Application of NLP techniques and Email Spam Filtering Algorithms to SMS Data

Haining Zhang

1. Introduction

Natural Language Processing (NLP) has gained more and more attention recently due to the computational characterization and analysis of human languages. It has been used in many areas such as machine translation, spam detection, information extraction, automated summarization, medical and Q & A systems. This project is focus on using techniques in NLP to reprocess the SMS dataset and comparing different performances for several classification models.

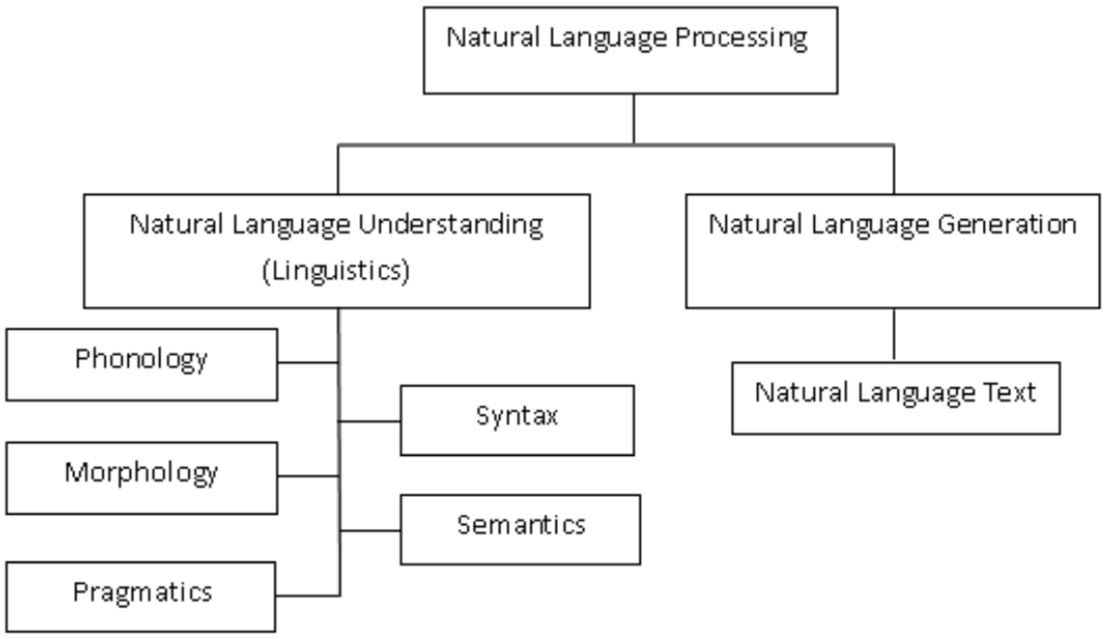
Mobile or SMS Spam is a real and growing problem, mainly due to the availability of a very cheap large amount of prepaid SMS packages, as well as a higher response rate as it is a trustworthy personalized service. SMS spam filtering is a relatively new task. It inherits many of the spam filtering problems and solutions. But it also poses its own special challenge.

In this project, a variety of popular email spam filtering algorithms were implemented on the Short Message Service (SMS) dataset. Different methods of using a matrix to represent the data set in NLP were attempted. In addition to using only tokens, other features of the message, such as the proportion of numbers or capital letters, are explored. The final classification results are presented. Unsupervised learning of the data set was also performed.

2. Natural Language Processing

Natural Language Processing (NLP) is a theoretically inspired series of computational techniques used to analyze and represent naturally occurring text at one or more linguistic analysis levels for human-like language processing of a range of tasks or applications.

Language can be defined as a set of rules or symbols. We combine symbols and use them to deliver or broadcast information. NLP basically can be divided into two parts, that is, natural language understanding and natural language generation They evolved into the task of understanding and generating text. The relation shows as follows.


Natural Language Processing
Natural Language Understanding (Linguistics)
Natural Language Generation
Phonology
Syntax
Natural Language Text
Morphology
Semantics
Pragmatics

The goal of NLP is to make computer analyze and understand the human language. By utilizing NLP and its components, one can organize the massive chunks of text data, perform numerous automated tasks and solve a wide range of problems such as – automatic summarization, machine translation, text classification, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation etc.

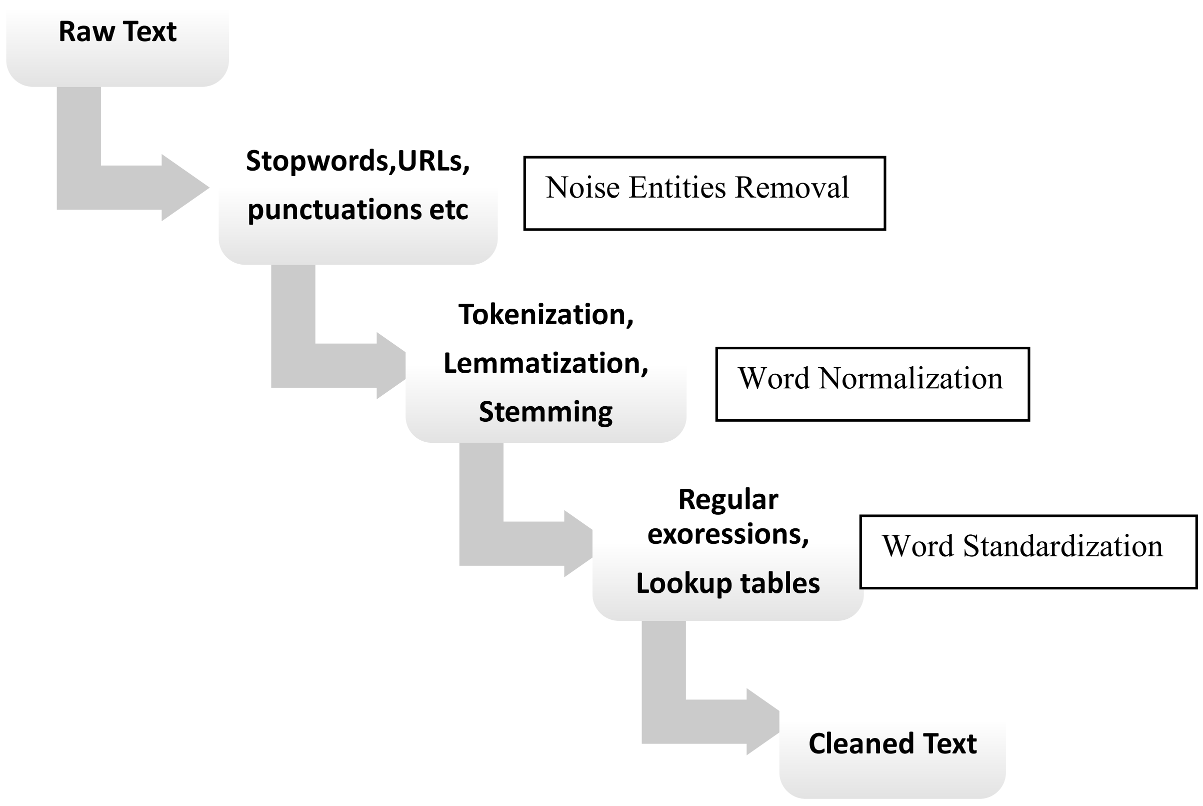
Since this project only focus on text classification, then I would only introduce some concepts and methods used in this area.

3. Text Preprocessing

Because text is the least-structured form of all available data, so various types of noise exist and data cannot be easily analyzed without any preprocessing. The whole process of text cleaning and standardization makes it noiseless and can be analyzed. It is known as text preprocessing, which consists mainly of three steps:

- Noise Removal
- Lexicon Normalization
- Object Standardization

The following picture shows a general text processing.


Raw Text
Stopwords,URLs, punctuations etc
Noise Entities Removal
Tokenization, Lemmatization, Stemming
Word Normalization
Regular exoressions, Lookup tables
Word Standardization
Cleaned Text

3.1 Noise Removal

Raw texts usually have a lot of noise, which is not related to the context of the data and the end-output. Before using the text, we should remove the noise by several methods.

Here is some common noise in text: stopwords, URLs or links, social media entities (mentions, hashtags), punctuations and industry specific words etc.
Stopwords refer to words that appear many times in a document without any meaningful context in themselves, such as: is, am, the, of, in etc.

Usually we use a noise list to filter, and according to the actual collection of documents to select the appropriate list. Then just eliminate those tokens which are present in the noise list.

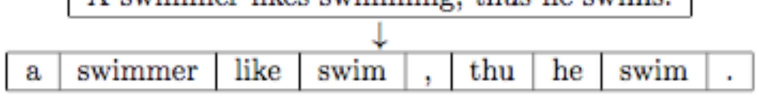
Another approach is to use the regular expressions while dealing with special patterns of noise. Regular expression is a sequence of characters mainly used to find and replace patterns in a string or file. It is popular among programmers and can be applied in many programming languages like Java, JS, C++ etc.

3.2 Lexicon Normalization

Lexicon Normalization deals with the multiple representations exhibited by a single word.
For example, “sing”, “singer”, “sung”, “sings” and “singing” are the different variations of the word – “sing”. All of them exhibit a same meaning in text. So we need to remove the word punctuation to get a reduced text.

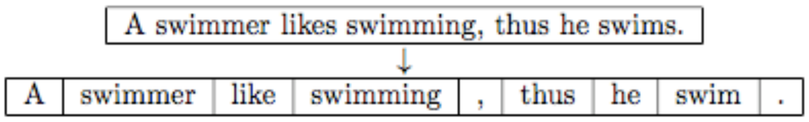
The common lexicon normalization methods are:

Stemming:
It reduces inflected (or sometimes derived) words to their word stem, base or root form. For example, a stemming algorithm reduces the words “fishing”, “fished”, and “fisher” to the root word, “fish”. Here is an example of stemming:



Lemmatization:

It is an organized and step by step procedure of obtaining the root form of the word. It makes use of vocabulary (dictionary importance of words) and morphological analysis (word structure and grammar relations). The following is an example of lemmatization, compared with stemming:



Bag of words:

Bag of words – defined as a matrix where each row represents a document and columns represent the individual token. The sequential order of text is not maintained.

Building a "Bag of Words" involves 3 steps:
1.Tokenizing – the act of breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols and other elements, which are called tokens.
2.Counting
3.Normalizing
Each unique word in the dictionary will correspond to a feature.

3.3 Object Standardization

Text data often have some words than cannot be recognized by usual models. Such as abbreviations or colloquial slangs. So we can prepare a dictionary or use regular expression to fix this noise. After that, we fix abbreviations such as “rt → retweet, dm → direct message”.

4. Text to Features

After the text processing, we have to transfer the text to features which model can used.
Based on the goal of analysis, we can choose several features, such as Syntactical Parsing, Statistical features, Entities / N-grams / word-based features and word embeddings.

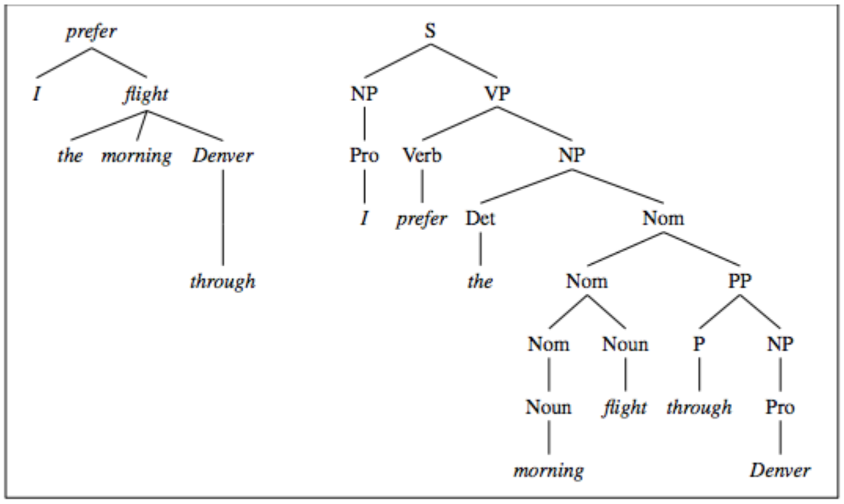
4.1 Syntactical Parsing

Syntactic parsing is the task of recognizing a sentence and assigning a syntactic structure to it. There are two important attributes of text syntactics - dependent trees and part of speech tagging.

4.1.1 Dependent trees

Sentences consist of spelled words. Relations between the words are illustrated with directed, labeled arcs from heads to dependents. In dependency grammars, the syntactic structure of a sentence can only be expressed in words in a sentence, while the word contains a set of related directional binary grammatical relations. Each relation can be represented by a triple (relationship, governor, subordinate).

Next is an example of dependency tree analysis for the sentence: “I prefer the morning flight through Denver.”



4.1.2 Part of speech tagging

In addition to the grammatical relationship, each word in the sentence is also associated with part of the speech tag (nouns, verbs, adjectives, adverbs, etc.). Tags define the usage and function of words in sentences. We can use these tags to classify the words and get new features.

4.2 Statistical features

In statistical models, we need numerical features. That means we need to convert the text files into numerical feature vectors. Each unique word in the dictionary will correspond to a feature. The most common statistical feature is Term Frequency times inverse document frequency (TF-IDF).

4.2.1 TF-IDF

It is a weighted model that aims to convert text documents into vector models based on the appearance of words in the document, regardless of the ordering.

TF-Term Frequencies: i.e. #count(word) / #total words, in each document.
IDF-Inverse Document Frequency– IDF for a term is defined as logarithm of ratio of total documents available in the corpus and number of documents containing the term T.
TF-IDF: Finally, we can even reduce the weightage of more common words which occurs in all document. This is called as TF-IDF, i.e Term Frequency times inverse document frequency.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

4.2.2. Count / Density Features

Count-based or density-based features can also be used for modeling and analysis. These features may seem trivial but have a large impact on the learning model. Some of the features are: number of words, number of sentences, number of punctuation and industry-specific words.

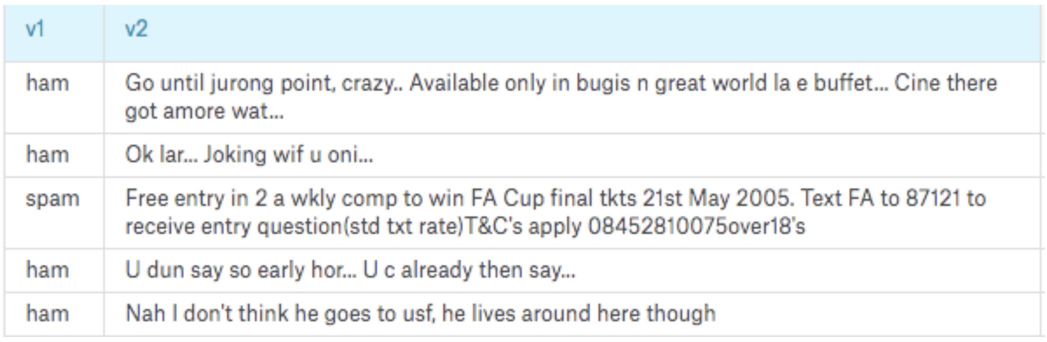
5. Application of Email Spam Filtering Algorithms to SMS Data

5.1 SMS Data Description

The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,572 messages, tagged according being ham (legitimate) 86.6% or spam 13.4%.

The files contain one message per line. Each line is composed by two columns: v1 contains the label and v2 contains the raw text.

Table 1: The first five messages



v1	v2
ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
ham	Ok lar... Joking wif u oni...
spam	Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over18's
ham	U dun say so early hor... U c already then say...
ham	Nah I don't think he goes to usf, he lives around here though

Figure 1 shows that, longer messages between 125 and 200 characters in length have a higher probability of spam than shorter messages of less than 125 characters in length. The histogram of the ham is positively skewed, while the histogram of spam is negatively skewed.

As can be seen from Figure 2, hams tend to have a smaller numerical portion than spams.

In Figure 3, the difference in the proportion of capital letters in hams and spams is not very large. However, you can still include it in the features and see if it contributes to improving the accuracy of the classifications.

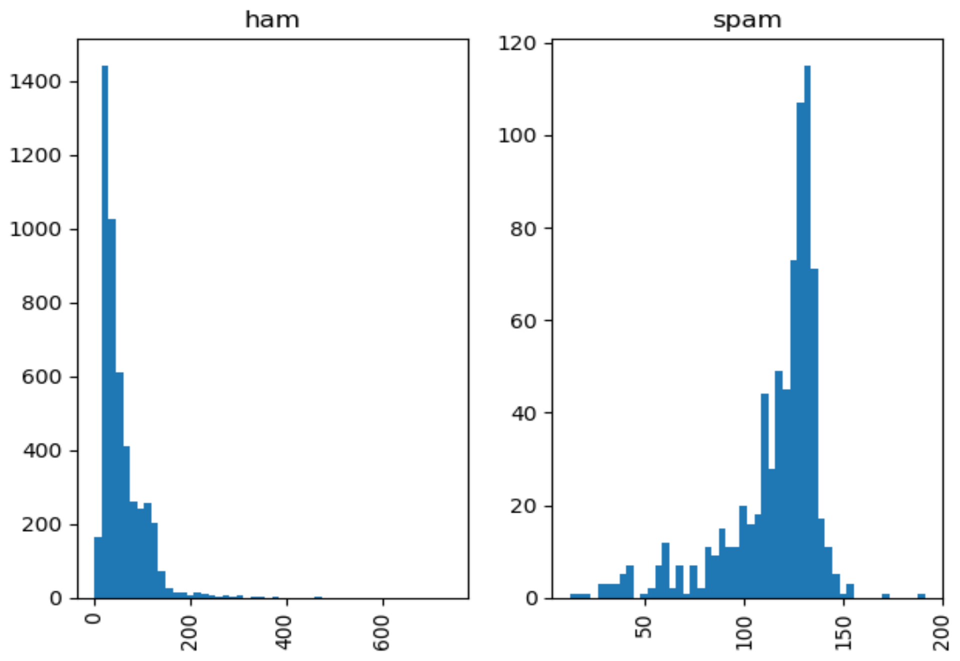


Figure 1: Histogram of message length by type

5.1 Convert Text to Sparse Matrices using Vectorizers

The most common way to extract numerical features from text content involves tokenization, that is, assigning each token a unique integer id, counting the occurrence of a token in the text content, and sometimes normalizing the rows of the result matrix before applying any algorithm. The token can be any text - a word, a phrase or a sentence. This conversion process is generally called vectorization.

Now convert the raw messages (a sequence of characters) into vectors (a sequences of numbers) using the bag-of-words representation. Here, one number will represent each unique word in a text. Then transfer it to **CountVectorizer** and the **TfidfVectorizer.** The process is finished in Python.

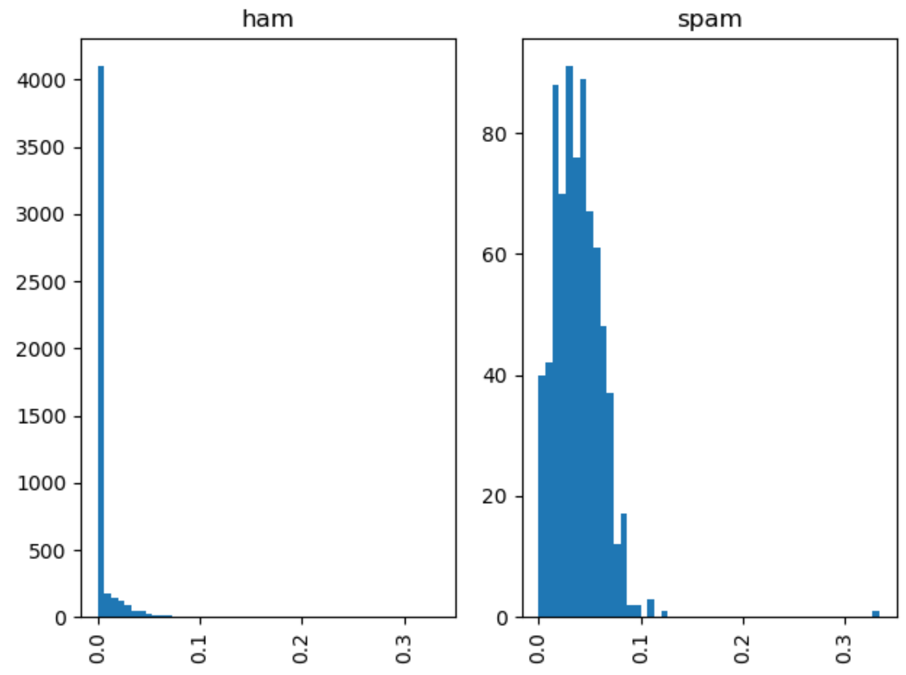


Figure 2: Histogram of proportion of numbers in message by type

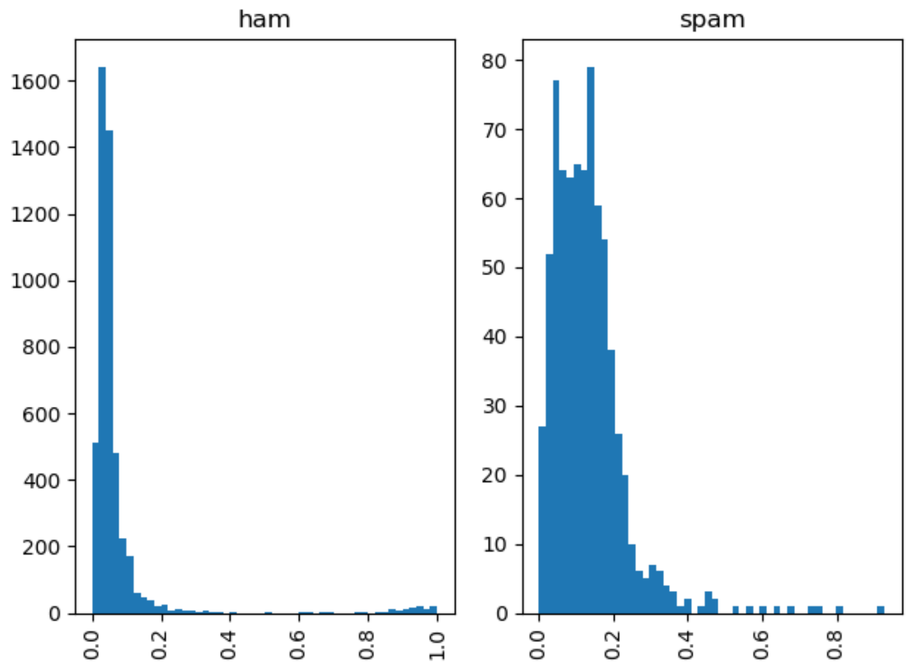


Figure3: Histogram of proportion of capital letters in message by type

5.2 Application of Vectorizers to the SMS Data

We remove the stop words from each SMS message and then use two vectorizers to convert the message into a sparse matrix. The row number of the generated matrix will be equal to the total number of messages in the corpus, which will be the number of tokens in the cleaned corpus. The tf-idf conversion uses the default option Euclidean norm in the normalization step. The resulting matrix is "sparse in the sense that while there are a large number of tokens in the corpus, a single text can only contain a small portion of them, so the matrix will have many 0 entries.

5.3 Classification Algorithms, Model Tuning, and Results

With messages now represented as vectors, it is time to train our spam/ham classifier.

Divide the data into training (80%) and testing (20%) sets and fit classifiers.

The classification algorithms considered include Support Vector Machine, Logistic Regression with elasticnet penalty, Decision Tree, Multinomial Naive Bayes, K-Nearest Neighbors, and three ensemble methods: Random Forest, AdaBoost and Bagging. All algorithms are quite commonly implemented in machine learning applications. They are all provided by the Python sklearn package.

Each of these models have been tuned to give their best performance on the sparse matrices generated by CountVectorizer and TfidfVectorizer, respectively. The results are summarized below:

It can be seen from Tables 2 and 3 that Support Vector Machine, Logistic Regression with elasticnet penalty, Decision Tree and AdaBoosting have better performance with CountVectorizer, while the other three ensemble methods, Multinomial Naive Bayes, K-Nearest Neighbor , Random Forest and Bagging perform better on TfidfVectorizer.

It is also worth noticing that K-Nearest Neighbor has the lowest accuracy. Considering the fact that 84.6% of the testing data is in a single ham class, it is not much better than a random guess.

The reason why TfidfVectorizer is not outperforming CountVectorizer might be due to the relatively small sample size. With 9,376 tokens and 5,572 messages, it is not quite necessary to consider weighting schemes such as tf-idf. Therefore, we consider using CountVectorizer in our subsequent analysis. It is also worth noticing that, in terms of accuracy, the ensemble methods are not significantly better than the first three relatively simple methods. Considering the computation efficiency, we use the first four methods in the next section.

Table2. Performance of classifiers on the testing data obtained by CountVectorizer

Classifier	Prediction.Accuracy (%)
Multinomial Naive Bayes	97.67
Support Vector Machine	97.93
Decision Tree	96.59
Logistic Regression	98.48
K-Nearest Neighbor	92.38
Random Forest	97.58
AdaBoost	96.50
Bagging	96.23

Table3. Performance of classifiers on the testing data obtained by TfidfVectorizer

Classifier	Prediction.Accuracy (%)
Multinomial Naive Bayes	98.30
Support Vector Machine	97.58
Decision Tree	95.70

Logistic Regression	96.41
K-Nearest Neighbor	92.91
Random Forest	97.85
AdaBoost	96.32
Bagging	96.50

5.4 Assessment of Usefulness of Additional Features

In 5.2, we created three additional features of the message: length, the proportion of the numbers, and the proportion of capital letters. It makes sense to add these features to the sparse matrix to help improve classification accuracy.

First, attach each feature to the sparse matrix obtained using CountVectorizer as a additional column. Then, four selected algorithms are run on the extended matrix. It turns out that including the length distort all models. A possible explanation might be that there are too much hams compared to spams in the training set and test set, and the non-standardized scale is too large compared to the entries in the original matrix. Normalization of the length using its maximum value also proved to be unhelpful because the longest message has 910 characters and the normalization suppresses the observation into range (0, 0.2).

5.5 Discussion for Dealing with Imbalanced Data

Since more than 85% of the messages are hams, the data set is essentially imbalanced. Common methods of dealing with imbalanced data include upsampling relatively small populations, downsampling relatively large populations, and using ensumble methods with weak learners. However, it has been demonstrated in the model adjustment step that the use of a large number of weak learners does not make the ensemble method better than the simpler method.

In this case, it is more reasonable to downsample the hams. If we upsample the spam class, we might have multiple identical messages in the training and test dataset. Using the information from the message to correctly classify the exact message does not mean that the classifier is good. In fact, it is rare that two messages are exactly the same. Therefore, we can try use the downsampling method to deal with the imbalance in future.

6. Summary

In this project, we implemented multiple classification algorithms on the SMS spam dataset. The classifier has different performance depending on how the dataset is vectorized. The accuracy of the classifier also depends on the proportion of spam and ham information in the data. When dealing with imbalanced data, we encountered a trade-off between conservation and security. This is reflected in the real world, because sometimes Gmail will mark useful emails as spam and put them in a spam mailbox, or allow spam to enter our inbox.

One of the limitations we have made in this project is the relatively small sample size and the imbalance between ham and spam in the data set. In fact, if we are implementing an SMS spam filter, we hope to collect more data to improve its accuracy.

Reference

1] Sarah Jane Delany , Mark Buckley , Derek Greene, 2012
SMS spam filtering: Methods and data

[2] Houshmand Shirani-Mehr ,SMS Spam Detection using Machine Learning
Approach

[3] https://www.kaggle.com/uciml/sms-spam-collection-dataset

[4] https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[5] https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/

[6] http://sebastianraschka.com/Articles/2014_naive_bayes_1.html

[7] https://www.analyticsvidhya.com/blog/2017/01/ultimate-guide-to-understand-implement-natural-language-processing-codes-in-python/

[8] http://dataaspirant.com/2017/02/06/naive-bayes-classifier-machine-learning/

[9] Elizabeth D. Liddy, 2001, Natural Language Processing Elizabeth D. Liddy

[10] Diksha Khurana1, Aditya Koli1, Kiran Khatter1,2 and Sukhdev Singh1,2 1Department of Computer Science and Engineering Manav Rachna International University, Faridabad-121004, India 2Accendere Knowledge Management Services Pvt. Ltd., India, Natural Language Processing: State of The Art, Current Trends and Challenges