Jason Rumengan

From:	no-reply@qut.edu.au
Sent:	Saturday, 11 June 2022 7:12 PM
To:	Jason Rumengan
Subject:	We've processed your Assignment extension request FORM-AEX-144931

Hi Jason,

Thank you for your assignment extension request **(FORM-AEX-144931)**.

We have approved your request and the due date for your assignment **Assignment 2**, for unit CAB420 has been extended by 48 hours from the original due date. If your unit outline does not specify that your assignment is eligible for an extension, this confirmation email is not valid and unless you submit by the original due date, the late assessment policy will apply.

You are responsible for ensuring that this assignment is eligible for extension before submitting it after the original due date. Check your unit outline for eligibility.

As you indicated this is a group assignment, you are also responsible for informing other members of your group of this extension.

Be aware that a copy of this email is kept on file. You should not alter this email in any way. Email notifications that have been altered or differ in any way from the original may result in an allegation of student misconduct as set out in the Student Code of Conduct.

Need extra support? You can access free, confidential counselling with qualified professionals. We also offer planning and support if you have a disability, injury or health condition that affects your ability to study. If you are struggling to meet your university commitments, academic support is also available.

Have a question? You can contact us by email or phone. We're also available to help you on campus or via online chat. Visit the HiQ website for our contact details and opening hours.

Email us

+61 7 3138 2000

HiQ is your place to go for **student services, support** and **general enquiries:** qut.to/abouthiq

hi, how can we help you?

Ref ID: 10553185 FORM-AEX-144931

Twitter Spam Detection
Assignment 2 – CAB420

Group 1

Member	Student ID	Email
Gilbertus Priastian	11137444	n11137444@qut.edu.au
Thomas Fabian	10582835	N10582835@qut.edu.au
Jason Rumengan	10553185	jason.rumengan@connect.qut.edu.au
Rodo Nguyen	10603280	n10603280@qut.edu.au

Contents

1. Introduction/Motivation

Social spam is unwanted spam content appearing on social networking services and any website with user-generated content (blog, comments, chat, etc.) (Markines et al., 2009). Some common appearances of social spam are bulk messages, hate speech, malicious links, false information, fake news, and others. Spam has negative impacts on the people exposed to spam, such as swaying opinions due to false information or fake news. For example, a Twitter spam operation was implicated in disseminating political tweets predicated on false information during the 2020 US elections (Collins, 2020). It is also notable that spam is not only done by humans but by computer programs as well (or 'spambots'). Therefore, both the speed and the scale of spam can accelerate quickly, as shown by Lee et al. (2021) being able to collect over 2.3 million spam tweets over seven months from December 2009.

Twitter is a typical example of this problem due to its high number of spambots in recent years. This micro-blogging website is an appealing target, since it exhibits an enormous number of daily users, with 229 million recorded in the first quarter of 2022 according to Associated Press (2022), and numerous influential figures around the world sharing their messages and ideas.

Given the potential societal impacts of Twitter spam and the magnitude of spam tweets precluding the manual inspection and removal of spam tweets, the application of machine learning (ML) algorithms for Twitter spam detection has become a prominent topic in research, as noted in Section 2. Spam detection using non-ML methods has been researched in other media, such as email, before this (Park & Deshpande, 2006), and ML models (a hidden Markov model) were applied shortly after (Gordillo & Conde, 2007). This report aims to research traditional and state-of-the-art machine learning algorithms, evaluate the performance and runtimes of their implementations, and compare the algorithms, in the context of detecting spam tweets.

2. Related Work

In this game of cat-and-mouse between spammers and spam detection system developers, computational classification techniques have been optimized to increase the accuracy of these systems. Various papers on text classification; particularly in spam detection in emails, tweets, reviews, and SMSes; and tweet sentiment analysis; were investigated to find viable algorithms for Twitter spam classification systems. This is because text classification models may be generalisable to Twitter spam detection. These papers cover a variety of deep learning (DL) and non-DL models, such as Support Vector Machines (SVM) with and without Bag-of-Words (BoW), Naïve Bayes (NB), Random Forests (RFs), K-Nearest Neighbours Classifiers (CKNN), Artificial Neural Networks (ANNs), Decision Trees (ID3, Cart, ADTree, J48), Recurrent Convolutional Neural Networks (RCNNs), Long Short-Term Memory (LSTM), Random Multi-model Deep Learning (RMDL), and Transformers.

For non-DL models, NB and SVMs, the algorithms investigated in this report, were found to be the most reliable by Wang in 2010, as these algorithms retained performance, even when models were trained on imbalanced datasets comprising either a portion of similar samples or static dictionaries. In the same year, a further study by Benevenuto et al. (2010) investigating SVMs, the state-of-the-art machine learning classifier at the time, found that removing features also minimally affected results provided features related to the tweets' contents were kept, as Twitter spammers often forge their user features, such as the number of followed accounts and/or followers, to mimic non-spam accounts.

In 2011, McCord and Chuah extended this study by implementing RFs, outperforming both NB and SVMs on imbalanced Twitter spam detection datasets. Sharma (2011) then compared Decision Trees with RF and found Decision Trees improved email spam detection performance. However, both papers pre-processed data into aggregated numerical representations instead of word-based representations, which may limit the information learnt by the model. As a result, RFs and Decision Trees will not be considered.

Gautam & Yadav (2014) then revisited NB by combining it with WordNet Semantic Analysis to analyse tweets' sentiments, resulting in the best accuracy compared to NB alone and other methods such as SVMs, proving its superiority in finding semantic and lexical relations. While this may apply to detecting relations that indicate spam, this is deemed to be out of scope for this report due to the complexity of semantic analysis.

As researchers began to focus on understanding the interrelationships between words, Saab et al. (2014) implemented ANNs in email spam detection and found that their performance is on par with previous machine learning classifiers, particularly SVMs. However, they note that ANNs are less affected by feature reduction because of their layered network architecture.

Sharaff (2015) then attempted a comparison of traditional approaches, such as the J48 Decision Tree and LazyIBK, Weka's K-nearest neighbours classifier (CKNN) implementation in email spam classification. However, these models returned underwhelming results compared to methods such as SVMs and NB, as CKNNs, without proper hyper-parameter tuning, are known to be susceptible to noise affecting performance.

For DL models, LSTM and Transformers were selected for this report, as they can consider relationships between words in text. In 2015, Lai et al. experimented on DL models in general text classification by using an RCNN, an architecture combining deep CNNs (DCNNs) consisting of convolutional layers and recurrent NNs (RNNs) comprising multiple cells learning on elements of a sequence in order. This yielded better accuracy compared to other ML classifiers examined. This proves the ability of not only DCNNs to learn richer representations of data using many learnt layers, but also RNNs to retain more information from previous words in the sequence. This is because DCNNs may not explicitly consider information from previous words in a sequence in training and inference, while RNNs do.

This rise of DL in spam detection continues with the work of Sharariar et al. in 2019, where LSTMs returned better results than RCNNs and other classifiers on spam customer review detection. This is because LSTMs' architecture allows models to ignore irrelevant information from past time steps and update the model's cell state with useful information, solving RNNs' limitation of 'forgetting' information from early elements in a sequence.

Continuing in 2020, Gomaa applied an RMDL model that uses a deep ANN, DCNN, and deep RNN in one model for SMS spam detection, outperforming the LSTMs, RNNs, and RCNNs. This is because predictions are made using an ensemble of models, vastly improving performance by greatly increasing the model's capacity to learn diverse patterns and representations in the data.

Finally, transformers, the state-of-the-art model in natural language processing, were examined in spam detection in 2021 by Lie et al. This showed that transformers had the highest accuracy compared to previous DL models in SMS spam detection, as well as spam detection using the same utkML dataset used in this report. Transformers achieved this using the new 'multi-head self-attention' mechanism that allows transformers to compute the relationship between each element in a sequence in parallel.

3. Dataset and Evaluation Protocol

3.1. Dataset

The dataset to be used is taken from the University of Tennessee's Machine Learning Group (utkML)'s Twitter Spam Detection Competition on Kaggle (2019). This dataset was selected, as it contains enough data to reduce overfit risk on non-deep learning methods, while also balancing the computational demands of training on too large a dataset for deep learning methods.

The 'training' set is the dataset used in this report. Given the competition 'test' set is unlabelled, the set was not used. The competition's 'training' set for this competition contains 12,598 tweets, with the columns in Table 1, 'Type' being the response. Given the 'Id' column is only an index, the column was not used.

Column name	Description	Type
Id	A unique identifier for tweets in the set	Positive integer index
Tweet	The text of the tweet	Text string
following	The number of accounts following the account that posted the tweet	Positive integer
followers	The number of accounts followed by the account that posted the tweet	Positive integer
actions	The total number of favourites, replies, and retweets associated with the tweet	Positive integer
is_retweet	Whether the tweet is a retweet	Categorical
location	The account's location, as input by the account user	Text string
Type	Whether the tweet is spam	Categorical

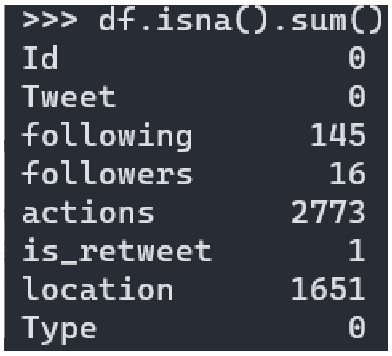
Table 1: The utkML dataset columns used

The dataset will be split into training, validation, and testing sets by an 80-10-10 ratio to ensure models have enough data to train without overfitting. The data was split randomly using a consistent random state to ensure all models were built using the same data. As the utkML dataset contains a relatively balanced set of 51% quality tweets and 49% spam tweets, this random split was appropriate. Further inspection of the training, validation, and test sets shows that the class distribution is largely retained, with quality tweets comprising 52%, 50%, and 51% of the sets, respectively. As a result, where hyper-parameter selection does not dictate the use of class weights, they will not be used in model training.

All models will be trained on the training set and evaluated on the test set. Where hyper-parameter selection is required for non-deep learning (DL) models, the validation set is used.

3.1.1. Imputation

While there are null/missing values in the dataset as noted in Figure 1, the Kaggle page suggests that imputation by using average values or zero/empty strings is acceptable. Furthermore, the removal of data or columns may still be acceptable given the number of samples made available.



```
>>> df.isna().sum()
Id              0
Tweet           0
following     145
followers      16
actions      2773
is_retweet      1
location     1651
Type            0
```

Figure 1: Number of missing values in the dataset

In this report, we decided to impute the following, followers, and actions columns by the mean value of the columns in the training set. This is because zero was found as a valid value, so using zeros may skew the dataset such that the models may be negatively affected because of ambiguities between tweets with actual zero values and imputed zeros. However, we decided to impute the location column by using an empty string, as no empty string was found as a valid location value. This means that using an empty string is appropriate, as models may be able to learn that an empty string indicates the absence of a location value. Finally, given there is only one tweet without an 'is_retweet' value and imputing categoricals without bias is non-trivial, the tweet was removed.

3.1.2. Standardisation

As examining the values for the 'following', 'followers' and 'actions' in the dataset shows massive variations in range as shown in Figure 2, these columns were standardised after imputation. This ensures models are less susceptible to columns with large ranges dominating training.

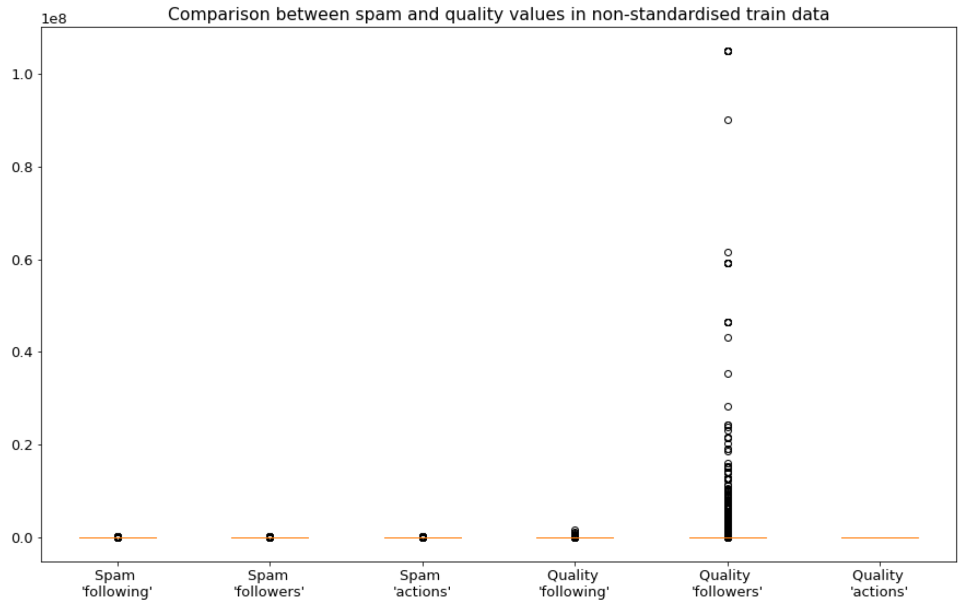


Figure 2: Pre-standardised data ranges show quality tweets may have up to 100M followers, significantly skewing ranges

An examination of the data ranges of 'following', 'followers' and 'actions' in the training and test sets after standardisation as shown in Figure 3 indicates different value ranges in these columns between spam and quality tweets. As a result, these columns are deemed to be useful for spam detection. Note that standardisation has greatly reduced the range of the data while retaining outliers. Furthermore, imputation by mean values may have obscured some user behaviour patterns by assuming users behave similarly.

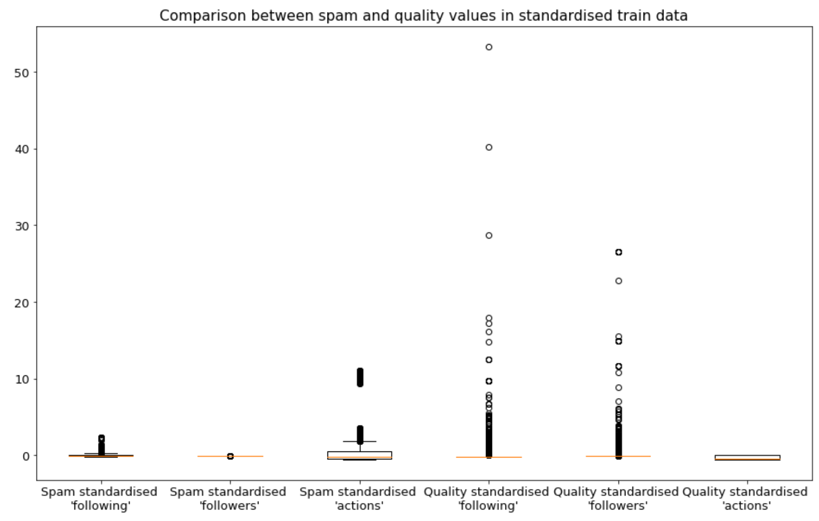


Figure 3: Comparison between spam and quality values in standardised train data

3.1.3. Text Representation

As machine learning methods only accept numerical representations of data, the dataset's tweet and/or location columns must be transformed into a numerical representation. A bag-of-words (BoW), where a document is transformed into a histogram showing how many times a word appears in the document, was used for the non-DL methods, with tokenisation conducted by the scikit-learn BoW implementation. Meanwhile, the DL methods use a simple tokenised representation provided by a Keras Tokenizer layer to assign a numerical representation for each word in the tweet.

It is worth noting that with the same dataset, all models require the same representation to be used on all datasets to pick up words/tokens that it has trained on. Otherwise, performance may be reduced because any changes made to the configuration will affect the values and orders of tokenised sequences or BoWs. As a result, in evaluating each model, the test set was transformed based on the BoW and Tokenizer trained on the training set beforehand.

Note that tokenizers fitted on the training set will remove any unseen words (i.e., items that are not in the tokenizer vocabulary) for the Keras Tokenizer layer and simply ignore them for BoWs. This may a problem where unknown words from tweets with potentially diverse topics are ignored since too few words or characters are recognised from the training data. Consequently, the model may lose more meaningful information and perform worse. As a result, the training data should be diverse and large enough, or each model is trained and only used for certain topics to avoid losing information from new words.

Before tokenisation, all tweets are set to lowercase for consistency, as the dataset may not have enough instances of one word capitalised in different ways for the model to learn based on word capitalisation. Furthermore, tokenisation preceding both transformations will retain all alphanumeric characters.

3.1.3.1. Bag-of-Words Representation

Tweets and/or location strings will be tokenised using scikit-learn's default tokenisation pattern, (?u)\\b\\w\\w+\\b, which finds tokens/words by finding two or more alphanumeric characters, ignoring all punctuation. This is because non-DL models are subject to the curse of dimensionality requiring a significantly greater number of training set samples than the number of dimensions. As a result, only alphanumeric words will be considered to reduce the dimensionality of the transformed dataset.

BoW was selected, as word vectorisation approaches such as Word2Vec, where each word is given a unique vector embedding, greatly emphasise the order in which words appear. This is because such approaches create columns that correspond to each word position in the tweet. However, given that tweets in the corpus may not consistently follow grammatical rules dictating the order of words or even use the same language, the order of words may not be a key feature in spam detection.

Furthermore, whether a tweet is spam is more likely to be determined by the words contained in the tweet than the order in which the words appear. Using word vectorisation, the model may be unable to easily learn to detect spam based on the words in a tweet, focusing on order instead.

Moreover, word vectorisation approaches return a variable-length embedding for each document, depending on the number of words in the document. Given that ML models require a fixed-length input and tweets have a variable number of words, this necessitates padding the word embeddings to a fixed length. Depending on the highest number of words in a tweet in the corpus and the size of the vector embedding, this may significantly increase the number of dimensions. While the non-DL methods perform well with high-dimensional data, they remain limited by the curse of dimensionality. Furthermore, as embeddings are often padded with zeros, this introduces meaningless or noisy data, potentially lowering performance.

Meanwhile, BoW returns a fixed-length output, as it returns a histogram of word occurrences, with each word in the corpus considered. While this may introduce a high number of dimensions, hyper-parameters exist to remove words from the BoW representation based on their frequency in the corpus, reducing the number of dimensions.

Word vector embeddings also introduce dense dimensions into the dataset, forcing the support vector machine (SVM) to compute a hyperplane through samples that may be spread more evenly in the feature space and the naïve Bayes (NB) model to compute more feature posterior probabilities, lowering performance. Meanwhile, BoWs introduce sparse dimensions. This is because all words from all tweets are each represented as a single column in the dataset with all tweets and/or locations converted into a BoW format. As a tweet always contains a fraction of the words found in the corpus, most of these columns will be zero (i.e., the word was not found in the tweet) for each tweet, creating a sparse feature space.

BoWs have the following hyper-parameters. Frequency normalisation using term frequency-inverse document frequency (TF-IDF), commonly used in text classification tasks to ensure that common words do not dominate the dataset in magnitude, may be used. This is achieved by scaling the number of word occurrences in a document (term frequency) with a measure of how many documents contain the word (the inverse document frequency) $idf(t)$, given by $idf(t) = \log\frac{1+n}{1+df(t)} + 1$, where t is a term, n is the number of documents, and $df(t)$ is a function showing how many documents contain a term (document frequency), then normalising this value by a norm (Pedregosa et al., 2011). Given that TF-IDF normalises by L1 or L2 norms in scikit-learn, this can be investigated as a hyper-parameter.

BoWs can consider singular words (unigrams) and ordered pairs of words (bigrams). However, in this report, only unigrams will be considered because of the small training set size relative to the number of dimensions a BoW can potentially generate. For this training set, 27,894 dimensions, corresponding to the number of words found in the training set, could be generated from the tweet column alone. Considering unigrams and bigrams would significantly increase the number of dimensions generated, lowering performance. While using solely pairs of words may improve performance, given the diversity of tweet content potentially causing specific ordered pairs of words to appear in few tweets, this may also lower performance.

BoW maximum frequency is used to remove words that exceed a certain document frequency value, assuming common words provide no meaningful data for the model to learn from for spam detection. Likewise, BoW minimum frequency removes words below a document frequency value, assuming rare words equally provide no meaningful data for spam detection.

These hyper-parameters will be selected for each model separately in Section 4. This is because different models may have different requirements concerning the use of frequency normalisation, maximum/minimum frequencies, and other considerations. Particularly, for the SVM, which uses all columns for the reason noted in Section 4.1, the two text columns in the dataset, the tweet and the user's location, were processed with separate BoWs. This is to allow the model to explicitly learn to detect spam based on both features.

3.1.3.2. Tokenised Representation

For the DL models, tweets are tokenised but still retain special characters, particularly #, @, *, “, ‘, :, \-, \n, %, ,, ., and ;. This is because these characters have specific meanings on Twitter and they are expected to be used differently between spammers and legitimate users. Furthermore, DL models are less susceptible to the curse of dimensionality lowering performance. Therefore, they are kept in the tweets to give more detailed information to the model.

For example, with the mentioned separation process, '#' and 'artificial_intelligence' will be tokenised into two values. But if they are not separated, the string will remain as one – '#artificial_intelligence' – which will then be turned into one integer value and may carry less meaning. And indeed, our experiment shows that an LSTM model trained with special characters separated achieved higher performance by a small margin (see Appendix 1).

After fitting the Tokenizer on the training data, its vocabulary contains 28,078 words, lower than our preset maximum vocabulary size of 50,000. This may not cover a wide range of words found in the test data or in newly emerging tweets and should be noted in interpreting results. Zero padding is applied at the end of shorter tweets to match the greatest tweet length found, as DL models were configured to use fixed-length inputs for improved runtimes.

3.1.4. Text Observations

Regarding the ‘location’ column, inspection of its content shows that both spam and human accounts put unreal locations or unmeaningful symbols/text. Furthermore, the same valid locations may not be recorded as the same string (e.g., United States can be recorded as ‘United States’, ‘us’ or ‘US’). Due to these issues potentially lowering the column’s discriminative power, the NB model in Section 4.2 and the transformer in Section 4.4 will not use this column to prevent lowered performance due to the curse of dimensionality and overfitting, respectively.

Furthermore, note that due to a relatively small training set size relative to previous work noted in Sections 1 and 2, tweets may cover limited topics and languages. As a result, performance may be reduced where test set tweets cover different topics as noted in Section 5 and may not be generalisable to languages other than English.

3.2. Evaluation Protocol

Models will be trained and evaluated on machines with Intel Core i7-equivalent performance[1] with no powerful graphics processing unit for parallelisation due to time and budgetary constraints precluding the acquisition of consistent and powerful computing resources. Models will be evaluated based on the following criteria in this priority:

1. Time taken to evaluate the model on the test data (faster is better)
2. Weighted F1-score of tweets that are spam (higher is better)
3. Time taken to train the model on the training data (faster is better)

These criteria reflect the real-world requirements of a model used to detect spam. First, while a highly accurate model would help reduce spam, legitimate users are likely to be frustrated by an anti-spam system that takes an excessive amount of time to process their tweets. Second, while users’ negative perceptions of spam may mean models that aggressively detect spam are favourable, given the scale of a social media platform such as Twitter, this model may be used to automatically remove spam. As a result, a weighted F1-score is used to balance false negatives and positives, as false positives may frustrate legitimate users and increase the workload of Twitter’s human moderation team in assessing false positives. Third, model training is expected to be conducted infrequently, so a long model training process is reasonable.

4. Methodology

The following methodologies explored were selected based on the results of related work posed earlier in Section 2. These also represent a range of ML models, from “traditional” non-DL models to state-of-the-art DL models. Metrics used to determine appropriate selections included accuracy, suitability to the dataset, and feasibility based on available hardware.

Non-DL models are implemented using scikit-learn, while DL models are implemented using Keras (Pedregosa et al., 2011; Chollet, 2015).

Where hyper-parameter selection was required, a grid search was used to ensure that hyper-parameters maximise model performance by comprehensively searching for a combination that maximises the weighted F1-score computed on the validation set. The weighted F1-score was used to ensure that the selected hyper-parameters promote balanced precision and recall performance across all classes.

4.1. Support Vector Machine

Support vector machines (SVMs) are a supervised, non-deep learning method for classification. SVMs aim to compute a linear hyperplane to separate data points into two classes which maximises the distance between the hyperplane and the farthest data point from each class. From this, the class of a sample can be inferred by determining which side of the hyperplane it is located in.

SVMs were selected, as they are known to perform well with sparse, high-dimensional data. As discussed in Section 3.1.3.1, the conversion of tweets into a data format accepted by SVMs significantly increases the number of parameters and leads to a sparse data structure. SVM’s higher performance with such data is caused by sparse, high-dimensional data causing samples of different classes to be spaced far apart from each other, increasing the likelihood that the classes are linearly separable. This allows SVMs to easily compute a linear hyperplane, allowing for convergence and higher performance in the task. As a result, the SVM will be trained with all columns used.

[1] The SVM was known to be evaluated on an Intel Core i7-1165G7 laptop and the transformer was evaluated on an Intel Core i7-8700.

This implementation extends the work of Benevenuto et al. in 2010 as noted in Section 2 by using a BoW representation of tweets and location strings for richer information. Meanwhile, their SVM used numerical feature aggregations from tweets for classification.

4.1.1. Hyper-parameter Selection

4.1.1.1. Grid Search Values

Due to computational constraints[2] causing a fully exhaustive grid search across both the BoW and SVM hyper-parameters to be impractical, the grid search was first conducted over the BoW hyper-parameters, assuming a linear SVM with $C = 1$ without class weights, the scikit-learn default SVM hyper-parameters (Pedregosa et al., 2011). The selected BoW hyper-parameters were then used to conduct a grid search over the SVM hyper-parameters. Note that because there are two text columns, two sets of BoW hyper-parameters were tuned. The hyper-parameter values used in the grid search and the selected values (in bold) are shown in Table 2. Furthermore, the grid search only considers whether both BoWs use TF-IDF or not, instead of a combination of the two, to reduce hyper-parameter search times.

Hyper-parameter	Values
Tweet and location bag-of-words (BoW) frequency normalisation	None, **term frequency-inverse document frequency (TF-IDF)**
Tweet BoW maximum frequency	0.125, 0.25, **0.5**, 0.75, 1.0
Location BoW maximum frequency	0.125, **0.25**, 0.5, 0.75, 1.0
Tweet and location BoW minimum frequency	0.0001, **0.005**, 0.001, 0.05, 0.01, 1
Tweet and location BoW TF-IDF use of IDF	**Yes**, no
Tweet BoW TF-IDF normalisation norm	L1, **L2**
Location BoW TF-IDF normalisation norm	**L1**, L2
SVM class weights	**None**, balanced weights
SVM slack variable/ C	0.001, 0.01, 0.1, **1**, 10, 100, 1000
SVM kernel	Linear, **radial basis function (RBF)**, polynomial
SVM number of degrees (polynomial kernel only)	3, 4, 5, 6
SVM γ (RBF kernel only)	**'scale'**, 'auto', 1000, 100, 10, 1, 0.1, 0.01, 0.001, 0.0001

Table 2: Hyper-parameter values used for the BoW + SVM grid search and the selected values in bold

These hyper-parameters and values were used for the following reasons. The values selected for maximum and minimum frequencies allow a reasonable range of frequency values to be tested up to 1.0 and 1, where no words are removed, respectively. While the dataset is balanced as noted in Section 3.1, the use of class weights to deal with class imbalance is investigated for completeness.

The slack variable determines how much the SVM's classification operation will tolerate misclassifications (i.e., where classes are not fully separated by the hyperplane), with lower values being more tolerant. The values selected are a reasonable range of orders of magnitude for the slack variable.

The SVM kernel determines the kernel trick used to transform the feature space. Kernel tricks are used to allow the SVM's hyperplane to separate classes non-linearly. Barring the linear kernel, each kernel has a hyper-parameter. The number of degrees determines the number of degrees used for the polynomial kernel trick, with a reasonable number of degrees considered. γ determines how far a sample's influence extends in computing hyperplanes, with greater values increasing overfit risk by causing the SVM to compute class separations that tightly follow the training set sample distribution. A reasonable range of orders of magnitude was considered, as well as scikit-learn's options to compute γ based on the dataset's statistics (Pedregosa et al., 2011).

4.1.1.2. Grid Search Results

Both BoWs yielded better performance on the linear SVM using TF-IDF. This may be caused by tweets and locations containing many common words that do not provide any discriminative features between legitimate and spam tweets. They both also used a minimum document frequency of 0.5%, meaning rare words were deemed to provide little to no ability to discriminate between spam and quality tweets and were dropped from the BoWs.

Notably, the location BoW uses a stricter maximum document frequency of 25%, while the tweet BoW uses a more permissive 50%. This indicates that the common words in users' location strings may be less meaningful than those in tweets. It also used L1-norm normalisation, while the tweet BoW opted to use the L2-norm. This may be caused by the location column having fewer words across

[2] This grid search was originally conducted using a device running Ubuntu 20.04 on the Windows Subsystem for Linux 2 on Windows 11 with an Intel Core i7-1165G7 CPU and 16 GB of RAM.

all data points and the stricter maximum document frequency causing the unnormalized TF-IDF values to be within a similar range. This may necessitate L1-norm normalisation to promote sparsity and improve SVM performance. Meanwhile, with a larger number of words in the tweet corpus and a more permissive maximum document frequency value, the tweet TF-IDF values may have a larger range, meaning that L2-norm normalisation remains appropriate.

The SVM performed best on the validation set using $C = 1$, no class weights, and an RBF kernel with the 'scale' γ setting, computed as $\frac{1}{D \times \sigma^2}$, with D being the number of dimensions and σ^2 being the dataset's variance, returning a γ of 0.222 (Pedregosa et al., 2011). The use of an RBF kernel with a small γ and C value may suggest that the numerical columns in the dataset introduced some non-linearity, necessitating minor adjustments through a kernel trick and the slack variable to allow the SVM to compute a hyperplane.

4.2. Naïve Bayes Classifier

Naïve Bayes (NB) uses Bayes' theorem and assumes that all features are independent. In other words, this classifier assumes that the presence of one feature in a class does not affect the presence of another one. Given our Twitter dataset in a BoW representation has lost word order information, each word is a 'unique' feature with little to no relation with other words, which means NB is an appropriate mechanism for classifying spam in this case. Furthermore, as corroborated in Section 5.1, the probabilistic nature of words being more likely to appear in spam or quality tweets matches how Naïve Bayes works as described below.

Naïve Bayes works by calculating the product of probabilities where some X_i is found given Y, such that X_n is the list of features or words within a tweet and Y is whether the tweet is spam or not. Note that as the denominator in Bayes' theorem is constant across a dataset, as it is the probability of each feature appearing, it can be ignored in training and used for normalisation only if required. The generic formula and its representation for a tweet of 'foo bar baz' is shown below.

$$P(Y|X) = P(X_0|Y) \times P(X_1|Y) \times \ldots \times P(X_n|Y)$$
$$P(spam|foo, bar, baz) = P(foo|spam) \times P(bar|spam) \times P(baz|spam)$$

As shown by the algorithm, the tweets must first be vectorised into individual features using a BoW without TF-IDF. This was used because NB can theoretically compute feature probabilities better with count values (i.e., how many words appear in a tweet, essentially simplifying features to a near-categorical value).

However, as Naïve Bayes treats features independently, certain features were dropped from the dataset, as these issues were observed in both quality and spam tweets:

1. The number of followers and followed accounts were dropped, as quality and spam accounts had some overlap in value ranges.
2. The 'location' of each account was dropped, as it contained inconsistent information such as non-existent locations or rare Unicode characters such as emoji.
3. Naïve Bayes works on frequency distributions. As these columns either may not have a consistent frequency distribution or are real values, the algorithm may become numerically unstable if these columns were used.

Our implementation extends works by Wang (2010), and Gautam and Yadav (2014) by using all words in the tweet in a BoW representation and applying this to spam detection. A multinomial NB model as implemented in scikit-learn was used, as BoWs are essentially a multinomial distribution and this model has been known to work well in text classification tasks (Pedregosa et al., 2011).

4.2.1. Limitations

Naïve Bayes is limited by the assumption that features are independent. As a result, the following limitations when using NB for our dataset apply. First, NB reports better accuracy when there are more samples than features. However, tweets have a large vocabulary including misspelling, introducing more words than expected. Second, spam and quality tweets frequently share common words, which may affect the class posterior probability of a particular entry due to the independence assumption, causing misclassifications.

4.2.2. Hyper-parameter Selection

A grid search was performed for the BoW vectorizer and NB model, aiming to maximise the weighted F1-score the NB model returns with the validation set. The values in Table 3 were investigated.

Parameter	Best	Tested
CountVectorizer()		
max_df	0.5	0.125, 0.25, 0.5, 0.75, 1
min_df	0.0001	0.0001, 0.005, 0.001, 0.05, 0.01, 1

token_pattern	(?u)\b\w\w+\b	(?u)\b\S\S+\b', '(?u)\b\w\w+\b'
MultinomialNB()		
α	1	0, 0.25, 0.5, 0.75, 1
fit_prior	FALSE	TRUE, FALSE

Table 3: Grid search hyper-parameter values and results for NB's BoW

The maximum and minimum frequency values tested mirror the SVM's BoW in Section 4.1.1.1 for consistency. Notably, the NB BoW was tested against a more permissive token pattern that treats a substring of at least two non-whitespace characters as a word to examine its impacts on performance.

The α hyper-parameter for multinomial NB sets whether Laplace smoothing (at α = 1), Lidstone smoothing (at 0 < α < 1), or none (at α = 0) is used to approximate a non-zero feature probability for words that do not exist in the training set (Pedregosa et al., 2011). This is necessary, as a zero feature probability, or $P(x_i|y)$, will cause the entire class posterior probability as noted in the generic algorithm above to be zero, skewing predictions and reducing performance (Jayaswal, 2020). The fit_prior parameter, if true, means the NB model will learn the class prior probability (i.e., $P(y)$), or use a uniform value if false.

While Naïve Bayes works optimally with fewer features than samples, which is possible with stricter maximum and minimum frequencies, the hyper-parameter search returned permissive values, indicating that less frequent words were required to achieve higher performance. This may be caused by the model requiring additional sparsity to better discriminate between spam and quality tweets by greatly reducing class posterior probabilities.

Furthermore, the default token pattern, selected by the grid search, separates components of URLs by removing symbols. Meanwhile, it was expected that keeping URLs whole would have been preferable. This may be due to whole URLs being more likely to be unique across the entire dataset, reducing performance by preventing the model from learning from the general presence of URLs in tweets.

Furthermore, Laplace smoothing, which assumes a normal distribution for non-existent words in the training set, was selected (Jayaswal, 2020). This may mean the normal distribution is a theoretically reasonable assumption for this dataset and model. Moreover, the model may have made accurate inferences solely by using feature probabilities, making class prior probabilities unnecessary.

4.3. Long Short-Term Memory (LSTM)

RNNs, in general, are a DL method that effectively learn from sequences such as tweets, as their mechanism of processing sequence elements in order using 'cells' allows them to propagate information from the previous time step to the next, enabling the model to learn based on the order of elements in sequences. LSTMs extend the concept of RNNs by explicitly maintaining a cell state across all cells and using gates to choose what information to add or remove for the next cell. This allows them to learn long-term dependencies, as information from the beginning of the sequence can be kept up to the end of the sequence without the risk of the model 'forgetting' information, provided they are useful. Accordingly, LSTMs can consider and produce predictions from an entire sequence. This LSTM implementation extends the LSTMs shown by Sharariar et al. (2019) by investigating the addition of numerical data to an LSTM.

4.3.1. Model Design

With the provided dataset, there are three possible inputs for the model: input A containing the tokenised tweet sequences; input B containing the number of followed accounts, followers, and actions, as well as whether the tweet is a retweet; and input C containing the tokenised location sequences.

In total, there are three models utilising three input combinations that were experimented as shown in Table 4.

Model	Input(s) utilised	Purpose
Model 1	Input A	Check performance with a single tweet input
Model 2	Inputs A and B	Compare the performance with Model 1 to examine input B's effectiveness
Model 3	All three inputs A, B and C	Compare the performance with Model 2 to examine input C's effectiveness

Table 4: LSTM models experimented with

To learn embeddings that aid in spam detection from the tokenised sequences, input A and input C are followed by LSTM layers. The number of units selected in an LSTM layer is a value that is higher but close to the input length (or the sequence length). By doing this, the LSTM layer is expected to have a suitable number of dimensions to cope with the complexity of the input (the complexity is assumed to highly correlate with the input length).

Input B is connected to a Flatten layer and then a Dense layer to learn from the numerical input. The layer is comparatively simple, only containing four neurons, to prevent model overfitting.

A Concatenate layer is used to combine many single-dimensional inputs (i.e., outputs of Dense layers) into one single-dimensional output. After this layer, a Dense layer is added for feature aggregation. Dropout layers are inserted after the final LSTM and final Dense layers to avoid overfitting by introducing random disconnections to encourage neurons to learn more. Finally, as spam detection is a binary classification problem, the final Dense layer uses a sigmoid activation function to provide the "probability" of a tweet being spam or quality.

To decide the number of LSTM layers in a stack, another experiment comparing Model 2's performance with single and double LSTM layers indicates that the model with two layers is slightly better and converges to the maximum validation accuracy and minimum validation loss faster by twofold as shown in Appendix 2. This can be explained by the second LSTM layer providing a more complex feature representation of the sequence. Therefore, finer details of the sequences may be discovered, which accelerates the learning curve. However, double LSTM layers may increase inference times compared to one layer due to the increased number of computations needed. Deeper LSTM models are avoided, as gradients are likely to be harder to propagate, complicating training.

Overall, the test performance of all three models were high. However, Model 2 achieved the highest test performance as shown in Table 5 and its model architecture can be seen in Figure 4: LSTM Model 2 architecture. Diagrams of the models 1 and 3's architectures can be found in Appendix 3.

Metrics	Model 1	Model 2	Model 3
Training time (s)	1754 s	2212 s	1645 s
Inference time (s)	3.0 s	4.39 s	3.5 s
No. of epochs used	62	91	57
Testing accuracy	.916	.97	.952
Testing weighted f1-score	.92	.97	.95

Table 5: LSTM models' Results

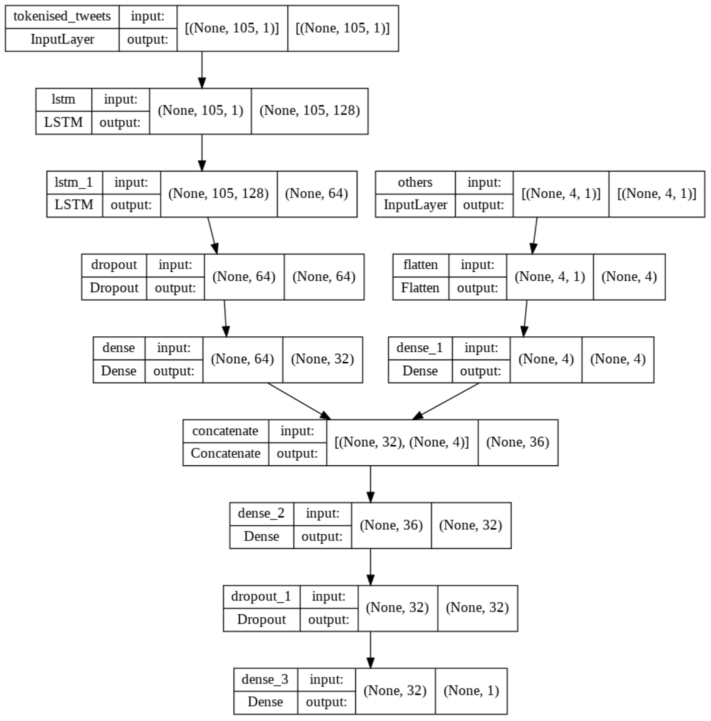


Figure 4: LSTM Model 2 architecture

The results confirm our speculation earlier that the 'following', 'followers', 'actions', and 'is_retweet' columns provide valuable information to the learning process. On the other hand, even though Model 3 performed marginally worse than Model 2 did, Model

3 consumed 37% fewer epochs and finished its training faster. It is conjectured that 'location' data provides meaningful data at the beginning, accelerating convergence, but may have also created noise and lowered performance. Interestingly, training times per epoch were similar among the three models despite the difference in model complexities, while inference times were inconsistent with the model complexities. This is likely because of inconsistently available CPU capacity.

4.3.2. Model Training

Two callbacks were utilised in training. A model checkpoint was used to save only the model with the lowest validation set loss. Early stopping was used to stop training when validation loss is not decreased within 25 epochs. Given the model is trained for a maximum of 300 epochs, this high patience balances allowing the model to train to convergence while curbing overfit. Overfit risk is also mitigated by using the weights of the model with the lowest validation loss as saved by the checkpoint instead of the weights saved when early stopping ends training.

Given the model is trained from scratch due to difficulties in finding an appropriate pre-trained model, the Adam optimiser with the Keras default learning rate of 10^{-3} was used to ensure gradient descent can find the global minima. A batch size of 128 was used to ensure the model learns from a representative collection of samples.

4.4. Transformer

Transformers are a DL model extending LSTMs' ability to learn from sequences by learning long-range relationships between words using word positions in embeddings and weights for each word pair. As these pairwise relationships are learnt in all directions, unlike LSTMs that learn from one word to the next in at most two directions, transformers have exhibited state-of-the-art performance in natural language processing. This pairwise learning also means that transformers are suited to sequences that may inconsistently follow grammatical rules such as tweets.

This is accomplished by an attention-based module known as multi-head self-attention with positional tokens. This implements self-attention, which uses learnt key, query, and value weight matrices to compute the weights of each word pair independently while still considering each word's position in the sequence.

Quoting from the original paper, "multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions" (Vaswani et al., 2017). In other words, transformers can learn multiple weight matrices and combine their results using another learnt weight matrix to learn a better data representation.

In addition, the independent nature of these matrix multiplications enables parallelization. Given large matrix multiplications and parallelisation are easily done by modern GPUs, this also provides transformers with a runtime advantage over LSTMs, provided such GPUs are in use. While transformers can be used for supervised or unsupervised learning, as spam detection is a classification task, our transformer will use supervised learning.

This transformer implementation aims to investigate whether a simpler transformer can yield similar or improved performance to the complex implementations proposed before (Vaswani et al., 2017; Liu et al., 2021).

4.4.1. Model Design, Hyper-parameters, and Training

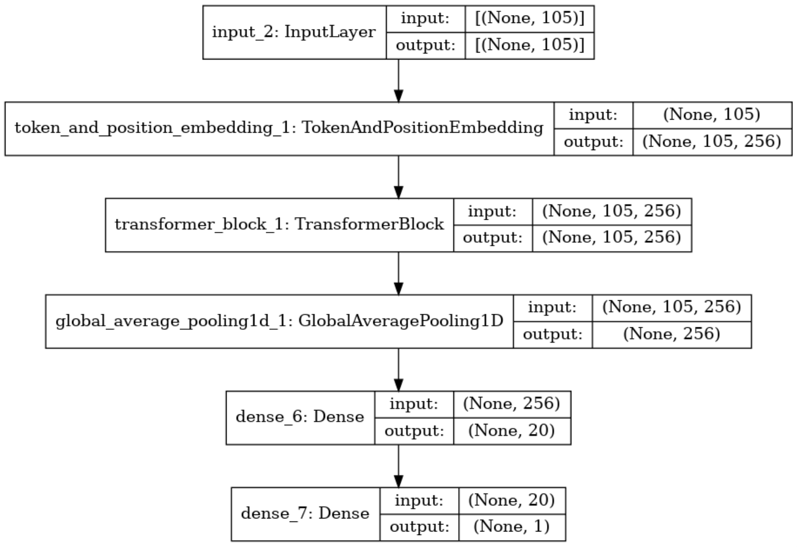


Figure 5: Transformer model architecture

As the model architecture in Figure 5 shows, the transformer will only be trained on the tweet. This is because transformers' reliance on learning three weight matrices and the use of element-wise Dense layers significantly increases the number of parameters learnt. As a result, using other columns may worsen overfitting.

The input will be tokenized with each word's position within a 256-dimension embedding. This is only a quarter of the embedding size that was used in the paper introducing transformers but is more suited to the smaller dataset size and limited computing resources available for training (Vaswani et al., 2017). These limited computing resources (i.e., GPU absence) also precluded hyper-parameter grid searching, meaning hyper-parameters were manually tuned to maximise validation set accuracy.

Transformers have the following hyper-parameters. The number of attention heads denotes the number of key-query-value weight matrix triplets learnt by the transformer. The hidden layer size controls the output size of the initial feed-forward network that processes the weighted output from the attention heads before another feed-forward network constructs a sequence of the same length as the original input sequence.

After manual tuning, the hyper-parameters that maximised validation accuracy are 8 attention heads and a 32-dimensional hidden layer. Presumably, several attention heads were used to ensure the transformer can learn multiple relationships in tweets. Furthermore, the relatively small hidden layer size may have aided in feature aggregation to improve performance and curb overfitting.

The same batch size and optimiser were used as the LSTM for the same reasons. However, due to transformers being known to readily overfit, the epoch count was greatly reduced to five.

5. Result Analysis and Evaluation

5.1. Overall Performance

	Weighted precision	**Weighted recall**	**Weighted F1-score**	**Training time on 8,968 samples**	**Inference time on 1,500 test set samples**
Non-deep learning methods					
BoW + SVM	0.96	0.96	0.96	1.839 s	0.214 s
BoW + Naïve Bayes	0.95	0.95	0.95	0.022 s	0.000 s
Deep learning methods					
LSTM	0.97	0.97	0.97	2,212 s	4.39 s
Transformer	0.96	0.96	0.96	498 s	5.630 s

Table 6: Model result/timing comparisons. Highlighted model has the best overall performance

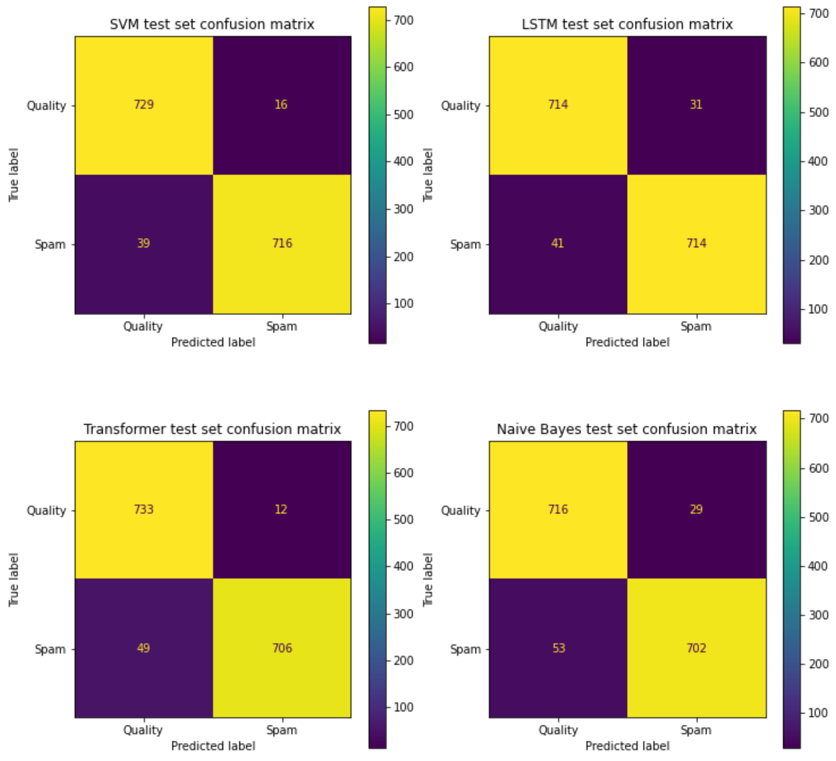


Figure 6: Confusion matrices indicate more false negatives in these models

Based on the performance metrics and confusion matrices for all models as shown in Table 6 and Figure 6, respectively, the LSTM has the best raw performance. This is because LSTMs allow tweets to be processed as a sequence of words, meaning the model learns the relationships between successive words in the tweet. Given the tweets in the dataset, once tokenised for the LSTM, only have a maximum length of 105 and the second LSTM only processes a sequence of 128, the LSTM layers' cell states are still able to retain information from earlier words in processing later words.

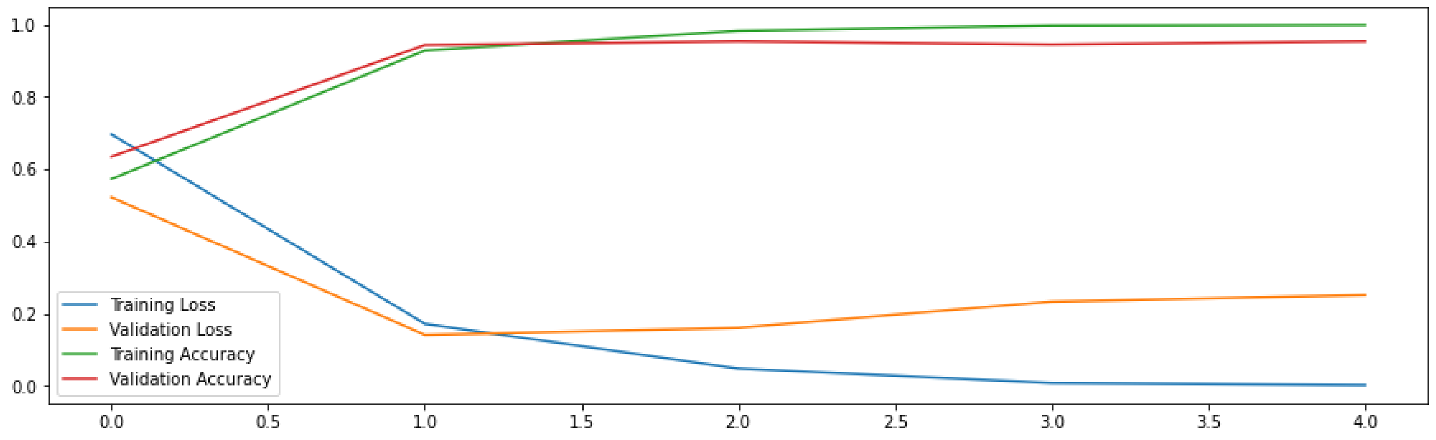


Figure 7: The transformer's training graph shows rapid overfitting

While deep learning performance is subject to variations due to random initialisations, the transformer may have performed marginally worse than the LSTM due to transformers quickly overfitting to the training set, as shown in the training loss and accuracy graph in Figure 7 above. This may be caused by transformers learning the relationships between all word pairs in a tweet. Given the low number of words and samples for a transformer, very few relationships were learnt. As a result, the model's query, key, and value weights can be trivially learnt from a few samples to capture these relationships, compared to the English-French dataset containing 36 million 25,000-token sentences transformers exhibited state-of-the-art performance in (Vaswani et al., 2017). This is exacerbated by transformers relying on element-wise Dense layers after multi-head self-attention is applied, as well as the use of eight heads. This massively increased the number of parameters required, with this model training about 9.6 million parameters.

Notably, the BoW + SVM and NB models performed comparably well to all deep learning models, with the LSTM and transformer potentially being marginally better due to a favourable random initialisation, despite the BoW representation destroying information regarding the order of words and SVMs and NB models being more computationally simple to compute.

Histogram intersection of word sums between…	Value for SVM BoW (only 276 words kept)	Value for NB BoW (all 27,894 words kept)
Words in spam and quality tweets in the training set	0.587	0.447
Words in spam and quality tweets in the test set	0.507	0.449
Words in spam tweets in the training and test sets	0.729	0.649
Words in quality tweets in the training and test sets	0.776	0.649

Table 7: Histogram intersections of frequencies between spam and quality words show dissimilarities. The training and test sets exhibit similar frequencies in spam and quality words. Values are from [0..1] where 0 means histograms are dissimilar and 1 means they are identical

This may be caused by the training and test sets containing quality tweets where certain words rarely appear in spam tweets and vice versa as shown by the histogram intersection values in Table 7. Furthermore, the training and test sets share relatively similar word frequencies for both spam and quality words. This may mean both models can simply detect spam tweets based on the presence or absence of certain words regardless of their order in the tweet, improving performance. This ability to trivially identify spam tweets in this dataset by the presence or absence of certain words may mean that the DL models' capacity to learn complex relationships in sequences is unnecessary, explaining its similar performance.

Notably, the hyper-parameter selection process for NB appears to be counterintuitive, with more words kept improving performance, despite it significantly increasing the number of dimensions and marginally lowering performance due to the curse of dimensionality. This may be due to the BoW now being able to store words that appear significantly more frequently, if not only occurring exclusively, in either spam or quality tweets. This is proven by the histogram intersection for spam and quality words kept by the BoW for the NB model being more dissimilar than the BoW for the SVM, as shown in Table 7.

This introduces some sparsity into NB's dot product calculation of the joint log-likelihood of a tweet being spam or not, improving performance. However, this is then offset by the high number of dimensions requiring a high number of training set samples for numerical stability in cases where probability vectors are dense and to prevent overfitting. Furthermore, using all words caused the word frequency distributions between the training and test sets to differ more than when fewer words were kept, lowering performance on the test set due to potential overfitting.

Furthermore, the use of Laplace smoothing may have lowered performance by falsely assuming the feature probability of words that do not exist. This also somewhat reduces the sparsity of the feature probabilities, offsetting the sparsity introduced by more words being kept in the BoW.

As a result, SVMs slightly outperformed the NB, as the SVM simply ignores words that do not exist in the training set BoW in inference. This means that the sparsity of the dimensions is retained, allowing the SVM to easily compute an effective hyperplane with few words kept in the BoW to curb the curse of dimensionality.

5.2. Comparisons with Related Work

Our models generally outperform the models described in Section 2. This is because our non-DL models encoded tweets into a BoW, providing a richer and less order-dependent representation than numerical features extracted from tweets or Word2Vec/GloVe representations. Furthermore, tweets are shorter in length, improving performance in non-DL models by curbing the curse of dimensionality. Moreover, some of the models explained in Section 2 were trained on small datasets, potentially lowering performance due to overfitting.

Particularly, our transformer implementation was simpler than the model presented by Liu et al. (2021) and used a larger training set, meaning our higher test set performance may be caused by reduced overfitting. Furthermore, our transformer pre-processing may have provided the model with a more useful embedding for tweets, which may use words in ways that are not properly captured by pre-trained dictionaries.

Both recurrent neural networks described in Section 2, the RCNN and LSTM, performed similarly to our LSTM (Lai et al., 2015; Sharariar et al., 2019). This indicates that our pre-processing and implementation have successfully captured a helpful representation for spam detection. Furthermore, the NB model described by Wang (2010) performs similarly to our NB model, though the described model uses numerical predictors. This may be due to a higher number of samples, with about 500,000 tweets used. Moreover, the RF performed similarly to our models, which may be caused by a much larger dataset of the most recent 100 tweets from 1,000 users, amounting to feature extraction from 1 million tweets (McCord & Chuah, 2011). Alternatively, this may be due to RFs' ability to learn diverse patterns in data for spam detection by using multiple decision trees for classification.

Notably, the RMDL detailed by Wael (2020) performs better than our models. This is because an RMDL is an ensemble of deep learning models, allowing predictions to be based on a diverse range of learnt weights. However, this likely comes at the cost of greater computational requirements for training and inference, given the size and complexity of the ensemble. Furthermore, this complexity may cause overfitting given our dataset's smaller size, lowering performance in this case.

5.3. Discussion of Failure Cases

Based on the confusion matrices in Figure 6, all models have a higher rate of false negatives (i.e., spam tweets labelled as quality). This may be due to a minor class imbalance, where the overall dataset contains 51% quality tweets. After splitting the dataset, this is largely retained with minor variations, with the training set containing 52% quality tweets and the test set containing 50% quality tweets.

For the BoW + SVM model, the higher false negative rate may be because while the tweets can intuitively be classified as spam, as shown in Table 8, most of the words in the tweets were not present in the training set tweets due to a small training set. Some of the words were also filtered by the minimum and maximum document frequency hyper-parameters. This may be caused by the spam tweets covering specific themes not covered in the training set tweets or by spelling variations causing the words to not be found in the learnt BoW representation. This can also be seen in the users' location strings either having most words filtered out of the BoW or having none in the BoW at all.

Index	Spam tweet	Tweet BoW		Location	Location BoW	
		Count	Filtered		Count	Filtered
4	RT iswdopn: Marr2rrCarr FoxNews KatTimpf CNN It's getting scarier by the minute isn't it	5	4	home is where the heart is ❤	1	4
10	Quebec As Folk #MakeTVShowsCanadian @midnight	1	3	Near Worthing, UK	1	2
82	#IAmNotThePresidentBecause even @BarackObama isn't good enough	2	4	Kent, WA	0	2

Table 8: Examples of false negative spam tweets from the test set show that few of their words were in the learnt BoW representation

Index	Spam tweet	Tweet BoW		Location	Location BoW	
		Count	Filtered		Count	Filtered
14	I'd rather regret the things that I have done than the things that I have not done." Lucille Ball *Please RT* #Quotes	9	2	United States of America	1	0
257	$LUV - Creative Planning Has $5,572,000 Stake in Southwest Airlines Co #LUV http://goo.gl/fb/HXAOXL	5	3		0	0
346	Webby http://algo.fyi/1dfa5e #politics #trending	2	2		0	0

Table 9: Examples of false positive quality tweets from the test set

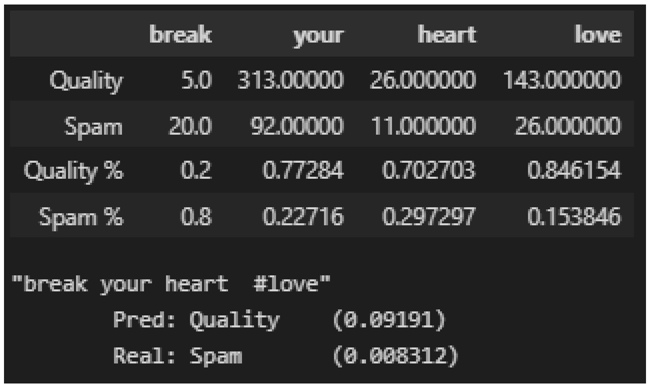
Notably, false positives as shown in Table 9, have more words found in the tweet BoW, but are still misclassified. Further analysis of the words found in the tweet BoW shows that this may be caused by the tweet containing a word that overwhelmingly appears only in spam tweets in the training set such as the examples in Table 10.

Example word	Total TF-IDF value in training spam tweets	Total TF-IDF value in training quality tweets	Total TF-IDF value in test spam tweets	Total TF-IDF value in test quality tweets
rt	85.442	17.931	11.526	3.366
co	913.962	14.623	169.349	3.418
politics	59.548	2.699	6.646	1.117

Table 10: Example of words in the false positive quality tweets that have a higher frequency in spam tweets

As BoW destroys the order of words, the SVM is only able to discriminate between spam and quality tweets based on the existence of certain words, without considering context words around a given word. Particularly, as tokenisation splits URLs such as "https://t.co/" (Twitter's URL shortening service), this may be the cause of the word "co" being frequently found in spam tweets. While BoWs can be configured to consider order using n-grams, this remains an inferior option compared to deep learning methods, as using n-grams significantly increases the number of dimensions used.

This is also found in the NB model, where failed predictions resulted from tweets that utilised words more commonly found within the incorrect class. This can also occur when one or two words have a significantly higher frequency in one class than the other.



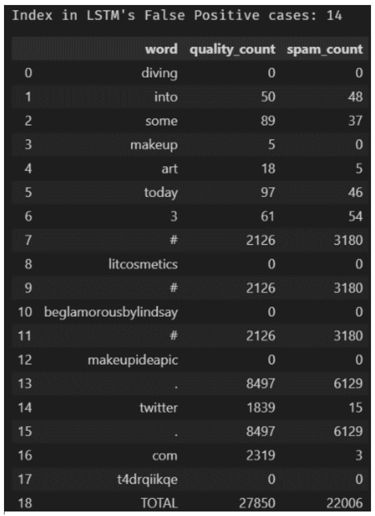
	break	your	heart	love
Quality	5.0	313.00000	26.000000	143.000000
Spam	20.0	92.00000	11.000000	26.000000
Quality %	0.2	0.77284	0.702703	0.846154
Spam %	0.8	0.22716	0.297297	0.153846

```
"break your heart  #love"
        Pred: Quality    (0.09191)
        Real: Spam       (0.008312)
```

Figure 8: NB models report false negatives when a word overwhelmingly appears in quality tweets

The example tweet in Figure 8, 'break your heart #love', contains words that are more commonly found in tweets flagged as quality. Furthermore, given the few words in the tweet, the probability prediction is dominated by these words' higher probability of being found in a quality tweet.

Notably, the LSTM and transformer models also share the same pattern, as most of their false positives have words that are more frequently found in spam than in quality tweets (see LSTM samples in Appendices 4 and 5). Additionally, in the LSTM, some words or characters' appearance in the tweet may have a larger impact than others due to higher frequencies. For example, Figure 9 shows most false positives have words that were more popular in quality tweets, but they were overshadowed by the hashtags ('#'), which were found more in spam tweets, appearing three times. Furthermore, as discussed in Section 3.1.3.2, words that do not exist in the training set are removed completely in the validation and test data without any replacement value. This likely corrupted the meaning of the tweet and removed the model's ability to acknowledge the removed words after those hashtags and learn from a meaningful sequence, causing misclassifications.



```
Index in LSTM's False Positive cases: 14
```

	word	quality_count	spam_count
0	diving	0	0
1	into	50	48
2	some	89	37
3	makeup	5	0
4	art	18	5
5	today	97	46
6	3	61	54
7	#	2126	3180
8	litcosmetics	0	0
9	#	2126	3180
10	beglamorousbylindsay	0	0
11	#	2126	3180
12	makeupideapic	0	0
13	.	8497	6129
14	twitter	1839	15
15	.	8497	6129
16	com	2319	3
17	t4drqiikqe	0	0
18	TOTAL	27850	22006

Figure 9: An example of characters (i.e., hashtag '#') that have greater weights

False Negative Tweet	Word	Frequency of Word in Training Spam Tweets	Frequency of Word in Training Quality Tweets
How Can You Still Call That A Skirt??	How	95	155
	Can	88	222
	You	286	912
	Still	33	60
	Call	29	22
	That	231	345
	A	660	1029
	Skirt	None	2

Table 10: Transformer false negative and word frequency table

An example from the transformer model shown in Table 10 shows the transformer's failed predictions follow the same pattern as previous methods. The table shows that words from the false negative tweet are found more in quality tweets in the training set compared to the training spam tweets. This dataset imbalance in word frequencies affects the model's ability to understand the words in the sequence, as most of its knowledge of the words in the tweet would be retrieved from studying the training set. Given the training set shows these words are more commonly found in quality tweets, the model misclassified this tweet as a quality tweet.

To reiterate, these DL models may have misclassified tweets in the same manner as the non-DL models, as the dataset's tweets are easily separated into quality and spam tweets simply based on the presence or absence of certain words. As a result, these models simply learnt to classify based on this, potentially due to overfitting given a small training set relative to the models' complexity.

5.4. Training Times

The NB model trains the fastest, as NB is trained by computing $P(x_i|y)$, the probability of a feature value occurring given the data point belongs to the class y (Pedregosa et al., 2011). Given this is trivially computed using the count of a value occurring and a log-likelihood, training times are very low. Meanwhile, SVMs must compute a hyperplane to separate classes by maximising the margin between the hyperplane and each class' support vector. This necessitates multiple attempts to find the support vectors that allow this margin to be maximised, exponentially slowing down computation with many training set samples, as there are more possible support vectors to test and a higher risk of classes not being linearly separable complicating computation.

However, these computations are simple compared to the deep learning methods. The LSTM trained for the longest, as LSTMs include more complex computations, with each sequence effectively being processed by four Dense layers, which are matrix multiplications, coupled with several elementwise operations. Furthermore, unlike most DL methods, such as convolutional neural networks, which allow for parallelisation due to most computations being independent of each other, LSTMs operate over sequences in order, precluding parallelisation and lowering training speeds. Moreover, deep learning methods are trained through an iterative process, training the model using a batch of training set data before using gradient descent to optimise weights. In this LSTM's case, this is repeated until the validation loss does not increase. The use of two LSTM layers increases training times further by increasing the number of operations needed.

While transformers have more matrix multiplications in the self-attention layer, combined with an elementwise Dense layer increasing the number of operations further, training times are lower. In this case, this is because the transformer was only trained for a few epochs to prevent overfitting. Further examination shows that the higher number of operations required in transformers significantly increased training times per epoch. With larger datasets, sequences, and sufficient parallel computing resources (i.e., a large GPU, such as an Nvidia P100), lower training times would be attributed to self-attention being a series of matrix multiplications for each element that can be done independently of each other, allowing for parallelisation.

5.5. Inference Times

The NB model infers the fastest, as NB infers using the equation $\hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i|y)$, where $P(y)$ denotes the probability of the class y occurring (i.e., spam/quality) and n denotes the number of features that exist in the dataset (Pedregosa et al., 2011). As this is essentially a dot-product operation between vectors, this can be computed quickly. Meanwhile, as SVMs infer by determining each sample's location relative to the hyperplane, SVMs infer more slowly.

However, both DL models infer significantly more slowly than the non-DL models described above. This is because LSTMs and transformers infer by passing the data into the layer with the learnt weights. Given the complexity of these layers as noted in Section 5.4, inference times are naturally high. The transformer inferred slower, as they have more matrix multiplications to compute, increasing their inference times without parallelisation.

6. Conclusions and Future Works

Generally, all models performed well in spam detection. This means that automated Twitter spam detection as proposed in Section 1 may be viable using these ML models, reducing reliance on manual moderation. However, language is an ever-evolving paradigm where various subcultures offer different meanings for words, different spellings, and slang. Considering this, with the introduction of new words or slang and its use on Twitter, the word representations underpinning these networks and by extension, the networks themselves may require retraining with each new word. Particularly with DL models that are time-consuming to train, this retraining requirement may limit its utility.

Furthermore, as false positives and negatives were detected, solely moderating spam using this system may affect the platform's operational value. This is because false positives may frustrate users and necessitate manual moderation to reverse the false removal of legitimate tweets. Furthermore, an excess of false negatives defeats the purpose of implementing automated spam detection and may further motivate spammers to devise methods to circumvent the automated system. As such, caution must be taken when using ML models for automated spam detection to prevent negatively affecting the user experience. This may mean retaining a human moderation team to assess user-reported cases of false positives or negatives.

Based on the evaluation protocol posed in Section 3.2, the Naïve Bayes classifier performed the best. This is because while its weighted F1-score is marginally lower than all models, its high training and inference speeds allow the classifier to be retrained to accommodate new words and scale with high tweet volumes, providing a balance between performance and computational requirements.

Future works should consider the following. First, these models should be trained and evaluated on consistent computing resources that include GPUs for parallelisation. While training and inference times may relatively remain the same when models are compared, even with consistent resources, this provides greater certainty in the results. Furthermore, this may greatly improve the runtimes of DL models, particularly transformers due to the independent nature of their operations as Section 5.4 describes, which may make DL models more viable compared to the NB model.

Second, to overcome the issue noted in Section 5.3 of test set tweets containing words unseen in the training set and a higher number of false negatives caused by potential class imbalance, a larger, balanced dataset should be used for training these models. This also enables investigations into whether models can generalise to diverse tweet topics, whether using bi-/trigrams can improve performance in non-DL models using BoW representations, and whether user-related features may improve performance with larger datasets.

Third, this can also be used in conjunction with improved pre-processing that considers misspellings or combinations of words without spaces to allow the model to learn from tweets with these characteristics instead of ignoring them, improving performance.

Fourth, as the DL models' performance may have been reduced by the Tokenizer removing words from test set tweets that do not exist in train data, a tokeniser implementation that replaces unseen words with a representative value instead of removing them should be investigated.

Finally, assuming a smaller dataset remains in use, the transformer could be modified to reduce overfitting by increasing the model's dropout or by reducing the number of heads and/or hidden layer size.

7. Appendices

Statement of contributions:

Name	Contribution	Percentage	Signature
Jason Rumengan	Implementation and reporting of BoW + SVM, evaluation reporting, report editing and proofreading	25%	[illegible]
Thomas Fabian	Naïve Bayes Implementation/Reporting, Introduction, Conclusion, Report Editing/Formatting	25%	[illegible]
Rodo Nguyen	LSTM implementation, data pre-processing, data discussion, future works, report formatting, LSTM failed cases	25%	[illegible] Made possible by MS Paint
Gilbertus Priastian	Transformers implementation and reporting, Related Work, References, Fail Cases, Report Editing	25%	[illegible]

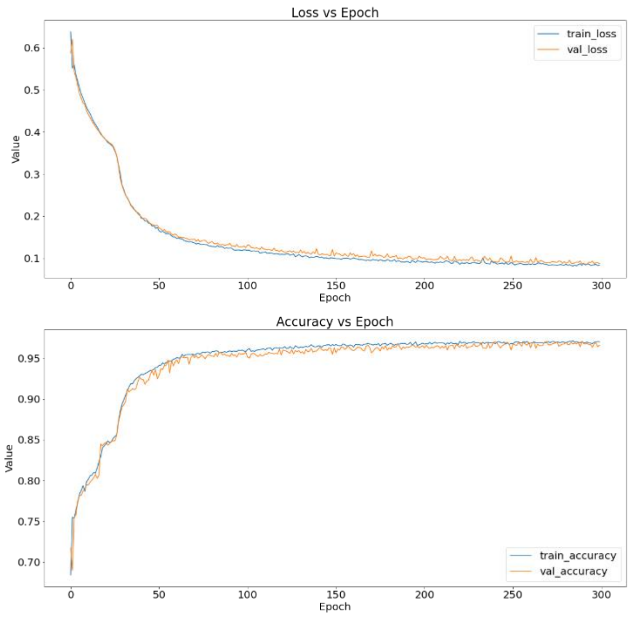
1. Comparison between model performances with special character separation and no special character separation.

	Test accuracy	F1-score (weighted average)
Train with no special character separation	0.96	0.96
Train with special character separation	0.97	0.97

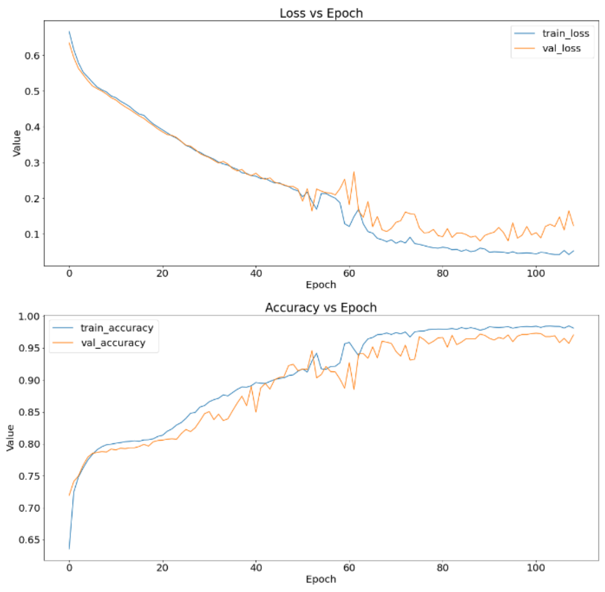
Note: The models above are Model 2 design from section X.

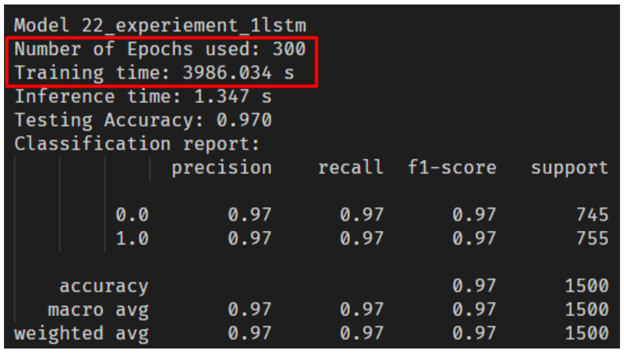
2. Comparison between using single and double LSTM layers

Using single LSTM

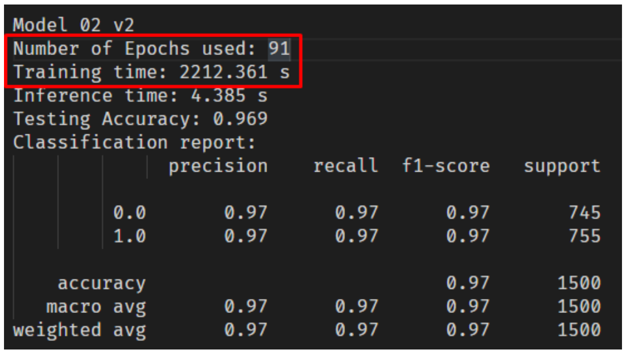


Using double LSTM





```
Model 22_experiement_1lstm
Number of Epochs used: 300
Training time: 3986.034 s
Inference time: 1.347 s
Testing Accuracy: 0.970
Classification report:
              precision    recall  f1-score   support

         0.0       0.97      0.97      0.97       745
         1.0       0.97      0.97      0.97       755

    accuracy                           0.97      1500
   macro avg       0.97      0.97      0.97      1500
weighted avg       0.97      0.97      0.97      1500
```

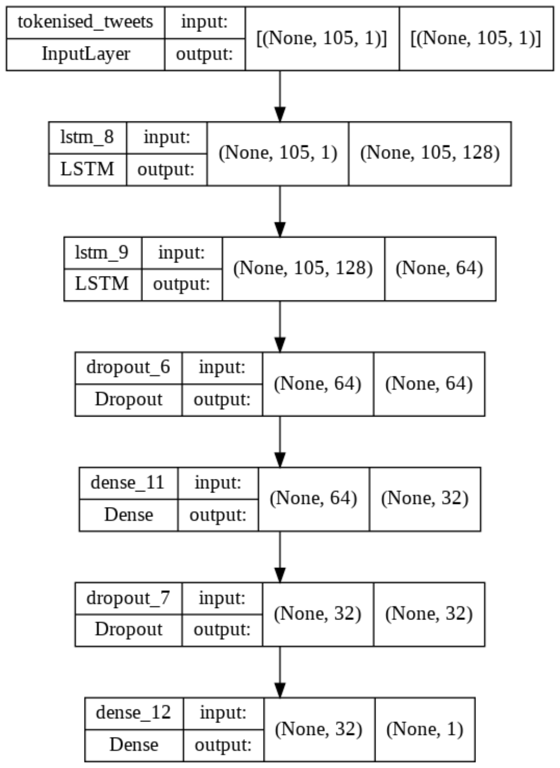


```
Model 02 v2
Number of Epochs used: 91
Training time: 2212.361 s
Inference time: 4.385 s
Testing Accuracy: 0.969
Classification report:
              precision    recall  f1-score   support

         0.0       0.97      0.97      0.97       745
         1.0       0.97      0.97      0.97       755

    accuracy                           0.97      1500
   macro avg       0.97      0.97      0.97      1500
weighted avg       0.97      0.97      0.97      1500
```

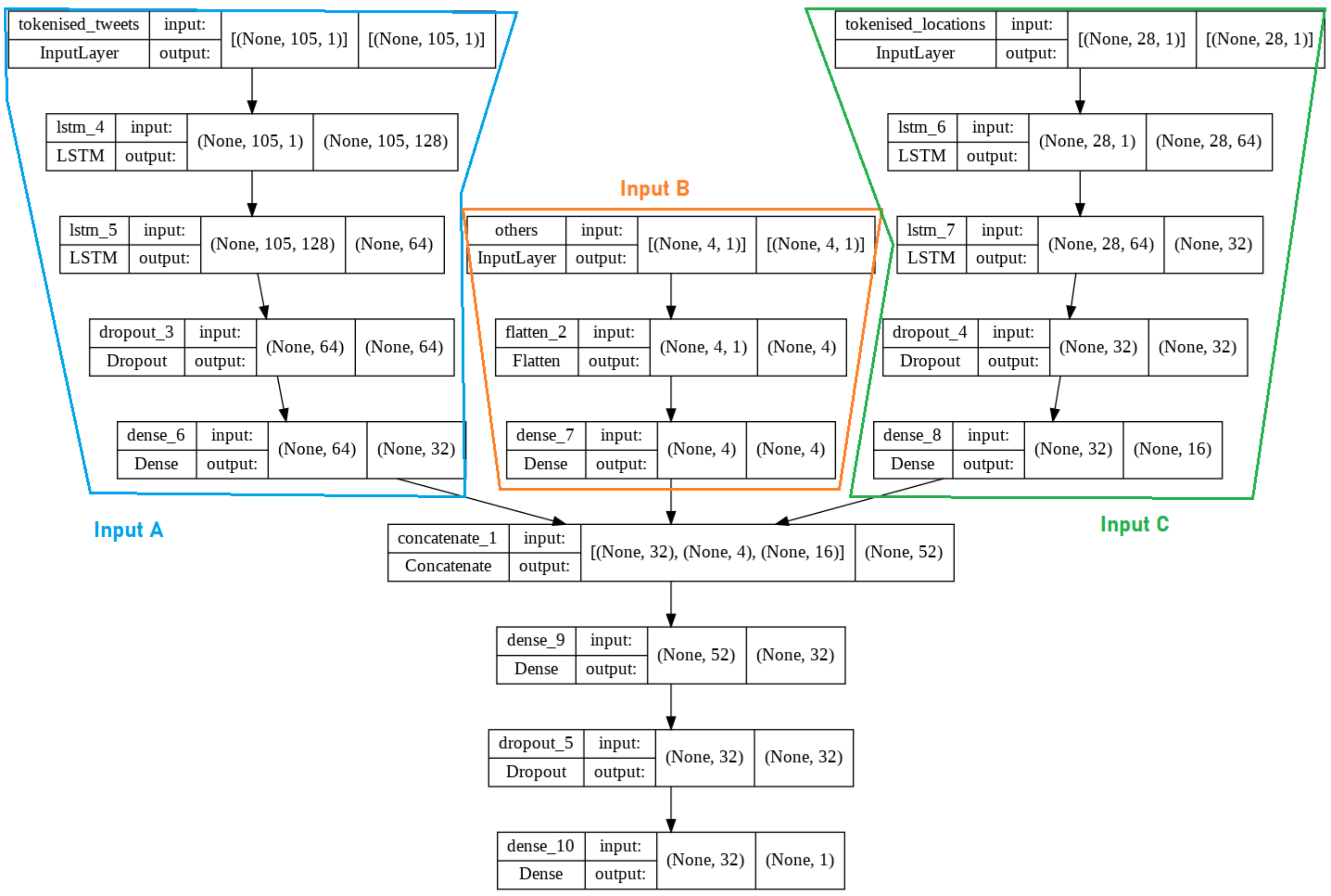
Note: The models above are Model 2 design from section X.

3. Models 1 and 2’s architectures

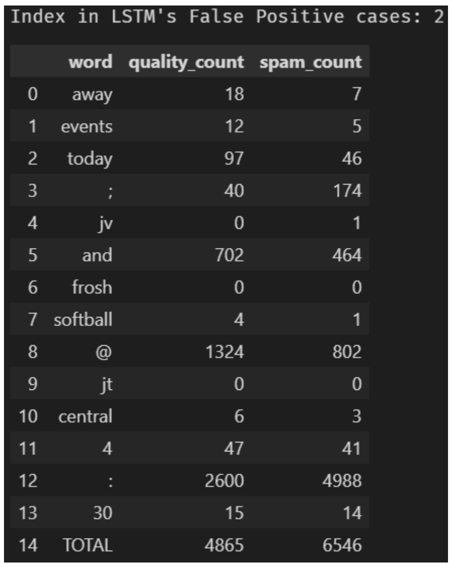
Model 1



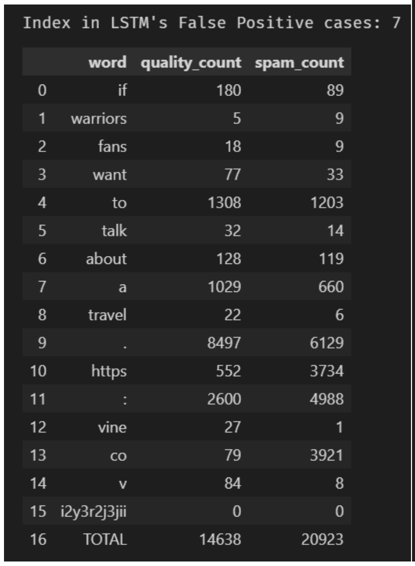
Model 3


tokenised_tweets InputLayer input: output: [(None, 105, 1)] [(None, 105, 1)]
lstm_4 LSTM input: output: (None, 105, 1) (None, 105, 128)
lstm_5 LSTM input: output: (None, 105, 128) (None, 64)
dropout_3 Dropout input: output: (None, 64) (None, 64)
dense_6 Dense input: output: (None, 64) (None, 32)
Input A
Input B
others InputLayer input: output: [(None, 4, 1)] [(None, 4, 1)]
flatten_2 Flatten input: output: (None, 4, 1) (None, 4)
dense_7 Dense input: output: (None, 4) (None, 4)
tokenised_locations InputLayer input: output: [(None, 28, 1)] [(None, 28, 1)]
lstm_6 LSTM input: output: (None, 28, 1) (None, 28, 64)
lstm_7 LSTM input: output: (None, 28, 64) (None, 32)
dropout_4 Dropout input: output: (None, 32) (None, 32)
dense_8 Dense input: output: (None, 32) (None, 16)
Input C
concatenate_1 Concatenate input: output: [(None, 32), (None, 4), (None, 16)] (None, 52)
dense_9 Dense input: output: (None, 52) (None, 32)
dropout_5 Dropout input: output: (None, 32) (None, 32)
dense_10 Dense input: output: (None, 32) (None, 1)

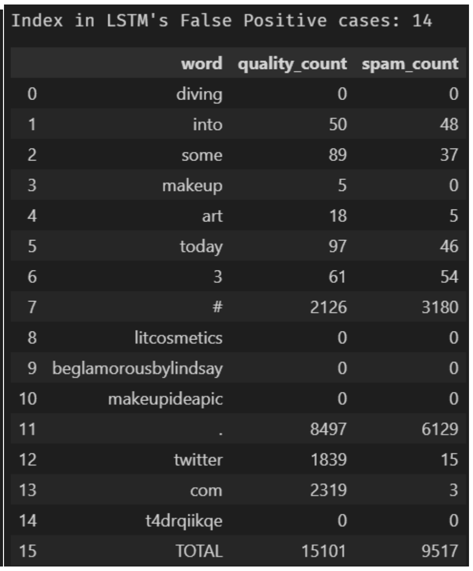
4. Samples of LSTM's false positive cases


Index in LSTM's False Positive cases: 2

	word	quality_count	spam_count
0	away	18	7
1	events	12	5
2	today	97	46
3	;	40	174
4	jv	0	1
5	and	702	464
6	frosh	0	0
7	softball	4	1
8	@	1324	802
9	jt	0	0
10	central	6	3
11	4	47	41
12	:	2600	4988
13	30	15	14
14	TOTAL	4865	6546

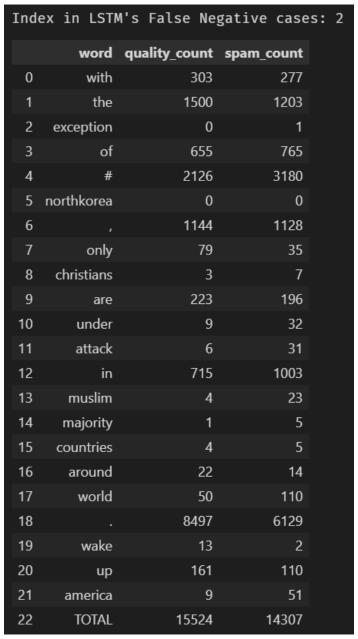

Index in LSTM's False Positive cases: 7

	word	quality_count	spam_count
0	if	180	89
1	warriors	5	9
2	fans	18	9
3	want	77	33
4	to	1308	1203
5	talk	32	14
6	about	128	119
7	a	1029	660
8	travel	22	6
9	.	8497	6129
10	https	552	3734
11	:	2600	4988
12	vine	27	1
13	co	79	3921
14	v	84	8
15	i2y3r2j3jii	0	0
16	TOTAL	14638	20923


Index in LSTM's False Positive cases: 14

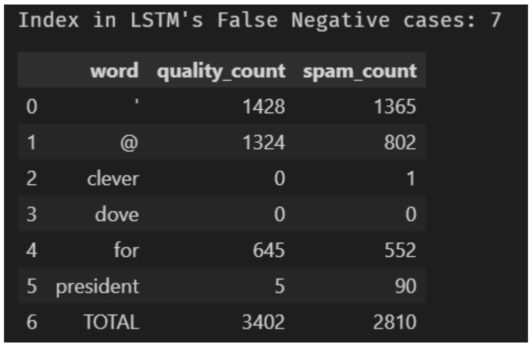
	word	quality_count	spam_count
0	diving	0	0
1	into	50	48
2	some	89	37
3	makeup	5	0
4	art	18	5
5	today	97	46
6	3	61	54
7	#	2126	3180
8	litcosmetics	0	0
9	beglamorousbylindsay	0	0
10	makeupideapic	0	0
11	.	8497	6129
12	twitter	1839	15
13	com	2319	3
14	t4drqiikqe	0	0
15	TOTAL	15101	9517

5. Samples of LSTM’s false negative cases



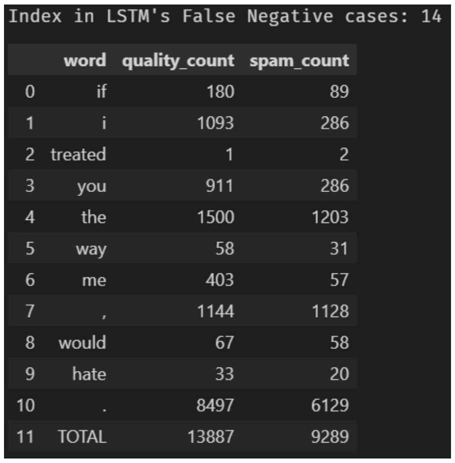
```
Index in LSTM's False Negative cases: 2
```

	word	quality_count	spam_count
0	with	303	277
1	the	1500	1203
2	exception	0	1
3	of	655	765
4	#	2126	3180
5	northkorea	0	0
6	,	1144	1128
7	only	79	35
8	christians	3	7
9	are	223	196
10	under	9	32
11	attack	6	31
12	in	715	1003
13	muslim	4	23
14	majority	1	5
15	countries	4	5
16	around	22	14
17	world	50	110
18	.	8497	6129
19	wake	13	2
20	up	161	110
21	america	9	51
22	TOTAL	15524	14307



```
Index in LSTM's False Negative cases: 7
```

	word	quality_count	spam_count
0	'	1428	1365
1	@	1324	802
2	clever	0	1
3	dove	0	0
4	for	645	552
5	president	5	90
6	TOTAL	3402	2810



```
Index in LSTM's False Negative cases: 14
```

	word	quality_count	spam_count
0	if	180	89
1	i	1093	286
2	treated	1	2
3	you	911	286
4	the	1500	1203
5	way	58	31
6	me	403	57
7	,	1144	1128
8	would	67	58
9	hate	33	20
10	.	8497	6129
11	TOTAL	13887	9289

8. References

Associated Press. (2022, April 28). Twitter daily user growth rises as Musk readies to take control. *Al Jazeera*. Retrieved 5 May 2022, from https://www.aljazeera.com/economy/2022/4/28/twitter-daily-user-growth-rises-as-musk-readies-to-take-control

Benevenuto, F., Magno, G., Rodrigues, T., & Almeida, V. (2010). Detecting spammers on Twitter. *7th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference*. http://www.cse.fau.edu/~xqzhu/courses/Resources/FBG.Spam.pdf

Chollet, F. (2015). *Keras*. https://keras.io

Collins, B. (2020, August 28). Viral pro-Trump tweets came from fake African American spam accounts, Twitter says. *NBC News*. https://www.nbcnews.com/tech/security/viral-pro-trump-tweets-came-fake-african-american-spam-accounts-n1238553

Gautam, G., & Yadav, D. (2014). Sentiment analysis of Twitter data using machine learning approaches and semantic analysis. *2014 Seventh International Conference on Contemporary Computing (IC3)*, 437-442. https://doi.org/10.1109/ic3.2014.6897213

Gomaa, W. H. (2020). The impact of deep learning techniques on SMS spam filtering. *International Journal of Advanced Computer Science and Applications*, *11*(1). https://doi.org/10.14569/ijacsa.2020.0110167

Gordillo, J., & Conde, E. (2007). An HMM for detecting spam mail*. Expert Systems with Applications*, *33*(3), 667-682. https://doi.org/10.1016/j.eswa.2006.06.016

Jayaswal, V. (2020, November 22). Laplace smoothing in Naïve Bayes algorithm. *Towards Data Science*. https://towardsdatascience.com/laplace-smoothing-in-na%C3%AFve-bayes-algorithm-9c237a8bdece

Kaggle. (2019). *UtkML's Twitter spam detection competition*. Retrieved May 6, 2022, from https://www.kaggle.com/competitions/twitter-spam/overview

Kumar, A.K., & Sahni, S. (2011). A comparative study of classification algorithms for spam email data analysis. *International Journal on Computer Science and Engineering*, *3*(5), 1890-1895. https://www.researchgate.net/publication/267718689_A_Comparative_Study_of_Classification_Algorithms_for_Spam_Email_Data_Analysis

Lai, S., Xu, L., Liu, K., & Zhao, J. (2015). Recurrent Convolutional Neural Networks for Text Classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, *29*(1). https://dl.acm.org/doi/10.5555/2886521.2886636

Lee, K., Eoff, B., & Caverlee, J. (2011). Seven Months with the Devils: A Long-Term Study of Content Polluters on Twitter. *Proceedings of the International AAAI Conference on Web and Social Media*, *5*(1), 185–192. https://ojs.aaai.org/index.php/ICWSM/article/view/14106

Liu, X., Lu, H., & Nayak, A. (2021). A spam transformer model for SMS spam detection. *IEEE Access*, *9*, 80253–80263. https://doi.org/10.1109/access.2021.3081479

Markines, B., Cattuto, C., & Menczer, F. (2009). Social spam detection. *Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web - AIRWeb '09*. https://doi.org/10.1145/1531914.1531924

McCord, M., & Chuah, M. (2011). Spam detection on Twitter using traditional classifiers. *Lecture Notes in Computer Science*, 175–186. https://doi.org/10.1007/978-3-642-23496-5_13

Park, J.S., & Deshpande, A. (2006). Spam detection: Increasing accuracy with a hybrid solution. *Information Systems Management*, *23*(1), 57-67. https://doi.org/10.1201/1078.10580530/45769.23.1.20061201/91773.7

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Louppe, G., Prettenhofer, P., Weiss, R., Weiss, R. J., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* https://doi.org/10.5555/1953048.2078195

Saab, S. A., Mitri, N., & Awad, M. (2014). Ham or spam? A comparative study for some content-based classification algorithms for email filtering. *MELECON 2014 - 2014 17th IEEE Mediterranean Electrotechnical Conference*, 339-343. https://doi.org/10.1109/melcon.2014.6820574

Shahariar, G. M., Biswas, S., Omar, F., Shah, F. M., & Hassan, S. B. (2019). Spam review detection using deep learning. *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 0027-0033. https://doi.org/10.1109/IEMCON.2019.8936148

Sharaff, A., Nagwani, N. K., & Dhadse, A. (2015). Comparative study of classification algorithms for spam email detection. *Emerging Research in Computing, Information, Communication and Applications*, 237–244. https://doi.org/10.1007/978-81-322-2553-9_23

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *31st Conference on Neural Information Processing Systems (NIPS 2017)*, 5998-6008. https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Wang, A. H. (2010). Don’t follow me: Spam detection in Twitter. *2010 International Conference on Security and Cryptography (SECRYPT)*, 1-10. https://ieeexplore.ieee.org/abstract/document/5741690