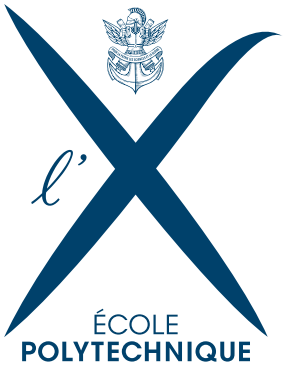

INSTITUT POLYTECHNIQUE DE PARIS

Tweet Challenge Report

Swann Chelly : swann.chelly@polytechnique.edu
Anass Ben Bouazza : anass.ben-bouazza@polytechnique.edu
Hugo Artigas : hugo.artigas@polytechnique.edu
Kaggle Team : Anass - Swann - Hugo

***Abstract*—Our goal for this project was to predict the number of retweets of tweets published during the first height of the current coronavirus pandemic. To do so, we first experimented several pre-processing methods in order to make the most sense out of the data. This step resulted in three datasets, over which we then trained and optimised two models : a neural network and XGBoost. Our second pre-processing method and its XGBoost model enabled us to reach an MAE loss of 150.25.**

I. Introduction

The emergence of social media this past decade as a major way for people to get information also caused the rise of misinformation among populations. Indeed, the very decentralised nature of websites such as Facebook or Twitter implied that no control on the veracity of the information that spread was done. As such, several entities made a playground out of these places and tried to influence public opinion in ways that were favorable to them : the Cambridge Analytica scandal of 2018 exemplifies this phenomenon.

The INF554 challenge arises from the need that we have to understand the spread of information on social media. Here, we had to predict the number of retweets samples from the COVID19 Twitter Dataset got. We describe in this report our methodology.

The models we designed had to minimise the Mean Absolute Error (MAE) :

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |\hat{p}_i - p_i| \tag{1}$$

with $(\hat{p}_i)_{i=1}^N$ the predicted number of retweets and $(p_i)_{i=1}^N$ the actual number of retweets in the evaluation dataset.

II. Data pre-processing

The first step we had to go through was data pre-processing, during which we encountered one major challenge : we had to transform the string variables in the data into meaningful numbers. In order to do so, we used three pre-processing methods, which we will call *Preproc 1*, *Preproc 1bis* and *Preproc 2*.

But before even that, we first set to ourselves to clean up the dataset : we removed all non-letter characters from all strings, made them into lower case and created the following features, of which we explain the instinct behind :

- *hashtags_length* : Number of hashtags. Tweets that had more hashtags would be more visible for the community.
- *day* : Day of the week of the release time. Tweets that were published during non-working days had more chances to get the initial number of retweets that would then make them go viral.
- *time*: Distance of the release time away from noon. Tweets that were published during non-working hours had more chances to get the initial number of retweets that would then make them go viral.
- *text_length*: Length of the text (without user mentions and hashtags). Tweets that were either too wordy or not enough would attract less retweets, and catchphrases would be more likely to spread widely.
- *user_mentions_length* : Number of users mentioned. Tweets that mentioned other users invited discussion or controversy, and as such reactions.

Finally, we divided the *train* dataset into training and validation datasets with ratio 70/30.

A. *Preproc 1 : Tensorflow Tokenizer*

Our first instinct when trying to process the string variables was that there were for instance buzzwords or hashtags that made tweets much more likely to go viral. To traduce this feeling into features, we used *Tensorflow*'s tokenizer on each of the *user_mentions*, *urls*, *hashtags* and *text* variables separately, in order to get four vocabularies of the 1000 most used words

in each feature. Using these vocabularies, we were then able to transform strings into sequences of numbers and by padding these outputs, we obtained matrices of size $(N, 1000)$ with N the number of tweets in the training set. At indices i, j of these matrices is 1 if the word j of the vocabularies is present in tweet i's corresponding feature, 0 otherwise.

In order to identify these buzzwords, we performed PCA on these matrices and visualised the projections of the tweets on the first two axes of the PCAs for each feature. Distinct clusters were present (Figure 1). We then used k-means clustering to associate to each tweet and for each feature a label according to the clusters it belonged to. A second improvement, which led to Figure 1, was to consider the frequency of a word instead of only its presence. We then looked for the most common word in each cluster (Table I).

We could however not make sense of the two clusters present in the case of *text*. This failure motivated us in exploring two other pre-processing methods for text : *Preproc 1bis* and *Preproc 2*. Since the first two axes of the PCA for *text* captured almost 70% [1] of all eigenvalues, we decided in its case to keep the coordinates of the *text* variable on these first two axes instead of the cluster.

To get a quick validation of the influence of our clusters, we looked at the mean number of retweets for tweets in each cluster. This induced us to think that indeed, the cluster variables had meaning : for instance, tweets that mentioned Donald Trump were on average five times more retweeted that tweets that mentioned someone else, and had superior retweet variance : $\mu_{trump} = 566$ with $\sigma_{trump} = 2470$ against $\mu_{other} = 116$ with $\sigma_{other} = 1466$.

B. *Preproc 1bis : TF-IDF*

However, as explained in part III, *Preproc 1* did not lead to satisfactory predictions, and *Preproc 1bis* is an evolution of *Preproc 1* that arised from this issue.

We did not give up on trying to identify words in *text* that were common in the dataset and that could lead us to better prediction of the retweet counts. We therefore explored more frequency-based text vectorization methods and stumbled upon an article, which gave us inspiration, that used the TF-IDF statistic for tweet classification [1]. This time, simple and common words such as "and" or "is", that held little insight and that polluted our previous clustering of *text* in *Preproc 1*, were less accounted for. This helped us achieve better performance, as highlighted in table II.

The output of this method is a matrix representing the TF-IDF statistic of each word in each tweet. This matrix initially was of size (665776, 504790) for the *train* dataset. Such a gigantic, and sparse, matrix held little information as was and would have involved enormous computing times when applying our models. We therefore used NMF to reduce its 504790 features to 10. We then stripped the two PCA coordinates of *text* from the *Preproc 1* dataset and added these 10 variables in order to get *Preproc 1bis*.

[1]This was obtained when performing a PCA on the frequency matrix of words in the sentences

C. *Preproc 2 : Bidirectional Encoder Representations from Transformers (BERT)*

We also had the intuition that the meaning of a tweet could deliver useful information in order to get back to its number of retweets. For example, we had the feeling that messages that revealed infuriation or euphoria would attract more reactions than milder tweets. We therefore searched for inspiration in previous works on Tweet classification [5, 9] and tried to replicate their methods. As such, we used BERT [3], a pre-trained model for language representation developed by Google Research, in order to make sense of the text.

For *Preproc 2*, we first removed from the variable *text* all content, such as urls or retweets, that was already accounted for during *Preproc 1*. We then tokenize and pad the texts using *transformers*, a *Python* module that implements BERT. Finally, we feed these tokenized and padded texts into BERT Base, the output of which we only retain the coordinates of the full text in the 768-dimensions embedding space.

Since using all of BERT's output for all 951111 tweets would have lengthened greatly both the training and validation of our models and hampered our optimization efforts, we used PCA to generate 50 features from those 768. These 50 last variables are then added to *Preproc 1*'s output, and we remove the untreated *text* from the dataset as well as the two coordinated we obtained during *Preproc 1* for the *text* variable.

III. Model selection and tuning

Using our two pre-processing ideas, which at first only led to *Preproc 1* and *Preproc 2*, when then moved on to model selection and tuning. Our first experiments on several algorithms using Huber loss (table II) funneled our efforts towards two models : neural networks and XGBoost.

The difficulties we met during this step also led to the creation of *Preproc 1bis*, which gave us the best predictions we could achieve, when combined with an XGBoost model. The detail of all the tests we did can be seen in table II.

A. *Neural Network*

Since we had little to no experience working with neural networks before this project, we thought this challenge would be a perfect opportunity to hone our deep learning skills, and therefore decided our first model would be a NN.

To do so, we first normalized the data in order to remove scale effects. Then we used bayesian grid search with *Keras*'s *Tuner* library so as to select the number of hidden layers and the hyperparameters for our model. The best neural network we were able to build included 3 hidden layers, the first layer had sigmoid activation in order to sparse the values, other layers had ReLU activation in order to create non-linearity in our model.

Using it on *Preproc 2*, we obtained an important decrease for the test loss, reaching as low as 132.43 MAE (figure 2), when using L1 regularization and cross validation. This performance did not however reflect on the Kaggle leaderboard, where our neural network did not manage to achieve under 155 MAE.

B. *XGBoost*

We also experimented with XGBoost on *Preproc 1* and *Preproc 2* in parallel of our deep learning efforts.

Before all attempts at optimization, this did not lead to suitable predictions, as we obtained MAEs larger than 180. However, we figured out that this issue was caused by the fact that *Scikit-learn*'s implementation of XGBoost minimized the mean squared error by default instead of the MAE, and we set out to test several loss functions that would better approximate it. Since the MAE is non-differentiable and cannot be used directly, we compared three such functions, which we had found in the literature [10, 2, 4, 8]: the Huber, 'Fair' and *Log-Cosh* loss functions.

$$Huber : L(y,f) = \begin{cases} \frac{1}{2}(y-f)^2 & \text{for } |y-f| \leq 1, \\ |y-f| - \frac{1}{2} & \text{otherwise.} \end{cases} \quad (2)$$

$$Fair : L(y,f) = |y-f| - log(1+|y-f|) \quad (3)$$

$$Log-Cosh : L(y,f) = log(cosh(y-f)) \quad (4)$$

This helped us lower the test loss to around 146 for *Preproc 2*. However, *Preproc 1* did not perform so well. We understood that since the results of the PCA for the *text* variable were the only ones we could not understand, we had to do some further research in order to come up with a better treatment of *text*. This resulted in *Preproc 1bis*.

We then tried bayesian grid search in order to optimize the hyperparameters of our models. We observed that whatever the interval we set for the number of estimators in XGBoost, the search always indicated its upper bound to be optimal, we thus chose to first optimize this parameter, then optimize all others. However, the more estimators we used, the more likely we were to overfit. This induced us into testing both L1 and L2 regularization in our models, and L2 regularization resulted in our best and final model, that used *Preproc 1bis* :

- Loss function : Huber
- *gamma* = 7.445
- *n_estimators* = 20000
- *max_depth* = 14
- *reg_lambda* = 10
- *min_child_weight* = 0

Finally, since the model never predicted integers, we truncated all predictions so that they had no decimals. This enabled us to lower the MAE since the numerous tweets that had zero retweets were now predicted to have exactly zero retweets.

Once we had obtained this final model, we tried to confirm the importance of the features we had created by plotting their relative importance in the XGBoost model (figure 3). These plots revealed that our previous clustering of *user_mentions*, *urls* and *hashtags* is important for performance since they are among the ten most important features in terms of gain. The same can be said for the new variables generated in *Preproc 1bis*, that have an important weight in the model.

These plots also indicated that in order to improve the model, it would have been interesting to try to better understand patterns in *user_followers_count* and *user_statuses_count*.

IV. Discussion

Several improvements could have been made in various steps of our work in order to achieve better performance.

One such improvement would have been to spend more effort analysing the output of our TF-IDF or BERT pre-processing in *Preproc 1bis* and *Preproc 2* the same way we did in *Preproc 1*. By testing several dimensionality reduction algorithms (t-SNE [6], UMAP [7], ...), then applying unsupervised learning methods, we might have been able to extract more useful information from our data.

Trying to apply transformations to the *retweet_count* variable could have resulted in interesting results as well. Indeed, we have observed that the vast majority of samples had close to no retweets, so much so that *retweet_count* had an important variance and was absolutely not uniformly distributed : an appropriate transformation could have made this crucial feature more understandable for our algorithms.

V. Conclusion

We succeeded during this challenge in coming up with our own pre-processing technique, then analysing its output and comparing its performance against methods used in previous research in order to extract as much information as possible from textual data. Our attempts at tuning both neural networks and an XGBoost model then enabled us to reach a MAE loss of 150.25 on the private dataset in the Kaggle competition.

References

[1] Ali Alessa and Miad Faezipour. "Tweet Classification Using Sentiment Analysis Features and TF-IDF Weighting for Improved Flu Trend Detection". In: *Machine Learning and Data Mining in Pattern Recognition*. Ed. by Petra Perner. Cham: Springer International Publishing, 2018, pp. 174–186. ISBN: 978-3-319-96136-1.

[2] Pengfei Chen, Guangyong Chen, and Shengyu Zhang. *Log Hyperbolic Cosine Loss Improves Variational Auto-Encoder*. 2019. URL: https://openreview.net/forum?id=rkglvsC9Ym.

[3] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[4] Peter J. Huber. "Robust Estimation of a Location Parameter". In: *Breakthroughs in Statistics: Methodology and Distribution*. Ed. by Samuel Kotz and Norman L. Johnson. New York, NY: Springer New York, 1992, pp. 73–101. ISBN: 978-1-4612-4380-9. DOI: 10.1007/978-1-4612-4380-9_35. URL: https://doi.org/10.1007/978-1-4612-4380-9_35.

[5] Guoqin Ma. "Tweets Classification with BERT in the Field of Disaster Management". In: 2019.

[6] Laurens van der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: (2008).

[7] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. 2020. arXiv: 1802.03426 [stat.ML].

[8] William J. J. Rey. "Introduction to Robust and Quasi-Robust Statistical Methods". In: Springer, Berlin, Heidelberg, 1992. ISBN: 978-3-642-69389-2. DOI: 10.1007/978-3-642-69389-2. URL: https://doi.org/10.1007/978-3-642-69389-2.

[9] Hamada M. Zahera. "Fine-tuned BERT Model for Multi-Label Tweets Classification". In: *TREC*. 2019.

[10] Zhengyou Zhang. *Parameter Estimation Techniques: A Tutorial with Application to Conic Fitting*. Research Report RR-2676. INRIA, Oct. 1995, pp. 35–37. URL: https://hal.inria.fr/inria-00074015.

APPENDIX

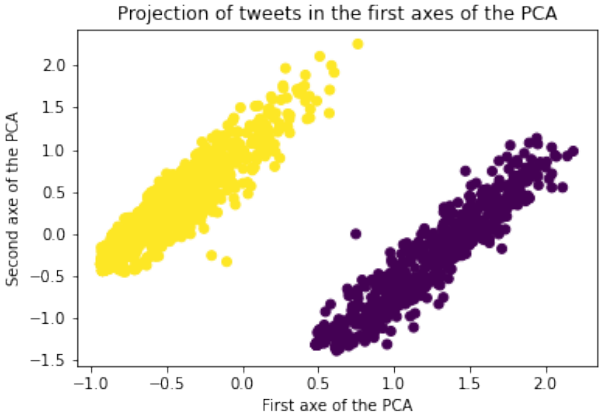

Fig. 1: Projection of the output of Tensorflow tokenizer (One Hot Encoded) on PCA axis

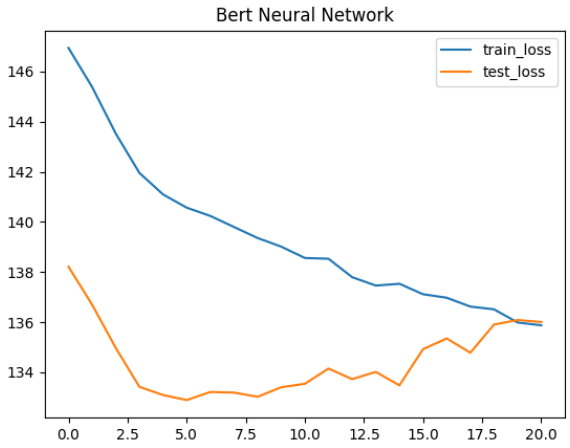

Fig. 2: Train test loss for our neural network with *Preproc 2* on 100 epoch and LASSO regularization of 10^{-2}. The loss was computed every 5 epochs.

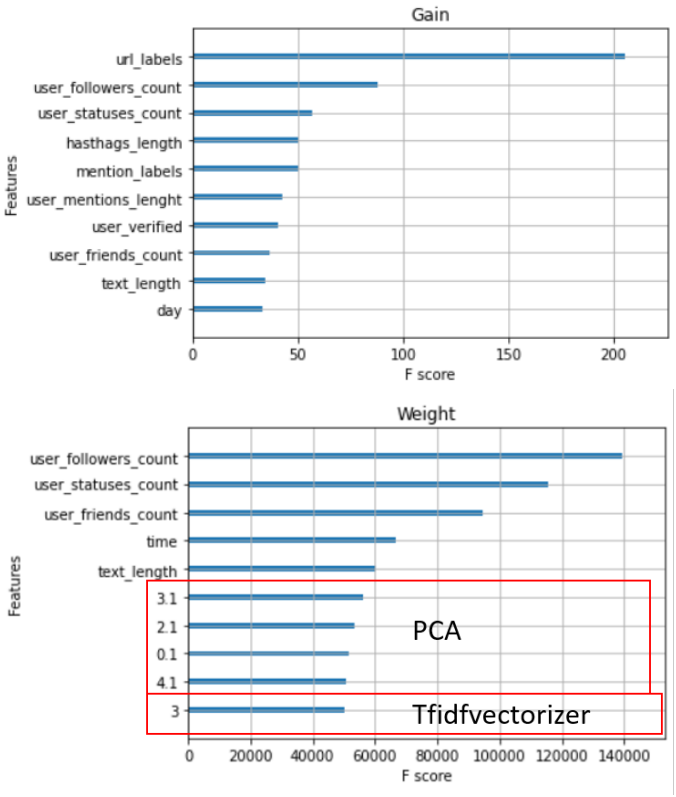

Fig. 3: 10 most important features according to gain and weight criteria in our final XGBoost model. Gain is the average gain of splits that do use the feature. Weight is the number of times a feature appears in a tree.

Cluster	1	2	3	4
Hashtags	No #	COVID-19#	coronavirus#	other#
Urls	No url	Twitter	other url	
User Mention	No mention	Donald Trump	other user	

TABLE I: Most common patterns in all clusters for each string variable

	Neural Network		XGBoost		SVM	Random Forest
	No Tuning	Tuned	No Tuning	Tuned	No Tuning	
Preproc 1	149.71	142.84	162.22	146.81	170.49	165.63
Preproc 1bis	138.50	136.96	151.01	**133.94**	153.36	153.00
Preproc 2	139.08	132.43	146.33	134.11	152.45	149.98

TABLE II: Comparison of MAE on our validation set for several models over all three pre-processing methods. "No Tuning" means we used Huber loss and default hyperparameters (except for NN where we kept squared error). "Tuned" means we specifically tuned the model for each one of *Preproc 1, 2* and *3*. Our best and final model is in bold.