Project17: Email Spam Classification

1st Moustafa Khairi
Project Leader

2nd Peng Zuojia

3rd Zhang Shouhua

4th Liu Qishu

***Abstract*—Spam detection is one of the major applications of Machine Learning in the interwebs today. Pretty much all of the major email service providers have spam detection systems built-in and automatically classify such mail as 'Junk Mail'.**
Being able to identify spam emails is a binary classification problem as emails are classified as either 'Spam' or 'Not Spam' , also known as 'Ham'.
The main pipeline for any NLP (natural language processing) project includes data pre-processing, feature extraction, and modeling phases. Through this project, we will implement different combinations of text pre-processing, feature extraction and reduction, and ML (machine learning) models to reach the best pipeline for email spam classification.
Also, we will test how paraphrasing can influence the performance of our model. Moreover, we aim at deploying our best model on AWS (Amazon Web Services) to get an automated pipeline and build a simple web app that takes an email as input and predict whether it is a spam email or not.

I. Project Repository

All project files can be found through this link:
https://github.com/MustafaAwny/Email-Spam-Classification-Deployment/

II. Introduction

Nowadays, spam emails occupy more than 70% of all email traffic [1]. Spam emails are considered a huge challenge that got a broad attention in research. Big world company such as Google spent a lot of resources to solve such a problem. Many filtering approaches are being proposed everyday to face the spammers development techniques to break the spam filtering systems.

The negative effect spam has on companies is greatly related to financial aspects and the productivity of employees in the workplace. A typical user receives about 40–50 emails per day. Automating the spam filtering process is the end goal that will save companies, users, and teams a lot of time they spend on this silly process and help them invest this time in more productive tasks.

The main target for individuals and companies is to establish an efficient automatic email filtering systems [2], [3]. This system does not necessarily classifies whether this is a spam or not but it can be used for multi-class classification, for example, ads, regular content, spam and so on.

Email classification techniques are classified into different categories: supervised machine learning, unsupervised machine learning, semi supervised machine learning, content-based learning, statistical learning [4], [5], Heuristic or Rule Based, Previous Likeness Based and Adaptive ones .

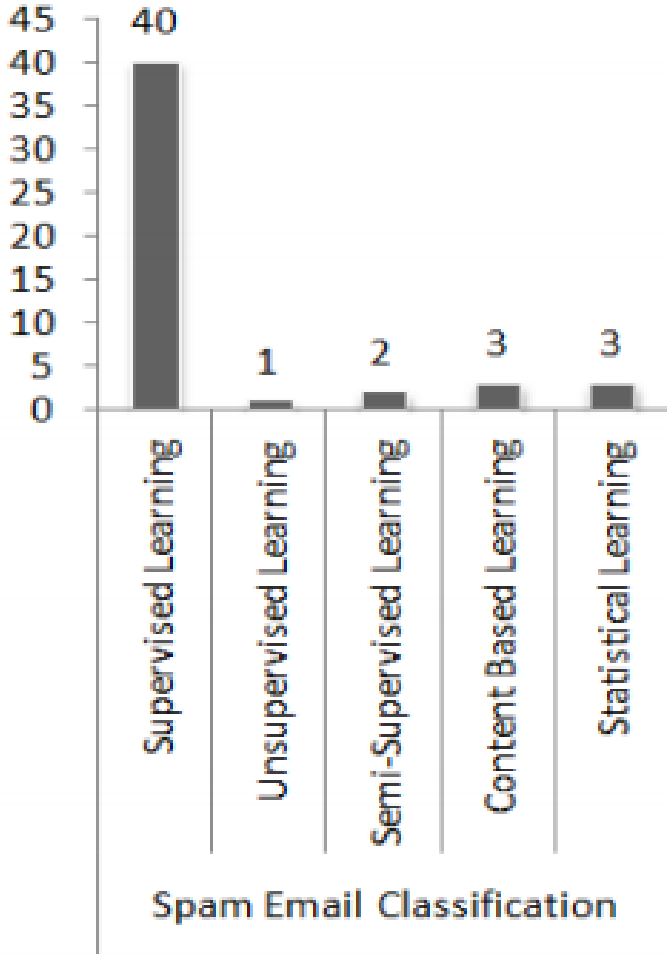

Fig. 1. Classification Techniques Frequencies — Ref: Email Classification Research Trends: Review and Open Issues

Supervised machine learning models depend on the data being labeled to be able to make predictions. It can be divided in Regression and Classification models and our focus in this project is on the classification models.

The Naive Bayes classifier is a simple statistical algorithm that provides amazingly better results. It has been used in many spam classification studies [6, 7, 8, 9], and has become somewhat of a benchmark. It is based on the Bayes theorem of conditional probability and it is called naive as it assumes the Independence of the features, no correlation between them[10].

SVMs or support vector machine is a very popular clas-

sification model because of its powerful performance and its relative low computation cost. The main purpose of the model is to minimize the classification error and the margin error. So, it is nut just about right classification but it is also concerned with accurate classification, having a good margin and that is why it is called support vector as a vector is used for calculating and maximizing the margin error [11, 12, 13, 14, 15, 16].

Most classification problems state of the art results are achieved using ANN or artificial neural network [17]. Every layer in the network consists of perceptrons followed by activation functions to handle the non-linearity which results in decent results.

Unsupervised learning can also be very useful when we do not have labeled data. It can be applied to solve different problems such as clustering and recommendation systems.

The KNN (k-Nearest Neighbour) classifier is an example based classifier; it means that the classifier needs training files for comparison rather than classification. It can be used to assign the email to one of the categories discussed before using the k most similar text chunk (neighbours). It is a way of measuring the similarity.

Semi-supervised machine learning is actually a supervised machine learning technique using small labeled data and is not a supervised machine learning method that requires large labeled data. This approach is conducted by using some labeled data to facilitate a classifier in labeling unlabeled data. One example is active learning.

As mentioned in the Machine Learning For Email Spam Filtering; Review, Approaches and Open Research Problems paper: Content-based email classification techniques use keywords in emails for classification. Statistical learning techniques assign a probability or score to each keyword, and the overall score or probability is used to classify incoming emails.

Heuristic or Rule Based Spam Filtering Technique: This approach uses already created rules or heuristics to assess a huge number of patterns mainly based on regular expressions. It assigns a score or anomalies score according to the number of detected patterns and filter the email based on a score threshold for example or another criteria. This approach needs to be updated frequently with the new patterns discovered as the spammers use them to escape without been noticed from email filters [18]. A good example of a rule based spam filter is SpamAssassin [19], [20].

Previous Likeness Based Spam Filtering Technique: This approach is memory based and it is similar to the KNN approach. It uses different similarity metrics to measure the matching between the new email and the predefined classes or categories. A similar approach is widely used in recommendation systems to recommend new products based on the user's history.

A. Related Work

Pre-processing techniques are wildly used in spam detection. It is necessary to prepare the emails to be ready for analysis. These pre-processing steps can affect the overall performance of the detection algorithm. The varied combination of pre-processing methods used in many spam detection studies motivates us to conduct further investigations. So we chose multiple pre-processing methods(stop-word removal, non-word removal and lemmatization) and formed different combinations to see which performs the best.

A GANN (Generalized Additive Neural Network) [21] is being used because of its powerful ability of extracting features even within the case of unsupervised learning where labeled are not provided. Pattern recognition is another benefit of a GANN and can help in the classification of spam messages.

M. Basavaraju et al. (2010) proposed the text based clustering method for spam detection. The Porters stemming and stopping techniques are used for the text data pre-processing. Different clustering models such as the Hierarchical ones are used for the clustering purpose. A combination of neighboring techniques is used as the final classification model, which produced good results. The vector space model is used to calculate the inverse document frequency of each word i.e. tf-idf test patterns. After clustering of training patterns the non-spam data are stored in the centroids. Test patterns and centroids are passed in to the classification module for spam and non-spam detection.

Sudhakar. P et al. (2011) developed fuzzy logic concept for spam detection. They applied five fuzzy rules with five fuzzy parameters. The 5 fuzzy parameters are sender address, sender IP, subject words, content words, and attachments. All the mentioned parameters are used for making a comparison to what was called black and white lists t measure the similarity and if there is a strong match based on some threshold, the email would be classified as spam or ham, black or white list. One of the cons of this approach is that it not memory wise efficient.

Rafiqul Islam et al. (2010) presented a paper on instance selected method for classification of spam. The ISM is used to select subset of instances. The new dataset helps to reduce the spam rate. The approach is similar to the feature reduction techniques but it related to number of samples in the dataset. It is about selecting a smaller yet representative sampe of the dataset that will be feed to the feature extractors and the models after that. They used tokenization and domain specific feature selection methods for feature extraction. They included the behavioural features like the frequency of sending/receiving emails, email attachment, type of attachment, size of attachment and length of the email for improving performance and reducing false positive problems. They have tested five base classifiers Naive Bayes, SVM, IB1, Decision Table and Random Forest. They also included adaptive boosting as meta-classifier for their test. They concluded that their classification method achieved 97 $\%$ of accuracy.

Malayalam Paraphrase Detection by Sindhu.L Sumam Mary Idicula in 2016. Observations: South Dravidian is employed as input language. The projected linguistics approach for distinguishing the paraphrases includes 3 phases – matching identical tokens, matching lemmas matching with synonyms replaced. Similarity comparison is performed at the

sentence level mistreatment the Jaccard, Containment, Overlap and cos similarity metrics.

Convolutional Neural Network for Paraphrase Identification by Wenpeng YinHinrich Schutze in 2015. Observations: Conferred the deep learning design "Bi-CNN-MI" for paraphrase identification (PI). BiCNN-MI:"Bi-CNN" stands for double CNNs,"MI"for multi granular interaction options.

Paraphrase Detection Using Recursive Autoencoder by Eric Huang in 2011.

A Semantic Similarity Approach to Paraphrase Detection by Samuel Fernando Mark Stevenson in 2009.

B. Novel Approach

Our novel approach lies in experimenting with different approaches in all the NLP phases. We are not just benchmarking our results using the state of the art techniques or algorithms used, but we aim at benchmarking the results using other practical approaches that lead to performance increase.

Also, we use paraphrasing and record its influence on the model performance, which is something not popular in email spam classification. Paraphrasing is about rewriting sentences using different words leading to the same meaning of the original sentence is called paraphrasing while paraphrasing identification task is the task of detecting the sentence that is paraphrased from another [27].

III. Email Spam Datasets

PU [22]: total 7101 email (spam = 3020 and ham = 4080).

SpamBase [23]: total 4601 email (spam = 1318 and ham = 2788).

EnronSpamCorpus [24]: total 30041 email (spam = 13496 and ham = 16545).

SpamAssasin [25]: total 10744 email (spam = 3793 and ham = 6951).

TREC [26]: total 92189 email (spam = 52790 and ham = 39399).

CCERT [27]: total 34360 email (spam = 25088 and ham = 9272).

A. Data Used

The data set is Euron-spam corpus (http://www2.aueb.gr/users/ion/data/enron-spam/). It contains 33716 emails in 6 directories. Each of the 6 directories contains 'ham' and 'spam' folders. Total number of non-spam (ham) emails is 16545 and total number of spam emails is 17171. Because there are some emails that can not be decoded in Euron-spam corpus, we filtered them. There are 31978 emails left. Total number of non-spam emails and spam emails are 16183 and 15795 respectively.

IV. Methodology

Figure2 [28] shows the general architecture of automatic email classification. As shown in the figure, the email classification process consists of three phases: pre-processing, feature extraction, and modeling or classification. To develop an automatic email classifier we need to use data engineering to gather the dataset. Data gathering is a very important step as it controls the output of the next phases. If the data is corrupted or has errors, this means that all our effort is dedicated in the wrong direction. The second phase is data pre-processing or data cleaning.

The data cleaning task is generally known as data pre-processing. In the pre-processing phase, an email is converted into token of words. The pre-processing level also eliminates unnecessary words or stop words to reduce the amount of data that needs to be examined. Finally, in the pre-processing phase, stemming and lemmatization are performed on token of words to convert them into their root forms. The output of this phase is directly fed to the next phase which is the feature extraction phase.

Feature engineering is a broad area of research and it depends on a lot of factors such as the main objective, the domain knowledge, the data type and many more. It is a very important step and helps a lot in increasing the model performance.

The last phase is the modeling or classification phase where the feature extracted through the previous phase are feed to the machine learning model to classify whether these features represent a spam email or not. The machine learning models used in this phase can be classic machine learning models or deep learning models.

A. Pre-processing

Generally, in pre-processing domain, the most common methods are: noise removal(stop word, alpha numeric word and punctuation), stemming, lemmatization, term frequency and TF-IDF, etc. In our case, we chose stop-word removal, non-word removal and lemmatization as pro-processing methods.

- Noise removal Noise in the language processing domain means the text is not relevant to the context of the data and the end result. In our case, it can be stop words, alphanumeric words and punctuation. We are going to remove all the stop word in the text, in order to do that, we use a list of stop word, then iterates the text by token and removes tokens which are present in stop word list.
- Lemmatization There are two ways to normalize the word form: Stemming and Lemmatization. In contrast to stemming, lemmatization is an organized procedure for obtaining the root form of the word, by utilizing dictionaries and morphological analysis.
- Non-word removal The result of this method is that, every space and the words whose length equals one will be removed. The efficiency can be improved by using this method.

B. Feature Extraction

After the above pre-processing, we separately divided the spam and non-spam emails into training set and testing set in 60:40 in split randomly. We extract feature words from training set and adopt three feature extraction strategies described as follows:

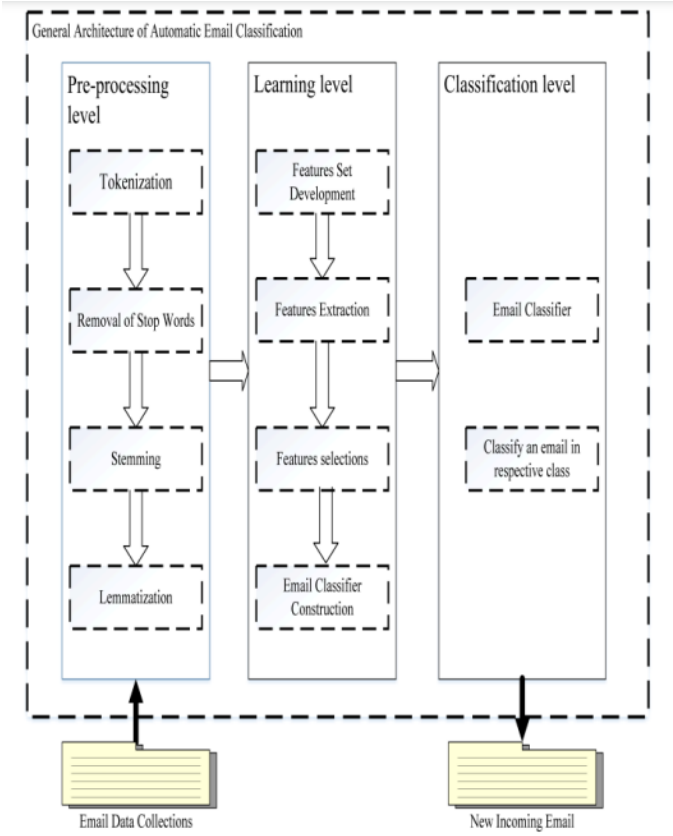

Fig. 2. NLP Pipeline — Ref: Email Classification Research Trends: Review and Open Issues

- Created a dictionary as the feature set using the 3000 words with highest word frequency.
- Extract 3000 features with TF-IDF.
- Extract 3000 features with bigram TF-IDF.

C. Feature Reduction

After feature extraction, we will have large amount of features extracted from emails. Using the full feature-set is redundant as some components are correlated and it can be reduced to a smaller set without losing much information. We use two methods, principal component analysis(PCA) and linear discriminate analysis(LDA) to implement feature reduction.

In our experiment, after extracting TFIDF features, we use PCA or LDA to reduce feature dimensions, then use the reduced features for model training. Finally, we will test the model and find the correlation between accuracy and initial feature size and principal components. Figure 3 illustrates the pipeline and role of feature reduction.

Principal Component Analysis Principal component analysis(PCA) is a traditional feature reduction approach. It is a popular unsupervised linear method which projects the original feature set into a linear subspace. The subspace is obtained by finding maximum variance so that maximum information can be preserved. [36]. Principal components are eigenvectors of the covariance matrix of the original feature set, and PCA consists of computing eigenvalues and eigenvectors [37].

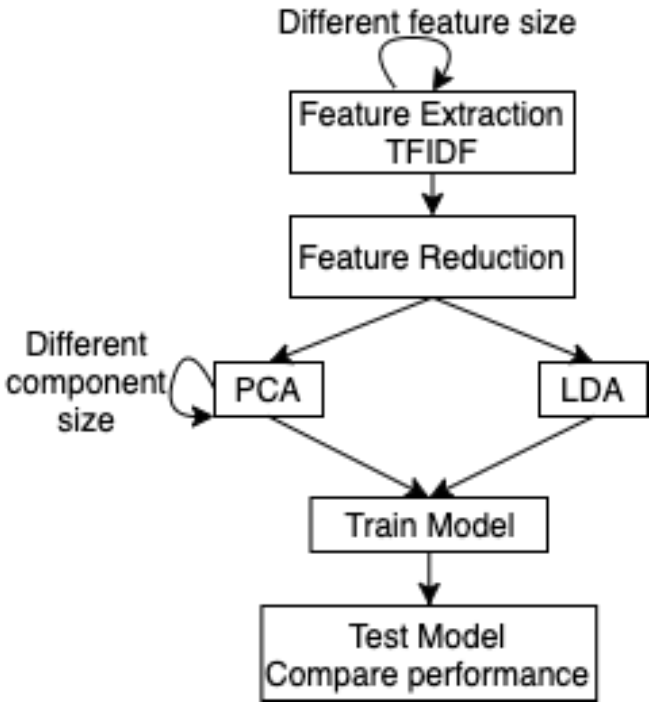

Fig. 3. Feature reduction Pipeline

Linear Discriminate Analysis Linear Discriminate Analysis (LDA) is a method that best separates data into different classes by considering differences among features in the original feature set. LDA consists of maximizing between-class scatter differences as well as minimizing in-class scatter differences. LDA is achieved by obtaining the highest ratio of between-class variance to in-class variance [38].

D. Modeling (classifiers)

LSTM We constructed a very simple LSTM (long short term memory network) that with an embedded layer to take the text sequence into consideration using PyTorch framework.

```
def __init__(self, embedding_dim, hidden_dim, vocab_size):
    """
    Initialize the model by settingg up the various layers.
    """
    super(LSTMClassifier, self).__init__()

    self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
    self.lstm = nn.LSTM(embedding_dim, hidden_dim)
    self.dense = nn.Linear(in_features=hidden_dim, out_features=1)
    self.sig = nn.Sigmoid()

    self.word_dict = None
```

Fig. 4. LSTM Network Architecture

We trained it for only 10 epochs with hidden dimension of 200 and used the binary cross entropy loss as our loss function[31].

E. Paraphrasing

As we mentioned before, major difficulties faced in natural language processing are ambiguity where the same text has several possible interpretations. The paraphrase is a way of conveying the same content without compromising the meaning. It is an alternate form in the same language stating the same semantic content with the help of reframing or rearranging the phrases of a sentence. Paraphrases can occur at the word level, phrase level or even sentence level. Paraphrasing is of two types: Paraphrase Generation and Paraphrase Detection [33]. Our main problem here is concerning paraphrasing generation. We want to increase the our dataset samples using paraphrasing and test how this will affect the model performance.

We used the open source implementation of the "**An Empirical Comparison on Imitation Learning and Reinforcement Learning for Paraphrase Generation**" paper [34] to apply reinforcement learning to paraphrasing generation. Generating paraphrases from given sentences involves decoding words step by step from a large vocabulary. To learn a decoder, supervised learning which maximizes the likelihood of tokens always suffers from the exposure bias. Although both reinforcement learning (RL) and imitation learning (IL) have been widely used to alleviate the bias, the lack of direct comparison leads to only a partial image on their benefits.

Method: as mentioned in the paper. Given an input sentence x = (x1, x2, · · · , xS) with length S, a paraphrase generation model outputs a new sentence y = (y1, y2, · · · , yT) with length T that shares the same meaning with x. The widely adopted framework on paraphrase generation is the encoder-decoder framework [35]. We use the pointer-generator model [38] as the base model, which is state-of-the-art model on paragraph generation.

Psedu-code as mentioned in the paper.

As mentioned in the paper, the evaluation is done on the Quora Question Pair Dataset and the Twitter URL Paraphrasing Dataset [36]. Both datasets contain positive and negative examples of paraphrases, and we only keep the positive examples for our experiments as in prior work of paraphrase generation. For the Quora dataset, we follows the configuration of and split the data into 100K training pairs, 30K testing pairs and 3K validation pairs. For the Twitter dataset, since our model cannot deal with the negative examples as do, we just obtain the 1-year 2,869,657 candidate pairs from https://languagenet.github.io, and filter out all negative examples. Finally, we divided the remaining dataset into 110K training pairs, 3K testing pairs and 1K validation pairs. We used BLEU as our metric [37]. For the convenience of comparison, we also calculate the average of the scores.

So, after training the model as per figure 6

We used the trained wights to do inference on our own dataset.

F. Deployment

We used AWS (amazon web services) SageMaker to construct and deploy our main pipeline from end to end. Our goal

Input: A training example $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$, current schedule rates $\alpha^{(i)}, \beta^{(i)} \in [0, 1]$, learning rate η
Initialize $L(\boldsymbol{\theta}) \leftarrow 0$
for $t = 1, \ldots, T$ **do**
  $p_1, p_2 \sim \text{Uniform}(0, 1)$
  $\tilde{y}_{t-1} \leftarrow y_{t-1}$ **if** $(p_1 < \alpha^{(i)})$ **else** $\hat{y}_{t-1}$
  $\boldsymbol{h}_t = \boldsymbol{f}(\boldsymbol{h}_{t-1}, \tilde{y}_{t-1}, \boldsymbol{x})$
  $\hat{y}_t \leftarrow \texttt{Decode}(\pi(y \mid \boldsymbol{h}_t))$
  $\tilde{y}_t \leftarrow y_t$ **if** $(p_2 < \beta^{(i)})$ **else** $\hat{y}_t$
  $L(\boldsymbol{\theta}) \leftarrow L(\boldsymbol{\theta}) + \log \pi(\tilde{y}_t \mid \boldsymbol{h}_t)$
end for
$\delta\boldsymbol{\theta} \leftarrow \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \cdot r(\tilde{\boldsymbol{y}^{(i)}}, \boldsymbol{y}^{(i)})$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta \cdot \delta\boldsymbol{\theta}$

The REINFORCE Algorithm. When $\alpha = 0$, $\beta = 0$, and $\texttt{Decode}(\pi(y \mid \boldsymbol{h}_t))$ is defined as as:

$$\begin{aligned} &\texttt{Decode}(\pi_\theta(y \mid \boldsymbol{h}_{t-1})) \\ &\quad = \text{Random\_Sampling}(\pi_\theta(y \mid \boldsymbol{h}_{t-1})), \end{aligned} \tag{3}$$

Fig. 5. Paraphrasing Algorithms Logic

```
Save best model at /content/Reinforce-Paraphrase-Generation/log_twitter/best_model/model_best_67000
steps 67000, train_loss: 189.034378, val_loss: 2.550899
steps 67100, current_loss: 136.221619, avg_reward: 0.000000
steps 67200, current_loss: 208.854156, avg_reward: 0.000000
steps 67300, current_loss: 172.767822, avg_reward: 0.000000
steps 67400, current_loss: 178.524017, avg_reward: 0.000000
steps 67500, current_loss: 156.246979, avg_reward: 0.000000
steps 67600, current_loss: 210.816025, avg_reward: 0.000000
steps 67700, current_loss: 210.217163, avg_reward: 0.000000
steps 67800, current_loss: 117.631966, avg_reward: 0.000000
steps 67900, current_loss: 234.012878, avg_reward: 0.000000
steps 68000, current_loss: 153.777283, avg_reward: 0.000000
Save best model at /content/Reinforce-Paraphrase-Generation/log_twitter/best_model/model_best_68000
steps 68000, train_loss: 153.777283, val_loss: 2.543934
steps 68100, current_loss: 182.034973, avg_reward: 0.000000
steps 68200, current_loss: 147.179276, avg_reward: 0.000000
steps 68300, current_loss: 160.749725, avg_reward: 0.000000
steps 68400, current_loss: 157.881042, avg_reward: 0.000000
steps 68500, current_loss: 173.906311, avg_reward: 0.000000
steps 68600, current_loss: 153.432114, avg_reward: 0.000000
steps 68700, current_loss: 144.322266, avg_reward: 0.000000
steps 68800, current_loss: 164.634460, avg_reward: 0.000000
steps 68900, current_loss: 177.991089, avg_reward: 0.000000
steps 69000, current_loss: 218.003952, avg_reward: 0.000000
steps 69000, train_loss: 218.003952, val_loss: 2.549614
```

Fig. 6. Reinforcement Learning Model Training

will be to have a simple web page which a user can use to enter an email text. The web page will then send the email off to our deployed model which will predict the whether the entered email is spam or not.

We uploaded the training dataset to an S3 Bucket in order for our training code to access it. For a model to be deployed on AWS SageMaker, a model should comprise three objects; Model Artifacts, Training Code, and Inference Code.

There are three parameters that we may wish to tweak to improve the performance of our model. These are the embedding dimension, the hidden dimension and the size. We will likely want to make these parameters configurable in the training script so that if we wish to modify them we do not

need to modify the script itself.

When a PyTorch model is constructed in SageMaker, an entry point must be specified. This is the Python file which will be executed when the model is trained. The way that SageMaker passes hyperparameters to the training script is by way of arguments. These arguments can then be parsed and used in the training script.

When deploying a model you are asking SageMaker to launch a compute instance that will wait for data to be sent to it. As a result, this compute instance will continue to run until you shut it down. This is important to know since the cost of a deployed endpoint depends on how long it has been running for. Once deployed, we can read in the test data and send it off to our deployed model to get some results. Once we collect all of the results we can determine how accurate our model is.

Web App Now that we know that our model is working, it's time to create some custom inference code so that we can send the model an email which has not been processed and have it determine its class.

By default the estimator which we created, when deployed, will use the entry script and directory which we provided when creating the model. However, since we now wish to accept a string as input and our model expects a processed email, we need to write some custom inference code.

We will store the code that we write in the serve directory. Provided in this directory is the model.py file that we used to construct our model, a utils.py file which contains the email-ToWords and convertAndPad pre-processing functions which we used during the initial data processing, and predict.py, the file which will contain our custom inference code. Note also that requirements.txt is present which will tell SageMaker what Python libraries are required by our custom inference code.

When deploying a PyTorch model in SageMaker, you are expected to provide four functions which the SageMaker inference container will use.

ModelFn: This function is the same function that we used in the training script and it tells SageMaker how to load our model. InputFn: This function receives the raw serialized input that has been sent to the model's endpoint and its job is to de-serialize and make the input available for the inference code. OutputFn: This function takes the output of the inference code and its job is to serialize this output and return it to the caller of the model's endpoint. PredictFn: The heart of the inference script, this is where the actual prediction is done and is the function which you will need to complete. For the simple website that we are constructing during this project, the InputFn and OutputFn methods are relatively straightforward. We only require being able to accept a string as input and we expect to return a single value as output. You might imagine though that in a more complex application the input or output may be image data or some other binary data which would require some effort to serialize.

The default behaviour for a deployed PyTorch model is to assume that any input passed to the predictor is a numpy array. In our case we want to send a string so we need to construct a simple wrapper around the RealTimePredictor class to accomodate simple strings. In a more complicated situation you may want to provide a serialization object, for example if you wanted to sent image data.

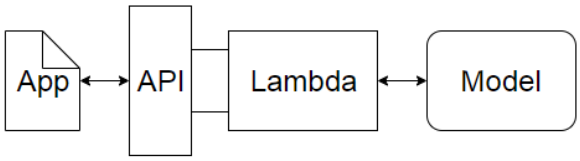

Fig. 7. Deployment Phases

The diagram figure 7 gives an overview of how the various services will work together. On the far right is the model which we trained above and which is deployed using SageMaker. On the far left is our web app that collects a user's email, sends it off and expects a positive or negative sentiment in return.

In the middle is where some of the magic happens. We will construct a Lambda function, which you can think of as a straightforward Python function that can be executed whenever a specified event occurs. We will give this function permission to send and recieve data from a SageMaker endpoint.

Lastly, the method we will use to execute the Lambda function is a new endpoint that we will create using API Gateway. This endpoint will be a url that listens for data to be sent to it. Once it gets some data it will pass that data on to the Lambda function and then return whatever the Lambda function returns. Essentially it will act as an interface that lets our web app communicate with the Lambda function.

Setting up a Lambda function The first thing we are going to do is set up a Lambda function. This Lambda function will be executed whenever our public API has data sent to it. When it is executed it will receive the data, perform any sort of processing that is required, send the data (the email) to the SageMaker endpoint we've created and then return the result.

Part A: Create an IAM Role for the Lambda function Since we want the Lambda function to call a SageMaker endpoint, we need to make sure that it has permission to do so. To do this, we will construct a role that we can later give the Lambda function.

Part B: Create a Lambda function

Part C: Setting up API Gateway Now that our Lambda function is set up, it is time to create a new API using API Gateway that will trigger the Lambda function we have just created.

V. Results and Discussions

A. Classification

The feature vector of each email was created according to the dictionary as described in section 3.Then the feature matrix of training set and testing set were built. We checked how the classifiers such as Support Vector Machines, Multinomial Naive Bayes (MNB) models, Decision Tree (DT) and Random Forest (RF) performed. We got the results on 12792 emails of

TABLE I
THE COMPARISON OF CLASSIFICATION RESULTS

Dictionary size	Classifier	Confusion matrix	Accuracy
3000	MNB	[[6049 269] [232 6242]]	0.9608348968105066
	SVM	[[6085 233] [174 6300]]	0.968183239524703
2000	MNB	[[5994 324] [265 6209]]	0.9539555972482802
	SVM	[[6091 227] [174 6300]]	0.9686522826766729
1000	MNB	[[5873 445] [256 6218]]	0.9452001250781739
	SVM	[[6023 295] [186 6288]]	0.9623983739837398
500	MNB	[[5596 722] [280 6194]]	0.9216697936210131
	SVM	[[5822 496] [215 6259]]	0.9444183864915572

TABLE II
CONFUSION MATRIX

	ham	spam
ham	TN	FP
spam	FN	TP

testing set in Table1. The format of the confusion matrix is shown in Table 2. We can see that SVM has performed slightly better than Multinomial Naive Bayes classifier in detecting spam emails correctly. SVM and MNB are much better than DT and RF classifiers. We also did the same work as above with the class CountVectorizer and 3000 feature words. The result is shown in Table3. However, it was much worse than the result with the dictionary. This is very unexpected. We designed the new program to repeat the procedure using tf-idf feature and bigram tf-idf feature separately. Reduce the number of feature from 3000 to 2000, 1000 and 500. The

TABLE III
THE RESULTS WITH COUNTVECTORIZER

Classifier	Confusion matrix	Accuracy
MNB	[[3838 2480] [2065 4409]]	0.6446998123827392
SVM	[[4418 1900] [2385 4089]]	0.6650250156347718
DT	[[131 6187] [0 6474]]	0.5163383364602877
RF	[[1169 5149] [592 5882]]	0.5512038774233896

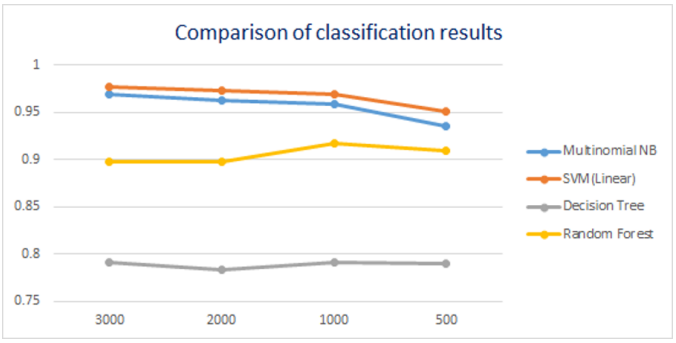

Fig. 8. Comparison of classifiers with TF-IDF

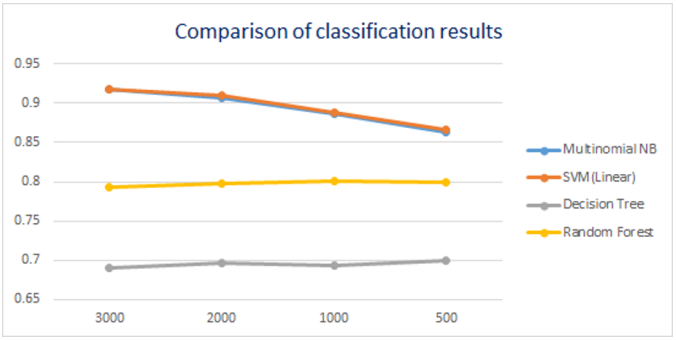

Fig. 9. Comparison of classifiers with Bigram TF-IDF

results are shown in Table4, Table5, Figure8, Figure9.

According to the results, SVM and Multinomial Naive Bayes classifiers are much better than Decision Tree and Random Forest classifiers. SVM classifier has performed slightly better than MNB classifier. RF classifier has performed better than DT classifier. SVM and MNB classifiers perform better with more features. DT and RF classifiers fluctuate as the number of features decreases. They perform better with TF-IDF feature than with bigram TF-IDF feature. SVM and MNB classifiers perform slightly better with TF-IDF feature than with the dictionary.

LSTM

The LSTM model reached the best accuracy, 0.9756, very easily. Training loss in figure 10

```
Epoch: 1, BCELoss: 0.6699544471137377
Epoch: 2, BCELoss: 0.5918508999201716
Epoch: 3, BCELoss: 0.501250513962337
Epoch: 4, BCELoss: 0.4434727558067867
Epoch: 5, BCELoss: 0.388338459997761
Epoch: 6, BCELoss: 0.3590703418060225
Epoch: 7, BCELoss: 0.32612842625501204
Epoch: 8, BCELoss: 0.30269722549282774
Epoch: 9, BCELoss: 0.2787549413588582
Epoch: 10, BCELoss: 0.2681142274214297
2020-11-03 13:01:31,072 sagemaker-containers INFO     Reporting training SUCCESS
```

Fig. 10. LSTM Training Loss

B. Pre-processing Strategy

In order to find out the best pre-processing strategy, we propose 7 different strategies combining those 3 methods:

TABLE IV
THE RESULTS OF CLASSIFICATION WITH TF-IDF FEATURE

Feature number	Classifier	Confusion matrix	Accuracy
3000	MNB	[[6093 225] [165 6309]]	0.9695121951219512
	SVM	[[6128 190] [97 6377]]	0.9775641025641025
	DT	[[3732 2586] [94 6380]]	0.7904940587867417
	RF	[[5093 1225] [75 6399]]	0.8983739837398373
2000	MNB	[[6011 307] [175 6299]]	0.9623202001250781
	SVM	[[6075 243] [92 6382]]	0.9738117573483427
	DT	[[3689 2629] [139 6335]]	0.7836147592245153
	RF	[[5103 1215] [97 6377]]	0.8974358974358975
1000	MNB	[[5972 346] [187 6287]]	0.9583333333333334
	SVM	[[6078 240] [149 6325]]	0.9695903689806129
	DT	[[3672 2646] [27 6447]]	0.7910412757973734
	RF	[[5362 956] [98 6376]]	0.9176047529706066
500	MNB	[[5702 616] [204 6270]]	0.9358974358974359
	SVM	[[5864 454] [172 6302]]	0.9510631644777986
	DT	[[3661 2657] [26 6448]]	0.7902595372107567
	RF	[[5289 1029] [124 6350]]	0.9098655409631019

TABLE V
THE RESULTS OF CLASSIFICATION WITH BIGRAM TF-IDF FEATURE

Feature number	Classifier	Confusion matrix	Accuracy
3000	MNB	[[6093 225] [165 6309]]	0.9695121951219512
	SVM	[[6128 190] [97 6377]]	0.9775641025641025
	DT	[[3732 2586] [94 6380]]	0.7904940587867417
	RF	[[5093 1225] [75 6399]]	0.8983739837398373
2000	MNB	[[6011 307] [175 6299]]	0.9623202001250781
	SVM	[[6075 243] [92 6382]]	0.9738117573483427
	DT	[[3689 2629] [139 6335]]	0.7836147592245153
	RF	[[5103 1215] [97 6377]]	0.8974358974358975
1000	MNB	[[5972 346] [187 6287]]	0.9583333333333334
	SVM	[[6078 240] [149 6325]]	0.9695903689806129
	DT	[[3672 2646] [27 6447]]	0.7910412757973734
	RF	[[5362 956] [98 6376]]	0.9176047529706066
500	MNB	[[5702 616] [204 6270]]	0.9358974358974359
	SVM	[[5864 454] [172 6302]]	0.9510631644777986
	DT	[[3661 2657] [26 6448]]	0.7902595372107567
	RF	[[5289 1029] [124 6350]]	0.9098655409631019

stop-word removal, non-word removal and lemmatization.

Strategy1: Stop-word + non-word removal + lemmatization
Strategy2: Stop-word removal
Strategy3: Non-word removal
Strategy4: Lemmatization
Strategy5: Stop-word removal + non-word removal
Strategy6: Stop-word removal + lemmatization
Strategy7: Non-word removal + lemmatization

By using the result of the previous step: SVM classifier performs the best with the tf-idf feature in detecting spam emails correctly. So we will decide which strategy is the best by comparing the acurrency of the SVM classifier with the ti-idf feature.

After comparing the accuracy under SVM classifier with the tf-idf feature, strategy2 which only applied stop word removal has the poorest performance, on the other hand, the strategy which only applied lemmatization has the best performance.

C. Reducing feature size

1) PCA performance: Figure 12 illustrates the correlation between PCA performance with both the initial feature size and the principal component size.

x-axis is the number of principal components after feature reductions (we keep the most significant components). y-axis is the accuracy of the model, which as a result is also an indicator of the performance of PCA. The model is LinearSVC

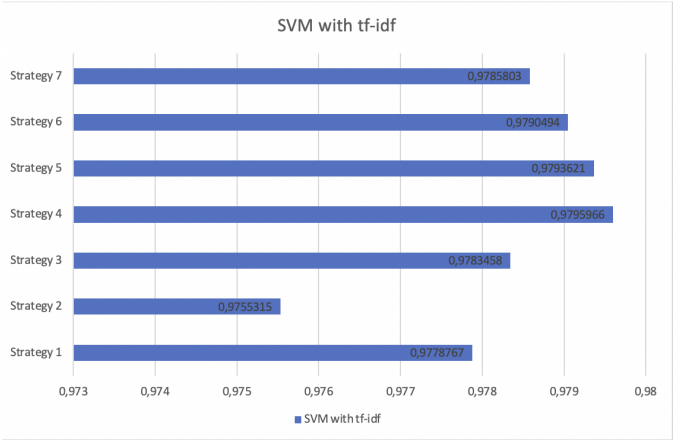

Fig. 11. Comparison of different strategies

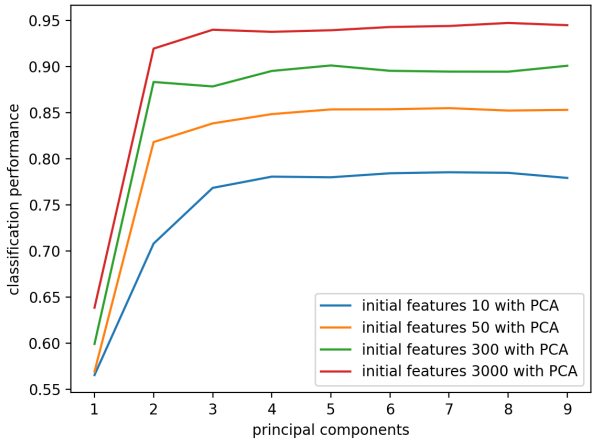

Fig. 12. PCA performance against feature size and principal component size

as it has the best performance according to our previous work.Different colors indicate different feature sizes of the original feature set, which are 3000, 300, 50 and 10 in from top to bottom respectively.

We can conclude 2 things from the graph:

1) With the same size of principal components, PCA performs better with larger initial feature set.
   When initial feature size is 10, the highest accuracy it can achieve is 0.75 while when initial feature size is 3000, the highest accuracy is above 0.9.
2) With the same size of initial feature set, PCA performs better with a larger principal component size.
   To analyse the correlation between principal components and PCA performance, we focus mainly on the red line, which is for the initial feature size of 3000. The accuracy is just above 0.65 for 1 principal component and increases rapidly to above 0.9 for 2 principal components. After this point, accuracy remains nearly constant.

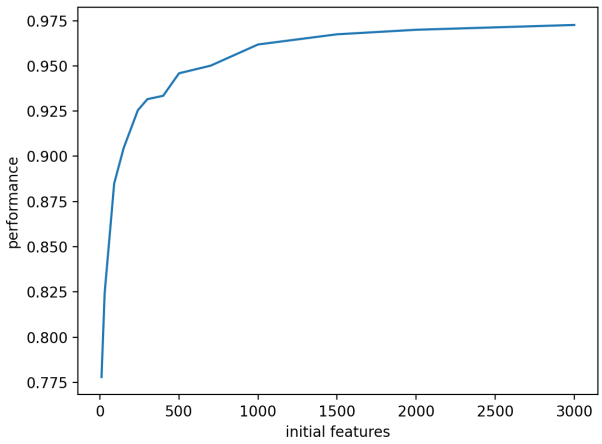

Fig. 13. LDA performance against different feature size

2) LDA performance: Figure 13 shows the correlation between LDA performance and initial feature size. As the initial feature size increases, LDA performance increases logarithmically. The accuracy is just above 0.775 when the initial feature size is 10. The accuracy increases rapidly to above 0.925 when the initial feature size is 500. After this point, the accuracy grows slowly as the feature size increases and it reaches 0.96 when the initial feature size is 3000.

D. Paraphrasing

We can see the output of the reinforcement model inference on our dataset in figure 14 and figure 15

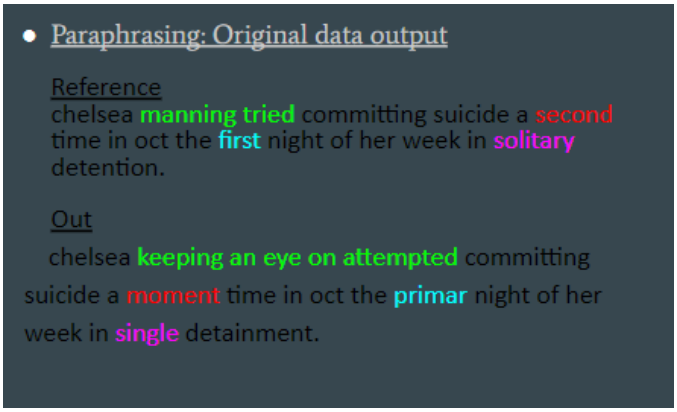

Fig. 14. Output on the original dataset

Model Performance

Paraphrasing increased the SVM model accuracy with about 0.4 after trying adding 50, 70 and 100 paraphrased samples to our dataset.

On the other hand, paraphrasing did not increase the LSTM model performance which makes sense as LSTMs mainly focuse on taking the sequence into consideration to be able

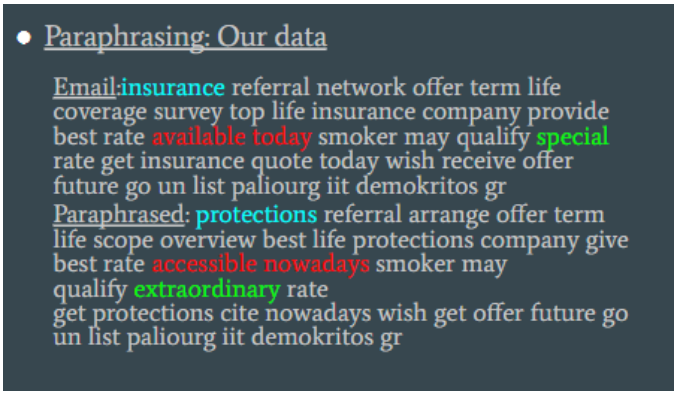

Fig. 15. Output on our Email dataset

to learn the context of the text. So, paraphrasing will not add anything to the context and therefore, will not affect the model performance.

Web App Deployment

The simple web app layout in figure 16

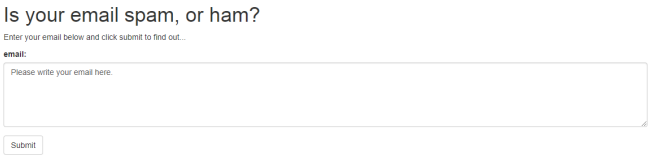

Fig. 16. Web App Layout

VI. Overall discussion

A. Classification

SVM and MNB classifiers are much better than DT and RF classifiers in accordance with the results. SVM classifier has performed a bit better than MNB. They perform better with TF-IDF feature than with bigram TF-IDF feature. SVM and MNB classifiers perform a bit better with TF-IDF feature than with the dictionary. In general, SVM classifier performs the best with the TF-IDF feature in detecting spam emails correctly.

As we mentioned before, the LSTM model reached the best accuracy very easily and after only a 10 epochs of training. It is important to mention that the SVM also reached a close accuracy to the LSTM one.

In general accuracy is not a good indicator in classification problems but in our case, the dataset is balanced and it is a binary classification problem. So, we do not need to use other metrics to compare such as Precision, Recall and so on.

There will be always the trade-off between higher accuracy, latency, computational power and computational cost. So, it depends on our main objective and resources we have.

In the Machine Learning world, experts tend to prefer the simplicity over complexity. For example, using SVM in our case would reach a decent accuracy and it is much less resources consuming than the LSTMs networks and the Neural Networks in general.

So, if our main objective is to achieve the best possible accuracy and we have the required resources, we will definitely go for the LSTM network as there are many potential ways to increase the accuracy. For example, training for more epochs, changing the hidden dimension, adding more layers to the network, we only used two layers, or using Transfer Learning.

If our main objective is to reach a decent accuracy with the minimum resources and building a real-time application, SVM would be a better option, especially for the inference purpose.

B. Pre-processing strategy

Even thought applying lemmatization as pre-processing strategy has the best result, if we compare the result which is 0.9795966 with the one without using any pre-processing strategy which is 0.9781113195747342, we can see that the improvement is not obvious at all. That is probably due to the classifier that we chose, since SVM is a non-probabilistic classifier, which means building hyperplanes can be done without going through pre-processing stages. So the pre-processing strategy may not be very helpful when using SVM classifier.

C. Reduced feature size

A reduced feature size can still achieve high accuracy with certain conditions, including a big initial feature size. Accuracy will increase logarithmically with bigger initial feature size or more principal components. The accuracy after applying PCA is above 0.94 with 3000 initial feature and 3 principal components. The accuracy after applying LDA is above 0.96 with 3000 initial feature size.

When we analyse the effect PCA has on accuracy with different parameters, we use principal components size as x-axis and use different colors to represent a different initial feature size. It's straightforward to find the correlation between the accuracy with both initial feature size and principal components in the same graph, which can be regarded as a highlight of this section.

D. Paraphrasing

There are many novel deep learning approaches that achieve state of the art results in paraphrasing that we have not tested yet as it is not the main objective of our project.

Also, there are many free online tools that are being used for the paraphrasing purpose, but we choose to go for the reinforcement learning approach to have a more deeper intuition about how paraphrasing works. In addition, when doing paraphrasing on many samples, using online tools would force us to do it manually which is time consuming and do not scale with large datasets size.

The simplest way to do paraphrasing is by using Google Translate API. For example, to translate a paragraph from English to another language an then, translate it back to English but as we mentioned this is not an efficient way to automate the process.

It is important to mention that there were a lot of improvements that can be done to the paraphrasing model we used. Due to the limited resources, we stopped training the model and used the saved weights for the inference purpose. Training on a large dataset such as Quora one from scractch requires a lot of resources and a powerful GPU. So, training for a longer time would definitely get us better results.

E. Deployment

We used the ml.p2.xlarge quota resource on AWS SageMaker which is not provided in free accounts but we asked for a resource limit increase and they accepted our request.

For more complex LSTMs architectures and for a longer training time, we would need a more powerful resource which which is paid. So, we had to compromise and minimize the resources we use.

VII. Conclusion

Building an end to end pipeline for text classification task in not an easy thing, especially, when trying many combinations of models, pre-processing techniques, feature extraction methods and features reduction approaches.

Implementing every phase of the data stack, engineering, analysis, modeling and deployment helped us to form a strong intuition of the whole pipeline and the problems faced during linking the output of each phase to the other.

Manipulating text based dataset is not a trivial task and it depends on the objective of the project, the resources used and the machine learning models used.

Paraphrasing has tons of applications in the NLP world such as plagiarism detection and many other applications. But as we mentioned, most of its applications are for the detection purpose. Paraphrasing generation has not much intention yet in research but there are many promising deep learning approaches that solve the generation problems such as the ones that are GANs based.

A. Potential Improvements

Text pre-processing and feature extraction techniques are endless and there are many other different approaches that we have not tested such as the BoW (bag of words) and PoS (part of speech tagging).

Paraphrasing generation has a growing trend in research and there are a lot of more efficient yet simpler approaches other than the DRL (deep reinforcement learning) models.

Applying trasnfer learning in the modeling phase with LSTMs or Neural Networks would definitely increase the model performance and can achieve the state of the art results.

References

[1] Aladdin Knowledge Systems, Anti-spam white paper, Retrieved December 28, 2011.

[2] J. D. Brutlag and C. Meek, “Challenges of the email domain for text classification,” in Proc. ICML, 2000, pp. 103–110.

[3] W. W. Cohen, “Learning rules that classify e-mail,” in Proc. AAAI Spring Symp. Mach. Learn. Inf. Access, 1996, p. 25.

[4] I. Androutsopoulos, J. Koutsias, ”An evaluation of naïve bayesian anti-spam filtering”. In Proceedings of the Workshop on Machine Learning in the New Information Age, 11th European Conference on Machine Learning (ECML 2000), pages 9–17, Barcelona, Spain, 2000.

[5] I. Androutsopoulos, G. Paliouras, ”Learning to filter spam E-mail: A comparison of a naïve bayesian and a memory based approach”. In Proceedings of the Workshop on Machine Learning and Textual Information Access, 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD 2000), pages 1– 13, Lyon, France, 2000.

[6] J. Hidalgo, ”Evaluating cost-sensitive unsolicited bulk email categorization”. In Proceedings of SAC-02, 17th ACM Symposium on Applied Computing, pages 615– 620, Madrid, ES, 2002.

[7] K. Schneider, ”A comparison of event models for naïve bayes anti-spam e-mail filtering”. In Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics, 2003.

[8] I. Witten, E. Frank, ”Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations”. Morgan Kaufmann, 2000.

[9] C. Miller, ”Neural Network-based Antispam Heuristics” , Symantec Enterprise Security (2011), www.symantec.com Retrieved December 28, 2011

[10] Du Toit, J. V., de Waal, D. A., 2010. Spam Detection using Generalized Additive Neural Networks. SATNAC 2010 Conference Proceedings.

[11] G. Mujtaba, L. Shuib, R. G. Raj, N. Majeed and M. A. Al-Garadi, ”Email Classification Research Trends: Review and Open Issues,” in IEEE Access, vol. 5, pp. 9044-9064, 2017, doi: 10.1109/ACCESS.2017.2702187.

[12] V. Christina, S. Karpagavalli, G. SuganyaEmail spam filtering using supervised machine learning techniques Int. J. Comput. Sci. Eng., 02 (09) (2010), pp. 3126-3129

[13] T. Subramaniam, H.A. Jalab, A.Y. TaqaOverview of textual anti-spam filtering techniques Int. J. Phys. Sci., 5 (12) (2010), pp. 1869-1882

[14] M.F. PorterAn algorithm for suffix stripping Program: Electron. Libr. Inf. Syst., 14 (3) (1980), pp. 130-137

[15] T.A. Almeida, A. YamakamiContent-based spam filtering The 2010 International Joint Conference on Neural Networks (IJCNN), Barcelona (2010), pp. 1-7

[16] G. Sakkis, I. Androutsopoulos, G. Paliouras, V. KarkaletsisStacking classifiers for anti-spam filtering of E-mail Empirical Methods in Natural Language Processing (2001), pp. 44-50

[17] A. Attar, R.M. Rad, R.E. AtaniA survey of image spamming and filtering techniques Artif. Intell. Rev., 40 (1) (2011), pp. 71-105

[18] J.R. Mendez, F. Díaz, E.L. Iglesias, J.M. CorchadoA comparative performance study of feature selection methods for the anti-spam filtering domain Advances in Data Mining. Applications in Medicine, Web Mining, Marketing, Image and Signal Mining, Springer Berlin Heidelberg (2006), pp. 106-120

[19] Z.S. Torabi, M.H. Nadimi-Shahraki, A. NabiollahiEfficient support vector machines for spam detection: a survey. (IJCSIS) Int. J. Comput. Sci. Inf. Secur., 13 (1) (2015), pp. 11-28

[20] L. Pelletier, J. Almhana, V. ChoulakianAdaptive filtering of spam Second Annual Conference on Communication Networks and Services Research (CNSR'04) (2004)

[21] M. Balakumar and V. Vaidehi, Ontology Based Classification and Categorization of Email. New York, NY, USA: IEEE Press, 2008.

[22] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, Machine Learning: An Artificial Intelligence Approach. New York, NY, USA: Springer, 2013.

[23] M. Alian and A. Awajan, ”Paraphrasing Identification Techniques in English and Arabic Texts,” 2020 11th International Conference on Information and Communication Systems (ICICS), Irbid, Jordan, 2020, pp. 155-160, doi: 10.1109/ICICS49469.2020.239485.

[24] Analysis of Paraphrase Detection using NLP Techniques

[25] Sumit Chopra, Michael Auli, and Alexander M Rush. 2016. Abstractive sentence summarization with attentive recurrent neural networks. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 93–98.

[26] Ronald J Williams. 1992. Simple statistical gradientfollowing algorithms for connectionist reinforcement learning. Machine learning, 8(3-4):229–256.

[27] Wuwei Lan, Siyu Qiu, Hua He, and Wei Xu. 2017. A continuously growing dataset of sentential paraphrases. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing,

pages 1224–1234, Copenhagen, Denmark. Association for Computational Linguistics.
[28] Zichao Li, Xin Jiang, Lifeng Shang, and Hang Li. 2018. Paraphrase generation with deep reinforcement learning. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing.
[29] Kishore Papineni, Salim Roukos, Todd Ward, and WeiJing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In Proceedings of the 40th annual meeting on association for computational linguistics, pages 311–318. Association for Computational Linguistics
[30] Zhang, Z. and Sabuncu, M. Generalized cross entropy loss for training deep neural networks with noisy labels. NeurIPS, 2018.
[31] W. Li, W. Meng, Z. Tan, and Y. Xiang, "Towards designing an email classification system using multi-view based semi-supervised learning," in Proc. 13th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom), Sep. 2015, pp. 174–181.
[32] W. Li and W. Meng, "An empirical study on email classification using supervised machine learning in real environments," in Proc. IEEE Int. Conf. Commun. (ICC), Jun. 2015, pp. 7438–7443.
[33] M. R. Schmid, F. Iqbal, and B. C. M. Fung, "E-mail authorship attribution using customized associative classification," Digit. Investigat., vol. 14, pp. S116–S126, Aug. 2015.
[34] J. R. Mendez, M. Reboiro-Jato, F. Diaz, E. Diaz, and F. Fdez-Riverola, "Grindstone4Spam: An optimization toolkit for boosting e-mail classification," J. Syst. Softw., vol. 85, pp. 2909–2920, Dec. 2012.
[35] M. T. Banday and S. A. Sheikh, "Multilingual e-mail classification using Bayesian filtering and language translation," in Proc. Int. Conf. Contemp. Comput. Informat., 2015, pp. 696–701.
[36] Sarveniazi, A. (2014) An Actual Survey of Dimensionality Reduction. American Journal of Computa-tional Mathematics, 4, 55-72.
[37] Camastra, F.. "Data dimensionality estimation methods: a survey." Pattern Recognit. 36 (2003): 2945-2954.
[38] Fisher, R. A. (1936). "The Use of Multiple Measurements in Taxonomic Problems". Annals of Eugenics. 7 (2): 179–188