Data Challenge INF554 – Team CCGD

Team members :

- Clément Garancini, X2020, clement.garancini@polytechnique.edu
- Charles de Malefette, X2020, charles.de-malefette@polytechnique.edu
- Jean Dimier de La Brunetiere, X2020, jean.dimier-de-la-brunetiere@polytechnique.edu

Section 1: Feature Selection/Extraction

The available features for our model were:

- the tweetID (even if used only to keep track of tweets)
- the timestamp
- a label for verified accounts, the statuses count, the followers count, the friends count
- the text of the tweet, deprived from stopwords in the dataset, the mentions (but they were all set to empty lists, so this one is actually unusable), the hashtags, the URLs and included in the tweet the favorites count

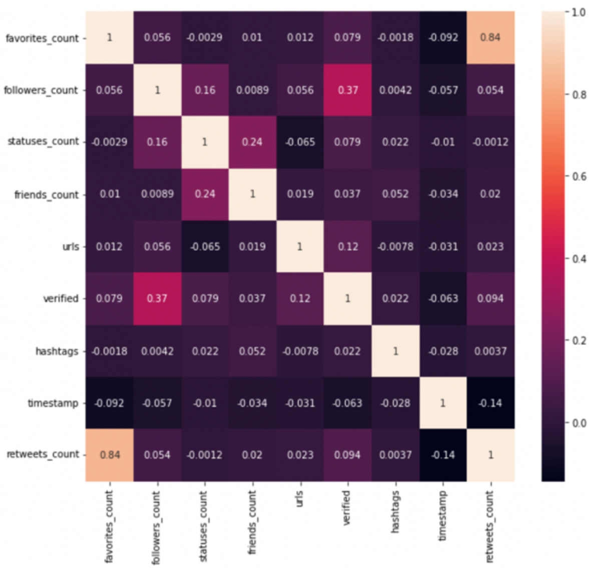
Intuitively, all these features don't play the same role in the degree of retweetability of a tweet, and that some will be highly correlated with the number of retweets in the end.
The most obvious one is for sure the "favorites count": tweets that have a high number of retweets tend to have an even higher one of favorites, and if one figure is low, the other is often low too.
Then come the features describing the user: the verified label, the followers count, the friends count and the statuses count. We can feel that oldest is the account, highest will be all those numbers; but we can have a clear overview of the potential spreading of a tweet with the followers count: if a tweet by someone followed by ten thousand people, verified, is to be retweeted, say, at least 30 times, it is likely to be retweeted hundreds of times, from empirical experience.
Then comes the actual content of the tweet: the URLs included, and the text where stopwords were retired, and the timestamp. It is of course the main element for a human to actually know if a tweet is going to be retweeted, *but* it surely is not well understandable for a machine, that will not be able to sense its partisan side, or whether it is humoristic or not. URLs can be of great help, provided we can analyze them: we found only not free APIs to have sort of a PageRank for URLs, an example of which is the API of similarweb.com. Hashtag and URLs can give visibility, but it is hard to estimate the consistency of each hashtag or URLs. That's why we will just consider the hashtag and URLs count and not their content.

Now, that was for the intuition behind feature selection: let us analyze them to confirm or infirm this intuition. We began by displaying the correlation matrix between each feature:

Some conclusions:

- The number of followers and the boolean features "verified" are corelated which is logical concerning the running of tweeter. The verified profiles are most of the time the count that have many followers. However, we cannot draw any decisive conclusion about that because the correlation rate is still low. (0.37).

- There is a huge correlation (0.8) between the favourite count and the number of retweets. It was predictable and it shows that it will be the main feature to consider.



Then, let us introduce a figure from a research paper [1] that helped us understand how those features were correlated, through a PCA with the loadings corresponding:

Table 3 Principal component loadings

Metrics	PC1	PC2	PC3	PC4	PC5
Retweet count	−0.1623	**0.4346**	0.1635	−0.0026	−0.1009
Favorites count	**−0.6294**	0.3908	0.1922	−0.1128	−0.1880
Followers count	**−0.7599**	0.2736	0.0522	−0.0983	−0.0857
Followees count	−0.1336	−0.0907	**−0.4627**	−0.2494	0.1182
Listed count	**−0.8431**	−0.1549	−0.0498	0.1500	0.1871
Statuses count	−0.4256	**−0.5016**	−0.3781	0.2795	0.2410
Hashtags count	−0.1585	**−0.5661**	0.4377	−0.0517	0.0309
Mentions count	0.0394	0.2194	0.0786	−0.1607	**0.7697**
URLs count	−0.1288	**−0.5483**	0.2539	−0.3388	−0.3248
Publication time	0.0076	−0.0728	0.3639	**−0.5186**	0.3707
Days count	−0.0370	0.0070	−0.5072	**−0.6604**	−0.1691

Data reported in bold are the most relevant in the context

The features are not exactly the same, but they mostly follow the main ideas of our analysis.

To dig deeper, we chose to run ours, and here are the results we got:

```
array([[ 7.75630102e+01,  4.78778188e+01, -1.43119277e+01],        Favorites_count
       [ 1.43951985e+04,  2.59836914e+05, -9.71006658e+03],        Followers_count
       [ 1.57172115e+03,  2.26571426e+04,  1.11356139e+05],        Statuses_count
       [ 9.92030913e+01,  3.73940333e+01,  6.01601755e+02],        Friends_count
       [ 1.67329793e-02,  2.68924364e-02, -3.85885629e-02],        Urls_count
       [ 1.08553298e-02,  6.32157986e-02,  1.00581192e-03],        Verified
       [ 1.86104369e-02,  3.42501891e-03,  1.63759343e-02],        Hashtag_count
       [-4.72480509e+09,  7.99191078e-01,  7.47142665e-03]])       Timestamp
```

These seem to be relatively close to the precedent one, which is reassuring.

In most of the models we tested, we first only counted the number of URLS and hashtags and did not take into account the text in order to have only numerical values. Hence, we have first results to get the hang of these types of models and develop an intuition for the best model to use. We would have liked to analyze a PageRank-like score for URLs and hashtags, but we did not manage to find a viable solution as mentioned above.

Our main goal was then to add the textual content of the tweets with the TF-IDF Vectorizer, and then used a Transformer from sklearn.preprocessing to mix it with the numerical values; we spent too much time on using a transformer without success.

2. Model choice, Tuning and Comparison

XGBoost:
We tried to fit different types before going for one in particular. We had a first idea of what to use thanks to the afore-mentioned paper [1], with tables such as the following, comparing global performances and difficulty to implement/explain:

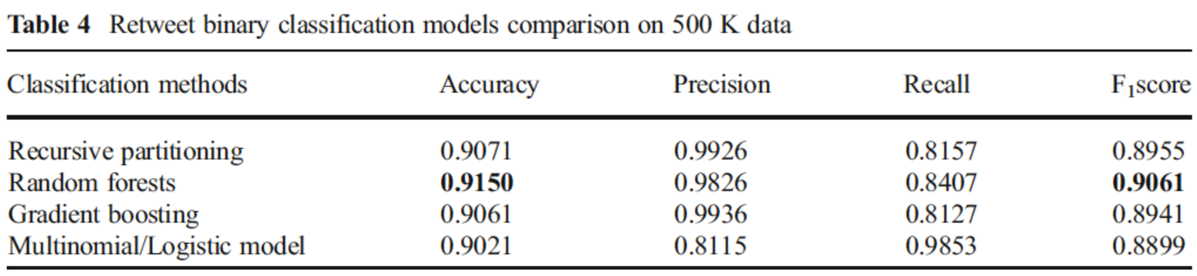


Table 4 Retweet binary classification models comparison on 500 K data

Classification methods	Accuracy	Precision	Recall	F_1score
Recursive partitioning	0.9071	0.9926	0.8157	0.8955
Random forests	**0.9150**	0.9826	0.8407	**0.9061**
Gradient boosting	0.9061	0.9936	0.8127	0.8941
Multinomial/Logistic model	0.9021	0.8115	0.9853	0.8899

We tried random forest methods, but the way we implemented it was too computationally expensive, so we let it aside, in favor of Gradient Boosting and Extreme Gradient Boosting (XGBoost).
XGBoost had the advantage of being parallelizable, which was a great asset for us to conduct hyper-parameter tuning in a reasonable time. In the end, the algorithm was in most cases well-designed enough for us to fit it even without the GPU in very short time (less than one minute without textual content for example). For this tuning, we first conducted a GridSearchCV method, but results were disappointing, and we have not figured out yet what is the reason for that, even if we suspect the actual Cross-Validation process and a bad implementation from ours.
Therefore, we just re-created a GridSearchCV-like loop by hand and tuned our parameters this way: the parameters tackled were classical, that is to say: `n_estimators`, `learning_rate`, `max_depth`, and `subsample`.
The hyperparameters we came up with were not more exotic: `n_estimators= 120 learning_rate"max_depth=7`, `subsample = 0.80`.
The subsample parameter not set to 1 was our main weapon against overfitting, according to what we studied in our courses, in INF but also in MAP from the theoretical point of view.

Neural Network:
We then tried to implement a neural-network based model of deep-learning for the text. To do so, we tried to follow the main ideas of [2] in a simpler manner, which implemented an Attention-Based Neural Network, to “mimic” human behavior when looking at a tweet. We used keras which is a deep learning API written in Python running, on the machine learning platform Tensorflow. Two layers with sigmoid (and ReLu in a second time) activation functions were added to the model. However, the result was very disappointing because the score was constant over the epochs. The explanation is that we choose the mean-absolute-error function as the loss function. Our model was not suitable, so we tried to use the model TextRegressor from autokeras that tests alone some of the possible layers combination and uses the best combination of layers to get the best result after training. The loss results were

still bad, so we abandoned the neural network use. Either the network was not the appropriate one, or the choice of the word embedding method was not the good one.

Word embedding:
Concerning the choice of the word embedding method, we thought of using the bag-of-words model which works on a whole text, and which creates a metric that puts semantically close words also close with respect to this metric. However, our database only provided keywords and not whole texts. In order to train such a model, it was therefore necessary to obtain a database of French texts on the topic of the elections, but we had the idea too late to find such a corpus and to implement it. Consequently, we tested the TF-IDF method. TD-IDF stands for Term Frequency — Inverse Document Frequency and analyzes the frequency of a word in a text and a corpus to return the importance of the word in that text. However, we did not manage to combine successfully the analysis of the text with the other features. The word embedding is not used in our final method.

Finally, by analyzing the results returned by the algorithm, we ran a small improvement to gain performance in an easy way: we realized that to predict an almost-0 probability of being retweeted, the prediction was around 0.12: we just applied a rounding transformation for such small values predicted, which indeed brought a slight improvement in the performances.

References:

[1] Assessing the reTweet proneness of tweets: predictive models for retweeting, *Paolo Nesi, Gianni Pantaleo, Irene Paoli, Imad Zaza, 2018*

[2] Retweet Prediction with Attention-based Deep Neural Network, *Qi Zhang, Yeyun Gong, Jindou Wu, Haoran Huang, Xuanjing Huang,* 25th ACM International, October 2016