Report Project INF554 - Machine and Deep Learning (2022-2023)

Data Challenge : Retweet Prediction | Team : Noobs

Engineering students X2020 :

Jamil Alhussein
alhussein.jamil@polytechnique.edu

Emmanuel Gnabeyeu
emmanuel.gnabeyeu-mbiada@polytechnique.edu

Youssoufa Talba
mamoudou.talba-youssoufa@polytechnique.edu

Professor :

Michalis Vazirgiannis

1 Project Description

The goal of this data challenge was to study and apply machine learning/artificial intelligence techniques to a real-world regression problem. The mission was to build a model that can accurately predict the number of retweets a tweet will get.We used the Twitter dataset with the 2022 French presidential election as the central topic.

For each tweet in the test set, our model should predict the number of retweets it will get after its publication. ‘

The evaluation metric is the Mean Absolute Error (MAE) calculated by dividing the sum of absolute differences between the predicted number of retweets (p_i) and the observed number of retweets (a_i) by the number of observations (N), i.e., $MAE = \frac{1}{N}\sum_{i=1}^{N}|p_i - a_i|$

2 Feature Selection/Extraction

2.1 Data cleaning

We dropped some columns that were not very usable : ’urls’ and ’mentions’. In order to use the hashtags, we merge texts and hashtags, expecting it will add keywords, before dropping the ’hashtags’ column (notice that hashtags are present only in less than 18% of the dataset).

2.2 correlation matrix

We did a statistical correlation analysis (for Feature Selection) by using Scatter Plot and Correlation matrix which allow us to consider the most discriminative features to attempt our regression task.

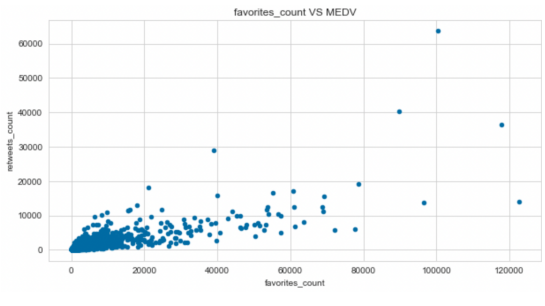

FIGURE 1 – scatter plot

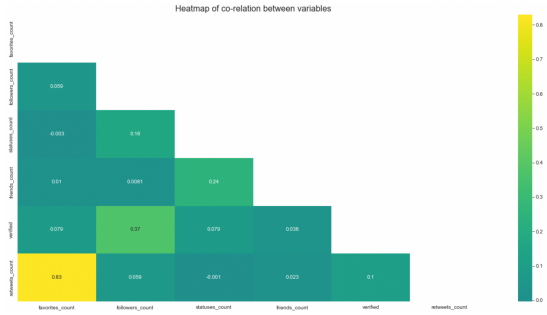

FIGURE 2 – correlation Matrix

2.3 Text processing

2.3.1 TF-IDF

TF-IDF is a composite score representing the power of a given word to uniquely identify the document. It is computed by multiplying Term Frequency(TF) and Inverse Document Frequency(IDF) TF. After cleaning our text + hashtags by removing unecessary symbols and caracters, we did a TF-IDF vectorialisation of text and hashtag with 100 as max_features. This gave us 100 more features and we then further reduce the number of feature aither by Principal component analysis(PCA) or by K-means clustering.

2.3.2 Transfer learning with Word2Vec

One could use pretrained models in order to convert text into numerical data. This allows us to focus on other tasks such as : classification, sentiment analysis, etc. We used Google’s Word2Vec

as our choice in Text vectorization (transfer learning). After converting each word in our corpus in vector of shape (300,1) , we next produced a single vector for each text sample by taking the mean of vector representation of each word present in the tweet. We believe that this single vector carries valuable information about the tweet. To this vector we could apply additional treatement(k means, pca etc.)

2.3.3 candidate

One interesting part of a tweet is its subject. In our case this could be the candidate or the country it's talking about. Considering a tweet, we could extract proper names first, then categorize these tweets based on those names, taking into consideration possible spelling mistakes that is : "Macron" and "Marcon" should go in the same category. Eventually, each tweet gets assigned a number which designates the candidate, the number being -1 if we couldn't find any proper names inside the tweet.

2.3.4 sentiment analysis

We thought that it might be also interesting to add sentiment of a text as an additional feature. For this we use **Textblob** package that would tell us if a tweet is **neutral, positive, negative** we then assign numerical values to these sentiments to be used later in our models.

2.4 Dimensionality Reduction using PCA and Standardization

Word2Vec and TFIDF give us 300 and 100 features respectively. In order to avoid possible high dimensionality problems as well us to speed up the training process, we decide to use a PCA analysis.

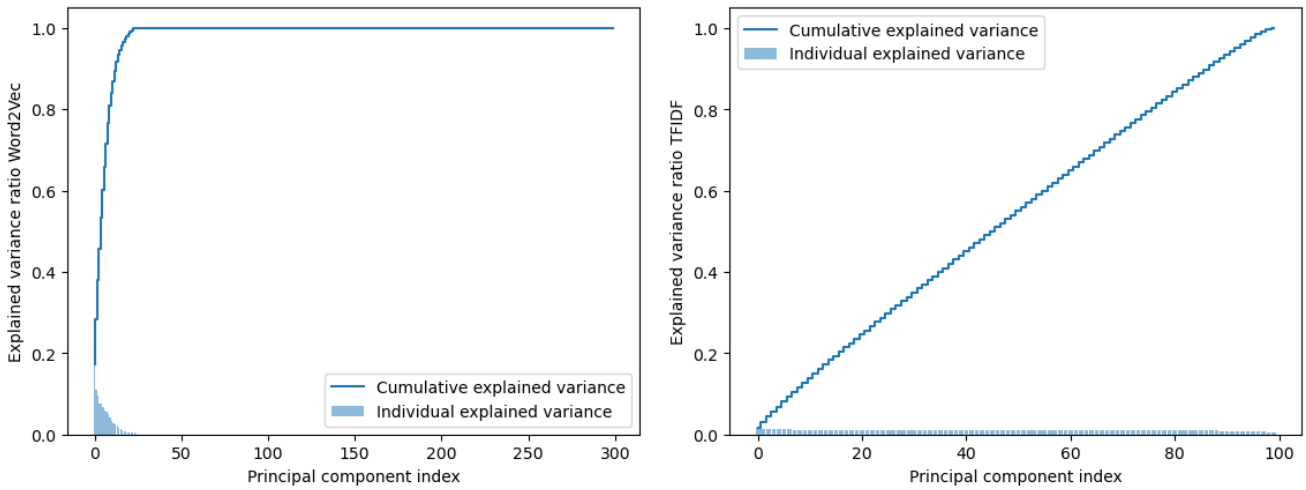

FIGURE 3 – PCA analysis of word2vec(left) and of TFIDF(right)

This figure tells as that using 25 components is enough for word2vect while PCA doesn't seem very suitable for TFIDF components. We decide to only keep 25 as well. Finally we apply a normalization by standard deviation to our numerical data.

2.5 Dimension reduction by Unsupervised learning

After the vectorialisation using either TDIDF or Word2Vec, we tried to train a K-means clustering in order to group our text data in several categories, then reducing 100 features to one categorical feature and next step, we tested several model. We used the Elbow method to select the optimal number of cluster.

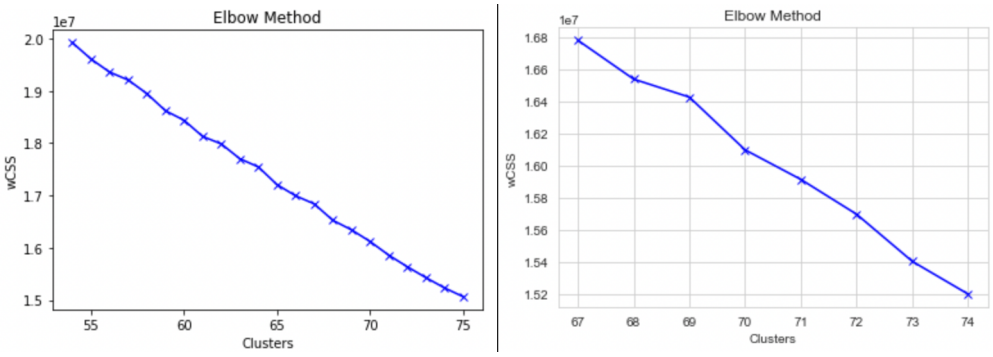

FIGURE 4 – Elbow method for K-means clustering

3 Model Choice, Tuning and Comparison

We didn't know at first which model was most suitable : We decide to test as many as we can starting from very basic models to complex ones. For some of these model in general, we tune hyperparameters by using a Grid Search of scikit learn model selection.
Also, we started by using only numerical data with no text processing. For each iteration, where we would create new features we would try and check if adding these features led to a better model using Net.ML a framework that allows to test many regression models and give back the best choice. However this solution wasn't very usable since some of the models were developed for the framework itself.

The results of basic models only using numerical features : Then by adding text processing and

	MSE	MAE	MSLE	R2	Model
0	8364.878058	6.400095	0.259403	0.945214	RFRegressor
4	8007.420985	6.699376	0.364185	0.709554	MLPRegressor
2	8956.462569	6.891657	0.321614	0.812682	KNeighborsRegressor
1	8143.490857	7.128499	0.776982	0.891920	GradientBoosting
3	9766.441500	9.646227	1.174634	0.686163	LinRegressor

	MAE	MSE	MSLE	R2
RFRegressor	6.464326	8271.589146	0.264852	0.943812
MLPRegressor	7.202093	11896.252809	0.328068	0.722976
KNeighborsRegressor	7.345764	11133.638604	0.323445	0.818254
xgboost	7.371401	16376.006660	0.250650	0.982558
GradientBoosting	8.111548	17873.699932	0.760662	0.893092
LinRegressor	10.072526	13813.459587	1.112888	0.693644

FIGURE 5 – Basic Models without(left) and with(right) textprocessing

using a more complex model namely a neural network having two hidden layers of 20 neurons each and relu activation functions we were able to get to around 2.9 of MAE loss on the training dataset and around 6.4 on Kaggle.

4 Conclusion and Prospects

Feature extraction from text comes as a very intricate and time consuming process. The fact that tweets were in french only made this worse since most existing data processing packages work only with English.
Finding the correct model wasn't a breeze either : There are just too many available models to choose from and most take way too long to train.
When it comes to neural network a huge difficulty was finding the right architecture that wasn't too complex nor too naive : while optimizers allowed us to avoid learning rate adjustment problems in most architectures, we had a rough time finding the right number of layers and the right activation functions to be deployed.
We also noticed that a very long training time sometimes meant great results on the training set and terrible ones on the test set, we had to try different number of epochs to get the right amount of training.