Spam Filtering Using Natural Language processing (TF-IDF, Word2Vector, RNN)

Deepesh Bhatta
University of Texas at Arlington
dxb7305@mavs.uta.edu

Prithul Bahukhandi
University of Texas at Arlington
1001730554

Sunil Joshi
University of Texas at Arlington
sunil.joshi@mavs.uta.edu

Abstract

The objective of this project is to classify the text messages and predict whether it is spam or not using Natural Language Processing (NLP). In this project, we will be using 3 models, Random Classifier with TF-IDF Embedding, Random Classifier with Word2-Vec Embedding and Recurrent Neural Network. We will also compare each model to see which provides better result. Finally, the goal of this project is to build a model that will filter maximum number of spams with minimum loss of real messages.

1 Introduction

If we see the stat, 3.8 billion of people are using smart phone which is increase in more than 1 billion if we compare this back to 2016. 37.2 million consumers were opt-in to receive advertisement SMS in 2016, which is projected to rise to 48.7 million in 2020. 23 billion text messages are sent in a day, due to this huge popularity of SMS, we have seen uptick in number of fake or misleading advertisements. Even though SMS spam is nearly equal to one percentage, it might still result to huge financial loss.

Not only these spams are annoying, but customer can also lose real money from it. Some gullible customers can even respond to these messages and bear a financial loss by subscribing expensive services or non-existent services or may end up calling with premium rate. There might also be a risk of exposing personal/financial information to shady sites. Not only customers, but phone service providers might also suffer financially because of high maintenance in addition to damage to customer trust.

Thanks to the advancement in technology, Natural Language Processing can generate meaningful information from any text, which can be used for text classification that is very helpful for spam detection.

2 Dataset

The dataset which we will be using for this project can be found on Kaggle. The spam.csv file contains one message per line, where each line is composed by two columns: v1 contains the label (ham or spam) and v2 contains the raw text message. This file contains 4825 “ham” labeled messages and 747 “spam” labeled messages.

The main issue for our dataset is that it’s in string but our classifier needs dataset in some sort of numerical feature vector. And before that we need to perform cleansing of data. Some words in English which are necessary for correct grammar does not have any effect on the meaning of phrase even after removing it. These types of words are called stop words. Popular python library NLTK, Known as Natural Language Toolkit, provides common stop words. This list of stop words is stored in a Python list, by iterating over it we can remove those stop words from our dataset. Then, we remove the punctuation by checking character to see if they are in punctuation. We

use tokenization to split the text to each word and numpy library to change the label, spam and ham to 0 and 1 for analysis.

Label	Text
Ham	4825
spam	747

3 Methods

We have used different models to classify the given text. Before training the following proposed models, we split our data into a training set and a testing set in a ratio of 80 : 20 percentage by using scikit-learn train-test-split method.

3.1 Random Forest Classifier with TF-IDF Embedding

Random forest consists of large number of individual decision trees that operate as an ensemble[1]. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

TF, Text Frequency, is the number of times a word appears in a document which tells us how important a word is in a document. IDF, Inverse Document Frequency, is ratio which tells us the importance of the word in the entire collection of documents, also known as corpus. We can get TFIDF score by multiplying both TF and IDF.

For our model, we import TFIDF Vectorizer from scikit-learn feature extraction, which is used to fit the training data. During fitting, the TFIDF function discovers the most common words in the corpus and saves them to the vocabulary. Then, the fitted data is transformed to TFIDF matrix with the shape <4457 x 8347> for training data and <1115 x 8347> for testing data. By invoking tfidf-vect.transform method, TFIDF scores is computed. The weight of each word is normalized by the number of times it appears in the corpus, so a word that appears in only 10 percentage of all documents will be assigned a higher value (and thus treated more importantly) then one which appears in say 90 percentage of documents. Finally, we will be training our model in Random Forest classifier which can be directly imported from sci-kit ensemble method. Then, we predict the test data to check the performance of our model and following output is generated for label[9].

```
predicted: 1
expected: 1
```

The accuracy of our model is 97.6 percentage while precision is 100 percentage and recall is 80.6 percentage.

	precision	recall	f1-score	support
0	0.97	1.00	0.99	976
1	1.00	0.81	0.89	139
accuracy			0.98	1115
macro avg	0.99	0.90	0.94	1115
weighted avg	0.98	0.98	0.97	1115

3.2 Random Forest Classifier with Word2Vector Embedding

WORD2VEC model determines the semantic and syntactic meaning of co-occurrence word by using unsupervised learning, which is used to map a word to a vector. Words which have similar meaning such as 'house' and 'home' map to similar vectors, while dissimilar words have dissimilar vectors. A model can be able to use the information preserved in its embedding to learn the relationship between words, due to this feature a model trained on the word 'house' would be able to guess home even if it did not train on this word.

We use GENSIM library to train data on WORD2VEC model. When fitting the Word2Vec,we need to specify the target size of the word vectors, we'll use 100. After that, we will replace the words in each text message with the learned word vector. The word vectors for each sentence is averaged and vector of zeroes is assigned if model did not learn any of words from text messages in training. After that, the averaged data is now trained using Random Forest Classifier similar to model 1, which aggregates the votes from different decision trees to decide the final class of the test object. The predict method will give the following output for label[9].

```
predicted: 0
expected: 1
```

For this model, the accuracy is found to be 87.9 percentage, precision of 53.3 percentage and recall of 23 percentage.

	precision	recall	f1-score	support
0	0.90	0.97	0.93	976
1	0.53	0.23	0.32	139
accuracy			0.88	1115
macro avg	0.72	0.60	0.63	1115
weighted avg	0.85	0.88	0.86	1115

3.3 Recurrent Neural Network

Recurrent Neural Network is a generalization of feedforward neural network where output of previous step is fed as input. It has the memory so it can remember all information from previous step. RNN converts the independent activations into dependent activations by providing the same weights and biases to all the layers, thus reducing the complexity of increasing parameters and memorizing each previous output by giving each output as input to the next hidden layer [2]. For making a decision, it takes in consideration of the current input and the output learned in previous step.

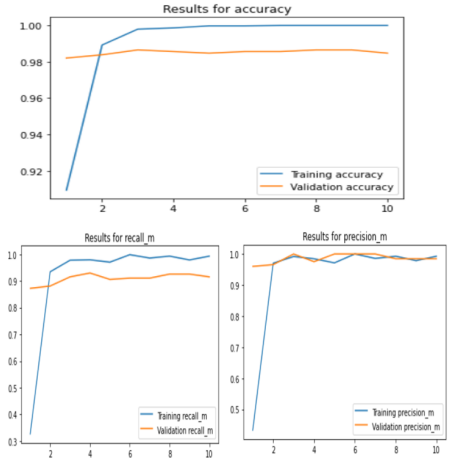
We need to preprocess the data, for which we used tokenizer method which can be directly imported from tensorflow.keras.preprocessing.text which is used to convert the sentences to sequence of numbers, which is then padded using pad sequence method.

```
Found 8261 unique tokens.
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 32)          264384
_________________________________________________________________
lstm_1 (LSTM)                (None, 32)                8320
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 273,793
Trainable params: 273,793
Non-trainable params: 0
```

We compile above model using binary cross entropy as a loss function, Adam as an optimizer and accuracy, precision, and recall as metrics. We then fit our dense classifier using model.fit() method. It uses padded training data and training labels for training the model and testing data for validation. Epoch is set to 10 which is number of times our model will work through the entire training data set. The predict method will give the following output for label[9].

```
predicted: [5.0294257e-06]
expected: 0
```

The training accuracy is 100 percentage, while validation accuracy is 98.48 percentage. Similarly training precision is 99.29 percentage, validation precision is 98.48 percentage, training recall is 99.29 percentage, and validation recall is 91.59 percentage.



4 Result

We have used three evaluation metrics that is accuracy, precision, and recall to compare the performance of each model. Precision shows how precise/accurate our model is out of those predicted positive, how many of them are actual positive[3]. Recall actually calculates how many of the actual positives our model capture through labeling it as Positive (True Positive)[3].

90 percentage precision means 10 percentage of message flagged as spam wasn't spam. This is still an unacceptable rate for losing possibly large number of important messages. 80 percentage of recall means 20 percentage of spam is not detected as spam, which might be considered as worst.

MODEL	ACCURACY	PRECISION	RECALL
RFC-TFIDF	97.6%	100%	80.6%
RFC-WORD2VEC	87.9%	53.3%	23%
RNN	98.48	98.48	91.59

5 Conclusion

Since our dataset contains only 13 percentage of message that is labeled as spam, and more than 86 percentage as ham, accuracy metrics might be misleading in this case while measuring performance of each model. So, for this dataset, Precision and recall is more important to compare these models. Among these models, RNN can be considered as best model to predict spam, having high precision of 98.48 percentage and recall of 91.59 percentage. While RFC-TFIDF is also good with precision of 100 percentage but not quite as good as RNN with only 80.6 percentage of recall. RFC-WORD2VEC is worst model because of very low precision and recall.

6 References

[1] Yiu, Tony. "Understanding Random Forest." Medium, Towards Data Science, 14 Aug. 2019, towardsdatascience.com/understanding-random-forest-58381e0602d2.

[2] aishwarya.27. "Introduction to Recurrent Neural Network." GeeksforGeeks, 3 Oct. 2018, www.geeksforgeeks.org/introduction-to-recurrent-neural-network/.

[3] Shung, Koo Ping. "Accuracy, Precision, Recall or F1?" Medium, Towards Data Science, 10 Apr. 2020, towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9.

[4] Delany, Sarah Jane, Mark Buckley, and Derek Greene. "SMS spam filtering: Methods and data." Expert Systems with Applications 39.10 (2012): 9899-9908

[5] Clement, J. "Internet Users in the World 2020." Statista, 24 Nov. 2020, www.statista.com/statistics/617136/digital-population-worldwide/.

[6] O'Dea, Published by S., and Dec 10. "Smartphone Users 2020." Statista, 10 Dec. 2020, www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/.

[7] "26 SMS Marketing Stats to Know in 2020." 99firms, 21 July 2020, 99firms.com/blog/sms-marketing-stats/.

[8] Learning, UCI Machine. "SMS Spam Collection Dataset." Kaggle, 2 Dec. 2016, www.kaggle.com/uciml/sms-spam-collection-dataset.