# SMS Spam Filtering using NLP and Deep Learning

Berkeley D. Willis

University of Bellevue

Abstract

Spam, originally known as junk mail until a Monty Python skit, has existed for decades if not longer and only escalated to ridiculous heights with the creation of email. In email is where it became infamous and so common place to receive hundreds or possibly thousands of messages in an inbox. This of course led to spam filters in order to reduce the number of spam emails. These filters are built using different methods, using simple filters for titles, email endpoints, and eventually machine learning models. This didn't stop the senders of spam, and eventually with the entrance of Simple Messages Service (SMS) brought new challenges. SMS which is commonly received today as a text message on a cellphone is treated a little bit differently by users, which makes them in some ways riskier to the user. However, similar spam filters integrating deep learning techniques and NLP would be able create effective SMS spam filters. This though also has new challenges, because commonly text messages can have much more relaxed speech and shorthand that is used and requires new NLP techniques in order to increase their effectiveness. Using these techniques this project was able to create a set of deep learning models through attempting to translate the shorthand to full text and lemmatization models with 98.8% accuracy.

*Keywords:* Deep Learning, Spam, Embedded Learning, Bi-directional LTSM, NLP, Tokenization

SMS Spam Filtering using NLP and Deep Learning

## Introduction & Background

Spam emails have always been a bit of a problem, with them commonly having some type of scam that is targeting the recipient, and trying to get them to do something that they usually wouldn't. This would be one of the primary reasons why spam filters were created, in order to try to both help manage the amount of spam email that can be sent/stored by the service provider, and it helped protect user's from these spam messages.

Eventually SMS came onto the scene and with smart phones allowing internet access allowed scammers even more access to people, and even possibly more ways to exploit the recipients of these new messages. This type of phishing messages via spam is really quite successful, and may be more successful than email spam. There is a reason why these are so successful and easy for people to fall for, they write that it's primarily the manipulation just like email spam but SMS phishing isn't publicized as much which means that the best practices and awareness advice isn't shared as much for SMS spam (Moore). Though it is true that it does use the same type of manipulation and that SMS spam isn't publicized as much, it could be argued it's also more successful because of the ease of access to the end user. Email is very accessible, it is true, but many people are even kind of trained to respond or handle text messages in a much more rapid manner. An average person is likely to respond to a text message faster than they would and email, and with the immediacy of text messages it could help create a heightened emotional state when trying to take advantage of the recipient.

When talking about the general content and how to avoid falling victim to one of these messages, it is still very similar to how someone would handle an email spam message. The FTC

even does have an article on recognizing and reporting these messages, saying to be aware of messages promising free gifts, prizes, offering low interest credit cards, paying off student loans, suspicious activity on your accounts, or even package notifications (FTC). These are very similar tactics by trying to invoke some type of emotional response in the recipient, many of the examples here being that they would receive something for their cooperation. Though something to think about is the fact that many companies also do have access to us and give us updates on progress or orders, like those that deliver packages. A good example of this is commonly found in Amazon messages, because they give SMS updates on deliveries but a common spam message is for someone pretending to be them saying that the receiver had won a sweepstakes (Fripp). Again this extra access these companies have made it much more familiar to a user to receive a message from them, possibly lowering their guard.

The amount of spam that is created and sent is massive as well, collecting and utilizing data on millions of users. According to a single Techcrunch article, a single SMS spamming operation that produced tens of millions of messages had a database of over 80 million people and sent 50 million messages a month with millions of records of successes (Whittaker). This gives a bit of scale to the problem of SMS spam, because this is just a single operation that was shut down. This in combination with other challenges of cleaning the data is what makes this problem difficult and interesting in order to create effective models to identify spam messages.

## Method

### Data Collection

The data was collected using kaggle again, containing really a collection of different spam and non-spam SMS messages. The actual origins from this dataset though is from a

different collection effort and a paper studying the new datasets studying filtering with different results with the new dataset (Almeida, et. al.). This basic dataset also does have a format that has multiple messages, many of them seem to be some number of messages in succession without responses in between them. When taking a look at the messages it seems that to make them useful it might take some serious analysis and cleaning, depending on what the goals of the model would be.

Any other data that would be used is going to be from libraries and be described further in the tools section below. Any "new" data would be in an attempt to translate the text messages shorthand into full form, for examples "u" to "you" or "wkly" to "weekly." In all actuality if the tools work the data isn't really added, just merely transformed into their full form in order to create a more condensed tokenized library.

**Analysis**

The analysis conducted was to evaluate the state of the data, decide what features will be relevant for a classifier model, strategize on how to best clean the data and how to transform it into the most useful form for a model. The initial look revealed that the message pool had 4,825 non-spam messages and 747 spam messages, totaling to 5,572 messages and approximately 13% of the messages being spam.

Next, looking at what features would likely matter, for one some type of metadata such as message length could very well be relevant. In order to do this all of the columns that contained message data were combined into a single field, and lengths were calculated from this combined field.

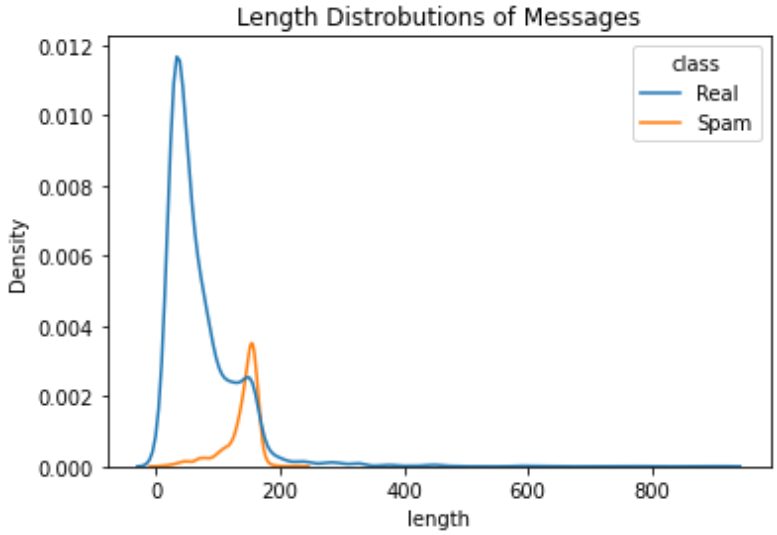


Figure 1: Length calculations distributed and split by classifications

In Figure 1, some interesting observations can be made, seeing that the bulk of real messages are typically shorter than the bulk of spam messages. This does mean that this will likely be information needed for a model, if not a feature itself a length it can help identify boundaries since it may take more data in order to identify spam messages. Even in another paper where the writers had created multiple non-deep learning machine learning models, they had noticed that length was a very reliable feature for them to identify spam messages and helped them create models up to 97% accurate (Shirani-mehr). This means that length could be another very easily viable feature that can be wagered in deep learning, even if NLP tokenization is being used with some creativity.

Something else that would be helpful from exploring the data, is seeing some of the more common words for each classification of messages. This did require a slight bit of data cleaning, just primarily pulling out the numbers and special characters, after which word clouds were

created to visualize the prevalence of certain common words where capitalization was still left in.


Figure2: Word Clouds of non-spam and spam messages respectively.

In Figure 2, it can be seen that there were some very common words and some shorthand for symbols that are in both. The more interesting cloud here though of course is the one for spam messages. A few of these words are what would be expected, asking for a call, a text message, to claim something, stop, cash, and of course free. Many people that receive messages that have this type of offer if they cooperate are more likely, because they see that they will get something out of it. Despite some initial thoughts that keep the capitalization with this text, it is somewhat possible to believe that if they have the word free in all capital letters that it is pretty likely to be spam. However, according to a few papers that were able to create a very effective convolutional neural network that the most crucial step for them was the preparation of the data, which included throwing them all to lowercase and more (M. Popovac et. al.). There are specific reasons why this is required for the creation of neural networks that will be elaborated when describing the models for this project.

***Tools.*** In order to clean that data properly, create a tokenization dictionary to translate the data into a matrix form, to run lemminization, and to translate some of the abbreviated text into its full form. Much of the basic operations like filtering can be done by the standard library, but

for more advanced Natural Language Processing operations will be handled by the NLTK library. The lemminization process is basically taking a word and reducing the inflection forms and other derivational forms to the common base word, for example from “am”, “are”, or “is” to the common form “be.” However, this lemmatization operation from the NLTK library won’t be very effective with the multitude of forms that would be simplified by it, which is where the GingerIT library will help. Using the GingerIT library text can be sent to an offsite API where the text is read as a whole, identifies abbreviations of words and returns the full text with all of the abbreviations replaced with what is seen as the most probable word or phrase. Finally the library that will be used to create and train deep learning models will be Tensorflow. Any other extra libraries might be for ease of operation or speeding up certain operations through parallelization.

**Data Cleaning and NLP Operations**

The primary reason that this data has to be cleaned so thoroughly is because much of this is going to come down to the tokenized library. This tokenized library is going to identify the most common words in all of the text, which means different capitalizations of certain words or different forms will cause the available vocabulary of the tokenizer more limited or even less accurately representing the collection of text. The idea is to reduce the tokenizers library in order to make it very representative and highly accurate, and hopefully giving better insights in ways to identify spam messages. The first obvious way to do this is by removing special characters and numbers, which in many cases might not give any type of good indication of being spam or not considering it would likely just be missing context. A quick example would be a number that

could possibly represent an amount of money or an address, and end up just taking up space in the vocabulary of the tokenizer.

Something to keep in mind if these types of data cleaning and NLP operations that are required to create the mode would have to be re-implemented to do the same operations on new incoming messages in order to run them through the model for classification. The reason why this must be kept in mind is scalability in production, because if there is a slow section of the program and there is no way to speed it up it might not be a fully feasible solution. Reason being is because there are billions of messages being sent every day, so any single service provider is going to be dealing with tons of data for spam filtering. This will be seen as a bit of an issue when it comes to the usage of GingerIT, and compensating measures that had to be put in place. In a fairly large study on some basic email spam detection models, the deployment of such models is given an entire section just because it required a completely different strategy to make them effective with multiple rules and operations, down the specific language that would be used in such a deployment for SpamAssassin (Massey et. al.). This paper won't go into such detail, but some considerations will be given or suggestions in order to compensate in a large scale environment.

The way to make many of the deep learning models work with these is to create a tokenizer, which will take all of the various text and create a vocabulary dictionary of all of the most common words to a certain size of the library. This is an NLP operation to create this type of tokenization library using NLTK. Then after the tokenizer is created the text is then converted to a sequence, which is basically just an index position of a word in the tokenizer's vocabulary if it exists as such, to a defined length. However, this could possibly create a large number of

sequences of different lengths, which will not be able to run through a unified model. This can be very easily solved through padding, which just takes a sequence and adds 0s until the sequence has been filled to the specified length in order to make all sequences the same length.

At this point it could technically be run through deep learning models and effective models could be created, but there can be much more done in order to possibly shrink the vocabulary down and make it more accurate on a large scale. The more broadly a tokenizer's vocabulary with less repetition the more patterns it might be able to get because of the condensing of information. One of the ways this can be done is via lemminization, another NLP operation that is already supported in the NLTK library. This process will essentially take words in different forms or conjugations and simplify into a common form, such as "apples" to "apple" or "children" to "child" all in order to compact the tokenizer's vocabulary as far down as possible. This would have to be done after the full cleaning process for any words, in order to get them all in their near final form so that they can be converted to their base form but probably before the final removal of all special characters.

Though this does leave in a lot of text that can't be transformed via lemmatization because they are misspelled or they are in shorthand. Many of these can either intrude in the vocabulary or possibly drop important terms out of it completely because of the dilution of it's multiple forms. This is also colloquially very common in SMS messages, to shorten longer words or entire phrases down to make it faster and easier to type out like "you" to "u" and "be right back" to "brb" with many other variations that are possible. GingerIT is a library that is able to handle this type of transformation, and can take entire messages for transformation. When evaluating multiple libraries that could do this, this library is one of the few that not just fix

misspellings but convert shorthand to full text. However, upon closer inspection of the library it doesn't just run a model immediately on it, but actually sends it to an offsite API which seems to be throttled in some way. This isn't directly an issue, but it does mean in order to make this easier to handle for larger sets is to parallelize it in hope that it will get more access to the API's resources. This is another spot where this might be able to be improved further with other changes. Nevertheless, it needs to be run before any possible cleaning because some shorthand will rely on in some cases, but should be run right before lemmatization. This should further reduce the vocabulary that the tokenizer needs to track for, and be able to better generalize the data and handle it.

The final steps in preparing this data would be taking the vectorized data and possibly adding on any other important features that can be represented in numeric form, such as the length. The length of the messages can be somewhat incorporated by the length of the sequences and the max length of the vocabulary in the tokenizer, but that doesn't give the model the full information of the length of the message as a feature. So with this in mind, if there are any features such as this or any other type of numeric identifiers, they will need to be added after the sequences are created and padded out, so that they are in a specific area of the input matrices for the models to use.

**Model**

The models that will be used are going to be wagering on deep learning using the keras library with dense layers, there will be a different implementation using LSTM, and lastly a bi-directional LSTM or bi-LSTM models. These basic deep learning models are here to see which ones using the NLP techniques are most effective in combination. Creating dense models

is a good starting place, but LSTMs are good for finding trends with more context of sequence order. This way LSTM as a recurrent neural network can try to identify order trends in hopes to find patterns in words that might better indicate if it is a spam message. However, this is only in a single direction, sequences only going forward and doesn't fully use the context of the sequence behind it. This is where bi-directional LSTMs come in, the same type of neural network but does sequence analysis going in both directions to get context of sequence ahead and behind. This should theoretically have some advantages, and should be further enhanced by the NLP modeling and operations.

For the NLP modeling to clean and set up the data for the deep learning models, the idea again is to get it to be not only clean as possible but to enable the deep learning models to better generalize the data for pattern recognition. Each one of these will be added incrementally in hopes to see which are the most effective. The overall model, or rather the most built out, is to send all of the text to lowercase, run the text through the GingerIT library to get the fullest form possible, clean the rest of the special characters and numbers out of the text, build the tokenizers with padding for shorter messages, and then ladd any relevant metadata features such as length to the padded sequences. The last phase isn't technically NLP, but will be part of the same functions, just to give as much helpful feature data for training. As well since the LSTM are very sequence oriented models, the only ones that might be able to wager the other features might only be non-recurrent neural networks since this would be sequence independent.

This will result in many models that will be built and tested, each of these individual measures during training and validation of the data will be stored in the Appendices for further review.

## Results

The results are quite revealing that the accuracy of such a deep learning model is already very high performing to begin with, an accuracy around 98%. These accuracies make it difficult to make sure that the change between the models and their data preparation are the causes for changes in the accuracy. To try to adjust for this as much as possible, the splits in data were decided to use the same seed, so that the indices for each training and testing dataset were the same and therefore the same entries. However, there is a small amount of randomization that still enters into these models, which gives them a slight amount of variation in combination with the changes in data preparation.

| Model Type | Loss | Accuracy |
|---|---|---|
| Embedding Dense Model Using Basic Cleaning w/metadata | 0.050969 | 98.2960% |
| Bi-LSTM Model Using Basic Cleaning w/metadata | 0.135638 | 97.2791% |
| Embedding Dense Model Using Basic Cleaning wo/metadata | 0.050164 | 98.7444% |
| Bi-LSTM Model Using Basic Cleaning wo/metadata | 0.101603 | 96.7695% |
| Embedding Dense Model Using Cleaning With Stopwords | 0.051686 | 98.6547% |
| Bi-LSTM Model Using Cleaning With Stopwords | 0.077817 | 98.5592% |
| Embedding Dense Model Using Cleaning With Lemmatization | 0.050125 | 98.7444% |
| Bi-LSTM Model Using Cleaning With Lemmatization | 0.074230 | 98.5780% |
| Embedding Dense Model Using Cleaning With All Strategies | 0.049945 | 98.3856% |
| Bi-LSTM Model Using Cleaning With All Strategies | 0.065849 | 98.8000% |

Table1: Loss and accuracy measures for each model in the project run.

Using Table 1 above, it can be seen that the variation between each model's accuracy and overall loss when evaluating with only the test data was very small. Out of the models that were

recorded above the best performing models were the combination strategies with the usage of a Bi-LSTM. Keep in mind that the differences between these are so small and gains are quite hard to measure in this very limited dataset with just under five thousand records. There is also a possibility that there is a margin of error since the original models are so very accurate.

One fairly certain take away though is that the method of combining the metadata with the sequence data this way wasn't successful, but that is primarily because these models are sequence based for the most part, especially with LSTM which are recurrent neural networks. There are some possible ways that this can be fixed using different methods of combining this data in another model.

## Discussion and Future Works

The findings were quite revealing but leave a lot of possible wiggle room since the difference between some of these models are tenths of a percent which is significantly small. Part of this is again the overall accuracy of the more basic models and strategies of cleaning. There are some things that could be improved in the amounts of data that are being used, methods of cleaning, and the creation of the models created that would give more conclusive data.

The data that was used is not small, but it could be much larger in order to make a more accurate model, or at least give more confidence in the averaging of such measures. The data that can be cultivated could be more recent, which one of the more common things that are seen in spam SMS messages today like clickable URLs. However, this type of data likely won't be effective if it is introduced in the models that require sequential data. Instead, on method that may work better is creating a new model that uses this type of data that can be used in another Machine Learning technique that can handle this type of data more appropriately, such as linear

regression or convolutional neural network. Then this new model can be incorporated with the better performing models that use sequential data in an ensemble model. If they do become more accurate, the question would only be how to best build out a production implementation.

# References

Almeida, T.A., GÃ³mez Hidalgo, J.M., Yamakami, A. Contributions to the Study of SMS Spam Filtering: New Collection and Results. Proceedings of the 2011 ACM Symposium on Document Engineering (DOCENG'11), Mountain View, CA, USA, 2011.

Deep, M. 25 July 2020. The Ultimate Guide To SMS: Spam or Ham Detector Using Python. Retrieved from https://towardsdatascience.com/the-ultimate-guide-to-sms-spam-or-ham-detector-aec467aecd85

Fripp, C. 18 March 2021. Got this text from Amazon? It's a fake. Retrieved from https://www.komando.com/security-privacy/amazon-texting-scam/783190/

FTC, N. D. How to Recognize and Report Spam Text Messages. Retrieved from https://www.consumer.ftc.gov/articles/how-recognize-and-report-spam-text-messages

Massey, B. Thomure, M. et. al. N. D. Learning Spam: Simple Techniques For Freely-Available Software. Retrieved from https://www.usenix.org/legacy/publications/library/proceedings/usenix03/tech/freenix03/full_papers/massey/massey_html/spam.html

M. Popovac, M. Karanovic, S. Sladojevic, M. Arsenovic and A. Anderla, "Convolutional Neural Network Based SMS Spam Detection," 2018 26th Telecommunications Forum (TELFOR), Belgrade, Serbia, 2018, pp. 1-4, doi: 10.1109/TELFOR.2018.8611916.

Moore, J. 22 January 2021. Why do we fall for SMS phishing scams so easily? Retrieved from https://www.welivesecurity.com/2021/01/22/why-do-we-fall-sms-phishing-scams-so-easily/

Pradeep Kumar Roy, Jyoti Prakash Singh, Snehasish Banerjee, Deep learning to filter SMS Spam, Future Generation Computer Systems, Volume 102, 2020, Pages 524-533, ISSN 0167-739X, https://doi.org/10.1016/j.future.2019.09.001.

Shirani-mehr, H. (2013). SMS Spam Detection using Machine Learning Approach. Retrieved from http://cs229.stanford.edu/proj2013/ShiraniMehr-SMSSpamDetectionUsingMachineLearningApproach.pdf

Shrestha, S. 26 July 2019. NLP: Spam Detection in SMS (text) data using Deep Learning. Retrieved from https://towardsdatascience.com/nlp-spam-detection-in-sms-text-data-using-deep-learning-b8632db85cc8

Whittaker, Z. 9 May 2019. An unsecured SMS spam operation doxxed its owners. Retrieved from https://techcrunch.com/2019/05/09/sms-spammers-doxxed/

## Appendix A

How Cleaning Changed WordClouds

The original WordClouds that were generated were using the raw data with little changes, so it brought the question of what had changed after all of the data preparation stages had changed anything. Ideally, once again, it should change a few things since it's reducing the overall vocabulary to something that is more manageable and better generalizes the text contained in all of the messages.


Figure 3: New WordClouds generated from the text that has been operated on.

The newer visualizations did clean up a few things, like removing junk words or single characters, as well as in a few cases changing the sizes of some of the words and their significance. Some of the more common words that are in spam messages show very similar words but the size has changed quite a bit.

## Appendix B

Using Metadata in Models

The thought here was that the more possible data points that a deep learning model could use, the more accurate model. At least it would theoretically make it more accurate if the data would make a good classification predictor because of some type of strong correlation between the class and the data point. This is true of the length of an SMS message from Figure1, where longer messages were primarily spam messages. When integrating this data in the sequence data it was put in the beginning and sent through the same types of models.

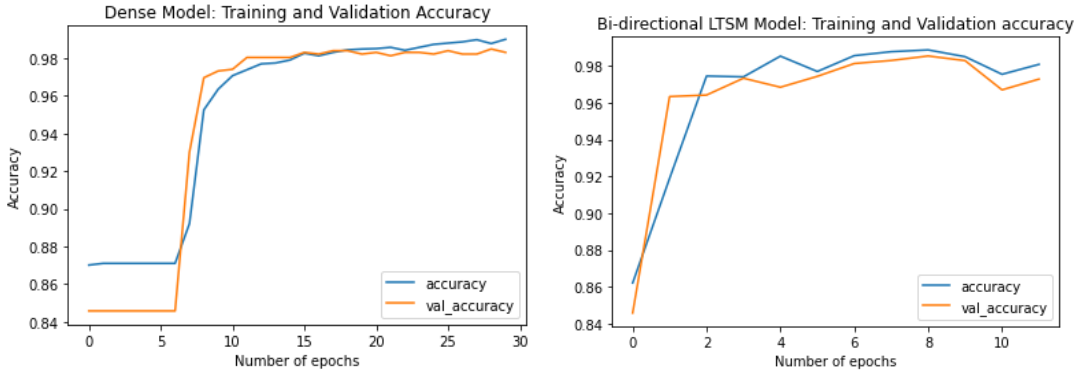


Figure 4: Accuracy trends from training models with metadata inclusion

The performance of these models wasn't bad by any means, it just wasn't the best out of all of the other models, even if the models were rerun the results were never really as good.

## Appendix C

Using Same Models Without Metadata

In order to figure out whether or not the metadata would work properly and well with the sequential data is already being processed by the model. When removing the metadata and re-running the same exact models, they seemed to improve the classifiers slightly. This is likely just because it is not the same nature of the data. This indicates that this type of metadata from the text or other data points that are just not sequential would just fair better in other types of Machine Learning methods.

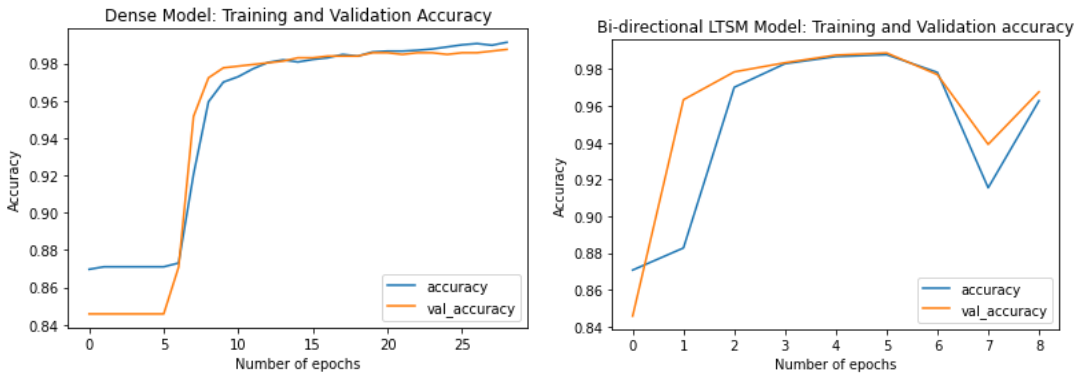


Figure 5: Accuracy trends from training models without the metadata inclusion

Looking at the performance of these it did improve without the metadata, which isn't much of a surprise. This just means that metadata will not be attached at the beginning or the end of subsequent runs of the models.

## Appendix D

Models With Removing StopWords

Now to continue to build the same model types with differing data preparation and NLP operations in order to change the tokenizer and the vocabulary used. In this run the data still had the basic cleaning operations, but this time stop words are being removed in order to reduce the vocabulary dirtying the tokenizer's sequences. These could be just singular characters like "u" or "i" and others that don't offer a lot of information.

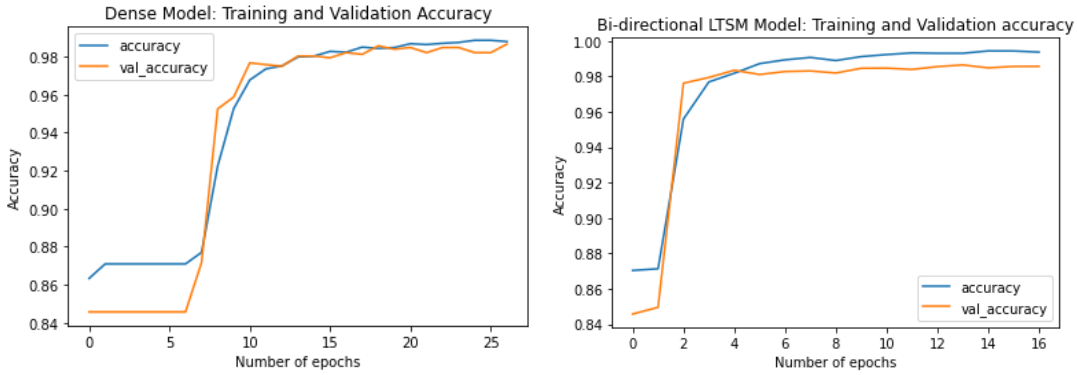


Figure 6: Accuracy trends from training models with the stopwords removed before tokenization

After removing these extra words to reduce and simplify the vocabulary, the performance didn't improve as much as hoped. However, this could just be a small difference due to some slight randomization on the creation and fitting of the model.

## Appendix E

Models After Running Lemmatization

After running the previous model the next run will be using even cleaner data. This time it will be run with the same cleaning operations, removing stop words, and then running the lemmatizer to reduce the vocabulary to common words. A possible unintended consequence though might be that the usage and context related to this is removed, not a major concern but a possible issue or reason for any dip in performance.

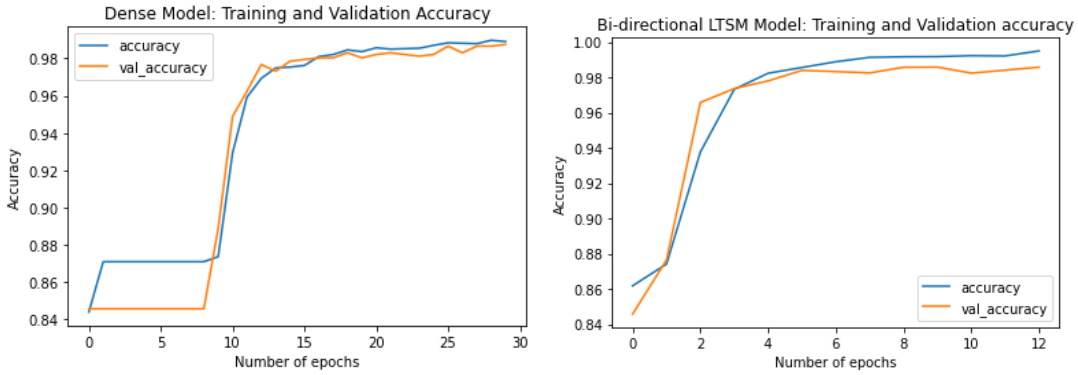


Figure 7: Accuracy trends from training models after cleaning and lemmatization

The models had performed rather well here, and didn't seem to have any major issues in processing the sequences. When running experiments this was one of the better performing models on reruns as well.

## Appendix F

Models After Translating Shorthand

This was the last model that was experimented with, in hopes that the continuous reduction and simplification in the library would continue to improve the models. This time the text would immediately be sent directly through the GingerIT library in order to translate any possible shorthand to the full form, then it would be run through the cleaning process, removing stop words, and running the lemmitzer again. The idea here is that data is possibly unintentionally removed during the cleaning process because of the short hand and of course it not making sense.

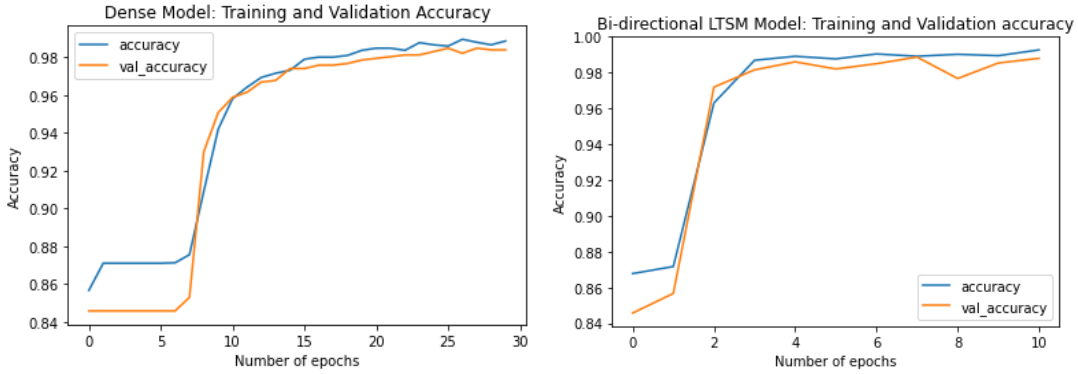


Figure 3: Accuracy trends from training models after translation and lemmatization

The performance for this was the best out of all of them, but it usually had a bit of a dip on certain runs. This is likely due to some unfortunate issues with the library, it is fairly reliable but there wasn't an available model that was highly reliable and got these types of translations and corrections.