Tweet Classification Challenge

Giannis Nikolentzos and Michalis Vazirgiannis

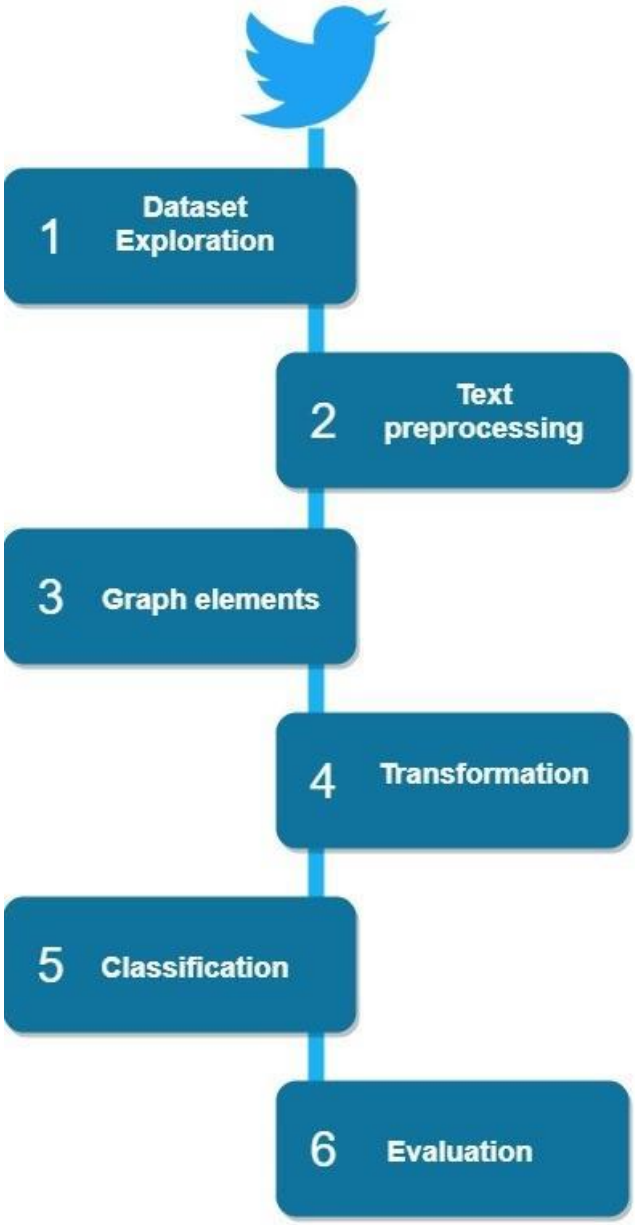
Fragkiadakis Menelaos **p3351822**
Rallis Panagiotis **p3351816**

Part time 2018-2020

Introduction

Twitter is one of the platforms widely used by people to express their opinions and showcase sentiments on various occasions. Sentiment analysis is an approach to analyze data and retrieve sentiment that it embodies.
The tweet format is very small, which generates a whole new dimension of problems like the use of slang, abbreviations, etc. This article reports on the exploration and preprocessing of data, transforming data into a proper input format and classify the user's perspective via tweets about Covid19 pandemic by building supervised learning models using Python and NLTK library.

Quick Workflow:



1 Dataset Exploration

A sample of the tweets is depicted. As we can observe the text consists of dictionary words, slang, emojis, hashtags, URLs, etc.

```
In [25]: print(posts[0])
```

```
Since everyone is on #lockdown because of #COVID19, I want to spread some positivity and surprise some people! 😈 I am giving five people that retweets this a free #AnimalCrossing Nintendo Switch Bundle, must be following me for a DM! 🎁 Good luck &amp;amp; ! ❤️ #ACNH https://t.co/sIAkfuxZhK
```

```
In [38]: print(posts[6])
```

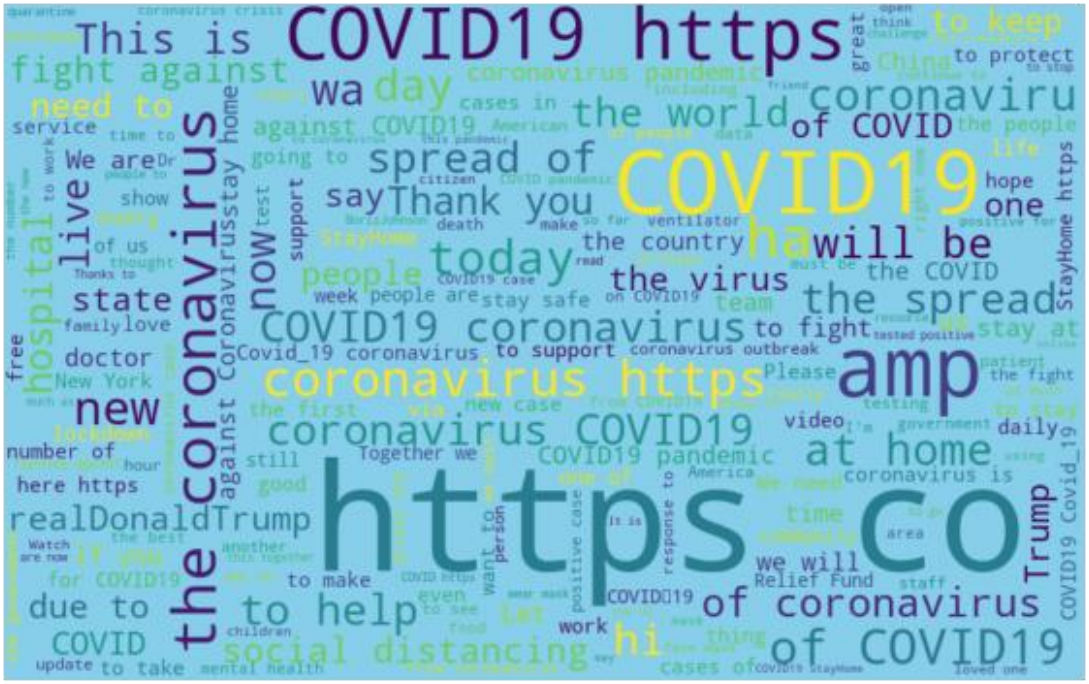
```
Once @BorisJohnson @MattHancock have finished their #COVID19 stint in isolation, I highly encourage them to join us in hospital + spend 12 hours in PPE (or sometimes paucity of) to see the lengths healthcare workers go to keep the  running and care for patients.
```

```
In [39]: print(posts[7])
```

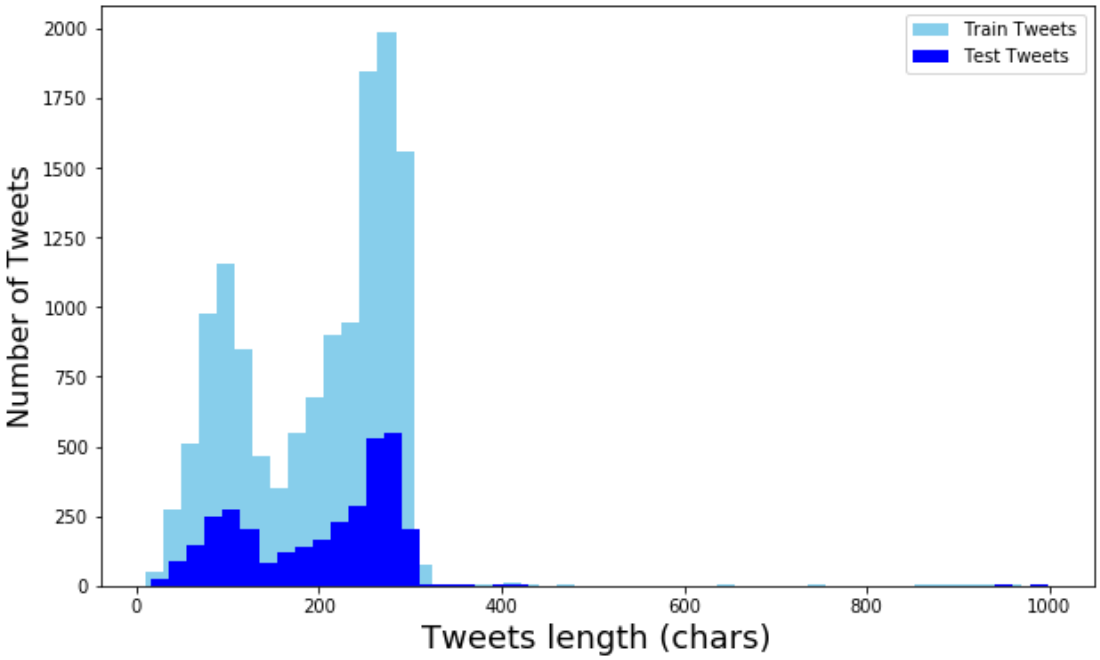
```
Communal @RSSorg are always in the forefront to strengthen country in it's fight against any crisis.  Swayamsevaks distributing grocery items, cooked food across the country during this difficult time of #Covid19.  #FeedTheNeedy is the foundation principle.  https://t.co/ViaIqgM0Bs
```

To examine how well the given sentiments are distributed across the training dataset, one way to accomplish this task is by understanding the common words by plotting word clouds.

A word cloud is an image made of words that together resemble a cloudy shape. The clouds give greater prominence to words that appear more frequently in the source text.



Below is depicted the length of the Tweets in our Train and Test data



2 Text preprocessing

The main issue with text data is that it is all in text format (strings). However, Machine learning algorithms need some sort of numerical feature vector in order to perform the task.

2.1.1 Basic text pre-processing

1. Translate tweets and words in English
2. Remove http(s) urls and emails
3. Remove Twitter user tags(@user)
4. Remove Html tags (> etc)
5. Remove Punctuations and Special characters
6. Replace emojis with descriptive text
7. Divide words in segments, especially for hashtags (#DonaldTrump to Donald Trump)
8. Replace phone numbers
9. Remove stop words in English

2.1.2 Tokenization

It is just the term used to describe the process of converting the normal text strings into a list of tokens i.e. words that we actually want. Sentence tokenizer can be used to find the list of sentences, and Word tokenizer can be used to find the list of words in strings.

2.1.3 Stemming

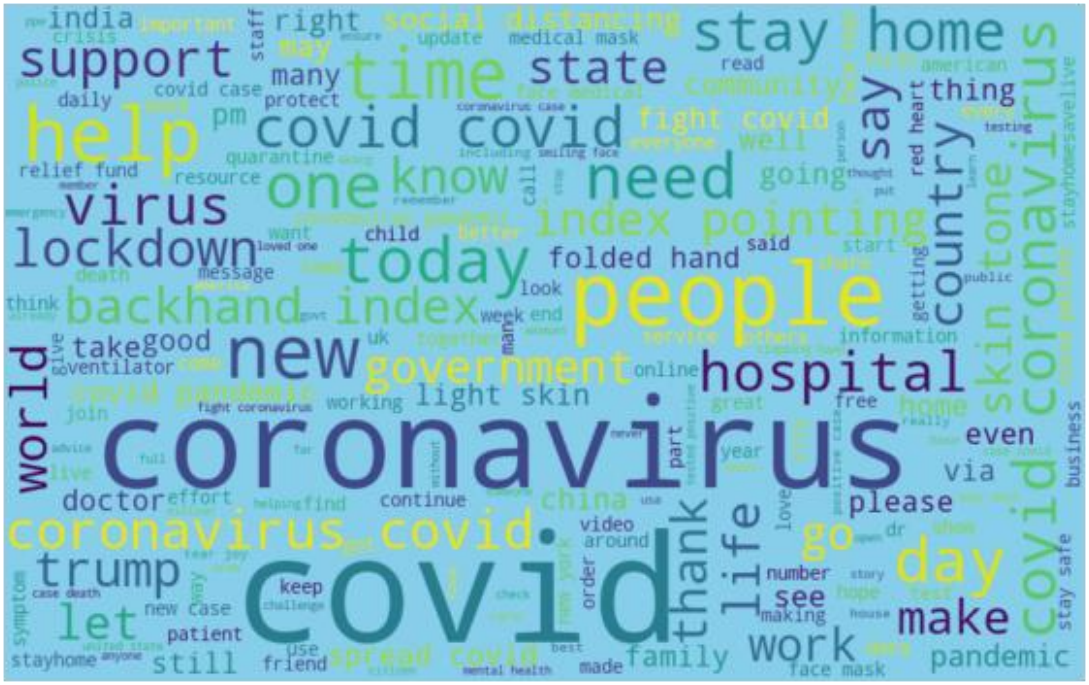
Stemming is the process of reducing inflected (or sometimes derived) words to their stem, base or root form — generally a written word form. For example, if we were to stem the following words: “Stems”, “Stemming”, “Stemmed”, “and Stemtization”, the result would be a single word “stem”.

Having completed all the aforementioned processes we end up with the following results regarding to initial metrics:

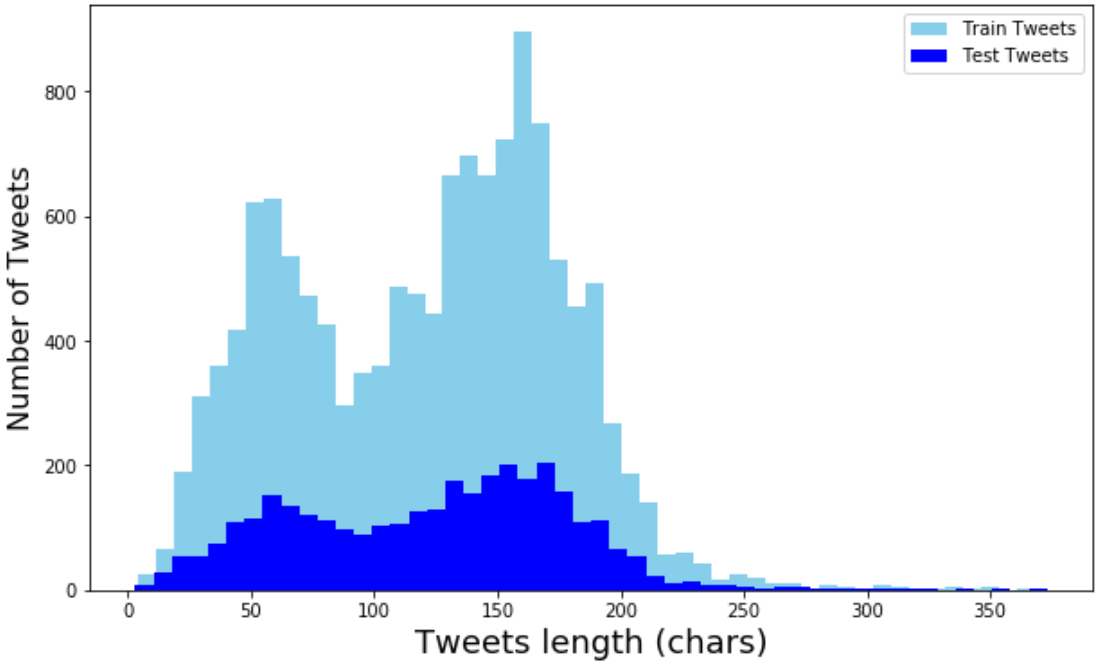
The same three sample tweets after data cleansing

```
In [40]: print(clean_str(twitter_posts[0]))

since everyone lockdown covid19 want spread positivity surprise people smiling face number heart giving five people  tweet  fre
e animal crossing nintendo switch bundle must following  wrapped gift good luck  red heart acnh

In [41]: print(clean_str(twitter_posts[6]))

boris johnson matt hancock finished covid19stint isolation highly encourage join  hospital spend number hour ppe sometimes pauc
ity see length health care worker  keep running care patient

In [37]: print(clean_str(twitter_posts[7]))

communal rss org always forefront strengthen country fight crisis swayamsevak  distributing grocery item cooked food  cross cou
ntry difficult time covid19feed the needy foundation principle
```

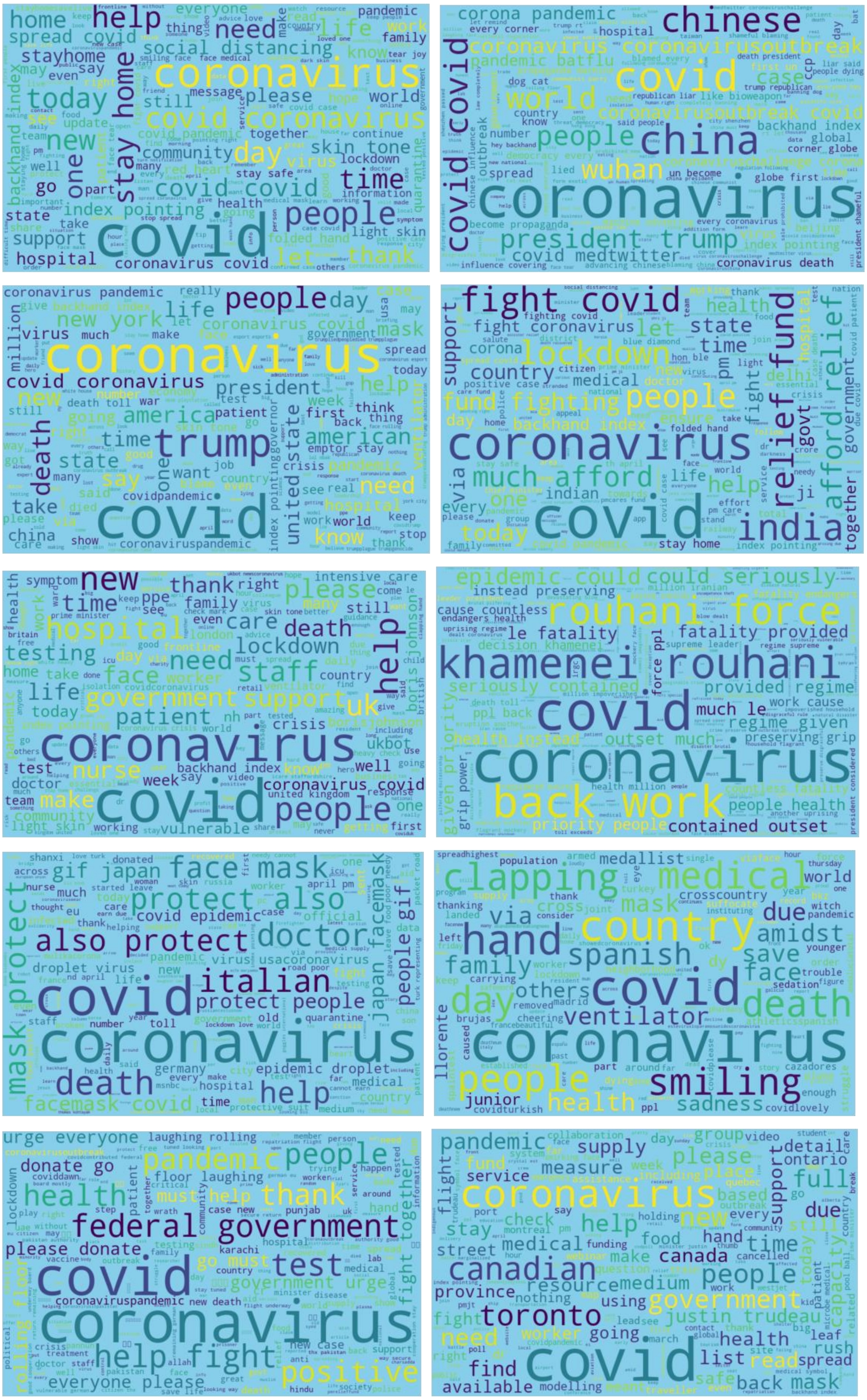
In addition, below is presented the corresponding word cloud

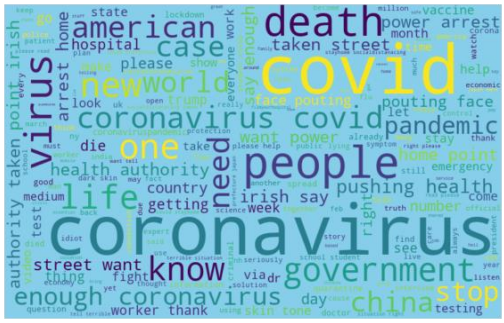


As expected the length of each tweet has decreased significantly after we have implemented the text preprocessing.



Below, a word cloud per category


home help everyone pandemic
need life work family
social distancing stayhome
coronavirus
today stay home world
covid coronavirus
new community day skin tone
one time
index pointing people
covid
support hospital let thank
corona pandemic chinese
coronavirus coronavirusoutbreak
pandemic batflu covid case
world
people china
wuhan
covid covid
coronavirus
president trump
covid medtwitter
coronavirus death
coronavirus pandemic people day
new york life
coronavirus
covid coronavirus president help
america
trump american
death state
need
covid
index pointing united state
china know
fight covid health
fight coronavirus state
lockdown time
country fund
fighting people
relief fund
coronavirus
much afford
covid india
today
new thank right
please
hospital care death
testing need lockdown help
face staff
government support uk
coronavirus
nurse backhand index
covid people
make community
epidemic could could seriously
rouhani force
khamenei rouhani
covid
coronavirus
back work
contained outset
gif japan face mask
protect also
also protect doctor
italian
covid protect people
coronavirus
death help
facemask covid
clapping medical
country
hand spanish
family covid
day face
death
coronavirus
people smiling
health
urge everyone
pandemic people
health thank
federal government
please donate test
covid
coronavirus
help fight
positive
pandemic supply
please
coronavirus
help
canadian
people
government
toronto
covid
need



3 Graph Elements

We create two graphs from the **retweet_weighted.edgelist** and **retweet_processed_weighted.edgelist** accordingly in order to engineer some additional features that will help our classification process.

Specifically from the first graph, we extract the

- Page rank
- Betweenness centrality
- Closeness centrality

The feature of betweenness centrality was not included at the final train set while it did not improve the accuracy of the algorithm.

From the second graph we use

- The partition attribute using the community louvain module
- The Degree of each node
- The posts per class per user for the neighbors.

3.1.1 Deepwalk

Using the Deepwalk implementation of the lab with the number of walks = 20 and length of walks =20 there were created the word 2 vectors of size 128.

4 Transformations

4.1.1 Extracting Features from Cleaned Tweets

TF-IDF stands for Term Frequency–Inverse Document Frequency: In a large text corpus, some words have high frequency but very little meaningful information. If we use the count data directly to train a model, the very frequent tokens with little meaning will have a significant influence over the low frequency but far more interesting terms. So it is necessary to normalize and re-weight the word count result.

After several experiments with different sizes of features, the optimal choice was 12000.

```
Train matrix dimensionality:  (13219, 12000)
Test matrix dimensionality:  (3306, 12000)
```

4.1.2 Text-Based Features

In addition, the following text-based features were generated from the posts

- Total number of characters in the tweet including spaces
- Total number of words in the tweet
- The average length of the words used in the essay
- Total number of hashtags in the tweet
- Count of numbers in the tweet

4.1.3 Doc2vec

Using gensim library, we generate doc2vec embeddings for the tweets of our dataset

To summarise, the used features for our model were a concatenation of TF-IDF scores, Graph related attributes, the word 2 vec generated from random walks and finally the doc2vec.

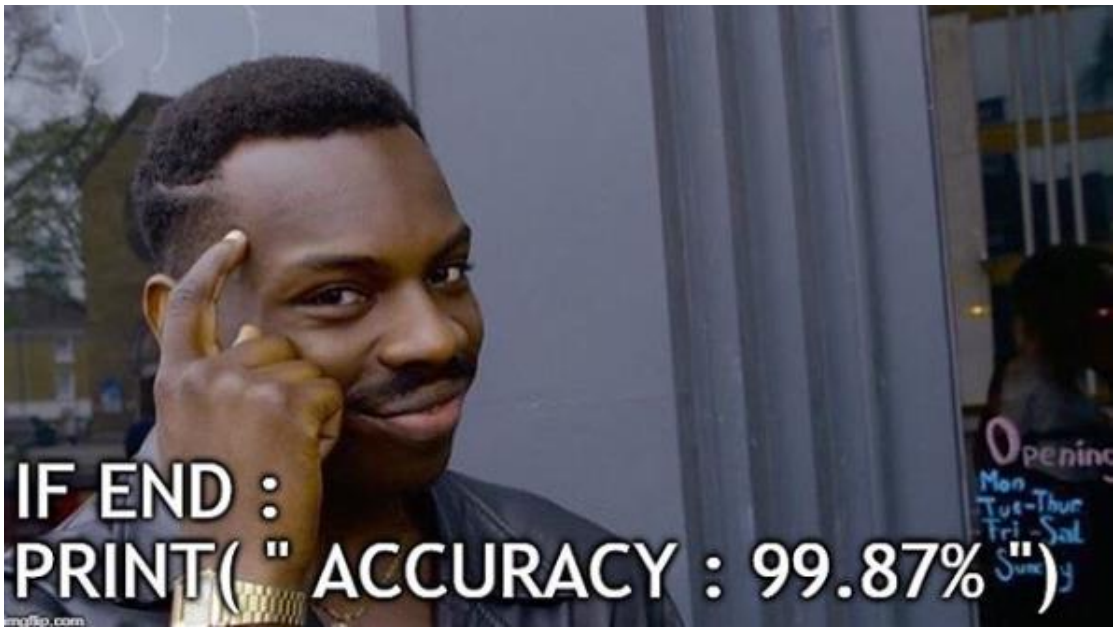
5 Classification

Having used a plenty of different implementations on several classifiers we end up with the Logistic Regression classifier which achieves the most accurate score with a **liblinear** solver.

Our final choice for the aforementioned classifier came from the below facts:

- Simple implementation
- Fast training time
- Better F1 score
- Better multiclass loss

6 Evaluation



We need to evaluate our algorithms against some criteria. So the criteria that we have considered to measure the performance of the implemented models are Confusion Matrix, Accuracy, and F1 Score.

- Confusion matrix usage to evaluate the quality of the output of a classifier. The diagonal elements represent the number of points for which the predicted label is equal to the true label, while off-diagonal elements are those that are mislabeled by the classifier. The higher the diagonal values of the confusion matrix the better, indicating many correct predictions.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

-
- F1-score: This is the harmonic mean of Precision and Recall and gives a better measure of the incorrectly classified cases than the Accuracy Metric.

$$\text{F1-score} = \left(\frac{\text{Recall}^{-1} + \text{Precision}^{-1}}{2}\right)^{-1} = 2 * \frac{(\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

The below results were predicted using a logistic regression with liblinear solver and C = 3.8

```
train f1-score: 93.08%
test f1-score: 72.32%

test data confusion matrix
Predicted    0   1    2    3    4   5   6   8   9   10  11  12  13  14
True
0         1228   0   79   49   40   0   0   0   0   0  10   0   1   4
1           29  93   30    6    3   0   0   0   0   1   0   1   1   0
2          127  10  417    7    6   1   0   0   0   6   2   4   0   1
3           42   1    3  281    1   0   0   0   0   0   1   0   0   1
4          101   1   12    1  144   0   0   0   0   1   1   0   0   2
5            6   1    9    0    0  32   0   0   0   0   0   0   0   0
6           12   1    5    2    2   0   4   0   0   0   1   0   0   0
7            5   0    1    0    0   0   0   0   0   0   0   0   0   0
8            6   1    1    4    1   0   0  10   0   0   0   0   0   0
9           15   2    2    0    0   1   0   0   4   0   0   0   0   0
10          47   5   39    1    6   0   0   0   1  22   2   1   0   0
11          15   1    7    3    1   0   0   0   0   0  59   0   0   0
12          17   2   21    3    1   0   0   0   0   5   0  30   0   1
13          14   0    1    0    0   0   0   0   0   0   0   0  39   0
14          49   0    3    0    5   0   0   0   0   0   0   0   0  28
              precision    recall  f1-score   support

           0       0.72      0.87      0.79      1411
           1       0.79      0.57      0.66       164
           2       0.66      0.72      0.69       581
           3       0.79      0.85      0.82       330
           4       0.69      0.55      0.61       263
           5       0.94      0.67      0.78        48
           6       1.00      0.15      0.26        27
           7       0.00      0.00      0.00         6
           8       1.00      0.43      0.61        23
           9       0.80      0.17      0.28        24
          10       0.63      0.18      0.28       124
          11       0.78      0.69      0.73        86
          12       0.83      0.38      0.52        80
          13       0.95      0.72      0.82        54
          14       0.76      0.33      0.46        85

    accuracy                           0.72      3306
   macro avg       0.76      0.48      0.55      3306
weighted avg       0.73      0.72      0.71      3306
```

and correspond to our best submitted score on Kaggle's leaderboard.

Conclusion

The final submitted implementation is a result of numerous different approaches on the data preprocess, on the feature engineering both textual and graph features. Also the final choice of the classifier and its parameters was a continuing recurring process that ended with the 7th place in our leaderboard.

The paragraphs below present the additional techniques and approaches that we used, however, the score did not improve.

Additional Implementations

On this part, it will be described briefly, the additional implementations for each part of the workflow.

1 Preprocess

1.1 Language

One of our major concerns about the data was related with the tweets in other languages. Using a language detection library, we realized that there are some tweets which contain words in other languages than english.

```
'af', 'ca', 'cy', 'da', 'de', 'en', 'es', 'et', 'fi', 'fr', 'hr',
'id', 'it', 'lt', 'nl', 'no', 'pl', 'pt', 'ro', 'sk', 'sl', 'so',
'sq', 'sv', 'tl', 'tr', 'vi'
```

We try several experiments like:

- totally remove these tweets
- add one more categorical feature with the language code of tweet
- remove stopwords in all languages not only in english

All the above tests performed less accuracy than the translated tweets and were not added at the final implementation.

1.2 Hashtags

When it comes to hashtags, we thought that an n-gram will be performed different if we replace the symbol # with the word “hashtag”. However, the final model was less accurate than the simple removal of the hashtag symbol.

1.3 Most Common Words

Looking at the word cloud images above it is clear that some words like covid, coronavirus, pandemic etc, are very common in all classes. Therefore, one of our experiments performed removing the 5 most common words. This approach did not help the classifier which achieved a lower score than the dataset with the most common words. This behaviour was expected while the TF IDF score of most common words is lower than other most important words.

2 Features

2.1 Pre Trained embeddings

Firstly, using the pre trained embeddings of **Glove** 50 dimensions as a dictionary, we created a vector for each tweet according to the value of each word in Glove and we concatenate the extra feature to the main dataset (tfidf 3 grams - graph - deep walk - embeddings). There was not any increase in accuracy using a logistic regression or an SVM.

After that, we multiply the pre-trained embedding of each word with the according tf-idf score but the result was the same.

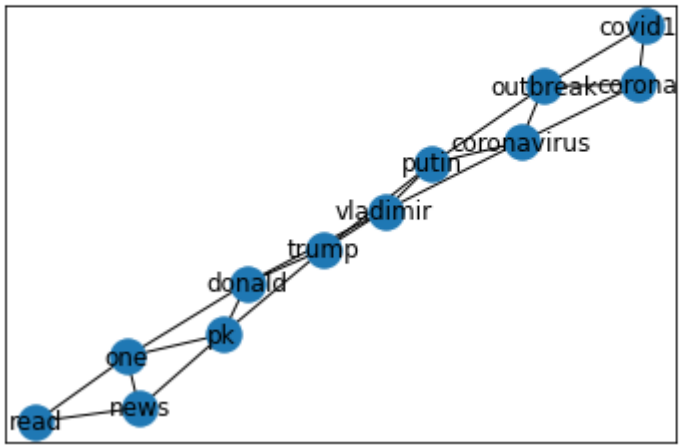
Finally, we created our own pre-trained embeddings of size 50 using the `Word2Vec library` with no increase of the prediction too.

2.2 Graph of words

In this implementation, we transform the tweets to graph of words using the **Grakel** library. Applying only the new features in an SVM algorithm, it achieved an f1 score around 0.6538. Combining the n-grams, the feature from graph, the feature from deep walk and the graph of words, the most accurate algorithm was the logistic regression, which achieved a score near to 72%, but still lower than the Multi-Class loss on Kaggle, without the graph of words.

```
print("Example of graph-of-words representation of document")
nx.draw_networkx(G_train_nx[1], with_labels=True)
```

```
Example of graph-of-words representation of document
```



3 Algorithms and Architectures

3.1 Logistic Regression

This algorithm was tested using grid search in order to find the best solver and the most appropriate logspace. In order to achieve an f1 score near to the score of logistic regression (around 72%) with liblinear solver, using the solvers “lbfgs” and “newton-cg”, we perform a normalization on data produced by the graph and the deep walk. Both solvers performed about similar accuracy with the liblinear solver but in much more computational time.

3.2 SVM

This algorithm was tested using rbf and a linear kernel. Also in this implementation, it is necessary to normalize the features of graph and deep walk. The computational time of these implementations was higher than in logistic regression and the f1 score lower than 72%.

3.3 Smote and Class Weight

Looking at the classified tweets of the training dataset, it is clear that there is strong unbalance among the 15 classes. In order to overcome this issue, we implement an extended dataset using smote and a class weight dictionary. These implementations were checked in every particular run of the above algorithms, combining them or with separate runs.

While the train models achieved to predict items of class 7, which contains only 6 tweets in the training dataset, the f1 score of the model and the multi-class loss on kaggle were lower than the logistic regression without smote or class weights.

3.4 Convolutional Neural Network

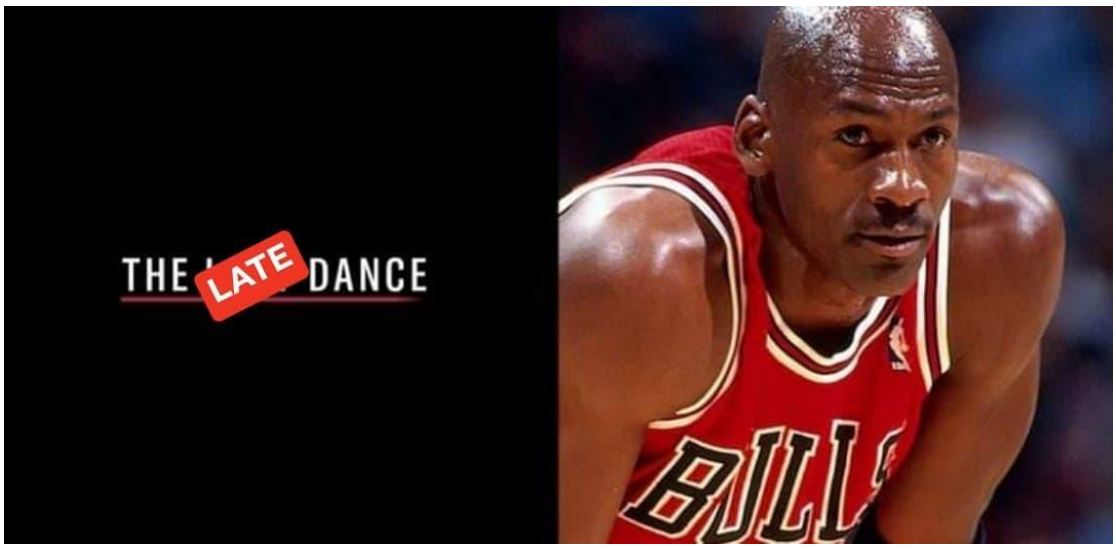
Using the pre trained embeddings of Glove 50 dimensions and the vectorized tweets of 100 length, we created a Convolutional Neural Network of 1 dimension, with 128 filters and 2 hidden layers, relu activation function, max pooling, one extra dense layer of size 256 with relu activation function and an output layer with softmax. Among each layer we performed a dropout 0.2.

We tried to train our neural network for 50 epochs, but the validation accuracy was stopping to increase at 67% around epoch 18.

In this point we have to specify that we were not able to apply an architecture, combining the information of text and the feature of graph and deep walk.

3.4.1. Looking deeper at the length of each tweet, we realize that most of them contained less than 100 words, except a small percentage with length over 80 words. In order to handle this issue, without losing any useful information, we performed an SVD to reduce the dimensionality of vectorized tweets. However, the neural network returned an `-inf` accuracy in each epoch, probably due to a wrong implementation.

4 Ensemble Methods



While the F1 scores of Logistic Regression and SVM were really close, we implemented an ensemble architecture using the Voting Classifier, combining the aforementioned two classifiers and the average predicted probabilities (soft vote) to predict the class labels. This implementation performs a lower F1 Score than the logistic regression (71.32%), but a slightly higher Multiclass loss score on leaderboard. Unfortunately, this submission performed some hours after the deadline and we lost the opportunity to climb one position higher.

Submission and Description	Private Score	Public Score	Use for Final Score
VotingClassifier_submission (2).csv 19 hours ago by Panagiotis Rallis add submission details	0.88566	0.91019	☐