Capstone Project

Esther Liao
1/6/17

I. Definition

Project Overview

As the world continues to move forward in an age of ever advancing technology, calls to action maintaining a free-flowing internet have become increasingly challenging with the presence of spam messages. Many email service providers have rigorously worked on updating spam filters that correctly identify spam messages. However, the same has not been nearly as true for text messages even as text messages have become more and more widespread. While SMS spam certainly inherits features from email spam, it is still challenging because text messages are inherently shorter than their counterpart.

As text messages are easier to read in full and at the same time potentially carrying important information, it is absolutely critical that SMS spam is classified correctly. At the same time, SMS spam can cause a disruptive experience and should be properly identified as email spam is filtered. As text messaging becomes an increasingly popular mode of communication, there should be developments made to properly classify SMS spam to allow people an optimal experience with text messaging.

Problem Statement

The more urgent nature of text messages makes it vulnerable to spam attacks which could potentially prevent or delay the reception of important information. The goal is to train a Convolutional Neural Network (CNN) to perform a binary classification to classify text messages as either spam and non-spam.

Metrics

As this is a binary classification, I will use accuracy to account for true positives and true negatives. The equation for accuracy is as follows:

$$accuracy = \frac{true\ positives + true\ negatives}{dataset\ size}$$

For the purposes of this project, I aim to get an accuracy of over 98%.

II. Analysis

Data Exploration

For this project, the SMS Spam Collection Dataset from the University of California in Irvine will be used[1]. The dataset has been made publicly available on Kaggle. It contains a total of 5,572 messages that are tagged as ham (legitimate) or spam. The dataset itself is split between two columns *v1* and *v2*, one representing the label (ham or spam) and another containing the raw text. The distribution of the dataset is not very balanced, with 4,825 of these messages tagged as being "ham" while the remaining 647 messages being tagged "spam."

	v1	v2
1	v1	v2
2	ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
3	ham	Ok lar... Joking wif u oni...
4	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
5	ham	U dun say so early hor... U c already then say...
6	ham	Nah I don't think he goes to usf, he lives around here though
7	spam	FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs to send, £1.50 to rcv
8	ham	Even my brother is not like to speak with me. They treat me like aids patent.
9	ham	As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has been set as your callertune for all Callers. Press *9 to copy your friends Callertune

Figure 1. SMS Messages from the SMS Spam Collection Dataset

Every message inside the dataset is non-empty and labeled either ham or spam. Looking through the dataset, the lengths of the messages greatly vary with some messages only having one word to others that have multiple sentences. In order to perform analyze the messages, I will convert each message to a corresponding word vector.

[1] http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/

Exploratory Visualization

The word clouds below provide visualizations for frequently seen words in spam and ham messages. The word clouds were created by going through the words in each message and creating a dictionary containing the frequencies of words that had at least 4 characters. The decision to only consider words that had at least 4 characters was made to avoid considering words such as “I” or “you” that initially prominently displayed on both the spam and ham word clouds.

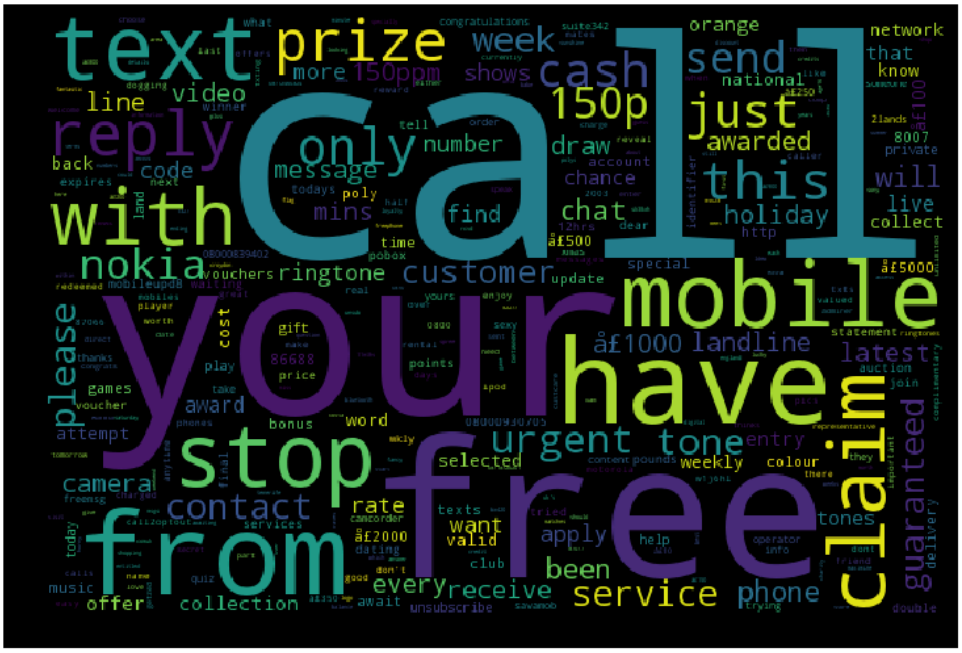


Figure 2. WordCloud Generated from Spam Messages

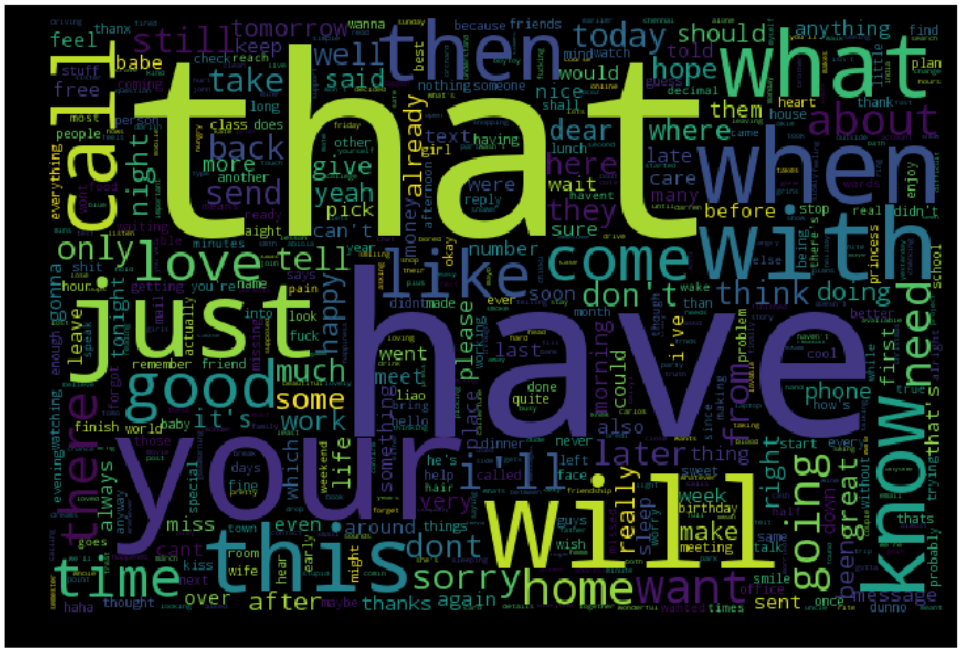


Figure 3. WordCloud Generated from Ham Messages

From the word clouds above, we still see word appearing in both. However, there are words that we see in both that seem to be good indicators of what might be spam or ham messages. For example, in the spam word cloud, we see the words “free” and “claim” which might be used to entice users to take part of a potential scam. In the ham word cloud, we see words such as “when” and “will”, action words that seem to indicate relevant events that the receiver should want to know about.

Algorithms and Techniques

The algorithm used is a Convolutional Neural Network (CNN), a state-of-the-art algorithm for classification. The CNN is comprised of multiple layers which assign different probabilities to the inputs based on specified parameters. Using a CNN for this task is advantageous as there are many different parameters that can be used to fine-tune the model. For my model, I used the following parameters:

- Training length (number of epochs)
- Batch size (how many messages to look at for each training step)
- Dropout (randomly setting a fraction rate of input units to 0 to help prevent overfitting)
- Activation functions (to apply an activation function to the output to normalize)
- Loss function (to optimize the model based on the steps it takes)

The first model was one included as a kernel to the dataset on Kaggle, and this CNN achieved an accuracy of 99.8%. This model used three layers, a binary cross entropy loss function, the Adam optimizer, and “relu” activation functions for the first two layers and a sigmoid activation function applied to the last layer. While 99.8% is a high accuracy already, I added an additional layer as seen in the notebook, adding in the “tanh” activation function. Additionally, I used the Adagrad optimizer. Indeed the results of running both of these models on a total of 10 epochs with batch sizes of 32 inputs and a validation split of 0.2, we see that the modified model reaches an accuracy of 96.72% on the first epoch whereas the former model reaches an accuracy of 95.04% on the first epoch.

The second layer contained an additional layer with increased dropout rates between each layer. I believe that this helped the model avoid overfitting while also quickly reaching higher accuracy rates from the start. I also used Adagrad as the optimizer instead, and applying gradient descent along with a higher dropout rate at each layer likely helped the model train itself to such a high accuracy rate.

Benchmark Model

As a benchmark, I will applied K-nearest neighbors to the data. While testing with the K-nearest neighbors classifier, I saw that setting the number of neighbors to 1 gave the highest accuracy score of 95.16%. In order to preprocess the data for K-nearest neighbors, I used a vectorizer (the TfidfVectorizer) to fit the message data. I decided to use an algorithm that did not make use of deep learning because I wanted to compare the results. Especially with data involving natural language, it makes sense that there are non-linear patterns being observed in the messages. As such, it seemed to make the most sense to try K-nearest neighbors as this is a clustering algorithm and compare the results of it to that of applying a CNN. However, the 95.16% accuracy achieved by K-nearest neighbors is still less than the accuracy achieved by the CNN discussed earlier on its run on the very first epoch.

III. Methodology

Data Preprocessing

To preprocess the data, I used a label encoder and tokenizer to preprocess the messages. The label encoder set the labels “ham” and “spam” to a flat 1-dimensional vector. Once the labels had been encoded, then the tokenizer was applied to vectorize the messages. To complete preprocessing, the messages were then translated to matrices using the “texts_to_matrix” function.

Implementation

Implementation of both K-nearest neighbors and the CNN were both done in a Jupyter notebook. For the CNN, I used Keras with a Tensorflow backend. The first thing I did was to rename the columns from their default values of “v1” and “v2” to more descriptive titles “label” and “text” respectively. For the visualizations, I wrote a function wordFreqs(text) that took in text and output a dictionary recording word frequencies from the text. After creating dictionaries for both the spam messages as well as the ham messages, I created two word clouds for the two groups of messages.

While creating the two word clouds, I ran into some complications with deciding whether or not I wanted to vectorize the data or if I wanted to loop through the words myself. I decided to loop through the words and apply the Keras text_to_word_sequence filter to each of the messages instead. Even after doing so though, I noticed that it was more informative to only count the frequencies of words with at least 4 characters as described earlier.

In implementing the K-nearest neighbors classifier, I referenced the documentation as well as example code to run the classifier. After fine-tuning the nearest neighbors parameter, I found that a nearest neighbors value of 1 gave the highest accuracy score of 95.16%.

In implementing the CNN classifier, I found an example of a simple CNN. I then modified this classifier by fine tuning the parameters described in Algorithms and Techniques. I did multiple trials with differing numbers of layers, dropout rates. I noticed that changing the optimizer could drastically change the accuracy rate, and after multiple trials, I found that a combination of using Adagrad with increased dropout rates between layers yielded one of the consistently highest accuracies among all 10 epoch runs.

The code for this section can all be referenced in the included Jupyter notebook.

Refinement

The accuracy of 99.98% attained by the CNN is quite high. Refining the algorithm might come with having a larger dataset that would be able to extensively stress test the performance of the algorithm against the data. While refining the model, I found that adding greater dropout rates between layers improved performance. Indeed, with the final model presented, accuracy jumped from 96.72% to 99.19% from epoch 1 to epoch 2.

Project Design

I will begin by preprocessing the data and cleaning extraneous data included with the dataset so that we are left with the text and spam label. While doing text analysis on the words, I will either filter through the words in each message or provide some sort of stripping of the message when applying machine learning (e.g. upper/lower case, punctuation, etc.). I will convert the text into word vectors, doing multiple trials to try and achieve optimal results. Different visualizations and graphs will be used to depict the results of applying the classifier in question. For testing purposes, depending on time complexity, I might use a smaller segment of the data. After getting the results of the highest achieving classifier, I will attempt to generate a word map for both spam as well as ham messages to see which words the classifier prioritized when identifying if a message was spam or not.

IV. Results

Model Evaluation and Validation

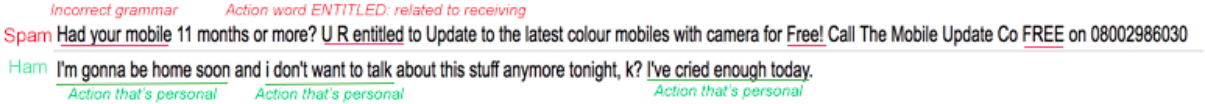
With a 99.98% accuracy, the final model is reasonable and aligning with solution expectations. The final parameters of the model are appropriate as they allow for the biggest jump in accuracy from the first epoch to the second epoch. After testing the model with different input models, it continues to perform just as well as it does in the displayed trial in the notebook. Results from the model with regard to this specific dataset of SMS messages should be able to be trusted. However, 5,000 SMS messages are not representative of the millions of messages that are sent on a daily basis. Therefore, results from this model with regard to all SMS messages should be tread carefully. Much more testing and training with a larger dataset should be done in order to capture the nuances in everyday texts to be able to differentiate between spam and ham messages. Additionally, this dataset is not evenly split between spam and ham messages, and this imbalance may also change how well the model performs on real time messages.

Justification

The final results here are stronger than that of the benchmark. The benchmark reached an accuracy 95.16% while the final model reached an accuracy of 99.98%. The final model increases accuracy by over 105%. The increase in accuracy is significant, but the 0.02% room for error that exists could still be detrimental if utilized in a real time setting where important messages are being transmitted. In these situations, if even one of those messages were classified as spam, this could cause a significant issue.

V. Conclusion

Free-Form Visualization



A very important aspect of this particular dataset is its relation to natural language. Language is at the basis of determining whether or not a message is spam. Indeed, in the two word clouds presented at the beginning, there were words that appeared in both. To better capture the nuances that differentiate spam messages from ham messages, we must look at the ordering and connotations of not just words but phrases in these messages. Above is an example of using

natural language processing to flag a message for being spam or not. There are many different phrases and meanings that lead to a message being spam or ham, and the most effective way to apply natural language processing in this setting would be to have weights on different phrases and interpretations. After all, that's how humans determine what things are more important than others. The free-form visualization provides the intuition to how we might weight these factors. For example, we might place a heavy weight on personal, first-person actions for ham messages. On the other hand, we might place a heavy weight on personal, second-person actions (Ex: "Had your mobile") for spam messages as these kinds of phrases tend to suggest that the recipient do something. These improvements are discussed in further detail in the Improvement section, but if these are taken into account, the classifier would likely be able to perform even better as it would be able to understand context in a way more similar to that of humans.

Reflection

Overall, this project was a very hands-on and educational experience with applying the machine learning techniques learned over the course of the last several months. The project is the epitome of all the things that I have learned so far, and I enjoyed being able to put together all these pieces in doing this assignment. Additionally, I felt that the other assignments had prepared me well to take on something of my own, and it was exciting to be able to go out and choose a topic that matters to me. As stated in the beginning of this paper, the topic of spam is something that needs to be dealt with. The fact that SMS spam detection is far behind email spam detection motivated my interest to explore the SMS spam dataset with a CNN. It was quite difficult at first to get started, but after reading through the sample projects, many different Kaggle kernels, and quite a bit of Keras, Scikit-learn documentation, I managed to get everything working. Debugging on my own was challenging, but it was also immensely rewarding in the end to see things work and understand exactly why they worked.

Improvement

As briefly mentioned above, natural language processing would be the next step to improving the results from today. While the preprocessing done here does touch on some natural language processing, it is not extensive enough for the algorithm to weight context the same way humans do. However, as complexity grows exponentially with the number of features, we would likely need a much larger dataset to capture these features and efficiently train a neural network on. For this approach, a neural network would still be the best way to go as the proposed "weights" determined by NLP would be the probabilities that each layer in the neural network assigns to its respective output. Perhaps the biggest challenges to making this kind of an improvement are:

1. Be able to accurately determine how to preprocess the text and weight it such that it captures the context of the message
2. Create an efficient neural network that can train itself on a significantly greater dataset with significantly more features while getting optimal results

If done, this suggested model could be in a position to begin reviewing SMS messages in real time and determining whether or not they are legitimate or mere spam.