COS30015:
Research Assignment

Effect of feature extraction in real-life spam classification situations

Dumindu Madithiyagasthenna 101596759

Submission: 27/10/2019 1pm
Due: 27/10/2019 5:30pm

Introduction

Classification is a typical problem within the world of machine learning, and it involves categorizing data into separate and distinct groups. Spam classification is very common within applied classification and is something that every person has experience with.

The research and the implementation discussed in this paper involves how the process of feature extraction and pre-processing of data can affect the performance of classification in the real-life situations.

***Note (i)**: there exist a multitude of different ML classification models that could possibly be better. But given the time, my expertise, and the scope of this assignment I have stuck to the more conventional and easier methods and avoided complicated methods such as deep neural networks and so on.*

***Note (ii)**: All of the implementation is available online and links to them have been provided at the end of this paper. The implementation was carried out using python functions within libraries, mainly Scikit-learn and Gensim.*

Motivation

Spam is very prevalent on the internet. A recent statistic claims that in March 2019, spam email was responsible 56 of all email-traffic(1). There is a lot being done to combat this but a lot of literature regarding spam classification, and classification in general talks about work done under ideal and very optimistic conditions. They usually have very well labelled training sample data available and have access to very balanced datasets(2).

But in the real-world when tackling situations such as this, it is often not the case. It is very hard to obtain pre-labelled high-quality data. And even then, the classes will usually be very unbalanced. Furthermore, it is difficult to get access to data that is relevant to the current time frame and the data it is going to be tested against.

Therefore, my efforts were put into investigating how, under such conditions which is much more common than the idealistic scenarios discussed in papers, can we improve the results of such classification.

A range of commonly used classifiers are compared under idealistic conditions and real-life extreme situations. Changes in feature extraction and data processing was carried out to find with combination of provides a solution most robust to such conditions.

Dataset

Spam

Spam is always evolving. Spam from the late 1990s are not the spam that we receive today. Scammers are getting craftier and their spam is getting harder to detect. For example, spam images have evolved from simple images to rotated images with speckled backgrounds and varying spacings and margins in text to throw off potential spam classifiers(3).

For this project, I have obtained 2 spam datasets, one from the year 2000, and another from 2019.

The year 2019 set is used to provide us a baseline under ideal conditions and the year 2000 set is used to simulate a real-life scenario where the data has drifted.

The datasets were obtained from Untroubled Software, which keeps an update to date record of raw spam emails over the years from 1997(4).

Raw emails which have not been pre-processed were used to control the pre-processing to see its effect on the final results.

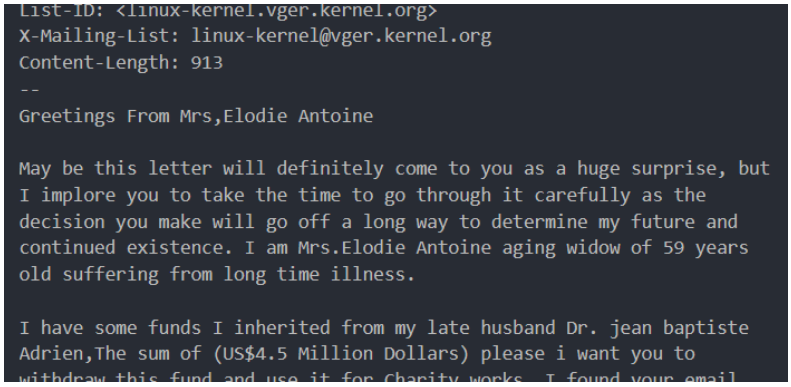

Figure 1Shown above is an image of a sample email from the spam dataset

Ham

Ham, or content which is not spam, has not evolved much though the years since it is contextual and personal. Therefore, we will be using just one dataset for ham in this research.

The data set was obtained from Enron, arguably one of the most famous email datasets, used widely in literature(2,5), which was released to the public domain after their court course by the Federal Energy Regulatory Commission (6).

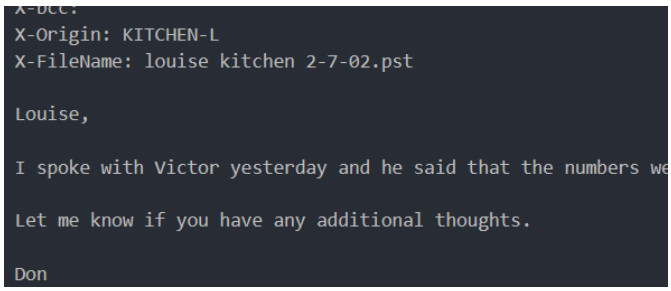

Figure 2 Shown above is an image of a sample email from the ham dataset

Pre-processing

Pre-processing is an important step used to make sure that the data used for this machine learning task is clean and standard. In many situations in the industry it is very hard to find pre-processed data therefore we will do our own.

We will be using few different pre-processing techniques:

Stop words

Stop words are words that are used commonly within a language. In English for example these words include words such as a, the, in, at, etc.

As part of the pre-processing step we will ignore these words by stripping them away to make sure that they do not affect our results since they do not provide an information useful for our classification.

Stripping Email Headers

The emails in the dataset have not been processed and have been exported directly from a mail server. Therefore, all these email data have the mail headers which includes various metadata relating to the email such as email addresses, authentication, etc.

Almost all of these are irrelevant to the content within the emails. Therefore, they will be taken out of this dataset sine they can also skew the results as stop words could.

HTML Decoding

Emails are usually encoded in HTML format. Html tags such as (such as , <a>, etc.) are not relevant to our content and pose the same issue as stop words and headers do, so they also will be stripped from the email. The effect of doing this will be discussed further later on in the paper.

Lemmatisation

Lemmatisation is another common pre-processing technique used to standardise the text within the dataset. Words can have different forms that they can take based on their context and grammatical sense. For example, the word “push” could take the form of pushed, pushing, pushes, etc. Lemmatisation refers to the process of converting this to the base form (in this case “push”) of 1 single word.

(This is similar to stemming but lemmatisation takes into account the surrounding context of the word.)

Other pre-processing

To standardise the text within these emails even further, the text is then converted to lower case and certain useless characters are then removed, such as numbers punctuation, encodings, etc.

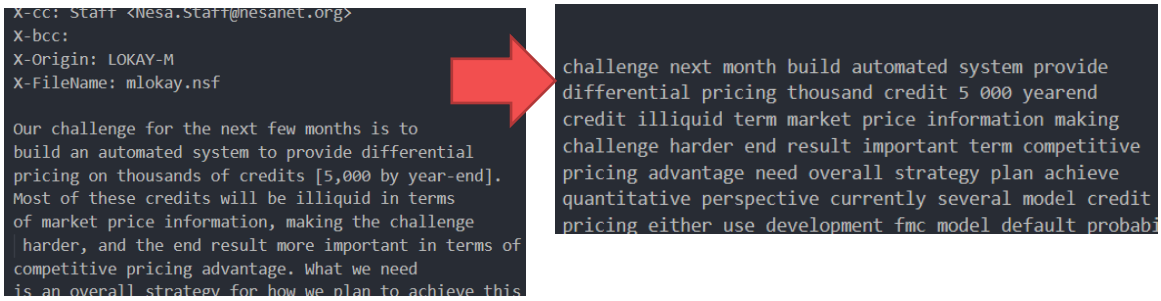

Figure 3 Above shows a sample email snippet before our pre-processing procedure (left) and after (right).

The code for this process is hosted on google colab:
https://colab.research.google.com/drive/1XHToq9ydufqVaAtXA4cNXmcFmNczIpqP
And can also be found in the git repository.

Idea Scenario

For mimicking an ideal scenario, usually seen in machine learning blogs and some literature we will be training and testing 4 different classifiers against data which is equally balanced

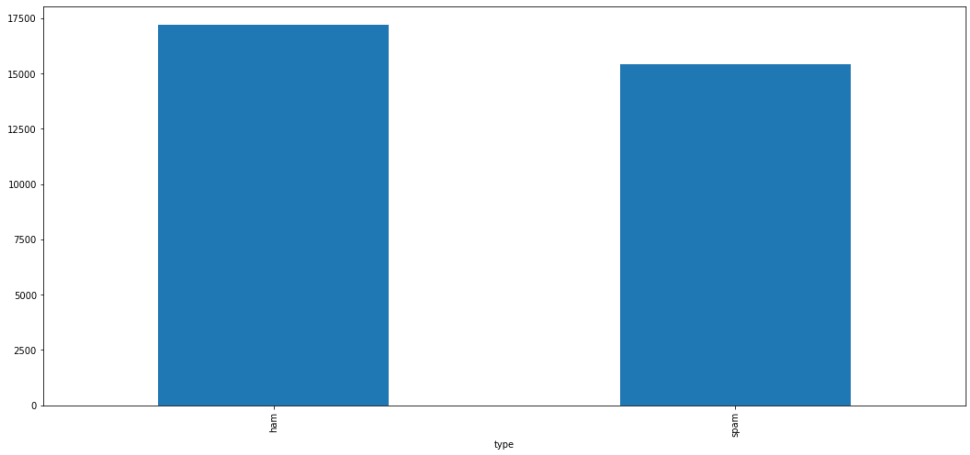

Figure 4 The spam vs ham classes shown are nearly perfectly balanced (53% vs 47%)

The spam data is from 2019 and is relevant to this current time frame, and it is split between the training and testing. The training and testing will be split in the conventional 80/20 split.

In literature surrounding spam classification, two very commonly seen classifiers are the Naïve Bayes and the Support Vector Machine (SVC) classifier; and they have been proven to be very good in this regard (2,7). But outside the scope of spam, classifiers like Logistic Regression and Random Forrest Regression have been used widely. Therefore, we will be using all 4 of them to simulate this ideal training scenario.

In machine learning the most basic used technique of extracting features out of text is to use a Bag of Words(BoW)approach. And the simplest BoW method is one that simply uses the frequency of words within the dataset as features. This is used as our baseline feature extraction method.

The data is pre-processed, their features extracted using the simple BOW method and have them run through all the 4 classifiers. The results of them are shown below.

Classifier	Precision		Recall		F1-score	
	ham	spam	ham	spam	ham	spam
MultinomialNB	0.96	0.99	0.99	0.95	0.97	0.97
Linear SVC	0.99	0.98	0.98	0.99	0.99	0.99
LogisticRegression	1	0.99	0.99	1	1	1
RandomForest	0.64	1	1	0.37	0.78	0.58

Table 1 Shows the results after running the 4 classifiers under our ideal situation.

It is apparent that the the f1 scores of most of these classifiers are very high, and classifiers such as Logistic Regression is nearly perfect. This can be credited to the fact this is a very idealistic situation.

***Note**: Multinomial Naive Bayes was the choice of Naïve Bayes classifier and the Linear SVC method was the chosen SVC method, purely out of the ease of implementation using sckit-learn.*

The code for this process is hosted on google colab:
https://colab.research.google.com/drive/1EIzv4woZvP0RroWGeASfvzKT5BDFg-lU
And can also be found in the git repository.

Real Scenario

Many situations are not like the one we discussed above. In real life data classification scenarios, we will be extreme cases. To simulate this, we will be using spam from the year 2000 for training and they will be tested against current 2019 spam. The classes will be also be highly unbalanced.

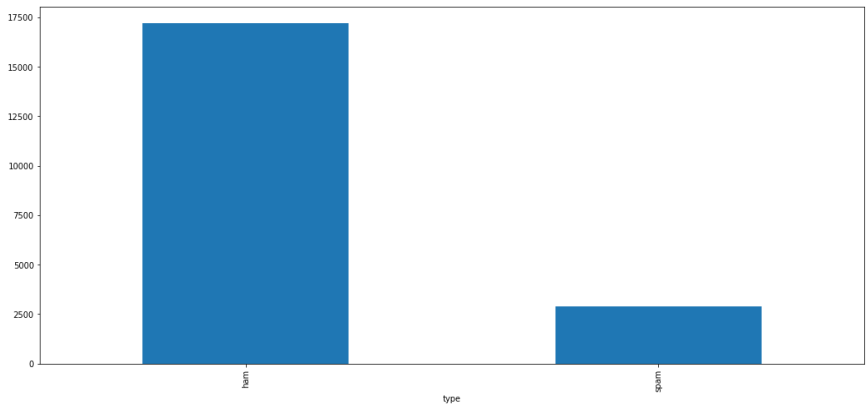

Figure 5 Classes of spam and ham of training data high skewed (86% to 14%)

To keep our comparison standard, we will use the same 4 classifiers as before. The data will be pre-processed using the same methods and the same feature extraction method (simple BoW) will be used.

Classifier	Precision		Recall		F1-score	
	ham	spam	ham	spam	ham	spam
MultinomialNB	0.7	0.98	0.99	0.52	0.82	0.68
Linear SVC	0.73	1	1	0.6	0.85	0.75
LogisticRegression	0.7	1	1	0.51	0.82	0.68
RandomForest	0.53	0	1	0	0.69	0

Figure 6 Results after running the classifiers using our worst-case scenario, leaving the other variables consistent

As you can see below, the results have dropped significantly as expected. The f1 scores have dropped by an average of 27% with our best classifier from before (LR) dropping by 25%.

What does this mean in the real world? This relates to spam emails coming straight to your inbox without getting filtered out. Or more dangerously, an important email getting put in the spam folder. Below we will discuss how different feature extraction methods may affect these results and if they can improve these results.

The code for this process is hosted on google colab:
https://colab.research.google.com/drive/10HGOYIeO0obaMnuZcdYMvRVn7aoaYqXh
And can also be found in the git repository.

Effect on feature extraction

There is a plethora of feature-extraction methods in the world of machine learning. But I will be focussing on 3 of them; Word2vec, Doc2vec, and Tf-idf.

TF-IDF (Term frequency–inverse document frequency)

This is also a BoW model. But instead of simply getting the count of words it uses a very different approach. Counting a be missing leading sometimes since words that occur more in a document would not necessarily mean that they are more relevant.

Tf-idf uses an inverse model which increases the value of a word depending on how much it occurs in a document and how less it occurs in other documents. This gives certain words uniqueness that relates to its document.

Let us see if this had any effect when running through our 4 classifiers

Classifier	Precision		Recall		F1-score	
	ham	spam	ham	spam	ham	spam
MultinomialNB	0.63	1	1	0.33	0.77	0.5
BernoulliNB	0.65	0.99	1	0.4	0.79	0.57
Linear SVC	0.68	1	1	0.49	0.81	0.66
LogisticRegression	0.72	1	1	0.56	0.84	0.72
RandomForest	0.53	0	1	0	0.69	0

Table 2 Results after extractiing features through the tf-idf method

The results have slightly improved for the Logistic Regression classifier. And can be attributed to the inverses effect of Tf-idf.

The code for this process is hosted on google colab:
https://colab.research.google.com/drive/1dLT_P3RP5KCn-iuAtjXCGoqgdSsz9AY2
And can also be found in the git repository.

Word2Vec

Word2vec is a popular pretrained word embedding model that vectorizes words based on context. For our Word2vec model I am using a model trained on 100 billion words from google news (8).

Classifier	Precision		Recall		F1-score	
	ham	spam	ham	spam	ham	spam
BernoulliNB	0.62	0.78	0.9	0.39	0.74	0.52
Linear SVC	0.69	0.98	0.99	0.52	0.82	0.68
LogisticRegression	0.71	0.98	0.99	0.55	0.83	0.71
RandomForest	0.53	0.4	1	0	0.69	0

Table 3 The results after the word2vec feature extraction

The results show an overall decrease in the results but an increase for the Logistic Regression classifier as with Tf-idf. This could be due to the nature of the Word2vec not fitting with the context of emails or since we only used a part of the model.

***Note**: that Multinomial Naïve Bayes method could not be used for word2vec extraction and doc2vec extraction*

The code for this process is hosted on google colab:
https://colab.research.google.com/drive/1xWYmpMiTFbjA6rJvWLSTsRpNnrUqdZK-
And can also be found in the git repository.

Doc2Vec

Doc2vec is very similar to the word2vec model but instead of finding relations between words it works with whole sentences, paragraphs and/or document. We are using the Doc2vec model provided by Gensim(9)

Classifier	Precision		Recall		F1-score	
	ham	spam	ham	spam	ham	spam
BernoulliNB	0.56	0.97	1	0.14	0.72	0.25
Linear SVC	0.56	0.98	1	0.13	0.72	0.24
LogisticRegression	0.57	0.92	0.99	0.17	0.72	0.28
RandomForest	0.53	0.98	1	0.01	0.69	0.03

Table 4 Results after doc2vec feature extraction method

After running this through our 4 classifiers the results also show an overall decrease as well. This could be due to the nature of the Doc2vec model not fitting with the context of emails.

The code for this process is hosted on google colab:
https://colab.research.google.com/drive/1Bf3N6oUW8_gXRQVXUpWjmlVpiEOTOyef
And can also be found in the git repository.

Effect on pre-processing

Pre-processing the data is an important step and over processing can lead to the loss of important features. As discussed before, we striped away all html tags and numbers. But these could point to some interesting features. Images and links embedded in html tags and mobile numbers can be useful features for when identifying spam. Because by experience, spam emails tend to have way more images, links and telephone numbers.

Let's go back to pre-processing step and make some changes:

- replace html link tags (`<a>`) with the word "link",
- image tags (`<img>`) with the word "image"
- image tags which have links hidden in them (eg: `<a href='..' <img .. ></a>`) as "image_with_link".
- mobile numbers will be replaced by the word "tele_number".

Following this, we will rerun the 4 classifiers with our feature extraction methods that improved the results (Tf-idf and Word2vec) and see the effects.

The code for this process is hosted on google colab:
https://colab.research.google.com/drive/10YT2DrTU2BQpBRA0PjGl90M6oQ_QBUdm
And can also be found in the git repository.

Word2Vec:

Classifier	Precision		Recall		F1-score	
	ham	spam	ham	spam	ham	spam
BernoulliNB	0.57	0.69	0.9	0.25	0.7	0.37
Linear SVC	0.83	0.99	0.99	0.78	0.91	0.87
LogisticRegression	0.86	0.99	0.99	0.92	0.89	0.96
RandomForest	0.53	0.29	1	0	0.69	0

Table 5 Results after the updated data pre-processing technique using the word2vec feature exytraction method

The results show a very large increase in the F1-scores, specifically with the SVC classifier and the Logistic Regression classifier.

TF-IDF:

Classifier	Precision		Recall		F1-score	
	ham	spam	ham	spam	ham	spam
MultinomialNB	0.72	1	1	0.58	0.84	0.73
BernoulliNB	0.53	0.87	1	0.03	0.7	0.07
Linear SVC	0.9	1	1	0.88	0.95	0.93
LogisticRegression	0.94	1	1	0.93	0.97	0.96
RandomForest	0.53	0	1	0	0.69	0

Table 6 Results after the updated data pre-processing technique using the tf-idf feature exytraction method

The results are very similar to that of word2vec, with a large increase in performance.

The code to these results is hosted on google colab:
https://colab.research.google.com/drive/1YfHNrfz2835oA6KIcFh_U3qLGBABKcjy
And can also be found in the git repository.

Conclusion

From the data gathered from this research we can clearly see the effect of relevant features in the robustness of classifiers. The feature extraction methods increased the performance of some models slightly, but what was more important was to have these features ready during the data pre-processing stage to make sure that they actually can be extracted.

When identifying these features, it is important to have domain knowledge of the topic. For this, I used my personal experience to identify that spam emails contain more links, images and telephone numbers on average than normal ham emails. And by stemming them (with standard names such as link, image, tele_number, etc..) and including this data in the dataset improved the performance very largely. The logistic regression classifier was up by 28.6% and the SVC classifier was up by 17.5% (when using Tf-idf method). Of course, this is just the starting point, more features can be obtained to make these results even better, such as including specific meta data from the mail headers such as sender country, smtp headers, etc(2).

As much as this paper discusses different feature extraction methods and data processing techniques, it also points to the difficulties and struggles of classification of data in the real world. With scammers increasing spam each day, many people rely on classification like this to be safe and know what is legitimate from what is a scam.

From my implementations and research, I have gathered that under extreme situations like the one I have simulated, the easiest and most robust way to train a spam email classifier is to use the logistic regression classifier with Tf-idf as the feature extraction method. And to standardise and include relevant features such as telephone numbers, and html entities such as images and links during the cleaning of the data.

Improvements and limitations

There are many other ways of extracting and classifying data such as deep neural networks and so on. These state-of-the-art methods could also provide with better results and provide more robust solutions.

And as mentioned before, adding more relevant features from the headers could also increase the results.

Resources

Link to Git repository that contains all code:

- https://github.com/DumiM/spam_filtering

Link to individual Jupyter notebooks that's hosted on Google colab (these are also in the repo):

- Initial pre-processing:
  - https://colab.research.google.com/drive/1XHToq9ydufqVaAtXA4cNXmcFmNczIpqP
- Idea Scenario:
  - https://colab.research.google.com/drive/1EIzv4woZvP0RroWGeASfvzKT5BDFg-lU
- Real-life Scenario (BoW):
  - https://colab.research.google.com/drive/10HGOYIeO0obaMnuZcdYMvRVn7aoaYqXh
- Real-life Scenario (Td-idf):
  - https://colab.research.google.com/drive/1dLT_P3RP5KCn-iuAtjXCGoqgdSsz9AY2
- Real-life Scenario (Word2vec):
  - https://colab.research.google.com/drive/1xWYmpMiTFbjA6rJvWLSTsRpNnrUqdZK-

- Real-life Scenario (Doc2vec):
  - https://colab.research.google.com/drive/1Bf3N6oUW8_gXRQVXUpWjmlVpiEOTOyef
- Re-processing of data to include more features:
  - https://colab.research.google.com/drive/10YT2DrTU2BQpBRA0PjGl90M6oQ_QBUdm
- Results using these new features (Tf-idf & Word2vec):
  - https://colab.research.google.com/drive/1YfHNrfz2835oA6KIcFh_U3qLGBABKcjy

References

1. Spam statistics: spam e-mail traffic share 2019 [Internet]. Statista. [cited 2019 Oct 26]. Available from: https://www.statista.com/statistics/420391/spam-email-traffic-share/

2. Dada EG, Bassi JS, Chiroma H, Abdulhamid SM, Adetunmbi AO, Ajibuwa OE. Machine learning for email spam filtering: review, approaches and open research problems. Heliyon. 2019 Jun 1;5(6):e01802.

3. Yan Gao, Ming Yang, Xiaonan Zhao, Bryan Pardo, Ying Wu, Pappas TN, et al. Image spam hunter. In: 2008 IEEE International Conference on Acoustics, Speech and Signal Processing. 2008. p. 1765–8.

4. SPAM Archive [Internet]. [cited 2019 Oct 26]. Available from: http://untroubled.org/spam/

5. Koprinska I, Poon J, Clark J, Chan J. Learning to classify e-mail. Information Sciences. 2007 May 15;177(10):2167–87.

6. Enron Email Dataset [Internet]. [cited 2019 Oct 26]. Available from: https://www.cs.cmu.edu/~enron/

7. W.A A, S.M El. Machine Learning Methods for Spam E-Mail Classification. International Journal of Computer Science & Information Technology. 2011 Feb 28;3.

8. Google Code Archive - Long-term storage for Google Code Project Hosting. [Internet]. [cited 2019 Oct 26]. Available from: https://code.google.com/archive/p/word2vec/

9. gensim: topic modelling for humans [Internet]. [cited 2019 Oct 26]. Available from: https://radimrehurek.com/gensim/models/doc2vec.html