CMT 307 Coursework 2 Group Project

Group Number	1
Project Title	Text Categorization
Supervisor	Dimosthenis Antypas
Student ID	C1009399 C21112778 C1999659 21107524 21036048 2062079 C21043704

Text Categorisation of the 20 NewsGroups data set using Neural Network models

Abstract

Exploring a range of machine learning algorithms including convolutional neural networks and recurrent neural networks were able to successfully categorise text to a high degree of accuracy. This analysis determined an optimised RNN Bidirectional LSTM model to be the most successful at this task, with an accuracy score of **79.02**%.

Introduction

Text categorisation is the automatic assignment of natural language (i.e., human language) to predetermined categories based on its content. (Lewis et al., 1994). This project classifies a document of English language text into 1 of 20 distinct categories having been trained on the well-known "20 Newsgroups" dataset. This data set contains over 20,000 text documents spread across 20 distinct categories, for example, "motorcycles", "hockey" and "religion". These plain-text data take the form of email messages about the specified topic.

To perform this, the data were appropriately pre-processed (including cleaning, tokenising, and vectorised). This project experimented with using simple algorithms such as Naïve Bayes and KNN and more advanced neural network models, with or without pre-trained embedding to categorise text into the relevant 20 NewsGroups categories.

Literature Review

Ethics of Using Neural Networks

Utilitarianism

When neural networks are used to evaluate statistics and quantitative data, they exhibit advantages and disadvantages. For instance, the algorithm can have significant ethical usage when performing beneficent tasks, like safely directing traffic. The algorithm can effectively identify complex and unintuitive patterns that average human being cannot remember. Additionally, the minority classes are likely to be crashed by the utilitarian philosophy in social situations. For instance, if we use the algorithm to determine whether a bank should grant certain people loans or not, and the race attribute is used as the boundary of the neural networks, the algorithm will make racial judgments. However, it is essential to note that the utilitarian philosophy enables neural networks to perform well. Therefore, it is necessary to set the algorithm with boundaries that do not cause harm.

Literature Review:

This project shows various challenges and techniques involved in text categorization. Naïve Bayes and KNN are used here as Naïve Bayes (Pavan, 2021) performs well for categorical input and

requires less training data similarly does KNN. Doc2vec model is used as it contributes to better efficiency in categorizing the text so does the author (Akhtar, 2021) explains too.

For even further accuracy, CBOW and Continuous skip-gram methods are also implemented since skip-gram is good for finding rare words and phrases while CBOW works with slightly better accuracy in frequent words (Ria, 2019), thus the combination of these two models yields better accuracy. The training speed can be maximized by using the hybrid model of count-based and window based which is the GloVe model- this study supports the same (Ria, 2019). Further studies show that the use of Bidirectional LSTM is a better approach as there are no restrictions on the flow of data and aids in better prediction (C. Li, G. Zhan, and Z. Li, 2018). The blog by Cezanne Camacho (Cezanne, 2017) explains the paper of Kim (Kim, 2014) that 300 kernels and 100 kernels for each height to cut off the words. This supports the use of CNN and 1-dimensional CNN with max pooling.

The above-mentioned authors have used various methods of deep learning and machine learning techniques and applied to categorize the news text in this project. These studies show that advanced text categorization is predominantly deep learning technique with subdomains and is also suitable for large data with multiclass text categorization.

Cleaning and Pre-processing

Text pre-processing is a critical task due to its major influence on text analysis and Natural language processing (NLP), with bad data pre-processing resulting in cumulative negative effects. The goal of pre-processing step is to convert data into a predictable and analysable format, allowing machine learning models to produce high-quality results. The outcome is a clean text in human language that has been organized into a machine-readable format because any text being fed to a model must be in a specific format. Only cleaning techniques that have been proven its effectiveness in text classification projects were followed which are capitalization, noise removal, stop words, and lemmatization.

Capitalization
We used the lowercasing technique to eliminate the sparsity issue and reduce the vocabulary size. It helped us in considering the same word written in caps, camelCase, and small letters as same. For example: GREAT, Great & great will be considered as same. This is the simplest technique of text pre-processing that deals with sparsity issues in the dataset by eliminating this variation, so it does not cause further problems.

Raw	Lowercased
Canada CanadA CANADA	canada
TOMCAT Tomcat toMcat	tomcat

Figure 1 An example of lowercasing as part of pre-processing

Noise Removal:

This method involves deleting text that isn't required for text mining or categorization, such as punctuation and special characters. The written code removed all the headers and numbers and then dropped any word containing '@','.com' or 'http' and the last step is removing all punctuations such as !, $, ( ), * and % etc.

Stop Words:
The noisy character of text corpora (abbreviations, irregular forms) affects text and document categorization via the internet. Here is a list of commonly repeated words that emerge in every text document "the", "a", "an", "so", "what" in the English language. By removing these words, we remove the low-level information from our text to give more focus to the important information.

Lemmatization:
The process of transforming a word into its meaningful base or root form is based on its context hence, the objective of the written code is to return any word to its basic form by removing inflections. The used function in the lemmatization method is WordNetLemmatizer() which is found Natural Language Tool Kit (NLTK). The following example shows the output of lemmatization method based on WordNet-based approach.

	original_word	lemmatized_word
0	trouble	trouble
1	troubling	trouble
2	troubled	trouble
3	troubles	trouble

	original_word	lemmatized_word
0	goose	goose
1	geese	goose

Figure 2 Examples of lemmatization

Word Embedding:
After the cleaning stage, data must be converted to numerical data/vector format before feeding it into a Machine Learning Model to perform mathematical operations in a technique known as word embedding. It helps to calculate a word's association with another word with a comparable meaning through the established vectors. As shown in the graphic below where word embeddings are mapped, words with similar semantic meanings are clustered together in space, demonstrating semantic closeness.

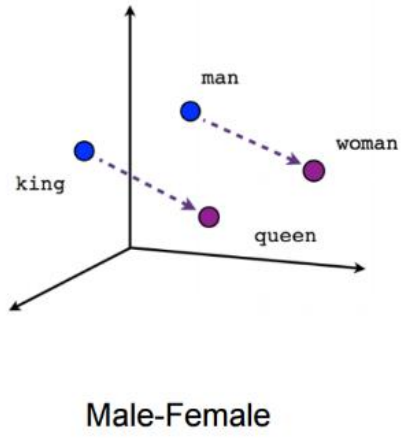


Figure 3 Examples of word embedding

There are many Embedding techniques for conversion, in this project, three techniques are used for word embeddings which are doc2vec in the first version, word2vec and GLOVE in the second version. We used Word2Vec which takes corpus of texts as input and generates a vector for each word. In vector space, closeness denotes semantic or functional similarity. So, Word2Vec models are good at identifying word-to-word semantic links. The Continuous Bag-of-Words (CBOW) model and the Continuous Skip-Gram Model were established as two distinct models that can be utilized as a part of the word2vec approach to accomplish the word embedding. The CBOW model learns the embedding by anticipating the current word depending on its context. The skip-gram learns by predicting the surrounding words given a current word.

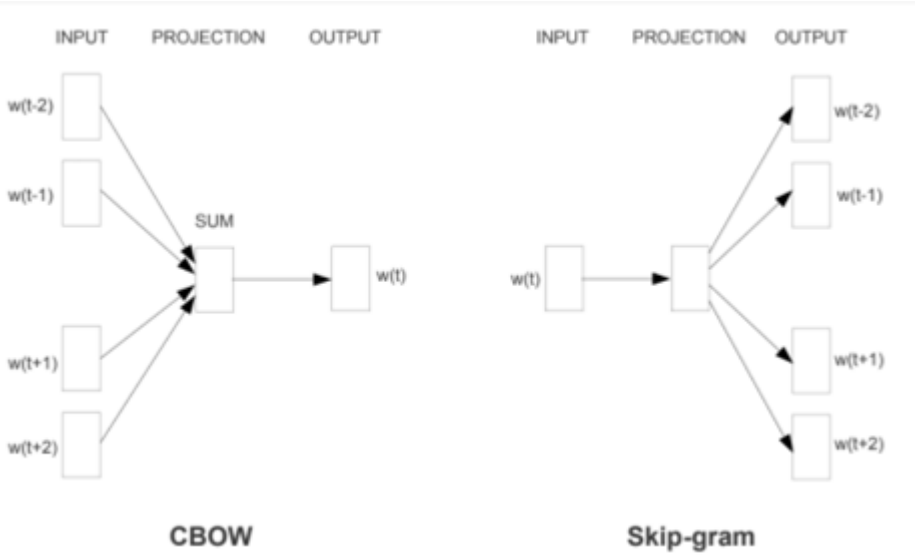


Figure 4 Visualisation of CBOW and Skip-gram models

We also utilised Doc2Vec technique which is a commonly used technique for embedding a document regardless of its length. It calculates a feature vector for each document, whereas Word2Vec calculates a feature vector for each word. Therefore, Doc2Vec depends on Word2Vec, with the addition of a new vector (Paragraph ID) as an input. The following diagram depicts the Doc2Vec model's architecture:

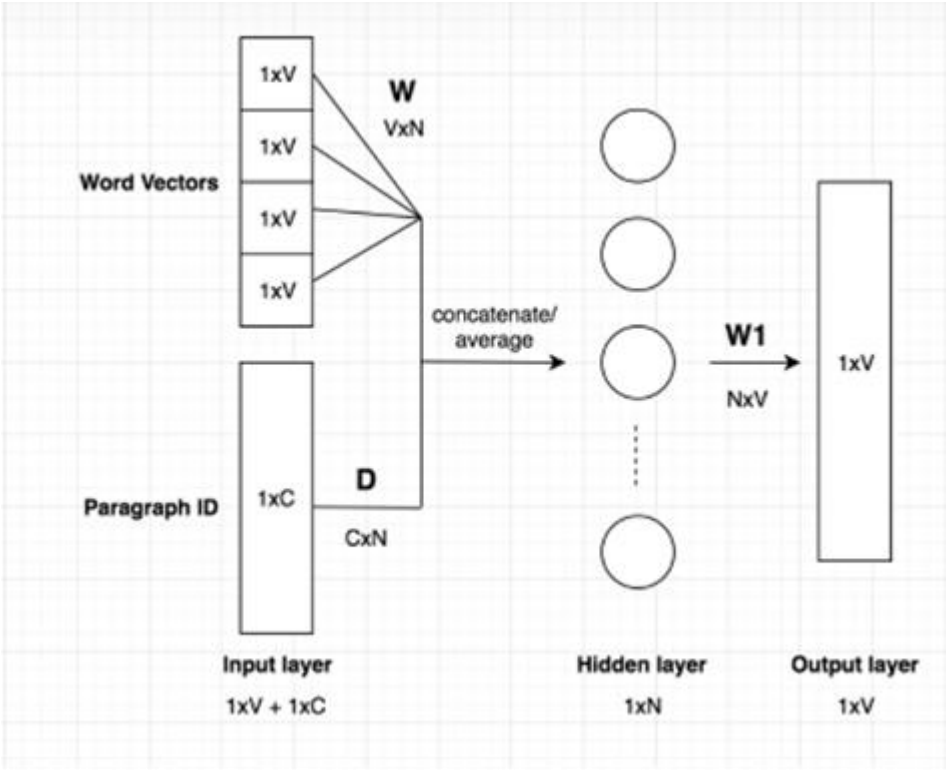


Figure 5 Architecture of the Doc2Vec model

The document vector D is trained along with the word vectors W, and after the training, it provides a numeric representation of the document.

Also used was GloVe (Global Vectors for Word Representation), an extension of Word2Vec to capture global contextual information in a text corpus by calculating a global word-word co-occurrence matrix. During training, Word2Vec only considers neighbouring words to capture the context while GloVe takes the full corpus and generates a large matrix that can take the co-occurrence of words. The co-occurrence matrix shows how frequently a word pair appears together.

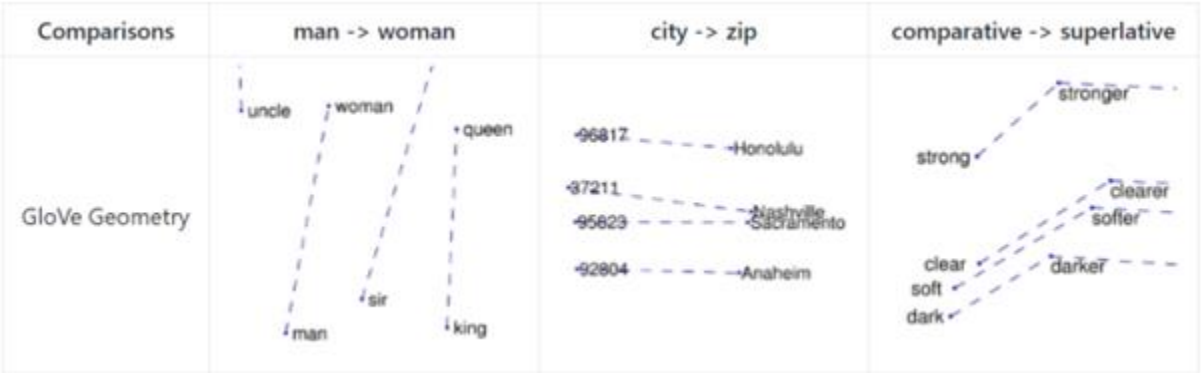


Figure 6 Examples of co-occurrences in GloVe

Descriptive Analysis

The 20 Newsgroup data set is comprised of approximately 20,000 distributed amongst 20 categories, each with approximately 900 documents in each. The dataset provided had already been split into training (60%) and testing (40%) datasets.

For effective categorisation the training data should be distributed approximately equally between each group to reduce bias when training.

Of the training data set there were between 377 and 600 documents with a corpus ranging from 35,869 to 101,427 (with the exclusion of stop-words) for each category. The proportion of the cleaned corpus ranges from 2.6% (rec.autos) to 9.2% (alt.atheism). This is within an order of magnitude so we can consider the data to be distributed equally. However, these differences may be a feature we would want to incorporate into our model.

Table 1 Summary statistics of the 20 NewsGroups training data

Index	Category	Training data Number of Documents	Training data Word Count	Average Words per Doc	Proportion of Corpus (%)
0	rec.sport.hockey	600	67385	112.3	4.9
1	soc.religion.christian	599	56353	94.1	4.1
2	rec.motorcycles	598	50948	85.2	3.7
3	rec.sport.baseball	597	48298	80.9	3.5
4	sci.crypt	595	42867	72.0	3.1
5	sci.med	594	79624	134.0	5.8
6	rec.autos	594	35869	60.4	2.6
7	sci.space	593	55166	93.0	4.0
8	comp.windows.x	593	49193	83.0	3.6
9	comp.os.ms-windows.misc	591	54828	92.8	4.0
10	sci.electronics	591	72390	122.5	5.3
11	comp.sys.ibm.pc.hardware	590	101427	171.9	7.4
12	misc.forsale	585	51022	87.2	3.7
13	comp.graphics	584	78629	134.6	5.7
14	comp.sys.mac.hardware	578	81608	141.2	5.9
15	talk.politics.mideast	564	92699	164.4	6.8
16	talk.politics.guns	546	85512	156.6	6.2
17	alt.atheism	480	125966	262.4	9.2
18	talk.politics.misc	465	88406	190.1	6.4
19	talk.religion.misc	377	54878	145.6	4.0
Total		**11314**	**1373068**		
Average		**565.7**	**68653.4**		

Common Words

Following the pre-processing of the text data but before vectorisation we performed several analyses on the data. Once stop-words are removed we found the most common words in the data set.

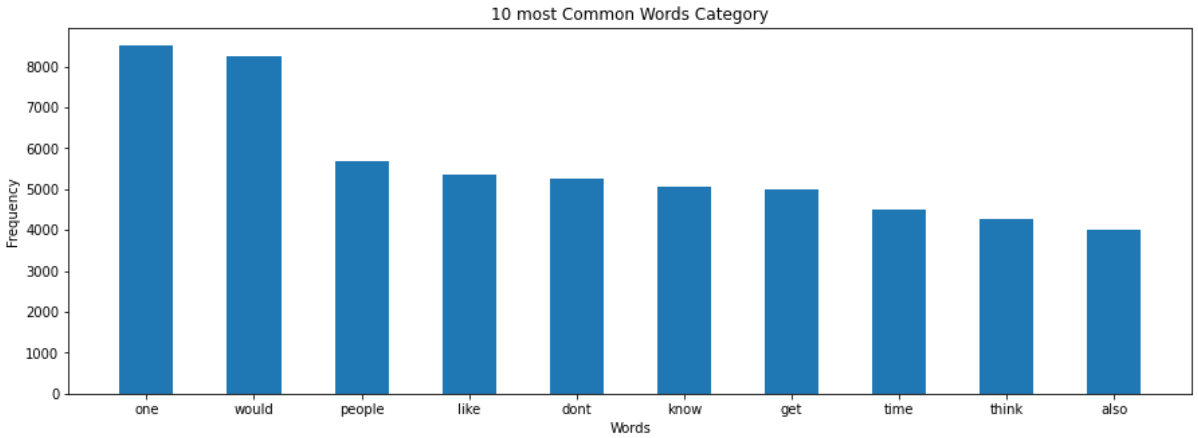


Figure 7 Most common words in the cleaned 20NewsGroups training data

As we can see in Figure 7, the most common words across all categories are generic words with little meaning. By analysing the most frequent occurrences of words in each category we can see if there are real differences between categorisations. Such processes could also identify additional project-specific stop-words which should be removed as part of the pre-processing procedure.

There is a marked difference in the top 10 most common words between each category. For example, for the category **rec.sport.hockey** (Figure 8), the words "*team*", "*game*" and "*player*" appear most frequently. In contrast the words "*drive*", "*scsi*", "*card*" and "*system*" appear most frequently for the **comp.sys.ibm.pc.hardware** category (Figure 9). From our prior knowledge of these topics, many of these results are to be expected.

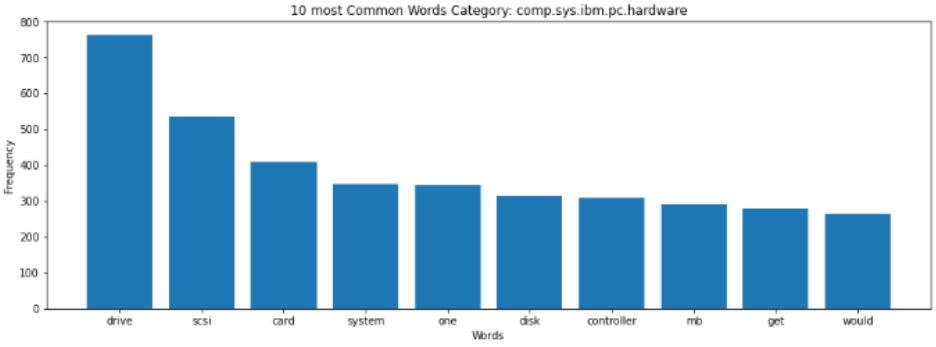


Figure 8 Most common words in the comp.sys.ibm.pc.hardware category

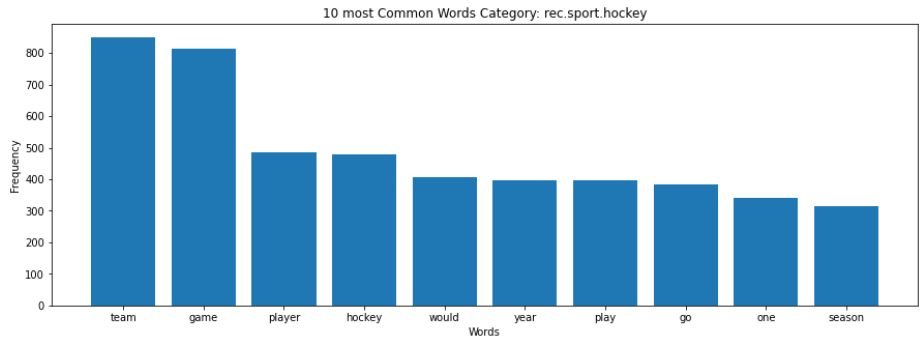


Figure 9 Most common words in the rec.sport.hockey category

Some words appear frequently in many different categories (such as the words “one” and “would”) and therefore may not be particularly useful at differentiating texts.

Bigrams

Bigrams are pairs of words that appear consecutively in the text. In some cases, pairs of words that appear frequently together hold a specific meaning and could have the potential to improve a neural network model. (Yang, 2020)

In this analysis, bigrams were identified using *nltk bigram* (Figure 10). The most common bigram which begins with “maxaxax ...” is a collection of letters found in every document of the *computer* category and therefore could be a significant indicator when categorising documents.

Other common bigrams are more familiar combinations such as “dont know”, “law enforcement” and “san francisco”.

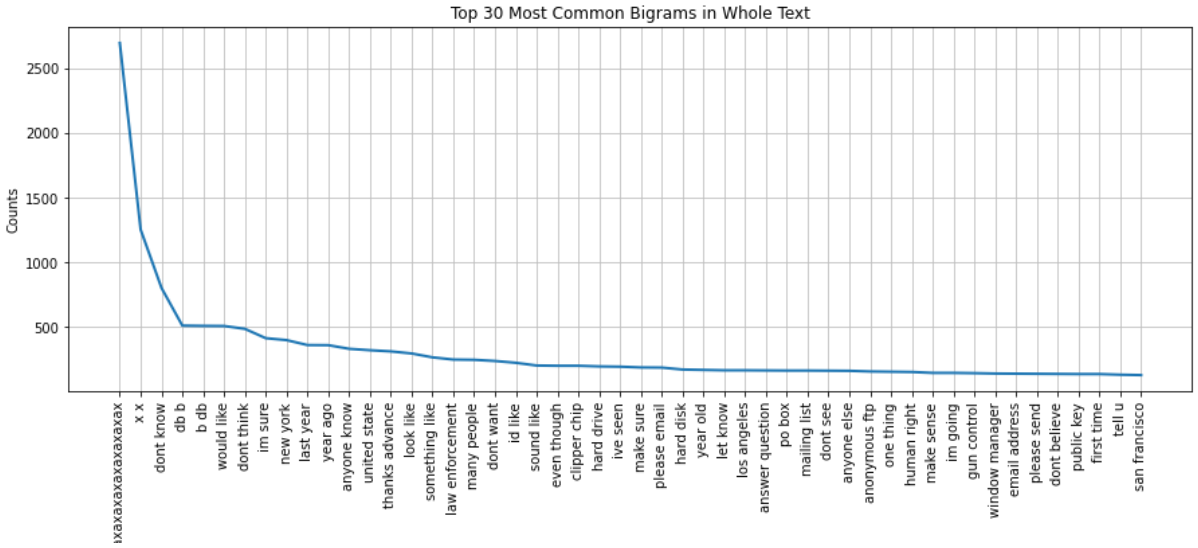


Figure 10 Most common bigrams in the cleaned 20 NewsGroups training dataset as identified using nltk bigram.

Pointwise mutual information (PMI) is a measure of association, that is how highly correlated variable x is with variable y. **BigramAssocMeasures** from nltk (in conjunction with **BigramCollocationFinder**, a more sophisticated method of identifying bigrams) calculates the PMI, which for the first 20 bigrams gave a value of 20.30. This suggests there are no significant differences between these bigrams.

Part of Speech (POS) Analysis

In the English language, words can be classified according to their grammatical properties. After applying tokenisation, we identified the number of nouns, verbs, adverbs, adjectives, and other (that is any character that could not be classified into the aforementioned groups) using ***nltk pos_tag***. For ease of comparability these results were scaled as a proportion of the total number of words in that category.

The POS composition (see figure 10) shows a similar proportion between each of the 20 categories, with the number of nouns accounting for roughly 50% of the total number of words.

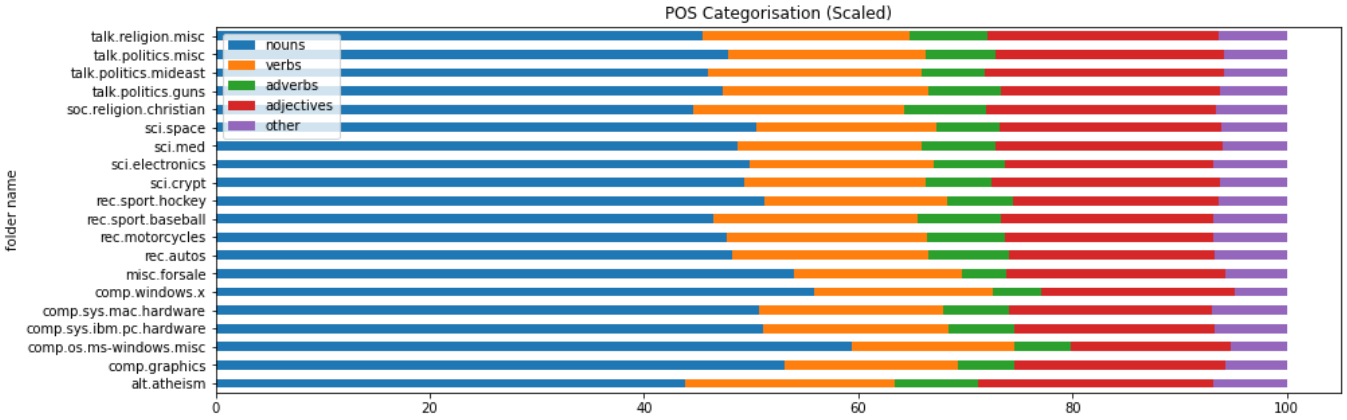


Figure 11 Proportion of parts of speech by each category

Model Implementation

Preparation of data

One-Hot Encoding for Multi Categorical Variables

After completing the Vectorization process, the category encoding code process is needed to be run at the dataset. The reason is the categories that are assigned as (y) are not encoded during the vectorization process and are still in string form.

The encoding process consists of two levels. Firstly by mapping each unique category to an integer number [0,1, 2, .. 19] using Numpy's function. Secondly, after that, keras.utils.to_categorical function is used to convert a class vector (integers) to binary class matrix. Finally, at the end of the encoding code block we will have encoded categories in [0s,1s] form so we are ready to plug into the models.

Advantages of using One-Hot Encoding For text classification projects

This process is straightforward to implement. There are no complex steps as well as it can be done in a short amount of code. In addition, variable investigation does not take much time as it is in binary form, computing will be fast. The feature space isn't greatly expanded: which is helpful especially during working in a virtual environment such as Google Collaboratory.

To summarize, using one-Hot Encoding for converting categories into binary form has approved its efficiency in terms of hardware such as memory space and software such as reducing the code lines.

Model evaluation

The dataset contains three parts, i.e., training, validation and test sets. The validation set is used to do parameter tuning during the model construction process, while test set is an isolated set to objectively evaluate model performance.

Non-neural network and “simple” neural network Models

Naïve Bayes

Naïve Bayes classifier was employed in this project, which is a kind of supervised learning model. Naïve Bayes is formed by Bayes theorem and naive assumptions of conditional independence between the features. In other words, the Naïve Bayes model structure makes no attribute variable have a significant weight for the decision result, and no attribute variable has a small weight for the decision result. Considering that there are only a few parameters used for model construction, Naïve Bayes is not quite sensitive to missing values in the dataset. In addition, Naïve Bayes is robust even datasets present different characteristics since it has relatively simple logic and model structure. Therefore, Naïve Bayes is widely used for classification tasks, especially text classification. It shows outstanding performance when employed in uncomplicated text classification.

The Naïve Bayes model was constructed through function **GaussianNB ()** from **sklearn.naive_bayes.** The 7580 training records formed by 40 variables were used for training, and a validation set of 3734 rows was used for parameter tuning. The core parameter var_smoothing is tuned through grid search. After seeing the optimal value of var_smoothing (0.0001), the final model was rebuilt and evaluated by a separate test set of 7537 observations.

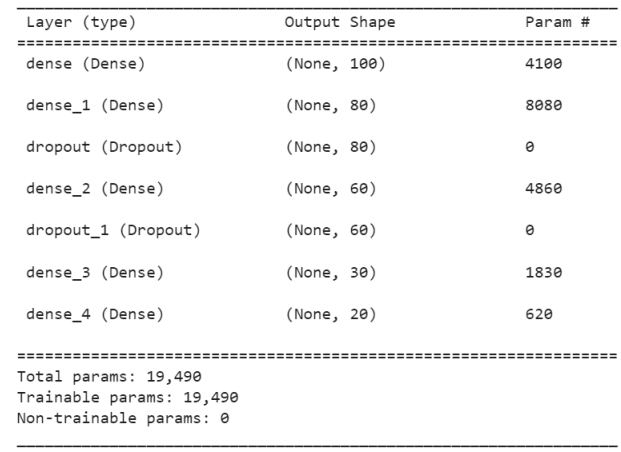
KNN

The second employed model is the k-nearest neighbours’ algorithm (k-NN), which is a non-parametric supervised learning method. In the KNN algorithm, the classification of a point is determined by the majority voting of its neighbours’ classes. And the most common class of the k nearest neighbours determines the class assigned to the point. KNN is one of the simplest classification models. One the parameter of K needs to be tuned.

The KNN model here was built by **KNeighborsClassifier ()** function from **sklearn.neighbors**. The same training, validation and test set for Naïve Bayes were also used here. Only K (n neighbors) was tuned with grid search. The final KNN model was trained with the optimal K of 1.

ANN

The Artificial neural network was built here with dense layer 1 of 100 neurons and a ReLU activation function and dense layer 2 of 80 neurons and a ReLU activation function. After that, a Dropout layer was added to prevent overfitting. Another dense layer was constructed with 60 neurons and a ReLU activation function, followed by a dropout layer and one more dense layer (30 neurons). The output layer used the sigmoid activation function. Details can be found in Figure 1212 and Figure 13.



```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               4100

 dense_1 (Dense)             (None, 80)                8080

 dropout (Dropout)           (None, 80)                0

 dense_2 (Dense)             (None, 60)                4860

 dropout_1 (Dropout)         (None, 60)                0

 dense_3 (Dense)             (None, 30)                1830

 dense_4 (Dense)             (None, 20)                620

=================================================================
Total params: 19,490
Trainable params: 19,490
Non-trainable params: 0
_________________________________________________________________
```

Figure 12 Sequential model structure.

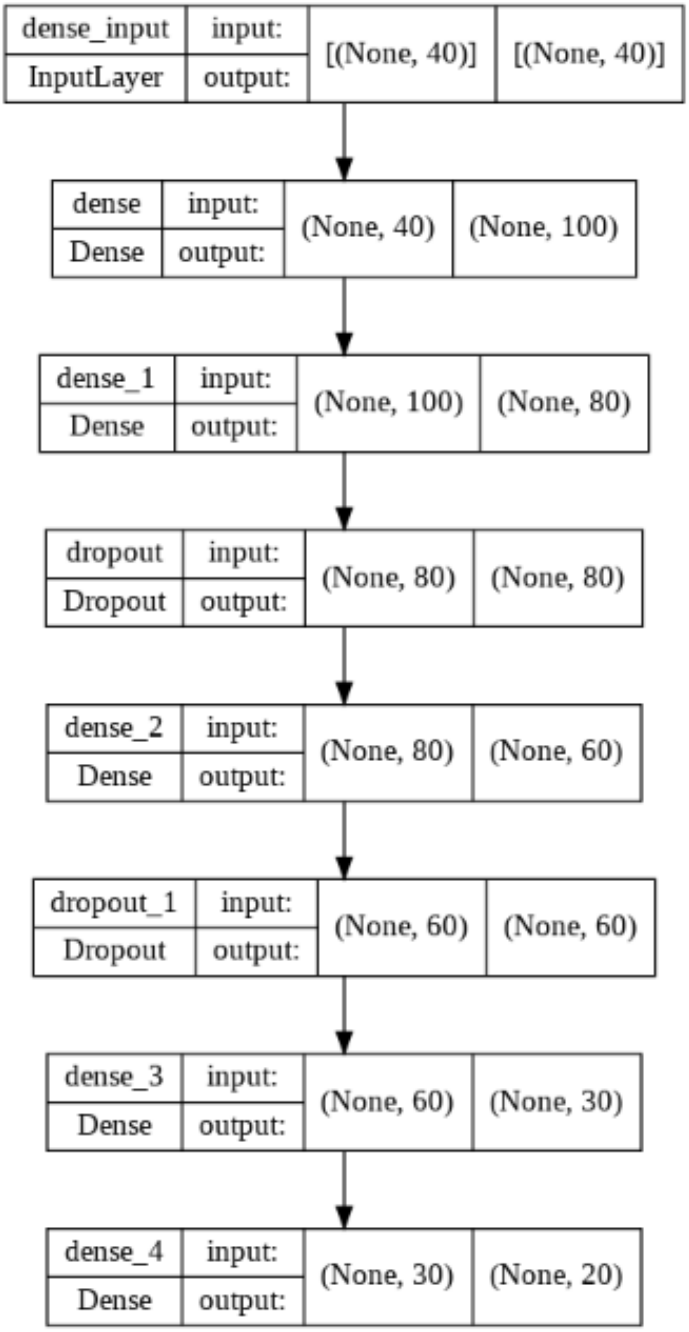


Figure 13 Graphical representation of Sequential Model.

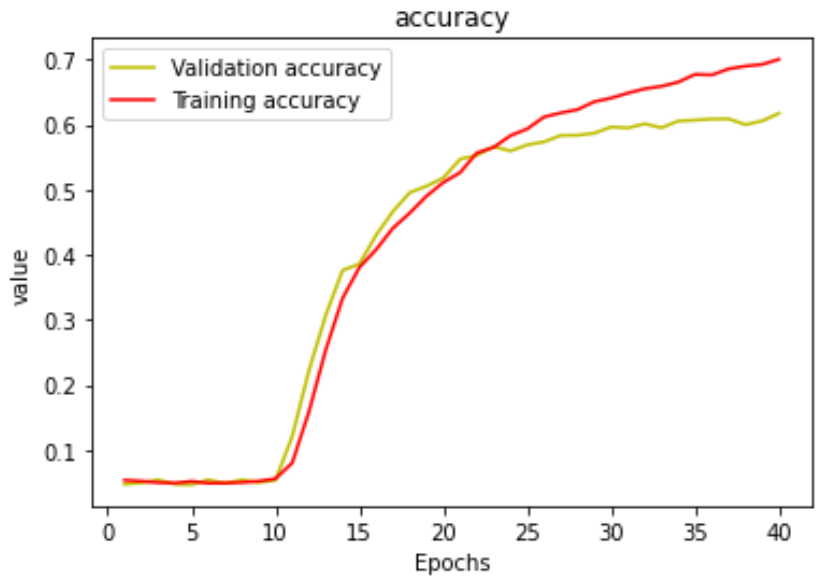


Figure 14 Accuracy and Validation Loss of Sequential Model.

CNN

In Convolutional neural network, the convolution layer aims to slide a filter over the input, which consists of a series of learnable filters. As shown in Figure 14 and Figure 15, 1 dimensional CNN was built without using pre-trained embedding. The embedding layer here was constructed by setting input_dim as 20002 and output_dim as 200, followed by a Dropout layer with a rate of 0.5. Thereafter, two 1d convolutional layers with filters and kernel_size of 128 and 7, as well as Global Max Pooling operation, were added. We then added a vanilla hidden layer, including a Dense layer and a Dropout layer. As for output layer, we projected it onto a multiclass unit output layer, and squash it with a softmax activation function. The model was compiled with a multiclass cross-entropy loss function and a rmsprop optimizer. The used package in Python is Keras built on top of TensorFlow.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, None)]	0
embedding (Embedding)	(None, None, 200)	4000400
dropout_2 (Dropout)	(None, None, 200)	0
conv1d (Conv1D)	(None, None, 128)	179328
conv1d_1 (Conv1D)	(None, None, 128)	114816
global_max_pooling1d (GlobalMaxPooling1D)	(None, 128)	0
dense_5 (Dense)	(None, 128)	16512
dropout_3 (Dropout)	(None, 128)	0
predictions (Dense)	(None, 20)	2580

Total params: 4,313,636
Trainable params: 4,313,636
Non-trainable params: 0

Figure 14 CNN model structure.

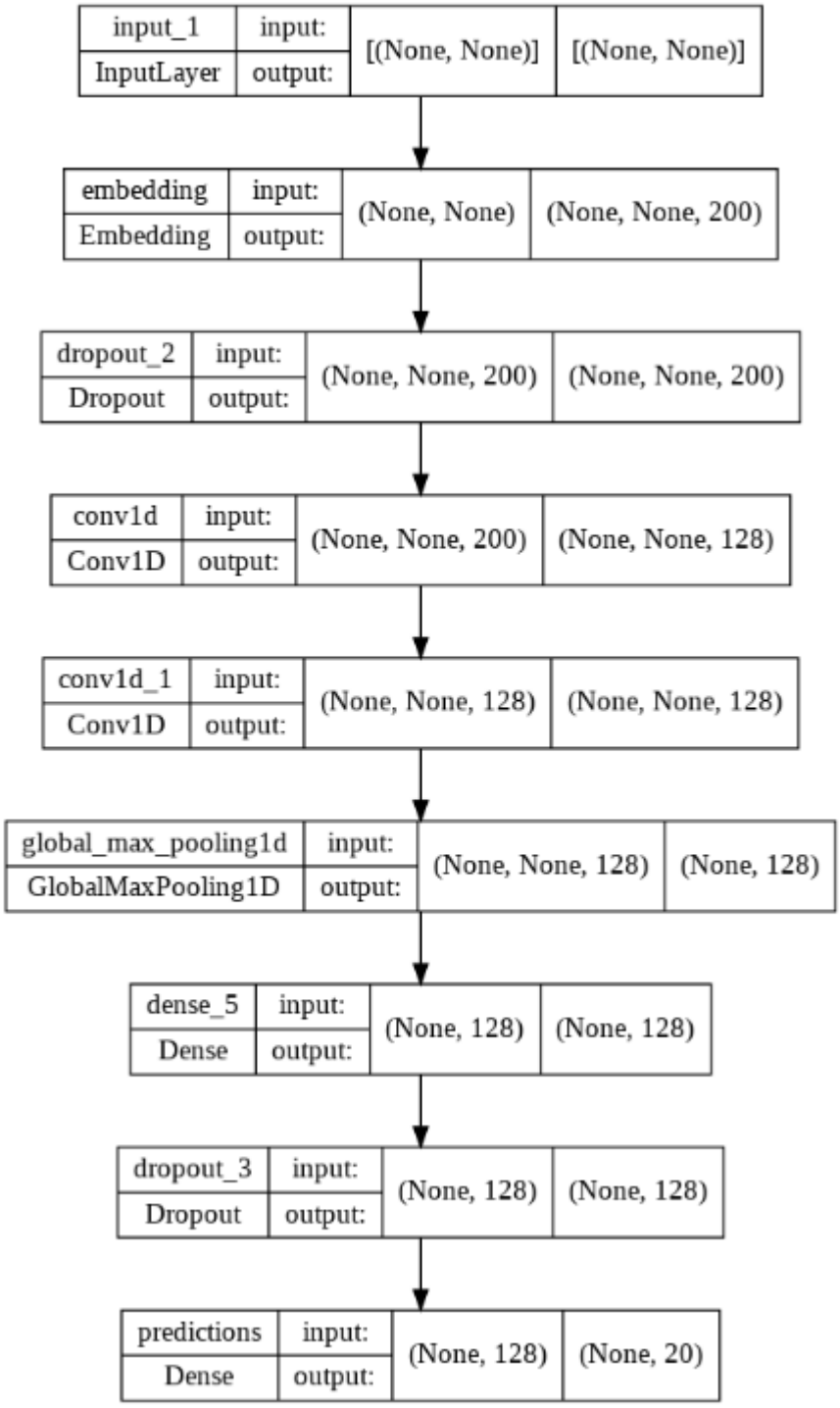


Figure 15 Graphical representation of CNN Model

Tuning Hyperparameters

Naïve Bayes

The var_smoothing was tuned with grid search for Naïve Bayes. The var_smoothing is the portion of the largest variance of all features that are artificially added to variances for calculation stability, which

aims to smooths the curve and account for more samples that are further away from the distribution mean. The candidate values for this parameter contain 1e-1,1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12, 1e-13, 1e-14, 1e-15. The grid search was realized by hold-out validation, which always trains the model with candidates on training set and tests performance with fixed validation set. And according to Table 2, the value of 0.0001 was finally selected.

Table 2 Grid search results for Naïve Bayes.

var_smoothing	Accuracy
1.000000e-01	0.412962
1.000000e-02	0.414301
1.000000e-03	0.411623
1.000000e-04	0.411355
1.000000e-05	0.411355
1.000000e-06	0.411355
1.000000e-07	0.411355
1.000000e-08	0.411355
1.000000e-09	0.411355
1.000000e-10	0.411355
1.000000e-11	0.411355
1.000000e-12	0.411355
1.000000e-13	0.411355
1.000000e-14	0.411355
1.000000e-15	0.411355

KNN

The K value of KNN represents the count of the nearest neighbours. A suitable K value is important for model training since a small value of k makes the model easy to be influenced by while a large value makes it computationally expensive. Therefore, the most appropriate K value for this dataset was determined by a grid search of hold-out validation. The candidates for K values include integers ranging from 1 to 30. And finally, the value of 1 was chosen here based on the results in Table 3.

Table 3 Grid search results for KNN.

K	Accuracy	K	Accuracy
1	0.568827	16	0.553830
2	0.500536	17	0.553026
3	0.524103	18	0.555169
4	0.532137	19	0.555972
5	0.541778	20	0.553026
6	0.541510	21	0.553294
7	0.549813	22	0.554365
8	0.549813	23	0.551152
9	0.555437	24	0.549277
10	0.551955	25	0.555437

11	0.551419	26	0.556776
12	0.549277	27	0.555169
13	0.555437	28	0.550884
14	0.556240	29	0.554097
15	0.557847	30	0.554901

CNN

As for the CNN model, we did not include any complicated grid search work because it is quite time-consuming. The model was trained with epochs of 50 and batch_size of 64. An early stopping strategy was used that stops model training when validation loss is no longer improving (no better than 1e-2 less for at least 2 epochs). The accuracy and loss for training and validation set during model training are shown in Figure 16 and 1511.

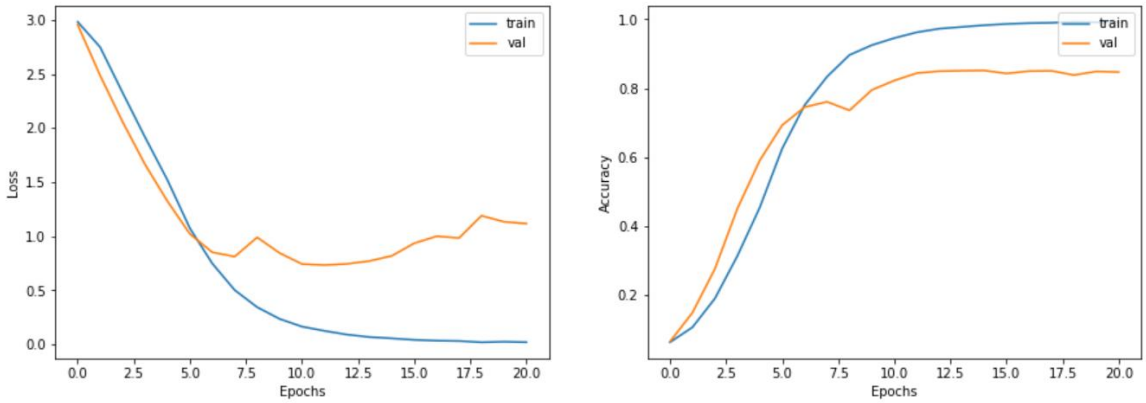


Figure 16 and 15 Accuracy and Validation Loss of CNN Model

Advanced Neural Network Models

For these advanced models word embeddings Word2Vec and GloVe are implemented in Python using the API provided by Gensim. These models use GloVe because of the model's simplicity in creating the embedding layer. In this project we only test the 100- and 200-dimension vectors for our embedding layer. 200 appears to give a better result compared to another parameter configurations.

The pre-trained GloVe word embeddings have already been downloaded and will be used. A zip file containing 400K words and their embeddings was downloaded for this example.

We used the embedding matrix, number of tokens [20000,30000], and embedding dimension [100,200] to develop our neural network's pretrained embedding layer. Since we are already employing training embedding in our Neural network, we set the training to false.

Because neural networks operate on vectors, we turned all the training, validation, and testing data into an array. This is critical as neural networks are designed to perform operations on vectors. The vectors are represented as multidimensional arrays in your code. Vectors are useful in deep learning for a variety of reasons, one of which is a particular operation called dot product. The dot product of

two vectors informs you how similar they are in terms of direction and is scaled by the magnitude of the two vectors that are being used to compute it.

1 dimensional CNN | Conv1D Maxpooling and GlobalMax Pooling Using PreTrained Embedding (from Keras)

This model was sourced from the Keras documentation (Keras, 2016) to serve as a comparison to other models. Feature mapping is made by convolutional layers of a convolutional neural network by applying learned filters to input data repeatedly. It's easier for lower-level features to be learned at input levels nearer the input, while higher-level features (like forms and specific patterns) are learned at input levels deeper in the model. This is because convolutional layers in deep models are stacked.

Maximum pooling is a process that looks for the biggest value in each patch of each feature map. Another type of pooling global pooling can be used to quickly summarise a feature in a data set. It is also possible to move from feature data to the model's predictions by using a linked layer rather than using a fully linked layer.

In this Conv1D one-dimensional convolutional neural network, we use Maxpooling and GlobalMax Pooling with size of five close to input and output dense layer respectively. We then use the pretrained embedding layer to train the model. Callbacl, earlystopping and modelcheckpoint are implemented to optimise the process.

Our choice of loss sparse_categorical_crossentropy", optimizer="rmsprop" are fully in line with the nature of our dataset. Various activation function-RELU and Softmax. In this model a dropout of 0.5 was applied at the end.

The testing accuracy **74.04**% is highest, with 200 embedding dimension(D) and 30,000 vocabulary (Val) as again 100D and 20,000(Val).

1 dimensional CNN | Conv1D-without using Pre-trained Embedding

Similarly, this one-dimensional convolution without the pretrained embedding, we use TensorFlow pure embedding. With similar features (activation, loss, optimizer etc). The testing accuracy **73.23%** is highest, with 200 embedding dimension(D) and 30,000 vocabulary (Val) as again 100D and 20,000(Val). Slightly less than the above model.

RNN-Bidirectional LSTM Model 50 Unit

Recurrent neural networks are a sort of neural network in which past time steps' outputs are used as inputs in the current time step. Recurrent neural networks with Long Short-Term Memory (LSTM) are one of the most intriguing forms of deep learning algorithms. Unlike other recurrent neural networks, the network's internal gates allow the model to be trained successfully using backpropagation through time and avoid the vanishing gradients problem.

LSTMs vary from multilayer Perceptron and convolutional neural networks in that they are specially developed to solve sequence prediction issues. Bidirectional LSTMs are a kind of LSTM that may be used to increase model performance in sequence classification issues.

Bidirectional Recurrent Neural Networks (RNNs) have a simple concept. It entails replicating the network's initial recurrent layer so that there are two layers side by side, then feeding the input

sequence as is to the first layer and a reversed duplicate of the input sequence to the second. On the input sequence, bidirectional LSTMs train two LSTMs instead of one. The first is based on the original input sequence, while the second is based on a reversed replica of the original input sequence.

In this model the bidirectional LSTM improves performance, with a similar hyperparameter like the first model. We use the pretrained embedding and loss='sparse_categorical_crossentropy', optimizer='adam',50 units of LSTM neutron and softmax activation.

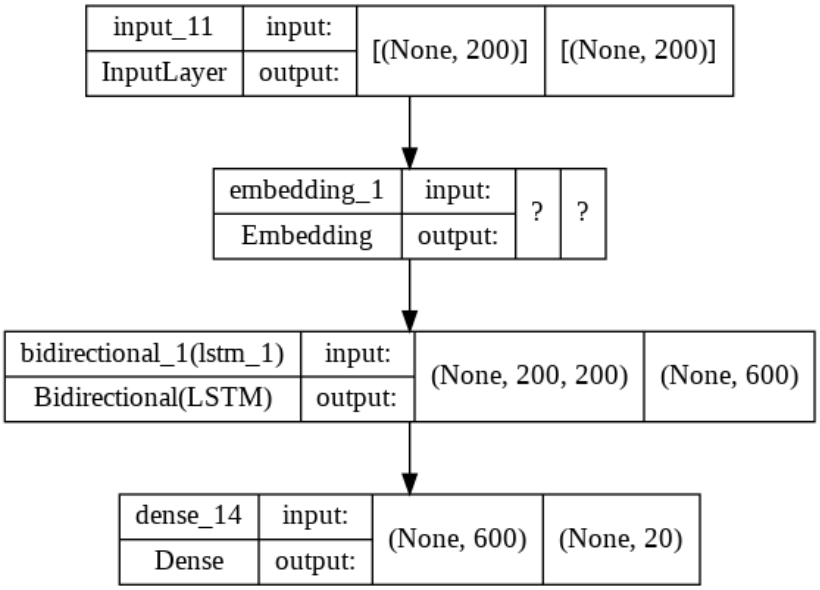


Figure 16 Architecture of RNN Bidirectional LSTM model

Initial results yielded a test accuracy of 75.81%. This outcome is improved to 79.02% when optimised with the following hyperparameter tuning; fewer than 20 epochs, a batch size of 30 and most importantly the number of units [50,500]. This result is the best we could achieve with up to 300 units, as the result declines after 300 units. We conclude this to be a very good result and is expected as Bidirectional LSTM is known to improve performance.

Results and Error analysis

A model with an accuracy score of above 70% and above can be interpreted as a good model and those with a score lower than this are likely to be flawed and have a high rate of mis-categorisation. However, a model which shows 100% accuracy is also not optimal as this would suggest overfitting.

Accuracy alone should not be relied upon, as it may not truly represent the effectiveness of the model. Precision, recall, and F1 score are also other metrics we can be used to evaluate machine learning models.

Table 4 Evaluation metrics by model

Model	Test Accuracy	Precision	Recall	F1 Score
Naïve Bayes	0.3836	0.4721	0.3749	0.3837
KNN	0.4715	0.4759	0.4648	0.4676
ANN	0.5653	0.5384	0.5501	0.5366

CNN 1D	0.7323	0.7439	0.7276	0.7302
CNN MaxPooling	0.7404	0.7539	0.7296	0.7342
CNN 1D (Keras)	0.7608	0.7776	0.75586	0.7611
RNN-BiD-LSTM	0.7581	0.7575	0.7527	0.7525
RNN-BiD-LSTM (optimised)	0.7902	0.7923	0.7845	0.7854

The non-neural network models, Naïve Bayes and KNN performed poorly with low accuracy scores of 38% and 47% respectively (Table 4). The most basic neural network model faired somewhat better with an accuracy score of 56%. This was bettered by the CNN 1D which had an accuracy score of 73% and similar values for precision, recall and F1-score. Of the more advanced models, the RNN Bidirectional LSTM had the highest accuracy. Hyperparameter optimisation increased accuracy from 75% to over 79% resulting in a highly efficient model.

Learning Curves

The performance of machine learning models can be evaluated by using a diagnostic tool known as learning curves, which come in two forms, plotting training loss and validation loss by each epoch, and plotting the training and validation accuracy by each epoch. The training curve represents how well the model is 'learning' the training data whilst the validation curve (calculated from the split validation data set) represents the generalisation of the model. (Machine Learning Mastery, 2022)

For a deep-learning model where the training and validation curves diverge, this inflection point should identify the last epoch when tuning. Continuing training of the model past this point would lead to overfitting of the data.

The learning curves of the model with the highest accuracy (RNN Bidirectional LSTM) is shown in Figure 17. However, the lines are not closely aligned which suggests an unrepresentative training dataset.

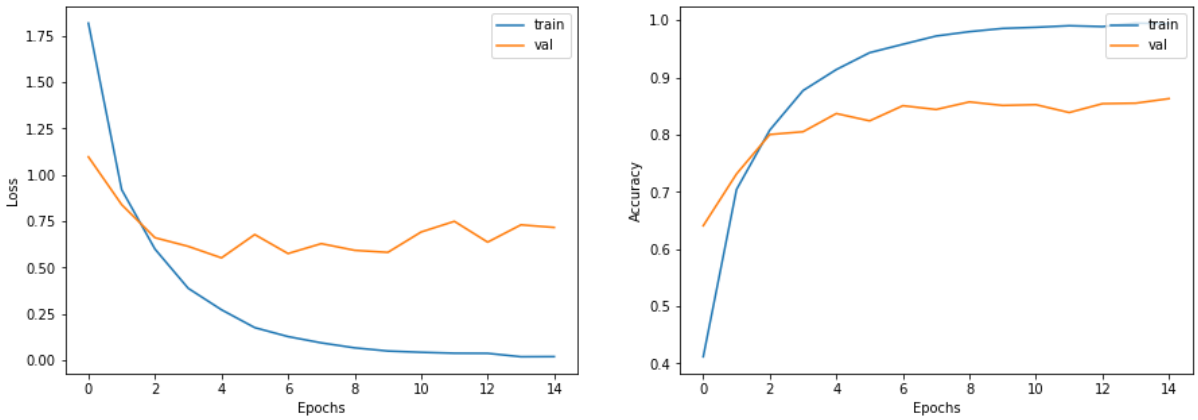


Figure 17 Learning curves of the RNN Bidirectional LSTM model

Bias and Variance

Machine learning algorithms have a calculable variance and bias from which we can evaluate the model and determine whether it overfits or underfits the data. A model with high bias could be interpreted as being too simple. This would result in a low variance which we interpret as underfitting the data. In contrast, model one with high variance and low bias would be interpreted as overfitting the data. Therefore, there is a trade-off between model complexity to reduce the effects of overfitting and underfitting (Huigol, 2020).

The following scatter plot (Figure 18) shows the relationships between bias and variance for each of the models. This analysis shows CNN and RNN models to be optimum in terms of the bias-variance trade-off.

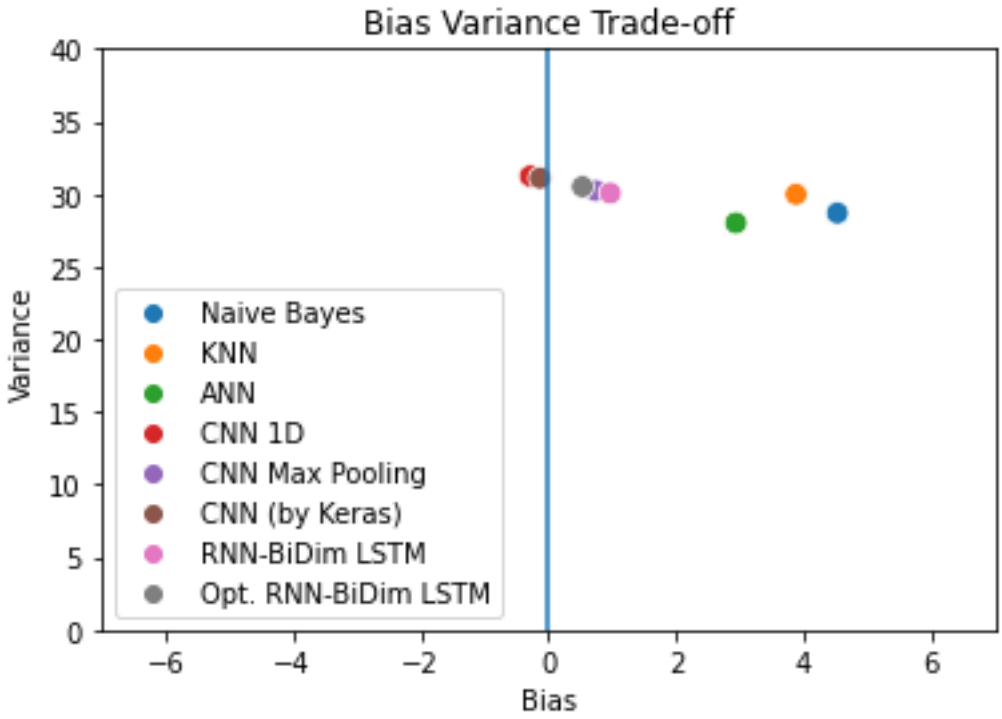


Figure 18 Bias-Variance trade-off comparison

Confusion matrix

Figure 19 is a confusion matrix of the Optimised RNN Bidirectional LSTM model revealing how well the model predicted each category compared to the 'ground truth' test data set. There is a strong diagonal line (indicated in yellows and greens), although there are some categories that are commonly misidentified (or 'confused') as others. For example, in our best performing model alt.atheism was mis-categorised as talk.religion.misc a total of 53 times. Using our qualitative knowledge, we know these topics are broadly like each other, and would not be surprising for them to be miscategorised by a human. Conversely, we can see that talk.politics.guns is miscategorised as sci.med a total of 16 times. We would assume these topics are very different from each other and so such a result is surprising and could suggest a weakness with the model.

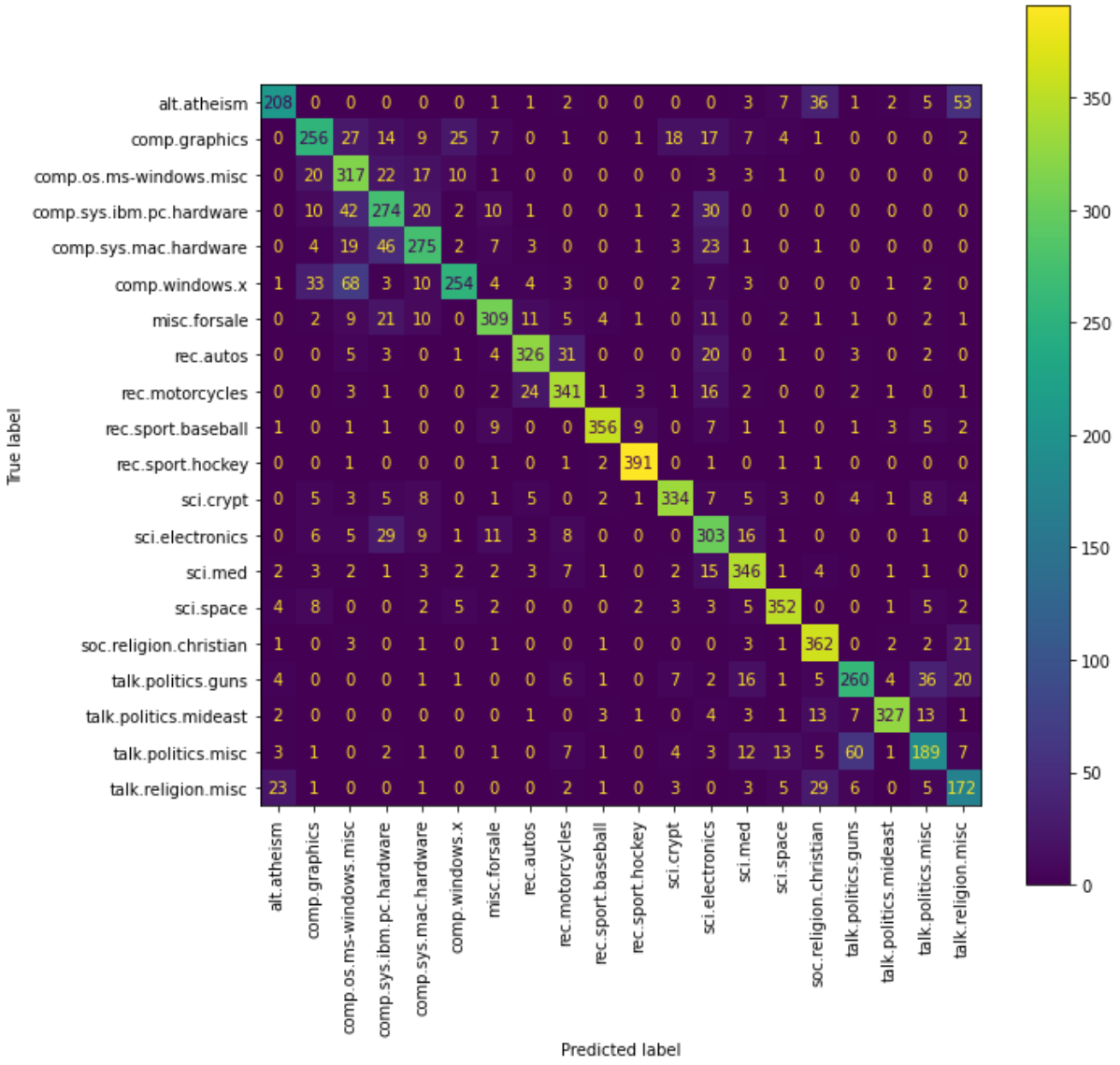


Figure 19 Confusion matrix of the optimised RNN Bidirectional LSTM model

Conclusion

Using different machine learning algorithms resulted in a wide range of effectiveness. We found the RNN Bidirectional LSTM to be the best model with an accuracy of 79%. However, this model is not without its flaws as the learning curves suggest issues with the representativeness of the training data.

There is still an opportunity to improve upon these models, for example, increased hyperparameter optimisation could lead to improved results. Whilst we explored bigrams in the data analysis stage, we did not incorporate these features into our models explicitly, partially due to time constraints. The final model also did not explicitly incorporate the POS-tagging explored during data analysis. Both these features could be one area on which to focus more research on the future. We could have also explored other embedding and vectorising methods.

In addition, the field of machine learning and deep learning is evolving quickly resulting in the development of new powerful highly effective algorithms which should be fully explored.

References

Introduction

Lewis, D et al. 1994. A Comparison of two learning algorithms for text categorization. *Symposium on Document Analysis and IR, ISRI, Las Vegas.*

Literature Review:

Pavan, V. 2021. Naive Bayes Explained: Function, Advantages & Disadvantages, Applications in 2022 [Online] Available at: https://www.upgrad.com/blog/naive-bayes-explained/

Dogru, Hasibe & Tilki, Sahra & Jamil, Akhtar & Hameed, Alaa. (2021). Deep Learning-Based Classification of News Texts Using Doc2Vec Model. 91-96. 10.1109/CAIDA51941.2021.9425290

Ria, K. 2019. NLP 101: Word2Vec - skip-gram and CBOW [Online] Available at: https://towardsdatascience.com/nlp-101-word2vec-skip-gram-and-cbow-93512ee24314

Ria, K. 2019. NLP 102: Negative Sampling and GloVe [Online] Available at: https://towardsdatascience.com/nlp-101-negative-sampling-and-glove-936c88f3bc68

C. Li, G. Zhan, and Z. Li, "News Text Classification Based on Improved Bi-LSTM-CNN," 2018 9th International Conference on Information Technology in Medicine and Education (ITME), 2018, pp. 890-893, doi: 10.1109/ITME.2018.00199

Cezanne, C. 2017. CNNs for text classification [Online] Available at: https://cezannec.github.io/CNN_Text_Classification/

Kim, Y. 2014. Convolutional Neural Networks for Sentence Classification [Online] Available at: https://www.connectedpapers.com/main/1f6ba0782862ec12a5ec6d7fb608523d55b0c6ba/graph
A. Bhavani and B. Santhosh Kumar, "A Review of State Art of Text Classification Algorithms," 2021 5th International Conference on Computing Methodologies and Communication (ICCMC), 2021, pp. 1484-1490, doi: 10.1109/ICCMC51019.2021.9418262

Pre-processing

https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908

https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/

https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

https://towardsdatascience.com/nlp-embedding-techniques-51b7e6ec9f92

Data Analysis

NLTK. 2022. Documentation. Sample usage for collections. Available at: https://www.nltk.org/howto/collocations.html [Accessed 09/04/2022]

Yang, S. 2020. Text analysis basics in Python. Available at https://towardsdatascience.com/text-analysis-basics-in-python-443282942ec5 [Accessed 10/04/2022]

Implementation – One-hot encoding

https://www.analyticsvidhya.com/blog/2021/05/how-to-perform-one-hot-encoding-for-multi-categorical-variables/

https://medium.com/@morga046/multi-class-text-classification-with-doc2vec-and-t-sne-a-full-tutorial-55eb24fc40d

The Keras Blog. 2016. Using pre-trained word embeddings in a Keras model. Available at: https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html [Accessed 16/04/22]

Results Analysis

Brownless, J. 2019. How to use Learning Curves to Diagnose Machine Learning Model Performance. Available at: https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/ [Accessed: 19 April 2022]

Huigol, P. 2020. Bias and Variance in Machine Learning. Available at: https://www.analyticsvidhya.com/blog/2020/08/bias-and-variance-tradeoff-machine-learning/ [Accessed: 19 April 2022]

APPENDIX

The data can be initially sourced using the following:

```
pip install wget
import wget
site_url = 'http://qwone.com/~jason/20Newsgroups/20news-bydate.tar.gz'
file_name = wget.download(site_url)
#print(file_name)
import tarfile
my_tar = tarfile.open(file_name)
my_tar.extractall(path) # specify which folder to extract to
my_tar.close()
```