INF 554: Covid19 retweet prediction challenge

Katherine Hong, Guillaume Loranchet and Flavien Vidal

December 2020

1 Introduction

The goal of this project is to predict the number of retweet given several features like user's number of followers, number of friends, the content of the tweet (text and hashtags) etc. We consider the problem as a regression problem: we are trying to predict over a continuous space. The main difficulties were the size of the dataset (655 777 tweets for the training set), the unequal distribution (more than 70% of the tweets have 0 retweets) and finally the feature selection (no feature clearly stands out).

2 Feature Engineering

Throughout this project we noticed that the feature engineering part was surely one of the most crucial part of building our models. If we have useful features, the model will perform a lot better. We spent a considerable amount of time on this part by meticulously analyzing each feature of the dataset to try to extract the most crucial features.

We have a total of 10 attributes: 5 numerical attributes (id, timestamp, user_statuses_count, user_followers_count, user_friends_count), 1 Boolean attribute (user_verified), 4 text attributes (user_mentions, urls, hashtags, text) and 1 target variable (retweet_count).

2.1 Input features transformations:

Hashtag Analysis

The hashtags within each tweet are included and are comma separated. If nothing is mentioned the field is left empty. We want to encode the hashtags in a way that maps a given tweet to categories (the main topics) defined by the user's hashtags. To do this we start by analyzing the different tags. Only 11.02% of the examples contain at least one hashtag and the distribution of the most used hashtags is as follows: PHOTO We realize that the most used hashtags are in fact often the same but written in different ways. By keeping the present hashtags, our model will consider " coronavirus", "coronavirus ", "CORONA_19 "or "COVID19" as belonging to different categories. However, these four hashtags should belong to a single category: "CORONAVIRUS". In order to make the best use of this feature we need to standardize it first by:

- removing whitespaces, dashes and underscores,
- converting all lowercase characters to uppercase characters,
- converting all strings related to the coronavirus disease by "COVID".

Once we have normalized the hashtags, the 10 most used hashtags are as follows: PHOTO

We can see that the most used hashtag is obviously ”COVID”, but other hashtags are very much used and can have an importance on the popularity of a tweet. Indeed, it seems that tweeting about a popular topic such as Covid19 increases the chances of being retweeted as shown: PHOTO

After normalizing the ”hashtags” feature we have created a new attribute that determines the number of hashtags of a given tweet. This attribute is strongly correlated to the number of retweeting.

Timestamp

The time at which the tweet was published is an important data. To analyze it we extracted:

- the date part,
- the time part,
- the week ordinal of the year,
- the month of the year,
- the day of the week (with Monday=0, Sunday=6),

We observed that the data were taken over 5 days, so we wished to keep only the day of the week, the time of day and the timestamp since there seemed to be a relationship between these parameters and the popularity of a tweet.

User_verified

A Boolean field indicates whether or not the user has been verified by Twitter. The analysis of this field showed that certified accounts generally generated more retweets than others. We kept this feature intact.

User_statuses_count - user_followers_count - user_friends_count

We also keep track of the total number of tweets the user has posted, the number of followers the user has, and the number of friends the user follows.

user_mentions - urls

Users that are mentioned in the tweet are separated by commas and if nothing is mentioned, the field is empty. We have transformed this data into 2 features: one gives the presence or not of a mention in the field (1 or 0), the other gives the number of mentions in the field. After training and testing our estimators it seemed that both features were about equally important. The same applies to URLs included in the tweets.

text

The text of the tweet as it was posted by the user is an essential data in the prediction of our target. Users retweet contents they liked or found interesting enough to share with others. Nevertheless, it is very difficult to work on such a feature. To analyze text data, we need to transform it into data that can be digital input. This is done by mapping raw text to counts of words. Meaning is built from ordered sequences of words that refer to each other, often separated across sentences or paragraphs. However, in our case and in the vast majority of text analysis this complexity is ignored, and we will rely solely on counts for language tokens (words). We are inspired by the bag-of-words representation of text and more particularly we use sklearn’s term frequency-inverse document frequency vectorizer function. After several tests we chose to build a vocabulary that only consider the 100-top max_features ordered by term frequency across the corpus. 100 new features corresponding to the 100 most used words are then created. They allow to have a general idea of the topics most often discussed in the tweets. In addition to creating these features we decided to perform a sentiment analysis on the content of the tweets. To do this we created a function using the TextBlob library and the sentiment method. After applying this function to the ”text” series we get both the polarity of a tweet and its subjectivity (between -1 and 1). Finally, we decided to create a feature containing the size of the text that was correlated to our target.

2.2 Target feature transformation:

Retweet_count

The actual number of retweets the tweet receives at the time of the crawl is the target we are trying to predict. The distribution of the target feature is tail-heavy. It is highly imbalanced (right-tailed): PHOTO Since the data are skewed with a disproportionate number of retweets between 0 and 10 (about 70% of the dataset), a logarithmic transformation on the number of retweets seems to be the most appropriate. Nevertheless, we performed tests on different transformations including: log2(1+target), ln(1+target), log10(1+target), (-1)/(1+target)+1 and sqrt(target) The difference between the various logarithms applied did not highlight a particular transformation but in general log transformations seemed more appropriate. We finally chose to apply a natural logarithm to the target. PHOTO The new distribution may at first glance seem only slightly better but applying this transformation has boosted our results very significantly.

Target feature transformation: So far we had handled the feature engineering of the numerical and text features manually by analyzing each feature independently. We created a single transformer able to handle all the created features and applying them the appropriate transformations. We imported the ColumnTransformer class and get the list of numerical and categorical columns and constructed the preprocessed dataset. We used StandardScaler for numerical values, TfidfVectorizer for the text feature and CountVectorizer for the hashtags categories. We tried to fiddle the methods used in the pipelines on the different estimators.

3 Prediction

3.1 Multilayer Perceptron Regressor

To facilitate feature selection, we implemented a Multilayer Perceptron Regressor (MLP). We tested a lot of different combinations of parameters. The configuration used the most was 6 hidden layers of size 64,128,64,32,16,8, learning rate set to adaptive. We also used early-stopping to avoid over-fitting with different stopping parameters (see table). We also tried to simplify to network (e.g (128,128,64)) for hidden layers but it seems to constantly be 2 points below the first one and thus under-fitting a bit.

3.2 Gradient Boosting

Gradient boosting was originally yielding poor result due to non-optimal data processing but then became one of the best method even we default parameters. We still ran a Grid Search Cross Validation with $max\_depth : [2, 5, 10]$, $n\_estimators : [25, 50, 100]$, $learning\_rate : [0.01, 0.1, 0.5]$. We found that max depth = 10, n estimators = 100 and learning = 0.1 were optimal. We also tried to increase the max depth and the number of estimators and the training score was always much higher but the testing score was just a few points above. It was thus over-fitting quite a lot.

We also used XGB Regressor that sometimes gave better results than normal Gradient Boosting.

3.3 Linear Regression

Linear regression is one of the most straightforward methods. However, simply applying the linear regression model did not give us promising results. The data is often polarized: most tweets have 0 or near zero retweets, while some have very high retweet numbers. As we see in Figure 1, after a linear regression, applying the predictive model on the evaluation set will sometimes give us negative retweet count predictions.

To avoid this problem, we simply replaced all negative predictions with 0. Another issue with linear regression is, since the retweet counts are either near zero or large numbers, such discrepancies will create a steep slope in the linear regression model. When the model make new predictions, it's more likely to make mistakes if the distance between the new features and the linear regression model is large. To reduce the mean absolute error, we took the square root of all predictions. This significantly reduced the mean square error to 148 on our 30% preliminary test dataset. Our linear regression and prediction post-processing code is shown in Listing 1.

3.4 Random Forest

Similarly to Gradient Boosting, Random Forest has a lot of hyper-parameters to tune. This time, we used a Randomized Search Cross Validation to be able to test more combinations. After comparing 50 different combinations among: n_estimators: $[1000, 1500, 2000]$, max_features': $['sqrt','log2']$, max_depth: $[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None]$, min_samples_split: $[2, 5, 10]$, min_samples_leaf: $[1, 2, 4]$, bootstrap: $[True, False]$, we obtained that the best combination of parameters is: n_estimators: 2000, min_samples_split: 10, min_samples_leaf: 4, max_features: $sqrt$, max_depth: 70, bootstrap: $False$. However, the data was noisy and it was hard to see a parameter stepping up. We also tried to run other RSCV but due to huge computation it requires, it often crashed.

3.5 Voting Regressor

The idea behind a voting regressor is to take several regressors and combine them to obtain, in principle, a better regressor. Once again, we tried we different combinations of regressors. The first combination we tried was with the gradient boosting regressor and the MLP as they are quite different models. We, then tried to add other models such as Random Forest, Linear Regressor, but the result were not much better, and sometime (on the validation. We end up using only Gradient Boosting and MLP.

4 Conclusion

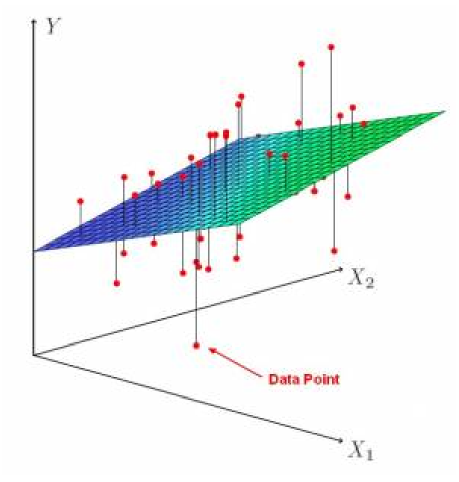
After testing several models, the main information we gain in the importance of processing the data. Indeed, most on the models we had we quite simple in the end we the change in data made the most difference. One the most important tools we used were probably the Grid Search and Random Search as they allowed us to test a lot of different parameter and find the most optimal tuning. Not over-fiting was the hardest part of the project, as the data had a lot a noise and was uniformly distributed.

References

[1] Classify structured data with feature columns,
`https://www.tensorflow.org/tutorials/structured`$_data/feature_columns$

https://github.com/guillaume-lrt/Covid19-retweet-prediction-challenge
hrefhttps://docs.google.com/spreadsheets/d/1B$_4FUW1MnA9V5TLlrCAA8BchYrd1LKKe7zp1YVO07B4/editg$
$0Results$

A Appendix



6cm

Figure 1: Multivariable Linear Regression

Listing 1: Linear Regression code

```
y_pred = modelLR.predict(x_eval)

y_pred1 = np.where(y_pred<0, 0, y_pred)
y_pred2 = np.rint(np.sqrt(y_pred1))
print(y_pred2)

diff = np.absolute(y_pred2 - y_eval)
error = 1/ y_eval.shape[0] * np.sum(diff)
print(error)
```