# SPAM FILTER THROUGH DEEP LEARNING AND INFORMATION RETRIEVAL

**by**

Weicheng Zhang

A dissertation submitted to Johns Hopkins University in conformity with the requirements for the degree of Master of Science

Baltimore, Maryland

April 2018

# ABSTRACT

Spam has always been irritating and overwhelming since the first day it occurred, and people has been trying to build spam filters to get rid of spams. So building such an effective and precise spam filter can be really important and meaningful. Nowadays, however, most spam filters are based on traditional methods like statistical models or keyword filtering functions. These models ignore the text meaning and words' relations in the context, and only take the occurrence of certain words into count, which will not perform well under circumstances where words in the spam show strong connections. With deep learning models getting popular, we are able to build a spam filter based on these new models and solve such problem. Moreover, using highly effective information retrieval methods for noise reduction will further improve the performance of our spam filter. In this paper, I first introduce two deep learning models for spam filtering, LSTM and CNN, which are used to accomplish the text classification task. Then, I introduce the noise reduction module based on an information retrieval tool called Elasticsearch, which helps us reduce the false positive rate of our classifier. Then, I compare the results of different approaches, and discuss why the two deep learning models with noise reduction perform 10% better than baseline model, and under what circumstances they are suitable. Finally, I discuss some possible future works that will further improve the performance of the spam filter. As the conclusion of this work, spam filter through deep learning and information retrieval outperforms traditional models, and may be the trend in the future.

# ACKNOWLEDGEMENTS

I will thank the following people as they had a significant influence on my capstone project: Dr. Timothy Leschke from JHUISI, as my mentor of this capstone project, who is patient and supportive through the whole project, and has given me helpful advices to broaden my view of this topic. Dr. Xiangyang Li from JHUISI, who gives me generous help during this capstone project.

# TABLE OF CONTENTS

# 1. Introduction

Spam filter is important to people daily life, as it can filter out detrimental or irrelevant messages from suspicious senders. The goal is always to build an efficient spam filter with high precision, which let the model that spam filters based on get evolved and improved in the past twenty years. Moreover, people want to reduce the false positive rate of the spam filter, which stands for the rate that hams are misclassified as spams.

From 1990s, different organizations try to take measures like setting up keyword filters or whitelists to avoid cluttering spams in mailboxes. This approach became very popular at that time, but later was found to be low precision and highly redundant. Later statistical models like Naïve Bayes (Vangelis et al., 2006) (T Sun et al., 2013) (M. Tariq Banday, 2008) became popular and performed well in many tasks. However, most of these models had obvious limitations and performed poorly under certain circumstances. For example, Naïve Bayes has a strong independent assumption to the input, $P(x_1, x_2 \dots, x_n|y) = \prod_{i=1}^{n} P(x_i|y)$, which means to such model, under the condition of a certain class, the probability distribution of each element in the input is considered independent and can be counted individually. This can lead to poor precision when input elements share strong connections. In spite of these limitations, statistical models can still be strong baselines.

Recently, models based on deep learning have become increasingly popular. Among all models, Long-Short Term Memory (LSTM) (Joulin, Grave et al., 2016) and Convolutional Neural

Networks (CNN) (Kim et al., 2014) are the most popular and effective ones. LSTM uses input, output and forget gate to let the model capture the context information is the message as well as remember certain useful information. CNN, which is usually an image classification model, can now be deployed as a text classification model, which can also capture context information as well as words meanings by deploying convolutional kernels to $W \times V$ matrix, where W stands for the size of text window and V stands for the number of dimensions of word vector space. However, both techniques are rarely deployed to spam filters.

Meanwhile, with the development of information retrieval, many effective and intelligent tools have occurred in the past decade. Elasticsearch is one of the most broadly used tools. It can return similar indexed materials based on the content of the query, and rank these results by their matching scores. This is also an exciting tool used to build the noise reduction module of the spam filter, which can reduce the false positive rate of the spam filter.

In the present work, I train both LSTM and CNN models to classify different messages to be either ham or spam. LSTM is set to have two hidden layers, and CNN has one layer of convolution for feature extraction of word vectors. Despite little tuning of hyper parameters, CNN has much better performance than baseline model, while LSTM is suitable for circumstances where elements show strong connection between each other. Then only to the messages that are classified as spams, they will go through the noise reduction module to further reduce the false positive rate of the spam filter. The results turn out to be even better than the classification results, with little noise introduced. In the conclusion part, I will state out that CNN model is more general to use, and

LSTM model is also a good choice under circumstances with strong dependency compare to traditional statistical models. Also, noise reduction is suitable for both models and can reduce the false positive rate significantly. For the last part of this paper, I will discuss future works that may further improve the performance of my current spam filter.

## 2. Literature Review

Spam filter has developed in the past twenty years, from basic word filters to statistical spam filters. In the 1990s, Internet began to be popular for the first time. And soon in 1996, Hotmail became the first web-based email provider, and changed the way people sent and received messages. At the beginning, people just use email as a convenient way to communicate, with little regulation. But this absence of regulation soon made a chaos in the market, and users' mailboxes were soon filled with irrelevant messages. With the increasing number of spam, ISPs started to use crude filters based on keywords, patterns or special characters. This can be concluded as the first phase of the development of spam filters. ISPs tried their best to improve the performance of their spam filters by guessing what their users actually wanted in their inbox and tried to filter out spams, but early attempts to create appropriate and accurate filters failed spectacularly. These highly inaccurate spam filters let ISPs to offer communications to senders and ESPs. And this later involved into whitelisting spam filters, which whitelisting legitimate senders to avoid doing any analysis to the content of messages.

Later in 2000s, when people more and more realized the limitation and poor performance of traditional spam filters, they began to introduce statistical methods to spam filters to improve the performance. The most popular methods include Naive Bayes, Support Vector Machine, AdaBoost and Maximum Entropy Model. These models were well established and tested in the past ten years. They could perform well, without being sensitive to feature selection strategy and easily scalable to very high feature dimension and good performances across different datasets. But soon people

realize these statistical models has their limitations. For classifiers like SVM, it needs strong and informative vectors that are highly discriminative to performs well, which means these classifiers depends heavily on the performance of representation learning models. Back then, representation learning models like Word2Vec hasn't occurred yet, so most of these models are based on bag of words or feature extraction techniques. This means spam filters based on SVM may have poor performance when features for some datasets are hard to extract.

Also, Naive Bayes was popular and efficient for some datasets. The whole model has a very strong independent assumption as introduced previously. This makes it suits datasets with weak context information perfectly, and widely used in text categorization task. It also enjoys a large popularity in anti-spam researches and often serves as baseline method for comparison with other approaches. However, Naive Bayes assumes that each word is independent from each other, and words follow same probability distribution. In other words, this model believes that sentences are generated from a word generator, where words are generated based on different possibilities. This apparently is not how a message is normally generated. As it ignores the context information between words, it will not capture useful information in the messages to discriminate messages belong to different classes efficiently. As the nature of this model is counting words in the messages separately, it doesn't involve training a model with high dimension parameters to minimize the loss from training and keep the useful information. So the model is simple, but easily loss critical information in messages.

Deep learning models are designed to fix these problems. Recurrent Neural Network(RNN) model was first proposed to capture the context information in the message. To accomplish such goal, the whole model is built based on time series. In other words, the model takes the words in messages in order, and process words at the head of each messages first. This allows the model to exhibit dynamic temporal behavior for a time sequence. Unlike traditional neural networks, RNN model can use the internal state to process sequences of inputs. This makes the model applicable to tasks related to natural language processing with strong context information. Also, when we stack the RNN cells to several layers, we can capture the context information in messages even better, from both order of a message.

However, this model suffers a lot from two problems: vanishing gradient and exploding gradient. Both of the problems occurs when the length of the input message is too long. RNN, just like traditional neural networks, uses back propagation to update parameter matrices, and the effort of back propagation will accumulate from the last layer of the neural network to the first layer. And as RNN is time based, we can consider each time of getting an input word to be a hidden of neural network. So in this situation, as the gradient of updating parameters will accumulate together, whichever word comes at first will have either very small or very large impact on the changing of parameters. Like when we have two numbers 0.99 and 1.01, both of which are very close to 1, we can time it many times. After a certain amount of timing, 0.99 will be close to 0, while 1.01 will be close to infinity. This is basically why vanishing and exploding gradient problem exists. So to concur this problem, LSTM model was proposed. In LSTM, three gates were proposed

to let the model remember only the inputs that matters, and forget the inputs that have little useful information or are noises. To each LSTM cell in the time series, three gates were deployed with separate calculating matrices. This approach makes LSTM able to handle the two problems that occur in RNN model, and let it be more advanced and more popular. Other than using gates to handle incoming messages, LSTM model is very similar to RNN model, which focus mostly on the context information of the message.

While most models focus on either words or context information coming from a message, not many models focus on both. The proposed CNN for text classification model gives people a new way to process messages. CNN model is used mostly in image processing field. This neural network model depends on processing kernel called convolutional kernel. This kernel can extract features from different channels of an image, and make the image classifiable after going through multiple hidden layers. In 2014, Yoon Kim proposed a CNN model that applied convolutional kernel to text classification. This work considers text as a channel of an image, and use the same window to truncate the message into several pieces. By using the word vectors of each word, we can form several matrices that are similar to matrices built from image channels. And deploy convolutional kernel to these matrices will help us extract features from the message, and classify it to certain class. This approach apparently can be differentiated from previously mentioned models. As convolutional kernel has certain filter window to extract features, it can process several consecutive words once, which means the extracted features contain context information. Also, it only takes the maximum feature into consideration, instead of updating all word vectors, it only

updates a small part of words. CNN model for text classification considers both context and word information, and the gets popular right after it was proposed.

One more thing that can be helpful for building spam filter is noise reduction model. Most spam filters don't have noise reduction model, and only take the result from text classifier as the final result. Before the use of effective information retrieval tools, people tried to use ranking models for messages, like TF-IDF model. The ranking score for each message can be considered to be accurate, as it took both number of words in a single document and all documents in to calculation. However, it was low in efficiency, because to each query, the model needs to parse it to several words, and go through all documents for each query word. The time complexity for this approach is $O(mn)$, where m is the number of words in a query, and n is the number of messages we need to go through. Though the time complexity is in polynomial time, when the number of messages we have is huge and queries are long, the model will run slow to get the results.

In this situation, introducing Elasticsearch, which is an information retrieval tool, as noise reduction model can be a good choice. Elasticsearch is a search engine built by Elastic. It is a distributed, RESTful search and analytics engine that is capable of solving a growing number of use cases. It can conduct many types of searches like structured, geo, metric. It returns results based on the score of each document for a specific query, which takes single scores like TF-IFD, which we mentioned before as a ranking model, and norm score into consideration. This complicated score algorithm makes sure that the result is returned in high precision.

Other than this, Elasticsearch is really effective as a search engine. It implements inverted indices with finite state transducers for full text querying and BKD trees for storing numeric and geo data, as well as a column store for analytics. This means instead of looking for a real string, it uses HashMap to map strings to numbers, and use numbers for retrieval process, and all these happens in the memory. Also, as Elasticsearch is a distributed system, it can back up and speed up easily, with holding large amount of data. Also, Elasticsearch is scalable. It runs the same way on a single node and in 300-node cluster. It can scale horizontally to handle multiple events per second, while automatically managing queries and indices on different nodes effectively. What's more, it can handle multiple types of data at the same time. Sometimes we can have different kinds of data for the same task. This may because some data miss some fields of information, and cannot have the same format as other items do. We can handle this situation using Elasticsearch. Elasticsearch puts all documents a specific index and type, and each document have several fields. For all data in different format, we can index them into Elasticsearch following the same format, while missing contents in some fields will not hurt the retrieval process in any way. This makes Elasticsearch applicable to all datasets with now limitation. Also, Elasticsearch has several APIs like RESTful API and Java API. This means using it is not limited to a specific kind of language. So Elasticsearch can be a useful and effective tool for building our noise reduction model. All these attributes of Elasticsearch is critical to our model, as the pipeline model is effective only when all parts of it are effective with high precision.

## 3. Proposed Solution

My proposed solution tries to achieve the goal to build an effective spam filter with high precision, especially with low false positive rate. Normally, traditional spam filters (Androutsopoulos et al., 2000) (Sahami et al., 1998) take the text classifier as the only way to get the class of a message and ignore noise reduction module, shown in figure 1. This model is neither highly accurate nor highly efficient. To solve this problem, I first introduce LSTM and CNN model to be the text classifier, and then also consider a noise reduction model for better precision.

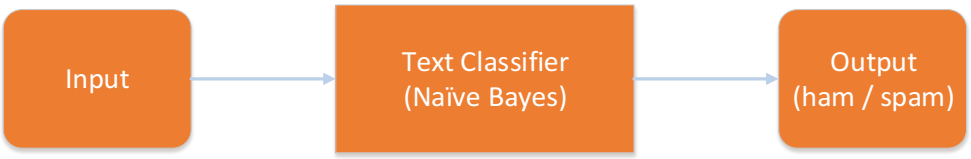


*Figure 1. Traditional Model Overview [1]*

To build such spam filter, I propose a pipeline model which takes input messages first to the text classifier, and the text classifier will be first trained by certain amount of training data, and then get connected to the whole model. As for output result, if a message is classified as ham, it will not go through the noise reduction module and will be set as ham immediately. A 'spam' message after classification will instead go through the noise reduction module, which take the input message as a query and use a certain kind of analyzer to analyze it, search for the closest matching results and arrange them by their score. The suspicious 'spam' takes the label after noise reduction as its output class.

Without digging into details, the high-level pipeline design is shown in Figure 2. In this model, we take advantage of both high precisions from deep learning model based text classifier and high efficiency from Elasticsearch engine.

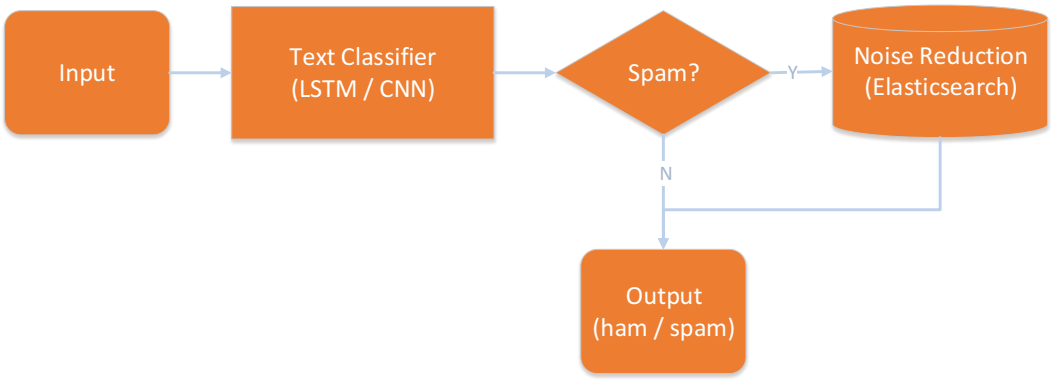


*Figure 2. Proposed Model Overview [2]*

Note that LSTM and CNN are two separate models, and are combined as LSTM / CNN only because I deploy both models in this work. Changing text classifier to either LSTM or CNN let us be able to capture both words meanings and context connections at the same time. And adding the noise reduction module can take fully advantage of training dataset, which helps to improve the precision of the spam filter.

# 4. Model

I now describe the details of models I used in building this spam filter. The model architecture, shown in figure 2, shows the steps of data processing. Based on this, I will first introduce text classifier with both LSTM and CNN, and then discuss about noise reduction using Elasticsearch in detail.

## 4.1. Text Classifier

Text classifier is the most important part of this spam filter. Its precision determines how well our model can perform directly. In this work, I introduce two deep learning models as text classifier, LSTM and CNN.

A simple and efficient baseline for text classification is using Naïve Bayes, which counts the occurrence of each word, and use the words frequency to calculate the probability of a given message that can exist under certain class. Then compare these different probabilities and select whichever class with highest probability to be the class of the given message, which in spam filter are ham and spam.

### 4.1.a. LSTM

I use LSTM model based on (Joulin, Grave et al., 2016). The basic structure of the model is shown in figure 3. I improve the previous mentioned model and make it to be a two layer LSTM model, which can have better performance. I define each word has a randomly initiated embedding

in vector space of k dimensions, while the vocabulary size being V. Let $x_i \in \mathbb{R}^k$ be the k-dimensional word vector corresponding to i-th word in the message. This embedding is used as the input to each LSTM cell.

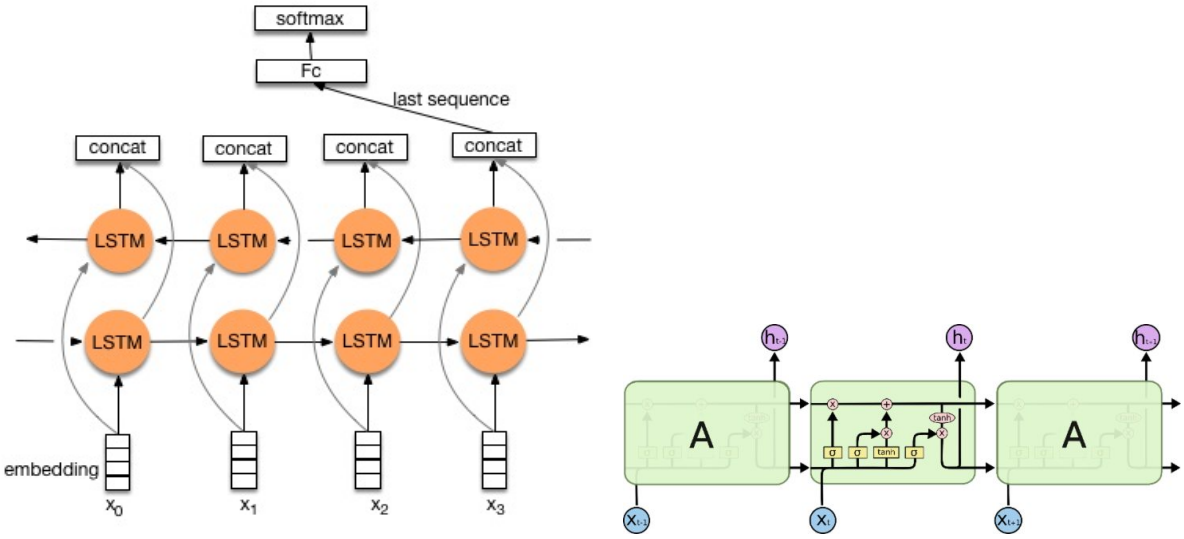


*Figure 3. Two layer LSTM [3] and single LSTM cell [4]*

For each LSTM cell, it contains three gates to control weight either more or less to current input word, which are input gate, forget gate and output gate. Input gate, which is $i_t = \sigma\left(W^{(i)}x_t + U^{(i)}h_{t-1}\right)$ is in charge of how much of current result should we keep, and forget gate $f_t = \sigma\left(W^{(f)}x_t + U^{(f)}h_{t-1}\right)$ means how much of past result should we forget. These two gates will form a new memory cell, which is $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t, where\ \tilde{c}_t = tanh\left(W^{(c)}x_t + U^{(c)}h_{t-1}\right)$. What's more, output gate $o_t = \sigma\left(W^{(o)}x_t + U^{(o)}h_{t-1}\right)$ stands for how much of the total result we should pass to the next level, and set the output of the cell in this time stamp t to be $h_t = o_t \circ \tanh(c_t)$. As we can see, each of the three gates and the new cell has its own

parameter matrix W and U, which means that each gate will be trained separately and not to let one set of parameter matrix to hold too much information, which will affect the performance of LSTM model. So after going through all three gates, each LSTM cell will output a vector that can be used for classification. As shown in figure 3, LSTM cells are organized by time sequence, and are stacked together to be two layers. And we will concatenate the result of the two layers together to be our output result. Based on time sequence, we will only take the last sequence output, and go through Softmax function $p_i(x) = e^{\theta i * x} / \sum_{t=1}^{n} e^{\theta t * x}$, where i stands for the i-th class, and the result is the probability that the sequence belongs to this class. And then we output the class that has the highest score to be the classification result.

One more step is that we need to generate word embedding for each word in our vocabulary. In this work, I randomly initiate the word embedding matrix for all words, and use loss function of LSTM model to update both parameters in LSTM cells and the word embedding matrix with size $V * k$. This means all word embedding is locally trained, with no pre-trained vectors used. This suits better for the purpose of this work, as locally trained embedding can better represent the meaning of each word under spam filter task, where words can have different meaning or even contrary attitude to general meaning. For example, 'join' can be a positive and harmless word globally, but can be very common in spams in context like 'join us and click the link'.

We can see that LSTM model takes both each word and the order of words in the text to consideration. This means that in the last sequence, we will get the information of each words as well as the connecting information within the message. And as LSTM has three gates, though it is

a time based model, it will not suffer too much from vanishing or exploding gradient problem like RNN or GRU models do. So choosing LSTM to be the text classifier will let spam filter works better on catching the context information of the messages.

### 4.1.b. CNN

The CNN model's architecture is shown in figure 4, which is implemented and improved based on the work of (Kim et al., 2014). This model more straight forward. First we still set $x_i \in \mathbb{R}^k$ be the k-dimensional word vector corresponding to i-th word in the message. Also, we set the window size of truncating the message to be w, vocabulary size to be V and word embedding to be in a vector space with k dimensions. So to the whole message, the model will first truncate it to several window-size limited pieces, and use word in each pieces to form an embedding matrix, as shown in figure 4.

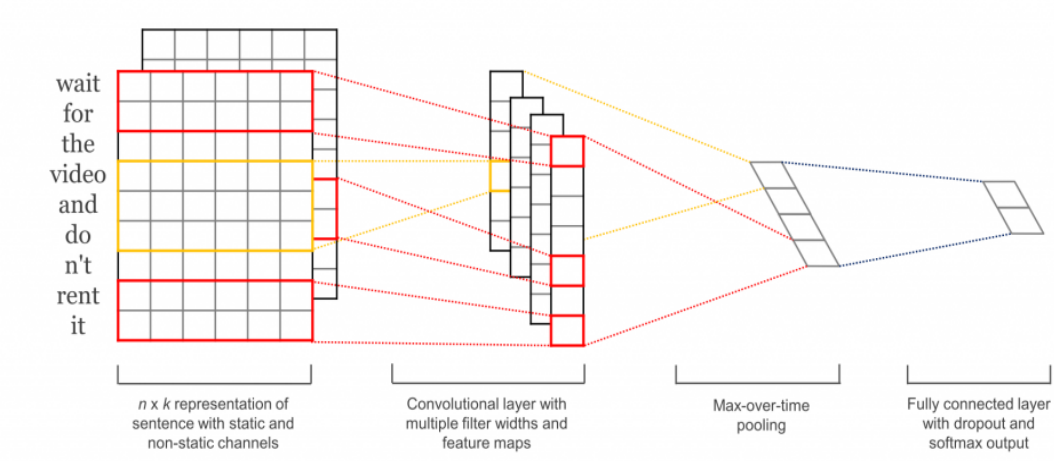


*Figure 4. CNN model for text classification [5]*

The embedding matrix $X_i$ is simply concatenated together using word embedding $X_i = x_1 \oplus x_2 \ldots \oplus x_w$, where i is the number of text piece. In this work, we apply one hidden layer of convolutional kernel to extract information from $X_i$. To generate a new feature in the hidden layer, we need to deploy kernel to h words in the matrix, where h is the number of words that can handle by kernel by one operation. So we have $c_i = \sigma(w \cdot x_{i:i+h-1} + b)$, where w and b are weight and bias, while $\sigma$ is sigmoid function for non-linear purpose. Using each kernel, we can generate features in hidden layer $C_{ti} = [c_0, c_1, \ldots, c_{w-h+1}]$, where $C_{ti}$ is the vector representation of all extracted features of t-th kernel. In this hidden layer, we can set m convolutional kernels, which means all kernels will generate m new feature vectors, from $C_0$ to $C_{m-1}$. This is for only one word embedding matrix, and there should be s = $\left\lfloor \frac{len(text)}{w} \right\rfloor$. And to all these feature vectors that extracted from one kernel, the result will be added up $C_t = \sum_{i=1}^{s} C_{ti}$.

Later, when we want to get a value from each $C_t$ to form a vector that can be later used in Softmax function, we will extract one value from each of our feature vector. To do that, we simply get the maximum number in the feature to represent the feature generates from this kernel, which means we get $\hat{c} = \max(C_t)$. This is the max-over-time pooling processing to the feature vectors. And then we use this concatenated vector after max pooling to calculate the probability for each class by going through Softmax function, and whichever has the highest score is our predicted class.

Based on this model, some improvements are made to better suit the task. As mentioned in 3.1.a, words may have different meanings and sentiment under different circumstances and context,

I improve the model by using locally trained vectors instead of globally trained vectors. This can prevent introducing extra noise from global material to text classifier. So now we train model parameters as well as word embedding, until the result converge.

As we can see, CNN model uses convolutional kernel to extract context information from texts, as well as words themselves. However, unlike LSTM, it first truncate the message into several pieces based on the size of window, which means it will separate a whole sentence into several independent sentences, and then it only takes the maximum feature in the feature vector into later calculation, instead of taking the whole sentence like LSTM does. So CNN model focus less but still much on context and word connecting information, and more on word themselves. So CNN model reaches a perfect balance between traditionally statistical model, which only count on words, and LSTM model, which fits better for messages with strong word connections. It now can handle more general kinds of dataset.

## 4.2. Noise Reduction Module

After going through the text classifier model, the message will be classified as either spam or ham. As to general definition, we define spam as positive here. If a message is classified as ham, then it goes directly to the mailbox of user, and user will view this email and deal with it manually.

However, if a message is defined as spam, according to normal operation, this message will be sent to ‘Junk’ or ‘Deleted’ folder and user will miss these emails. If the message is truly spam, then the spam filter does a great job and help the user get rid of the malicious message. But if the

message is actually a ham, then we misclassified this email to be spam, which increases the rate of false positive. False positive misclassification will result in severe damage to user. For example, users will miss critical business information just because the message looks like spam to the filter and gets deleted.

In this work, I use noise reduction model to fix this problem. This model is built on an effective information retrieval tool called Elasticsearch. In Elasticsearch, we firstly index our training dataset that trains our text classifier into Elasticsearch, with setting specific analyzer. Analyzer will first use tokenizer to tokenize the input text. For example, normal English tokenizer will choose to split texts by spaces. And then it will filter out stop words, which are normally tags and punctuations. At last, analyzer will change all terms into lowercase and index them into Elasticsearch. The whole process is shown in figure 5.

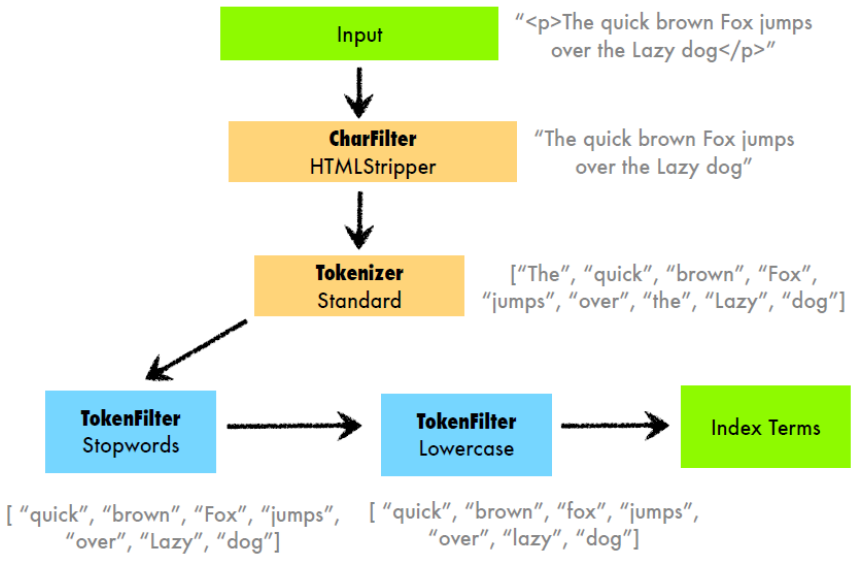


*Figure 5. Elasticsearch analyzer [6]*

Later when we have a message that classified as spam and needs to go through noise reduction module, Elasticsearch simply use the same analyzer as shown in figure 5 to analyze the query message to a set of terms and search indexed files for best match. The reason that we must use the same analyzer is that it is the only way to look for Elasticsearch to search for the perfect match. For example, for different analyzers, to analyze 'i love computer security', one may get terms 'i', 'love', 'computer', 'security', while another will get 'i', 'love', 'computer security', which treats the last phrase as a whole element instead of two words. As in Elasticsearch, element is the smallest unit to search for, if we use the first analyzer to index training dataset and use the second to analyze our query, we will never get match of element 'computer security', because only 'computer', 'security' exists.

Then Elasticsearch will return top k best matches arranged by score from high to low, where k can be set manually. The score is got by combination of several single scores $\text{score}(q,d) = \text{queryNorm}(q) \times coord(q,d) \times \sum_{t \in q} TF(t \text{ in } d) \times IDF(t) \times t.getBoost() \times norm(t,d)$, where q is the query, and d is the document we want to get the score. $\text{queryNorm}(q) = 1/\sqrt{\sum_{t \in q} IDF(t)^2}$, which is for normalization purpose. $\text{coord}(q,d) = \left(\sum_{t \in q} count(t,d) * score(t)\right) \times \left(\sum_{t \in q} count(t,d)\right)/len(q)$, which shows the reward for documents that has more occurrence of query terms. Boost parameter is set to be 1, so it won't affect the final score. $\text{norm}(d) = 1/\sqrt{num\_terms}$, where num_terms stand for the number of terms in the field that we match the term in the document. The score can show how well the document match our input query.

In this module, when I get the returned results from Elasticsearch, I add the score up to two scores based on the class of each result, and predict the message to be spam or ham depending on which score is higher.

As we can see from the description of the noise reduction module, we can see that this model takes fully advantage of the information in the training dataset, and help to further lower the false positive rate of the result. Although it may also make the true positive classification result to become false negative, which means classifies spam as ham, this is more tolerable to users, as the probability is low and handling spam manually is much safer than automatically deleting important messages that users will never get.

# 5. Experiments, Results and Discussion

## 5.1. Datasets

In this work, I test the spam filter with two datasets: Grumble and Ling-Spam dataset, which represents two different circumstances to test our spam filter.

- **Grumble**: This dataset contains 5,574 labeled messages in total, with 4,827 hams and 747 spams. Each message in this dataset has relatively shorter in length, with mostly 15 to 40 words per message. Also, words in each message has strong connection with each other, which makes each message has clear sentence meaning.
- **Ling-Spam**: This dataset contains 2,893 labeled messages in total, with 2,412 hams and 481 spams. Unlike Grumble dataset, this dataset is much longer in length for each message, with more than 500 words per message. Also, this dataset separate items, including punctuations, in each message with a single space. So connection between words are much less strong than it is in the Grumble dataset. There are four kinds of pre-processed forms of the same dataset, and in this work we use the one with lemmatiser enabled and stop-list enabled to avoid noises and keep the text clean.

As to these two datasets, to see how the spam filter performs under different training and testing ratio, I set the ration of training to testing from 1:9 to 9:1, from sparse training data to rich training data, and see how each spam filter performs.

In this work, I test two spam filters, one using LSTM and one using CNN. Both spam filters will use noise reduction module to further improve the performance, which will compare to the original performance and show in figures.

## 5.2. Parameters and Training

In LSTM text classifier model, I use 2 layers of LSTM cell and input batch size of 128. Embedding size k is set to be 128, and learning rate is 0.001. The model runs 20 epochs to make the loss converge.

In CNN text classifier model, I use convolutional kernel window size w to be 3,4,5 with 256 filters for each window size. The model is set with 128 dimensions of word embedding, with batch size equals 128 and runs 30 epochs. To avoid overfitting problem, the model uses L2 regularization with $\lambda = 0.1$.

As to baseline model, I use Naive Bayes as baseline. As a statistical method, Naive Bayes is a strong model with good performance under many circumstances. In parameter setting, I set default priori probability to be 0.5, and Out of Vocabulary(OOV) words probability to be 3.7e-4.

## 5.3. Results and Discussion

In experiments, in order to show the overall performance of different models, I user the training ratio to be the variable in each set of experiments, and will conduct the same experiment in two different datasets.

In the following experiments, we only use precision as the metric to evaluate the performance of spam filters, as $\text{Preicision} = \text{TP}/(\text{TP} + \text{FP})$, which can show the performance of models with true positive and false positive rate.

### 5.3.a. Grumble

The following figure 6 shows how three text classification models perform under different training ratio. Notice that none of the models in figure 6 has noise reduction module attached.

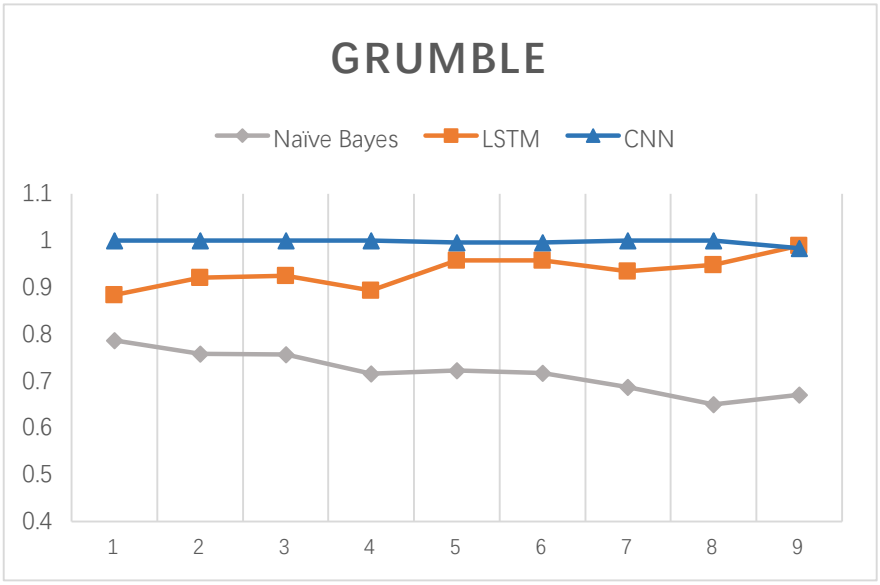


*Figure 6. Three models in Grumble [7]*

As we can see from figure 6, comparing three models, Naive Bayes has the worst performance with 70% precision in average, while LSTM is much better in precision and CNN performs even better, with most in more than 98% accuracy. And to the affection of training ratio increasing, to baseline model, the performance reaches the highest when training dataset is the smallest, and then the precision keep dropping to 67%. This means when training dataset becomes larger, the

assumption that words are independent in Naive Bayes let the model ignores many useful context information. And as some words can exists in both spam and ham, larger training dataset will introduce more noise to this model, and decrease the precision. However, with the increasing of the training dataset, as LSTM doesn't count on words, it can seize more context information and help the model to make more precise predictions. So to LSTM, it needs rich training data to train the model. CNN performs extremely well even with little training data. This is because the CNN model is based on both words information and context information. Extracting features using filter window helps the model to seize context information, and taking only the maximum value of features to count help the model focus only on important words instead of all words. So CNN can perform well under all training ratios.

Figure 7 shows the LSTM model without and with noise reduction model.

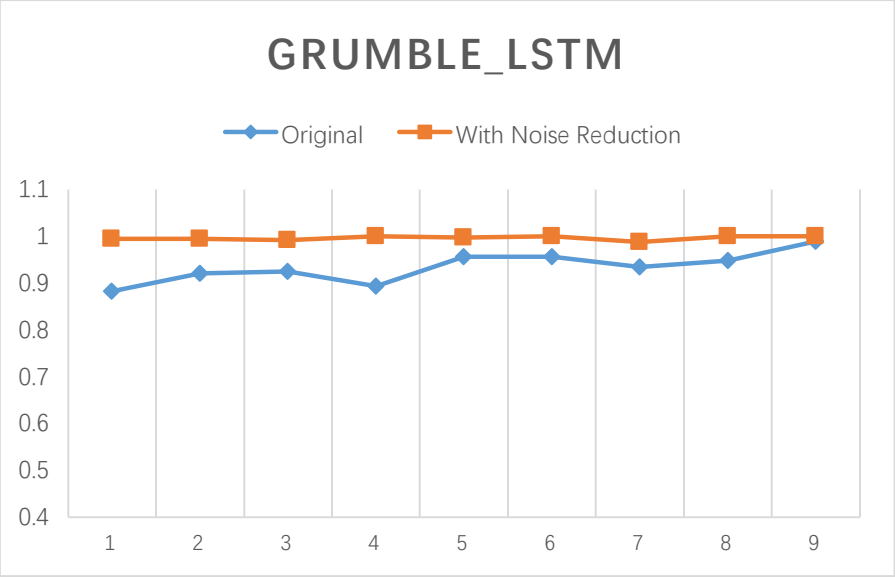


*Figure 7. LSTM original and with noise reduction in Grumble [8]*

We can see that noise reduction helps with the precision increment of LSTM model. Messages that are classified as spam can only change from False Positive(FP) to True Negative(TN) or True Positive(TP) to False Negative(FN). Increasing of precision means that mostly FP are changed to TN, which means hams that are misclassified as spams are set back to hams. There can be some noise introduced that makes a message from TP to FN, which means true spams are misclassified as hams, but this is with little probability and as mentioned before, people are more willing to handle spams manually than to miss important messages. The same situation happens to CNN, where introducing noise reduction module helps the spam filter to performs better, shown in figure 8. As the performance of original CNN is good enough, the two lines mostly overlapped together. This also shows that noise reduction will not introduce much noise to the spam filter model, otherwise TP will drop significantly and precision will also drop, which will hurt the result.

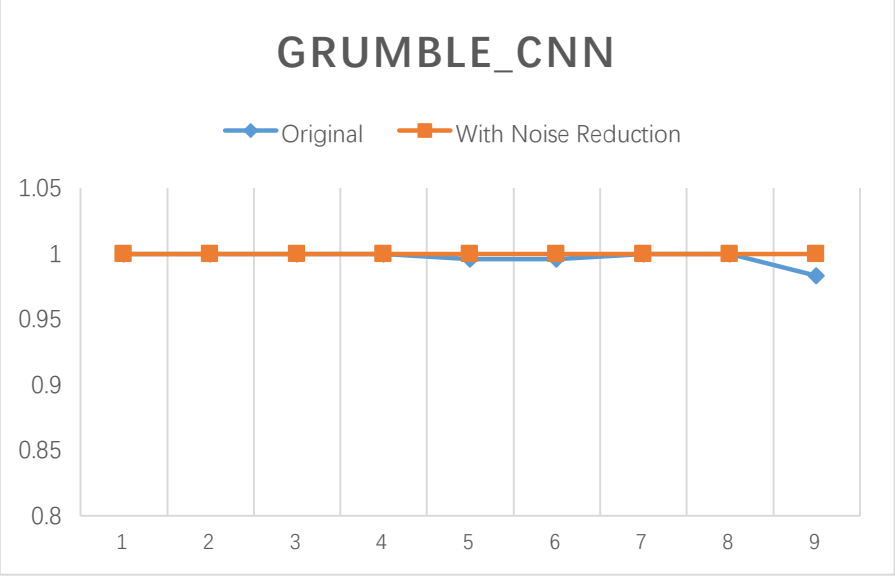


*Figure 8. CNN original and with noise reduction in Grumble [9]*

5.3.b. Ling-Spam

Figure 9 shows the precision of three models with no noise reduction using Ling-Spam. Other than using a different dataset, the meaning of figure 9 is the same as figure 6.

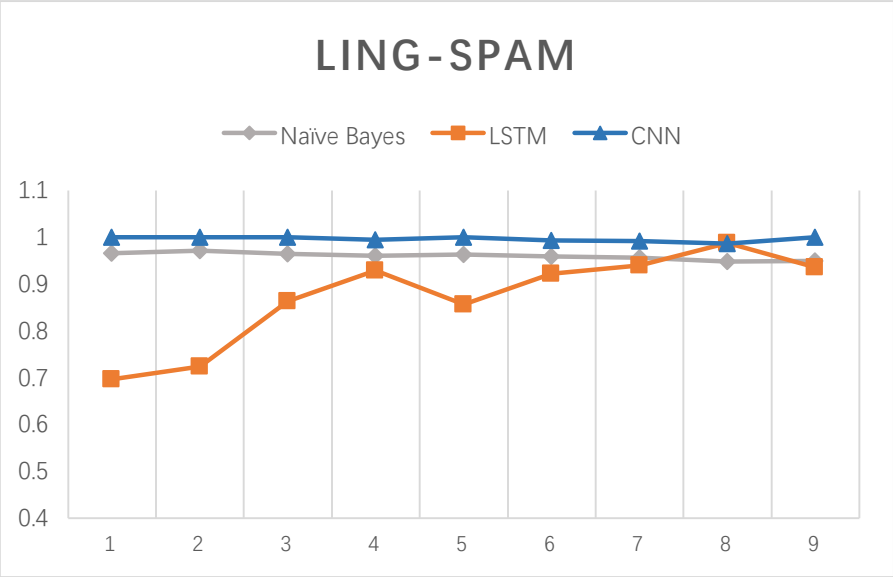


*Figure 9. Three models in Grumble [10]*

As we introduced previously, comparing to Grumble, Ling-Spam is a dataset with longer messages and weaker words connections. So we can see that baseline model performs well in this situation. This is because Naive Bayes consider each word to be independent, which is the situation here. With more training data, Naive Bayes keeps having a good performance with a little drop in precision, which means the noise introduced by new data is not too much compare to the information from new data.

To LSTM, the performance is not satisfying at first, as the context information is little and training data is small. But later with the increment of training data, the performance of LSTM gets

better and reaches more than 93% after we have 60% of training data. So even to dataset with weak context information, LSTM will perform well when there is rich training dataset. To CNN model, the precision of the model keeps higher than 98% from small to large training ratio. Because the nature of this model, it can handle more general datasets, with either strong or weak context information.

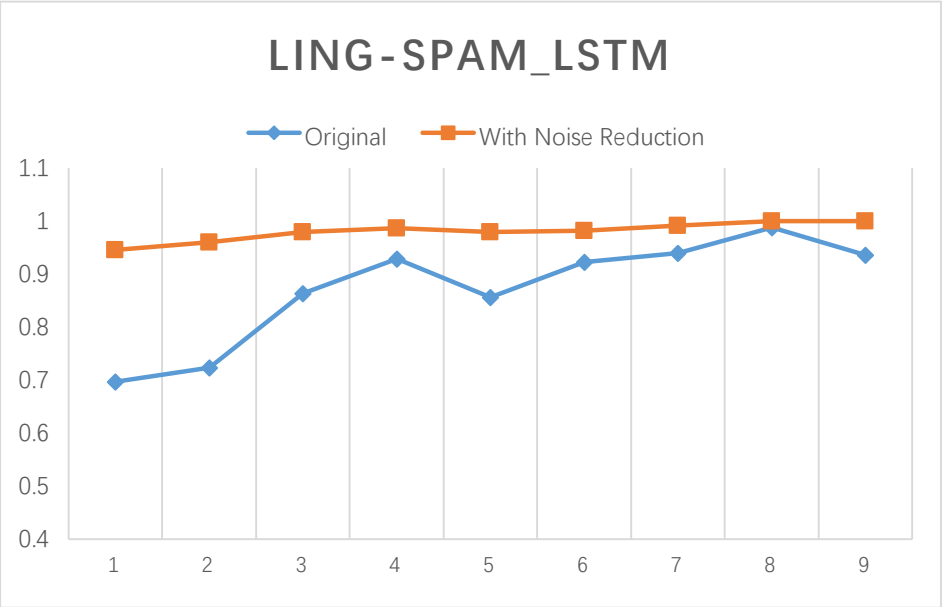


*Figure 10. LSTM original and with noise reduction in Ling-Spam [11]*

In figure 10, we can see that though LSTM doesn't have good precision when the training ratio was small, noise reduction module handles FP messages perfectly well. No matter what the model precision is, noise reduction can always process the messages that are classified as spam correctly.

Because CNN model can perform extremely well even without noise reduction module, the two lines, like in figure 8, overlaps together. This again proves that noise reduction module can only improve the precision of the model, without making much TP to FN.

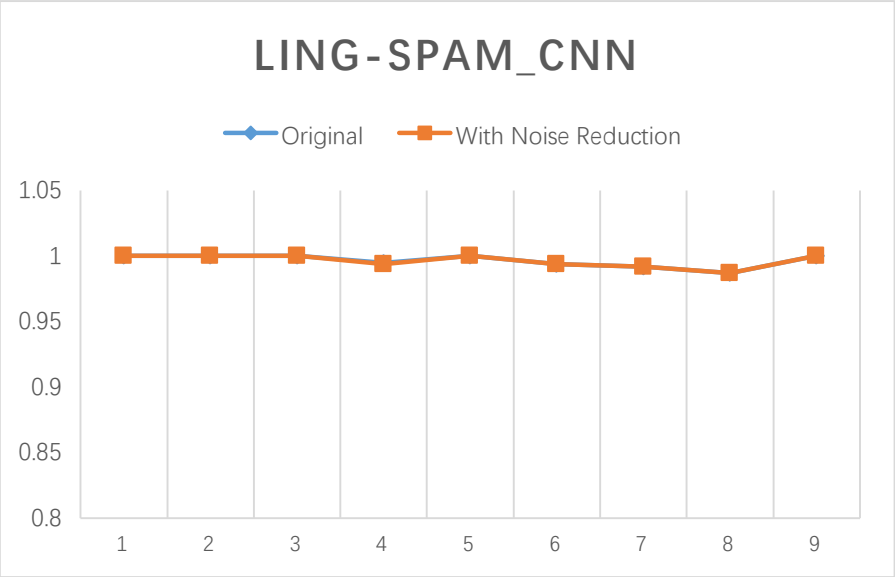


*Figure 11. CNN original and with noise reduction in Ling-Spam [12]*

From the overall experiments, we can say that building text classifier using LSTM and CNN model can help with the precision of spam filters, and introducing effective noise reduction module can definitely further improve the performance.

# 6. Future work

My approach of building LSTM only used two layers of LSTM cells. In the future work, this can be further improved by adding more layer of cells. Also, in the work of (Kim et al., 2014), the author also proposed a CNN-multichannel model, which combines two pre-trained set of word vectors. In this model, each set of vectors is treated as a channel and each filter is applied to both channels. From the results of that paper, this model can improve the performance of CNN model.

Also, currently the noise reduction model only adds up scores of matching results from different models separately and compare the two numbers. This is basically a weighted K-nearest neighbors (weighted KNN) approach. In the future work, we can develop more ways to explore the returned results to get even better noise reduction performance.

# 7. Conclusion

In this present work, I have built spam filters based on LSTM and CNN model with noise reduction, and conduct a series of experiments on these two models as well as on Naive Bayes models to see the performance. From the results, we can say that CNN model can give perfect performance with high precision under datasets with either sparse or rich context information, while LSTM prefers datasets with strong words connections. But LSTM can also performs well when the training data gets larger in weak context information dataset. Both models outperform Naive Bayes, which is a popular in traditional spam filters. With noise reduction model, spam filters can get further improved and the precision is remarkably high. Spam filter based on deep learning and information retrieval techniques can be the trend in the future.

# TABLE OF FIGURES

# BIBLIOGRAPHY


[1]. Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov, “Bag of Tricks for Efficient Text Classification”, *arXiv:1607.01759*, 2016.

[2]. Yoon Kim, “Convolutional Neural Networks for Sentence Classification”, *EMNLP 2014*, 2014.

[3]. I Androutsopoulos, G Sakkis, G Psaliouras, V Karkaletsis, CD Spyropoulos, “Learning to filter spam e-mail”, *European Conference on Workshop on Machine Learning*, 2000.

[4]. Mehran Sahami, Susan Dumais, David Heckerman, Eric Horvitz, “A Bayesian Approach to Filtering Junk E-Mail”, *Proc AAAI*, 1998.

[5]. Vangelis Metsis, Ion Androutsopoulos, Georgios Paliouras, “Spam Filtering with Naive Bayes – Which Naive Bayes?”, *CEAS 2006*, 2006.

[6]. Tianhao Sun, “Spam Filtering based on Naive Bayes Classification”, 2013.

[7]. M. Tariq Banday, Lifetime Member, “Effectiveness and Limitations of Statistical Spam Filters”, *International Conference on “New Trends in Statistics and Optimization”*, 2008.