Team Batiki

COVID19 Retweet Prediction - INF554

ALBUQUERQUE SILVA Igor
igor.albuquerque-silva@polytechnique.fr
GALVÃO LOPES Aloysio
aloysio.galvao-lopes@polytechnique.fr
MAIA MORAIS Lucas
lucas.maia-morais@polytechnique.fr

December 2020

1 Feature Selection

We started our work by analyzing histograms and distributions to have a better understanding of the data and what our models had to do. Our first realization was that the dataset is extremely unbalanced. We tried to create a histogram with the retweet count in the axis `y` and bins in the axis `x`, even in a logarithmic scale the group with almost 0 retweets dominated completely. Around 60% of the dataset has no retweets. This proved to be probably the greatest challenge about this dataset, which also made very difficult to apply approaches that involved neural networks. Using the `langdetect` library, we also predicted that about 91% tweets are in english.

We then looked at word maps and created a few features involving the timestamp and textual data to analyze feature importance. These features were created based on intuition, thinking about what elements in a tweet might increase its retweet count, like the number of hashtags, for example.

Table 1 shows their correlation in relationship with `retweet_count`. We can note that `n_followers` has the highest correlation, with 0.13. To have a better feature importance measurement, we trained a random forest model with those features, because it also takes into consideration relationships between variables to predict their importance, differently from correlations. This second measurement gives more importance to `n_followers` and `n_statuses`, with 0.37 and 0.16, respectively.

Table 1: Feature importance measured using correlations and a random forest model.

	n_followers	verified	n_words	n_chars	n_friends	timestamp	n_urls	n_statuses	n_covid	n_#	n_@	hour
Correlation	0.13	0.06	0.04	0.03	0.02	0.02	0.02	0.00	-0.00	-0.00	-0.00	-0.01
Random Forest	0.37	0.00	0.05	0.07	0.09	0.12	0.01	0.16	0.01	0.01	0.00	0.04

1.1 Timestamp

We also used the software Tableau to visually look at variables relationships. The most interesting founding was that there is a gap in retweet count looking at a specific timestamp interval. This seemed promising, however in the middle of the competition we realized that timestamps may contain wrong information. We found cases of tweets that happened in years much before 2020 by searching for their texts manually. Appendix A provides proof of this. There was no way of knowing how many wrong timestamps there were, so using this feature in our models was discouraging because older tweets will naturally have more retweets than newer ones.

1.2 Textual features

Besides from the manually extracted features explained above, we tried multiple tweets embeddings. In this section we'll explain each one of those, and in the Models section their performances will be explained further.

1.2.1 TF-IDF

Using the library `texthero` we cleaned and removed stopwords from each tweet and created TF-IDF vectors (first for a limited sample of the dataset, then limiting the size of the vectors to 100) and did an analysis on these vectors. This analysis consisted of a KMeans dividing the dataset in 5 clusters, then analyzing feature correlations within each of these groups. The goal with this analysis was to use textual features to divide the dataset into groups, hoping that predicting retweet count within each of these groups would be easier. We also utilized PCA to project the word embeddings in two dimensions to visualize the groups and try to find patterns.

Since we used subsamples of the dataset to analyze these groups the results varied a lot, but we found up to 0.3 variation in the correlations of `n_followers` from two different clusters. Appendix B contains more details on this analysis.

1.2.2 GloVe

The official **GloVe website** [3] contains pre-trained word vectors from uncased tweets in many dimensions. We downloaded these word vectors of size 25 and 300 to use them as features. However, these tweets aren't from a COVID-19-related corpus. We minimized out-of-vocabulary words by clearing our text of the symbols that weren't in the vocabulary and expanding word contractions. The word "covid" wasn't present in the vocabulary, so we created it by taking the mean of synonym words like "coronavirus", "cancer" and "virus", to make sure we're taking the most out of this embedding. With this preprocessing, 57.38% of unique words are in the vocabulary, which consists of 98.92% of all text.

1.2.3 BERT

BERT is a neural network originally published in 2018 and is today one of the state of the art models form NLP tasks. From the first moment we thought that using BERT for our NLP tasks would bring very good results. We tried initially to use **bert-base-cased** pretrained model from **huggingface** [1], then we tried to fine tune this model on our dataset by adding an extra layer and training it, but we were unsuccessful, even after trying to rebalance the dataset. We found BERTweet [2], a pretrained model that includes twitter data from the COVID-19 period, this way we decided that it was appropriate to use this model to embed our the textual data in our model. This approach proved later to be the best one we found to select textual features, thus it's used in the final model.

1.3 Conclusions

We concluded that the most relevant features for retweet prediction are the numerical features provided directly in the dataset. We spent almost all of our time initially trying to optimize our regressor only based in numerical features, as any text-based approach to produced poor results. This way we decided to apply an initial regressor that would output potential values for retweet count based on numerical features and then use the text to further improve our results. This approach proved to be be the most successful in the end. We highlight that in the final model the BERT embeddings were used. It's also important to say that the process of model creation and feature selection happened simultaneously and iteratively, this way, every time we had a new idea, new ways of selecting features were used and new models were tested.

2 Models

2.1 GloVe-LSTM

One of the first models we tried was using the GloVe embeddings to train an LSTM-based neural network. It used two bidirectional LSTM layers concatenated with numerical features and it was implemented in Keras, with a few different variations tried. This model wasn't able to predict large retweet count values because of the dataset imbalance. Under-sampling and up-sampling techniques were tried, as well as predicting the logarithm of `retweet_count`, classifying in powers of 10 or classifying in 0 or 1, which slightly improved the results, but was still not very good.

2.2 Classical machine learning approaches

We realized that that any method involving neural networks would not be very effective as we had a very imbalanced dataset with a very small number of data points for the most popular retweets. These constraints are very problematic for neural networks. For this reason, we decided to try different methods. One of our main bets was a KNN Classifier as it's very robust in this imbalanced scenario, we also tried to use other methods present in **scikit-learn** [4] such as decision trees, random forests etc. KNN proved to be the best overall and managed to give us a 152.70 score only using three numerical features: `user_followers_count`,

`user_statuses_count` and `user_friends_count`. We optimized the classifier by varying the number of neighbors and concluded that three neighbors made the best classifier.

2.3 KNN optimizations

The base KNN approach could be considerably improved. The basic idea is that at least one of the nearest neighbors based on the euclidean metric for the numeric features provided a very good estimation for the number of retweets (this was proved by looking at the best possible neighbor and calculating the MAE), but the majority vote used to calculate the prediction wasn't the most optimal. Therefore we focused on coming up with solutions to find the neighbor that best predicted the number of retweets.

2.3.1 The median approach

One of the problems with our original KNN is that we used the base KNN classifier from [4], which uses majority vote. We realized that in fact, to minimize the error in the MAE metric, given a neighborhood, the best prediction is in fact the median on that group. By predicting each data point to the the median of a neighborhood we could consider more neighbors (10 proved to be the best) and also considerably improve our prediction score, which was now 150.39. In fact we showed that if each point in the neighborhood is equally likely the best prediction is the median.

On the other hand we know that each point in the neighborhood is not equally likely because they have different texts and numerical features. After varying the neighbor we found out that the median minus one was the best constant neighbor selection on our dataset. This means that's best to be conservative on the prediction value. This change got a score of 149.84.

2.3.2 The user verified contraction

To include information about the boolean `user_verified` we tried several different approaches based on KNN and the only one that worked was bringing closer the neighbors with the same value from a factor f defined manually. We then disregard the neighbors that are far by just considering a smaller number of closest neighbors. This allowed us to improve our prediction adding information about this variable, lowering about 0.1 our score.

2.3.3 The text selection approach

As using the median proved to improve considerably our prediction score we decide to also consider the text to select a better neighbor. To introduce textual features we took our best embedding, the BERT embedding.

For each point in the training set we calculated the best neighbor prediction (excluding itself) and we trained a GBR (Gradient Boosting Regressor) from [4] to calculate the best neighbor based on the numerical features and the text embedding. However we found that reducing the dimension of the text embedding (BERT embeddings are 768 dimensions wide) would improved the performance of this this regressor. This way before passing the text embeddings to the GBR we realized a PCA to reduce their dimension to 2. It's interesting to note that in the estimation of the feature importances on the GBR the score of the first PCA textual feature was around 0.55 while the score of the number of followers was around 0.19, this shows that for the GBR to select the best neighbor the text is in fact more important. This model got a score of 149.53.

2.4 Final approach

Our final approach was to calculate a neighborhood using KNN and the numerical features previously described. Then for this neighborhood, we bring closer the neighbors with the same value for `user_verified` and consider just the closest neighbors. Finally, we predict the best neighbor using XGBoost (to use GPU), feeding it with the numerical features and the two principal components of the text features extracted using BERT. We then optimized the values of the distance factor for the user verified variable, the number of neighbors, the number of closest neighbors and finally for the XGBoost the number of estimators, the learning rate and the max depth. With this, we obtained a final score of 149.29.

Appendix D contains more information on each of our models' performances.

Appendices

A Proof of timestamp incorrectness

For example, tweet 565532 of the train dataset contains the following text: "To all the little girls watching...never doubt that you are valuable and powerful & deserving of every chance & opportunity in the world.", and has 647993 retweets, on a verified account with 27641492 followers. If we search it on google, we find that Hillary Clinton posted the same text on November 9, 2016 (it is actually her pinned tweet), in a tweet which currently has 642.4K retweets, and she has 29.9M followers. It is clear that this is the same tweet, but the timestamp data places this on April 30, 2020.

Other examples like this were also found. We believe that these are retweets of these original tweets, and for some reason the `timestamp` column has incorrect data while the numerical columns contain correct data.

B TF-IDF vectors analysis

Figure 1 shows the KMeans of the TF-IDF embeddings from a sample of the training dataset. These groups are plotted in different colors, and the axis of the plot contain the two principal components of the PCA of those embeddings. We can see that the groups make sense in the plot.

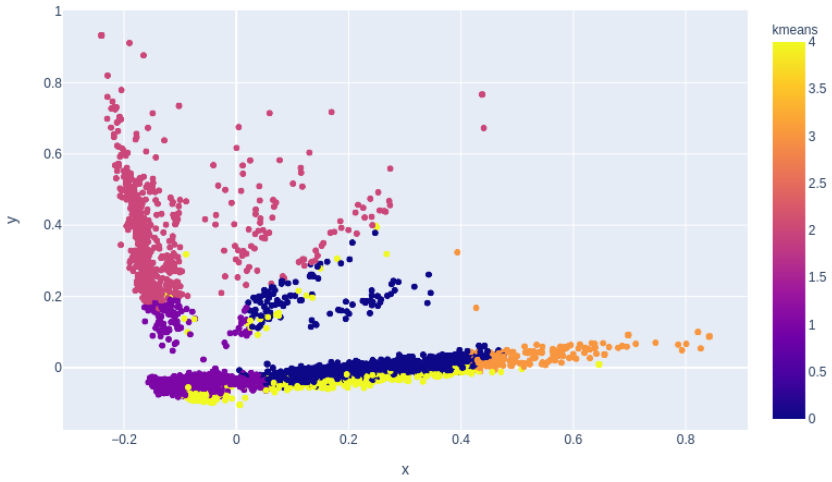

Figure 1: Two first principal components (x and y axes respectively) of the PCA decomposition using TF-IDF features for a sample of the train dataset. Colors represent their KMeans clusters.

C BERTweet results

Before using the results of BERTweet's embeddings we visualized them to verify their use was a promising approach. To do that we decided to use PCA for reducing the the dimension of the data points to 2, PCA is specially useful in the visualization use case as it tries to spread the points. We then colored the points based on the following criteria: first we calculate the two nearest neighbors for each point based just in numerical features (the ones described in the KNN section) then we colored red the points better approached by the neighbor with smaller retweet count and blue otherwise. This gave us an idea if the textual features are a good set of features to choose which neighbor in the train dataset better approximates each test data point. Figure 2 shows this visualization.

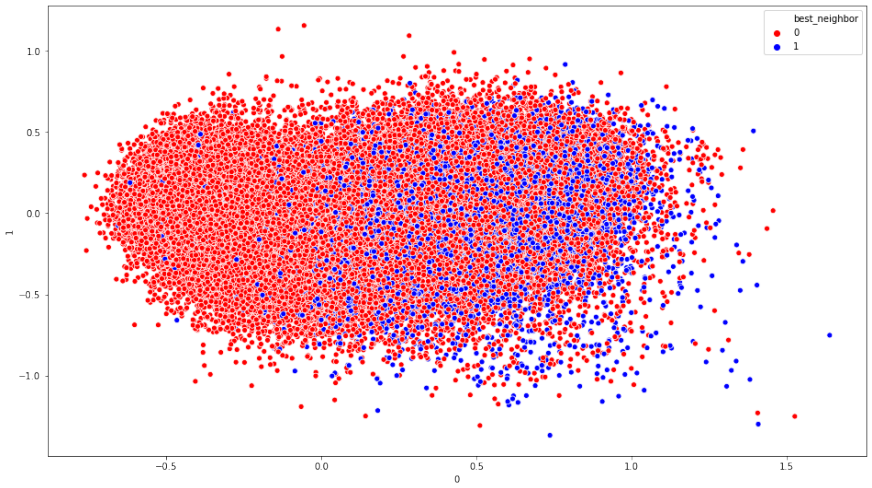

Figure 2: Two first principal components (x and y axes respectively) of the PCA decomposition using BERT features for the test dataset. The points in red are better approximated by the 2-nearest neighbor with smaller retweet count and the points in blue are better approximated by the neighbor with greater retweet count.

We can see that in the first principal component, this feature has a high relationship with the neighbor chosen, since the blue points are generally positioned to the right.

D Models cross-validation

To analyze the performances of our models, as well as to optimize their parameters, we did Monte-Carlo cross-validations. Table 2 shows our main models' results, as well as their comparisons with some classical methods such as linear regression and random forest. In this table we compare the performances of our main models in a performance order which is also the chronological order. The models are described below:

- KNN: in this model we use three numerical features `user_followers_count`, `user_statuses_count` and `user_friends_count` with a 3 neighbors KNN and euclidean distance. It was our first submission and it performed 152.70 in the score.
- Corrected-10-Median: the same as the original KNN but now considering 10 neighbors and taking the median minus one as the prediction. This approach was never submitted as the next one surpassed it.
- Corrected-Median-UV: the same as the Corrected-10-Median, but now getting 40 neighbors using the user verified feature to bring closer some neighbors, selecting the 10 closest ones and finally getting the median minus one. This submission gave us a better score, 149.84.
- Grouped KNN: based on the median value, group the median value of the neighbors based on the closest power of ten, then depending on the group of each median predict the point to a different neighbor, optimized for the group. Even if we got better results on our cross validation, in the actual dataset we performed worse, 150.27.
- XGBoost-KNN: using textual features extracted from BERT+PCA (two first components) and the same numerical features, using a XGBoost regressor to estimate the best neighbor. Our performance was 149.54.
- XGBoost-uv-KNN: integrate the same approach used in the Corrcted-Median-UV to improve the XGBoost-KNN performance. The final score was 149.29.

Table 2: Models MAE's for a Monte-Carlo cross validation.

	0	1	2	3	4	Average
XGBoost-uv-KNN	145.50632	139.22579	137.72118	140.47844	136.37302	139.86095
XGBoost-KNN	145.74780	138.72360	137.92012	140.62056	136.35116	139.87265
Grouped-KNN	145.93770	139.96711	137.74569	140.93580	136.61703	140.24066
Corrected-Median-UV	146.38611	140.08023	138.62018	141.46235	136.73156	140.65608
Corrected-10-Median	146.57085	140.09473	138.60681	141.52308	136.69178	140.69745
KNN	145.86524	139.18615	140.24764	143.34072	138.09455	141.34686
LogGradientBoosting	150.29024	143.07951	142.19235	145.13001	139.56261	144.05095
GradientBoosting	151.26142	144.10285	143.01257	145.65797	140.39442	144.88585
All-zero	155.43425	149.21205	147.99024	150.39015	145.28295	149.66193
RandomForest	235.29432	225.99619	231.22650	234.94854	231.32672	231.75845
LinearRegression	268.36285	260.95524	260.24058	265.35576	258.78022	262.73893

References

[1] *bert-base-cased · Hugging Face*. URL: `https://huggingface.co/bert-base-cased` (visited on 12/10/2020).

[2] Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. "BERTweet: A pre-trained language model for English Tweets". In: *arXiv:2005.10200 [cs]* (Oct. 5, 2020). arXiv: `2005.10200`. URL: `http://arxiv.org/abs/2005.10200` (visited on 12/10/2020).

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation". In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543. URL: `http://www.aclweb.org/anthology/D14-1162`.

[4] *scikit-learn: machine learning in Python — scikit-learn 0.23.2 documentation*. URL: `https://scikit-learn.org/stable/` (visited on 12/10/2020).