Email Spam Filter based on Machine learning Methods

CISC 372 Project
Date: Apr, 19, 2021

MINGJIA MAO (17mm78@queensu.ca)
SIZHE GUAN (17sg46@queensu.ca)
ZHENGHAN TAI (17zt12@queensu.ca)

Abstract

This report presents multiple approaches to implement an email spam filter, based on machine learning methods. The investigation includes five learning algorithms, Naive Bayes, KNN classifier, Support Vector Machines, Convolutional Neural Network, and the Recurrent Neural Network with GRU. We would explore on how to implement the models and evaluate the accuracy and effectiveness of the models. We studied how to pre-process the emails written in plain English, convert the word into numerical representations, as well as some word embedding methods.

Key-words: email classification, spam, machine learning, text processing

Introduction & Overview

In recent years, the increasing popularity of the individual utilization of emails and the low-cost advertising through this way, have attract the commercial marketers to send massive amount unsolicited messages to mailboxes of users, for which include product advertisements, pornography, illegitimate contents or unwanted information of any kinds, preventing us from making full and good use of time, these messages are usually referred to as 'Spam' mails, or more formally, Unsolicited Commercial E-mail (UCE), the term 'spam' stands for the 'irrelevant or inappropriate messages sent on the internet to a large number of recipients', the word is then used to refer to the mails that are sent in bulk, which would lead to disruptions and inefficiencies of the Internet services and communication between the users, imagine if your mailbox is full of junk spam messages while you are trying to find the specific one from the professor. Therefore, a solution to the spam problem is really important. Several counter-measures were proposed, for example, handcrafted rules, blacklists, both of them seem to be of little use as the rules need to be tuned manually which has very low efficiency while blacklists are also ineffective due to the forged sending address. Then what would be the optimal solution to this increasingly serious problem? [1] Our proposal and contribution to the problem is to implement learning models and instance-based methods to build a classifier that is able to help us detect and filter the spam emails automatically, for the methods that is relatively simple, we applied the ensemble method to get a vote from multiple models in order to get a result of low variance and low bias. For the methods that involves neural network, we need to make sure it does not overfit the training set. According to the papers online, we would probably be the first that implement RNN with pre-trained word embedding methods (GloVe), 1D-CNN and bagging of instance-based methods on the dataset that we selected, the final decision is based on the accuracy of each method.

The challenges would be finding out the optimal dataset, there are several spam datasets available online but they are either unprocessed or with different distributions, and one crucial and significant principle is: what we use as the input for training should be approximately the same as what the model will eventually encounter in the future, in the other words, the distributions of 'ham' and 'spam' should meet the situation in the real word. Therefore, we need to first explore on the global e-mail spam rate over the past few years. Another one might be the choose of models and how to actually implement these models based on what we learned from the course, accompanied with further reading from online materials. The other challenge would be how to pre-process the email contents from the online dataset, remove the punctuations, hyperlink, and any other sort redundant

information, and how to embed/vectorize the words with an efficient way.

Problem Statement

Our problems are, what would be the optimal solution to the detect the spam emails? Which learning algorithm should be implemented so that the accuracy is the highest? And is there any constrains or limit to each algorithm? Do we need to consider the feasibility of each implementation? For example, the complexity of training and the time requirement. How to pre-process the contents of the dataset and how to realize the word representation in the form that can be the input of the algorithms? Does the downstream natural language processing/pre-trained word embedding improve the performance of the deep learning architectures?

Solutions (with experiment results)

Generally, the optimal solution would be the use of machine learning methods, which showed the success in text categorization [Sebastiani 2002], in simple terms, they are supervised learning algorithms for which fed with the documents that are pre-labelled with 'spam' or 'ham', ultimately building an effective classifier based on that, the term 'ham' here stands for the legitimate messages with the label 0 while 'spam' has a label of 1. However, there are lots of alternatives with different limitations, we have to identify and choose the optimal learning method. Besides that, it is also important to realize the numerical representation of the email contents, so that the data would be the valid input of different learning methods.

Data set selection

Before we can do any exploration on our problem, it is crucial to find datasets for our models, we use the data from Enron Spam Dataset, which include six datasets in .tar.gz format, the six datasets were subjected to slight and limited pre-processing by removing the addresses of the sender and owner, for example, the information after the tag 'To:', 'Cc' and 'Bcc', and some other headers. In most situations, more data is usually better, we were told the training size of 5000 is too small during the video meeting, we then merge the six sets together so that there were 36609 entries which is rather enough, the empty entries were simply dropped as it would not have any impact to a binary classification problem. Next, we need to make sure the dataset is 'representative', just like I mentioned in the introduction, the training set should have similar characteristics to the unforeseen data, namely, we should expect the training set have approximately the same spam mail percentage as that in the real-world condition, therefore, we derive the statistic from the website statista (see GitHub repository), and choose the average global spam volume as percentage of total e-mail traffic over the past 5 years, from 2015-2019, which is 45%, we then drop some of the spam entries to make sure the distribution of 'spam' and 'ham' strictly follow the real-world situation. 20% of the data is sampled as testing set, without replacement, the remaining is the training set, both train and testing set follow the real-world 'spam' &'ham' distribution by using the Sklearn API train_test_split() with the argument stratify=x['label']. We also performed shuffle on the data frame in order to avoid the cluster of entries with the same label. [2]

Word representations and pre-processing

We then have to consider how to convert the literal email contents to numerical representation that is trainable data, as features in machine learning is numbers that we can perform mathematical operations on. Before that, we need to first remove the noisy/redundant data in each sentence, revealed in our code. We used the regular expression library to remove hyperlinks (strings start with 'http'), punctuations (from string library), whitespace, numbers (\d+ -> [0-9]) and etc. Next, we converted the processed sentences into tokens, and performed Lemmatization, [3]taking into consideration of the morphological analysis of each word, as well as removing stop words. We then applied two different methods of vectorizations, one is bag of words, along with TF-IDF, another is word embedding, self-trained embedding layers and pre-trained embedding models (GloVe). The reason is that word embedding layer can only be applied to neural networks, while we also proposed 3 classification methods with no use of ANN. Both methods would be discussed in details later on.

First groups of Classifiers

We first implement 3 relatively simpler learning algorithms, with a hard-vote bagging in the end to lower the bias and higher the reliability.

Naïve Bayes

The first method is naive Bayes classifiers based on Bayesian Theorem (P(Y|X) = $\frac{P(X|Y)*P(Y)}{P(X)}$, X can be treated as features, and Y can be treated as target labels. The Bayesian Theorem now becomes:

*P (Belongs to Label L| Given Features) = P (Given Features| Belongs to Label L) * P(Belongs to Label L) over P(Given Features).*

By computing the given probability, Naïve Bayes method check if the output probability is greater than or equal to 0.5 to do a binary classification. If tokenized words in a given content can be treated as a feature, and assuming all the features are independent, The Bayesian Theorem can be written as the product of probabilities of each tokenized word. Here in this case, Spam or Ham emails have two possible outcome labels, where 1 is spam and 0 is non-spam, which can be solved by a simple binary Naïve Bayesian model.

KNN Classifier:

In this case, the learning algorithm would be Memory-based, or "instance-based", for which does not construct a unique model for each category, but simply store the training examples (Aha et al. 1991, Wilson 1997).

The training data points are scatters through a Rn span for n features, and based on Euclidean distance, |AB| =$\sqrt{(x^1 - x^2)^2 + (y^1 - y^2)^2}$, the distance between all training data set and the new input data will be calculated and stored in descending order, the first Kth smallest training data points are selected and the most frequent label in the first Kth smallest data points are used to predict the label of new input data. Geometrically, it can be interpreted as finding the label of the K nearest Neighbor of new input data point.

SVM Classifier:

Support Vector machine treats the data as sets $(\vec{x_i}, y_i)$ where x are vectors, y are the labels and y are either 1 or -1. Support vector machines computes a hyper plane that separates y. The hyperplane can be written as $w^{\rightarrow} * x^{\rightarrow} - b = 0$ where w is a normal vector. Support vector machine predicts the label of new input data based on the sign of dot product of vectorized data point and the hyperplane.
However, when dealing with nonlinear separable data, simple support vector machine model may have poor performance. Thus, additions like kernel method can be introduced. Geometrically, the kernel function projects the data points into a higher dimension so that the data points become linear separable in that span and after applying the simple support vector machine on a kernel preprocessed dataset, it projects the hyperplane back to its original span to produce higher accuracy.

Generally, the 'Bag of word' method would compute frequency of words in a single content over the vocab list and store the frequencies as a row vector. Thus, a message (Email content) can be presented as a row vector, the index of an entry represents the position of word in the vocabulary list, and the value of entry represents the number of times a word appeared in this message. For Spam E-mail binary classification, it would, firstly, build a vocab list including all vocabs appeared in the training data set. Secondly, for each of the email that is used for training, count the frequency of words appeared in the vocab list and store the frequency into a vector, where the index represents the position of word in vocab list and the key value represents the frequency of the word. Combine the frequency vectors into a sparse matrix. Thirdly, apply a L2 normalization to the sparse matrix where words are weighted. High frequent words such as 'Like', 'Say', 'Is' may have lower weight since they are less important. Term Frequency Inversed Document Frequency is introduced to produce the normalized weight of words. First it computes the normalized term frequency, mathematically can be written as *(# of target in a content)/ (the size of vocab size of the content)* and then by computing *log (total number of contents/ # of content containing the target)*, it gives a weight representing how important a word is. By multiplying the Term Frequency and the Inverse Document Frequency, it gives the normalized importance vector of a word. The dataset is good to pass into the previous learning methods.

Results:

On our first trial, full dataset is used, and the Accuracy on test set for NB, KNN and SVM are shown in Figure 1.1, 1.2 and 1.3

Figure 1.1 Precision and Confusion Matrix for NB

	precision	recall	f1-score	support
0	0.99	0.94	0.97	15951
1	0.93	0.99	0.96	11738
accuracy			0.96	27689
macro avg	0.96	0.97	0.96	27689
weighted avg	0.97	0.96	0.96	27689

```
[[15028    96]
 [  923 11642]]
```

Figure 1.2 Precision and Confusion Matrix for KNN

	precision	recall	f1-score	support
0	0.18	1.00	0.30	2653
1	1.00	0.50	0.67	25036
accuracy			0.55	27689
macro avg	0.59	0.75	0.48	27689
weighted avg	0.92	0.55	0.63	27689

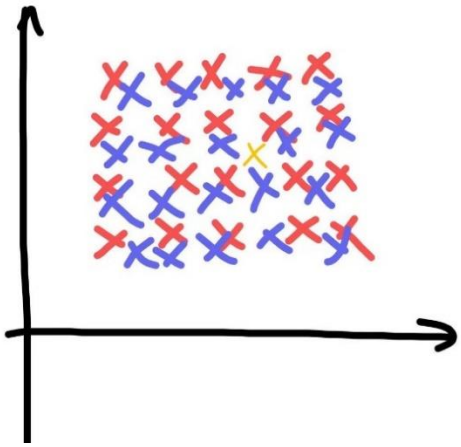
```
[[ 2653 12471]
 [    0 12565]]
```

Figure 1.3 Precision and Confusion Matrix for SVM:

	precision	recall	f1-score	support
0	0.97	0.99	0.98	14793
1	0.99	0.96	0.98	12696
accuracy			0.98	27489
macro avg	0.98	0.98	0.98	27489
weighted avg	0.98	0.98	0.98	27489

```
[[14673   451]
 [  120 12245]]
```

You would find out that accuracy on testing set for KNN model is rather low, that is because of following possible reason: The training data set is way too large so that the data is randomly spread and the data points are jumbled thus when the algorithm tries to find the K nearest neighbors, the frequency of the two labels might be about 1:1. Then the accuracy will become rather poor.
Here is a graph sample to help better explain:
Figure 1.4: KNN fail to make prediction



Thus, we tried to shrink the train data into a much smaller size to see if there would be any change in the performance of each model, we want to know if the size of the training data would have a huge impact to models. In this case the new train data size is about 500, and train three new models on the new train set. The performance of the new three models based on the re-sampled test set is as following:

Figure 2.1 Precision and Confusion Matrix for new NB model

	precision	recall	f1-score	support
0	0.99	0.94	0.97	15951
1	0.93	0.99	0.96	11738
accuracy			0.96	27689
macro avg	0.96	0.97	0.96	27689
weighted avg	0.97	0.96	0.96	27689

```
[[15028    96]
 [  923 11642]]
```

Figure 2.2 Precision and Confusion Matrix for new KNN model

	precision	recall	f1-score	support
0	0.92	0.94	0.93	14807
1	0.93	0.91	0.92	12882
accuracy			0.93	27689
macro avg	0.93	0.93	0.93	27689
weighted avg	0.93	0.93	0.93	27689

```
[[13965  1159]
 [  842 11723]]
```

Figure 2.3 Precision and Confusion Matrix for new SVM model

	precision	recall	f1-score	support
0	0.88	0.99	0.93	13548
1	0.98	0.87	0.93	14141
accuracy			0.93	27689
macro avg	0.93	0.93	0.93	27689
weighted avg	0.93	0.93	0.93	27689

```
[[13346  1778]
 [  202 12363]]
```

The performance of KNN model improved significantly and the performance of the other two models has slightly decreased.

To lower the bias of these 3 models, the concept of aggregating three of them is introduced and let the final prediction be the hard voting result of the three models, means the result is derived based on the label with more votes.

The performance of the new bagging model is as follows, note that KNN here is trained with only 500 entries while others use the fall set.

Figure 3.1 Precision and confusion matrix of aggregating all three Instance based learning models:

```
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.99      1136
         1.0       0.99      0.98      0.99      1243

    accuracy                           0.99      2379
   macro avg       0.99      0.99      0.99      2379
weighted avg       0.99      0.99      0.99      2379

[[1121   19]
 [  15 1224]]
0.9847144006436042
```

To summarize, performance of NB and SVM model has no significant change with vary of train data size, the models are very easy to implement, and the generalizability for these two models are good, so they can be used for both by personal and public needs. However, the feasibility of KNN model is very poor. In the real world, the amount and style of spam emails will increase over time and so does the training process, KNN models only has a fair performance on small training size but very poor performance once the training data gets large, KNN would not be a reliable classifier **alone**, and it is sensitive to the size of incoming data. See that Naïve Bayes has a very high accuracy of predicting unknown data. Thus, Naïve Bayes model has the best performance with a good generalizability. By combining the result of the three models, the overall accuracy score on the test data set is 0.984 which is very well performed.

Second groups of Solution: **ANN**

We then tried to build classifiers based on artificial neural networks, by training and refining the models, we are expecting a optimal architecture that is able to predict the label of emails with higher accuracy than any one of the 3 methods in the previous section.

Firstly, we need to convert the literal word into numerical representation by taking the use of tf.Tokenizer(), basically, each word would be assigned with an index based on its frequency in TF-IDF format, meanwhile we set the number of words, which refer to the size of the corpus/dictionary to 100000 (variable 'max_feature'), length of each sentence to 1200 (variable 'max_len'), thus, each sentence would become a sequence of integers, with a max length of 1200 integers, and padded with '0's if the original length is smaller than 1200, so that this would be a valid numerical input. Secondly, the **embedding layer** is introduced[4], compare to the classical one-hot representation, embedding vectorization to a word would result a vector with much lower dimensionality, for example, in one-hot, the vector would have a size of 10000 (features) for a vocabulary of 10000 words, meanwhile most of the value within the vector is 0, namely, the word vector is really sparse with high dimensionality, this would lead to the phenomena named 'Curse of dimensionality' [5], which lead great difficulty of extracting the relation and finding the optimal model, since both search space and the mode complexity increases a lot. Basically, the embedding layer would randomly initialize a word matrix of [NxM], N is the size of the dictionary(# of words in total), M is the size of the word vector,

specified with the variable ‘embed_size’, the original one-hot representation [1xN] * [NxM] yields a vector of [1xM], greatly lower the dimensionality. During the training/learning process of the neural network, the embedding layer would be updated/trained to minimize the loss. With this method, the subtle relationship between words in the domain that we are studying would be learned and revealed through the word embedding, for example, if two words are 'king' and 'queen', they may be 'close' to each other according to the trained word embedding. By applying this method, we would get semantically-meaningful **dense** real-valued vectors, and boost the process and improve the performance of the NN.

Convolutional Neural Network

Here we tried to use CNN to build the classifier. This may sound weird as CNN is usually used for visual image processing, and what we learned on the course was 2D-CNN, instead of 2-dimensional, texts have a one-dimensional structure where words sequence matter. Based on finding of 'Yoon Kim'[6], 1D CNN would simulate the process of n-gram model, in this case, the kernel would have the same size as the ‘embed_size’, e.g., same size as the word embedding vector, in our code, this number is 100, means the kernel has a width of 100, while for the height, Kim provides a use of different size simultaneously, in our model, we simulates 2/3/4-gram with the corresponding heights of 2,3,4, scanning multiple words at the same time, the kernel sizes therefore, are 2/3/4 x 100, and output a number for each scanning procss, the kernel would move one-by-one down a list of input embeddings, each time cover a window with the shape of 2/3/4 x 100 and produce one number based on the weight within the kernel, and eventually form a vector, the vector is then pooled with the largest number with GlobalMaxPooling1D(), each convolutional layer has 100 kernels means 100 such vectors would be produced for each layer, with the pooling process, instead of one convolutional layer connects with another convolutional layer, the 4 convolutional layers with different size of kernels would scan and produce the output at the same time, at last we concatenate the outputs of the GlobalMaxPooling1D layers (also 4, corresponding to each convolutional layer) and pass to a dense layer, totally 300 derived by the MaxPooling1D, finally reach the output layer with a sigmoid function for binary classification.

Figure 1 from Kim’s paper

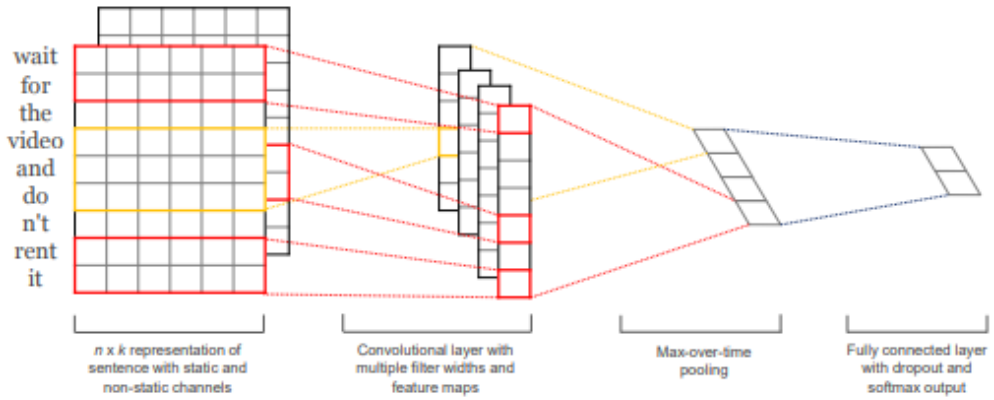
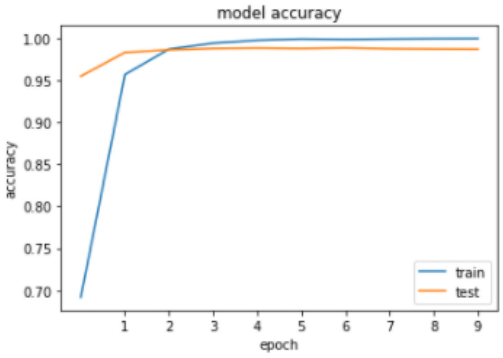
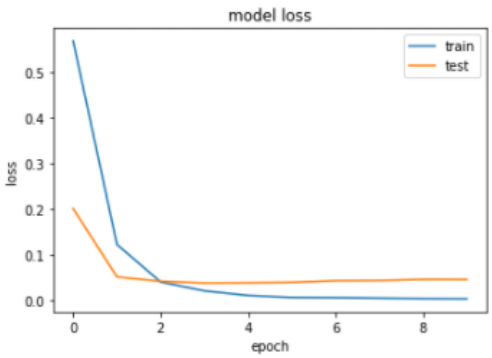


Figure 1: Model architecture with two channels for an example sentence.

The kernels with different size at the same time would be able to extract the relation between phrases

with different lengths, from 2-4 The upper bound is 4 is because words that are farther away than 4 would be less relevant to identify the patterns, for example, a phrase usually has the length around 4, for example, the phrase 'a delicious pie'. In additional, at each pooling layer, I dropped 60% of the output to avoid overfitting, this would be useful. By implementing a CNN with this trend, we got a really high accuracy, according to the testing set, the complexity of the model did not lead to overfit which is so good for us, as mentioned before, the word embedding would also be trained and updated, means there are so many parameters, along with the weight in each kernel of the convolutional layer, the model has relatively high complexity which is able to extract and identify the pattern with the length of 2/3/4. And the model showed feasibility and generalizability because neural networks are trained before taking into actual use, therefore we got plenty enough time to train the model and tune the hyper-parameters based on the data analysist's life cycle, in order to achieve the highest accuracy as possible, after all the process is done then take the model into real-world use.
The graphs below shows the accuracy ascending and loss descending in each epoch.





The confusion matrix, precision, recall and the f1-score based on the former statistic are shown below. We can see the f1_score is 0.9855 which is better than all of the 3 models in the previous section. We conclude CNN as a feasible way of implementing effective email-spam filter.

f1-score	Precision	Recall
0.9855	98.17%	98.94%

It is also noticeable that after the first epoch, it has sharp increase in accuracy, which can be seen from the history and graph (accuracy with respect to epochs), it has a high predicting accuracy on the test set (98%). However, if I setting trainable = False, it would have a relatively gradual increase of accuracy, while the overall predicting accuracy on the test set is relatively lower (95%), I did not attach any picture here as it is not a usually way, the reason of this phenomenon would be: if we don't train

the word embedding vectors, in this case, the vectors are randomly initialized based on the source code, then we just focus on updating the weight parameters within the filters/kernels of the CNN, therefore yield a smooth graph, however, if we introduce the concept of the relationship between words, then after the first epoch, the word embedding vectors are updated and the relation is derived, together with the updated filter parameters yield the higher accuracy.

Recurrent Neural Network

Initially, spam detection relied on the Naive Bayes approach and Naive Bayes is regarded as the baseline for dealing with spam. However following breakthroughs in deep learning, researchers have now turned their focus to neural networks to help them deal with the perennial problem of spam emails. Google recently reported that by introducing Neural Network's to g-mail's spam filters. It took them from 99.5% to over 99.9% accuracy, suggesting that neural networks especially when used in conjunction with Bayesian classification may be effective for enhancing spam filters [7]. We choose Bidirectional LSTMs and GRU layers as two main components of the RNN model. Implementing Bidirectional layer is because for some of the sentences, the context information is at the end of the sentence. Without the context info, ambiguity might arise. The use of providing the sequence bi-directionally was initially justified in the domain of speech recognition because there is evidence that the context of the whole utterance is used to interpret what is being said rather than a linear interpretation [8]. For example, as a verb, "project" means something is planned or expected. In another situation, “project” can also be used when you project someone or something in a particular way, you try to make people see them in that way. Here are two sentences:

A: The experts project a 5% consumer price increase for the entire year.

B: He just hasn't been able to project himself as the strong leader.

The first LSTM network will feed in the input sequence as per normal. And this sequence will be starting from the beginning of sentences and ending at the last words of sentences. For the two sentences above, the ambiguity will arise when training the model. So, reading the sentence in bi-direction helps the model to determine the exact meaning of the word.

In the aspect of preventing Gradient Explosion problem, we use LSTM instead of simple RNN model. LSTMs solve the problem using a unique additive gradient structure that includes direct access to the forget gate’s activations, enabling the network to encourage desired behavior from the error gradient using frequent gates update on every time step of the learning process[9]. While in the project we use GRU since it produces the similar performance with LSTM and it is more efficient than LSTM.

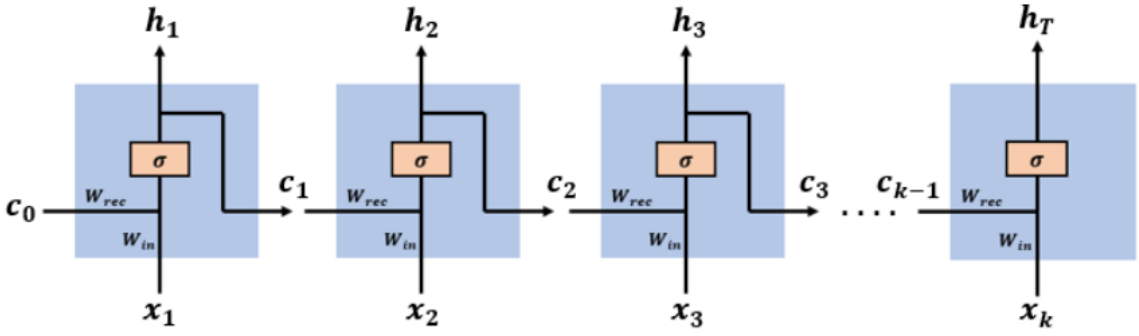


Figure: LSTM network

Then we start building the model and model involves components such as Embedding Layer, Bidirectional Layer and Gated Recurrent Units (GRU). So, in this model, we try the Embedding layer in two ways: [10]

1. RNN with self-trained word embedding;
2. RNN with pre-trained word embedding

First, we will need to make use of tokenizer to help us convert all the words to indexes.In the tokenization layer, they maintain a dictionary of unique words that map each word to an index. For example, dog -> 0, cat->1, and so on. The functions includes tokenizer "fit_on_texts()" and "texts_to_sequences()". Fit_on_texts() creates the vocabulary index based on word frequency. Its output is a dictionary with default name word_index. texts_to_sequences() transforms each text in texts to a sequence of integers. So it basically takes each word in the text and replaces it with its corresponding integer value from the word_index dictionary.

Then, the embedding layer has two ways to implement, the first is the self-trained word-embedding mentioned in the previous section, the other way refers to the pre-trained word embedding requires the data from other developers. There are three main sources which are Glove, Wiki-news and word2vec from Google. Our model implementing the Glove pre-trained word vectors. Glove provides 4 versions of word vectors that are 50-dimensional vectors, 100-dimensional vectors, 200-dimensional vectors and 300-dimensional vectors. Here we choose the 300d version and then transfer weight from 300-dimensional vectors to Keras embedding layer. The first step we take is loading the entire GloVe word embedding file into memory as a dictionary of word to embedding array. Next, create a matrix of one embedding for each word in the training dataset. We can do that by enumerating all unique words in the Tokenizer.word_index and locating the embedding weight vector from the loaded GloVe embedding. The result is a matrix of weights only for words we will see during training[11]. Now, the model can fit as well as it works with self-trained word embedding. In conclusion, the key difference is that the embedding layer can be seeded with the GloVe word embedding weights. We chose the 300-dimensional version; therefore, the Embedding layer must be defined with output_dim set to 300. Finally, we do not want to update the learned word weights in this model, therefore we will set the trainable attribute for the model to be False.

Next, the GRU layer is wrapped by Bidirectional layer which merge_mode is default. It means that the GRU layer running forwards and backwards produces two 2000 timestamps of 64 outputs and Bidirectional layer concatenates them to produce 100 timestamps of 128 (64 units + 64 units) outputs. The pooling layer receives the 2000 timestamps of 128 units outputs and takes the max vector over the steps dimension. So, in this model it takes input (None, 2000, 128) and produce (None,128) output. After pooling layer, there is a normal dense layer but the dropout layer behind dense layer is very important. In this model, it drops 20% percent of units. So, the model will not depend on partial features of units and be more general.

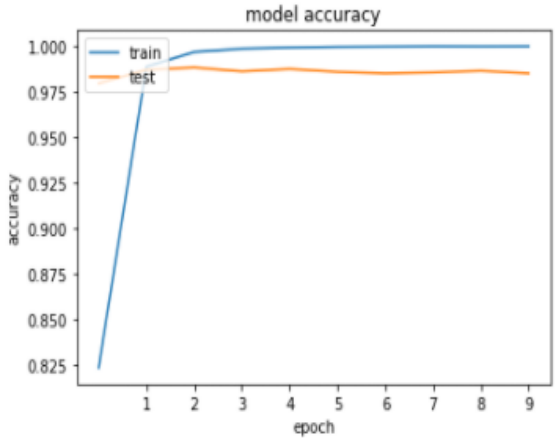


Figure1: RNN with self-trained word embedding

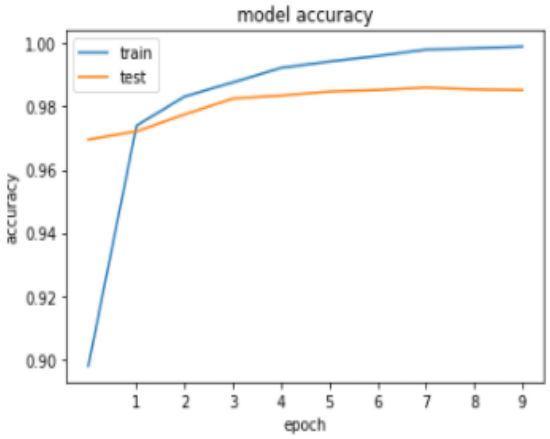


Figure2: RNN with pre-trained word embedding

The precision and recall of the two models do not differ by a very great margin. Some of the reason might be due to: Glove embedding is trained over sources that are very different from the data that we have in this problem, so the benefits we gain from the pre-trained embedding is not significantly helpful. Or The text data in this spam filtering problem is not too complicated. RNN with self-trained word embedding is a good enough model to capture the pattern.

In the graphs, RNN with self-trained word embedding reach a higher accuracy than RNN with pre-trained word embedding in the same time (one epoch). So, it proves that Glove embedding does not provide any helpful data. And self-trained word embedding is wonderful enough to handle the input dataset.

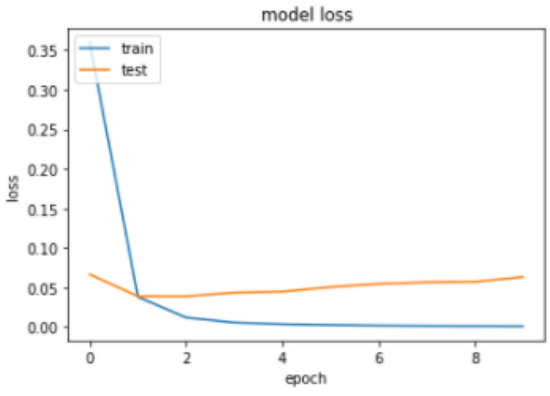


Figure1: RNN with self-trained word embedding

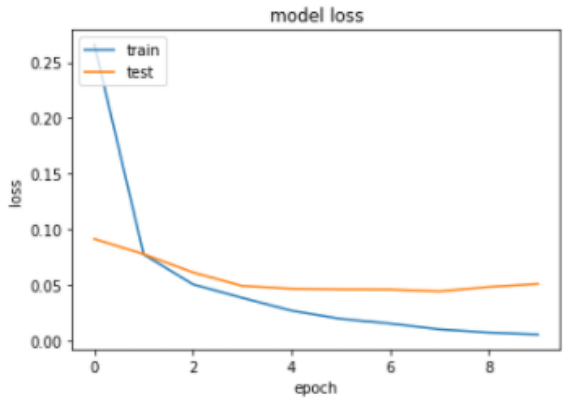


Figure2: RNN with pre-trained word embedding

The loss graphs show that we might need to stop training after 1 epoch because the loss has been increasing instead of decreasing.

	f1-score	**Precision**	**Recall**
Self-trained	0.9834	98.63%	98.05%

Pre-trained	0.9838	98.06%	98.71%

The slighly difference in Precision shows that Self-trained word embedding does better on finding the spam emails correctly. While in recall session, pre-trained word embedding does a good job in discovering the spam email. Therefore, the f1_score shows that two model are the same level and providing very well performance.

Conclusion

In conclusion, we tried 6 models to simulate the spam filtering system. While the models were spitted into two main groups. One group are instance-based learning models which refer to K-nearest neighbor method, others include Naive Bayes method, support vector machine. Another one is the implementation of neural networks containing two Recurrent Neural Networks and one Convolutional Neural Network architecture. The reason of division into 2 groups is: the first 3 have certain limitation and they're relatively easy to realize, the bagging of these three would have the optimal prediction with lower bias and variance. Based on the result of each model, the overall performances of neural networks are generally greater than first group of learning algorithm in the aspect of accuracy. The reasons might be TFIDF and Bag of words Vectorizers did not take into account of the ordering of the words in the sentence, and it is losing a great deal of information, especially the pattern of a sequence of phrases. Besides that, it may suffer from 'curse of dimensionality' as the vectors are sparse and contain a lot of '0's with redundant information, such input would cause lower efficiency and feasibility of the learning algorithm. It is noticeable that KNN is not able to handle dataset with relatively larger size, the larger the input, the poorer it would perform, therefore KNN should not considered or at least using ensemble method with the hard voting with the other 2 models, SVM and NB. Both SVM and Naïve Bayes are not that sensitive the input size, and both them perform good under both larger and small datasets, especially for Naïve Bayes, as a generative model, it tries to find out how the data was generated in the first place, the underlying distribution that produced the examples you input to the model, even small training set would yield good predictions on a much larger testing set. Therefore, NB has a high generalizability when you have limited training data, besides it is the easiest way to implement. In terms of the ANN methods, all of them requires large amount of computing resource to train and refine the model, however we can train the model in prior and take the incoming email as the input to predict its category. CNN with embedding showed the highest accuracy (f1-score) therefore we consider it as the optimal solution. For RNN with bi-directional GRU, the experiment result between applying pre-trained method or not showed that, GloVe word embedding did not improve the performance of the neural network, instead it may influence precision of the outcome, it may be just not compatible with the domain that we are studying, for example, using word embedding shown the relationship between weight to predict height. Overall, CNN that simulates the n-gram model would be the optimal solution with the highest accuracy while Naïve Bayes would be the most generalizable model that requires a small size of input but perform well, accompanied with ease to implement.

Reference

"Email Spam Filtering: An Implementation with Python and Scikit-learn." *KDnuggets*. Web. 22 Apr. 2021. <https://www.kdnuggets.com/2017/03/email-spam-filtering-an-implementation-with-python-and-scikit-learn.html>.

[1]: SAKKIS, GEORGIOS. "A Memory-Based Approach to Anti-Spam Filtering for Mailing Lists." Web. <http://www2.aueb.gr/users/ion/docs/ir_memory_based_antispam_filtering.pdf>.

[2]: "Natural Language Processing: Applications¶." *15. Natural Language Processing: Applications - Dive into Deep Learning 0.16.2 Documentation*. Web. 22 Apr. 2021.<https://d2l.ai/chapter_natural-language-processing-applications/index.html>

[3]: Bitext. "What Is the Difference between Stemming and Lemmatization?" *What Is the Difference between Stemming and Lemmatization?* Web. 22 Apr. 2021. <https://blog.bitext.com/what-is-the-difference-between-stemming-and-lemmatization/>.

[4]: Latysheva, Natasha. "Why Do We Use Word Embeddings in NLP?" *Medium*. Towards Data Science, 10 Sept. 2019. Web. 22 Apr. 2021.

[5]: Pantola, Paritosh. "Curse of Dimensionality." *Medium*. Medium, 10 May 2018. Web. 22 Apr. 2021. <https://medium.com/@paritosh_30025/curse-of-dimensionality-f4edb3efa6ec>.

[6]: Kim, Yoon. *Convolutional Neural Networks for Sentence Classification*. Rep. New York: Yoon Kim, 2014. Print.Web.3 Spetember 2014 <https://arxiv.org/pdf/1408.5882.pdf>

[7]: HosseinHossein 17911 Silver Badge77 Bronze Badges, Seth SimbaSeth Simba 1, and Douglas DaseecoDouglas Daseeco 7. "Spam Detection Using Recurrent Neural Networks." *Artificial Intelligence Stack Exchange*. 01 Mar. 1966. Web. 22 Apr. 2021.

<https://ai.stackexchange.com/questions/3472/spam-detection-using-recurrent-neural-networks>.

[8]: Brownlee, Jason. "How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras." *Machine Learning Mastery*. 17 Jan. 2021. Web. 22 Apr. 2021. <https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/>.

[9]: Arbel, Nir. "How LSTM Networks Solve the Problem of Vanishing Gradients." *Medium*. DataDrivenInvestor, 16 May 2020. Web. 22 Apr. 2021. <https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577>.

[10]: Farahmand, Meghdad. "Pre-trained Word Embeddings or Embedding Layer: A Dilemma." *Medium*. Towards Data Science, 28 Feb. 2021. Web. 22 Apr. 2021. <https://towardsdatascience.com/pre-trained-word-embeddings-or-embedding-layer-a-dilemma-8406959fd76c>.

[11]: Brownlee, Jason. "How to Use Word Embedding Layers for Deep Learning with Keras." *Machine Learning Mastery*. 01 Feb. 2021. Web. 22 Apr. 2021. <https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/>.