Comparing the Robustness of Machine Learning Approaches on Spam Filtering Problems

Kenneth Lee [1] **Mingye Fu** [1] **Jingwei Wan** [1] **Yating Ge** [1] **Samuel Becerra** [1] **Ruoyan Yin** [1] **Miao Hu** [1] **Hang Su** [1] **Yuntian Shen** [1] **Yuqing Yang** [1] **Yujia Lu** [1]

Abstract

Spam, a bulk of unsolicited messages sent by anonymous sources, has been a costly issue to human communication. Machine learning techniques have been shown promising to filter these messages as it adapts to the evolving characteristics of the spam. In this work, we focus on comparing various classification methods in machine learning to the problem of spam filtering. Overall, we conclude that using bidirectional gated recurrent neural network with tokenizer method is the most robust way we have found to handle this problem with our particular dataset. This work provides insights on which classification method may work better on spam filtering problem.

1. Introduction

Spam has been one of the major issues on the internet, which can take up to 75% of the total number of spam messages (MAAWG, 2006). It causes serious misuse of computational resource such as storage, network traffic speed and power to many companies (Siponen & Stucke, 2006). To tackle this issue, many of the messages needed to be filtered before they can reach the end users. Such method is called spam filtering.

A variety of methods have been proposed to build an effective spam filter, ranging from maintaining a list of IP addresses (Cowings et al., 2011) to using a reputation of the network domain (Buckingham et al., 2009). However, these reputation-based approaches are unable to adapt to the evolving characteristics of spam.

A more promising way is to make use of content-based filters, which are designed to automatically detect legitimate and spam messages. Some use the frequency of occurrence of some common words in the spam messages (Bruening, 2004; Yerazunis et al., 2005). In particular, machine learning is prone to dealing with classifying spam messages as it is developed to learn from the content of the spam with methods such as boosting tree (Carreras & Marquez, 2001), support vector machine (Sculley & Wachman, 2007), and statistical data compression (Bratko et al., 2006). It will be beneficial to review some of the recent work and shed lights on which one to choose for filtering spam messages.

In machine learning, spam filtering can be viewed as a binary classification problem setting. We selected multiple classification methods, which include support vector machine, k-nearest neighbours, Naive Bayes, recurrent neural network, random forest. Each classifier will be trained on the same dataset and tested by examining whether it can correctly predict a message is spam. As suggested by Davis, Jesse, and Mark Goadrich, we will use precision and recall as the evaluation metrics to measure the success of the model in terms of detecting spam when facing with a skewed dataset (2006).

In Section 2, we will talk about the internal workings behind each of the classification methods we selected and how we compare their performances. Results about the performance of each model in Section 3. Next, we will discuss the results in Section 4 and talk about the lessons learned from using different approaches. Lastly, we give a conclusion and potential future work in section 5.

2. Methods

2.1. Data Collection

The dataset is made available on Kaggle, an online community to allows users to find and publish data sets, explore and build models in a web-based data-science environment. It is named ”SMS Spam Collection Dataset” by Tiago A.

[1]Department of Computer Science, University of California, Davis, CA, United States. Correspondence to: Kenneth Lee <honlee@ucdavis.edu>, Mingye Fu <myfu@ucdavis.edu>, Jingwei Wan <>, Samuel Becerra <subecerra@ucdavis.edu>, Ruoyan Yin <ryin@ucdavis.edu>, Miao Hu <calhu@ucdavis.edu>, Hang Su <hxsu@ucdavis.edu>, Yujia Lu <yujlu@ucdavis.edu>, Yating Ge <ytge@ucdavis.edu>, Yuntian Shen <ytshen@ucdavis.edu>, Yuqing Yang <yyqyang@ucdavis.edu>, Github repository <https://github.com/ECS171-Project/Final-project.git>.

Almeida and José María Gómez Hidalgo. (Almeida, 2011; 2013; Gómez Hidalgo, 2012). The dataset contains 5580 samples with one predictor variable called text and an output variable specifying whether each text is a spam or ham.

2.2. Data Preprocessing

We have found that the data has a skewed distribution, meaning that we have only 13.41 percent of the data that is labeled as spam and the rest is labeled as non-spam, namely ham.

Next, we use the Natural Language Toolkit (nltk) package in Python to clean the data as follows:

2.2.1. Lemmetization

First, we lemmetize the data to ensure all the related forms of a word to have a common base form.

2.2.2. Stopwords and punctuation Removal

Then, after we removed the punctuation of each word, we take away the stopwords in the messages by using the default English stopwords list in nltk. Afterwards, we turn every word into lowercase.

The word clouds and the frequency plots for the spam and ham texts are shown in the figure 1.

2.2.3. Splitting Training and Testing datsets

Next, we splitted the dataset into training and testing set by having 15 percent of the data being testing and 85 percent being training.

2.2.4. Vectorization

In this project, it involves three different ways to tokenize both training and testing data: CountVectorizer, Term Frequency-Inverse Document Frequency(TF-IDF) Vectorizer, and Tokenization.

CountVectorizer: It converts collections of text into matrix of token counts, the column of which is the dictionary of all tokens,

TF-IDF Vectorizer: It is similar to CountVectorizer, but it calculates a TF-IDF statistic between the document and each term in the vocabulary. Each message is represented by a vector using each statistic as an element in the vector. The TF-IDF statistic tends to be low for words that have higher frequency (Ramos et al., 2003).

Tokenization: Lastly, for the recurrent neural networks, we use Tokenizer from the nltk library in Python for tokenizing the training and testing data. It maps each word in the entire document to an index. The tokenizer forms a dictionary to keep track of these indices. Then, each text message becomes a vector of indices assigned by the dictionary respectively. We then pad each vector with zeros to ensure every vector has the same length for training the recurrent neural network.

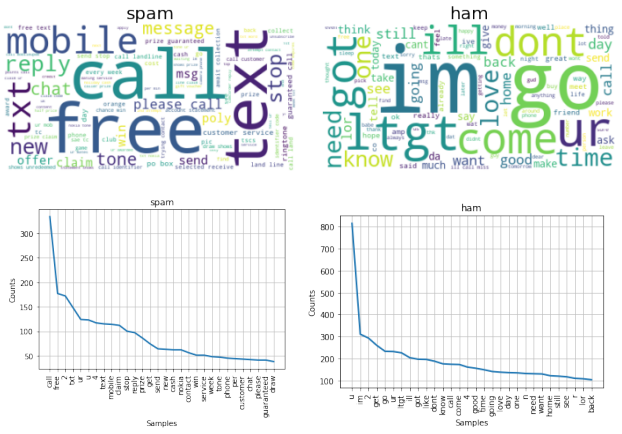


Figure 1. The word clouds for the spam and the ham text and first 30 most common words in each class

2.3. Support Vector Machine

Support vector machine (SVM) is a classification technique. One uniqueness of SVM is that the decision boundary only depends on a subset of the training data, known as the support vectors. SVM is a combination of kernel trick plus modified loss function, and it searches for a hyperplane with the largest margin.

Since SVM algorithm is based on a convex optimization problem, the global minimum could be found.

2.3.1. Maximal margin classifier

A linear SVM is also known as a maximal margin classifier, for the idea that it tend to use the decision boundary that maximize the margins, which minimize the worst-case generalization errors.

For a binary separable linear case, the training data could be separated by a hyperplane such that all the class 1 samples reside on one side of the hyperplane, and all the class 2 samples reside on the other side of the hyperplane . However, there could be infinitely possible choices of such kind of linear decision boundaries. For each decision boundary, we could move a parallel hyperplane to the closest sample belong to the class 1, and the closest sample belong to the class 2. The distance between these two hyperplanes is known as the margin of the classifier. According to the structural risk minimization (SRM) principle, to minimize the worst case generalization errors, the capacity of the model should be minimized (Vapnik, 1992). Since the larger the margin of the classifier, the less flexibility for the model, the decision boundary with the largest margin will be chose.

2.3.2. LINEAR SVM

For a m samples training set $(x_i, y_i), i = (1, 2, \ldots, m)$, x_i is the ith sample, let $y_i \in \{-1, 1\}$. The decision boundary could be written as $w^T x + b = 0$. The training of the SVM model is to estimate the parameter w and b. The prediction of class label could be defined as $y_i(w^T x_i + b) \geq 1, (i = 1, 2, \ldots, m)$. The margin of the classifier could be computed as $2/||w||^2$.

The model of a linear separable SVM can be then form as an optimization problem as below:

$$\min_w ||w||^2/2 \quad \text{s.t.} \quad y_i(w^T x_i + b) \geq 1,\ i = 1, 2, \ldots, m$$

For a nonseparable case, even if a decision boundary misclassifies some training samples, it could still be a good decision boundary. The soft margin approach, which learn a decision boundary that is tolerable to small training error, is used for the nonseparable case. The soft margin constraints are defined as

$$y_i(w^T x + b) \geq 1 - \xi_i, \text{ where } \xi_i > 0 \text{ for all i}$$

Here ξ_i is a slack variable, which provides an estimate of the error of the decision boundary on the ith sample that is misclassified. The object function is then modified to the form as below:

$$\min_{w,\xi} ||w||^2/2 + C(\sum_{i=1}^{m} \xi_i)^k$$

$$\text{s.t.} \quad \xi_i \geq 0,\ y_i(w^T x_i + b) \geq 1 - \xi_i,\ i = 1, 2, \ldots, m,$$

where C and k are user-specified parameters. Parameter C controls the number of errors we are willing to tolerate on the training set, which is usually chosen based on the validation set.

2.3.3. NONLINEAR SVM

For datasets that have a nonlinear decision boundary, data could be transformed into a new space $\Phi(x)$ such that a linear decision boundary could be found in the new space. Because of the difficulties in computation, the kernel trick method could be used so that computation in transformed dataset could be expressed in a similarity function in the original dataset. Here the similarity function is known as the kernel function. The kernel functions that satisfy Mercer's theorem are called positive definite kernel functions (Mercer, 1909).

2.3.4. TRAINING SVM

C-support vector classification was used for the model. To tune the hyper-parameter of C and kernel function in the SVM model, we evaluate the metrics of accuracy, precision, recall, AUC, precision-recall curve, and F1-score by 5-fold cross-validation. Then we take means of test metrics to evaluate different models. The kernel function is chosen in the set of 'linear', 'rbf', and 'sigmoid'.

2.4. K-nearest neighbor (KNN)

K-nearest neighbor (KNN) is a non-parametric classifier. The prediction of the label of a test point is assigned according to the vote of its K nearest neighbors' labels, where K is a user-defined parameter. KNN is a simple technique, and could work well when given a good distance metric and sufficient training dataset. It can be shown that the KNN classifer can come within a factor of 2 of the best possible performace if $N \to \infty$ (Cover & Hart, 1967).

For a test point x, the probability that its class label y=c is defined as

$$p(y = c|x, D, K) = \frac{1}{K} \sum_{i \in N_k(x,D)} 1(y_i) = c,$$

where $N_k(x, D)$ are the K nearest neighbors of the test point.

The estimate class label would then be defined as $\hat{y}(x) = argmax_c\, p(y = c|x, D, K)$.

2.4.1. THE CURSE OF DIMENSIONALITY

One of the main drawbacks for KNN is that it does not work well for high dimensional dateset. Consider a uniformly distributed training dataset in a D-dimensional unit cube (Hastie et al., 2009). To estimate the class label for a test point, we could extend the edge of a hyper-cube around this point until the hyper-cube contain a specific fraction of the dataset, which is denoted as f. Then the expected edge of the hyper-cube would be $f^{1/D}$. It can be seen that when D is large, the expected edge would be large even when we use a very small fraction of the dataset, which means that we need to consider neighbors that are far away from the test point. Since the neighbors that are far away from the test point might not be good predictors, KNN may not work well in this case. This is called the curse of dimensionality.

2.4.2. WEIGHT FUNCTION

Considering that the training points that are closer to the test point could have a larger effect on the prediction of class label, the K nearest neighbors could be evaluated using different weighting functions.

For a uniform weight function, each neighbor is weighted equally. For a fraction weight function, the weight of the ith nearest neighbor is 1/i. For a inverse distance weight function, the weight for each neighbor is the inverse of its distance to the test point.

2.4.3. Training KNN

KNeighborsClassifier in scikit-learn is used to train the KNN model. Classifier implementing the k-nearest vote.

To tune the parameter of K and weight function in the KNN model, we evaluate the metrics of accuracy, precision, recall, AUC, precision-recall curve, and f1 score by 5-fold cross-validation. Then take means of test metrics to evaluate different models. The setting of K is ranging from 5 to 15, and the weight function is chosen in the set of 'uniform' and 'distance'.

2.5. Naïve Bayes

The Naïve Bayes (NB) classifier is a group of simple probabilistic classifier based on a common assumption that all features are independent with each other (Xu, 2018). Given their simplicity, NB models are fast to run and often suitable for very high-dimensional data sets. Therefore, NB models are expected to work well with the tokenized data.

With the advantages often comes disadvantages, due to its simplicity, NB models have very few tunable parameters. NB, therefore, often serves as a baseline for a classification problem (VanderPlas, 2016). In our work, we specifically focused on three common forms of NB models: Gaussian Naïve Bayes, Multinomial Naïve Bayes, and Bernouli Naïve Bayes. The details of each form are explained as follows.

2.5.1. Gaussian Naïve Bayes

Gaussian NB assumes that each attribute follows a normal distribution, where the mean and standard deviation of each distribution are estimated from the training data (Metsis et al., 2006). As the term frequency (TF) of the data is not strictly normally distributed, Gaussian NB does not take sparse data as valid inputs. This method proves to be the most time consuming model to run among the three that we tested.

2.5.2. Multinomial Naïve Bayes

Under the Multinomial NB, each document is represented by the set of words occurrence from the document, so that the order of the words is not captured (Xu, 2018). Multinomial NB with Boolean attribute is selected in this test, as it generally performs better in spam detection (Metsis et al., 2006).

2.5.3. Bernoulli Naïve Bayes

In the Bernoulli model, each document is represented by a vector of binary features indicating which words occur and do not occur in the document. Each element in the vector is a Boolean expression of the occurrence or absence of the term (Xu, 2018).

2.6. Recurrent Neural Network (RNN)

Recurrent neural network has been shown to handle sequence models (Sutskever et al., 2014). We attempted four different RNN architectures for this spam filtering problem for comparison by using Keras, a deep learning package built in Python.

2.6.1. Simple RNN

Unlike FFNN, recurrent neural network relates output to previous computations. In order to apply simple RNN, input words are transformed into dense vectors, which are processed one by one. Previous outputs are held in memory of the simple RNN and are combined with current input to be processed. In spam classification, the input messages are sequential combinations of words. Thus, RNN is utilized instead of NN.

RNN model construction: With keras library, construct the first lay with word embedding function. Embedding represents the words with dense vectors. With simple RNN functions, build the models of one hidden layer with sigmoid tanh function.

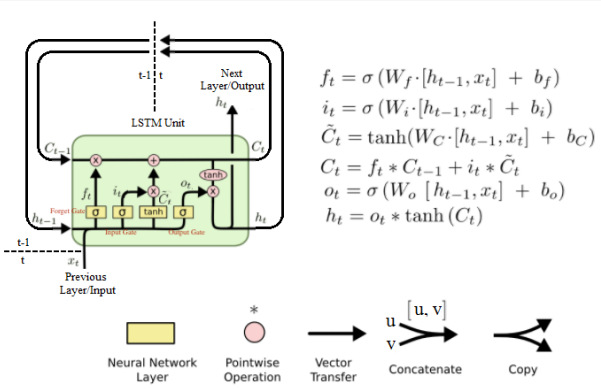
Model training: This operation is realized by the following steps: 1. Use Adam to optimize neural network parameters; 2. Use binary cross entropy loss function; 3. The epochs is set to 10, and the batch size is set to 60.

2.6.2. Long-Short Term Memory (LSTM)

LSTM: Recurrent neural networks with a Long Short-Term Memory architecture excel in various learning problems related to sequential data. In an LSTM architecture, the main idea is that there exists a memory cell that overtime can maintain its state. Through three non-linear gating units (input, forget, output) we can regulate the information flow into and out of the cell (long term memory). (Greff et al., 2016) These gates are essentially a neural network layer, each with a specific functionality. Using an LSTM architecture, we were able to read the spam message and fully understand the dependency from one word to the previous one.

The main components of this neural network architecture: word embedding input layer, LSTM layer, dropout layer, and an output layer. The output layer is a a sigmoid activation function. (Word Embedding and Dropout Layers are

explained in the next section: GRU).



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2. LSTM: Unit Diagram. Credit to "StackOverflow.com"

Forget Gate: The forget gate enables the LSTM cell to reset its own state. This allows learning of continual tasks. Essentially, this sigmoid layer decides what information the cell will keep or throw away from the cells at previous time step.

Input Gate: The input gate is defined in two parts. A sigmoid and a tanh layer that decided what values will be updated. Overall, the role of the input gate is to selectively update the cell state values with new values from the input.

Output Gate: The output gate decides what hidden information will be passed to the next cell.

Training Details: The input size of the embedding layer is the number of max words which we set to be 1000. We trained the model with a batch size of 128 over 10 epochs. We set a callback function to stop training if we see that the metric value loss is changed by 0.0001. We calculate the loss using binary cross entropy with an RMSprop optimizer. We set the activation in our hidden layer to be ReLU. The threshold value for the dropout layer is 0.5.

2.6.3. Bidirectional Gated Recurrent Unit (GRU) with embedding from scratch

There are five main components of this neural network architectures: word embedding input layer, GRU, bidirectional layer, global max pooling layer, and the dropout layer. The output layer is a sigmoid activation function.

Word Embedding Layer: Word embedding is to turn words into dense vectorized representation to represent the projection of the words in a vector space (Mikolov et al., 2013). Basically, the position of a word can be learned based on its surrounding words from the text within the vector space. We incorporate the embedding layer into the model so that the positions of the words can be learned along with the model. We specifically incorporate the word embedding to be our input layer of the neural work to let the model to learn from the "features" of each word based on the values of their representative vectors.

Gated Recurrent Unit: GRU is similar to LSTM architecture (Cho et al., 2014). The main difference between LSTM and GRU is that LSTM has 3 gate units: an input gate, a forget gate, and an output gate, whereas GRU operates on only two gates: a reset gate and an update gate, while both of them are to tackle the problem of vanishing gradient issues in learning long-term dependencies (Bengio et al., 1994). It is defined by the following equations:

$$\begin{aligned} r_t &= sigm(W_{xr}x_t + W_{hr}h_{t-1} + b_r) \\ z_t &= sigm(W_{xz}x_t + W_{hz}h_{t-1} + b_z) \\ \tilde{h}_t &= tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h) \\ h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \end{aligned} \tag{1}$$

where r_t is a set of reset gates, z_t is a set of update gates, and the activation h_t of the GRU at time t is a linear interpolation between the previous activation h_{t-1} and the candidate activation $\tilde{h}_t$.

Chung et.al shows that GRU is more computationally efficient and is better than the vanilla LSTM in terms of parameter updates and generalization (2014). In fact, Cho et al. found the GRU to outperform the LSTM on a suite of tasks (2014). Later, GRU is also shown to outperform the vanilla LSTM in many tasks (Jozefowicz et al., 2015). Therefore, it is promising that this architecture should perform better than the simple RNN and the vanilla LSTM architectures.

Bidirectional layer: Along with GRU, we also consider using bidirectional layer (Schuster & Paliwal, 1997). The intuition behind this idea is that we want our model to learn both the reverse and the normal order of the text to better understand the context of the text.

Global Max Pooling: Instead of using traditional fully-connected layers, we consider using global max pooling layer. It is because it has been shown that pooling layer is less prone to overfitting (Lin et al., 2013). We also do not need to fine tune the parameter for this layer as it takes the size of output of the previous layer. Since we create a sentence matrix for each text, where each row of the matrix represents a word by a vector. We then generate feature maps for each text message. Then, we use global max pooling to take the largest value out of all the feature maps to represent the text features.

Dropout: We add a dropout layer before the output layer to

further prevent overfitting problem from using non-linear function in the hidden layers and enhance the generalization ability of the model (Hinton et al., 2012). Note that our model uses rectified linear unit (ReLU) as the activation function, which is a non-linear function for the hidden layers (Agarap, 2018).

Training details: First, the input size of the embedding layer is the number of unique words plus one by the number of dimension we would like to use to represent each word vector, which we set to 100 in our case. We also set the batch size to be 512 and use Adam optimizer with a binary cross-entropy loss function. Besides the main five mentioned previously, we set the activation in our hidden layer to be ReLU (Kingma & Ba, 2014). The thresold value for the dropout layer is 0.1 to slightly penalize the model.

2.6.4. Bidirectional Gated Recurrent Unit (GRU) with pre-trained embedding layer

Similar to the architecture described in section 2.5.3, but we use a pre-trained word embedding layer called Global Vectors for Word Representation (GloVe) instead of training a word embedding layer from sratch (Pennington et al., 2014). GloVe was trained on a dataset that contains 6 billion tokens with 400,000 vocabularies. There are several downloadable options; we picked the smallest package of embeddings that has 300 dimensions.

Training details: The set up is the same as the GRU with wording embedding from scratch, except for the embedding layer, which will be replaced by the pre-trained word embedding layer.

2.7. Random Forest

The fundamental concept of random forest is, in data science speak, "A large number of relatively uncorrelated models(trees) operating as a committee will outperform any of the individual constituent models." Trees can protect each other from their individual errors.

2.7.1. Classifier

A random forest classifier is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting(Kam, 1995). Each of the decision tree operates as an ensemble (as shown in Figure 3), spits out a class prediction and the class with the most votes becomes our model's prediction (spam - 1 or ham - 0).

The random forest classifier works as the following steps:
1. Create a splitting rule.
2. Divide the data using the splitting rule.
3. Repeat until final sets (leaves) are in the same class.

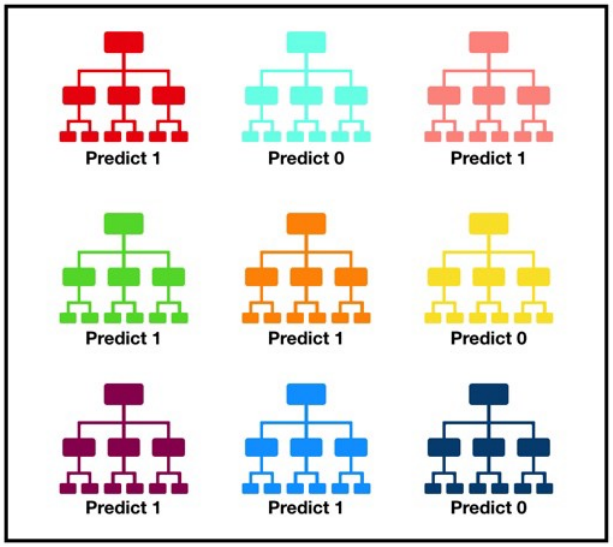


Figure 3. Random forest consisting of decision trees, each decision tree generates a prediction 0 or 1. Credit to: "towardsdatascience.com".

2.7.2. Bootstrap aggregation

The classifier uses bootstrap aggregation to ensure that the models diversify each other(Kam, 2002). Decision trees are very sensitive to the data they are trained on, which means small changes to the training set can result in significantly different tree structures. Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees. This is also called bagging and can be specified to the classifier as boolean 'bootstrap'.

2.8. Evaluation Metrics

Since spam filtering is a binary classification problem and the dataset has an imbalanced distribution for two output classes, precision and recall will be a more informative metric than the Receiver Operating Characteristic (ROC) curve to evaluate the classifiers we use in this work (Saito & Rehmsmeier, 2015). Nevertheless, we would like to get a more comprehensive view by plotting both ROC curve and precision and recall curve to examine the results.

2.8.1. Precision and Recall

As shown by equation 2, precision is computed as the ratio of the number of true positives divided by the sum of the true positives and false positives. It measures how precise our model is at predicting the positive class

Recall is a ratio of the number of true positive divided by the sum of true positive and false negative. It measures how good our model is at predicting the actual positive class.

One way to measure the robustness of the classifiers is to calculate their F_1 scores and compare them.

$$\begin{aligned} \text{Precision} &= \frac{\text{True Positive}}{\text{Positive + False Positive}} \\ \text{Recall} &= \frac{\text{True Positive}}{\text{True Positive + False Negative}} \end{aligned} \tag{2}$$

$$\begin{aligned} F_1 &= \frac{2}{\text{Recall}^{-1} + \text{Precision}^{-1}} \\ &= 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision + Recall}} \end{aligned}$$

2.8.2. ROC CURVE

A graphic way to illustrate the diagnostic ability of the binary classifier is receiver operating characteristic curve (ROC curve). It is a graphic tool that plot the true positive rate (TPR) against the false positive rate (FPR) at various threshold, as shown in equation 3.

$$\begin{aligned} \text{TPR} &= \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} \\ \text{FPR} &= \frac{\text{False Positive}}{\text{True Negative} + \text{False Positive}} \end{aligned} \tag{3}$$

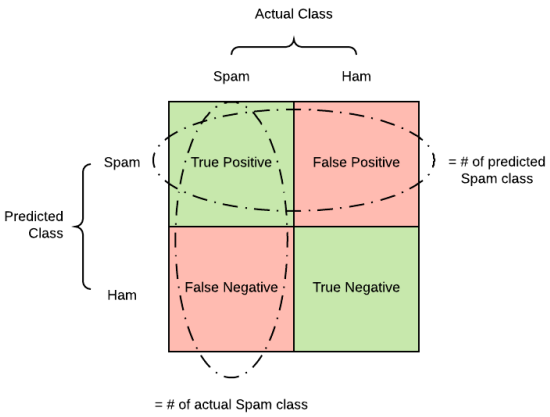


Figure 4. The relationship among true positive, true negative, false positive, and false negative for detecting spam classes.

3. Results

First of all, we will show the results on how we come up with the "best" model in each algorithm. We then show the comparative performance of all the selected models.

3.1. Recurrent Neural Network

Run the test set with the simple RNN, the accuracy is 0.8377, recall rate 0.9847, and F1 score 0.9053. The figures are improved with the following LSTM and GRU methods.

When using GRU with embedding from scratch, we can achieve F1-score of 0.98 with 3 layers and 36 nodes as shown by Figure 3.1. Then, we see that using GRU with pre-trained word embedding layer has not improved the F1-score. When using an simple LSTM architecture, we can achieve F1-Score of 0.94. We can conclude that our implementation using GRU was superior than the other RNN models.

GRU WITH EMBEDDING LAYER FROM SCRATCH			
	1 LAYER	3 LAYERS	6 LAYERS
6 NODES	0.96	0.94	0.89
12 NODES	0.97	0.97	0.91
16 NODES	0.97	0.94	0.94
36 NODES	0.97	**0.98**	0.95
GRU WITH PRE-TRAINED LAYER			
6 NODES	0.96	0.94	0.95
12 NODES	0.95	0.95	0.94
16 NODES	0.95	0.96	0.95
36 NODES	0.96	0.95	0.96

Table 1. Grid search result on the number of nodes and layers based on F1 score for GRU with embedding from scratch and pre-trained layer

3.2. Support Vector Machine (SVM)

F1-score is chosen to evaluate models, and results of CountVectorizer are shown in the Table 2 and Table 3; the results of TfidfVectorizer are shown in the Table 4 and Table 5.

C	RBF	SIGMOID
116	0.872158	0.827348
117	0.873146	0.828429
118	0.872239	0.828429
119	0.872239	0.830462
120	0.872239	0.831477
121	0.872239	0.831477
122	0.872239	0.833479
123	0.872239	0.835494
124	0.874342	**0.838643**
125	0.875313	0.838643
126	0.875313	0.838643
127	**0.876426**	0.838643
128	0.876426	0.838643
129	0.876426	0.838643

Table 2. Grid search result on penalty C and kernel function based on F1 scores for SVM using CountVectorizer

C	LINEAR
1	**0.921015**
2	0.920107
3	0.920107
4	0.920107
5	0.920107

Table 3. Tuning parameter on penalty C for linear SVM using CountVectorizer

C	RBF	SIGMOID
5000	0.911969	0.878048
6000	0.915720	0.892501
7000	0.916604	0.898400
8000	0.919273	0.902624
9000	**0.919462**	0.911055
10000	0.919462	0.911969
11000	0.919462	0.912942
12000	0.918578	0.914756
13000	0.917686	0.914812
14000	0.917593	0.916604
15000	0.916730	0.918359
16000	0.915963	0.919273
17000	0.915048	0.918507
18000	0.915048	**0.919462**
19000	0.915048	0.919462

Table 4. Grid search result on penalty C and kernel function based on F1 scores for SVM using TfidfVectorizer

C	LINEAR
1	**0.921015**
2	0.920107
3	0.920107
4	0.920107
5	0.920107

Table 5. Tuning parameter on penalty C for linear SVM using TfidfVectorizer

According to the tuning results, for CountVectorizer(), the best setting of the model is letting c = 1 and using linear function; for TfidfVectorizer(), the best setting of the model is letting c = 2 and using linear function.

3.3. K-nearest Neighbor

F1-score is chosen to evaluate model, and results are shown in the Table 6 and Table 7.

According to the tuning results, for CountVectorizer(), the best setting of the model is letting K = 5 and using distance weight function; for TfidfVectorizer(), the best setting of the model is letting K = 15 and using distance weight function.

K	UNIFORM	DISTANCE
5	0.478248	0.657320
6	0.362987	**0.63438**
7	0.381086	0.611692
8	0.292579	0.584416
9	0.294840	0.567174
10	0.191763	0.537632
11	0.194472	0.517309
12	0.142040	0.492708
13	0.147437	0.472931
14	0.122873	0.460070
15	0.122873	0.454586

Table 6. Grid search result on parameter K and weight function based on F1 scores for KNN using CountVectorizer

K	UNIFORM	DISTANCE
5	0.456013	0.635045
0	0.361404	0.6177060
7	0.369426	0.595392
8	0.281113	0.575599
9	0.291622	0.574205
10	0.226755	0.607130
11	0.442399	0.644259
12	0.400074	0.659664
13	0.595894	0.681316
14	0.568874	0.702779
15	0.627227	**0.708759**

Table 7. Grid search result on parameter K and weight function based on F1 scores for KNN using TfidfVectorizer

3.4. Random Forest

Similar with these methods above, F1-score will be utilized to evaluate model with two different data sets and 5-fold cross validation will be used to tune parameter 'N estimator'. Results are shown in the Table 8 and Table 9.

N ESTIMATOR	F1 SCORE
86	0.871376
87	**0.876238**
88	0.865451
89	0.872517
90	0.865305
91	0.876231
92	0.872773
93	0.874431

Table 8. Random Forest F1 scores for CountVectorizer

Based on Table 8 and Table 9, 'N estimator' is chosen as 87 for CountVectorizer data set and 63 for TfidVectorizer data set. Fortunately, we can see from the tables that there is actually not much difference of F1 score between different models, which means our Random Forest model will work

N ESTIMATOR	F1 SCORE
59	0.884303
60	0.880786
61	0.884540
62	0.879439
63	**0.885576**
64	0.882206
65	0.879651
66	0.874307

Table 9. Random Forest F1 scores for TfidfVectorizer

perfectly in a broad range value of 'N estimator'. In sklearn v2.20, 'N estimator' was set 10 as default while it will be changed into 100 as default in the newest v2.02. The value we've got is pretty close to the default value. Actually, larger value range (from 10 to 150) of this parameter has been swept. Considering that the larger 'N estimator' is, the more computing resources will be consumed, only the range of the best value +/- 10 is shown here.

3.5. Naïve Bayes Models

The f1 scores of our test with two datasets are summarized in the Table 10 and Table 11 below. From the tables, we see that the three Naïve Bayes models show no significant differences between the two vectorizers except for the Multinomial NB whose f1-score over test and whole set dropped a lot. Among all the models, the Bernoulli NB and Multinomial NB significantly outperformed Gaussian NB, and Multinomial NB performs even better with f1 score lager than 0.90 over test set using counter vectorizer.

Table 10. Naïve Bayes f1 score report for Counter Vectorizer

DATA	GAUSSIAN	MULTINOMIAL	BERNOULLI
TRAIN	0.848606	0.976526	0.932668
TEST	0.671096	0.905830	0.896552
WHOLE	0.819037	0.966023	0.927454

Table 11. Naïve Bayes f1 score report for TF-IDF Vectorizer

DATA	GAUSSIAN	MULTINOMIAL	BERNOULLI
TRAIN	0.848606	0.904884	0.932668
TEST	0.668896	0.875000	0.896552
WHOLE	0.818837	0.900662	0.927454

3.6. Comparative Performance among all selected models.

Table 12 indicates that using different vectorizers does not make too much differences for the two best models. Interestingly, the best machine learning algorithms were different for each word2vec vectorization methods. For CountVectorizer vectorization metric, Naive Bayes algorithm outperforms the other models, achieving the highest f1 score of 0.905830. For TfidVectorizer vectorization metric, Support vector machines metric beats all other models with the highest f1 score of 0.917073. Overall, GRU model with f1 score of 0.98 outperforms all the models regardless of the difference of the vectorization methods.

Table 12. F1 scores for comparative performance among selected models.

MODEL	COUNTVEC	TFIDVEC
KNN	0.744186	0.772727
SVM	0.905473	**0.917073**
NB	**0.905830**	0.896552
RF	0.875000	0.903553

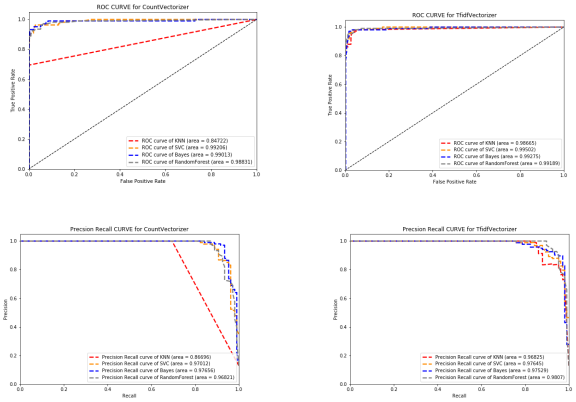


Figure 5. This figure shows ROC and PR curves for testing data by using different performance. In terms of the ROC, Support Vector Machines with TfidVectorizer vectorization is superior to all other combinations.

4. Discussion

Overall, the selected machine learning approaches perform well on this particular spam filtering task. We see that GRU approach is the best model among all. There are several lessons that can be learned from the comparison and the process of how we select various models. We summarize them as follows:

4.0.1. PRE-TAINED LAYER IS NOT ALWAYS BETTER

In the RNN architecture, we attempted to apply pre-trained word embedding layer to improve the model, but the result shows that it isn't always a better choice.

4.0.2. OVERFITTING

We see that from RNN architecture, the F1-score doesn't increase simply by increasing the number of nodes and layers in the model. The RNN may have suffered from overfitting issues. The structure of the layers will need further investigation. There maybe potential accuracy gain by adding more dropout layers in between the hidden layers to better regularize the model.

4.0.3. VECTORIZATION METHODS

Bidirectional GRU is able to achieve the highest F1 score based on tokenizer vectorization approach. Potentially, this gain may come from the fact that bidirectional GRU takes the order of the text into account when it "learns" how to classify the spam messages.

Also, we see that not all the models work well with the same vectorization method. For example, using CountVector and TD-IDF has significantly slowed down the training time of recurrent neural network due to the sparse matrices generated by the vectorizers.

4.0.4. LIMITED COMPUTATIONAL RESOURCE

As shown in results, SVM could be a potentially good solution to this problem. However, as we tried to test it with expanded data set, i.e. more than 30000 samples, it turns out that without considering the balance of numbers of sample and the dimension, it takes forever to train one model. How to reduce the dimension of the data without losing much information of texts and to gain better training time need to be considered further.

4.0.5. HYPERPARAMETERS TUNING

Tuning hyperparameters is a time-consuming and exhaustive task to know when the model should perform the best.

5. Conclusion and Future Work

From this work, we see that bidirectional GRU with tokenizer vector has outperformed all other selected models. In the future, we will increase the size of data and include a larger variety of spam messages. Also, we will also compare the model performance with other types of spam filtering methods such as reputation-based filtering methods to see if learning-based techniques are better than reputation-based techniques in terms of spam filtering.

References

Agarap, A. F. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

Almeida, T.A., G. H. J. S. T. Towards sms spam filtering: Results under a new dataset. In *International Journal of Information Security Science (IJISS)*, volume 2(1), pp. 1–18, 2013.

Almeida, T.A., G. H. J. Y. A. Contributions to the study of sms spam filtering: New collection and results. In *Proceedings of the 2011 ACM Symposium on Document Engineering (DOCENG'11)*, Mountain View, CA, USA, 2011.

Bengio, Y., Simard, P., Frasconi, P., et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Bratko, A., Cormack, G. V., Filipič, B., Lynam, T. R., and Zupan, B. Spam filtering using statistical data compression models. *Journal of machine learning research*, 7 (Dec):2673–2698, 2006.

Bruening, P. Technological responses to the problem of spam: Preserving free speech and open internet values. In *CEAS*, 2004.

Buckingham, J. T., Mehr, J. D., Rehfuss, P. S., and Rounthwaite, R. L. Network domain reputation-based spam filtering, February 3 2009. US Patent 7,487,217.

Carreras, X. and Marquez, L. Boosting trees for anti-spam email filtering. *arXiv preprint cs/0109015*, 2001.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Cover, T. and Hart, P. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

Cowings, D., Jensen, S., and Wiegner, C. Method and apparatus for maintaining reputation lists of ip addresses to detect email spam, August 30 2011. US Patent 8,010,609.

Davis, J. and Goadrich, M. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pp. 233–240. ACM, 2006.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28 (10):2222–2232, 2016.

Gómez Hidalgo, J.M., A. T. Y. A. On the validity of a new sms spam collection. In *Proceedings of the 11th IEEE International Conference on Machine Learning and Applications (ICMLA'12)*, Boca Raton, FL, USA, 2012.

Hastie, T., Tibshirani, R., and Friedman, J. Overview of supervised learning. In *The elements of statistical learning*, pp. 9–41. Springer, 2009.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Jozefowicz, R., Zaremba, W., and Sutskever, I. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pp. 2342–2350, 2015.

Kam, H. T. Random decision forests. In *3rd International Conference on Document Analysis and Recognition*, pp. 278–282, 1995.

Kam, H. T. A data complexity analysis of comparative advantages of decision forest constructors. *T. Pattern Anal Appl*, 5(102), 2002.

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

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Lin, M., Chen, Q., and Yan, S. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

MAAWG. Email metrics program: The network operators' perspective, 2006.

Mercer, J. Xvi. functions of positive and negative type, and their connection the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, 209(441-458):415–446, 1909.

Metsis, V., Androutsopoulos, I., and Paliouras, G. Spam filtering with naive bayes-which naive bayes? In *CEAS*, volume 17, pp. 28–69. Mountain View, CA, 2006.

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

Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Ramos, J. et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pp. 133–142. Piscataway, NJ, 2003.

Saito, T. and Rehmsmeier, M. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10 (3):e0118432, 2015.

Schuster, M. and Paliwal, K. K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Sculley, D. and Wachman, G. M. Relaxed online svms for spam filtering. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 415–422. ACM, 2007.

Siponen, M. and Stucke, C. Effective anti-spam strategies in companies: An international study. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, volume 6, pp. 127c–127c. IEEE, 2006.

Sutskever, I., Vinyals, O., and Le, Q. Sequence to sequence learning with neural networks. *Advances in NIPS*, 2014.

VanderPlas, J. *Python data science handbook: essential tools for working with data*. " O'Reilly Media, Inc.", 2016.

Vapnik, V. Principles of risk minimization for learning theory. In *Advances in neural information processing systems*, pp. 831–838, 1992.

Xu, S. Bayesian naïve bayes classifiers to text classification. *Journal of Information Science*, 44(1):48–59, 2018.

Yerazunis, W. S., Chhabra, S., Siefkes, C., Assis, F., and Gunopulos, D. A unified model of spam filtration. In *Proceedings of the MIT Spam Conference, Cambridge, MA, USA*, 2005.

Author contributions

- Kenneth Lee:
  - Wrote the following sections of the paper:
    - Abstract
    - Introduction
    - Data collection
    - Recurrent neural network introduction

    - Bidirectional gated recurrent neural work with embedding from scratch
    - Bidirectional gated recurrent neural work with pre-trained layer.
    - Evaluation metrics
    - Conclusion and Future Work
    - Discussion section 4.0.1, 4.0.2, 4.0.5
  - Partially contributed to the following sections of the paper:
    - Conclusion and Future work
    - Potential Challenges
    - Results about recurrent neural network
    - Data preprocessing
    - Dicussion section 4.0.3

- Miao Hu, Ruoyan Yin:
  - Together wrote the following sections of the paper:
    - Naïve Bayes
  - Partially contributed to the following sections of the paper:
    - Results about Naïve Bayes

- Samuel Becerra:
  - Wrote the following sections of the paper:
    - Long-Short Term Memory (LSTM)
  - Partially contributed to the following sections of the paper:
    - Results about recurrent neural network

- Yating Ge, Yujia Lu, Yuqing Yang:
  - Together wrote the following sections of the paper:
    - Support Vector Machine
    - K-nearest Neighbor
    - Results about Support Vector Machine
    - Results about K-nearest Neighbor
    - Results about comparative performance among all selected models
    - Evaluation metrics about ROC curve
  - Partially contributed to the following sections of the paper:
    - Data Collection
    - Data Preprocessing
    - Discussion

- Jingwei Wan, Hang Su, Mingye Fu:
  - Together wrote the following sections of the paper:
    - Random Forest Classifier theory
    - Bootstrap aggregation
    - Random Forest results analysis and tuning process

- Yuntian Shen:
  - Wrote the following sections of the paper:
    - Simple RNN
  - Partially contributed to the following sections of the paper:
    - Results about recurrent neural network