Word and Graph Embeddings for COVID-19 Retweet Prediction

Tam T. Nguyen, Karamjit Singh, Sangam Verma, Hardik Wadhwa, Siddharth Vimal, Lalasa Dheekollu, Sheng Jie Lui, Divyansh Gupta, Dong Yang Jin, Zha Wei*

ABSTRACT

In this paper, we present our solution for COVID-19 retweet prediction challenge. The proposed approach consists of feature engineering and modeling. For feature engineering, we leverage both hand-crafted and unsupervised learning features. As the provided data set is large, we implement auto-encoding algorithms to reduce feature dimension. To develop predictive models, we utilize ensemble learning and deep learning algorithms. We then combine these models to generate the final blended model. Moreover, to stabilize the predictions, we also apply bagging as well as down-sampling techniques to remove the tweets where number of retweets equals to zero. Our solution is ranked ***First*** on the public test set and ***Second*** on the private test set.

1 INTRODUCTION

Machine learning and Text mining have been proved to be a powerful tools for processing unstructured text data and making short-term prediction, whether they are still helpful during the time of crisis, e.g. the ongoing Coronavirus disease 2019 (COVID-19), is still in question. In this paper, we take advantage of both text mining and machine learning methods to build more accurate models to predict number of retweets in Twitter. Our proposed approach consists of two major parts: feature engineering and modelling. For feature engineering, we leverage hand-crafted features and auto feature learning based on neural networks models. We use most of state-of-the-art algorithms in text mining and machine learning. For example, we rely on word2vec [9], doc2vec [9], and BERT [4] to learn embedding features. For graph features, we utilize node2vec and pyTorch-biggraph [6] algorithms to extract features. And finally, we build predictive models using emerging boosting algorithms such as Catboost [1] and LightGBM [5]. Our solution is released as an open source on Github[1].

2 DESCRIPTIVE ANALYSIS

The provided TweetsCOV19 dataset [3] contains extracted tweet instances which can be broadly divided in three types. 1) Metadata: tweet ID, username, and timestamp, 2) Connection: number of followers, number of friends, and number of favorites, and 3) Tweet text based: entities, sentiment, mentions, hashtags, and URLs. The goal is to predict number of retweets for each tweet. The train, validation, and test datasets are sequentially provided with respect to time. The training data accounts for the tweet instances from October, 2019 to April, 2020.

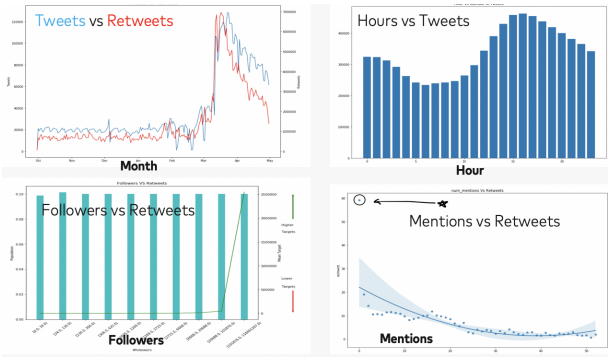

Figure 1: Retweets per Tweet against Number of Mentions

There are approximately 3.6M unique users in the train data and 300k common users between train and test data. On an average, the total number of tweets and retweets remain constant in the period of October, 2019 to February, 2020 as shown in Figure 1(a) and a sharp increment is seen in March, 2020. It is also observed that the number of tweets are higher in certain hours of the day in Figure 1(b), which might be due to data distribution of day and night, depending on the geographical location. We also observe the there is a negative correlation between number of mentions and number of retweets per tweet as shown in Figure 1(d).

While exploring the number of mentions with retweets per tweet, there are two key observations as shown in Figure 1(d). First, for zero mentions, the averaged number of retweets per tweets is the highest. This might be due to the fact that influential people use less mentions and get retweeted way more than the average. Second, as the number of mentions increase, there is a downward trend in retweets per tweet.

3 FEATURE ENGINEERING

Our feature set consists of both hand-crafted features and unsupervised learning features.

3.1 User-level Feature

We aggregate data by users to generate user-level features. For example, we group data by users and concatenate all tweets belonging to the users. We then generate features for them as follows.

3.1.1 *Hand-crafted user feature.*

- Followers and friends ratio. Along with the number of followers and friends, we derive the ratio of followers/(friends+1).
- Tweet count. Number of tweets for each user is calculated.
- Statistical features. Since the followers, friends and favourites vary for a given user over time for each user, various statistics like sum, mean, median, min, max, standard deviation are used to aggregate these columns separately for a user.
- Time based features. The number of tweets made by a user in a particular month and hour, number of different hours

*Corresponding author: Tam T. Nguyen E-mail: nthanhtam@gmail.com
[1]https://github.com/nthanhtam/cikm-cup-2020.git

In: Dimitar Dimitrov, Xiaofei Zhu (eds.): Proceedings of the CIKM AnalytiCup 2020, 22 October, 2020, Gawlay (Virtual Event), Ireland, 2020, published at http://ceur-ws.org.

in which a user tweeted, the average time between two consecutive tweets were used as features. Also the total number of unique users who tweeted on a particular date and hour are also calculated.
- User sentiments. For each user, the mean and standard deviation are calculated for both positive and negative sentiments. Also, the number of tweets for each value of positive and negative sentiments for a user are calculated.

3.1.2 ***Hashtags/Mentions****.* There are two kind of hashtags/mentions features in our models.

Hand-crafted Feature. We calculate count features from hashtags and mentions as follows.

- Number of unique hashtags and mentions used by a user are given as input for each tweet of that user.
- Number of tweets that hashtags or mentions occur in.

Word2Vec Features. There are only a few mentions in a tweet in the given data set. Most of the time, users don't mention anything in their tweets. If we directly use mentions as features (e.g. one-hot encoding), the feature space will be very sparse. Therefore, we propose to learn dense embedding features for mentions by considering each mention as word and train a word2vec model of 64 dimension using Gensim [9].

3.1.3 ***Entity*** *Embedding Feature.* In this section, we introduce our approach on how to extract embedding features from entities as follows.

- Pre-process entity triples and keep matched entities only.
- Aggregate entities by concatenating them based on user
- Train different embedding models with various parameters such as embedding size and vocabulary size.
- Extract embedding features for each user.

For embedding models, we try word2vec, doc2vec, and BERT models. In the following sections, we present how we train these model in detail.

Doc2Vec. In order to generate doc2vec features [9], we treat all entities of a user as a document. We then train doc2vec model using different embedding sizes such as 32, 64, 128, and 256. We then extract features and train our predictive models. Based on our experimental results on our internal validation set, we choose the embedding size of 64 as it gives us the best score.

Bert. Similar to doc2vec model, we aggregate entities by users and treat these as documents. We choose the top attention layer of BERT model to extract features. As this layer has dimensions of 1024, we reduce its dimension using:

- Truncated SVD [8] with 55 components.
- Autoencoder: We train a enoder-decoder neural networks with input and output size as 1024. We try different size of hidden layer choose the size of 128 which gives us good performance on both auto-encoding and predictive models.

3.1.4 ***URL*** *Feature.*

- Number of unique URLs used by a user
- Number of legit URLS in the tweet.
- URL field is fitted and transformed on a Count Vectorizer [8]. This is given as input to a SVD (Singular Value Decomposition) with 5 principal components to obtain 5 URL_SVD features.
- Domain name and suffix(url part followed by the domain name) are extracted for each URL in a given tweet and following are created:
  (1) A flag that indicates the presence of 'Twitter' word in the domain name.
  (2) For each domain name, number of tweets with that domain name are recorded. If a tweet has multiple URLs, then statistics like min, max, average are computed on domain frequencies for each tweet.
- Same features are created for suffix also.

3.1.5 Sentiment Feature.

- For each tweet, the product of the number of followers and the sentiment values, both positive and negative were taken. For the negative sentiment values the absolute of the sentiment value was taken.
- Similarly, the product of the number of friends and the sentiment values was also used as a feature.

3.2 Tweet-level Feature

For tweet-level features, we generate the features using label encoding for user ID, Hashtag, Mention and URL. We also use some features given in the raw data directly like favourites, friends, followers etc. Note that we only use label encoded features directly in tree-based/boosting models. For neural network models, we use embedding layer on top of these.

3.3 Trend-level Feature

Text based attributes of tweets such as hashtags $\{h_i\}$, mentions $\{m_i\}$, and entities $\{e_i\}$ can have a time based trend, also for each tweet there can be multiple values of the above. Since these attributes can have a time based trend, we create features to capture those trends. For the explanation purpose, we will use $\{a_i\}$ for $\{h_i\}$, $\{m_i\}$, and $\{e_i\}$. First we created features for each $\{a_i\}$ and since one tweet can have multiple $\{a_i\}$, we aggregate these features to a tweet level.

- Score: Number of tweets which have used a_i in that day
- Age: Number of days since a_i came into existence.
- Life: Number of days a_i has been active through the course of the data.
- Trend: Change in the score of a_i from the previous day.
- Peak: Fraction of tweets of a_i this day.

Below are the aggregate tweet level features:

- Score based features: we take average and max of the score of all a_i in a tweet.
- Weighted score features: Instead of normal average and max of scores, we take weighted average where two type of weights are used: inverse of 'Age' and 'Life' of a_i
- Age and life based features: we take average, min and max of 'Age' of all a_i in a tweet attribute. Similar features are created for life.

- Trend: we take average trend, max trend of all a_i in a tweet attribute.
- Peak: we take average and max of peak values of all a_i in a tweet.

3.4 Graph-based Feature

In this section, we introduce our graph-based feature set. In order to generate the features, we build a few kind of graphs such as user-entity graph (UE), user-hashtag graph (UH), and combined user-hashtag-URL-mention graph (CG).

3.4.1 UE. To generate graph feature, we build a user-entity graph as follows: (i) each node represents a user or an entity and (ii) create an edge between a user and an entity if user tweets about that entity. We use pyTorch-biggraph [6] to generate user level embedding of the size 64, which we use as features in our models.

3.4.2 UH. Similar to user-entity graph, we build a user-hashtag graph as follows: (i) each node represents a user or a hashtag and (ii) create an edge between a user and a hashtag if the user uses that hashtag in his tweet. Similar to UE, we also use pyTorch-biggraph [6] to generate user level embedding of size 64 and use it as features.

3.4.3 CG. This graph is similar to the above two graphs but we replace entities and hashtags by mentions and URLs. The edges of the graph is identified whether users use mentions or URLs in their tweets.

Additional to embedding graph features, we use traditional graph algorithms to extract information of users from the graphs such as centrality, average neighbor degrees, etc. Moreover, we also train node2vec [7] models to extract embedding features.

4 MODELING APPROACH

We use hold-out testing technique to validate our models where 80% of the data is for training and 20% is for testing. Our final solution is an ensemble of 3 major popular machine learning algorithms: Catboost [1], LightGBM [5], and Neural Networks using Keras [2].

4.1 Boosting Models

We use Catboost (CAT) [1] and LightGBM (LGB) [5] to train two separate models on features discussed in Section 3. For LightGBM algorithm, we also use bagging technique for LGB by repeating training it using various random seed numbers. Doing so, we can stablize its predictions in the validation and test sets. We will discuss further on these algorithms' parameters in the next section.

4.2 Neural Network Models

4.2.1 Convolutional Neural Networks (CNN). Figure 2 shows the architecture of the CNN and feed forward based model. Input to the model are all numerical features discussed above, embeddings for entity, hashtag, and mentions. We also graph based features derived from user-hashtag graph as described in Section 3.4.2. 1D convolutional layers are used to extract features from local input patches allowing for representation modularity and data efficiency. We use 64 filters for each variable followed by max pooling. We then concatenate the output of these CNN Layers. Simultaneously,

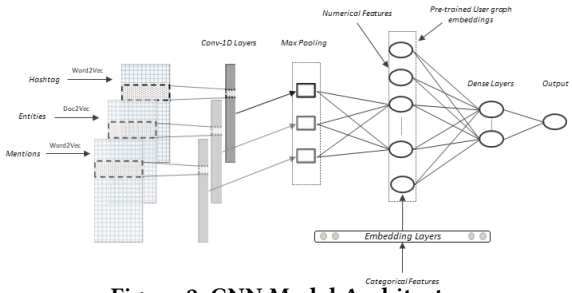

Figure 2: CNN Model Architecture

we pass categorical features like hour, weekday, month, user id, hashtag, and entity to the embedding layer. We concatenate all of these layers together and pass them to a feed forward neural networks and obtain the final retweet count.

4.2.2 Embedding Neural Networks Model. To develop embedding neural networks model, we classify features into two categories: numerical and categorical features. For numerical features such as no. of friends, no. of followers, etc., we use log scale to transform the data. For categorical features such as mentions, hashtags, and entities, we use embedding layer to deal with them.

4.3 Common-User Model

As discussed in Section 2, there are about 50% common users in train and test sets. This serves as the motivation to train a model specific to the common users only. Apart from the features discussed in Section 3, we create some additional features specific to common user model. We used catboost to train the model

K-Fold Target Encoding. Target encoding for each user is created to capture the popularity of a user, but a model using traditional target encoding (taking mean of the retweets for all the tweets of a user) tends to overfit the training data set and did not perform well. To tackle this problem, we use 5-Fold target encoding.

Moving Averages. To capture the trend of a particular user whether be the gain or loss in popularity over time, we create trend based features. For a user on a given day we find the moving averages of number of followers and friends for the last 3, 5, and 10 tweets. This is then used directly as a paramter in the model.

4.4 K-fold Down Sampling

We notice that most of tweets have no retweet count. If we use the whole training data to train models, the prediction in the validation and test sets tend to be smaller than the actual number of retweets. Hence, we use k-fold down sampling these tweets as follows:

- Split the training data into two sets $\mathbb{A}$ (retweets > 0) and $\mathbb{B}$ (retweets = 0).
- Apply k-fold cross validation on set $\mathbb{B}$ to select 80% of the data to have a new data set $\mathbb{C}$. Merge sets $\mathbb{A}$ and $\mathbb{C}$ to train predictive models and make prediction on validation and test sets. In this case, we have total 5 predictions , corresponding to each of the fold.
- The final prediction of validation and test sets will be the averaged prediction of k folds.

Table 1: Performance Comparison of various Strategies

Models	CAT		LGB		NN		CNN	
	val	test	val	test	val	test	val	test
base	0.12344	0.12433	0.13582	-	0.15693	-	0.12763	0.13139
base+common-model	0.12286	0.12374	-	-	-	-	-	-
base+kfold	0.12279	0.12366	0.13127	-	-	-	0.12649	0.13018
base+kfold+common-model	**0.12164**	**0.12285**	-	-	-	-	-	-

4.5 Ensemble

In practice, combining multiple machine learning models will help improve the performance of the final model. In order to do that, the simplest way is to use weighted average the prediction of selected models. Our ensemble model prediction is a linear combination of Catboost, LightGBM, and neural networks models. We estimate the weights based on the performance of the models on the leaderboard where the weights of Catboost, LightGBM, and neural networks are 0.6, 0.3, and 0.1, respectively.

5 EXPERIMENTS

In this section, we discuss the experiment settings including hyper-parameters selected for each model and the performance of each model in different settings based on our internal validation set.

5.1 Experiment Settings

We use two boosting based models: Catboost and LightGBM where we use larger learning rate and fewer trees for LightGBM (**LGB Parameters** - *num_leaves*: 35, *max_depth*: 8, *min_child_sample*: 100, *subsample*: 0.7, *colsample_bytree*: 0.7, *n_estimators*: 15k, *learning rate*: 0.2), and smaller learning rate for Catboost (**CAT Parameters** - *iterations*: 53k, *border_count*: 254, *max_depth*: 11, *learning rate*: 0.03).

5.2 Results

As mentioned in Section 3.4, we generate two types of embeddings from the following types of graphs: user-hashtag graph (UH) and user entity graph (UE), where the embedding size is 64 in both the cases. Table 2 shows the performance of Catboost model on different feature sets. In this setting, we would like to study the impact of graph based features on the performance of our models. Without using any graph features, Catboost (CAT) has the error of 0.12494. By using normal graph features such as UH and UE, we can improve the model 0.0015. It means that adding the graph embeddings improves the performance of the models.

Table 1 shows the performance improvement using k-fold and common model strategy with the baseline models. It shows that all our base models improves by using these strategies. Further Table 3 shows the performance of various ensemble approaches, it shows that ensemble of all four models yields the best performance. We use the weighted average approach to combined our predictive models. Please note that we don't have all results for every model. We only choose important models to submit and get the score.

Table 2: Graph Features on Internal Validation

Model + Embedding	CAT	CAT + UH	CAT + UE
Validation Score	0.12494	0.12464	0.12343

As we only know the score after submitting our submission, this table only shows part of all results. Note that we don't submit $CAT^{**} + CNN^{*}$ ensemble in the validation phase so we don't have its score. Based on the results, one can see that adding more models can improve the accuracy. For instance, $CAT^{**} + CNN^{*} + LGB^{*}$ ensemble works better than $CAT^{*} + LGB^{*}$ and out final ensemble consists of 4 algorithms.

Table 3: Comparison of Various Ensemble Approaches

Model	Valid Score	Test Score
$CAT^{**} + CNN^{*}$	-	0.121874
$CAT^{**} + LGB^{*}$	0.12518	-
$CAT^{**} + CNN^{*} + LGB^{*}$	0.12511	-
$CAT^{**} + CNN^{*} + NN^{*} + LGB^{*}$	-	**0.121094**

Note that $*$ indicates base model + k-fold and $**$: base + fold + common model.

6 CONCLUSION

We have presented our solution for COVID-19 retweet prediction which consists of feature engineering and bagging/down-sampling modelling techniques. The proposed approach is ranked ***First*** and ***Second*** on the public and private test sets, respectively. With an exhausted feature set and powerful modeling techniques, we hope that our solution provides a solid baseline for retweet prediction research, especially in the crisis time like COVID-19 pandemic.

REFERENCES

[1] Dorogush Anna Veronika, Ershov Vasily, and Gulin Andrey. 2018. CatBoost: gradient boosting with categorical features support. (2018).

[2] Francois Chollet et al. 2015. *Keras.* https://github.com/fchollet/keras

[3] Dimitar Dimitrov, Erdal Baran, Pavlos Fafalios, Ran Yu, Xiaofei Zhu, Matthäus Zloch, and Stefan Dietze. 2020. TweetsCOV19 - A Knowledge Base of Semantically Annotated Tweets about the COVID-19 Pandemic. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management.* Association for Computing Machinery, New York, NY, USA, 2991–2998. https://doi.org/10.1145/3340531.3412765

[4] Kenton Lee Jacob Devlin, Ming-Wei Chang and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805* (2018).

[5] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17).* Curran Associates Inc., Red Hook, NY, USA, 3149–3157.

[6] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. Pytorch-biggraph: A large-scale graph embedding system. *arXiv preprint arXiv:1903.12287* (2019).

[7] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. PyTorch-BigGraph: A Large-scale Graph Embedding System. In *Proceedings of the 2nd SysML Conference.* Palo Alto, CA, USA.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[9] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks.* ELRA, Valletta, Malta, 45–50. http://is.muni.cz/publication/884893/en.