# Machine Learning Engineer Nanodegree Capstone

Natural Language Processing with Disaster Tweets

## Project overview

Natural Language Processing is a complex field which is hypothesised to be part of AI-complete set of problems, implying that the difficulty of these computational problems is equivalent to that of solving the central artificial intelligence problem making computers as intelligent as people.[1]

Every year more and more data is produced and a great proportion of it corresponds to human generated unstructured texts, so the need to advance in the field of Natural Language processing is even more evident. It is important to notice that a lot of that data is generated in real time so the efficiency of text processing is also representing an important challenge.

This project focuses on the analysis of text generated messages in social media. In this particular case, we are going to analyse tweet messages generated in the Twitter social network with the aim to determine if a tweet has some content related to a natural disaster.

## Problem statement

I have chosen this Kaggle competition because I don't have much experience with NLP projects and I wanted to get started into this field. This project seems like a good opportunity to start applying text processing techniques and get fluency.

From Kaggle[2] competition page:

Twitter has become an important communication channel in times of emergency.
The ubiquitousness of smartphones enables people to announce an emergency they're observing in real-time. Because of this, more agencies are interested in programatically monitoring Twitter (i.e. disaster relief organizations and news agencies).

But, it's not always clear whether a person's words are actually announcing a disaster.

---

[1] "Natural language processing - Wikipedia."
https://en.wikipedia.org/wiki/Natural_language_processing.
[2] https://www.kaggle.com/c/nlp-getting-started/overview

In this competition, you're challenged to build a machine learning model that predicts which Tweets are about real disasters and which one's aren't. You'll have access to a dataset of 10,000 tweets that were hand classified.

# Metrics

In order to evaluate the performance of the model the metrics used are: accuracy, precision, recall and f1-score. These metrics will give us a good intuition about how well the model is classifying or misclassifying between disaster or non disaster. Additionally, I usedRMSprop as the optimizer inside the LSTM network.

# Analysis

## Data exploration

The datasets used are the included in Kaggle competition page. Three datasets are provided:

- train.csv
- test.csv
- sample_submission.csv

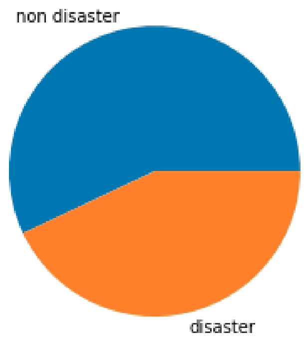
For the implementation in this capstone only train and test datasets will be used. Each of the samples of train and test datasets has the following information:

- **text**: The text of a tweet
- **keyword**: A keyword from that tweet (although this may be blank)
- **location**: The location the tweet was sent from (may also be blank)

After reading both train and test datasets some exploratory data analysis was made in order to get some insights from data. In next subsections the different data exploration steps will be detailed.

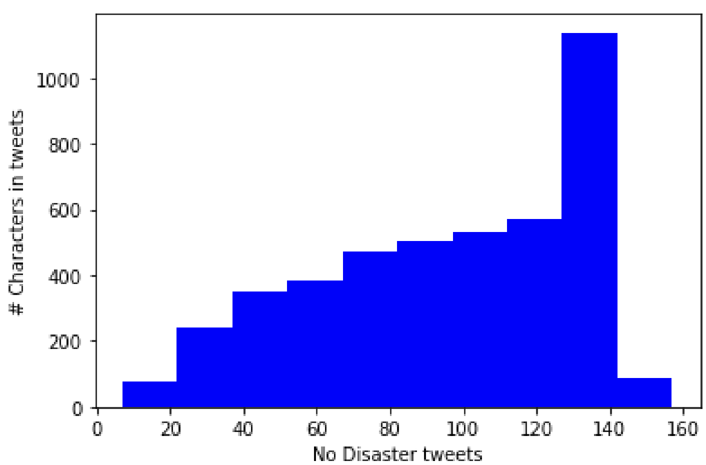
### Checking class distribution

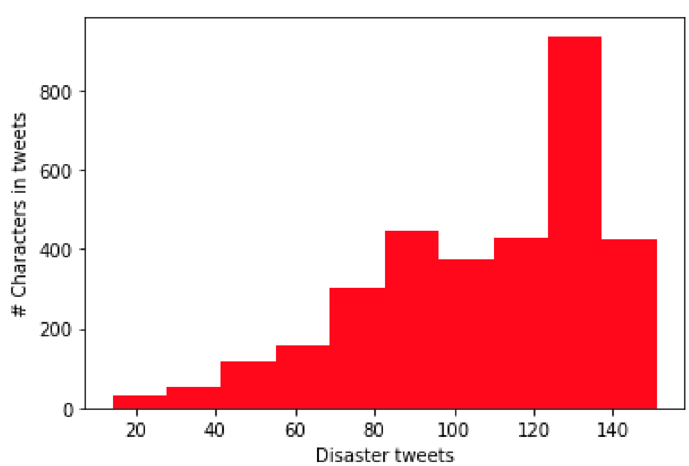
First, I analyzed the class distribution in the train dataset in order to know how many samples of each class there are. As we can see, there are more samples that represent non disaster tweets than disaster tweets but there is not a huge difference so we can consider that dataset is balanced.




## Get number of characters in tweets

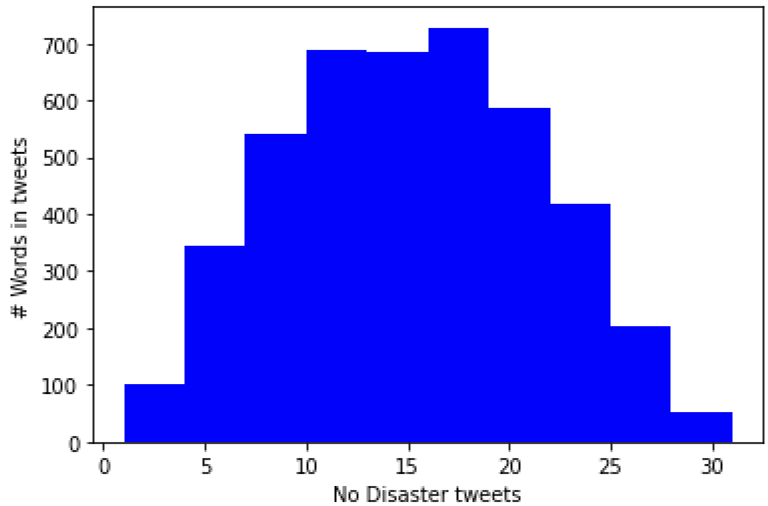
Related to the analysis of the number of characters in tweets for both disaster tweets (class 1, printed in red) and non disaster tweets (class 0, printed in blue) we can see that the distribution is more or less the same for both cases. Most of the tweets have between 120 and 140 characters.

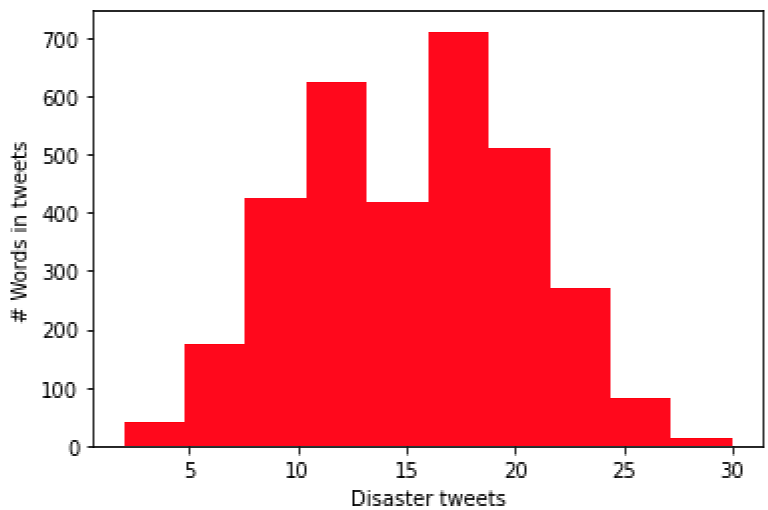


## Get number of words in each tweet

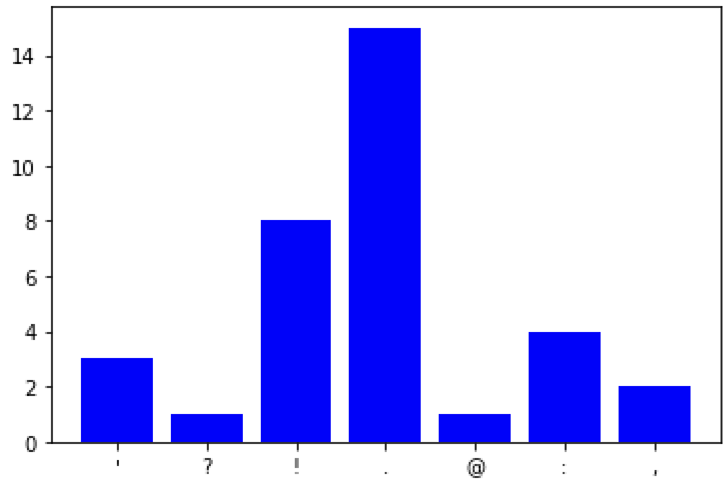
Next, the same operation described in previous subsection was performed but for number of words instead of number of characters for both disaster (printed in red) and non disaster (printed in blue) tweets.



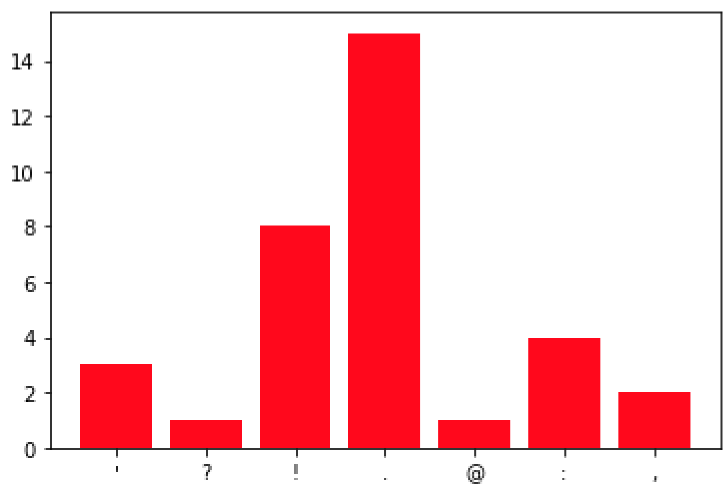

## Analize punctuations marks in non-disaster class

With the objective of doing a good cleaning of the data, it is necessary to know how many punctuation marks there are and what they are. In the figure below we can see them to give us an idea in order to take these punctuations marks into account when doing data cleaning.



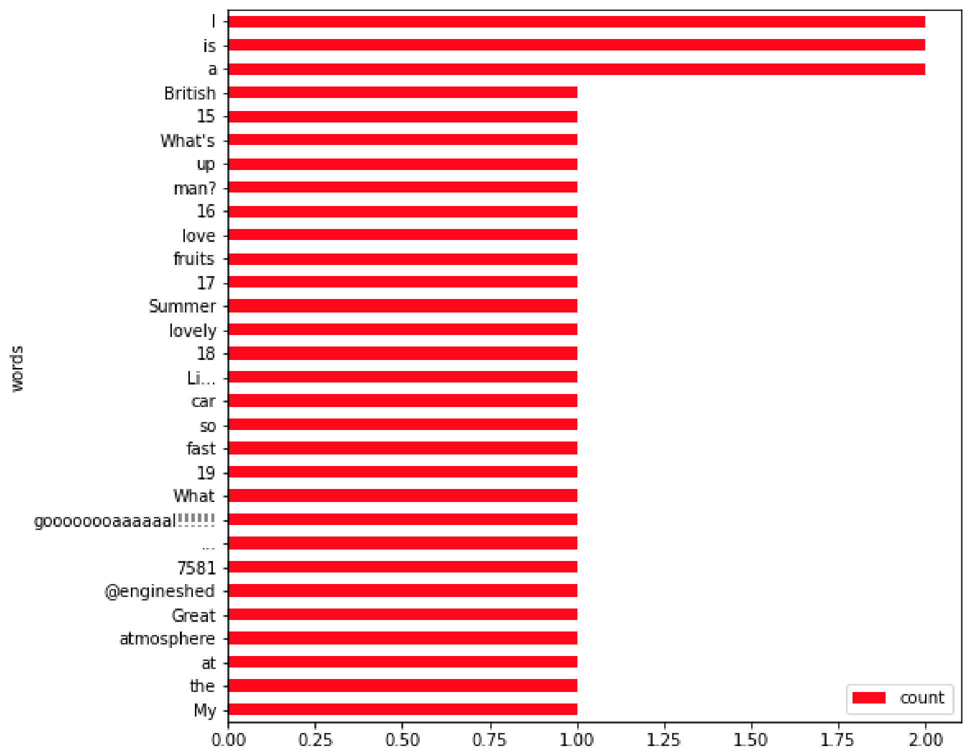

## Analize punctuations marks in disaster class

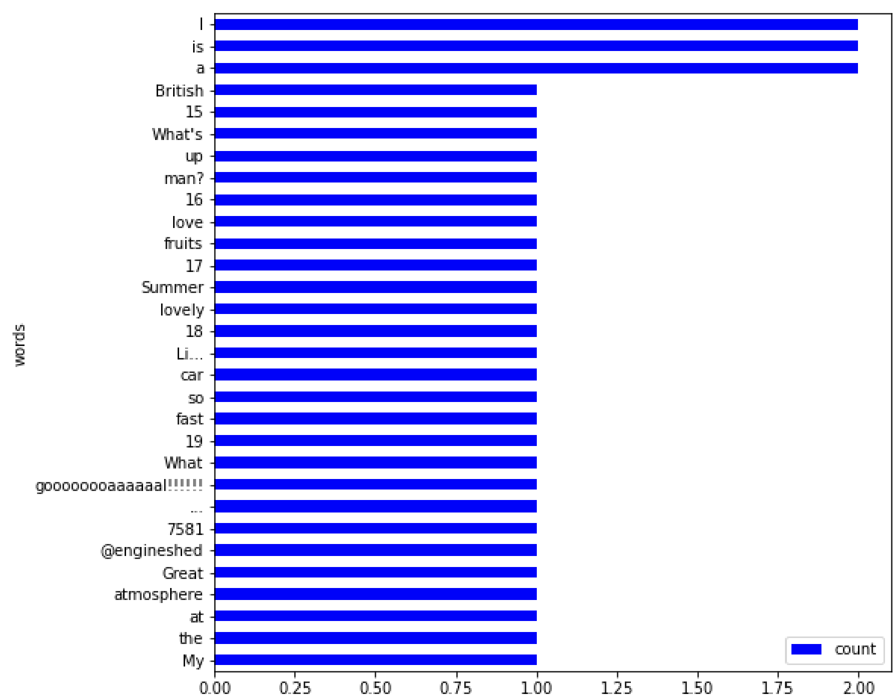
With the same goal as described in the previous subsection, I perform the same punctuation mark analysis over disaster class (1).




## Get the most common words

In order to get more insights about data I studied what are the more common words present into the dataset for disaster (printed in red) and no disaster (printed in blue). With this analysis I can notice that there are punctuation marks and stopwords at the top of this counting. This is important to take into account when doing data cleaning trying to remove this kind of character.

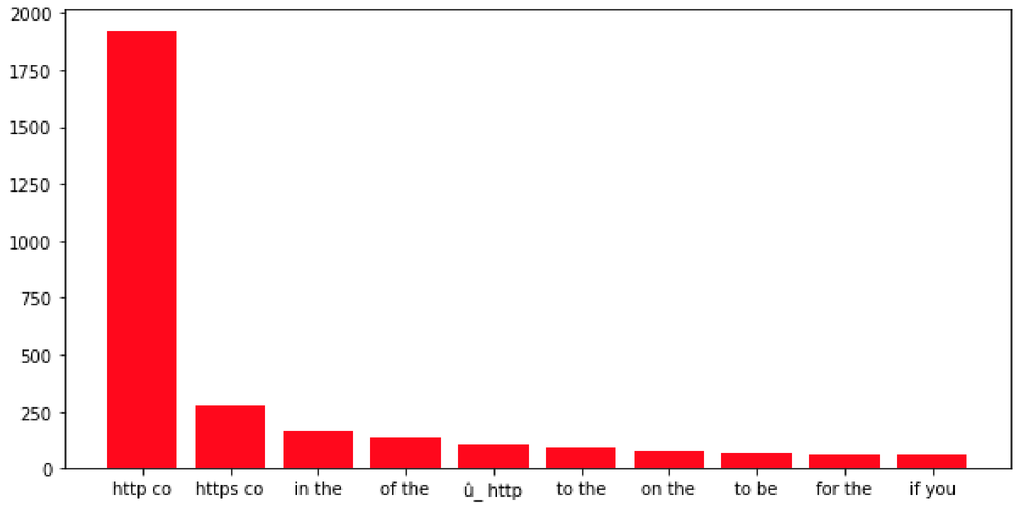


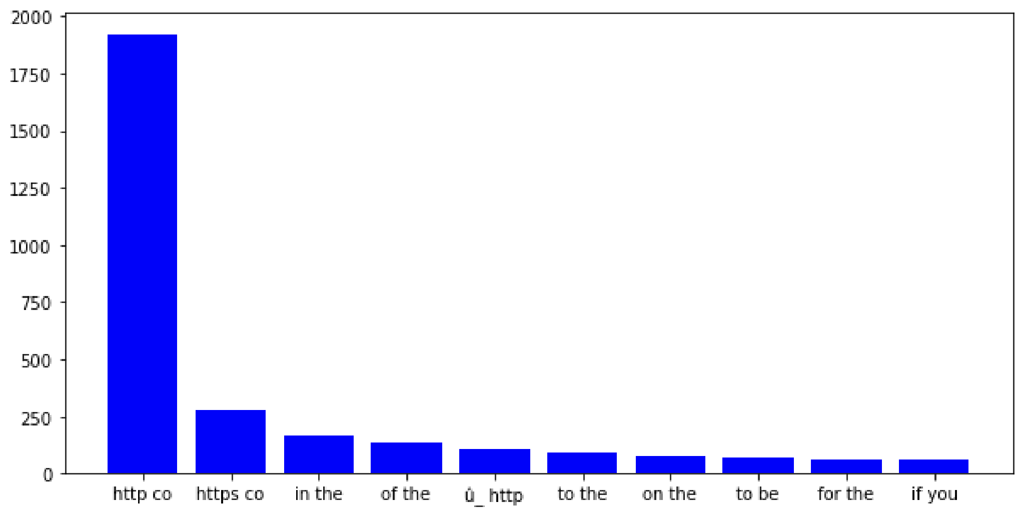




## Bigram analysis

When performing the same operation as in the previous section but for bigrams (or combination of two words appearing together) I noticed that it is present some related to links and stopwords.

Some this gave the clue that this kind of characters should also be removed when performing data cleaning. The figures below are for disaster tweets (printed in red) and no disaster (printed in blue).




# Data cleaning

In base of the insight getting during data exploration I performed next data cleaning operations:

- Remove text in square brackets.
- Remove links.
- Remove punctuation marks.
- Remove words containing numbers.

# Algorithms and techniques

Once the data has been cleaned, it is necessary to perform preprocessing on it in order to prepare the data to execute the designed algorithm. For this project I applied two preprocessing operations described next: Tokenization and application of GloVe algorithm.

## Tokenization

The first preprocessing step is performing a tokenization over the dataset in order to split the text into sentences of words. With this tokenization step we get the text into a format that is easier to convert to raw numbers, which can actually be used for the model later.

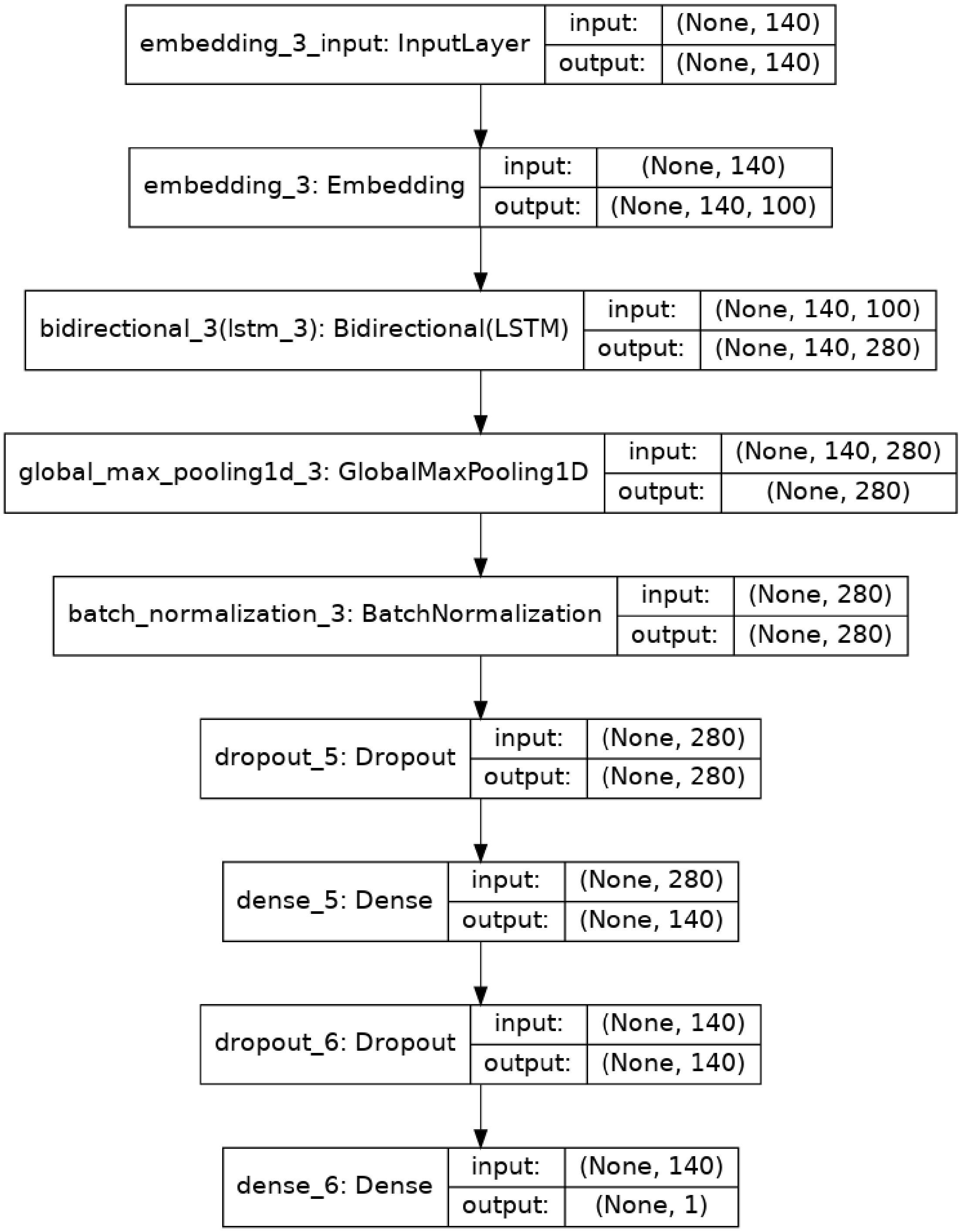
## GloVe

Next to tokenization I apply another preprocessing step in which I execute the GloVe algorithm. GloVe (Global Vectors for Word Representation) in an unsupervised learning algorithm that let us obtain a vector representation for words. The performance of this algorithm focuses on finding words co-occurrences over the whole corpus and generates the embeddings relating the probabilities that two words appear together. The code for the implementation of the load of this pre-trained model for word embedding was taken from a tutorial of the official Keras web page (https://keras.io/examples/nlp/pretrained_word_embeddings/).

## Model details implementation

For this project I have designed and trained a LSTM neural network using Keras library to define layers and Tensorflow as backend to build and execute the model. The topology of the neural network is presented in the figure below. The dimensionality of the inputs is calculated based on the dimension of the embedding matrix obtained when applying the GloVe algorithm.

The neural network starts with an Embeddings layer that holds the embedding vectors and its inner workings consist of a vector of indices that convert it to the corresponding vector. Next there is a LSTM layer setting to be bidirectional (this way form an acyclic graph where the cell memory from different time steps will be combined to produce the output prediction). Following, I applied a GlobalMaxPool layer to reduce dimensionality. In the next layer I applied BatchNormalization (it is a transformation that maintains the mean output close to 0 and the output standard deviation close to 1). In the rest of layers of the topology I applied a combination of Dropouts and Dense operations to prevent overfitting.Additionally, the chosen optimizer for the LSTM network is RMSprop.


embedding_3_input: InputLayer
input: (None, 140)
output: (None, 140)
embedding_3: Embedding
input: (None, 140)
output: (None, 140, 100)
bidirectional_3(lstm_3): Bidirectional(LSTM)
input: (None, 140, 100)
output: (None, 140, 280)
global_max_pooling1d_3: GlobalMaxPooling1D
input: (None, 140, 280)
output: (None, 280)
batch_normalization_3: BatchNormalization
input: (None, 280)
output: (None, 280)
dropout_5: Dropout
input: (None, 280)
output: (None, 280)
dense_5: Dense
input: (None, 280)
output: (None, 140)
dropout_6: Dropout
input: (None, 140)
output: (None, 140)
dense_6: Dense
input: (None, 140)
output: (None, 1)

## Benchmark

The execution of this project was made using a notebook created inside Sagemaker as we see during the Nanodegree. The selected kernel was *conda_tensorflow2_p36* because it has all the requirements needed to implement and execute this project.

To benchmarking and compare the results of the implemented model using a LSTM neural network I choose a simple machine learning model: MultinomialNB.

The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work.[3]

I trained a MultinomialNB model with the same train data as I am going to train the LSTM neural network. After that, I evaluated using the test dataset obtaining the following values for the metrics:

- Accuracy = 0.5837
- Precision = 0.5371
- Recall = 0.5386
- F1-score = 0.5379

# Results

Once the model is trained it is time to evaluate using the test dataset. The chosen metrics are accuracy, precision, recall and the relation between these two (F1-score) because they are very good metrics for supervised classification problems. In the case of precision and recall also give us a lot of information even if the dataset is unbalanced.

The results for all the chosen metrics are presented in the image below. As we can see for test data the accuracy obtained is 0.7859, so is really a high value. Also good values for precision, recall and F1-score are obtained. The exact values obtained was:

- Precision =0.8367
- Recall = 0.6510
- F1-score = 0.7323

Finally, as we can see these metric results for the LSTM neural network improves a lot the values obtained for the MultinomialNB model presented in the Benchmark section.

[3] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html