop spam. It is due to this major increase in spam emails that has led to the necessity of this project with an aim to train spam-filtering models better and stay up to date with the latest tricks spammers are coming up with every day and try to filter emails in the most effective ways possible.

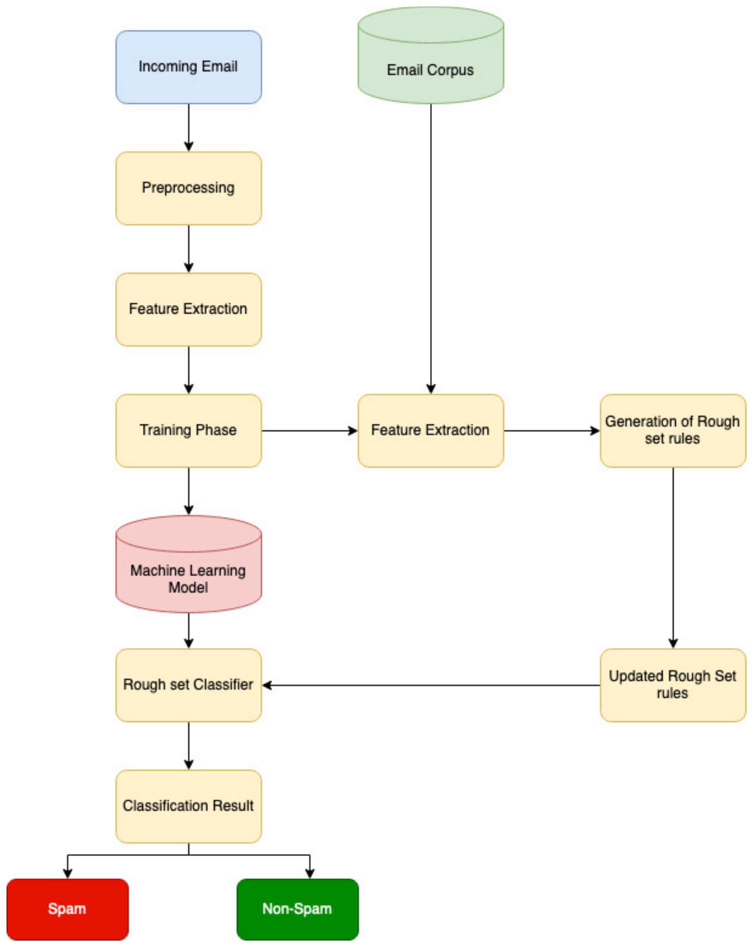
Significance

It is critical to take extra precautions to safeguard your equipment, especially if it processes sensitive information like user data. This is since clicking on a spam email might put your computer and personal information at risk of being infected with malware. Spam filtering implementation is crucial for every firm as it does not only helps keep junk out of email inboxes, but it also improves the functionality and usefulness of business communications by ensuring that they are only utilized for what they were intended for. Since many email-based attacks attempt to deceive users into clicking on a malicious file, asking them for their credentials, and other information, spam filtering becomes one of the most needed functionalities in such cases.

Objectives:

Typically, text classification is the primary methodology used for email screening. There are several alternatives available when determining how to include machine learning into a Python email categorization. The objective is to develop a Python-based spam filtering approach in which relevant spam are first filtered from the training dataset and then used to generate training and testing tables for various data mining algorithms.

Workflow diagram: Idea of workflow diagram



Features:

The proposed spam filtering system is expected to have the following features once fully implemented:

- Effective performance supported by the advantages and powers in natural language processing.
- Ability to categorize your emails as spam by training the classifier on your own dataset.
- Simplicity in usage.
- Fast performance once the training of the classifier is completed.
- Better model with higher accuracy in filtering spam e-mails.

Introduction

Unwanted email campaigns continue to be one of the top concerns, daily harming millions of consumers. Several methods to identify unwanted emails have been developed during the past few years. However, there have not been much work in the area of spam filtering together with sentiment analysis in recent studies. Everyday, scammers are getting more and more cunning, coming up with new ways of getting their spam messages undetected.

This project offers a way to support the idea that since spam is a business communication, its semantics are typically fashioned with a positive meaning. It makes use of sentiment analysis to generate the polarity scores for each message, and then assesses the accuracy of spam filtering classifiers both with and without the polarity scores.

Background

Techniques for Natural Language Processing (NLP) are gaining popularity. Employing sender information and text content-based NLP has proven to be beneficial for spam filtering techniques. Researchers confirmed that it is feasible to develop a system or an application to use text mining methods and semantic analysis to detect spam in various forms models of language, respectively. This project concentrates on the application of Sentiment Analysis among all NLP methods to further analyze the detected spam messages. This is a distinctive if we contrast this strategy with the conventional short spam detection methods, which pays attention to automated text categorization but ignores sentiment analysis.

Related Work

A study by Ezpelta et.al. (2016) proves that machine learning algorithms have outperformed all other suggested automatic categorization systems, with detection rates up to 96%. Their study demonstrates that the top 10 results of Bayesian filtering classifiers have been improved, reaching to a 99.21% of accuracy by producing the polarity score of each message using sentiment classifiers, and then comparing spam filtering classifiers with and without the polarity score in terms of accuracy.

Ghiassi et.al. (2022) talks of sentiment analysis and spam filtering as the two prominent uses of text classification in their study on Sentiment analysis and spam filtering using the YAC2 clustering algorithm with transferability. Text classification using traditional machine learning techniques is frequently difficult, non-transferable, and requires supervision. Their study proposes an unsupervised approach to text classification that is rather straightforward and generalizable across problem domains, while offering accuracy on par with or superior to that of existing approaches. It offers an integrated approach for Twitter sentiment analysis and spam filtering of YouTube comments that combines a new clustering algorithm, Yet Another Clustering Algorithm (YAC2), with a domain transferable feature engineering technique.

Whitelaw et.al (2005) in their study on Using appraisal groups for sentiment analysis report that few studies have attempted to leverage fine-grained semantic distinctions in classification features in sentiment analysis, which classifies texts by their "positive" or "negative" orientation. They discover that some forms of assessment seem to be more important than others for categorizing sentiment. However, their study is only limited to sentiment analysis without further input to text classification as whether it is spam or non-spam, serving as one of the huge limitations to the study.

Model

A sequential model was selected for the spam filtering task which helps with spam filtering by providing a way to quickly build, train, and deploy deep learning models. By building a model with the Keras Sequential Model, you can use a combination of layers to learn the features of spam emails and create a model that can accurately classify them as spam or not. This helps to reduce the amount of time and effort needed to manually sort through emails looking for spam. Additionally, the Keras Sequential Model can be used to fine-tune the model over time, allowing for improved accuracy and more reliable classification. Using the graphviz library, the model can be visualized as follows:

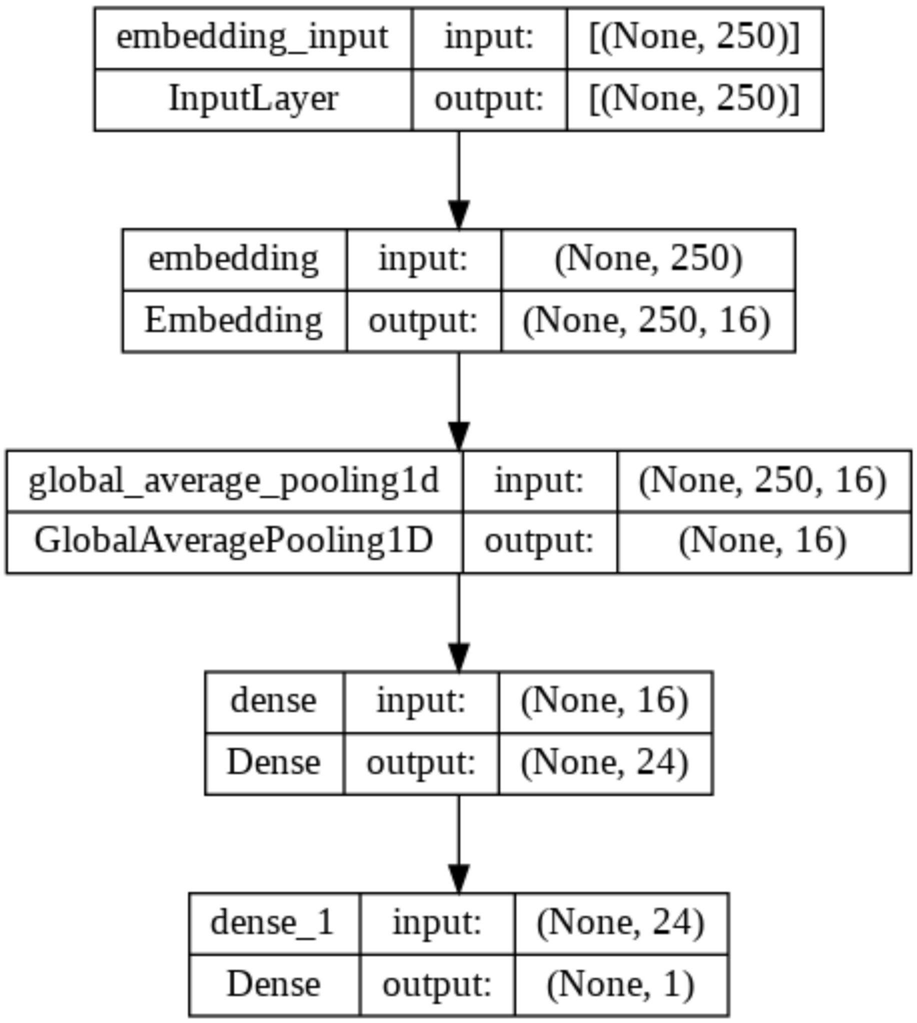
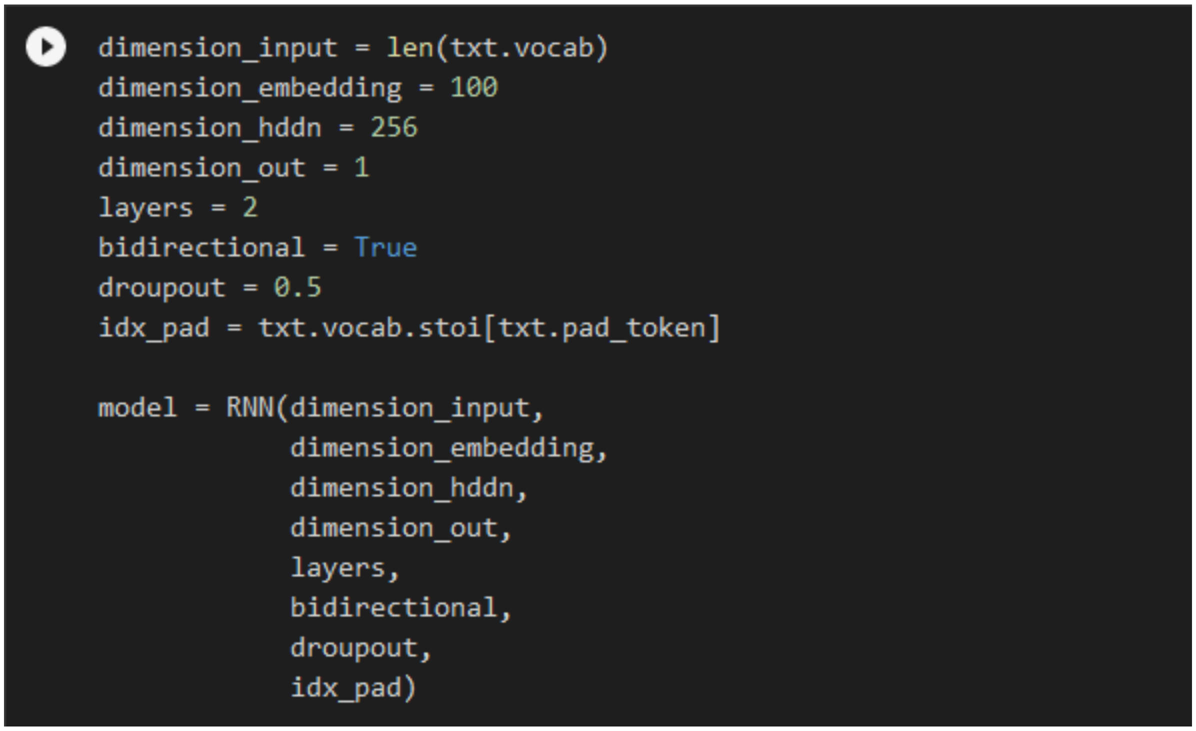


Figure 1: Graphviz visualization for the sequential model

For the sentiment analysis, RNN model was selected which was initialized as follows:



```
dimension_input = len(txt.vocab)
dimension_embedding = 100
dimension_hddn = 256
dimension_out = 1
layers = 2
bidirectional = True
droupout = 0.5
idx_pad = txt.vocab.stoi[txt.pad_token]

model = RNN(dimension_input,
            dimension_embedding,
            dimension_hddn,
            dimension_out,
            layers,
            bidirectional,
            droupout,
            idx_pad)
```

Figure 2: RNN Model initialization

Dataset

Description

The International Movie Database (IMDb) reviews for 50,000 reviews of films from across the world make up the dataset used for sentiment analysis; it's a binary classification dataset that classifies each review as either positive or negative. Being split at 50%:50%, it has 25,000 training samples and 25,000 test samples.

Features Design

Given the aim of the project to analyze the spam and non-spam messages, the selected features for the operation were:

- label
- message

The following plot is a visualization to show the count of each of these two features in the dataset.

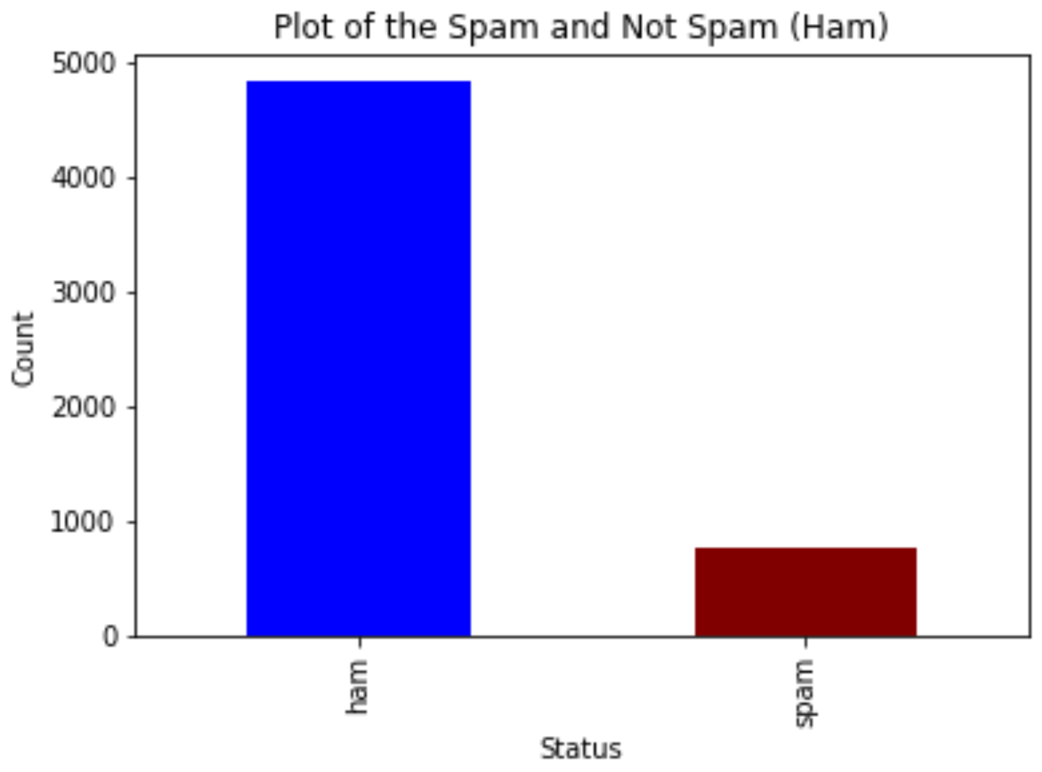


Figure 3a: Plot of spam and non-spam dataset

Analysis of Data

Data Pre-processing

Starting with the original dataset:

(1) We remove un-important features from the dataset to remain with only the spam and ham (non-spam) columns for filtering analysis;

(2) We then add the discovered classification to the original dataset after using sentiment analysis classifiers;

(3) We integrate the results from the two techniques on the original messages to produce the final dataset;

(4) Finally, the dataset's outcomes are then examined.

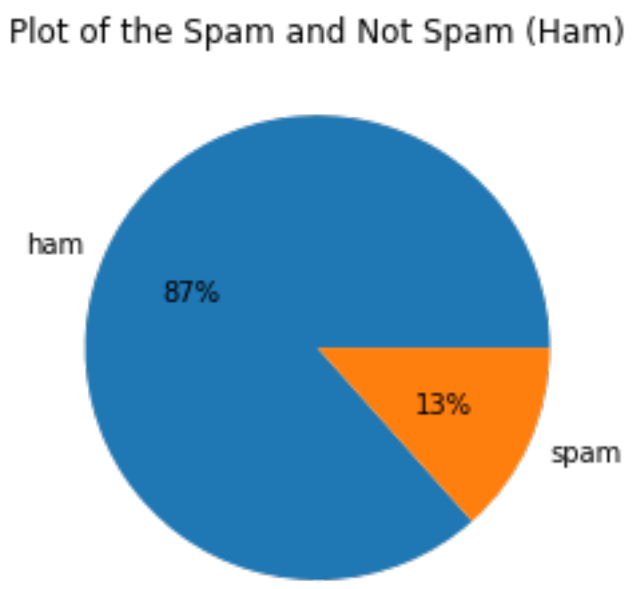


Figure 3b: Plot of spam and non-spam dataset

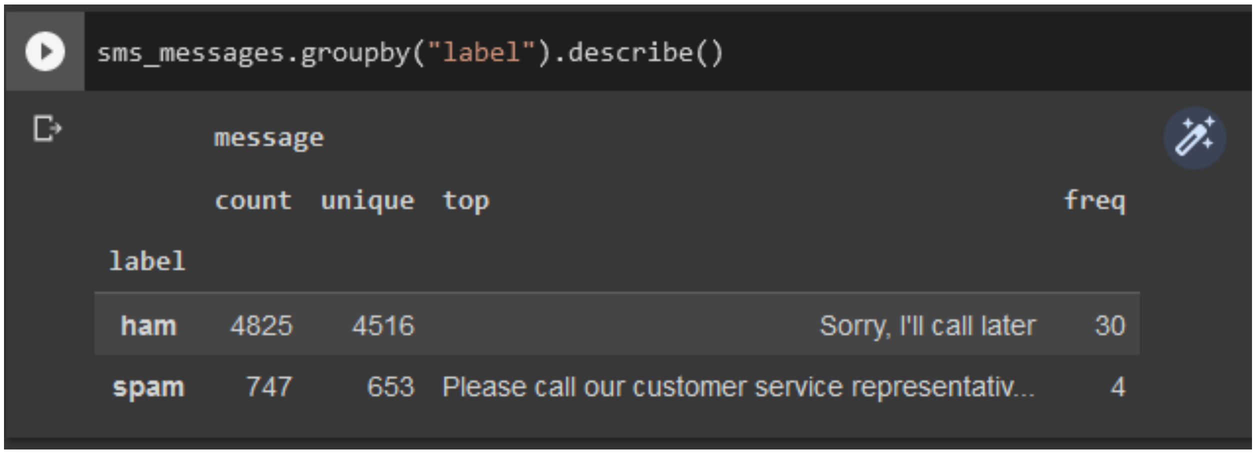
From the plot, it can be seen that majority of the dataset contains clean messages (ham). 87% of the dataset are the non-spam messages while the remaining 13% are the spam messages. Using this dataset, we can train our model to be able to detect future spam messages.

Implementation

Spam Filtering

Having retrieved the dataset for the initial part of the project, spam filtering, exploratory data analysis (EDA) is conducted on the dataset. A visualization of the available spam and ham messages in the dataset is generated for further observations and analysis as can be seen in the figure below:

The messages were then grouped by the label as shown in the following screenshot of the source code and its respective output:



```
sms_messages.groupby("label").describe()
```

	message			
	count	unique	top	freq
label				
ham	4825	4516	Sorry, I'll call later	30
spam	747	653	Please call our customer service representativ...	4

Figure 4: Screenshot of the description of the grouped messages dataset

For further analysis on the spam dataset, it was important to compare the length of the messages alongside their ham/spam classification. The analysis conducted resulted in the following plot:

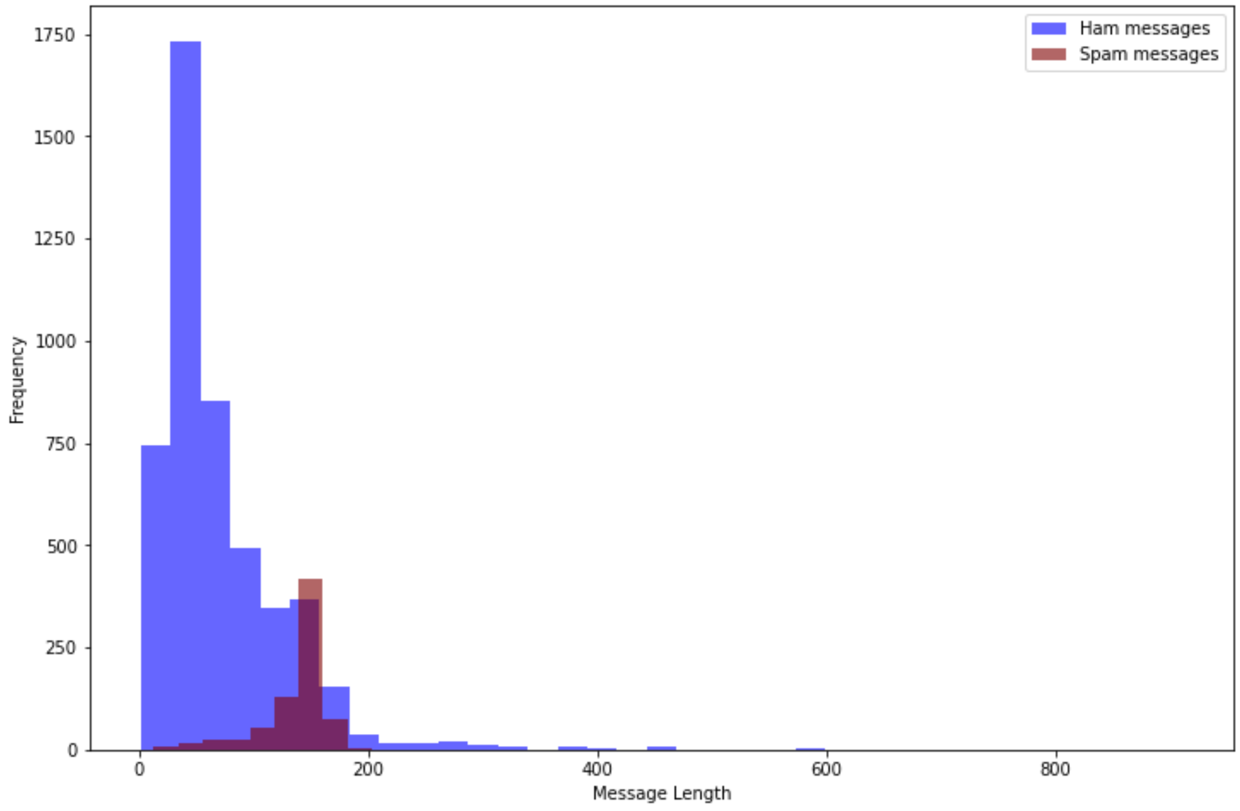


Figure 5: Plot of message lengths for the spam and ham messages

Majority of the messages had a length below 200 characters as can be observed in the above screenshot. Having observed the analysis in comparison to the length of the messages, it was necessary to also compare the frequency of the common words in the dataset.

The following is the plot of the most common words in the dataset for the non-spam messages:

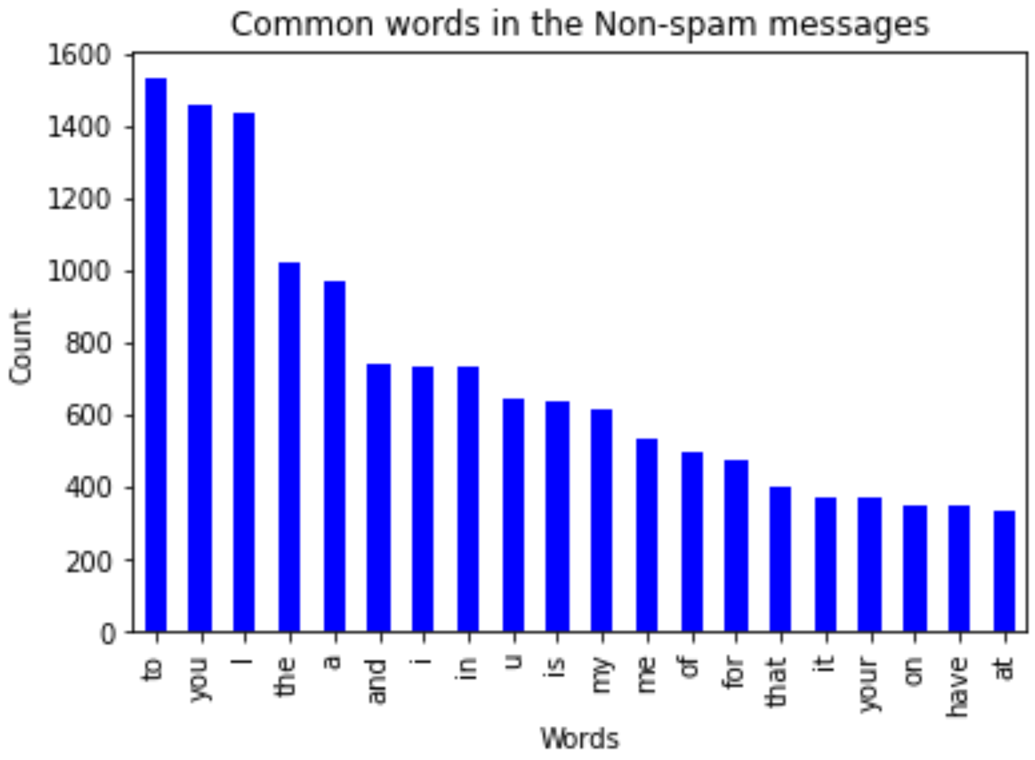


Figure 6: Plot of most common words in the non-spam messages

On the other hand, a plot of the common spam words in the dataset is shown below:

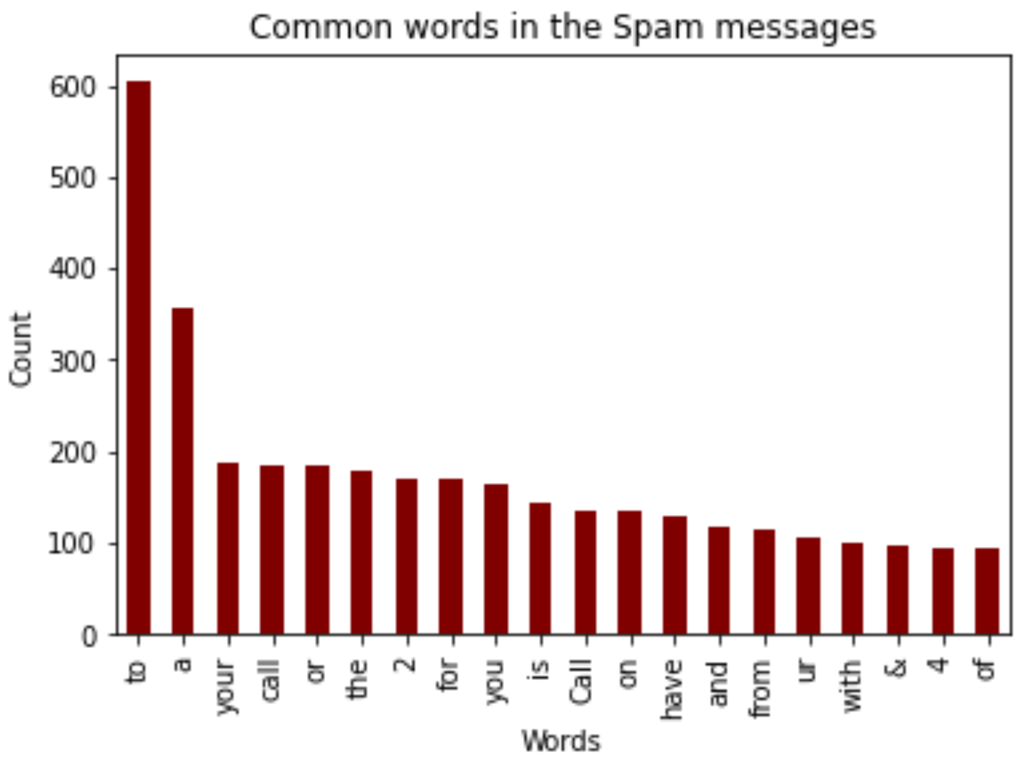


Figure 7: Plot of most common words in the spam messages

Using the available test data, the model was tested and its accuracy calculated. The model's accuracy was high enough to proceed with the analysis as it was ~98%. It was then saved to be used in the next process for sentiment analysis and other predictions.

Sentiment Analysis

For this next part of the project, pytorch library is utilized. Fortunately, it already has certain features installed that will help us work more quickly. For NLP projects, the torch.text library is a fantastic resource as it includes a loader for various popular NLP datasets, including the one we will used for the project, as well as a whole pipeline for data abstraction, data vectorization, data loaders, and data iteration.

To determine how data has to be preprocessed, we will utilize the field method of the data class. The preprocessing will be determined by the arguments we put into that function; we will only change a few parameters and leave the rest at their default values.

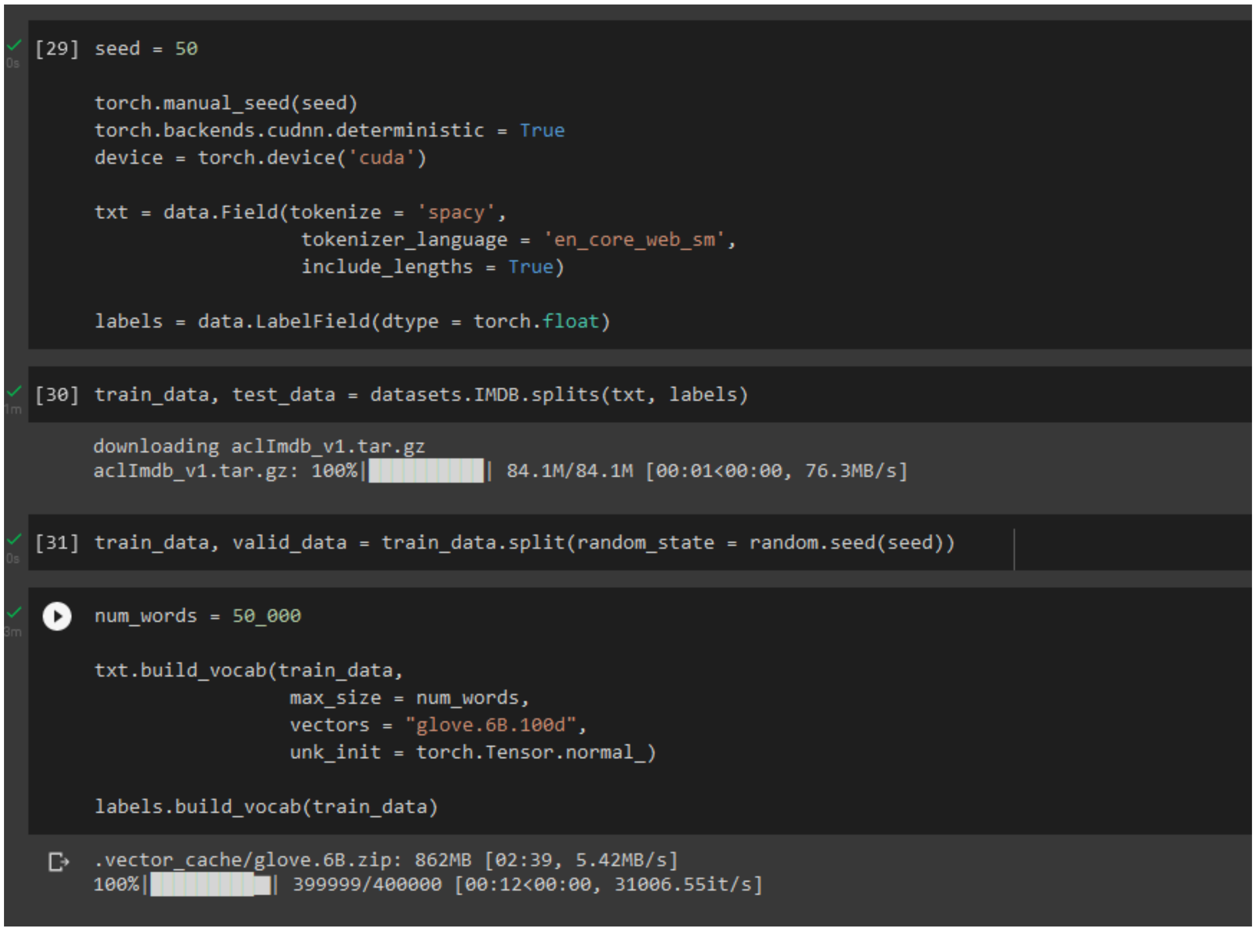
The first parameter, our tokenizer, is spacy, a potent tool for one-line tokenization, which controls how the sentences will be split down or tokenized in NLP standard. Although the default just tokenizes the string based on blank spaces, we advise using this. Additionally, we must instruct the spacy tokenizer on which language model to employ.

To do Python sentiment analysis, the IMDB dataset is retrieved and divided into train, test, and validation splits. Validation set is further generated from the dataset, which has previously been split into a train and test set. Afterwards, the number of words the model learns is limited to 25000; this will take the 25000 most frequently used terms from the dataset as training data as a result decreasing the model's workload considerably while maintaining accuracy.

Finally, we organize the data we have into a training, testing, and validation batch in order to get it ready to be given to the model in batches of 64 samples at a time.

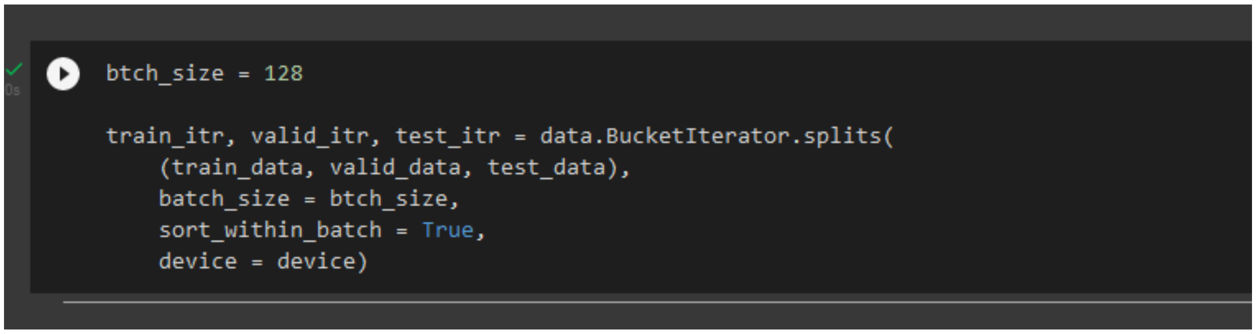
If a memory problem occurs, its value can be changed.

The following is the source code screenshots responsible for this:



```
[29] seed = 50

     torch.manual_seed(seed)
     torch.backends.cudnn.deterministic = True
     device = torch.device('cuda')

     txt = data.Field(tokenize = 'spacy',
                      tokenizer_language = 'en_core_web_sm',
                      include_lengths = True)

     labels = data.LabelField(dtype = torch.float)

[30] train_data, test_data = datasets.IMDB.splits(txt, labels)

     downloading aclImdb_v1.tar.gz
     aclImdb_v1.tar.gz: 100%|██████████| 84.1M/84.1M [00:01<00:00, 76.3MB/s]

[31] train_data, valid_data = train_data.split(random_state = random.seed(seed))

     num_words = 50_000

     txt.build_vocab(train_data,
                     max_size = num_words,
                     vectors = "glove.6B.100d",
                     unk_init = torch.Tensor.normal_)

     labels.build_vocab(train_data)

     .vector_cache/glove.6B.zip: 862MB [02:39, 5.42MB/s]
     100%|█████████▉| 399999/400000 [00:12<00:00, 31006.55it/s]
```

Figure 8a: Screenshot for data preparation and train/test splitting



```
btch_size = 128

train_itr, valid_itr, test_itr = data.BucketIterator.splits(
    (train_data, valid_data, test_data),
    batch_size = btch_size,
    sort_within_batch = True,
    device = device)
```

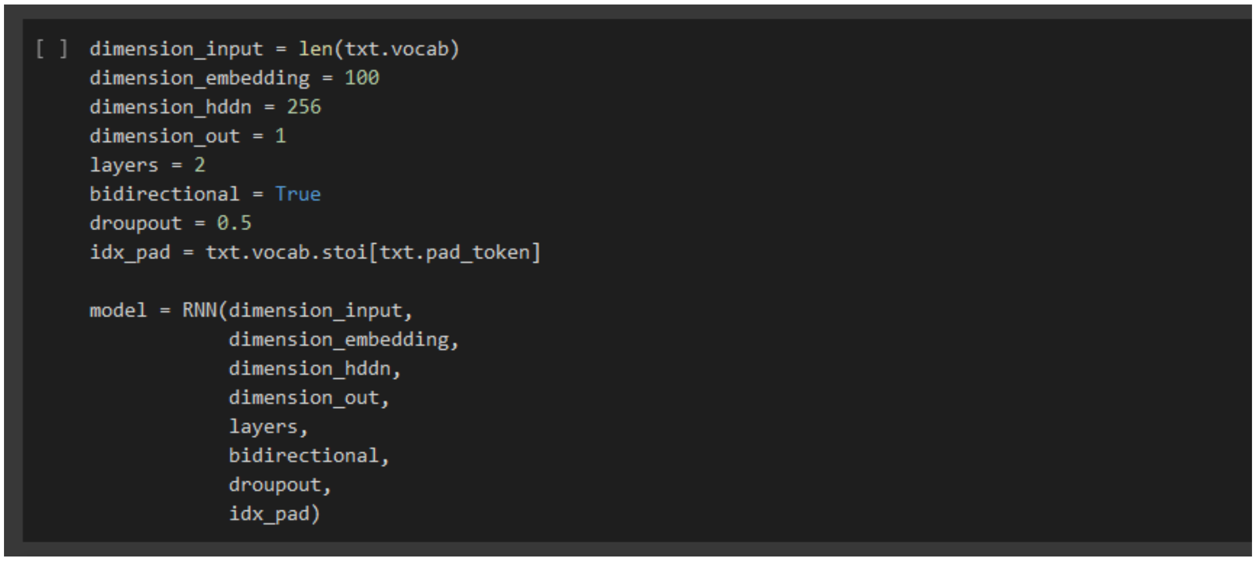
Figure 8b: Screenshot for data preparation and train/test splitting (cont.)

Now that the model is ready, its architecture is established. The next stage involves employing a LSTM RNN.

A multi-layer bidirectional RNN will be used. That implies that several RNN layers will be layered on top of one another. The motivation behind having more context than a single directional network is found in bidirectional RNNs. For instance, if a model needs to infer the next word in a phrase that is flowing through it, it would do so based on prior information. However, when two networks run in opposing directions and are piled on top of one another in a bidirectional network, it will also know what comes next. Both the sequence and the direction of the input are the reverse.

The parameters for the Python sentiment analysis model are then defined, and they are sent to an instance of the model class we just created. The throughput rate, hidden layer, output dimension, number of input parameters, and bidirectionality boolean are all defined. The vocabulary's pad token index that we produced before is also provided.

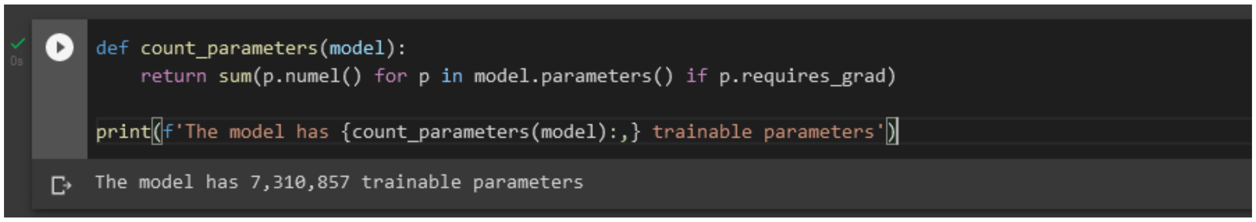
Below is a screenshot of the source code illustrating this:



```
[ ]  dimension_input = len(txt.vocab)
     dimension_embedding = 100
     dimension_hddn = 256
     dimension_out = 1
     layers = 2
     bidirectional = True
     droupout = 0.5
     idx_pad = txt.vocab.stoi[txt.pad_token]

     model = RNN(dimension_input,
                 dimension_embedding,
                 dimension_hddn,
                 dimension_out,
                 layers,
                 bidirectional,
                 droupout,
                 idx_pad)
```

Figure 9: Screenshot for the parameters for sentiment analysis

We then print certain model-related information determining how many trainable parameters are included in the model as shown below:

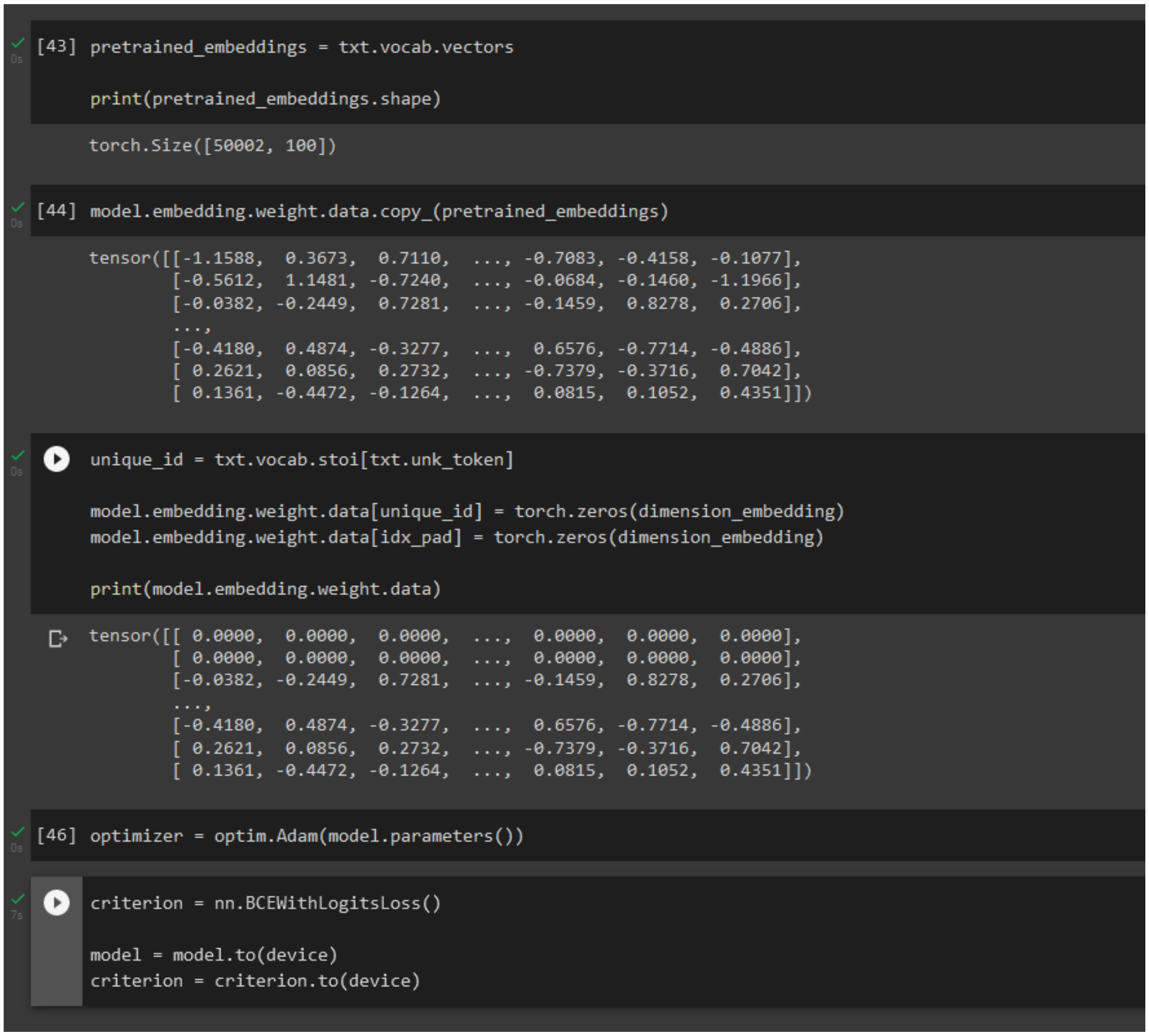


```
def count_parameters(model):
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

print(f'The model has {count_parameters(model):,} trainable parameters')
```

```
The model has 7,310,857 trainable parameters
```

Figure 10: Screenshot for number of trainable parameters

The pre-trained embedding weights are then obtained and copied to our model, allowing it to concentrate just on the task at hand—learning the feelings associated with those embeddings—instead of having to learn the embeddings. The original embedding weights are swapped out for pre-trained ones.



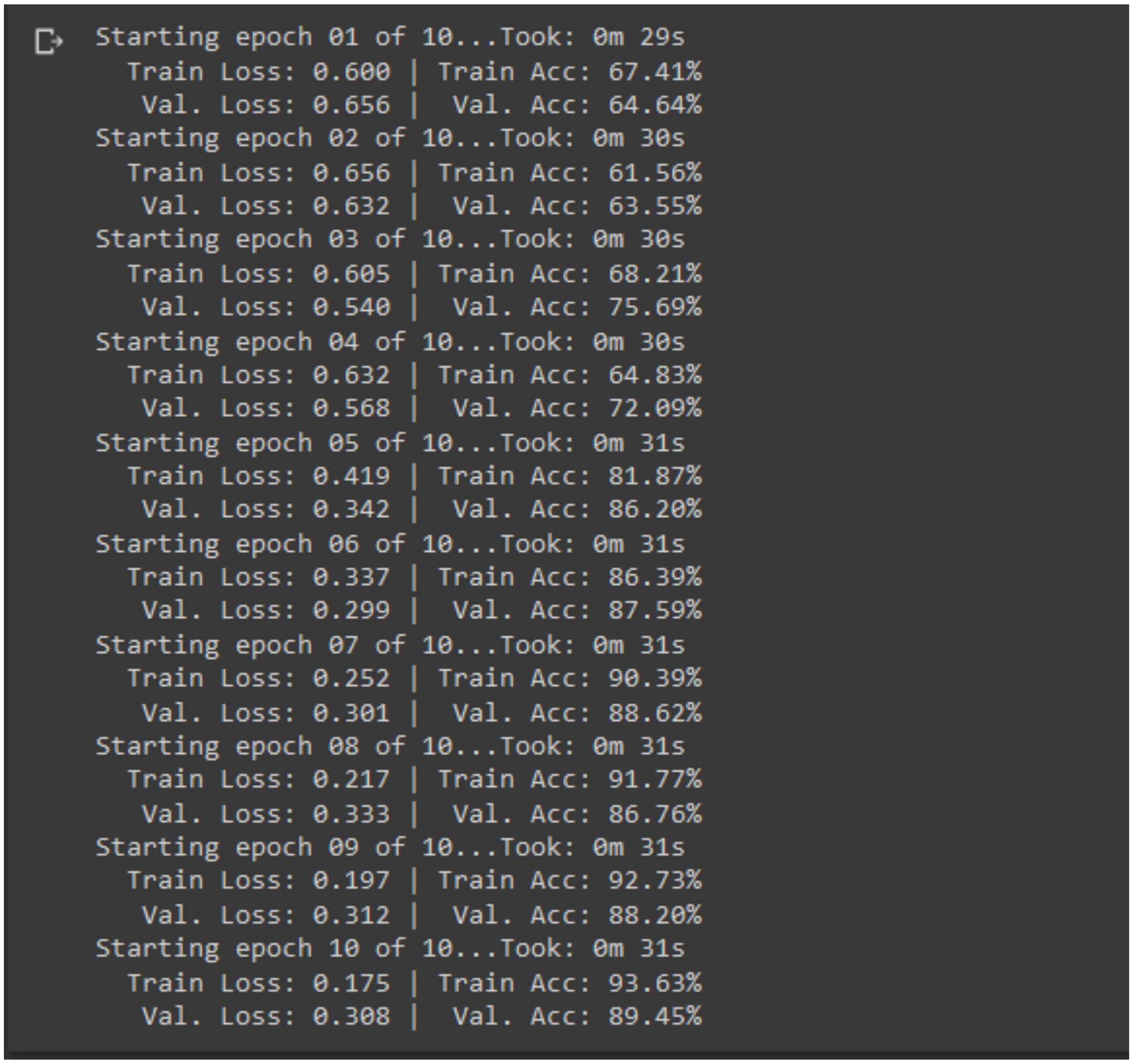
```
[43] pretrained_embeddings = txt.vocab.vectors

     print(pretrained_embeddings.shape)

     torch.Size([50002, 100])

[44] model.embedding.weight.data.copy_(pretrained_embeddings)

     tensor([[-1.1588,  0.3673,  0.7110,  ..., -0.7083, -0.4158, -0.1077],
             [-0.5612,  1.1481, -0.7240,  ..., -0.0684, -0.1460, -1.1966],
             [-0.0382, -0.2449,  0.7281,  ..., -0.1459,  0.8278,  0.2706],
             ...,
             [-0.4180,  0.4874, -0.3277,  ...,  0.6576, -0.7714, -0.4886],
             [ 0.2621,  0.0856,  0.2732,  ..., -0.7379, -0.3716,  0.7042],
             [ 0.1361, -0.4472, -0.1264,  ...,  0.0815,  0.1052,  0.4351]])

     unique_id = txt.vocab.stoi[txt.unk_token]

     model.embedding.weight.data[unique_id] = torch.zeros(dimension_embedding)
     model.embedding.weight.data[idx_pad] = torch.zeros(dimension_embedding)

     print(model.embedding.weight.data)

     tensor([[ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
             [ 0.0000,  0.0000,  0.0000,  ...,  0.0000,  0.0000,  0.0000],
             [-0.0382, -0.2449,  0.7281,  ..., -0.1459,  0.8278,  0.2706],
             ...,
             [-0.4180,  0.4874, -0.3277,  ...,  0.6576, -0.7714, -0.4886],
             [ 0.2621,  0.0856,  0.2732,  ..., -0.7379, -0.3716,  0.7042],
             [ 0.1361, -0.4472, -0.1264,  ...,  0.0815,  0.1052,  0.4351]])

[46] optimizer = optim.Adam(model.parameters())

     criterion = nn.BCEWithLogitsLoss()

     model = model.to(device)
     criterion = criterion.to(device)
```

Figure 11: Screenshot for model operations such as embedding

The next tasks are now for training and assessing the sentiment analysis model. The binary accuracy function, which we'll use to determine the model's correctness each time, is the first one. We specify the model training and model evaluation function. Here, the procedure is typical. The number of iterations in each epoch is determined by the batch size that we set, and we begin by looping over all of them. The model is given the text, and we obtain the predictions from it. We then calculate the loss for each iteration and propagate that loss backward. The only significant difference between the evaluating function and the training function is that we employ torch instead of backward propagating the loss through the model. No gradient descent is basically what torch.nograd means when assessing.

The training started after setting the number of epochs to 10. In order to comprehend or plot the training curve at a later level, we add the training and validation loss at each stage. The sentiment analysis model in Python with the lowest validation loss is saved. The model's saved checkpoint is then loaded and put it to the test on the earlier-made test set.

We acquired a respectable accuracy score of 89.45% during the dry run of the Python sentiment analysis model as can be seen in the next screenshot:



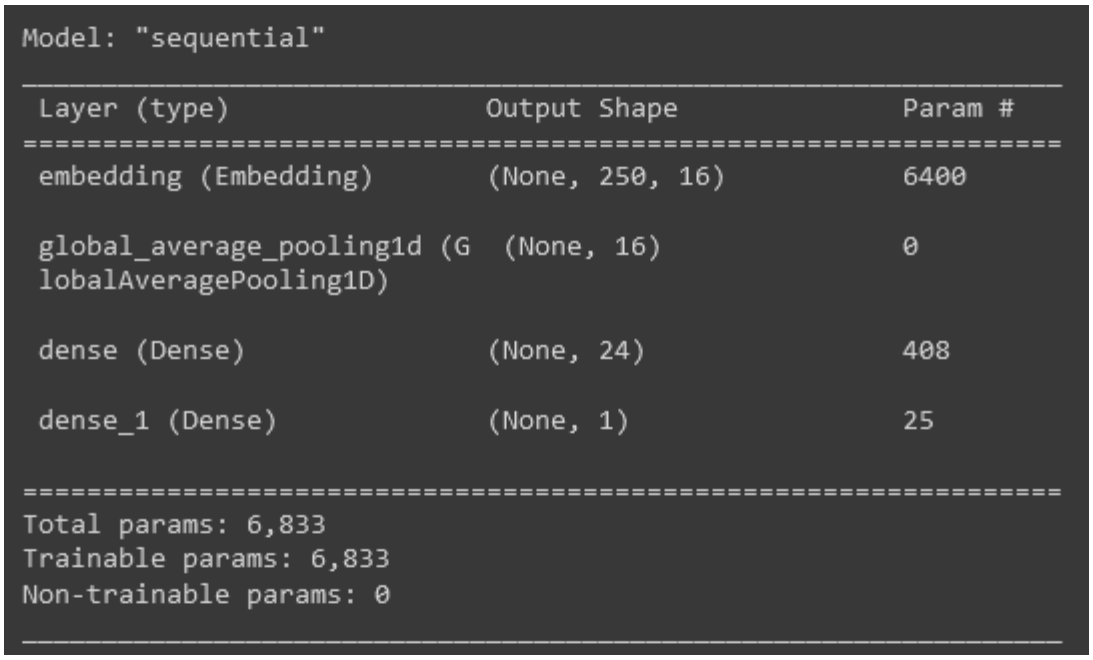
```
Starting epoch 01 of 10...Took: 0m 29s
  Train Loss: 0.600 | Train Acc: 67.41%
   Val. Loss: 0.656 |  Val. Acc: 64.64%
Starting epoch 02 of 10...Took: 0m 30s
  Train Loss: 0.656 | Train Acc: 61.56%
   Val. Loss: 0.632 |  Val. Acc: 63.55%
Starting epoch 03 of 10...Took: 0m 30s
  Train Loss: 0.605 | Train Acc: 68.21%
   Val. Loss: 0.540 |  Val. Acc: 75.69%
Starting epoch 04 of 10...Took: 0m 30s
  Train Loss: 0.632 | Train Acc: 64.83%
   Val. Loss: 0.568 |  Val. Acc: 72.09%
Starting epoch 05 of 10...Took: 0m 31s
  Train Loss: 0.419 | Train Acc: 81.87%
   Val. Loss: 0.342 |  Val. Acc: 86.20%
Starting epoch 06 of 10...Took: 0m 31s
  Train Loss: 0.337 | Train Acc: 86.39%
   Val. Loss: 0.299 |  Val. Acc: 87.59%
Starting epoch 07 of 10...Took: 0m 31s
  Train Loss: 0.252 | Train Acc: 90.39%
   Val. Loss: 0.301 |  Val. Acc: 88.62%
Starting epoch 08 of 10...Took: 0m 31s
  Train Loss: 0.217 | Train Acc: 91.77%
   Val. Loss: 0.333 |  Val. Acc: 86.76%
Starting epoch 09 of 10...Took: 0m 31s
  Train Loss: 0.197 | Train Acc: 92.73%
   Val. Loss: 0.312 |  Val. Acc: 88.20%
Starting epoch 10 of 10...Took: 0m 31s
  Train Loss: 0.175 | Train Acc: 93.63%
   Val. Loss: 0.308 |  Val. Acc: 89.45%
```

Figure 12: Screenshot for epoch operations

Results

Preliminary results for spam filtering

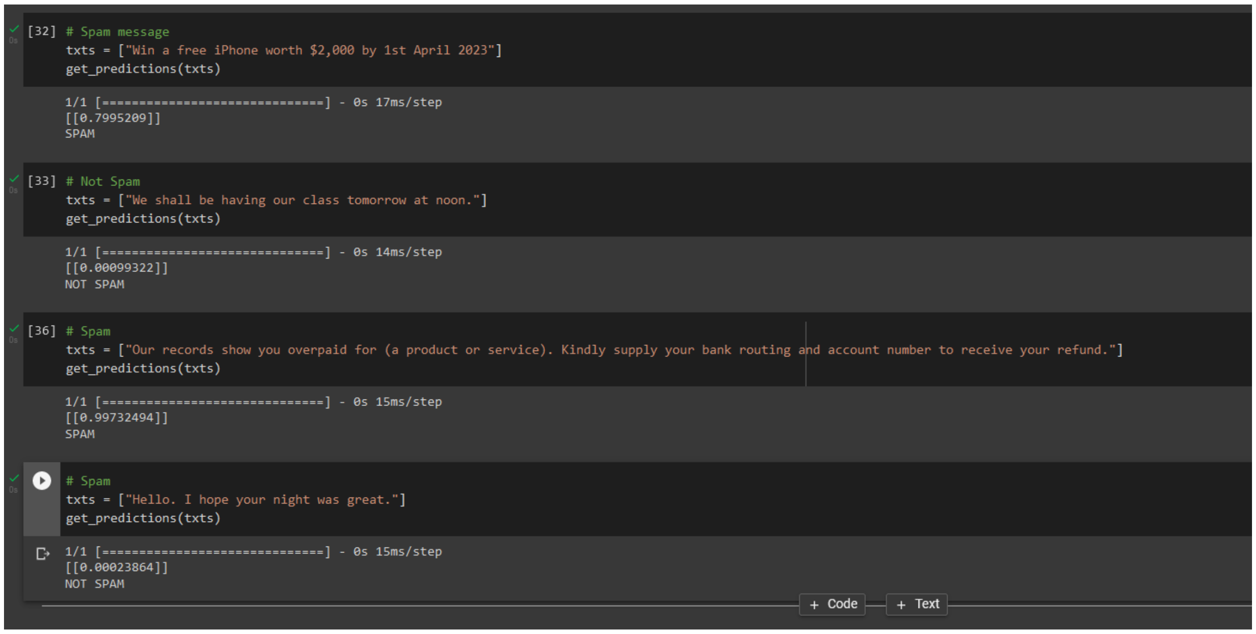
The following is the summary of the model used for spam filtering:



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 250, 16)           6400

 global_average_pooling1d (G  (None, 16)               0
 lobalAveragePooling1D)

 dense (Dense)               (None, 24)                408

 dense_1 (Dense)             (None, 1)                 25

=================================================================
Total params: 6,833
Trainable params: 6,833
Non-trainable params: 0
_________________________________________________________________
```

Figure 13: Screenshot for the sequential model summary for spam filtering

The model was used to test sample predictions as can be seen in the following screenshots which tested various message with the potential of being either spam or non-spam:



```
[32] # Spam message
     txts = ["Win a free iPhone worth $2,000 by 1st April 2023"]
     get_predictions(txts)

     1/1 [==============================] - 0s 17ms/step
     [[0.7995209]]
     SPAM

[33] # Not Spam
     txts = ["We shall be having our class tomorrow at noon."]
     get_predictions(txts)

     1/1 [==============================] - 0s 14ms/step
     [[0.00099322]]
     NOT SPAM

[36] # Spam
     txts = ["Our records show you overpaid for (a product or service). Kindly supply your bank routing and account number to receive your refund."]
     get_predictions(txts)

     1/1 [==============================] - 0s 15ms/step
     [[0.99732494]]
     SPAM

     # Spam
     txts = ["Hello. I hope your night was great."]
     get_predictions(txts)

     1/1 [==============================] - 0s 15ms/step
     [[0.00023864]]
     NOT SPAM
```

Figure 14: Screenshot showing testing of spam/ham messages

The analysis classifies new incoming messages as follows:

Message	Spam/Not Spam
Win a free iPhone worth $2,000 by 1st April 2023	SPAM
We shall be having our class tomorrow at noon.	NOT SPAM
Our records show you overpaid for (a product or service). Kindly supply your bank routing and account number to receive your refund.	SPAM
Hello. I hope your night was great.	HAM

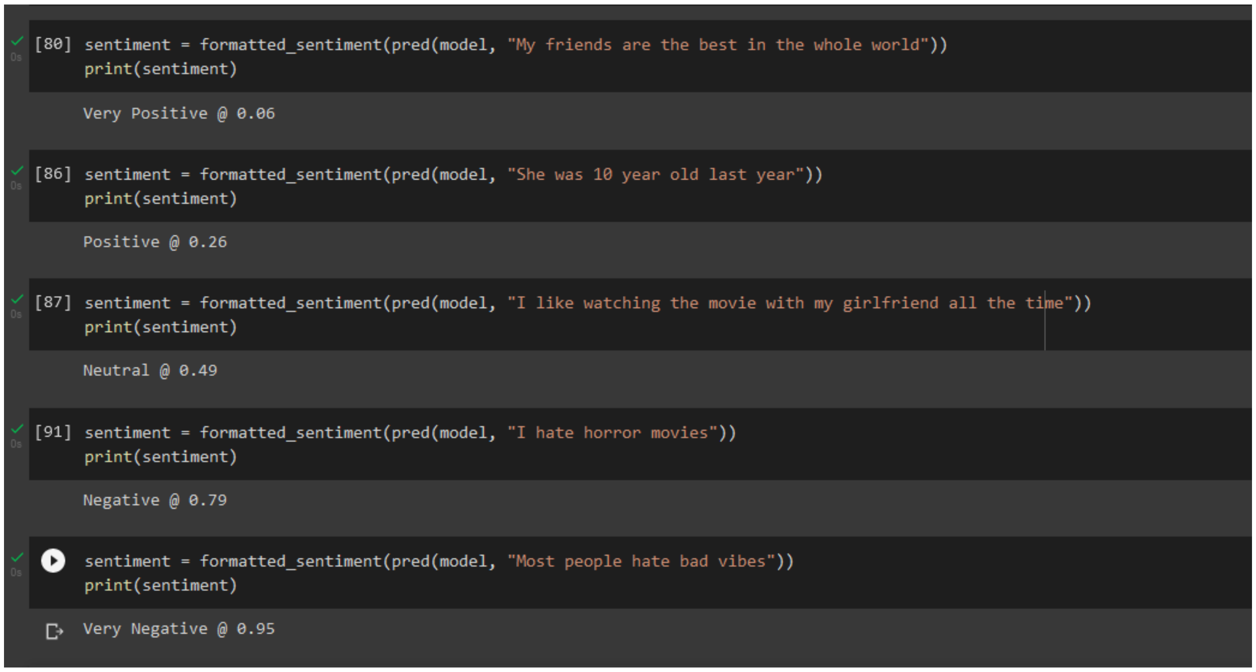
Table 1: Table of Spam/Ham for sample test cases

Preliminary results for sentiment analysis

On our data, we can additionally test the model. We will send it pertinent information for verification as it has been trained to classify movie reviews into very positive, positive, neutral, negative and very negative categories. In order to tokenize the data, we must provide the model, we will import and load Spacy.

Initially, when defining the preprocessing, we utilized the spacy library's built-in torch.text; however, in this case, we are not utilizing batches, and the necessary preprocessing may be handled by the spacy library. For this, we develop a predict sentiment function. Following preprocessing, we turn data into tensors so that it can be sent to the model.

The following are sample results for test texts for sentiment analysis:



```
[80] sentiment = formatted_sentiment(pred(model, "My friends are the best in the whole world"))
     print(sentiment)

     Very Positive @ 0.06

[86] sentiment = formatted_sentiment(pred(model, "She was 10 year old last year"))
     print(sentiment)

     Positive @ 0.26

[87] sentiment = formatted_sentiment(pred(model, "I like watching the movie with my girlfriend all the time"))
     print(sentiment)

     Neutral @ 0.49

[91] sentiment = formatted_sentiment(pred(model, "I hate horror movies"))
     print(sentiment)

     Negative @ 0.79

     sentiment = formatted_sentiment(pred(model, "Most people hate bad vibes"))
     print(sentiment)

     Very Negative @ 0.95
```

Figure 15: Screenshot for sample sentiment analysis test cases

As it can be seen, classification is either of the following:

- Very positive
- Positive
- Neutral
- Negative
- Very negative

Blending spam filtering with sentiment analysis

Now, with both the spam filtering model and sentiment analysis model implemented, both are then combined to analyze the relationship between the spam messages, non-spam messages and their respective sentiments in each category.

For each message in the spam and ham dataset, an extra column specifying its sentiment was added. Below is a plot to visualize this:

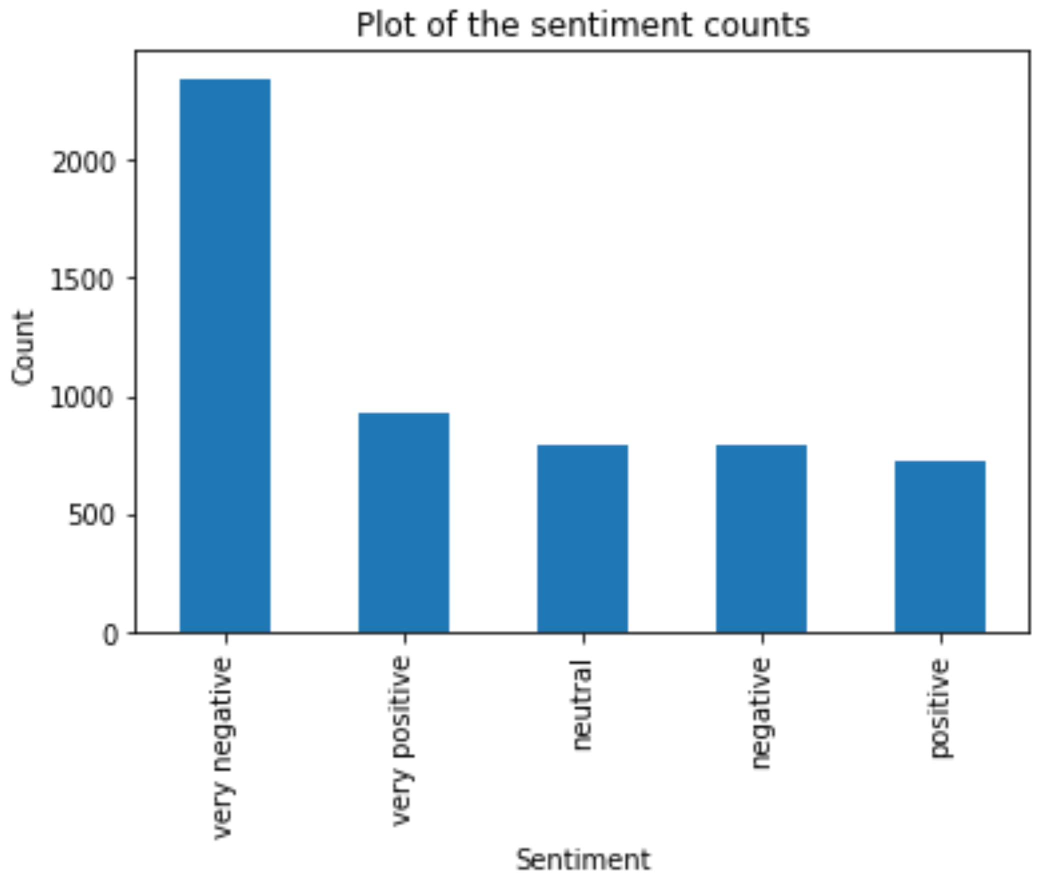


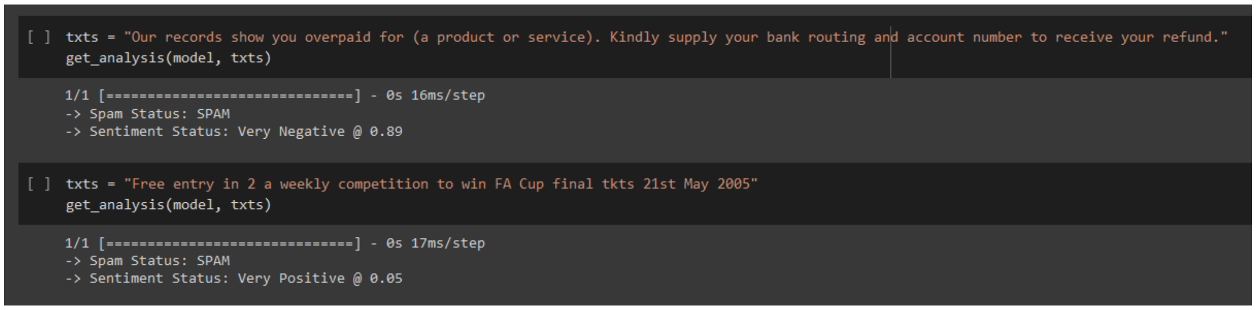
Figure 16: Plot for the sentiment counts for whole dataset

In a nutshell, it can be said that a majority of the messages available contain the negative sentiment as opposed to that implying positivity in them.

Results with improved features

Having blended the two models together, it is now possible to not only detect whether a message is spam or not, but we are now capable of grouping it also according to its sentiment value. The reverse is also true as other than identifying the sentiment status of a given statement or message, the results can also be used in the analysis for its spam/ham detection.

Below are sample results obtained after blending the two models together:



```
[ ] txts = "Our records show you overpaid for (a product or service). Kindly supply your bank routing and account number to receive your refund."
    get_analysis(model, txts)

    1/1 [==============================] - 0s 16ms/step
    -> Spam Status: SPAM
    -> Sentiment Status: Very Negative @ 0.89

[ ] txts = "Free entry in 2 a weekly competition to win FA Cup final tkts 21st May 2005"
    get_analysis(model, txts)

    1/1 [==============================] - 0s 17ms/step
    -> Spam Status: SPAM
    -> Sentiment Status: Very Positive @ 0.05
```

Figure 17: Screenshot for blended spam filtering and sentiment analysis screenshots

An interesting observation was made when analysis for the dataset was made for its overall sentiment based on whether the message was spam or not.

Below is the result of plotting visualizations for determining the sentiment for ham (non-spam) dataset:

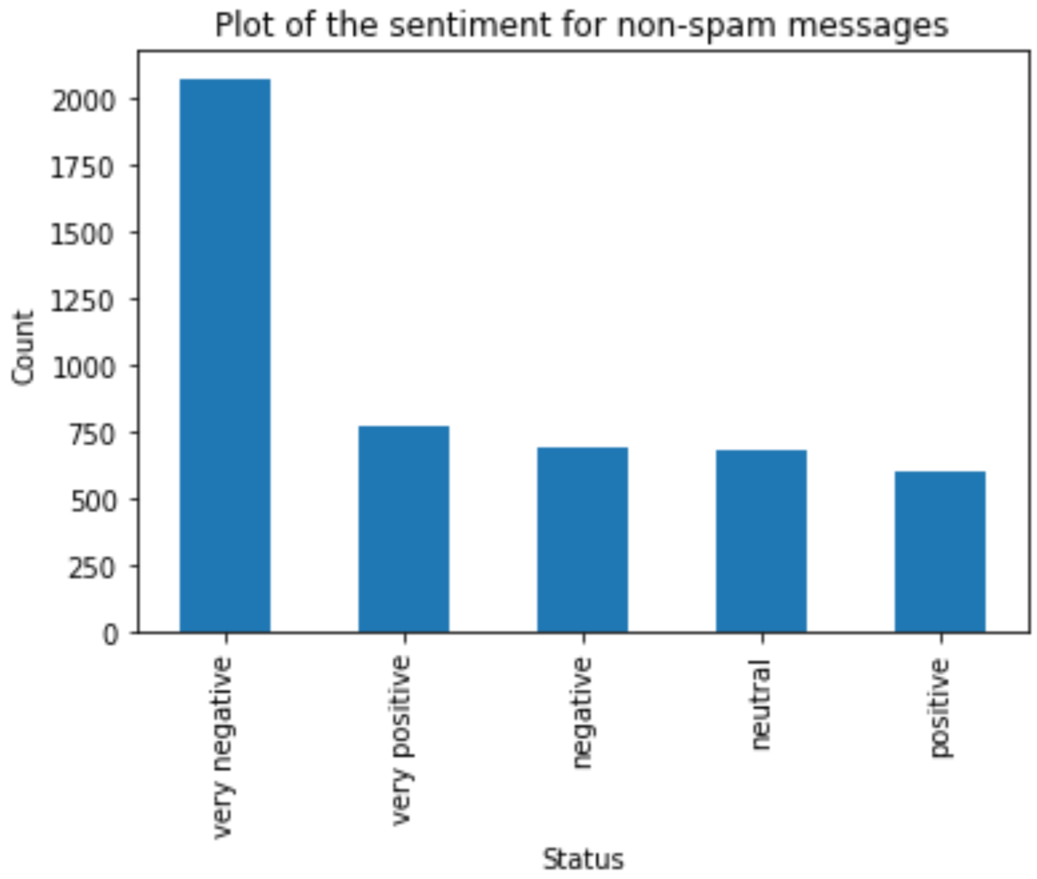


Figure 18: Plot for the sentiment counts for non-spam dataset

The following is its respective pie-chart:

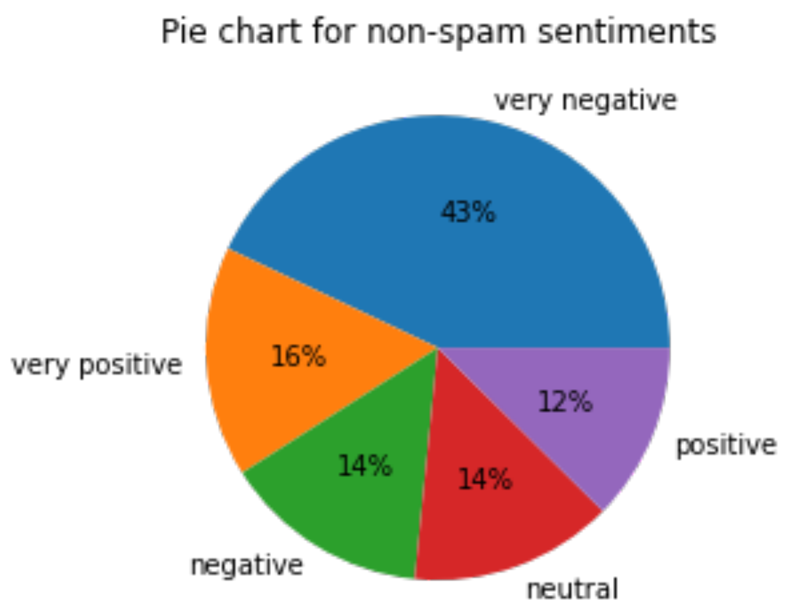


Figure 19: Pie chart for the sentiment counts for non-spam dataset

It can be observed that the most popular sentiments in this dataset was as follows (sorted from highest to least):

1. Very negative = 43%
2. Very positive = 16%
3. Negative = 14%
4. Neutral = 14%
5. Positive = 12%

Below is the result of plotting visualizations for determining the sentiment for spam dataset:

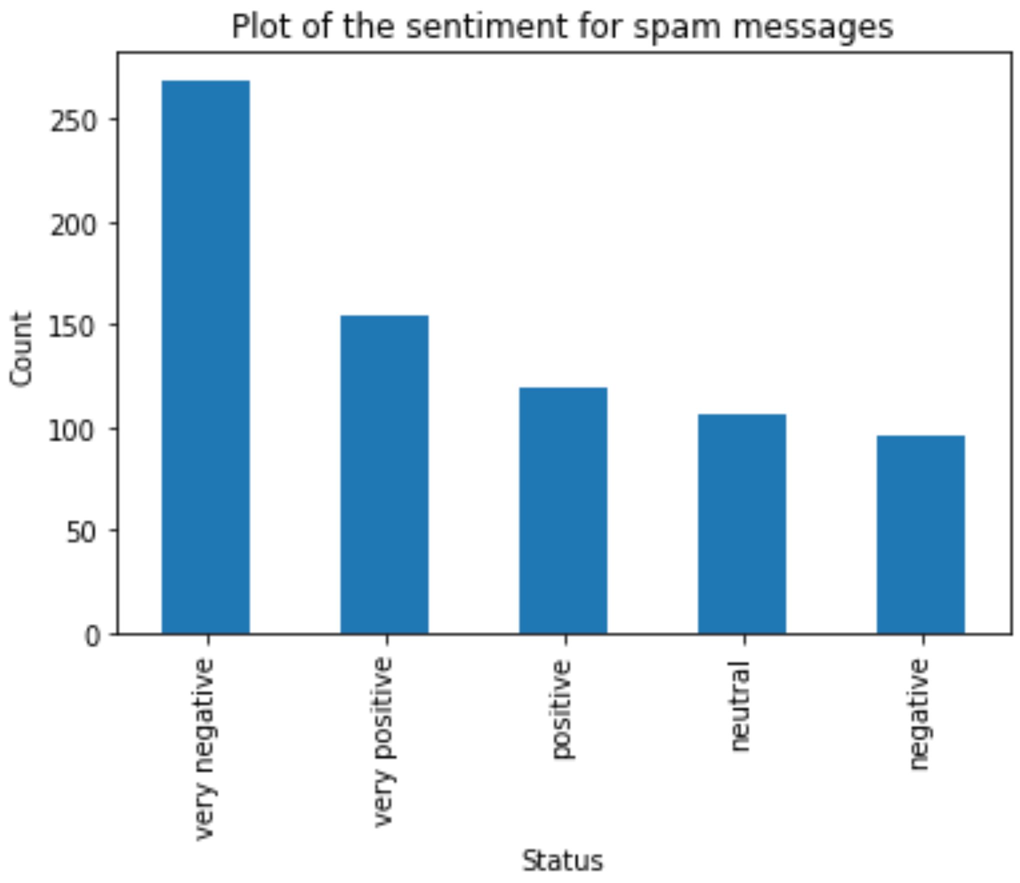


Figure 20: Plot for the sentiment counts for spam dataset

It’s respective pie-chart plot is as follows:

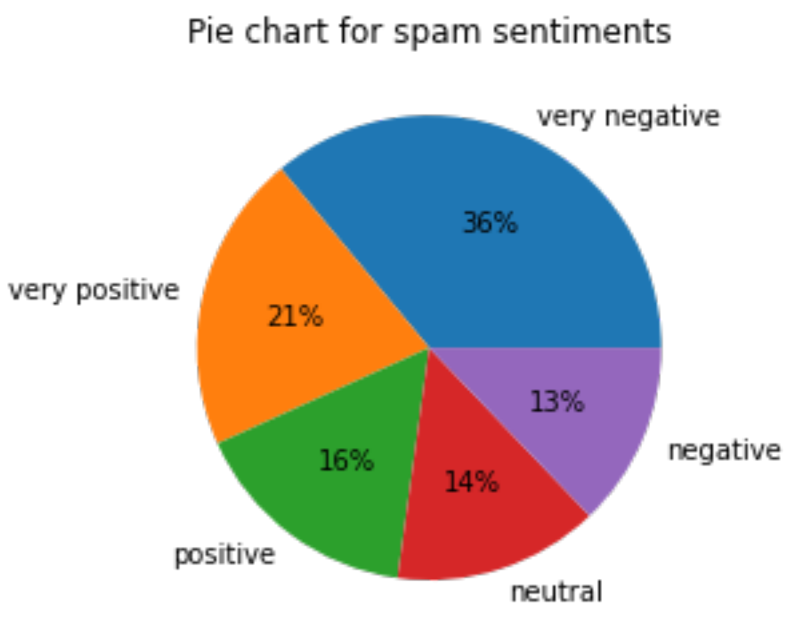


Figure 21: Pie chart for the sentiment counts for spam dataset

The popular sentiments generated from the spam dataset are as follows:

1. Very negative = 36%
2. Very positive = 16%
3. Positive = 16%
4. Neutral = 14%
5. Negative = 13%

As evident from the above visualizations and analysis, it can be observed that the spam dataset contains higher percentages of positive sentiments as compared to non-spam messages. This is likely due to the fact that most of the spam messages are meant to lure and as a result contain enticing words meant to attract the intended audience.

Using the results from this analysis, future models can integrate sentiment analysis with up-to-date datasets to refine their spam-detection algorithms and improve on their efficiency. The project identifies a relationship between the sentiment of the message and its classification as either spam or not spam. It can be applied in various sectors such as email spam detection algorithms and accuracy calculations, product reviews, SMS spam and sentiment filtering among other sectors.

Project Management

Implementation Status Report

- Work Completed

  Description: Dataset Collection
  Responsibility: Nathisha Marru, Shishira Rudrabhatla
  Contribution: Both contributed 50 percent each to the task of dataset collection

  Description: Building Model
  Responsibility: Venkata Sai Rahul Kumar, Nathisha Marru
  Contribution: Model of Spam filtering is built and tested and found accuracy and all teammates contributed to this task as this is major task in project.

  Description: Exploratory Data Analysis
  Responsibility: Abhishek Rangineni, Shishira Rudrabhatla
  Contribution: Exploratory data analysis is key for future tasks, so each contributed 50 percent as their part.

  Description: Sentiment Analysis
  Responsibility: Nathisha Marru, Shishira Rudrabhatla, Venkata Sai Rahul Kumar
  Contribution: We created a new model and Methodology for sentiment analysis involved everyone from the team.

  Description: Combining both Spam filtering and sentiment analysis
  Responsibility: Abhishek Rangineni, Venkata Sai Rahul Kumar, Shishira Rudrabhatla
  Contribution: This is focused and involved by each member of the team.

Description: Model testing and training

Responsibility: Abhishek Rangineni, Venkata Sai Rahul Kumar, Nathisha Marru, Shishira Rudrabhatla,

Contribution: We combined both spam filtering and sentiment analysis and built the new neural network model. This is focused and involved by each member of the team.

References

Ezpeleta, E., Zurutuza, U., & Gómez Hidalgo, J. M. (2016, April). Does sentiment analysis help in Bayesian spam filtering? In *International Conference on Hybrid Artificial Intelligence Systems* (pp. 79-90). Springer, Cham.

Ghiassi, M., Lee, S., & Gaikwad, S. R. (2022). Sentiment analysis and spam filtering using the YAC2 clustering algorithm with transferability. *Computers & Industrial Engineering*, *165*, 107959.

Whitelaw, C., Garg, N., & Argamon, S. (2005, October). Using appraisal groups for sentiment analysis. In *Proceedings of the 14th ACM international conference on Information and knowledge management* (pp. 625-631).

Debnath, K., & Kar, N. (2022, May). Email Spam Detection using Deep Learning Approach. In *2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON)* (Vol. 1, pp. 37-41). IEEE.

Ismail, S. S., Mansour, R. F., El-Aziz, A., Rasha, M., & Taloba, A. I. (2022). Efficient E-Mail Spam Detection Strategy Using Genetic Decision Tree Processing with NLP Features. *Computational Intelligence and Neuroscience*, *2022*.

https://www.researchgate.net/publication/220553508_Survey_on_Spam_Filtering_Techniques

https://www.researchgate.net/publication/228940659_Using_old_spam_and_ham_samples_to_train_email_filters

https://link.springer.com/article/10.1007/s10462-022-10195-4

https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

https://www.kaggle.com/datasets/nitishabharathi/email-spam-dataset?select=lingSpam.csv

https://www.kaggle.com/datasets/karthickveerakumar/spam-filter

https://www.sciencedirect.com/science/article/pii/S2405844018353404#fig5