Short Article

Combining Content and Social Features in a Deep Learning Approach to Vietnamese Email Prioritization

Nguyen Thanh Ha[1], Dang Dinh Quan[2], Tran Quang Anh[3]

[1] Hanoi Department of Information and Communications, Vietnam
[2] Hanoi University, Vietnam
[3] Posts and Telecommunications Institute of Technology, Vietnam

Correspondence: Nguyen Thanh Ha, thanhha140589@gmail.com
Communication: received 01 August 2021, revised 23 August 2021, accepted 23 August 2021
Online publication: 12 September 2021, Digital Object Identifier: 10.21553/rev-jec.271
The associate editor coordinating the review of this article and recommending it for publication was Prof. Vo Nguyen Quoc Bao.

Abstract– **The email overload problem has been discussed in numerous email-related studies. One of the possible solutions to this problem is email prioritization, which is the act of automatically predicting the importance levels of received emails and sorting the user's inbox accordingly. Several learning-based methods have been proposed to address the email prioritization problem using content features as well as social features. Although these methods have laid the foundation works in this field of study, the reported performance is far from being practical. Recent works on deep neural networks have achieved good results in various tasks. In this paper, the authors propose a novel email prioritization model which incorporates several deep learning techniques and uses a combination of both content features and social features from email data. This method targets Vietnamese emails and is tested against a self-built Vietnamese email corpus. Conducted experiments explored the effects of different model configurations and compared the effectiveness of the new method to that of a previous work.**

Keywords– **Email prioritization, Vietnamese, LSTM, word2vec, social features.**

1 Introduction

The global volume of email usage has been increasing steadily over the last decade. According to reports from Radicati, the number of daily sent/received emails is 306.4 billion in 2020, with an average increase of approximately 4.3% each year. Research has reported that email users are unable to reply all of their daily incoming messages. According to a study on Yahoo Mail data of 2 million users [1], the reply rate drop from 25% for people who receive less than 20 messages per day to as low as 5% for people who receive around 100 emails per day. A nationwide organizational survey about email usage at work in the US [2] reported that office workers receive around 41 legitimate emails per day on average. It is not always feasible to process all emails when there are too many. However, it is possible to optimize the benefits of reading emails by prioritizing the most important ones and the methods for solving this problem are categorized as *email prioritization* methods.

There have been two approaches to solving this problem: classification and regression. The email features used for email prioritization are various forms of content features [3–5], social features [6] or the combination of both [7]. Content features are the texts extracted from the email's subject and body while social features are calculated by building graphs [6] based on the email's sender and receiver address. The algorithms used in the various studies ranged from traditional machine learning models such as Bayesian classifier [6], support vector machine [7], support vector ordinal regression [5], artificial neural network [3, 4] to deep learning models such as multilayer perceptron (MLP) [8], stacked auto-encoders [9], temporal convolution network [10] and Long Short-Term Memory (LSTM) network [11]. The proposed works of classification or regression of emails into 3 or 5 importance levels has seen certain achievements. The reported results serve as a basis for ongoing research in the field of this problem.

In recent years, deep learning [12] has been widely applied to natural language processing tasks. Both feed-forward networks such as CNN and recurrent networks such as LSTM achieved good results for text classification. There are several reasons which lead to deep learning's superior performance. Traditional machine learning methods reply on hand-crafted feature selection to be effective. Deep neural networks can reply on their complex internal structure to automatically learn features from data. Recurrent neural networks have the ability to model ordered sequences, which is similar to how texts in natural languages are formed. Especially, an LSTM network has been reported to be able to learn very long dependencies in temporal data.

Word embedding is a novel technique for vector representation of texts. Each word in a text is represented by a fixed-size, real-valued word vector. Among the methods to generate word embeddings, *word2vec* [13] is the most popular algorithm which also has highest reported performance in many papers. It has the ability

to represent the semantic meaning of words in a natural language. In other words, using word2vec features enables a certain level of text comprehension for a machine learning model. It is intuitive that this characteristic could benefit a task which greatly depends on email's textual content such as email prioritization. Yet there hasn't been published research that used word2vec features for email prioritization.

This paper proposes a five-level email prioritization model which falls into the deep learning category. A novel combination of content features using word2vec embeddings and social features from training data analysis is utilized in the presented method. A deep neural network structure was suggested to accommodate the mixed inputs. This network, whose structure is described later in this paper, makes use of LSTM units among a variety of other neural network elements. The model's performance is demonstrated on a Vietnamese email dataset.

2 Related Works

2.1 Problem Description

The major purpose of email prioritization is to save email user's time by pointing out which received emails are the most important. For instance, the unread messages in a user's inbox can be sorted by the order of decreasing importance. If a user do not have enough time to handle all emails, it is most beneficial to handle a subset of the most important emails.

Before building a machine learning model to predict the priority of emails, it is crucial to clarify how email priorities are determined. An email with high priority will be placed at the top of the inbox, which means it gets the user's attention earlier than emails with lower priority. Some emails are urgent but not important while some are important but not urgent. This leads to a dilemma of whether to put the former above the latter. According to the Eisenhower Matrix [14], an urgent but not important task should be delegated while an important but not urgent task should be planned to be carried out later. It might not take long to delegate an urgent task or to reply to an urgent email, but there could be unfavorable consequences if the message is not handled in time. Therefore, urgency gets heavier weight when labeling emails for the experiments done in this paper. Ultimately, whether by importance or by urgency, it is up to the end user to decide the priority of an email. The authors choose 5 as the number of labels for the experiments in this paper. Those labels are, from the least to the most important: *delete, unimportant read, important read, reply later*, and *reply immediately*. Some studies use 3 labels [3, 4] to model email priority while some others [5, 7] use 5 labels. In general, using more labels makes it more difficult for the user to label his emails.

There is also a trend of personalization in email-related problems. One reason for the emergence of this trend comes from the fact that email is a type of sensitive data which usually contains personal information or business-related information. Secondly, it is not practical/feasible to provide a solution to an email-related problem that works effectively for all user groups. There has yet to be any research that pointed out the common characteristics of email data. Every user's inbox is supposed to have unique characteristics. Therefore, the most reliable clues that can be used to build predictive models for email-related problems for a particular user are his received emails.

2.2 Previous Methods

The adoption of social features for spam detection was first introduced in [6]. In this study, the authors calculated the *clustering coefficient* from email social network to measure the importance of an incoming email message. In 2009, another study [7] followed a similar approach to address the email prioritization problem, combining the textual content and several social network features. These features of social importance are extracted from unlabeled data using clustering algorithms.

Going towards a different approach for email prioritization, a later study [5] investigated ordinal regression's effectiveness against that of the combination of multiple binary classifiers. This work utilized only content features from email messages. Binary classifiers such as the SVMs can be employed in different voting schemes (OVA, OVO and DAG) to predict among more than two categories. It was discovered in [5] that the combination of binary classifiers outperformed ordinal regression on the same dataset. The result of this work partially suggests that email importance levels do not have ordinal relation with each other.

For real-time application of email prioritization, a number of studies ([3], [4]) explored the approach of using weighted keyword rules mechanism of SpamAssassin. Although keyword matching rules are fast to execute, this approach can only be effective if a good rule set can be constructed. In [3], the authors aims to solve the three-level (email-to-read, email-to-delete, email-to-reply) email prioritization problem by first using a learning-based method to generate SpamAssassin rule sets which serve as binary classifiers and then combining these classifiers using different voting mechanisms (OVA, OVO, DAG) to build a multiclass classifier. The work reported in [4] further enhanced the rule generation part by introducing more features (ham rules) during feature selection. In the same paper, the effect of automatic sample labeling based on actual user usage history was also investigated and good results were obtained.

During the emergence of deep learning techniques around the 2010s, an attempt to utilize its representation learning ability has been made [9] in the field of spam detection. The classification model used in this study is called *stacked auto-encoders*. An auto-encoder's nature is to compress its input into a representation with less dimensions and then de-compress

the representation back into the original input vector, effectively reducing the input's dimensionality while keeping most of the useful information. The neural network in [9] is comprised of multiple feed-forward layers of decreasing sizes and a softmax layer at the end. These feed-forward layers are taken from the trained hidden layer of multiple auto-encoders. This chain of layers reduces an input vector, a traditional one-hot encoded, into a representation of much lower dimensionality which is then classified by the last *softmax* layer to produce the prediction output. By using pre-trained weights, the author claimed to put the network close to the optimal solution even before training the entire network.

New deep network structures continued to be introduced. Using both text and image inputs, a multimodal network was proposed by [10] for spam classification. In this network structure, image and text are processed separately before being combined and further classified with a fully connected layer and finally a softmax layer. The image input goes through convolution layers while the text input is made of word embeddings and goes through *over-time convolution* layers and *over-time max pooling* layers.

A multilayer perceptron network with 2 hidden layers was used in [8] for the spam detection task on the SpamBase dataset. This dataset provides 57 numeric features which are the frequencies of 48 words, 6 chars and 3 other measurements regarding the sequences of capital letters from an email. The MLP was reported to perform better than the Naïve Bayes classifier on this task.

LSTM is known for the ability to learn long-term dependencies in temporal data and has been successfully applied in natural language processing tasks such as text classification and machine translation [12]. The authors of [11] defined *semantic LSTM* as the LSTM network which takes word embeddings as inputs. The basic idea in [11] is to combine Google's word2vec embeddings with WordNet and ConceptNet in order to maximize the number of words which can be converted into embeddings. WordNet and ConceptNet can be used to find semantically similar words. If a word's embedding cannot be found from word2vec data, the most similar word's embedding will be used to represent it instead.

3 Proposed Method

3.1 Data Pre-Processing

Raw email data contains headers, body and attachments that need to be extracted in order to obtain useful information. In this work, the sender's address, receiver's address, email subject and the email's textual body are extracted. The messages which are in HTML format have to be stripped of tags while maintaining paragraph structure. When conducting experiments, the word2vec algorithm requires separate sentences as training input instead of the whole text document. The following heuristics were applied on email body texts to enable better sentence detection:

- Since all spaces, including line-breaks, in HTML source code are rendered as spaces on the browser, it is necessary to strip line-breaking and non-breaking tags differently to correctly convert the email content into plain text. Content of a line-breaking tag will be turned into a single line in the tag-less output.
- Punctuations are usually good indicators for sentence detection. However, it is to be expected from email content that the sender does not always use correct grammar and punctuations. Sentence detection tools such as the VNCoreNLP toolkit [15] do not work well when punctuations are missing. As an alternative, the line-breaks are used as an additional measure to detect sentences from the text. The following heuristics are based on the assumption that humans are not inclined to place a line-break amidst a sentence. In the stripped version of the text, each line that doesn't terminate with a sentence ender will be modified. A sentence ender is one of the following three punctuation marks: a period (“.”), a question mark (“?”) and an exclamation mark (“!”). If a line is terminated with a punctuation mark which is not a sentence ender, it will be replaced with a period (“.”). If a line does not end with a punctuation, a period (“.”) will be added to the end of it.

The texts from subject and body also need to be segmented into separate meaningful words. Unlike English where the words with multiple syllables are put together as a continuous sequence of letters, multiple syllables in a Vietnamese word are separated by spaces. Based on the specific context, two successive syllables in Vietnamese can be recognized as a compound word or two singular words. In this paper, the authors adapted a method called VNCoreNLP described in [15] to extract words from email content (subject and plain-text body). Besides the word segmentation feature which covers sentence detection, this toolset also incorporates a POS Tagging method for the determination of word types. This feature is used to remove unwanted tokens which do not contribute to the text's meaning or are too specific to be good feature. These tokens include decorative symbols, meaningless character sequences and numeric values. VNCoreNLP outputs a set of detected sentences, each sentence consists of multiple segmented words, and each word is associated with its POS tag (a.k.a. word type).

The proposed method can possibly be applied upon emails in English or other languages provided that a suitable word segmentation technique is used. In order for word embeddings to be effective, each word vector should be associated with a meaningful word in the corresponding language. For English, a trivial tool such as sklearn's CountVectorizer should be adequate for content tokenization. In its default behaviour, this tool extracts words by finding alphanumeric sequences of at least 2 characters seperated by empty spaces and punctuations.

The sender's address can be an important information for determining the importance of an email message. Our approach to create a vector representation of the sender is based on the assumption that a sender would continue to send messages of similar importance as the ones he had sent in the past. The email messages in the dataset are labeled as one of the 5 importance levels. The number of messages from each importance label of a sender in training data can be counted so that each sender is associated with a set of 5 integers. A *sigmoid* function (1) is used to normalize these integer values into real values in the range $[0, 1)$.

$$S(x) = \frac{x}{1 + |x|} \tag{1}$$

The number of messages that the sender has received is also counted. A spammer address usually has little to no incoming emails. On the other hand, an important person is expected to receive a high number of emails. The number of inbound messages is also normalized using the formula in (1) and added to the sender's vector representation, making it a real-valued vector of 6 values.

3.2 Word Embedding

A word embedding is a fixed-length, real-valued vector which represents a word in natural language. There are two approaches to using word embeddings: pre-trained embeddings and online word embeddings. Pre-trained word embedding is technique of generating word embeddings from a corpus of sentences using unsupervised learning. The resulting word embeddings are then used as initial weights for an Embedding layer in a neural network. These weights are set to be *untrainable*. An Embedding layer takes a document, represented by a dense vector of word indexes, as input and outputs an $n \times m$ matrix with n being the document's length and m being a word vector's size. In a neural network, an Embedding layer is usually the first layer and is usually succeeded by other layers. Online word embedding is the technique of initializing random weights for such Embedding layer and set these weights to be trainable so that they will be trained along with the rest of the neural network. In the approach of using pre-trained embeddings, an implementation of the word2vec embedding training algorithm in the Gensim toolkit was used to train word vectors from the training data. It is commonly believed that word2vec requires a large dataset to train and it is also popularly advised to use publicly available pre-trained word embeddings such as those trained with the Google News corpus [1]. However, such pre-trained embeddings are not readily available for Vietnamese language. Therefore, the experiments in this paper were conducted within the scope of available email data. In the second approach, the Embedding layer from the Keras deep learning framework is used. This layer is placed as the first layer of the neural network and is trained along with the network. Using Embedding layer is very different from using pre-trained embeddings. Embedding layer is trained in a supervised way based on the model's loss function. Pre-trained embedding trains the word vectors in an unsupervised manner (using auto-encoder mechanism). This article compares the performance of these approaches in the experiments.

The size (dimensionality) of word embeddings is also a concern for many researchers. Google chose 300 as the dimensionality for their published word2vec embeddings though the reason behind was not specified. It is also the most popular choice in other works [16]. In this paper, the authors do not intend to propose a method for embedding dimensionality selection but examine the effectiveness of a smaller dimensionality instead, based on the assumption that a small dataset would require smaller word embedding dimensionality. Therefore, in both the first and second approach, the values of 300 and 128 respectively were chosen as the size of word vectors, and results obtained from these two choices were presented in Table II.

3.3 Network Structures

In order to effectively combine sender's properties with the email content, we propose the network structure at Figure 1.

An email's content is represented by a series of word vectors. The representation of the content part is an $m \times n$ matrix where m is the length of the text and n is the dimensionality of the word embeddings. The document length n is fixed at 300 to ease the experiments using the Keras API. Messages which are more than 300 words long are trimmed and those which contain less than 300 words are left-padded with zeros. There are 1,472 emails which are more than 300 words long, which accounts for 12.15% of the dataset. The mean length of those emails is 554, meaning 254 words are trimmed from them in average. The proposed method assumes that it is adequate to read the first 300 words of an email message to figure out it's importance level. This assumption is based on the following reasons: (a) most email messages in the collected dataset are shorter than 300 words; (b) typically, the main point or general idea is placed at the beginning of a message and it is counter-productive as well as counter-intuitive to do the opposite. The sender information is encapsulated in a vector whose size is 6. These two parts do not match in shape so they cannot be merged directly. However, a recurrent layer (a.k.a. recurrent network) can be used to process the series of word embeddings one by one and produce a vector as output. For textual content, which is a type of temporal data, a recurrent network such as the LSTM is the typical choice. An LSTM layer receives one word vector at a time and outputs one real value per internal unit. In other words, its output is a real-valued vector whose size equals the number of internal LSTM units. Each word vector fed to an LSTM layer causes its internal state to change, which in turn changes the values in its output vector. After all word vectors of a document are fed, the output vector is the prediction result of the LSTM layer for the input document. It is a

[1] https://code.google.com/archive/p/word2vec

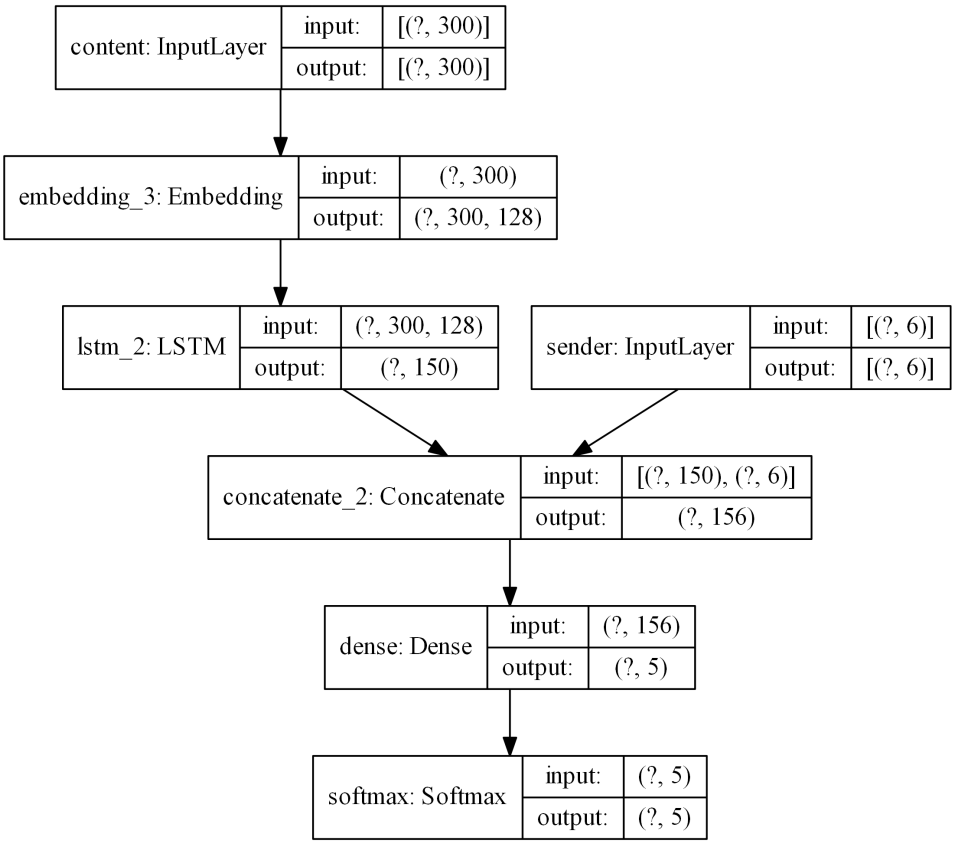

Figure 1. Proposed neural network structure for combining content features (word embeddings) with social features (sender's properties).

common practice to use the output of a recurrent layer as input to a feed-forward network such as MLP, CNN or *perceptron*. By concatenating the recurrent layer's output with sender's properties, sender information is effectively added to the representation of email content produced by the RNN. The concatenated vector is the combination of features from both content and sender. It can be used as input for further classification, e.g. by subsequent feed-forward layers. The fact that the output of an RNN is not a type of sequential data makes feed-forward networks the better candidates for processing the above-mentioned combined input vector.

3.4 Training the Networks

The training algorithm plays a significant role in the success of neural network training. Although the basic approach of training a deep neural network is using gradient descent with back-propagation, many specific gradient-based learning methods have been proposed to suit different network models and data. Each algorithm prioritizes a different subset of features and has a different way of moving the model in the search space, all to serve the purpose of finding an optimum set of weights for the neural network.

Adagrad [17] computes and maintains learning rates for individual units. The algorithm seeks to lower the learning rates of popular (regularly updated) features and raise that of infrequent ones, making the network rely more on rare, highly predictive features.

RProp [18] is another gradient descent variation where weights are adjusted using the sign and not the magnitude of the gradient. The adjusted amount is governed by an adaptive step size which is associated to individual weight. RMSProp [19] is a modified version of RProp for training with mini-batches. It addresses a problem with RProp which occurs when the gradients of successive mini-batches vary by a large amount, resulting in sudden weight increments or decrements because of magnified step size.

The Adam [20] algorithm also maintains adaptive learning rates for each unit's weight in a layer but, differently from Adagrad, it does it by incorporating the momentum mechanism. This mechanism involves taking a number of recent gradients into account when determining learning rates for the current iteration of weight update.

Cross-entropy (a.k.a. log loss or logistic loss) is the default choice of loss function for classification problems. This loss measures how confident a model's prediction result is. A 5-class classifier with a softmax (a.k.a. probability) output will generate a probability distribution which consists of 5 real numbers. Each number can be considered the prediction score for one of the 5 classes. The final prediction result is the class with highest predicted score. The distance between the score of the predicted class and other scores indicates how confident the classifier is about its prediction. A large distance results in a low cross-entropy value, which means that the classifier is more confident with its results. Cross-entropy assumes no relativity between classes, meaning that an email is not concluded as being more important than another.

Alternatively, loss functions which are usually used for regression models (such as Mean Squared Error or Mean Absolute Error) can be applied for this problem with the assumption that the emails actually have relative importance levels and that the distance between any two consecutive importance levels is fixed. In the context of this paper, it can be said that the difference between an *important read* email and a *delete* email is twice as much as that between an *important read* email and an *unimportant read* email.

Over-fitting is an undesirable phenomenon in which a model fits the training data so well that it loses generalization and has inferior performance on validation data. Over-fitting happens especially when training data is not highly representative. Training data is a subset of real data and do not cover all possibilities, thus the objects in the training set are called *samples*. The fundamental hypothesis of machine learning is that an ideal training set is able to *represent* all real data since it possesses the necessary features and the probability distribution of each feature resembles that of the real data. However, in real practice, there is a significant difference between the probability distribution of the training set and that of the real data, making the training data *noisy*. Over-fitting happens when the trained model also learns the noise from the training data, resulting in inaccurate predictions for the objects that it has never seen. Some of the techniques to avoid over-fitting are to improve the training data's representativeness or to stop training early to prevent the model from learning the *noise*. Dropout [21] is a technique to avoid over-fitting when training neural networks. This technique works by randomly disconnecting a portion of units in a particular layer during training. The aim is to avoid the situation where only a few units have significantly stronger impact than the remaining ones in a layer.

4 Experiments

During the research work, there is not a public email dataset with suitable labels for 5-level email prioritization. Therefore, the authors proceeded to build one by collecting emails from personal inboxes and from colleagues. The collected emails contain messages in both English and Vietnamese. For the experiments in this paper, only Vietnamese emails are selected. A simple tool was built to detect and remove duplicated or very similar messages. This tool calculates similarity between emails based on the Euclidean distances of one-hot encoding vectors. This tool removes one email out of a pair when the similar portion between them is more than 75% of the shorter message. Out of 7 collected email inboxes, 12,118 Vietnamese messages were selected and labeled by each email account owner.

For each attempt of model training, the dataset is split into *train* and *test* datasets with the ratio of 90% for training and 10% for testing. In order to conduct k-fold cross-validation with $k = 10$, the dataset is divided into 10 parts which have roughly the same ratio of labels. Each experiment is repeated 10 times so that each part of the dataset is used once for testing. The reported test results are averaged values of the measurements.

$$H(y, \hat{y}) = -\sum y_i * \ln \hat{y}_i \tag{2}$$

To evaluate the experimented models, three metrics are used: *accuracy, cross-entropy* loss and *macro* F_1 *macro* [22]. These metrics are suitable for multiclass classification problems which do not make the assumption of relativity between classes. Accuracy is the portion of correct predictions over all predictions. The cross-entropy loss (2) is used to measure the difference between the classification result $\hat{y}$ with the desired result y. The closer the prediction to the desired result is, the lower cross-entropy gets. A low cross-entropy value indicates that a classifier is more confident with its predictions.

$$P_m \frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i + fp_i}}{l} \tag{3}$$

Macro F_1 score (5) is calculated from *macro precision* (3) and *macro recall* (4). The macro metrics are obtained by first calculating the metrics for each label, and then take the unweighted average of these values. In opposite, *micro metrics* are the global average of metrics on individual samples. In classification problems in general, F_1 score is the balance between *recall* – the completeness of the prediction results – and *precision* – how reliable the prediction results are. In a multiclass (greater than 2) classification problem, *micro* F_1 score does not make sense since it is not possible to properly calculate precision and recall without narrowing the scope down to each label. Specifically, instead of four different outcomes as in binary classification (true positive, true negative, false positive, false negative), there are only two possible outcomes (correct, incorrect) in a multiclass problem.

$$R_m \frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i + fn_i}}{l} \tag{4}$$

All experiments are executed on a *c5.4xlarge* cloud-based server on the Amazon Web Services EC2 platform. The *c5.4xlarge* server is equipped with 16 vCPU, each vCPU is a *thread* (a.k.a. a *logical core*) of an Intel® Xeon® (Cascade Lake) processor [23], core speed is up to 3.0 GHz.

$$\text{Macro } F_1 = \frac{2 \times P_m \times R_m}{P_m + R_m} \tag{5}$$

4.1 Experiment 1

The goal of this experiment is to compare different neural network optimizers. The authors conducted experiments on three popular optimizer choices: Adam, RMSProp and Adagrad. These algorithms are popular variations of the classic SGD algorithm. They have different set of configuration parameters. The suggested defaults from their papers will be used in this experiment. The variable α is used to denote the initial learning rate while ε is a small number, usually 10^{-8} to 10^{-7}, used for avoiding division by zero. The recommended

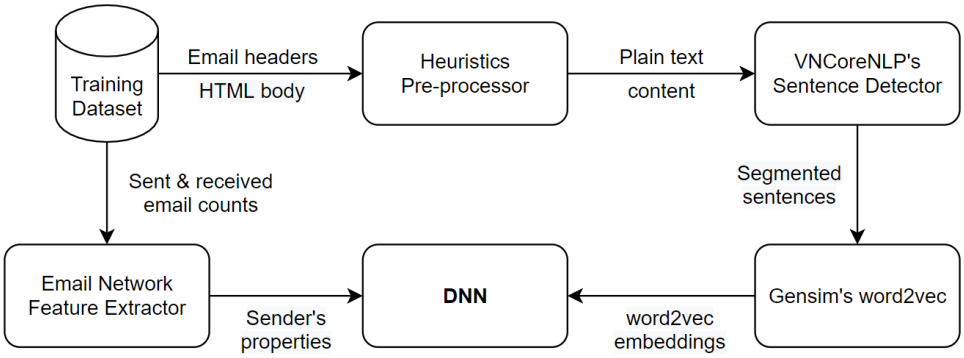

Figure 2. Data pre-processing workflow for experimenting on the network described in Figure 1.

α value for all three algorithms is 0.001. Each algorithm also has unique tuning parameters. For Adam, the parameters are set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$. For Adagrad, the initial accumulator value is set to 0.1. For RMSProp, the discounting factor *rho* is set to 0.9 and initial momentum starts at a value of 0.

For the same network model described in Figure 1, separate attempts will be made using pre-trained word2vec embeddings and trainable Embedding layer with dimensionality of 128. The results of this experiment are recorded in Table I using the three mentioned metrics. The pre-processing workflows for the two experiment setups are illustrated in Figure 2. It is important to clarify the difference between the pre-trained word embedding configuration and the online embedding one. The output of Gensim's word2vec algorithm consists of a list of words and corresponding word vectors. The Embedding layer of the DNN (Figure 1) in online embedding experiment does not require the weights from word vectors. On the other hand, these weights are needed to initialize weights for the Embedding layer in pre-trained embedding setup. Therefore, the same list of words (i.e. vocabulary) is used by the DNN in the online embedding experiment for consistency.

4.2 Experiment 2

The RMSProp optimizer has shown the highest overall performance in the previous experiment. However, the effectiveness of different embedding sizes is still in question. The authors trained the proposed model with different word embedding dimensionalities, namely 128 and 300, using RMSProp as optimizer. Pre-trained *word2vec* embeddings are created from training data using the *word2vec* implementation in the Gensim, a natural language processing toolkit. The resulting word vectors are imported as weights into an Embedding layer in the proposed neural network model. This Embedding layer is set to be *untrainable*, meaning its weights are not modified during network training.

4.3 Experiment 3

In this experiment, the authors reproduced the results of the classification approach from [5] using the described dataset. The SVC1 model, which is a Python implementation of SVM classifier, from the *scikit-learn* toolkit is adapted. To carry out this experiment, words segmented from email content are used to generate TF-IDF vectors, each vector represents an email message. To construct sample vectors in TF-IDF format, the email texts are segmented into words using the same steps as described in the data pre-processing section. For the comparison to be consistent, the selected voting mechanism for multiclass classification is OVA since the *softmax* output of the proposed model shares the same principles – each element of the softmax output corresponds to the model's prediction for a particular label. The multiclass prediction model based on SVM classifier and TF-IDF features is compared to the proposed model with 128-d word vectors and trained using RMSProp optimization algorithm.

Table I
Comparing Three Popular Neural Network Training Algorithms

Optimizer	Accuracy		Macro		Cross-entropy	
	(a)	(b)	(a)	(b)	(a)	(b)
Adam	0.6641	0.9115	0.3769	**0.8641**	6.6992	**0.6650**
Adagrad	0.5209	0.6448	0.1374	0.5090	**1.7875**	1.0729
RMSProp	**0.7134**	**0.9126**	**0.5014**	0.8632	5.9510	0.7260

(a) 128-d online embedding, (b) 128-d pre-trained word2vec

Table II
Comparing Different Word Embedding Setups

	Accuracy	**Macro F_1**	**Cross-entropy**
128-d pre-trained	0.9126	0.8632	0.7260
300-d pre-trained	**0.9185**	**0.8764**	**0.7146**
128-d online	0.7134	0.5014	5.9510
300-d online	0.7900	0.5918	4.2800

* The RMSProp optimizer is used in the above 4 attempts.

Table III
Comparing the Proposed Neural Network Model Figure 1 to Multiclass (OVA) SVM Model from [5]

	Accuracy	**Macro F_1**	**Cross-entropy**
DNN*, 300-d pre-trained	**0.9185**	**0.8764**	0.7146
OVA-SVM, epoch=50	0.7137	0.4529	0.7893
OVA-SVM, epoch=100	0.7847	0.5550	0.6161

5 Conclusion

In this paper, the authors proposed a classification model for email prioritization based on deep learning techniques. The proposed neural network utilizes not only state-of-the-art deep learning techniques, notably the LSTM units, but also a rich set of content and social features. Word embeddings generated with the *word2vec* algorithm are used as content features to meaningfully denote the email body. The various sender-related statistics are extracted to build a vector of social features. Experiments were done on a collected dataset of Vietnamese personal emails to investigate the effectiveness of the proposed model. The representation of content features using *word2vec* and the addition of sender's properties significantly improve the performance of a prediction model for email prioritization compared with a traditional machine learning approach such as SVM classifier and TF-IDF feature extraction. Additionally, this paper presented a few comparisons between different neural network configurations regarding word embedding dimensionality and the choice of optimizer. Bi-LSTM has been reported [24] to improve text classification performance for its capability to simultaneously learn the context information from both directions of the text. ELMo and BERT are emerging word embedding techniques which obtained state-of-the-art performances [25] in various tasks. However, due to their high computational complexity, the authors are only capable of using pretrained representations and thus decided not to include them in the current work. In future research, the authors wish to experiment with these techniques for text representation as well as unexplored neural network models, such as Bi-LSTM, for the email prioritization problem.

References

[1] F. Kooti, L. M. Aiello, M. Grbovic, K. Lerman, and A. Mantrach, "Evolution of conversations in the age of email overload," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 603–613.

[2] L. A. Dabbish and R. E. Kraut, "Email overload at work: An analysis of factors associated with email strain," in *Proceedings of the 20th Anniversary Conference on Computer Supported Cooperative Work*, 2006, pp. 431–440.

[3] H.-N. Thanh, Q.-D. Dinh, and Q. Anh-Tran, "Personalized email user action prediction based on spamassassin," in *Proceedings of the Context-Aware Systems and Applications: 5th International Conference*. Springer, 2017, pp. 161–171.

[4] H. N. Thanh, Q. D. Dinh, and Q. A. Tran, "Predicting user's action on emails: improvement with ham rules and real-world dataset," in *Proceedings of the 10th International Conference on Knowledge and Systems Engineering (KSE)*. IEEE, 2018, pp. 169–174.

[5] S. Yoo, Y. Yang, and J. Carbonell, "Modeling personalized email prioritization: classification-based and regression-based approaches," in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, 2011, pp. 729–738.

[6] P. O. Boykin and V. P. Roychowdhury, "Leveraging social networks to fight spam," *Computer*, vol. 38, no. 4, pp. 61–68, 2005.

[7] S. Yoo, Y. Yang, F. Lin, and I.-C. Moon, "Mining social networks for personalized email prioritization," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009, pp. 967–976.

[8] W. Yawen, Y. Fan, and W. Yanxi, "Research of email classification based on deep neural network," in *Proceedings of the Second International Conference of Sensor Network and Computer Engineering (ICSNCE 2018)*, 2018, pp. 73–77.

[9] G. Mi, Y. Gao, and Y. Tan, "Apply stacked auto-encoder to spam detection," in *Proceedings of the Advances in Swarm and Computational Intelligence: 6th International Conference*. Springer, 2015, pp. 3–15.

[10] S. Seth and S. Biswas, "Multimodal spam classification using deep learning techniques," in *Proceedings of the 13th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*. IEEE, 2017, pp. 346–349.

[11] G. Jain, M. Sharma, and B. Agarwal, "Optimizing semantic LSTM for spam detection," *International Journal of Information Technology*, vol. 11, pp. 239–250, 2019.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[14] S. R. Covey, *The 7 habits of highly effective people: Powerful lessons in personal change*. Simon and Schuster, 2004.

[15] T. Vu, D. Q. Nguyen, M. Dras, M. Johnson *et al.*, "VnCoreNLP: A Vietnamese natural language processing toolkit," in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, 2018, pp. 56–60.

[16] Z. Yin and Y. Shen, "On the dimensionality of word embedding," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, pp. 895–906.

[17] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research*, vol. 12, no. 7, pp. 2121–2159, 2011.

[18] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The rprop algorithm," in *IEEE International Conference on Neural Networks*. IEEE, 1993, pp. 586–591.

[19] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini–batch gradient descent." [Online]. Available: http://www.cs.toronto.edu/ hinton/coursera/lecture6/lec6.pdf

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[22] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.

[23] Amazon Web Services, Inc., "*Amazon EC2 Instance Types*. amazon web services (aws). retrieved may 26, 2021." [Online]. Available: https://aws.amazon.com/ec2/instance-types

[24] B. Jang, M. Kim, G. Harerimana, S.-u. Kang, and J. W. Kim, "Bi-LSTM model to increase accuracy in text classification: Combining word2vec CNN and attention mechanism," *Applied Sciences*, vol. 10, no. 17, p. 5841, 2020.

[25] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning–based text classification: a comprehensive review," *ACM Computing Surveys*, vol. 54, no. 3, pp. 1–40, 2021.