COVID-19 Retweet Prediction

Adhi Narayan Tharun (1003379), Lee Jia Le (1003516),

Lin Huiqing (1003810), Keith Goh (1003334)

Singapore University of Technology & Design

I. Task Description

With the many changes brought about by the pandemic, how we as a society live and interact with one another has been drastically changed as well. Considering that many countries have imposed restrictions on movement as well as lockdowns, it is no surprise that people have taken to social media not only as a source of social interaction and entertainment, but also as a source of information. Along with the rapid changes in both the local and global situation, the rate at which pertinent information bombarded individuals has increased sharply. With the changes on these two fronts, the increased proliferation of false information has been observed.

The effects of false information propagating have been observed as well, as uninformed individuals might act based on them and harm themselves and/or others around them. For instance, due to the misinformation spread by the United State's former President Trump on chloroquine, an Arizona man consumed the chemical believing it would cure Covid-19, and subsequently passed on [9]. To prevent similar tragedies from occurring, early detection of false information can be developed.

Though early detection of false information is critical to correcting them before they cause harm, the large volume of information which streams into social media restricts the ability of social media platforms or authorities from screening them. A way to improve this situation would then be to predict the virality of social media content as they are created and then allocate resources to content which is more likely to be viewed by many. With better screening of content with more potential to spread, potentially viral false information would have a higher probability of early detection.

In the case of Twitter, which is a social media giant, the virality of social media content can be modelled by the number of retweets, as each retweet forwards messages to multiple users, possibly causing a chain reaction if enough people do it. To help predict the number of retweets a tweet will get, features can be gathered from the user's information and the content itself.

As such, this project aims to predict the number of retweets based on contextual information of the user and content so as to predict the content's virality.

II. Dataset

The TweetsCOV19 dataset used is a publicly available dataset with more than 20 million tweets which were made between October 2019 and April 2020. Other than providing the target variable to predict, which is the number of retweets, this dataset provides preprocessed data for usage, such as calculated sentiment and entity scores. However, it contains neither the full tweets nor user-identifying for further analysis due to privacy concerns [3]. This then restricts the ability of users like us to explore other analytical methods to glean data which might be more helpful towards the problem.

A. Data Collection

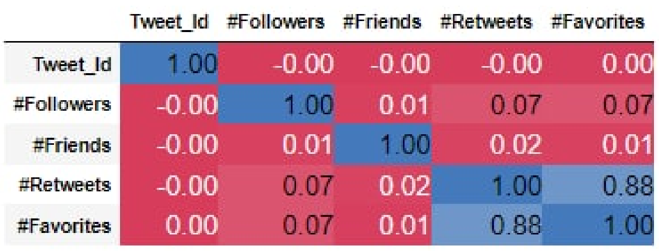
The dataset was available via the link https://data.gesis.org/tweetscov19/ which allowed us to download the 3 dataset timeframes. The data was available in 3 tsv files.

Initially the dataset was in a tsv file format, with each line containing features of a tweet instance. Features are separated by tab characters ("\t"). The following list indicates the feature indices.

1) **Tweet Id**: Long
   (Example: `1178791787386814465`)
2) **Username**: String. Encrypted for privacy issues
   (Example: `35234fe4a19cc1a3336095fb3780bcc1`)
3) **Timestamp**: Format ("EEE MMM dd HH:mm:ss Z yyyy")
   (Example: Mon Sep 30 22:00:37 +0000 2019)
4) **#Followers**: Integer
   (Example: 619)
5) **#Friends**: Integer
   (Example: 770)
6) **#Retweets**: Integer
   (Example: 589)
7) **#Favorites**: Integer
   (Example: 842)
8) **Entities**: String
   (Example:`moremagazine:More_%28magazine%29:-2.637330890370725;`)
9) **Sentiment**: String
   (Example: `2 -1`)
10) **Mentions:** String
   (Example: `NCTsmtown_127`)
11) **Hashtags**: String
   (Example: `NCT127`)
12) **URLs**: String
   (Example:`https://more.hpplus.jp/odekake/o-news/51119:-`:)

B. Data Exploration

To better understand the data, some initial data exploration was done.



	Tweet_Id	#Followers	#Friends	#Retweets	#Favorites
Tweet_Id	1.00	-0.00	-0.00	-0.00	0.00
#Followers	-0.00	1.00	0.01	0.07	0.07
#Friends	-0.00	0.01	1.00	0.02	0.01
#Retweets	-0.00	0.07	0.02	1.00	0.88
#Favorites	0.00	0.07	0.01	0.88	1.00

Fig. 1 Correlation between various attributes

An interesting finding as shown in Fig 1 was that there was a high correlation between the "#Retweets" column and "#Favourites" column. This was likely due to the fact that as the tweet gets retweeted more, more people are likely to favourite it.

Another insight was that the "Tweet_id" was unique to each tweet, and thus would not be helpful as a feature to predict retweets.

C. Data Preprocessing

1) Processing of Variables

For this phase, non-essential variables, such as tweet_id, were removed. This helps reduce the dimensionality of the whole model, reducing model complexity and reducing the chance of spurious correlations being drawn. Both of these aid with training. For categorical features, hash bins were employed. This was used for username, entities, hashtags, mentions and URLs. This serves as a way to encode these categorical features before they can be processed by various models. Another method which was considered was one-hot encoding, but was deemed unsuitable as it would drastically increase the dimensionality of the model and the model would not be able to account for new data in these columns. Hash bins were then chosen as a suitable middle-ground between complete distinction between the different categories and taking out these features.

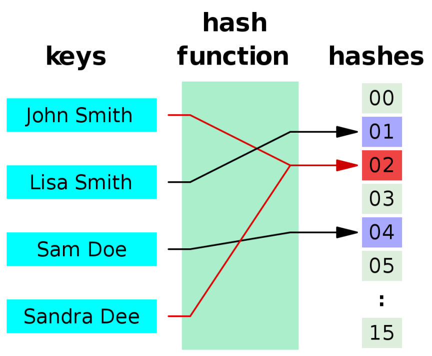


Fig. 2 Hashing Function[5]

Hash binning works by first hashing the variable, as seen in Fig 2, where some outputs will have the same output hash as others, and then binning those outputs. This will allow us to convert variables that have many different categories into ones that have less categories, by binning several categories together. This helps us with reducing the dimensionality of the columns. Since it is a deterministic approach, there is no need to remember the mappings for the different values when new values come in.

For the entities, the maximum, median and mean of the confidence scores were obtained and converted into separate features.

For the entities and URLs, as the median, mode and average of the number of items in these columns were 1. As such, only the entity with the highest confidence score was logged. Likewise for URLs, only the first URL was processed. These fields are then further processed as explained below.

For the entities, we took the entity with the highest score and converted them into embeddings, using the GloVe 200d embeddings from Stanford [11]. GloVe embeddings were chosen over other embedding methods, such as BERT, as these other embedding methods required the whole sentence to be inputted to give an accurate score for the entity, which we did not have as Fast Entity Linker (FEL) [15] had already been applied to the tweet. Hence, we settled on Glove embeddings in our case.

For URLs, we used regex expressions in order to extract the domain from the full URL as we felt that the domain would be enough to capture the logical semantics captured within this column, because the domain would be enough to reflect the credibility of the sources which the tweet was using. For instance, the domain capstone2021.sutd.edu.sg would be extracted from the full URL https://capstone2021.sutd.edu.sg/projects/n-able-a-navigational-wearable-for-the-blind-and-visually-impaired before hash binning. Thus, by having this column we can see if the url's credibility within the tweet would be useful in determining how useful they were.

For the hashtag and mentions columns, we determined that the mean, median and mode of the number of elements in those columns were 1 too, hence we decided to only take the first element in each list of hashtags and mentions and proceeded to hash bin them. This would reduce the dimensionality of the columns.

We stored the counts of the mentions, hashtags, URLs columns into the data. We also removed the rows that had URLs, hashtags or mentions with counts of more than 140, this is because the no. of such items are unlikely to be more than 140, due to the max characters in a tweet being 280, hence this is likely an error in processing, as we checked the data, which showed that the delimiters did not work as intended due to some quirks in the data.

As for timestamps, they had to be converted to meaningful features as well for the model to understand them. Timestamps are inherently cyclical, the time of the day repeats every 24 hours, and the months cycle every 12 months. As such, they should be encoded cyclical continuous features. Logically, the cyclical features of note would be the time of the day and month of the year, both of which may be related to volume of Twitter's user traffic depending on working schedules and seasons. Other cyclical features like the minute of the hour were not included as features as they are not expected to be significant, as according to the numerous digital marketing blogs [4], [6]. To encode the time of day and month of year as cyclical continuous features, the cyclical properties of the sine and cosine curves are utilised [8]. The year from the timestamp was left as an individual feature.

To summarise, Table I shows the conversion of data after pre-processing.

TABLE I
Conversion Of Data After Pre-processing

Pre-Processing Column	**After-Processing Column**	**No. of columns after processing**
Tweet id	Removed	0
Username	Hash binned into 256 columns	256
Followers	Remain	1
Friends	Remain	1
Favourites	Remain	1
Mentions	Took the first mention in each tweet and 64 hashed binned it + stored the count of the column	65
Hashtags	Took the first hashtag in each tweet and 64 hashed binned it + stored the count of the column	65
URLs	Took the first Url in each tweet and 256-hashed binned it after doing regex extraction + stored the count of the column	257
Entities	Took highest confidence and used Glove on it + stored highest confidence and mean and median of confidence	200+3
Sentiments	Split into two columns with positive and negative, and took the absolute	2
Timestamp	Converted into year + sin_second + cos_second + cos_month + cos_month	5
Retweets(target)	Remain	1
	Total	**857**

This is then the data which is used as input to the models explored.

2) *Processing for loading of data*

We randomly split this dataset into 3 parts using a 80:10:10 split into a train, validation and test set. The training set, which is used to train the model, is further split into a 80:20 split for cross-validation when training the model to check for overfitting. The validation set is used to tune the hyperparameters, for each of the 3 different models that we have developed while the test set is used for the comparison between different model architectures. This prevents data leakage across the different sets of data.

As the processed data is very large, having 19 million rows with 857 columns we have split them into smaller files with 100 rows each. This will allow the easier loading of the data.

To do so, we generated the list of files, which has 100 rows each, with the exception of the last file which has 63 rows. We then randomly split them into the respective sets, by using the list of files. To use the respective sets, we loaded the files within each list in batches, so as to not overload the computer with large volumes of data.

In order to have a fair comparison of the models, we have also used a seeded randomization so as to be able to achieve a matched stream data. This allows us to compare the different models without having to worry as much about data variance between the models.

For the training process, we generated a list of the files for each batch and processed all the files in each before proceeding on with the next. Using the 80:20 train-validation split, overfitting of models was checked. Each epoch in our training contained 7500 files, 6000 files for training and 1500 for validation.

For the validation and testing process, we then went through all the files in the validation and testing sets respectively.

III. Model Exploration

As the data we are working with is labelled, supervised learning algorithms were explored. Three types of algorithms were chosen, namely eXtreme Gradient Boosting (XGBoost), Linear Neural Networks, and Recurrent Neural Networks were chosen.

XGBoost was chosen due to its tendency to yield highly accurate results, especially in Kaggle competitions [2]. It is a decision-tree based ensemble method which was improved through various systems optimization and algorithmic enhancements [7], [14].

Linear neural network model was chosen because it is traditionally known to be good at forecasting. Since this problem requires the forecasting of retweets, we believed that this model would fit well. Given the large dataset of tweets and the various features for each tweet, the linear neural network will be able to model non-linear and complex relationships within each tweet. Linear neural networks don't impose any restrictions on the input variables, so it would be good for this problem since each tweet has a wide variety of features.

LSTM models were chosen as the tweets were timestamped and hence there may be an effect that previous tweets may have on the future tweets. This is true in the case of trending topics, where certain hashtags may gain traction due to the usage of the same hashtag, hence propelling further retweets of posts of similar nature. This may prove useful in predicting the number of tweets.

To evaluate between models, the mean squared log error (MSLE) was compared. MSLE was chosen as opposed to some other error like mean squared error due to the large range of values which is possible for the prediction target, which is the number of retweets. MSLE would then reflect the percentage difference between the true and predicted value [10]. Furthermore, MSLE penalises underestimates more than overestimates [10]. As the prediction target of retweets is meant to filter out relevant tweets for deeper analysis for fake news, this is in line with MSLE's properties as overestimates should not be penalised as much.

A. XGBoost

1) Architecture:

The XGBoost algorithm is a decision-tree based ensemble algorithm. The specific ensemble technique which it employs is the boosting algorithm.

A decision tree consists of nodes with conditions on model features. Based on the features of the particular sample, the predictions are then made based on the path which is taken. In an ensemble algorithm, many of these decision trees are trained, with each decision tree contributing to the final prediction. Specifically, XGBoost employs boosting which means that the prediction from each tree is summed to give a final prediction output.

Ideally, the model should be fitted with the whole dataset. However, due to resource constraints, this was not done. Instead, the model is first trained on a set of train samples for one epoch so as to create a set of trees. This is followed by training on other sets of train samples where no new trees are created and the existing trees are updated.

2) Feature Engineering:

Initially, the number of features after data preprocessing was 856, which is very large. As such, feature engineering was done to reduce this number.

To go about this, a default XGBoost model was first trained for 10 epochs on a random sample of 1,000 files from the train set. As each file has 100 training samples, this means that the model was trained on 10,000 random samples. After which, the most relevant features identified by the trained model were obtained.

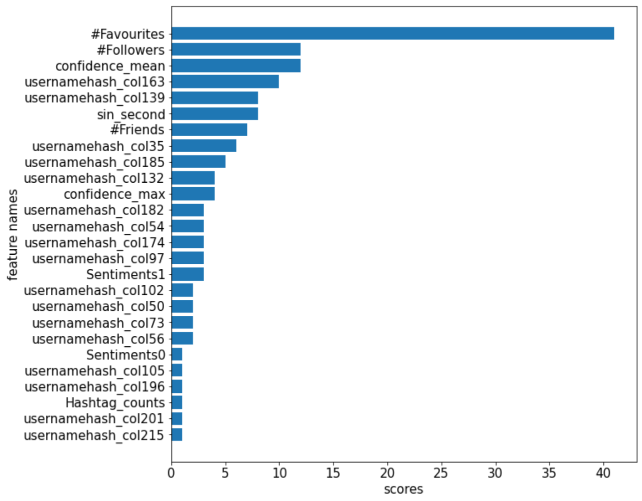


Fig. 3 Feature scores, ordered by scores in decreasing order.

Through this process, the most relevant feature was identified to be the number of favourites, followed by the number of followers tied with the confidence mean. Features related to mentions, URLs and entities were treated by the trained model as irrelevant.

Based on the feature scores as shown in Fig 3, the following features were kept:

- Number of favourites
- Number of followers
- Confidence score (including mean, max and media)
- Time (including sin_second and cos_second)
- Number of friends
- Hashtag counts
- Sentiments (including Sentiments0 and Sentiments1)
- Username (including all the username hash bins)

Note that though not all username hash features were identified as relevant based on Fig 3, the relatively large number of username hash features that show up mean that the username is relevant and the limited number of columns that show up might be due to the limited sample size trained on. Thus, all username hash bins were included as features.

Through this process of feature engineering, the number of features was reduced from 856 to 267. WIth the decrease in the number of features with respect to the training examples, the complexity of the model is decreased and overfitting would be less likely.

With the reduced number of features, a few experiments were conducted on the number of estimators and the presence of dropout.

For the following experiments, models at each epoch were saved, and early stopping was employed to train until it is certain that the model is overfitted. The models at the best performing epochs are then compared by their performance on the validation set.

3) Experiment 1 - Number of Estimators:

An important hyperparameter in the XGBoost algorithm is the number of estimators involved in the prediction. Theoretically, the greater the number of estimators, the better the predictions from the algorithm. However, if there are too many estimators, the model will reach a point of diminishing returns as the performance of the model increases minimally relative to the amount of resources used to train it.

As such, this experiment sought to explore the effect of the number of estimators on the model's performance. Due to the limited time and resources available to this project, the number of estimators experimented with are 2, 4, 6, 8 and 10.

Note that Fig. 4 excludes the MSLE of the first epoch as the MSLE is very large (>1000) for all models, and would cause the graph to lose much of its meaning in helping us to find the best trained model for each number of estimators used.

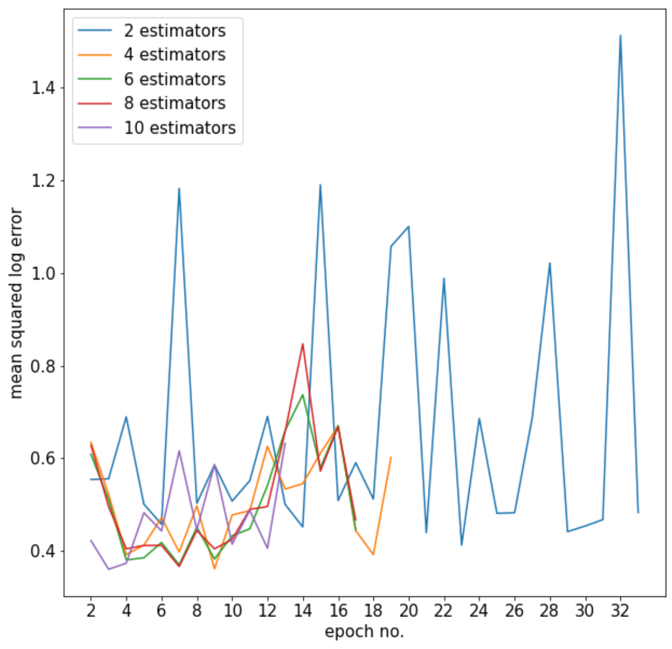


Fig. 4 Variation of MSLE on Evaluation Set with Epochs, for Each Number of Estimators

Based on the results as shown in Fig. 4, the models which are taken for the runs with 2, 4, 6, 8 and 10 estimators were the ones at epochs 23, 9, 7, 7 and 3 respectively.

The saved models at these epochs were then validated using the validation set. The resulting MSLE for each model is shown in Table II.

TABLE II
XGBOOST MODEL PERFORMANCE ON VALIDATION SET, WITH VARYING NUMBER OF ESTIMATORS

Number of Estimators	**MSLE**
2	0.4214
4	0.3680
6	0.3759
8	0.3667
10	**0.3570**

Based on Table II, the best model for this experiment is then the model with 10 estimators as it performs the best on the validation set at 0.3570 MSLE.

The results are largely in line with the theoretical hypothesis that an increase in the number of estimators will lead to an increase in performance as the model with 10 estimators performs better than all the other models which have lower numbers of estimators. Contrary to this hypothesis is that the model with 6 estimators performed worse than the one with 8 estimators.

Based on the general trend observed, the model would perform better with more estimators. Unfortunately, models with more estimators were not explored in this project due to time and resource constraints. In the industry, the number of estimators used usually ranges from hundreds to thousands [1].

4) Experiment 2 - Dropout:

Another hyperparameter which was adjusted was the presence of dropout. Dropout has a regularisation effect which prevents overfitting and helps the model learn from all features rather than relying on a few features exclusively.

The dropouts which were experimented in this project is the implementation as explained by Rashmi and Gilad-Bachrach known as DART, where whole trees are dropped [13]. With this technique, over-reliance on specific trees by the XGBoost algorithm will be prevented, allowing for better individual trees to be learnt.

Due to the time and resource constraints of this project, the comparison between models with and without the presence of dropout is done for models with only 2 estimators.

Note that Fig. 5 excludes the MSLE of the first epoch as it is very large (>1000) for all models, and would cause the graph to lose much of its meaning in helping us to find the best trained model for each hyperparameter.

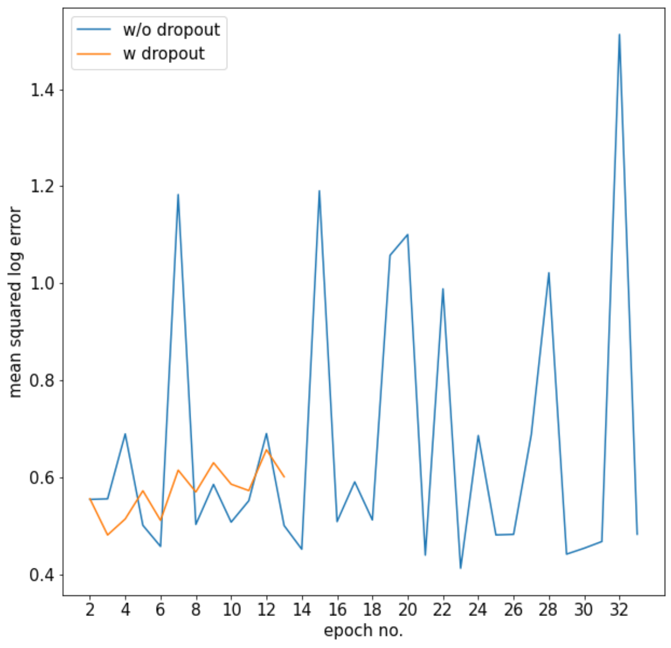


Fig. 5 Variation of MSLE on Evaluation Set with Epochs, with and without Dropout

Based on the results in Fig. 5, the epochs with the lowest MSLE would be 23 and 3 respectively.

The saved models at these epochs were then validated using the validation set. The resulting MSLE for each model is shown in Table III.

TABLE III
XGBOOST MODEL PERFORMANCE ON VALIDATION SET, WITH AND WITHOUT DROPOUT

Presence of Dropout	**Mean Squared Log Error**
No	**0.4214**
Yes	0.4923

Surprisingly, the model with dropout performed worse than the model without dropout in this case. This might be due to the low number of estimators used as there were only 2 estimators in total. This means that as DART drops trees in an attempt to learn better predictions, the algorithm would not be able to take advantage of XGBoost as an ensemble method during training.

As such, more experiments should be done with DART with more estimators to get a better understanding of how the implementation of DART affects model performance empirically. This was unfortunately not done in the course of this project due to limited time and resources.

5) *Conclusion:*

Based on the experiments done by varying the hyperparameters of the XGBoost algorithm, it is found that increasing the number of estimators lead to better performance, but the presence of dropout might not necessarily lead to better performance, especially in the case of a low number of estimators.

Among the models trained for XGBoost, the model which performed the best was the one with 10 estimators and epoch 3. The performance of this model was then measured against the test set, and the MSLE of the resulting predictions was 0.3564.

B. Linear Neural Network

1) Model Architecture

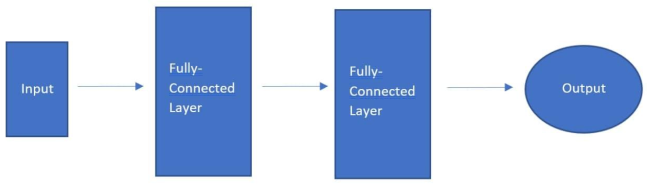


Fig. 6 Linear Neural Network Architecture

For the linear neural network, we used a feed-forward neural network made up of fully-connected hidden layers. We tried out different hyperparameters for the model.

For the input to the model, we took in each tweet in sequence, 856 features at a time and subsequently put them through the hidden fully connected layers.

2) *Crafting of Initial Linear Neural Network model*

For the initial model, we had 2 fully connected hidden layers with a final output prediction at the end. The linear neural network we used is unidirectional. This is the basic model that we started with.

3) *Early Stopping of model*

For the models, we used an early stopping mechanism to only keep models that have a validation loss lower than the previous models. This is to prevent overfitting of the models.

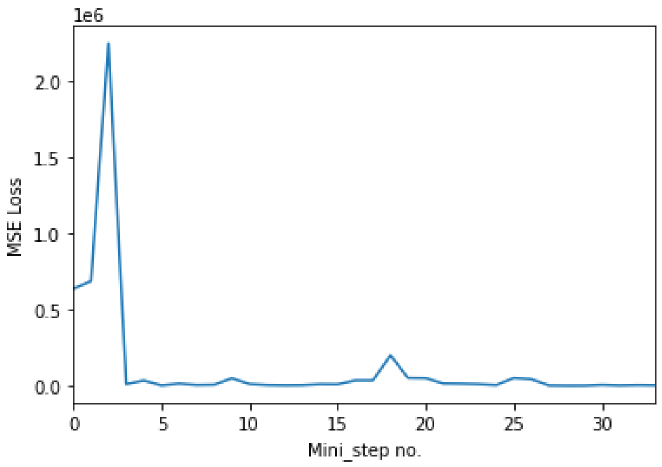


Fig. 7 Validation loss for Linear model

By using early stopping of the models, we can prevent overfitting. Though due to the batch training in our process, we will be unsure if it is an overfitted model or that due to the batches in the training data, there was a batch that was irregular as compared to the overall dataset, which would cause spikes in the data.

Thus, even though we had these early stopping measures, we also still continued to train the model and save the models once the training was done.

4) *Variation of models*

After the initial attempt, we realised that the model had overfitted really severely, so we decided to tweak the hyperparameters, such as dropout rate, the learning rate and the step size at which we will compare against a validation batch to see if our model has improved.

5) *Comparison between Linear Neural Network Models*

The dimensions of all the layers including input and output for all the models are 856 -> 256 -> 256 -> 1.

TABLE IV
NEURAL NETWORK MODEL PERFORMANCE ON VALIDATION SET

Model	**MSLE Score**
Learning Rate: 0.1, Dropout: 0.5, Step size: 50	7.0978
Learning Rate: 0.001, Dropout: 0.6, Step size: 10	2.3978
Learning Rate: 0.001, Dropout: 0.5, Step size: 25	**1.9196**

From the table, we are able to see that the model with step size 25 and learning rate 0.001 performed the best. The first model, overfitted really badly, due to a high learning rate and the other model with the lowest step size didn't perform as well because the performance check against the validation batch might have been too frequent.

C. LSTM model

1) Model Architecture

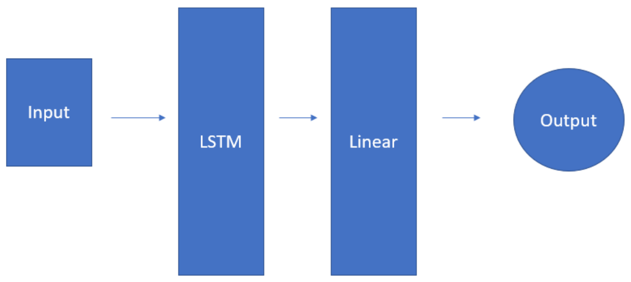


Fig. 7 LSTM Model Architecture

For the LSTM models we combined it with a linear model architecture. By varying the number of linear and lstm layers, we are able to tweak it such that it is a pure lstm model or one with linear layers. This allows for hyperparameter tuning later.

For the data used in the LSTM model, we did a windowing approach to load the data into the model. This allowed the model to train on the same data as the previous models, continuing the matched stream technique that we previously used. For the comparison between models, we would only be able to compare 71 rows for each file due to the windowing effect, whereas due to a window of 30, we will not be able to predict for the first 29 rows in the dataset. Hence for the comparisons that we will do with this model, we will only take the MSLE of 71 rows. In order to ease the comparison, we only used a window size of 30 when building the models.

2) Crafting of Initial LSTM model

For the initial model, we chose to go with a 2 layer model, where there are 2 LSTM layers, excluding the output layer. This would be the basic model that we are trying. The LSTM layers are bidirectional, using an empty hidden dimension when fitting the model. Learning rate for all models was set at 0.001, with an adam optimizer

3) Early Stopping of model

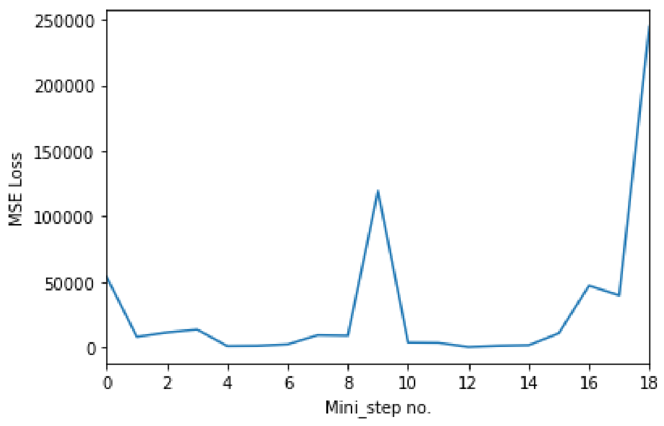


Fig 8. Validation loss for LSTM model

Similarly to the linear neural network model, we also included early stopping into the model due to prevention of overfitting which causes the MSE Loss of the validation set to shoot up as seen in the figure.

4) Variation of models

Tweaking the hyperparameters, we decided to increase the dimensions of the model and also increase the number of layers for comparison between the models. This allowed us to explore the model architecture model. Though we would have preferred to do a grid search cross-validation approach, the sheer size of the data coupled with the limited time and resources prevented us from doing so. Approaches to help overcome this hurdle will be further noted in the conclusion.

5) Comparison Between LSTM models

Using the validation set, we compared the three different models to choose which was the best model between the three.

Note:HD refers to Hidden dimension.

TABLE V
LSTM Model Performance on Validation Set

Model	**MSLE Score**
HD: 256, 2 layer LSTM	**1.8046**
HD: 256, 2 layer LSTM + 1 Layer Neural Network	1.8754
HD: 512, 2 layer LSTM + 1 Layer neural Network	2.4969

From the table, we are able to see that the basic model was the best. Though we would likely be able to get a better score by continually training the model, we had to stop in the interest of time.

While it is surprising that the more complex the model is the worse it performed, it was likely picking up spurious correlations within the data and hence overfitting the models.

6) Conclusion

The LSTM model certainly shows promise with respect to the Linear model and can certainly be explored further, with a grid search cross validation approach in order to find better hyperparameters. Though the results pale in comparison with the XGBoost model.

IV. Comparison Between models

Comparing the best models within each architecture with the rest of the models using the test set, the MSLE scores are reported below.

TABLE VI
Model Performance on Test Set

Model	**MSLE Score**
XGBoost with 10 estimators	**0.3564**
Linear Neural Network (Learning Rate = 0.001, Dropout: 0.5,Step size = 25)	1.9193

LSTM Hidden Dimension: 256, 2 layer	1.7984

With this comparison, the best model found during the course of this project would then be the XGBoost with 10 estimators, as it is able to achieve the best MSLE score on the test set. Given more time and resources, perhaps a different conclusion could be reached, as we recognise that there are some flaws in how the implementation was done, such as the lack of a learning rate scheduler and the limited number of estimators.

V. State-of-the-Art

To help identify state-of-the-art models for retweet prediction based on this dataset, competitions relating to this dataset were looked through. One of them is the CIKM AnalytiCup 2020's Covid-19 Retweet Prediction. From which, the first place was considered as the state-of-the-art model for this project.

A. Architecture

The approach taken by the state-of-the-art model was a deep learning approach, consisting of a mix of convolution neural networks, long short-term memory cells and dense layers, as seen in Fig 9.

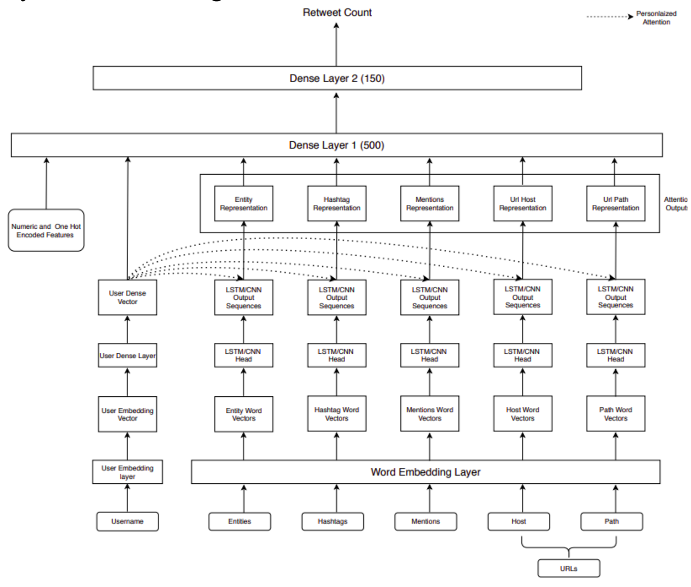


Fig. 9 State-of-the-Art Model Architecture [12]

It is structured with a personalised attention mechanism as shown in Fig 10 so as to get a better encoding of the strings which were included, such as the Entity, Hashtag, Mention and URL representations.

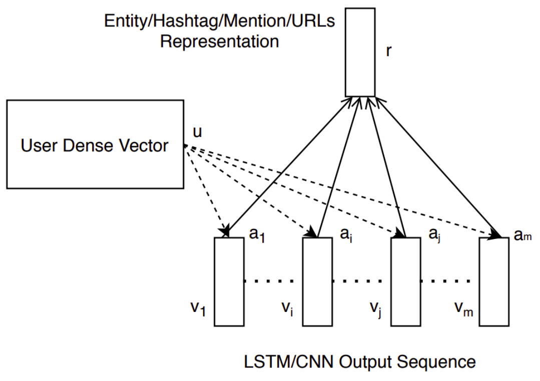


Fig. 10 Personalised Attention Mechanism [12]

With this architecture, they managed to get a MSLE score of 0.12860.

B. Comparison with Project Models

Unsurprisingly, the MSLE score of the state-of-the-art model was better than the models which were implemented in this project. Compared to the state-of-the-art implementation, the implementation for this project was much simpler.

In terms of strings data, the pre-processing done for this project was limited to hash binning, while the state-of-the-art implemented the personalised attention mechanism for the model to learn the representation for these strings in the context of the particular user. This approach is interesting, but too time and resource intensive to implement in this project.

The scale of the model implemented for the state-of-the-art was also much bigger than the one for this project. The state-of-the-art also made use of the whole train set to train the model, which is a luxury which could not be afforded in the context of this project with the limited time and resources.

However, perhaps the implementation of the state-of-the-art could benefit from how preprocessing was done for time in this project. In this project, time was processed such that the cyclical property of time in a day and months in a year was captured. This was neglected in the state-of-the-art as time was treated as linear, and thus a tweet sent at 0000 hours would be treated as 23 hours and 59 minutes away from a tweet published at 2359 hours, even though they might be posted 1 minute apart.

Though we would have liked to start the model with the same architecture as the state-of-the-art, our limited GPU resources caused us to create a model from scratch for this project, with techniques taught in class regarding the various model architectures.

VI. Project Setup

Below are the instructions to set up the code to run the GUI:

Running on Python Version 3.7,
git clone https://github.com/cre8tion/COVID19-Retweet-UI

On Windows,
pip install -r requirements.txt
set FLASK_APP=main
set FLASK_ENV=development
flask run

On Unix,

pip install -r requirements.txt
export FLASK_APP=main
export FLASK_ENV=development
flask run

The GUI is currently being hosted on https://covid19-retweet.as.r.appspot.com as well for ease of use.

As for code used in building the models and running the experiments, follow the instructions as specified in the READMEs in the following GitHub repository: https://github.com/silentfatez/covid19-retweet-prediction.

VII. Demonstration

For GUI created to explore models and data, the website is hosted live at https://covid19-retweet.as.r.appspot.com.

The website showcases our model in two parts, an Overview page and a Explore Data page.

The Overview page (Fig 11) shows the graphs we derived during model exploration for our various models (which can also be similarly found in this report). It aims to highlight some of our insights we gathered through the whole process, including the final comparison between models.

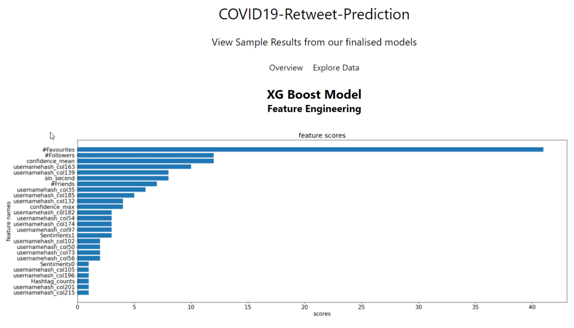
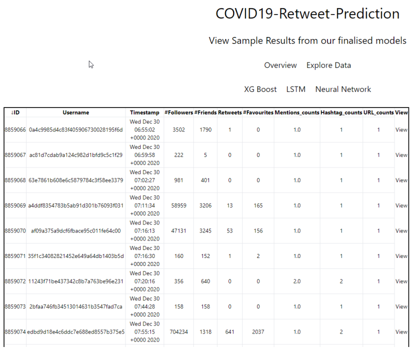


Fig. 11 Overview Page

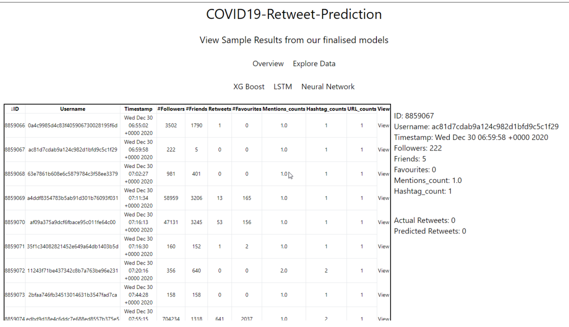
The Explore Data page (Fig 12 and 13) comprises mainly a table showing a small subset of the test data we selected for demonstration. Firstly, there are three different model types to choose when viewing the data, with XGBoost being the default model loaded. The table consists of various feature columns which are useful during the model training process, which can also be sorted in an ascending or descending order. Upon selection of a specific row of data, the page will also display the selected data separately, along with the predicted retweets generated by the selected model.



COVID19-Retweet-Prediction
View Sample Results from our finalised models
Overview
Explore Data
XG Boost
LSTM
Neural Network

↓ID	Username	Timestamp	#Followers	#Friends	Retweets	#Favourites	Mentions_counts	Hashtag_counts	URL_counts	View
8859066	0a4c9985d4c83f405906730028195f6d	Wed Dec 30 06:55:02 +0000 2020	3502	1790	1	0	1.0	1	1	View
8859067	ac81d7cdab9a124c982d1bfd9c5c1f29	Wed Dec 30 06:59:58 +0000 2020	222	5	0	0	1.0	1	1	View
8859068	63e7861b608e6c5879784c3f58ee3379	Wed Dec 30 07:02:27 +0000 2020	981	401	0	0	1.0	1	1	View
8859069	a4ddf8354783b5ab91d301b76093f031	Wed Dec 30 07:11:34 +0000 2020	58959	3206	13	165	1.0	1	1	View
8859070	af09a375a9dcf6fbace95c011fe64c00	Wed Dec 30 07:16:13 +0000 2020	47131	3245	53	156	1.0	1	1	View
8859071	35f1c34082821452e649a64db1403b5d	Wed Dec 30 07:16:30 +0000 2020	160	152	1	2	1.0	1	1	View
8859072	11243f71be437342c8b7a763be96e231	Wed Dec 30 07:20:16 +0000 2020	356	640	0	0	2.0	2	1	View
8859073	2bfaa746fb34513014631b3547fad7ca	Wed Dec 30 07:44:28 +0000 2020	158	158	0	0	1.0	1	1	View
8859074	edbd9d18e4c6ddc7e688ed8557b375e5	Wed Dec 30 07:55:15 +0000 2020	704234	1318	641	2037	1.0	2	1	View

Fig. 12 Explore Data Page



COVID19-Retweet-Prediction
View Sample Results from our finalised models
Overview
Explore Data
XG Boost
LSTM
Neural Network

↓ID	Username	Timestamp	#Followers	#Friends	Retweets	#Favourites	Mentions_counts	Hashtag_counts	URL_counts	View
8859066	0a4c9985d4c83f405906730028195f6d	Wed Dec 30 06:55:02 +0000 2020	3502	1790	1	0	1.0	1	1	View
8859067	ac81d7cdab9a124c982d1bfd9c5c1f29	Wed Dec 30 06:59:58 +0000 2020	222	5	0	0	1.0	1	1	View
8859068	63e7861b608e6c5879784c3f58ee3379	Wed Dec 30 07:02:27 +0000 2020	981	401	0	0	1.0	1	1	View
8859069	a4ddf8354783b5ab91d301b76093f031	Wed Dec 30 07:11:34 +0000 2020	58959	3206	13	165	1.0	1	1	View
8859070	af09a375a9dcf6fbace95c011fe64c00	Wed Dec 30 07:16:13 +0000 2020	47131	3245	53	156	1.0	1	1	View
8859071	35f1c34082821452e649a64db1403b5d	Wed Dec 30 07:16:30 +0000 2020	160	152	1	2	1.0	1	1	View
8859072	11243f71be437342c8b7a763be96e231	Wed Dec 30 07:20:16 +0000 2020	356	640	0	0	2.0	2	1	View
8859073	2bfaa746fb34513014631b3547fad7ca	Wed Dec 30 07:44:28 +0000 2020	158	158	0	0	1.0	1	1	View
8859074	edbd9d18e4c6ddc7e688ed8557b375e5	Wed Dec 30 07:55:15	704234	1318	641	2037	1.0	2	1	View

ID: 8859067
Username: ac81d7cdab9a124c982d1bfd9c5c1f29
Timestamp: Wed Dec 30 06:59:58 +0000 2020
Followers: 222
Friends: 5
Favourites: 0
Mentions_count: 1.0
Hashtag_count: 1

Actual Retweets: 0
Predicted Retweets: 0

Fig. 13 Explore individual data based on selected model

VIII. Conclusion

1) *XGBoost ease of approach*

Within the time and resource constraints of this project, it was found that XGBoost might be the best model for this problem, followed by a LSTM approach, then a Neural Networks. With more time and resources, the conclusion might change, especially considering how the state-of-the-art model identified which won the CIKM AnalytiCup 2020 was a deep learning algorithm, which is closer to the Linear Neural Networks and LSTM explored in this project.

2) *Hash Binning downfalls*

Regarding data preprocessing, hash binning may not be optimal in some cases as there would not be an even distribution of hashes and the binning of hashes together are completely arbitrary, which may cause some categories, that would have negative implication of binning together, due to collision, such as a accredited news sites and non accredited news site to be in the same bin, which is bad for predictions. A likely method, if we had more time to process the data would be to use an unsupervised learning algorithm to help compress the data, using clustering algorithms in conjunction with retweets to help cluster the tweets. This would help with the dimensional reduction and also provide a better way to help group the features.

3) *Learning Rate Optimisation*

The learning rate of the neural network models could also have been optimised better with an optimiser scheduler, however due to the time constraints, we could not go back to retrain our models.

4) *Downsizing Data*

Another point that we would like to take note of was that in hindsight, we could have trained our models on a small subset, maybe only 10% of the data and use that as our train, validation and test set before retraining it again for deployment. This would have saved us a considerable amount of time and would allow us to tweak our model parameters even more. While we noticed this halfway through the processing of the data, we did not have time to pivot to this method.

5) *Data Exploration*

More time could have also been spent on data exploration such that we would not have that many dimensions to train, and also the normalizing of the columns for easier training of the neural network models. This was all techniques that in hindsight, we should have employed to help with the training of the models.

6) Leakage in Real world

Another point to note is that there is data leakage in the learning of the algorithms as the number of favourites of the tweet is included in the features. The number of favourites is a number of users who favorited a tweet, which means that it is a number which is affected by the prediction target, which is the number of retweets. This is because as the number of retweets increases, the probability of the tweet being favorited increases as more users see it. This direct relationship is captured in both Fig 1 and 3, where the number of favourites is found to be much more correlated with the number of retweets compared to other features. Thus, though this project has included the number of favourites as a feature, just as the state-of-the-art implementation did, algorithms should not include such a feature if they are intended to be implemented in the real world.

References

[1] Brownlee, Jason. "How to Tune the Number and Size of Decision Trees with XGBoost in Python." Machine Learning Mastery, 27 Aug. 2020, machinelearningmastery.com/tune-number-size-decision-trees-xgboost-python/.

[2] Chen, Tianqi, and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 13 Aug. 2016, doi:10.1145/2939672.2939785. Accessed 7 Aug. 2021.

[3] Dimitrov, Dimitar, et al. "TweetsCOV19 - A Knowledge Base of Semantically Annotated Tweets about the COVID-19 Pandemic." Proceedings of the 29th ACM International Conference on Information & Knowledge Management, 19 Oct. 2020, doi:10.1145/3340531.3412765.

[4] Edmondson, Brian. "The Best Times to Post on Twitter for Maximum Effectiveness." The Balance Small Business, 18 Mar. 2021, www.thebalancesmb.com/best-time-post-twitter-2531471.

[5] "File:Hash Table 4 1 1 0 0 1 0 LL.svg." Wikimedia Commons, commons.wikimedia.org/wiki/File:Hash_table_4_1_1_0_0_1_0_LL.svg.

[6] Gotter, Ana. "The Best Time to Post on Twitter in 2021, According to Experts." AdEspresso, 12 July 2021, adespresso.com/blog/best-time-to-post-on-twitter/.

[7] "Why Does XGBoost Work so Well?: Data Science and Machine Learning." Edited by Jessica Li, Kaggle, Dec. 2020, www.kaggle.com/general/196541.

[8] London, Ian. "Encoding Cyclical Continuous Features - 24-Hour Time." Ian London's Blog, 31 July 2016, ianlondon.github.io/blog/encoding-cyclical-features-24hour-time/.

[9] McLaughlin, Eliott C., et al. "Fearing Coronavirus, Arizona Man Dies after Taking a Form of Chloroquine Used in Aquariums." CNN, Cable News Network, 25 Mar. 2020, edition.cnn.com/2020/03/23/health/arizona-coronavirus-chloroquine-death/index.html.

[10] Peltarion. "Mean Squared Logarithmic Error (MSLE)." Peltarion, peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-logarithmic-error-(msle).

[11] Pennington, Jeffrey. GloVe: Global Vectors for Word Representation, nlp.stanford.edu/projects/glove/.

[12] Raj, T Vinayaka. "CIKM AnalytiCup 2020: COVID-19 Retweet Prediction with Personalized Attention." CEUR Workshop Proceedings, vol. 2881, ceur-ws.org/Vol-2881/paper1.pdf. Accessed 9 Aug. 2021.

[13] Rashmi, Korlakai Vinayak, and Ran Gilad-Bachrach. "DART: Dropouts Meet Multiple Additive Regression Trees." Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, vol. 38, 9 May 2015, pp. 489–497. Proceedings of Machine Learning Research, proceedings.mlr.press/v38/korlakaivinayak15.html.

[14] Reinstein, Ilan, and KDnuggets. "XGBoost, a Top Machine Learning Method on Kaggle, Explained." KDnuggets, Oct. 2017, www.kdnuggets.com/2017/10/xgboost-top-machine-learning-method-kaggle-explained.html.

[15] Yahoo. "Yahoo/FEL: Fast Entity Linker Toolkit for Training Models to Link Entities to KnowledgeBase (Wikipedia) in Documents and Queries." GitHub, github.com/yahoo/FEL.