Coronavirus Tweets Classification

Iordachescu Anca (407), Chirut Veronica (407), Caprita Catalin (407), Rotaru Codrut (406)

Introduction

The ongoing global pandemic of COVID-19 has been present in many aspects of our lives in the last few years, including the field of Natural Language Processing (NLP), permitting for new case studies for **stance detection algorithms**, which deal with a subject's reaction to a claim made by a primary actor. Since this pandemic has been the topic of many discussions with differing (and often opposing) points of view among citizens all over the world, we have decided that it would be an appropriate subject to address in this project.

Description of the dataset

The **CoronaVirus tweets NLP** created by Aman Miglani[1] consists of two comma-separated values (CSV) files that describe the training dataset that contains 41157 samples and the test dataset that contains 3798 samples.

The training and test data is provided in the following format:

UserName	ScreenName	Location	TweetAt	OriginalTweet	Sentiment
6682	51634	State of Florida	18-03-2020	Consumer Alert CFO Floridians Must Beware of Fraud amp Scams Read more	Extremely Negative
16701	61653	NaN	21-03-2020	New Podcast Las Sobras Queens of Podcasts ft JuelzTheKing amp V on	Neutral
24162	69114	NaN	25-03-2020	Soap vs hand sanitizer What is it Professor explains what best kills the	Positive
37806	82758	Italy	08-04-2020	are aiming to take advantage of fears over as a means of conducting attacks	Negative
42403	87355	NaN	11-04-2020	Calls for stamp duty holiday as UK house prices are set to fall	Positive

Each line represents a tweet with one polarity category and relevant information, each separated by a space character:

- The first column shows encoded username;
- The second column shows encoded screen name;
- The third column shows location of the tweet;
- The fourth column shows the date of the tweet using short date format;
- The fifth column shows the tweet text;
- The second column shows the manually tagged label for the sentiment analysis classification task.

Exploratory Data Analysis

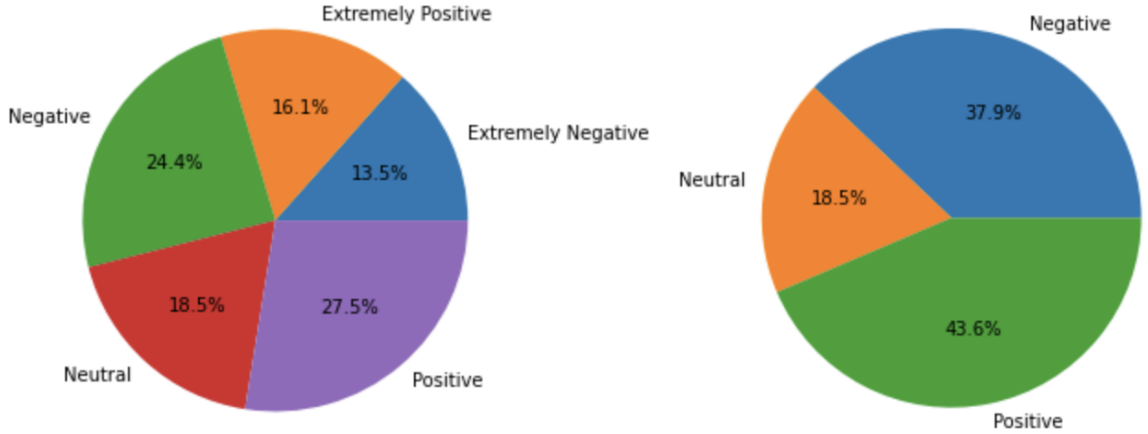
The exploratory data analysis of the datasets was crucial for this project in order to visualize the dataset and choose the machine learning models. We explored the fundamental features of the data by mostly plotting histograms and bar charts by studying different types of techniques described in [2] and [3].

1. *Checking missing and duplicate data*

Our first action was to check if the datasets contain any duplicate rows or missing features. It turned out that there are not any duplicate rows in both datasets, but there were a large number of missing values in the 'Location' column (train: 8590, test: 834). Because there is a noticeable amount of unnecessary data, we decided to drop the columns: username, screen name, location and tweet at and focus on the tweet text.

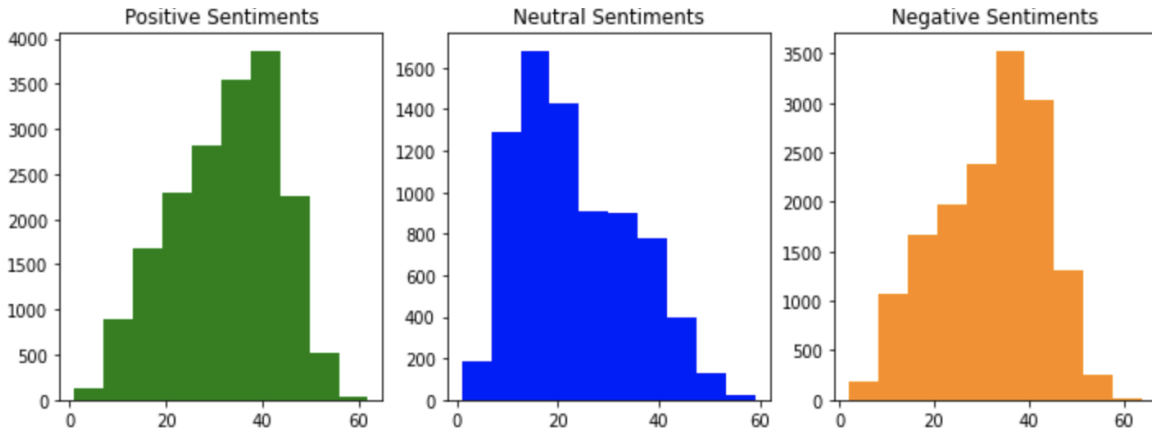
2. *The polarity category*

The labels of the sets fall into 5 categories: extremely negative, negative, neutral, positive and extremely positive and, in this project, we are going to change them to 3 most common classes used in sentiment analysis tasks: negative, neutral and positive. We converted them because our main goal was to observe if a user has positive or negative thoughts about the pandemic situation or doesn't have at all. Looking at the 3 ratings pie chart, we can easily observe that the dataset is unbalanced, the numbers of negative and positive tweets are greater than the number of neutral tweets.

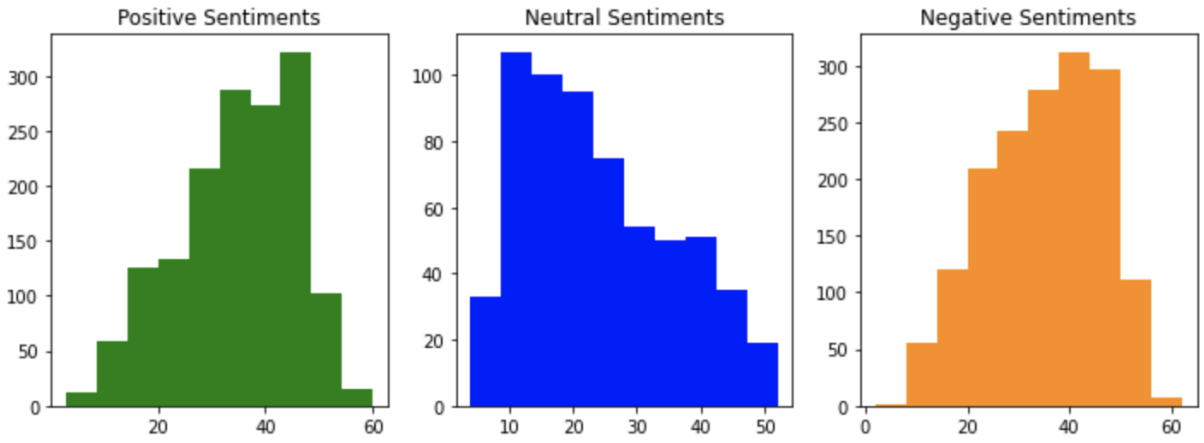


Left picture: class distribution of the initial labels. Right picture: class distribution of the converted labels.

3. *Number of words in tweets*



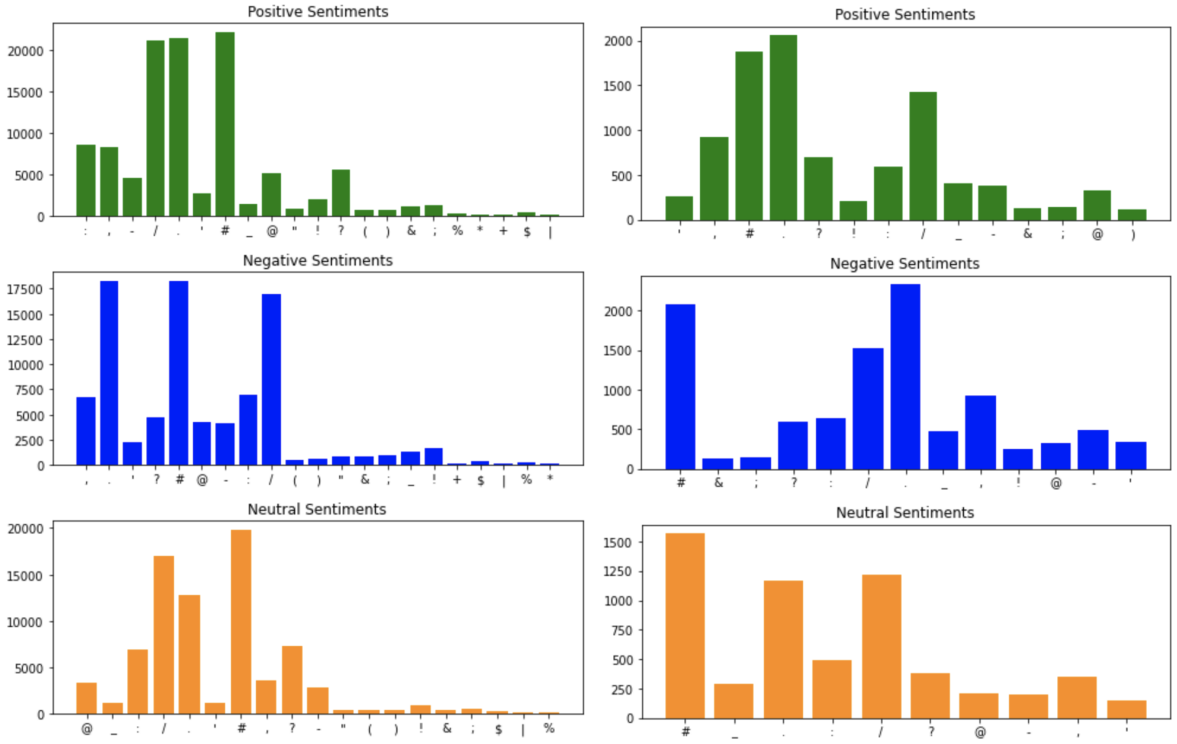
Number of words in training dataset



Number of words in test dataset

The next step of exploratory data analysis is visualizing the number of words by plotting histograms. In the figure above, the negative and positive sentiments histograms look like normal distributions with a large number of tweets having between 30 and 45 words, while the neutral sentiments histogram looks like a log-normal distribution with a large number of tweets between 10 and 25 words.

4. *Punctuation in tweets*

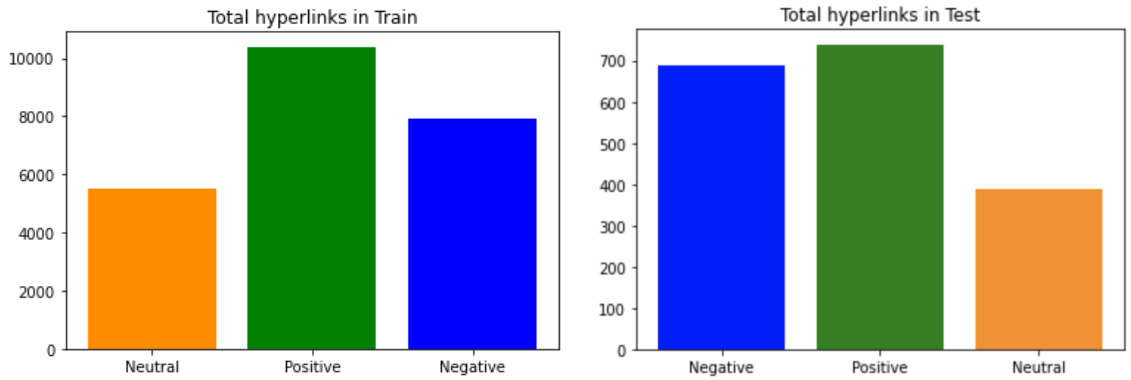


Left: the most used punctuations in the training set. Right: the most used punctuations in the test set.

Another interesting feature of tweet text we explored was the number of special characters. As the above bar plots show, tweet texts include a lot of usual punctuation

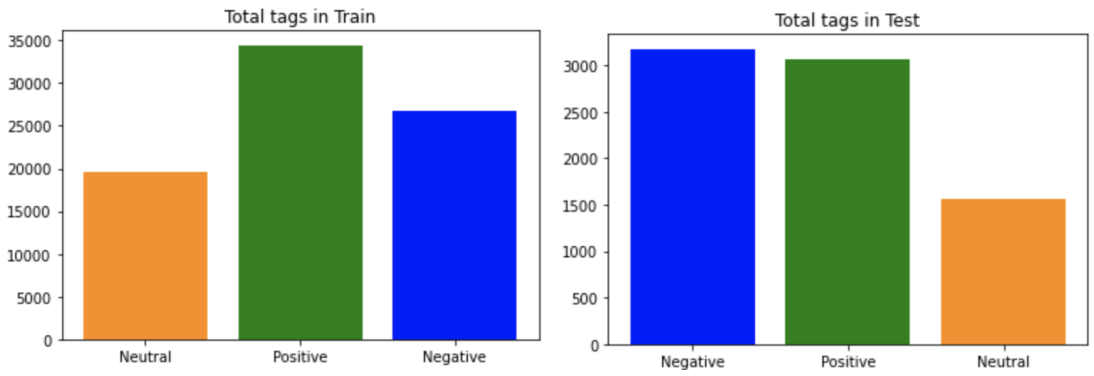
marks such as period, comma, question mark, apostrophe etc. At the same time, there are some **uncommon punctuation** such as: **slash** (/) and **hash** (#) that can indicate that tweets contain hyperlinks and hashtags, features explored below.

5. *Number of hyperlinks per sentiment*



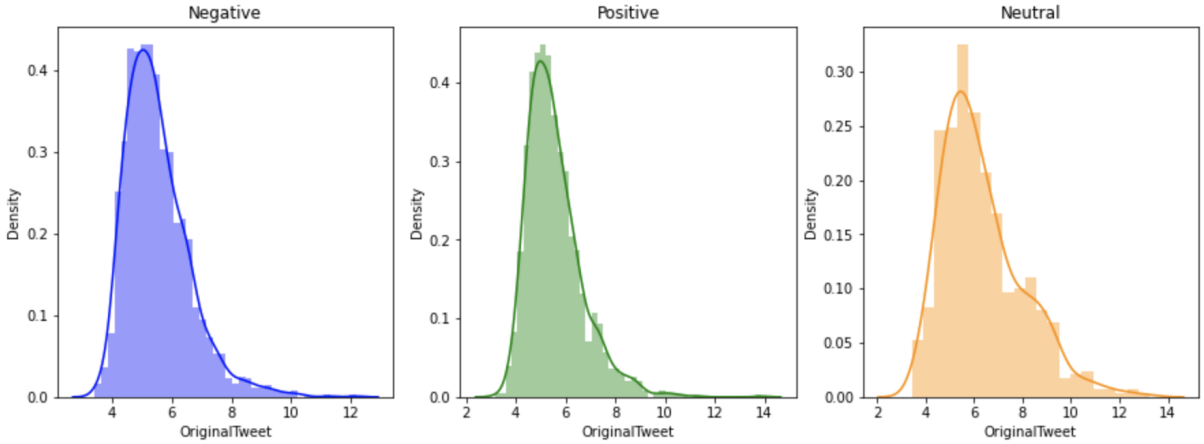
We could give an intuition on the barplot of the distribution of hyperlinks per sentiment. It may be the case that those whose reactions were positive felt as if they needed to distribute findings and raise awareness through social media. It is also evident that it is not a uniform distribution which may indicate it is a relevant feature.

6. *Number of hashtags per sentiment*



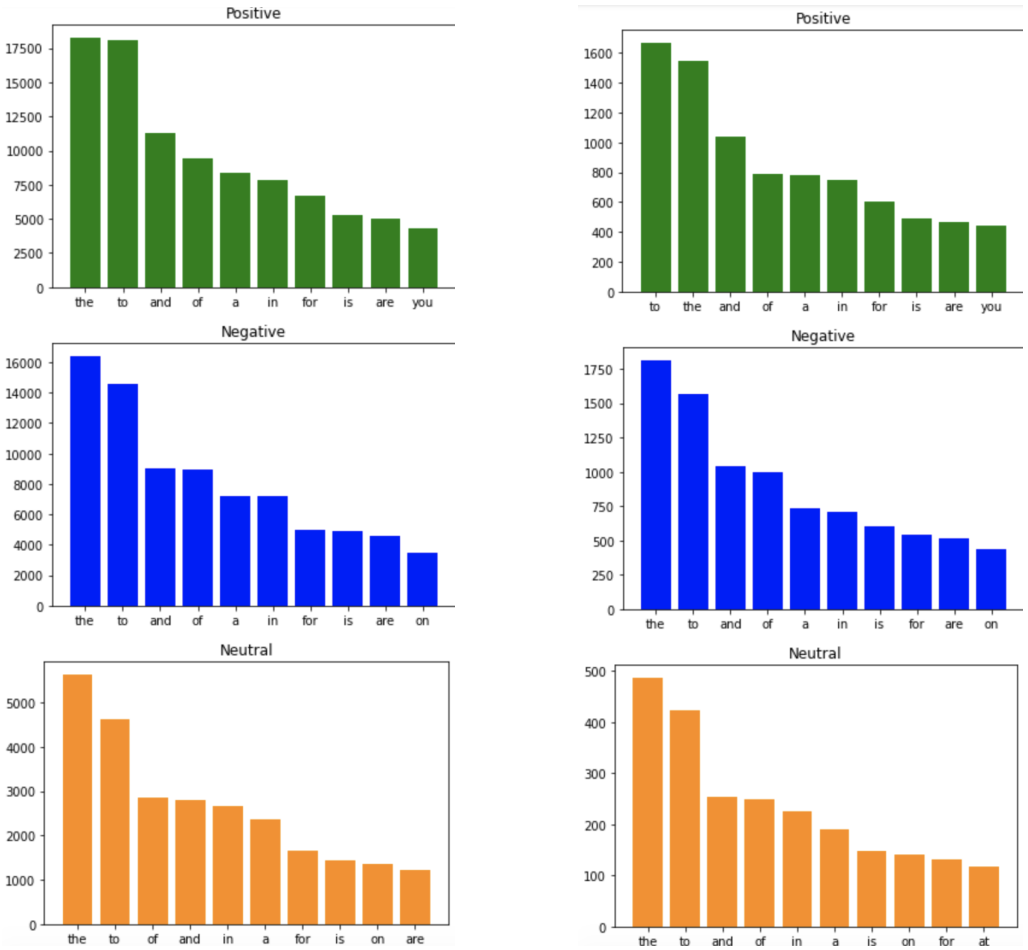
We could give an intuition on the barplot of the distribution of hashtags per sentiment. As we already saw in the barplot of the number of hyperlinks, it seems that the tweets that are positive contain, also, the largest number of hashtags, being followed by negative sentiment tweets.

7. *Average word length per sentiment*



Both the training and testing datasets show that the average word length for the Positive and Negative sentiments is greater than that of the Neutral sentiments which may suggest a stronger emotional response from the authors. However this measure does not help distinguish between Positive and Negative stanes.

8. *Most common stopwords*



Left: most common words in the training set. Right: most common words in the test set.

We can see that the top 10 most common stop words have thousands of occurrences so they could provide a lot of noise in the dataset. This may be enough of a motivation to consider removing them in the preprocessing phase.

Machine Learning Models

The purpose of the project is to predict whether the tweet sentiment is positive, negative or neutral. We implemented four machine learning models: Random forest, Logistic Regression, GloVe and Support Vector Machine, the optimal parameters and the best scores being listed below:

Algorithm	Parameters	Best Score (accuracy)
Random Forest	n_estimators=800, criterion='gini'	0.67
Logistic Regression	Tolerance=0.1, C=1	0.77
GloVe Neural Network	filters=32, kernels=5, dropout=0.5	0.80
Support-Vector Machine	C=2, kernel='rbf', gamma='scale'	0.73

1. Random forest classifier

- **Dataset split**: The initial training set is split into two sets: 90% training samples and 10% validation samples. This decision is made in order to have 3 different sets: one for training the data, one for validating the model performance and hyperparameter tuning and one for testing the final model.
- **Feature engineering**: Each tweet text is converted to lowercase, all the characters that are not letters are removed, all the stop words are removed and all the remaining words are lemmatized. After the preprocessing step, **CountVectorizer** is used in order to extract features because it converts a collection of text documents to a sparse matrix of token counts. The **max_features** parameter allows considering the top max_features ordered by term frequency and is used in the hyperparameter tuning step.
- **Model and hyperparameter tuning**: Random forest is a model used for supervised learning tasks that consist in classification and regression problems and is implemented based on constructing a multitude of decision trees at training time. In order to find the best performance of the Random forest model, I have tried different combinations of the next parameters:
    1. **n_estimator** (param of model) $\in$ *{50, 100, 200}*
    2. **max_features** (param of CountVectorizer) $\in$ *{100, 200, 400, 800}*
    3. **criterion** (param of model) = 'gini'

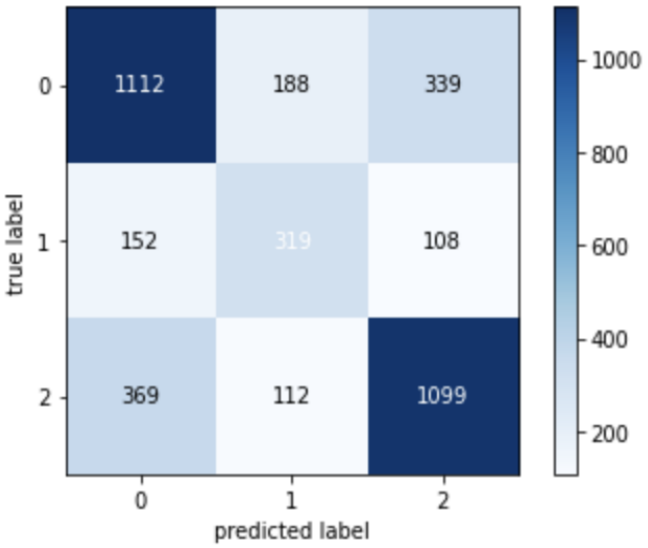
Score: Accuracy			
No. model	**n_estimator**	**max_features**	**Validation score**
RF1	100	50	0.557
RF2	100	100	0.553
RF3	100	200	0.555
RF4	200	50	0.599
RF5	200	100	0.604
RF6	200	200	0.605
RF7	400	50	0.647
RF8	400	100	0.6452
RF9	400	200	0.648
RF10	800	50	0.697
RF11	800	100	0.697
RF12	*800*	*200*	*0.703*

Hyperparameter tuning for Random forest classifier

- **Testing the model**: The figure above shows that the best model performance is achieved when the param *n_estimator* is equal to 800 and the param *max_features* is equal to 200. The **RF12** model is used for testing the performance on the test set. Test labels: *negative - 0, neutral - 1, positive - 2*.
  The accuracy for the testing dataset was 0.67. As the confusion matrix, precision and recall shows, the model is performing quite well on the negative and positive classes, but has poor results for the neutral class. The poor performance for the neutral class might be caused by the unbalanced dataset because the number of neutral tweets is very small compared with the other 2 classes.
  The next figures shows the accuracy, recall, precision, f1 score and the confusion matrix:

	precision	recall	f1-score	support
0	0.68	0.68	0.68	1639
1	0.52	0.55	0.53	579
2	0.71	0.70	0.70	1580
accuracy			0.67	3798
macro avg	0.64	0.64	0.64	3798
weighted avg	0.67	0.67	0.67	3798

Accuracy, precision, recall, f1-score of Random forest classifier



Confusion Matrix of Random forest classifier

2. Logistic regression classifier

- **Dataset split:** The initial dataset is once again split in a 90% and 10% manner, with the former percentile used for training, and the latter used for validation. This way, we have a set for training the data, one for tuning the hyperparameters and one for testing the final model.
- **Feature engineering:** Preprocessing the tweets consists in removing all non-alphanumeric characters, turning them into all lowercase characters, removing the stopwords, and then using the Porter stemmer technique, which removes the commoner morphological and inflexional endings from the words. Then, we use **TfidfVectorizer** to get the Tf-Idf index value for each word, and use the **max_features** parameter to only use the top features ordered by the Tf-Idf index.
- **Model and hyperparameter tuning:** Logistic regression is a supervised learning model which uses a logistic function to model the dependent variable. Generally used for predicting binary target variables, we can also use it in a multinomial way. This is also called a one vs all algorithm. As the name suggests in this algorithm, we choose one class and put all other classes into a second virtual class and run the binary logistic regression on it. We repeat this procedure for all the classes in the dataset. So

we actually end up with binary classifiers designed to recognize each class in the dataset. For tuning the hyperparameters, I have used the following combinations:

a. **tol** (param of model) ∈ *{0.1, 1}*
b. **C** (param of model) ∈ *{0.1, 1}*
c. **max_features** (param of **TfidfVectorizer**) ∈ *{3000, 5000}*

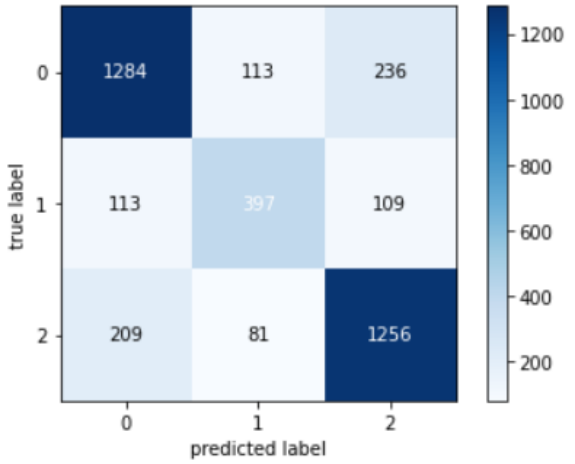
Score: Accuracy				
No. model	**tol**	**C**	**max_features**	**Validation score**
LR1	0.1	0.1	3000	0.726
LR2	***0.1***	***1***	***3000***	***0.793***
LR3	1	0.1	3000	0.726
LR4	1	1	3000	0.793
LR5	0.1	0.1	5000	0.725
LR6	0.1	1	5000	0.792
LR7	1	0.1	5000	0.725
LR8	1	1	5000	0.792

Hyperparameter tuning for Logistic Regression classifier

- **Testing the model**: The figure above shows that the best model performance is achieved when the param *tol* is equal to 0.1**,** *C* is equal to 1 and the param *max_features* is equal to 3000. The **LR2** model is used for testing the performance on the test set. Test labels: *negative - 0, neutral - 1, positive - 2.*
  The accuracy achieved was 0.77. As the confusion matrix shows, the negative and positive classes had the best performance regarding correctly predicted labels, and while the number of false positives for the neutral tweets is similar to the ones for negative and positive tweets, their percent is much higher, due in fact to the smaller number of these tweets compared to the other categories.
  The next figures shows the accuracy, recall, precision, f1 score and the confusion matrix:

	precision	recall	f1-score	support
0	0.80	0.79	0.79	1633
1	0.67	0.64	0.66	619
2	0.78	0.81	0.80	1546
accuracy			0.77	3798
macro avg	0.75	0.75	0.75	3798
weighted avg	0.77	0.77	0.77	3798

Accuracy, precision, recall, f1-score of Logistic Regression classifier



Confusion Matrix of Logistic Regression classifier

3. GloVe word embeddings neural network

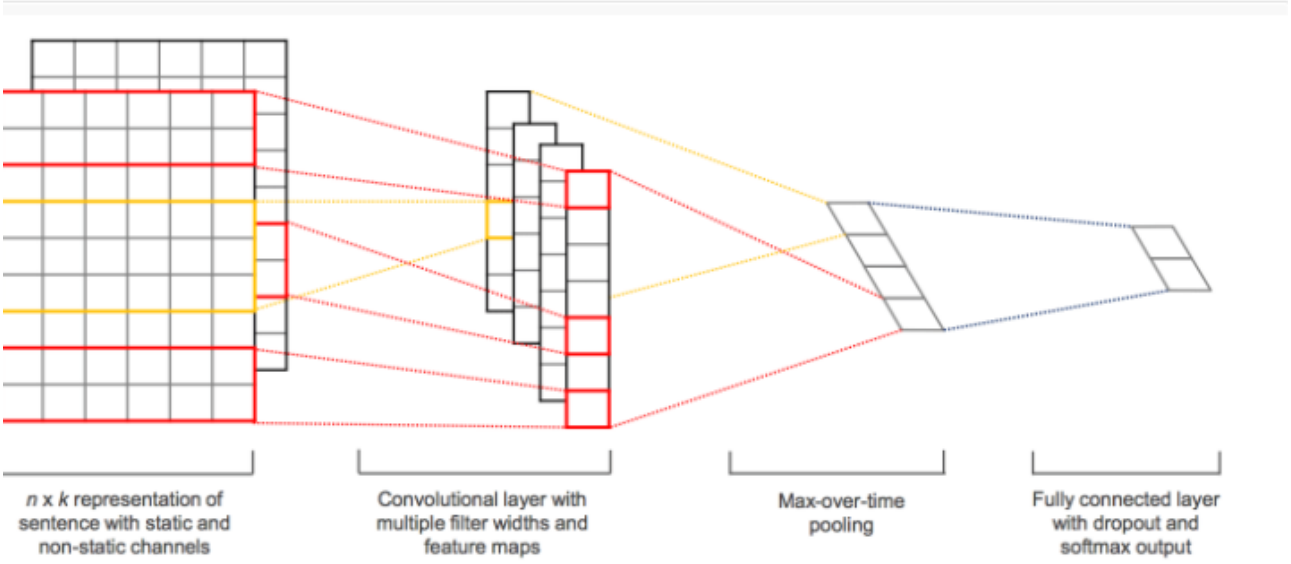
- **Dataset split:** the train-test split strategy is kept consistent to the one presented above, with 90% of the set for training and 10% for validation.
- **Feature extraction**: owing to the nature of the problem at hand, i.e. the classification of a particular stance of each sample from 3 possible options as inferred from a posted message, one can easily consider that the OriginalTweet column yields sufficient information. The preprocessing of the tweet consisted of: removing digits, mentions (words preceded by the '@' symbol), hashtags (words preceded by the '#' symbol), hyperlinks, HTML tags, emoji symbols and punctuations. It is worth mentioning that a disjunction of two terms is often represented colloquially with a slash ('/') symbol which meant that, instead of just eliminating the punctuation character, the words had to be split. Next, all common english contractions were decontracted to their original form. In order to fully leverage word embeddings from the GloVe algorithm, stopwords were not eliminated. However all words considered were lowercased.
- **Model and hyperparameter tuning:** GloVe (**Glo**bal **Ve**ctors for Word Representation) is an unsupervised learning algorithm which learns mappings of words from a given corpus to a higher n-dimensional vector space. The training is

performed on all non-zero entries of a word-to-word **co-occurrence matrix** which tabulates the log-probability of any pair of words to appear together in the given corpus. Essentially, the algorithm seeks to minimize a weighted least squares objective for the given matrix, motivating the intuition that the rate of word being seen with others might convey some meaning.

There are many pre-trained GloVe embeddings available online which provide a headstart for the learning of many NLP problems. For our problem, we used *glove.twitter.27B* from **StanfordNLP** which consists of over 27 billion tokens whose vector embeddings are of size 100.

We then used the **Tensorflow Keras TextVectorization** class to first build a vocabulary from the corpus of our preprocessed tweets**,** and second to map each sample sentence into a numerical representation of fixed size. We chose the size to be the maximum sentence length amongst all samples, which turned out to be 286. Each token in a sentence is then converted to its index in the dictionary, and for sentences with lengths smaller than 286, padding with 0s is performed. Using the indexes, we then built an embedding matrix for which every pre-trained embedding of a word found in our vocabulary is represented at the row equal to the words' index in the vocabulary.

The embedding matrix serves as the weight initializer for a Keras **Embedding** layer, the first layer in a deep convolutional neural network implemented using Keras and Tensorflow. As summarized in the figure below, the network takes as input a



Architecture of deep neural network

vectorized sentence using the TextVectorizer. The **embedding layer** maps the tokens

to their word embeddings, resulting in a batch size * 286 * 100 tensor. Next, a **one-dimensional convolution layer** applies a convolution operation on this tensor. The resulting feature maps are summarized using a **max pooling layer** and the result serves as input to a network of **fully connected** layers. The final layer is a **fully connected layer with 3 units**, representing the 3 possible outcomes, and uses a **softmax** activation to compute the most likely class. The most important hyperparameters that required tuning were:

- The number of **filters** , i.e. the resulting feature maps: *{32}*
- The **kernel size** for the convolution operation: *{3, 4, 5}*
- The **number of units** in the fully connected layer: *{16, 32}*
- The **dropout** rate between fully connected layers *{0.5}*
- The **learning rate** for the Adam optimizer *{0.001, 0.00001}*

The following table shows the result of hyperparameter tuning:

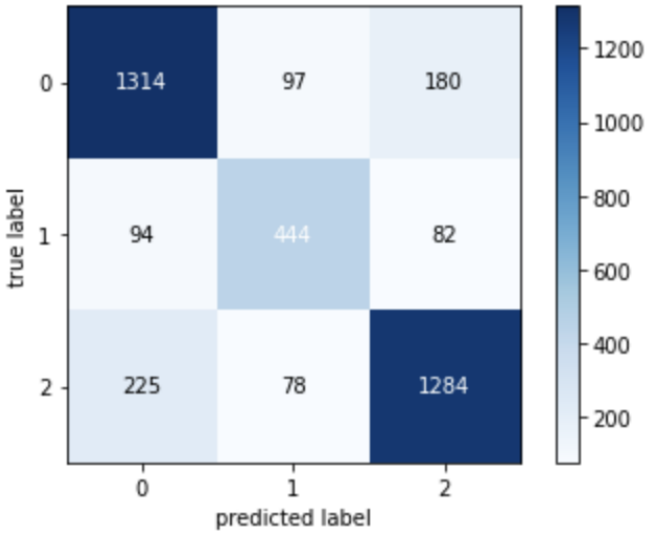
Score: Accuracy						
No.Model	**Filters**	**Kernels**	**Number of Units**	**Dropout**	**Learning Rate**	**Validation Score**
GV1	64	3	16	0.5	1e-4	0.826
GV2	64	3	16	0.5	1e-5	0.846
GV3	64	3	32	0.5	1e-4	0.817
GV4	***64***	***3***	***32***	***0.5***	***1e-5***	***0.842***
GV5	64	4	16	0.5	1e-4	0.810
GV6	64	4	16	0.5	1e-5	0.831
GV7	64	4	32	0.5	1e-4	0.807
GV8	64	4	32	0.5	1e-5	0.833
GV9	64	5	16	0.5	1e-4	0.816
GV10	64	5	16	0.5	1e-5	0.829
GV11	64	5	32	0.5	1e-4	0.799
GV12	64	5	32	0.5	1e-5	0.826

Hyperparameter tuning for GloVe classifier

- **Testing the model:** The best results have been achieved with the configuration highlighted in the table below. Test labels: *negative - 0, neutral - 1, positive - 2.*
  The model obtained accuracy equal to 0.8 and seems to perform better when distinguishing between opposite sentiments, and performed significantly poorly on the Neutral class. As outlined in the data exploration section, Neutral sentiments did not seem to offer many distinctive features, which could be considered as the main reason for the poor performance. Between Positive and Negative sentiments, the model seems to predict the latter better.
  Upon testing this configuration on the test data, we obtained the following accuracy, recall, precision and f1 scores:

	precision	recall	f1-score	support
0	0.79	0.84	0.82	1522
1	0.72	0.72	0.72	620
2	0.83	0.81	0.82	1587
micro avg	0.79	0.81	0.80	3729
macro avg	0.78	0.79	0.78	3729
weighted avg	0.79	0.81	0.80	3729
samples avg	0.79	0.79	0.79	3729

Accuracy, precision, recall, f1-score of GloVe classifier



Confusion Matrix of GloVe classifier

4. Support-Vector Machine

- **Dataset split:** for the train-validation split strategy, we kept the same percentage as above, 90% of the set for training and 10% for validation.

- **Feature engineering:** the approach used in this experiment is a bit similar to the ones above that were already presented. For **feature selection**, we filtered irrelevant dataset features, thus, we considered that the only fitting feature is OriginalTweet. For tweet **cleaning/preprocessing**, firstly, we removed the digits, mentions, hashtags, hyperlinks, HTML tags, emoji symbols and punctuation marks. Once again, all common English contractions were decontracted to their original form, such as "you're" to "you are". The stopwords were put aside from tweets and every character was lowered. Furthermore, in order to represent the tweets in a way that a Machine Learning model can understand, we need a **feature extraction** technique to convert the tweets into an array of features. For this part, we used **Word2Vec**, which is an unsupervised learning algorithm that represents words in a dense low dimensional space in a way that similar words get similar word vectors, so they are mapped to nearby points. We decided to use a pre-trained Word2Vec on Google News [5] (100B), which consists of 100 billion tokens whose vector embeddings are of size 300. For each tweet, we generated a list of embedding vectors of every word, then we computed the average of these vectors, resulting in a final embedding vector that will be used.

- **Model and hyperparameter tuning:** for the last model, we used Support-Vector Machine (SVM), which is a supervised machine learning model that maps the data into a high-dimensional feature space in order to categorize the data points, by creating an optimal boundary. For fine-tuning our model, we performed hyperparameter search using the following search space:

    - Regularization parameter C $\in$ *{0.01, 1, 1.5, 2}*
    - Kernel parameter $\in$ *{'linear', 'rbf'}*
    - Kernel coefficient for 'rbf' kernel gamma $\in$ *{'scale', 'auto'}*

Score: Accuracy				
No. model	**C**	**kernel**	**gamma**	**Validation score**
SVM1	0.01	linear	-	0.596
SVM2	0.01	rbf	auto	0.426
SVM3	0.01	rbf	scale	0.596
SVM4	1	linear	-	0.695
SVM5	1	rbf	auto	0.586
SVM6	1	rbf	scale	0.740

SVM7	1.5	linear	-	0.695
SVM8	1.5	rbf	auto	0.596
SVM9	1.5	rbf	scale	0.743
SVM10	2	linear	-	0.695
SVM11	2	rbf	auto	0.610
SVM12	***2***	***rbf***	***scale***	***0.744***

Hyperparameter tuning for Support-Vector Machine

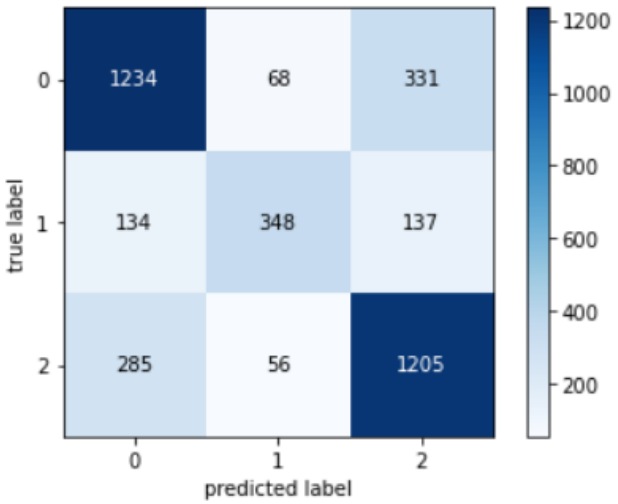
- **Testing the model:** The best performance of our model was obtained using the parameters that are highlighted in the table above, C=2, kernel='linear' and gamma='scale'. Test labels: *negative - 0, neutral - 1, positive - 2.*

  As we can see in the classification report below, the best accuracy score achieved by this model on the testing set is 0.73. Similar to the other models previously presented, this model performs better on the negative and positive examples, according to the confusion matrix. This behavior occurs because the dataset is unbalanced, having much fewer neutral examples than positive and negative ones. The model has a hard time identifying true positives for neutral examples, hence the low recall score for this class. However, the model seems to have a pretty good performance overall.

  The achieved accuracy, recall, precision and f1-score, as well as the confusion matrix, after testing the SVM12 model on the testing data, are as follows:

	precision	recall	f1-score	support
0	0.75	0.76	0.75	1633
1	0.74	0.56	0.64	619
2	0.72	0.78	0.75	1546
accuracy			0.73	3798
macro avg	0.73	0.70	0.71	3798
weighted avg	0.73	0.73	0.73	3798

Accuracy, precision, recall, f1-score of Support-Vector Machine classifier



Confusion Matrix of Support-Vector Machine classifier

Conclusions and future work

Modelling on a sentiment analysis dataset is a significant task in Machine Learning because it determines the data (in our case tweet text about the pandemic situation) is positive, negative or neutral. Exploring the dataset, we noticed that our set is unbalanced due to the small number of neutral tweets, there were some important characteristics of the text such as the presentence of hashtags, hyperlinks or the large number of words in a tweet that indicate a stronger emotional response towards Coronavirus pandemic. We implemented 4 machine learning models: 3 conventional machine learning models (Random Forest Classifier, Logistic Regression Classifier, Support Vector Machine Classifier) and a deep learning model based on a neural network. The model that performs the best is the Neural Network approach because it gets the best accuracy, recall and precision metrics of all models and the classes are well distinguished even if the dataset is unbalanced (the model has a harder time identifying true positive or neutral tweets than negative and positive one).

When it comes to *future work,* an interesting study would be fixing the unbalanced dataset in order to achieve good results for each class. This could be done by collecting more data or trying different approaches to solve this issue such as oversampling technique. It would also be interesting to resolve this problem as a 5 rating classification problem in order to explore the characteristics of the tweets that differ whether the sentiment is a usual one or an extreme one, for example positive and and extremely positive.

References

[1]https://www.kaggle.com/datatattle/covid-19-nlp-text-classification

[2]https://towardsdatascience.com/a-complete-exploratory-data-analysis-and-visualization-for-text-data-29fb1b96fb6a

[3]https://towardsdatascience.com/a-complete-exploratory-data-analysis-and-visualization-for-text-data-29fb1b96fb6a

[4] https://github.com/stanfordnlp/GloVe

[5] https://github.com/3Top/word2vec-api#where-to-get-a-pretrained-models