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 and intrusive experience to mobile users[1]. The existence of email spam filters have been helpful in providing a start to SMS spam detection, and this project will take a similar approach in classifying messages as such[2]. Since text messaging has become 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. This project will make use of the SMS Spam Collection Dataset from the University of California in Irvine to create a model that will be used to distinguish spam from legitimate messages[3].

[1] A. Lambert, "Analysis of SPAM", pp. 1-100, 2003.
[2] J. Hua, Z. Huaxiang, "Analysis on the content features and their correlation of Web pages for spam detection", China Commun., vol. 12, no. 3, pp. 84-94, Mar. 2015.
[3] http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/

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 as this is a binary classification problem.

Metrics

The dataset itself is split between two columns, one representing the label (ham or spam) and another containing the raw text from the SMS message. As the ratio of legitimate (ham) to spam messages is quite imbalanced, I will be using the F1 score which can be understood as a weighted average between precision and recall with 1 being the optimal score and 0 being the worst score[4]. The formula for F1 is as follows:

$$F1 = \frac{2(precision * recall)}{(precision + recall)}$$

Additionally, I will use accuracy to account for true positives and true negatives. The equation for accuracy is as follows:

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

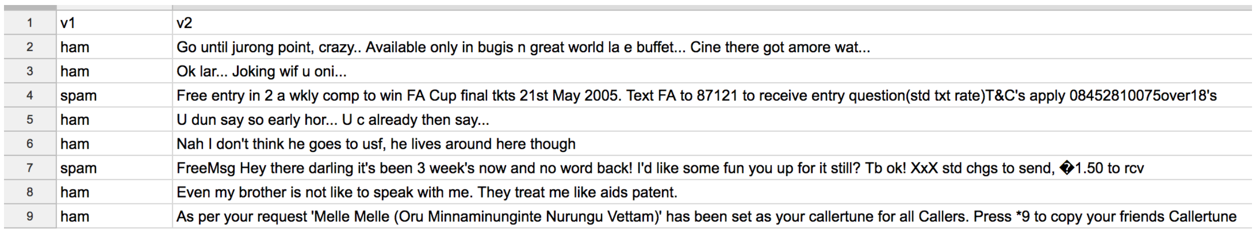
For the purposes of this project, I aim to get an accuracy of over 98%.

[4] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

II. Analysis

Data Exploration

For this project, the SMS Spam Collection Dataset from the University of California in Irvine will be used[5]. 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."



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

Looking at the dataset as a whole, I added a column that contained the character lengths of each of the messages. The mean length of a message is 80 characters long, with a standard deviation of 61 characters. As the standard deviation is a measure of variance, we see that the messages can vary quite a bit from the mean. This makes sense as some messages might be extremely short replies while others might have important details and be much longer. Within the dataset, there were a total of 114 outliers in the 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.

[5] 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.

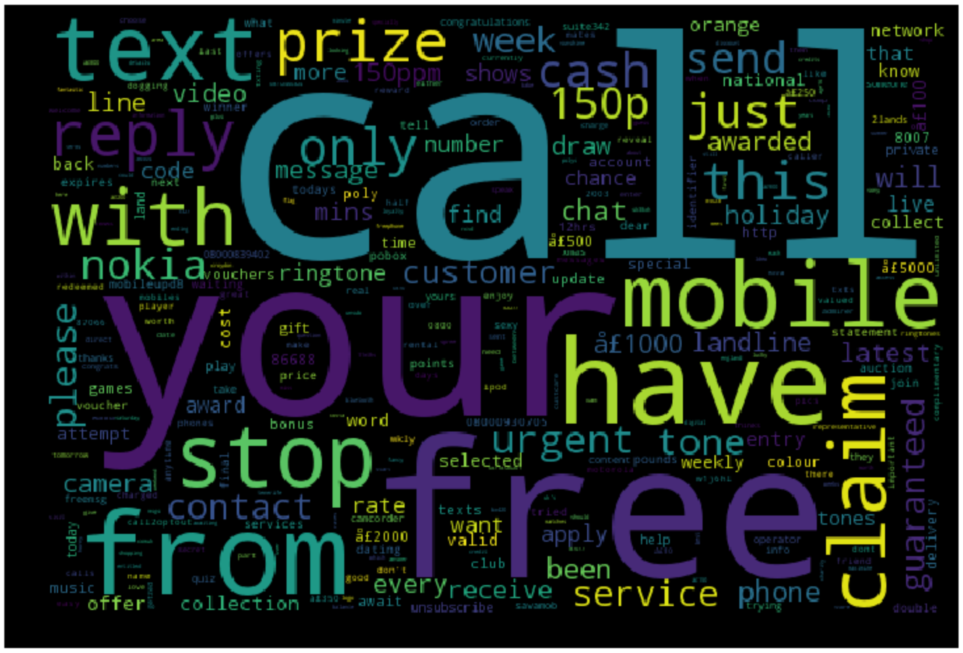


Figure 2. WordCloud Generated from Spam Messages

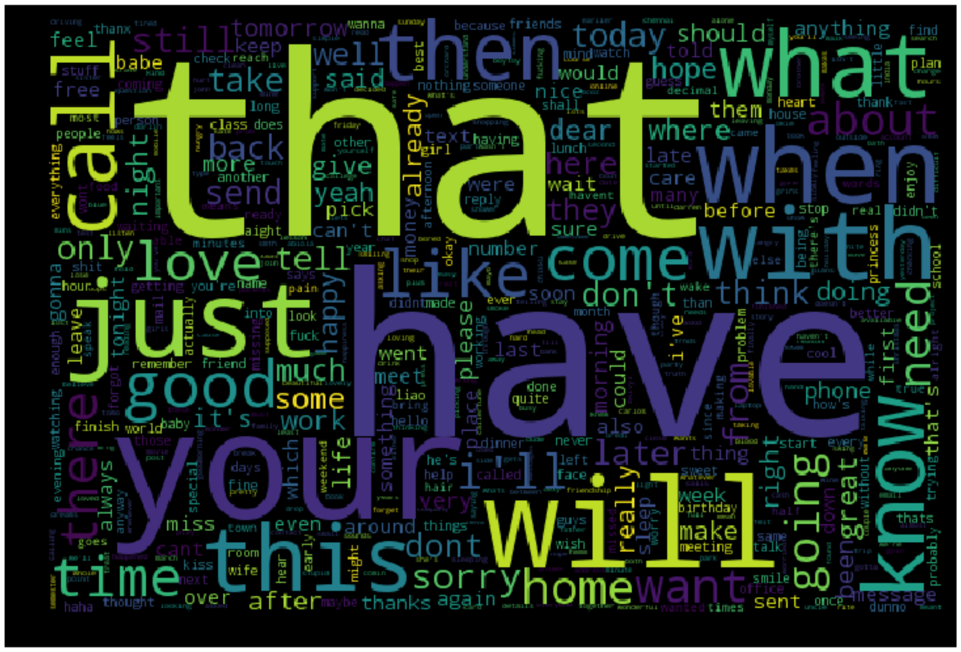


Figure 3. WordCloud Generated from Ham Messages

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.

On a high level, each layer in the network contains a "convolution" of the input data. The convolution applies some kind of function to the input data, converting it from its original state to another matrix. For example, for a black and white image, the convoluted result might be a matrix with 0 representing the white pixels and 1 representing the black pixels. By using multiple layers in the CNN, the convolutions from the input layer are used to compute results of the output.

With natural language processing, we can use Keras's text preprocessing to convert the text into matrices that can then be fed into our CNN. An example of how the CNN might process the text matrix is included below:

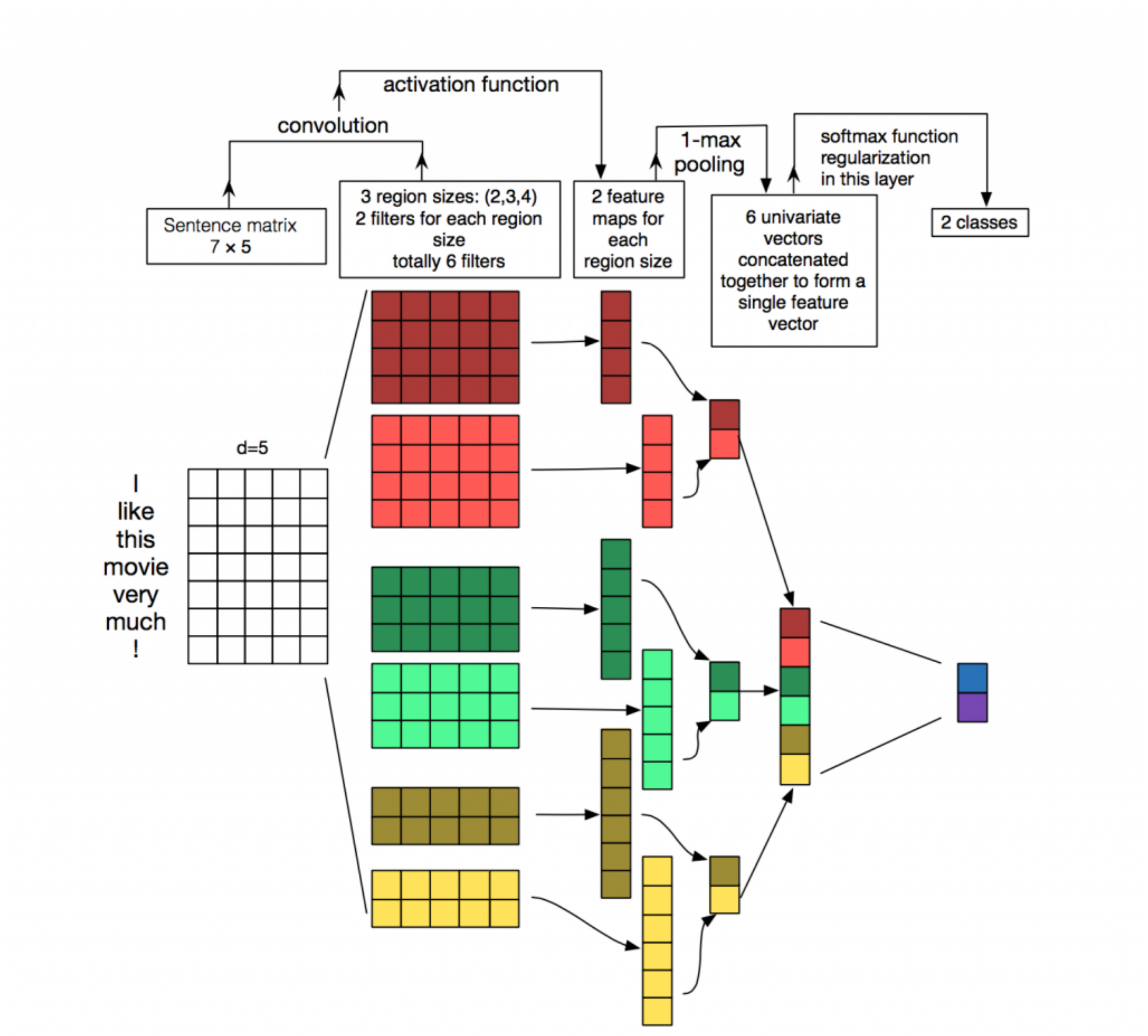

activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Figure 4. Example of CNN for sentence classification. Binary classification is assumed and displayed two possible output states.[6]

As depicted by the model above, the layers have differing dimensionality. Being able to change the dimensionality of the outputs from layers is extremely beneficial for efficiency and fitting purposes. One way to do this is to apply pooling which provides a fixed size output matrix.

[6] Zhang, Y., & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification.

Pooling layers subsample their input[7]. The max pooling operation keeps information about the presence of features in the text but loses information about exactly where it happened, something that is incredibly valuable in scanning texts for similar recurrences of words/phrases. By restricting the size of the output matrix, it is easier to avoid overfitting with too many parameters. Another way we can regularize the CNN will be to use a dropout layer that forces a fraction of the neurons in the neural network to "dropout" and force the network to learn features that are more holistically critical to correctly classifying the data in question.

We can apply filters to each layer to detect specific features that might be useful to us. While training, the CNN learns the values of its filters. The layers then compute probabilities that ultimately provide a high level tool- in this case, deciding whether a text message is spam or legitimate.

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

[7] Marc Moreno Lopez, Jugal Kalita "Deep Learning applied to NLP", arXiv:1703.03091v1 [cs.CL] 9 Mar 2017

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.

I decided to use a CNN for this model because CNN's are advantageous when it comes to recognizing patterns within a larger picture. Other neural networks such as the recurrent neural network (RNN) use the output from layers to inform decisions moving forward. I did not want to use an RNN as I wanted a model that recognized patterns within text the way CNN's are able to do so.

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% and an F-1 score of 44.56%. 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 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 was used to transform the categorical labels "ham" and "spam" to numerical labels. Once the labels had been encoded, then the tokenizer was applied to vectorize the messages. I tokenized the data using Keras's Tokenizer function as this takes advantage of breaking the text within the messages into different segments (tokens). Having vectorized the textual data, the messages were then translated to matrices using the "texts_to_matrix" function. With the messages all being represented by matrices, we can think of each message having been encoded into uniform data that can then be trained and tested on.

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. The simple model makes use of three layers. Between the three layers, two Dropouts with rates of 0.2 and 0.3 are applied respectively to avoid overfitting. The first two layers also made use of the 'relu' activation function while the last layer made use of the 'sigmoid' activation function. By using ReLUs, the neural network is able to speed up its training as the gradient computation is simply 0 or 1 depending on whether the result is negative or positive (where negative results are mapped to 0 and positive results are mapped to 1).

To improve upon the simple CNN, 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.

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. The F-1 scores are also considerably higher, with a F-1 score of 99.92% at the end of training which is considerably higher than the 44.56% result from running K-nearest neighbors.

To see how well the model performed with different input parameters, I tested the following:

- Batch size: 32, Epochs: 10, Validation split: 0.2
    - Accuracy: 99.98%
    - F-1: 99.92%
- Batch size: 64, Epochs: 10, Validation split: 0.2
    - Accuracy: 99.98%
    - F-1: 99.79%
- Batch size: 32, Epochs: 20, Validation split: 0.2
    - Accuracy: 99.98%
    - F-1: 99.93%
- Batch size: 32, Epochs: 10, Validation split: 0.4
    - Accuracy: 100%
    - F-1: 100%
- Batch size: 64, Epochs: 10, Validation split: 0.4
    - Accuracy: 100%
    - F-1: 100%

After tweaking the model results, I was able to achieve a 100% accuracy and F-1 score with the final results where I increased the validation split. Even though the model that I had with 99.98% accuracy was already better than the simple model that I started out with, it seemed as though increasing the validation split gave the most favorable results. After having changed the batch, epoch, and validation split values, the accuracy and F-1 scores seem to be fairly consistent with both being over 99% on each run. While it seems like the model works fairly well with the dataset at hand, the 100% figures could be pointing out a near perfect fit that the model has found for describing this particular dataset.

Unfortunately, 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. Improvements are discussed in further detail below.

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%. While 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

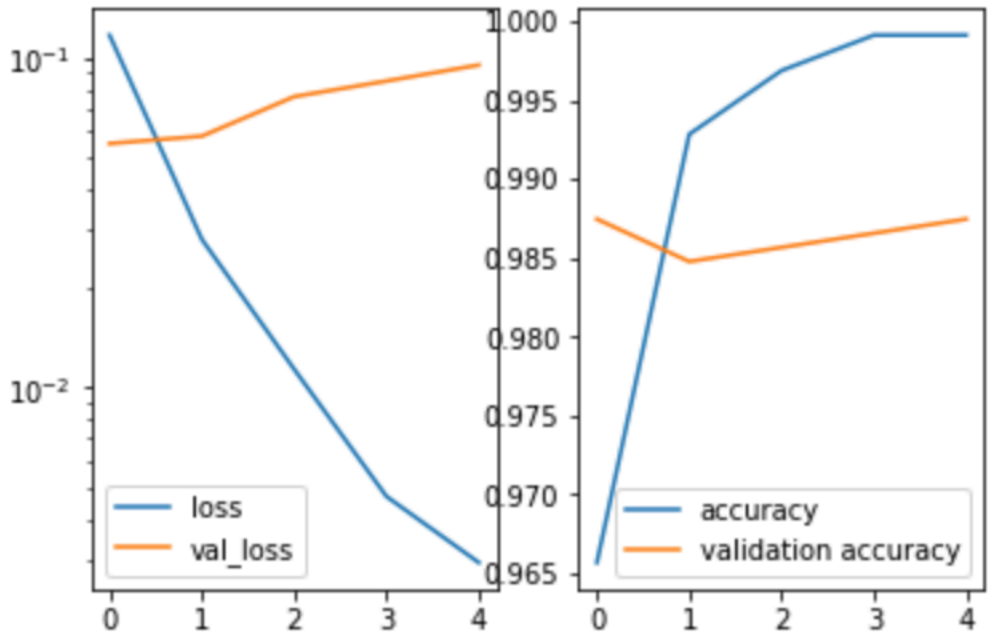


Figure 5. Live-loss and Accuracy Plots after Epoch 5, *Batch size: 32, Epochs: 10, Validation split: 0.2*

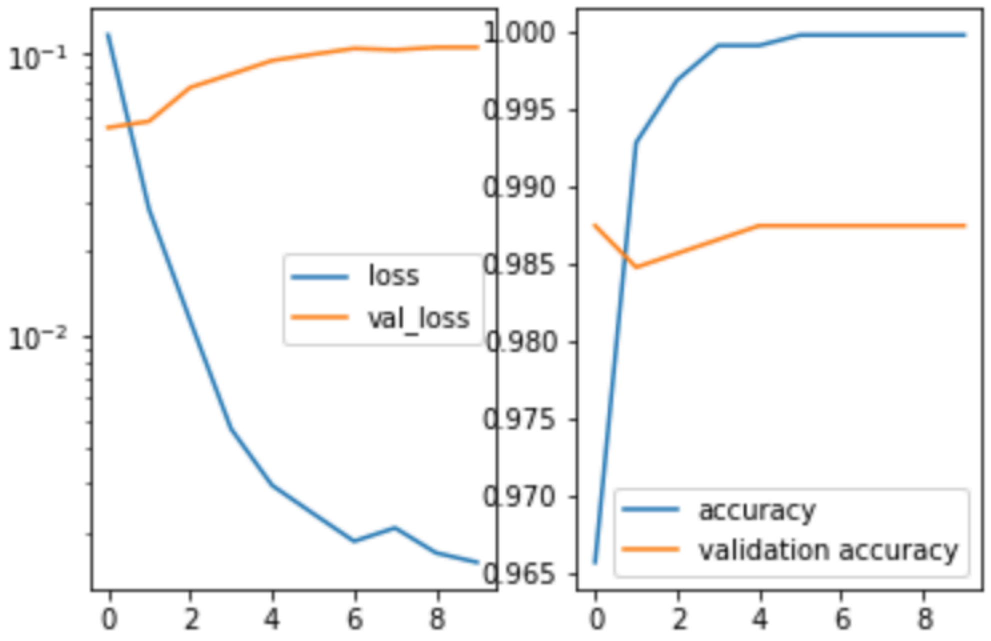


Figure 6. Live-loss and Accuracy Plots after Epoch 10, , *Batch size: 32, Epochs: 10, Validation split: 0.2*

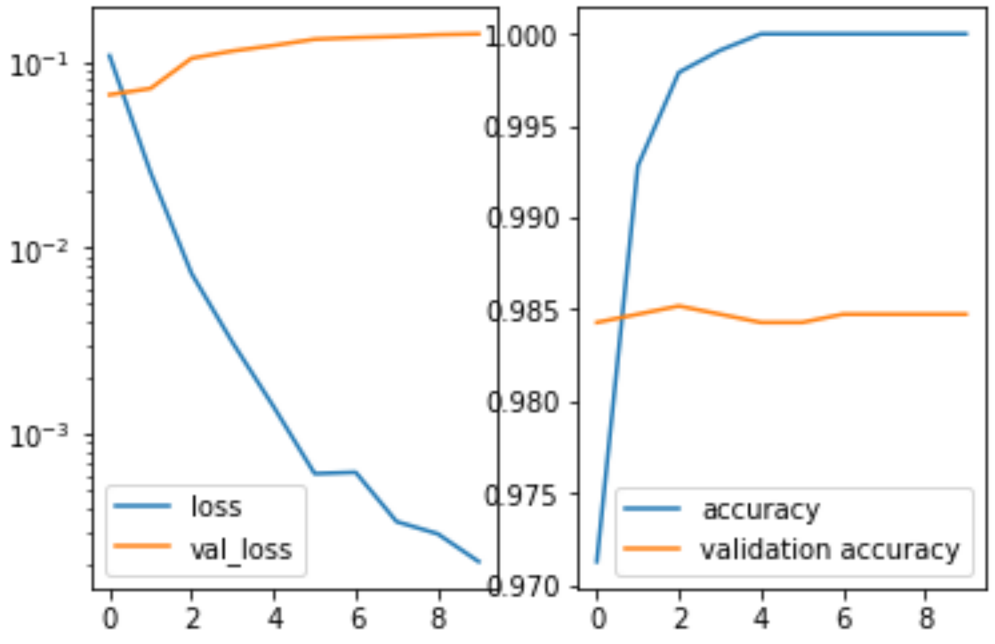


Figure 7. Live-loss and Accuracy Plots after Epoch 10, *Batch size: 32, Epochs: 10, Validation split: 0.4*

The two plots above show the changes in the live-loss and accuracy plots at two different points during training. The horizontal axis is the number of epochs while the vertical axis represents the values of loss and accuracy respectively. From the plots above, we see that the training loss

(loss) values are significantly lower than the validation loss (val_loss) values. Indeed, after 10 epochs and increasing the validation split to 0.4, we see that the training loss curve has an even steeper curve which represents the improvement the model has made in fitting the training data. As the possibility of the curve overfitting was discussed briefly above, the low training loss values seen at the tenth epoch provides additional evidence of the model overfitting to the data. Also, with each successive plot displayed above, we also see the gap between training and validation loss increasing, again supporting the conclusion that the model is overfitting to the data. Similarly, when we look at the plots of accuracy against validation accuracy, we see that the accuracy approaches 100% in all three plots. However, when we compare the last two plots (validation split of 0.2 against validation split of 0.4 after 10 epochs), we see that the validation accuracy of the last model that had a validation split of 0.4 is lower than that of the previous model that used a validation split of 0.2.

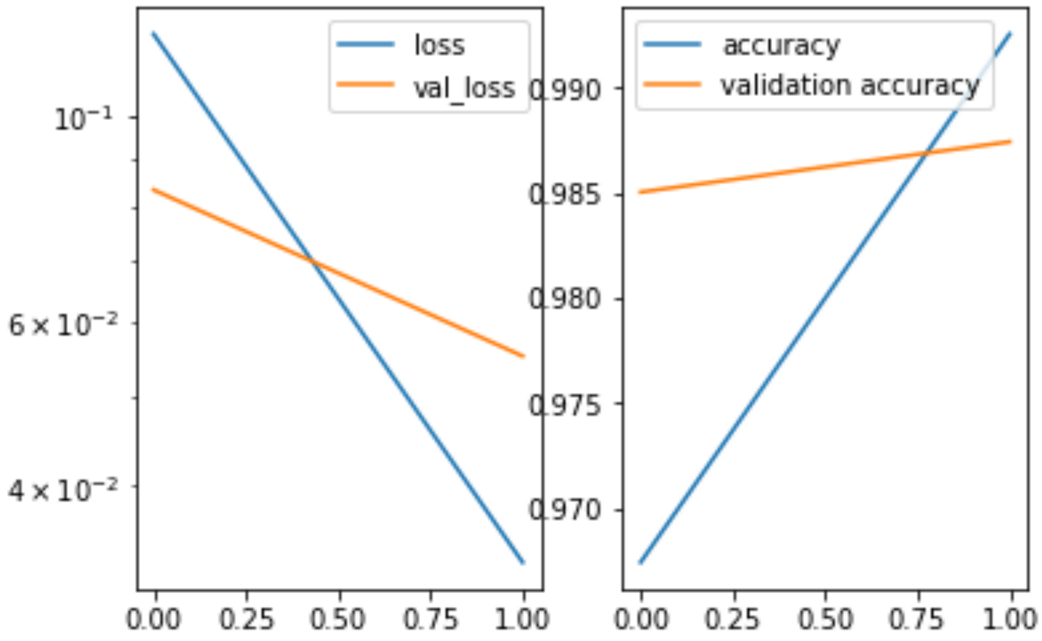


Figure 8. Live-loss and Accuracy Plots after Epoch 2, *Batch size: 32, Epochs: 2, Validation split: 0.3*

Indeed, when the number of epochs is lowered to 2, we see the gaps between both training, validation loss as well as training, validation accuracy less pronounced than above. While the overall accuracy of this model was 99.26% and F-1 score was 94.76%, this model might be more reliable as it shows fewer signs of overfitting. However, this can only be verified when applied to a more extensive dataset, something that is noted in the improvements section.

Reflection

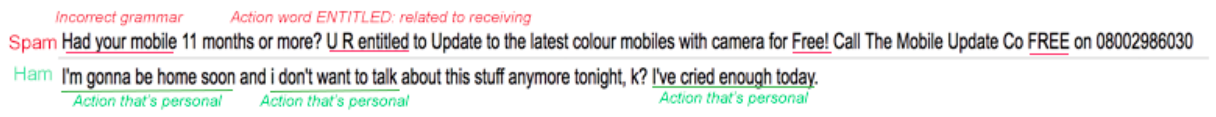
Overall, this project was a very hands-on and educational experience with applying the machine learning techniques learned over the course of the last several months. The project is the epitome of all the things that I have learned so far, and I enjoyed being able to put together all these pieces in 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. The fact that SMS spam detection is far behind email spam detection motivated my interest to explore the SMS spam dataset with a CNN.

Once I had decided that I wanted my project to be focused on the topic of SMS spam detection, I began by opening up the dataset and looking at its statistics. To do this, I added a column to the dataset that mapped each message to its length. From there, I looked into the statistics of the message lengths across the dataset and also identified outliers in the dataset. Seeing this overview at the start was helpful as it gave me a better idea of what I was working with. While starting, I also looked into publicly available kernels that accompanied the SMS spam dataset on Kaggle. These kernels made use of a variety of machine learning algorithms and techniques. They were also incredibly helpful to reference when I wrote my own development code as I could reference code that had proven to work. In the process, I also looked extensively at the official documentation for Keras and Scikit-learn as well as on StackOverflow for specific debugging issues.

In fine tuning the final CNN model I arrived at, I only adjusted one parameter at each run because I wanted to make sure that any perceived change could only be due to the parameter that had been changed. While this took longer as I had to run the model each time I changed a parameter, it was helpful in figuring out what parameters worked better or worse for my model.

Even though I ran the model each time I adjusted a parameter, it was still relatively quick in completing training across all 10 epochs. With fewer epochs, the model would have been even faster.

Improvement


Spam Had your mobile 11 months or more? U R entitled to Update to the latest colour mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986030

Ham I'm gonna be home soon and i don't want to talk about this stuff anymore tonight, k? I've cried enough today.

Incorrect grammar
Action word ENTITLED: related to receiving
Action that's personal
Action that's personal
Action that's personal

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.

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. Additionally, a larger dataset would also be advantageous for testing whether or not the current model is overfitting or if it is effective in classifying spam from legitimate messages for a different dataset. 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. Additionally, recursive neural networks (RNN) have become top models for natural language process as the layers in RNNs “feed-forward,” taking learned results and applying them forward through the layers.