Spam SMS Classifier

Arpan Khanna arpan.khanna@iitg.ac.in
Dheeraj Garg g.dheeraj@iitg.ac.in
Agney S Talwarr t.agney@iitg.ac.in
Lovish Aggarwal a.lovish@iitg.ac.in

Abstract

The growth of mobile phone users has led to a dramatic increase in SMS spam messages. One of the most widely used services in mobile networks is short message service. It offers dependable, personalised service, exceptional secrecy, and a high response rate. Spam SMS, also known as unsolicited SMS, may cause several issues for mobile users as a result. One of the major issues in internet and wireless network is to recognise such spam messages.
In this project, we'll build a classifier that divides SMS messages into spam and ham categories. The SMS data is very noisy. This pose a question to the ability of our model to correctly classify the SMS into spam and ham. Hence, this project's primary goals were to eliminate the noise—unstructured text data—and to analyse our model's performance both before and after the noise was eliminated.

1 Introduction

Over the past decade, the consequent evolution of the spam miracle, namely the mass transmission of spontaneous messages, has become a major theme of SMS utility for Internet Specialists (ISP) collaborations, corporate and private customers. Later checks revealed that more than 60 percent of all SMS movements are spam. Spam causes SMS frameworks to be overwhelmed in terms of transmission speed and server storage limitations, increasing the annual cost of partnerships by more than billions of dollars. In addition, phishing spam messages pose a real threat to the security of end customers as they attempt to trick them into revealing personal information such as passwords and record numbers with spoof messages claiming to come from trusted online organizations, such as household organisations move.

Every time a person receives an SMS spam message, their mobile phone notifies them of the message's arrival. The consumer will be unhappy when they learn the message is undesired, and SMS spam uses up some of the storage space on their mobile device.

SMS spam detection is a crucial operation that involves identifying and filtering spam SMS messages. It might be difficult for a user to recall and correlate recently received SMS messages with SMS that have already been received, as more and more SMS messages are sent daily. We will thus aim to construct an SMS text spam or ham detector utilising our understanding of natural language processing together with a combination of machine learning and data analysis.

Numerous variables, including the limited character limit for text messages, the use of new abbreviations, the use of non-standard orthographic forms, phonetic replacement, etc., have contributed to the enormous rise of noise in text. More issues will arise as a result of user creativity and linguistic individuality, such as ill-formed text that is frequently inappropriate as data for NLP (Natural Language Processing) activities and the gathering of unique text syntax that causes challenges for text processing.

The deviations present in the SMS are hampering for any standard natural language processing (NLP) system. Thus our main focus of the project is to remove the above mentioned problems and to compare the performance of our model both prior and after remove of the noise.

2 Methods

We have used the dataset by UCI Machine Learning from Kaggle[1] for the SMS messages.
Our proposed noise removal strategy involves three

[1]The dataset for SMS is available at https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

general steps: (1) confusion set generation, where we identify normalisation candidates for a given word; (2) ill-formed word identification, where we classify a word as being ill-formed or not, relative to its confusion set; and (3) candidate selection, where we select the standard form for tokens which have been classified as being ill formed
We also use an un-noisy dataset[2] in order to build our vocabulary and construct the bigram model that we use for context inference.

2.1 Confusion Set Generation

In this, we generate a set of IV normalisation candidates for each OOV word type based on morphophonemic variation.
First, any repetitions of more than 3 letters are reduced back to 3 letters.(e.g. cooool is redeuced to coool).
Second, IV words within a threshold T_c character edit distance of the given OOV word are calculated.
Third, the double metaphone algorithm (Philips, 2000) is used to decode the pronunciation of all IV words, and IV words within a threshold T_p edit distance of the given OOV word under phonemic transcription, are included in the confusion set. We set $T_c \leq 2$ and $T_p \leq 1$ (Han and Baldwin, 2011).
Examples of ill-formed words where we are unable to generate the standard lexical form are clippings such as fav "favourite" and convo "conversation". conversation".
In addition to generating the confusion set, we rank the candidates based on a bigram language model trained over clean data, i.e. text which consist of all IV words and pick the top 10% of candidates to get the reduced set.

2.2 Ill formed word detection

The next step is to detect whether a given OOV word in context is actually an ill formed word or not. Analysis reveals that most OOV words observed in SMS data are personal names (Han and Baldwin, 2011). Hence we use a dataset consisting of common names to decide whether a word is OOV(but properly formed) or ill formed.

2.3 Candidate Selection

For OOV words which are predicted to be ill formed, we select the most likely candidate from the confusion set as the basis of normalisation. We consider the following factors: lexical edit distance, phonetic edit distance, prefix substring length, suffix substring length and longest common subsequence length to capture morphophenic similarity. For context inference we have used a bigram model.
Both lexical and phonemic edit distance (ED) are normalised by the reciprocal of exp(ED). The prefix and suffix features are intended to capture the fact that leading and trailing characters are frequently dropped from words, e.g. in cases such as *ish* and *talkin*. We calculate the ratio of the LCS over the maximum string length between ill-formed word and the candidate, since the ill-formed word can be either longer or shorter than (or the same size as) the standard form. For example, *mve* can be restored to either *me* or *move*, depending on context. After normalizing these values we combine them to get the final score. On the basis of this score we will decide the suitable candidate which is the one having maximum score.

Using the techniques mentioned above we remove the noise from the SMS messages and then proceed with feature engineering, pre processing, model training and evaluation.

3 Experiment

We shall define the flow of the experiment. We shall do the EDA of the dataset, then we shall remove noise from the data. After this we plan to perform some feature engineering, then we clean our dataset and train it on different models.

3.1 Exploratory Data Analysis

Exploratory data analysis (EDA) is an approach of analyzing data sets to summarize their main characteristics, often using statistical graphics and other data visualization methods. We have checked our dataset for any anomalies like NaN values. We have further labeled the SPAM and HAM messages and plotted countplot for SPAM vs HAM.

SPAM	HAM
4825	747

Clearly the dataset is imbalanced and we will fix this using oversampling.
We also analyse the un-noisy dataset we use to construct the vocabulary and the bigram model.

[2]The un-noisy dataset at https://raw.githubusercontent.com/Phylliida/Dialogue-Datasets/master/MovieCorpus.txt

Total words	Unique words	Total sentences
1,39,538	10,563	15,944

The following figures shows the frequency distribution of unigrams and bigrams:

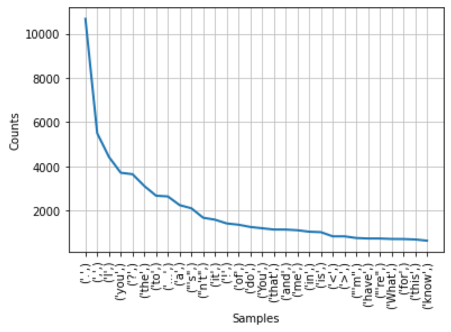


Figure 1: Unigram Frequency Plot

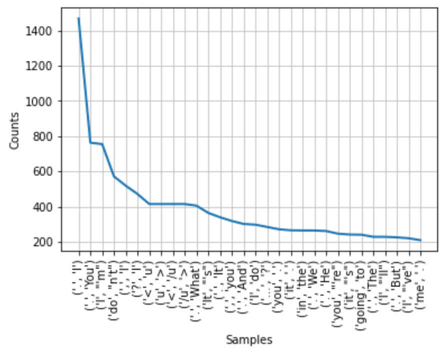


Figure 2: Bigram Frequency Plot

We observe that most of the frequently occuring words are punctuation. Later, we remove the punctuation and add <*unk*> after any sentence when it is over.

3.2 Noise Analysis

Here we remove noise from the dataset. As described in methods we select various candidates for OOV word which is identified using the vocabulary containing **10563** words.

After generating suitable candidate of an OOV word we identify whether the OOV word is an ill formed word or not. We identify this using a dataset[3] which contains the list of name of people. If OOV word is present in the names list then OOV word is not marked as an ill formed word otherwise we mark that as an ill formed word. The dataset consisted of **6485** names.

After identification of an ill formed word we select the most suitable candidate for ill formed word using various techniques which are: lexical edit distance, prefix substring, suffix substring and longest common subsequence for morphological errors, phonemic edit distance for phonemic errors and the bigram model for context matching.

We calculate the F1-score of various models using combination of two features at a time. The F1-score calculated using various combination of feature are given below:

Table 1: Correcting Morphological and Phonetic errors

F1-Score	
Multinomial Naive Bayes	0.942
Decision Tree	0.972
Random Forest	0.992
Voting (Multinomial + Decision Tree)	0.978

Table 2: Correcting Morphological errors and Using Context Matching

F1-Score	
Multinomial Naive Bayes	0.941
Decision Tree	0.975
Random Forest	0.991
Voting (Multinomial + Decision Tree)	0.979

Table 3: Correcting Phonetic errors and Using Context Matching

F1-Score	
Multinomial Naive Bayes	0.940
Decision Tree	0.973
Random Forest	0.991
Voting (Multinomial + Decision Tree)	0.979

3.3 Feature Engineering

- **Imbalanced Dataset & Oversampling**: Data imbalance usually reflects an unequal distribution of classes within a dataset. Imbalanced dataset poses a challenge for predictive modeling as most of the machine learning algorithms used for classification were designed around the assumption of an equal number of examples for each class. This results in models that have poor predictive performance, specifically for the minority class. This is a

[3]The dataset of names at https://www.kaggle.com/datasets/ananysharma/indian-names-dataset

problem because typically, the minority class is more important and therefore the problem is more sensitive to classification errors for the minority class than the majority class.
Our dataset consists of **5572** messages out of which **4825** messsages are Ham and the rest **747** messages are Spam which means the majority class contains about **86%** percent of the total messages. Clearly our dataset is imbalanced, so we have used **oversampling** to handle that.
Oversampling involves introducing a bias to select more samples from one class than from another, to compensate for an imbalance that is either already present in the data. Here, a bias is introduced by duplicating the spam messages.

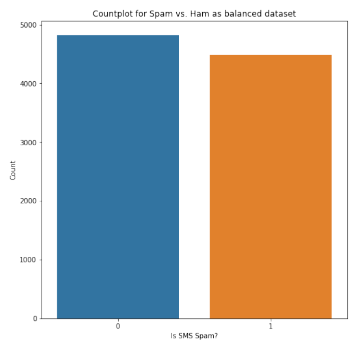


Figure 3: Dataset after Oversampling

- **Adding new Features**: A feature is any measurable input that can be used in a predictive model. In order to make machine learning work well on new tasks, it might be necessary to design and train better features. Thus, we have created new features like word_count, contains_currency_symbol and contains_numbers.

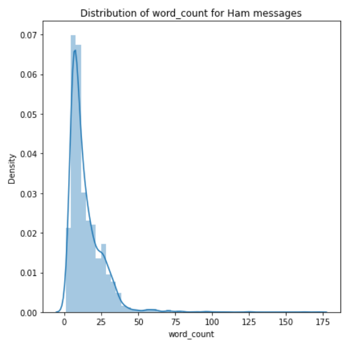


Figure 4: Distribution of word count for Ham

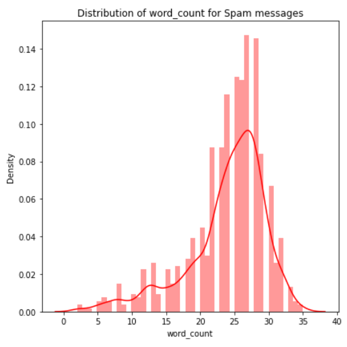


Figure 5: Distribution of word count for Spam

Using Fig 4 and Fig 5, we can infer that Spam messages word_count fall in the range of 15-30 words, whereas majority of the Ham messages fall in the range of below 25 words.

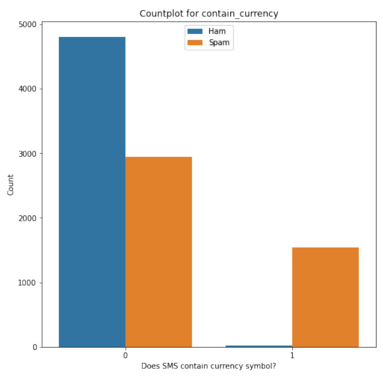


Figure 6: Plot for contains_currency

Using Fig 6, we can say that almost 1/3 of SAPM messages contain currency symbols, and currency symbols are rarely used in HAM messages.

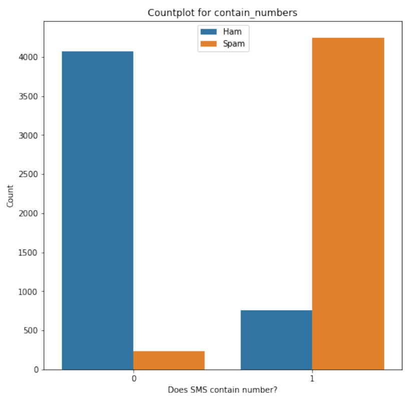


Figure 7: Plot for contains_numbers

It is evident from Fig 7 that most of the SPAM messages contain numbers, and majority of the HAM messages do not contain numbers.

3.4 Pre-Processing

Since data usually comes from a variety of sources and often in different formats, processing the text data is the first step in our solution which involves cleaning the raw data and tokenizing cleaned data. Data cleaning is done by:

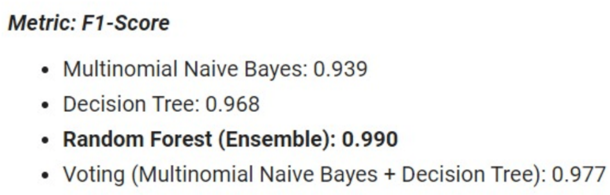
- Removing special character and numbers using regular expression
- Converting the entire SMS into lower case
- Tokenizing the SMS by words
- Removing the stop words
- Lemmatizing the words
- Joining the lemmatized words
- Building a corpus of messages

3.5 Model

We have selected four Machine Learning algorithms namely, Multinomial Naive Bayes (MNB), Decision Tree Classification Algorithm, Random Forest Algorithm and Voting Classifier. We have used the sklearn library to implement these models. Decision Tree and MNB algorithms are selected and fed to the Voting algorithm. In order to make the most economical use of the available data, we have used a 10-fold cross-validation (CV) whereby the dataset is partitioned to 10 folds. We ran the CV performance metric for all the four models and reported the average F1-score. The best performing model is chosen for classifying input messages into HAM and SPAM.

4 Results

We report our baseline performance without any noise cleaning in Fig 8.

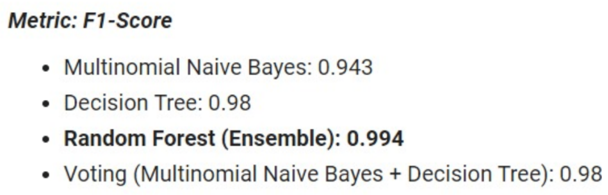


Metric: F1-Score

- Multinomial Naive Bayes: 0.939
- Decision Tree: 0.968
- **Random Forest (Ensemble): 0.990**
- Voting (Multinomial Naive Bayes + Decision Tree): 0.977

Figure 8: Before Noise Cleaning

We then show the performance of our models after noise cleaning in Fig 9. It can be seen that our models perform better after the noise cleaning. The percentage increase in F1-score is 0.43 for Multinomial Naive Bayes, 1.24 for Decision Tree, 0.4 for Random Forest Algorithm and 0.3 for Voting Algorithm.



Metric: F1-Score

- Multinomial Naive Bayes: 0.943
- Decision Tree: 0.98
- **Random Forest (Ensemble): 0.994**
- Voting (Multinomial Naive Bayes + Decision Tree): 0.98

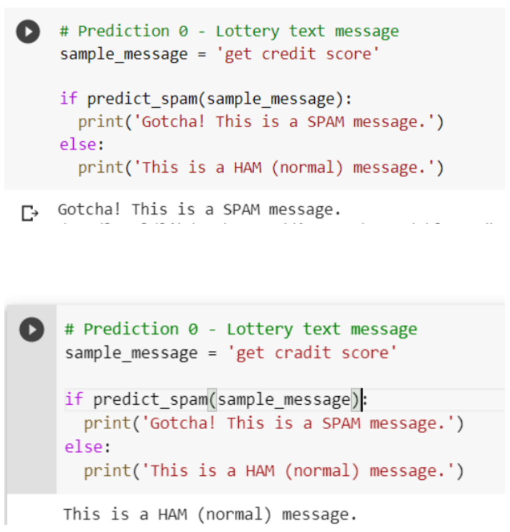
Figure 9: After Noise Cleaning

Random Forest Algorithm is the best performing model as it has the maximum F1-score. Therefore, we use this model for classifying any given input message into HAM and SPAM.

Here is an example showing the effect of Noise Cleaning on output.

The sample message is "*get credit score*" which is classified as a SPAM message. If we introduce a little noise in the message by replacing *credit* with *cradit*, then since there is no noise cleaning (in Fig 10), it is classified as a HAM message which is incorrect.

Now if we do noise cleaning, then we can see (in Fig 11) that the message "*get cradit score*" is also classified as a SPAM message.
Hence our Noise Cleaning method is working properly.



```
# Prediction 0 - Lottery text message
sample_message = 'get credit score'

if predict_spam(sample_message):
  print('Gotcha! This is a SPAM message.')
else:
  print('This is a HAM (normal) message.')
```

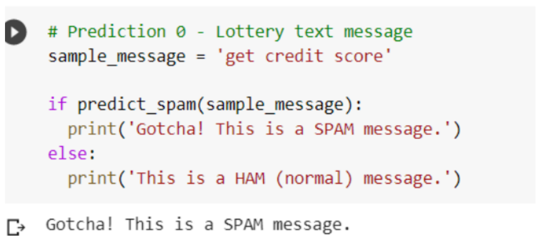
```
Gotcha! This is a SPAM message.
```

```
# Prediction 0 - Lottery text message
sample_message = 'get cradit score'

if predict_spam(sample_message):
  print('Gotcha! This is a SPAM message.')
else:
  print('This is a HAM (normal) message.')
```

```
This is a HAM (normal) message.
```

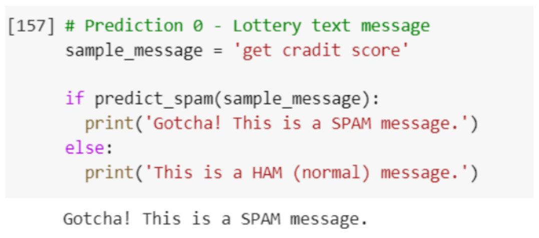
Figure 10: Before Noise Cleaning



```
# Prediction 0 - Lottery text message
sample_message = 'get credit score'

if predict_spam(sample_message):
  print('Gotcha! This is a SPAM message.')
else:
  print('This is a HAM (normal) message.')
```

```
Gotcha! This is a SPAM message.
```



```
[157] # Prediction 0 - Lottery text message
      sample_message = 'get cradit score'

      if predict_spam(sample_message):
        print('Gotcha! This is a SPAM message.')
      else:
        print('This is a HAM (normal) message.')
```

```
Gotcha! This is a SPAM message.
```

Figure 11: After Noise Cleaning

5 Conclusion and Future Work

Our study looked at the effects of reducing noise from our data as well as different methods for doing so.

After the noise was eliminated, our outcomes improved. The blatant existence of noise in the datasets was one of the most remarkable discoveries, and the urgent necessity to adequately remove the noise was highlighted by the significant improvement in our result that was observed after doing so. Additionally, the optimal method for message categorization is identified by our investigation. The outcomes of our analyses, which are reported here, demonstrate that accuracy and time vary depending on the method. The Random Forest Algorithm performs the best in our instance model. The code which we have written for analysis can found here.

In future work, we propose to pursue a number of directions. First, we plan to improve our ill-formed word detection classifier by introducing an OOV word whitelist. Additionally, we want to use a bootstrapping strategy to reduce noisy contexts. In this method, poorly formed words with high confidence and minimal ambiguity are changed to their standard forms and used as additional training data for the normalisation model.

6 References

Bo Han and Timothy Baldwin, NICTA Victoria Research Laboratory 2011. Lexical Normalisation of Short Text Messages.