Capstone Project Report. Machine Learning Engineering Nanodegree

Text Classification: Toxic Comment Analysis

Stefano Paccagnella

17 October 2021

Contents

Project Overview

In today's word 2.5 quintillion bytes of data are generated daily. The importance of interpreting this continuous stream of information has never been so important, and the actual use of AI and ML algorithms is finding fertile ground for achieving concrete and extraordinary results. Among these data exchanged, text represents a huge proportion of the overall information transmitted worldwide.

As unstructured data, the analysis of text can be extremely hard to achieve, but thanks to Natural Language Processing (NLP) and ML algorithms this type of analysis is now getting easier and easier, which is also translating in an increasing value for business. Among the main application of NLP, we surely have Sentiment Analysis. The main scope of Sentiment Analysis is to extract the meaning behind a text, by classifying it as positive, negative or neutral.

The application of NLP and Text Classification which will be used in this project is referring to a Kaggle competition held in 2017-2018, the "Toxic Comment Classification Challenge". The idea behind the competition is to try to go behind the scope of a traditional sentiment analysis approach, i.e., to classify a text as positive/negative and to develop a model which will recognize the level of toxicity of a target text. This will mean for example to classify the target variable as an "insult", "threats", "obscenity", etc., based on a pre-defined set of labels.

Project Statement

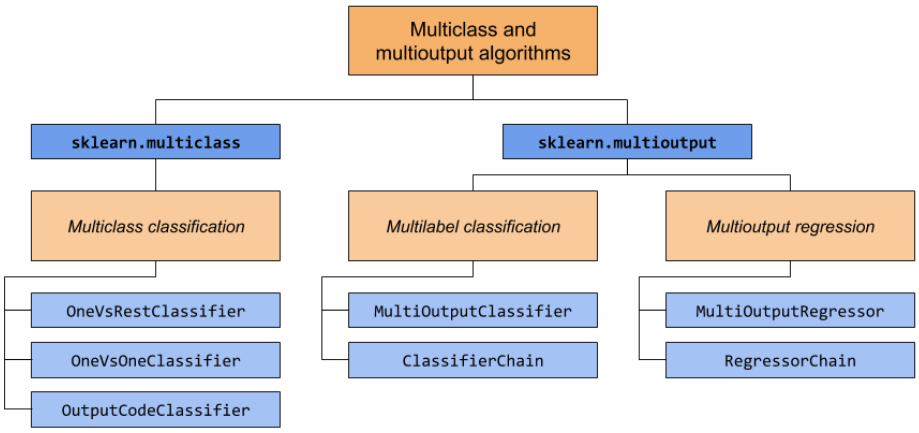
The use of "hate speech" and harassment online is a treat not only from a social point of views, but also economically and politically speaking. Therefore, the ability to classify comments, post, messages, etc. in the proper way, it is becoming every day more impelling for all online platforms.

In this context, the Conversation AI team has developed several models to improve the monitoring of conversation online. The main issue of these models (despite the error in the prediction as such) is the fact that they don't allow to label the type of toxicity of the comments analyzed. This means that we cannot really generalize the use of these predictions, as not all online platforms may be willing to act on a text (e.g., with censorship) with the same level of intervention. It will mainly depend on the internal policy of the company. Therefore, the possibility to develop a model that will grant the possibility to define and label the type/tone of a text data type into a specific category, e.g., "insult", "threats", etc., it is quite interesting.

In this project, we will be using several NLP models, with increasing complexity, to try to achieve our final goal. In the original project proposal, we were thinking of using several ML text classification algorithms, from "Bag of Words" to "BERT". We have decided to condense some of them to try to optimize our approach.

We will surely be using "Bag of Words" as our benchmark model. This model can be considered at the basis of text analysis and for this reason it will be used as the baseline to evaluate the performance of all other models.

As the dataset provided required to classify comments based on their "tone" of toxicity, we will be facing a multi-Output classification problem. This is something different compared to e.g., Sentiment Analysis were normally we address a Multiclass classification problem (e.g., "positive", "negative" and "neutral"). Based on the six toxicity classes predefined by our dataset, i.e., insult", "threats", "obscenity", etc., we will need to use a Multi Output Classifier (as shown in the picture below. Source: Scikit-learn library).



Metrics

In order to evaluate the performance of our models, we will use the AUC-ROC Curve. We have decided to use this evaluation metrics as the best one in terms of multi-class classification problem. AUC stands for "Area under the Curve", while ROC stands for "Receiver Operating Characteristics". More specifically, ROC is a probability curve and AUC represent the degree of measure of separability, i.e., it tells how well the model is capable of distinguishing between labels/classes. Ideally the best model should score an AUC closer to 1, which means is has a very good separability. At the same time the worst model has an AUC closer to 0, which means that has a bad score in separability. A model with AUC at 0.5 means that the model has no class separation capacity at all (see understanding AUC-ROC curve).

What is a ROC:

The ROC curve is plotted with TPR (True Positive Rate) against the FPR (False Positive Rate) where TPR is on the y-axis and FPR is on the x-axis.

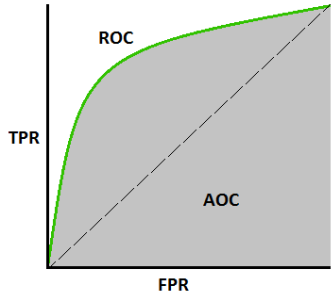


Figure 1 – AUC -ROC Curve

The terms used by the AUC-ROC curve can be explained as:

True Positive Rate (TPR) is a synonym for recall and is therefore defined as follows:

$$TPR=TPTP+FN$$

False Positive Rate (FPR) is defined as follows:

$$FPR=FPFP+TN$$

As the *AUC-ROC* curve is making a comparison between the rate of False Positive and the rate of False Negative, we are describing the sensitivity of a model:

$$Sensitivity = \frac{TP}{TP + FN}$$

Therefore, the ROC curve is defying a threshold that maximize the probability rate of TP Vs. the probability rate of FN.

What is a AUC:

The "Area under the Curve" measures the area behind the ROC curve. AUC of a classifier/model is equal to the probability that the classifier will rank a randomly chosen positive example higher than that of a randomly chosen negative example. More specifically is the ability of a classifier to distinguish between classes and it represents a summary of the ROC curve. Higher the AUC is, better the model is performing as it can distinguish between classes in a better way.

Analysis

The dataset, provided in the Kaggle "Toxic Comment Classification Challenge". It is composed by an extraction of Wikipedia comments, which have been manually pre-labelled according to

their level of toxicity. The classes identified are “toxic”, “severe toxic”, “obscene”, “threat”, “insult” and “identity hate”.

The data provided are:

- ***train.csv*** *– which is the dataset that will be used for training our model.*
- ***test.csv*** *– which will be used to test/predict the class of toxicity of comments not previously analyzed by our model.*

The goal is to develop a model which predicts a probability of each type of toxicity for each comment in the test file.

The table below is showing a summary of the “Train.csv” data (first four rows):

	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

It is also interesting to observe that in the “Train.csv” file, there are:

- 159571 comments, which corresponds also to the total number of rows of our dataset.
- The data provided are in the correct data type. “Id” and “comment_text” are string, while the toxicity classes are treated like integer, as per each class we have a hot-encoding feature. This feature is represented by the number 1, if the comments correspond to the class, or 0 if it doesn’t correspond to the class.
- It is also important to notice that there are no Null values in our dataset. As we know in case of Null, we would have to clean the dataset as ML algorithms do not accept null features.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 159571 entries, 0 to 159570
Data columns (total 8 columns):
 #   Column         Non-Null Count   Dtype 
---  ------         --------------   ----- 
 0   id             159571 non-null  object
 1   comment_text   159571 non-null  object
 2   toxic          159571 non-null  int64 
 3   severe_toxic   159571 non-null  int64 
 4   obscene        159571 non-null  int64 
 5   threat         159571 non-null  int64 
 6   insult         159571 non-null  int64 
 7   identity_hate  159571 non-null  int64 
dtypes: int64(6), object(2)
memory usage: 9.7+ MB
```

Similarly, it will be interesting to observe also the features of the “test.csv” dataset. It will be used on a later stage, as it will be the set of data on which we will be using the trained model. The first four rows of the “test.csv” data are represented below:

	id	comment_text
0	00001cee341fdb12	Yo bitch Ja Rule is more succesful then you'll...
1	0000247867823ef7	== From RfC == \n\n The title is fine as it is...
2	00013b17ad220c46	" \n\n == Sources == \n\n * Zawe Ashton on Lap...
3	00017563c3f7919a	:If you have a look back at the source, the in...
4	00017695ad8997eb	I don't anonymously edit articles at all.

As before for the "train.csv", it is also interesting to observe that:

- There are 1531634comments to be analyzed.
- There are 0 Null values. Therefore, no further cleaning will be needed.
- The data type is correct.

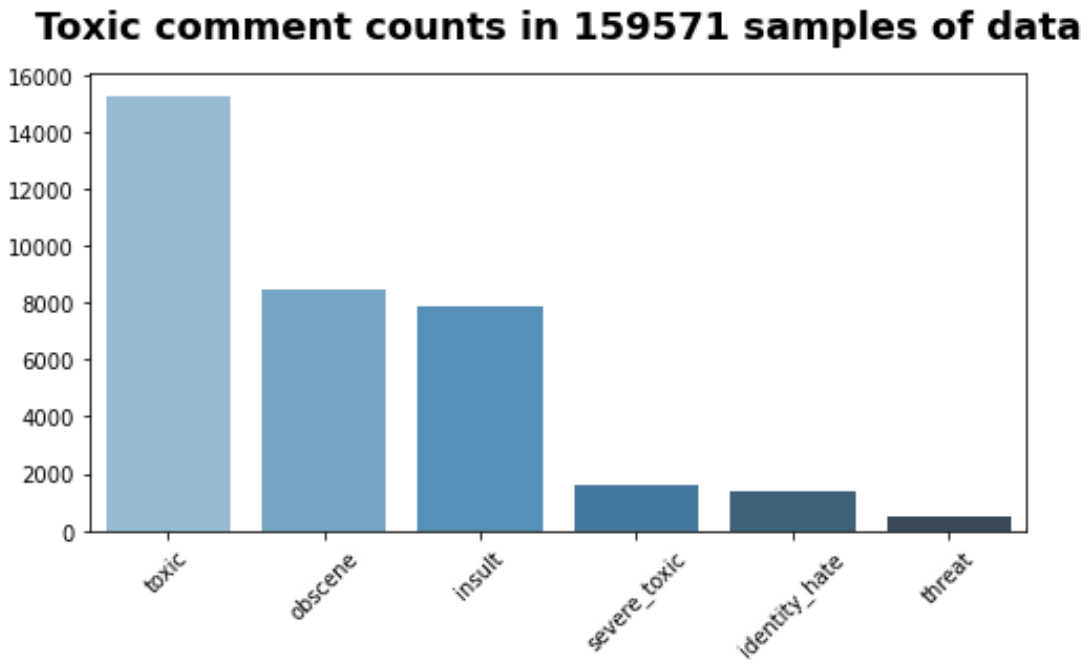
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153164 entries, 0 to 153163
Data columns (total 2 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   id            153164 non-null  object
 1   comment_text  153164 non-null  object
dtypes: object(2)
memory usage: 2.3+ MB
```

At this stage, it will be more interesting to focus on the "train.csv" dataset, as it will be the one used to train our model.

The first aspect we will focus it to observe how many labelled comments we have per class, and which is the % of non-labelled comments per each class. According to the analysis performed, the result is a quite unbalance dataset:

```
The total label for the category toxic are 15294 out of the 159571 comments in the training dataset. This is the 0.1%
of the total in the toxic category. This also mean that 0.904% is a not toxic comments.

The total label for the category severe_toxic are 1595 out of the 159571 comments in the training dataset. This is th
e 0.01% of the total in the severe_toxic category. This also mean that 0.99% is a not severe_toxic comments.

The total label for the category obscene are 8449 out of the 159571 comments in the training dataset. This is the 0.0
5% of the total in the obscene category. This also mean that 0.947% is a not obscene comments.

The total label for the category threat are 478 out of the 159571 comments in the training dataset. This is the 0.0%
of the total in the threat category. This also mean that 0.997% is a not threat comments.

The total label for the category insult are 7877 out of the 159571 comments in the training dataset. This is the 0.0
5% of the total in the insult category. This also mean that 0.951% is a not insult comments.

The total label for the category identity_hate are 1405 out of the 159571 comments in the training dataset. This is t
he 0.01% of the total in the identity_hate category. This also mean that 0.991% is a not identity_hate comments.
```

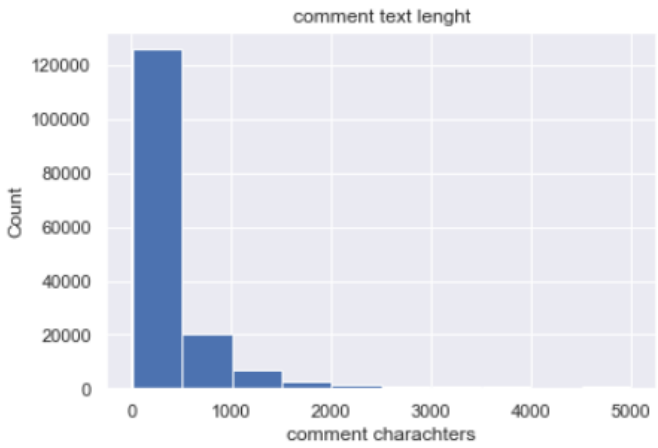
The major class representation is given by the “Toxic”, but anyway the non-labelled % per each class is quite high, as it is always around 90% of the targeted class. This is also shown by the below graph:



By further analyzing the dataset, we have also concluded that the % of non-labelled comments is 90.4% across all categories. This is very important as we will need to take into consideration this unbalance when we will train our ML models. Nevertheless, we need also to be cautious on how we are going to consider this unbalance, especially if we consider the unbalance across different classes.

Here, in fact, we are speaking mainly of a Multi-Label Classification problem, which is different from multi-class classification, i.e., trying to classify a text over multiple variables simultaneously, e.g., “positive”, “negative” and “neutral” (like in sentiment analysis). In our Multi-Label Classification problem, we must consider each category as a separate word. Therefore, in our dataset the unbalance is between what is classified as toxic (represented by the integer 1) and what is not toxic (represented by the integer 0) or more specifically not-labelled, etc. Our ML model will most likely predict over each single category separately and we will need most likely to consider this “internal” distribution, rather than an “external” distribution between different classes. It is also important to consider that the “external” unbalance between classes tells the model how frequent a label is compared to another, e.g., “toxic” may be a most likely prediction over “threat”, which is something we would like our model to keep considering.

Another aspect that we can analyze in our EDA is the lengths of characters in the comment text. As we are applying text classification algorithms, the text composition is quite important. From the below graph, we can observe that the majority of comments are contained between the 0-500 and 501-1000 bin of the histogram.



This is also quite evident if we consider the below table, which is providing some descriptive statistics information in relation to our comments. From this table, we can mainly observe that the mean of comments length is 394 characters, while the max is around 5000.

```
count    159571.000000
mean        394.073221
std         590.720282
min           6.000000
25%          96.000000
50%         205.000000
75%         435.000000
max        5000.000000
Name: comments_len, dtype: float64
```

ML models – Analysis, Methodology and Evaluation

As mentioned at the beginning, we will try to use several NLP models to predict the level of toxicity of the provided comments. The baseline model that will be used a benchmark is the "Bag of Words"

Bag of Words:

We have decided to start from the "Bag of words" model as it is the easiest way of representing text data. It has always obtained great results in all NLP problems from language modelling to document classification.

The way "Bag of words" is working is quite straight forward. It is a technique that enable the tokenizing of text and find out the frequency of each token. For example, the sentence "It is a sunny day" is deconstructed in to "it" "is" "a" "sunny" "day". The second step is than to transform the tokens into vectors (i.e., sequences of numbers) that can be analyzed by an algorithm. For example:

“It is a sunny day” could be represented by the vector: [1, 1, 1, 1, 1, 1, 0, 0, 0, 0]. This process is called vectorization and consist in:

1. Counting the number of times each word appears in a sentence.
2. Calculating the frequency that each word appears in a document out of all the words in the document.

For example, the below table is representing the frequency of each token in a specific text:

	1 This	2 movie	3 is	4 very	5 scary	6 and	7 long	8 not	9 slow	10 spooky	11 good	Length of the review(in words)
Review 1	1	1	1	1	1	1	1	0	0	0	0	7
Review 2	1	1	2	0	0	1	1	0	1	0	0	8
Review 3	1	1	1	0	0	0	1	0	0	1	1	6

This overall approach is what we are going to implement in our benchmark model. More specifically, we will:

1. Pre-process our dataset to clean the main data text issues.
2. Split our Train.csv dataset into train and test (validation) data
3. Creating the Bag of Words, or frequency matrix per each tokenized word.
4. Initialize and train a Multi Output Classifier with different algorithms.
5. Measuring the performance on the test (validation) dataset.

Data pre-processing:

In order to clean the comment text column and create a uniform text to be analyzed by our algorithm, we have built a cleaning text function that is:

- Eliminate all digits from the text
- Clean all unwanted characters with a Regex function
- Reduce all capital letters to lower characters
- Removing stop words, i.e., all English conjunction or unmeaningful text from English vocabulary
- Stemming and Lemmatizing, i.e., normalize text
- Re-transform token as List back to text string

The Normalization of text is a compulsory step in our analysis. Without the cleaning of unwanted text or by the presence of text that is repetitive, the prediction could be biased.

Data Split:

After having pre-processed all comments in the "Train.csv" file, we will now split it into proper training and test dataset. This is a compulsory step if we want to avoid of having a model which is trained on the same data that will be used for evaluating the performance of the model. We have decided to use 33% of our original dataset to be used to validate our final model.

```
#Prepare the dataset to be splitted into Train and Test data
X = df_train.comment_text  #predicotors
y = df_train[["toxic","severe_toxic","obscene","threat","insult","identity_hate"]] # independent variable

# Split of the dataset. Test size 33% and random seed = 42
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.33, random_state=42)
```

It is important to remember that this test dataset is nothing related to our "Test.csv". "Test.csv" it will be used to predict the class of appurtenance of other comments that have been separate from our original testing data. The test we have split from the train dataset is more of a validation framework.

Bag of Words – Frequency Matrix:

After our text has been normalized, the next step is to create our frequency matrix. We have decided to set the max number of features to 1000 as the bucket where majority of comments are felling (as per EDA analysis). We could have increased this value, but there is the risk of having a sparse input (for our NLP model), which may lead to misprediction.

Our final matrix will look like the one below:

	abc	abide	ability	able	abortion	about	absence	absolute	absolutely	absurd	...	yourselfgo	youth	youtube	youve	ytmndin	yugoslavia	zealand	zero
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0

In order to create it, we have decided to use the TF-IDF Vectorizer¶ from sklearn. This vectorizer is converting a collection of raw text to a matrix of TF-IDF features. Tf means term-frequency while tf-idf means term-frequency times inverse document-frequency (from documentation). It is quite interesting on how the TF-IDF works. Basically, instead of pointing out the relevance/frequency of token into a given text, its role is to scale down the impact of very frequent token which are less significant than rarer one. This is helping us to balance the potential unbalance of token across a specific text.

Multi Output Classifier:

The next step is to initialize a Multi Output Classifier. Since we need to classify each sentence as toxic or not, severe_toxic or not, obscene or not, threat or not, insult or not and identity_hate or not, we need to classify the sentence against 6 output variables (This is called Multi-Label Classification which is different from multi-class classification where a target variable has more than 2 options e.g., a sentence can be positive, negative and neutral.

For the same, we will be using MultiOutputClassifier from sklearn which as mentioned earlier is a wrapper. This strategy consists of fitting one classifier per target. The main algorithms we will be using over the Multi Output Classifier are:

- Logistic Regression
- Random Forest
- Multinomial Naïve Bayes

We will than keep the model that is better performing based on our AUC-ROC evaluation metrics.

Evaluate the Model

The final AUC-ROC score for each model is:

MODEL	AUC-ROC
Logistic Regression	64.38%
Logistic Regression with Hyperparameter	64.68%
Random Forest	72%
Multinomial Naïve Bayes	60%

By using all the different model mentioned before, we will be selecting the one which is based on the Random Forest model. This is because it has scored over 72% on our AUC-ROC metrics, while all others have been under this value.

We cannot say that 72% is a good score for a model to generalized and scaled up to other business problems. Nevertheless, it can be surely a good starting point and a benchmark for other more complex algorithms.

To conclude, we have applied our Random Forest model to our Test.csv, we have obtained the following:

	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	Yo bitch Ja Rule is more succesful then you'll...	0.466611	0.480072	0.475605	0.386971	0.491111	0.466479
1	== From RfC == \n\n The title is fine as it is...	0.556693	0.584647	0.560179	0.503448	0.552780	0.559693
2	" \n\n == Sources == \n\n * Zawe Ashton on Lap...	0.559679	0.590841	0.560179	0.503448	0.555667	0.559693
3	:If you have a look back at the source, the in...	0.552824	0.573864	0.551448	0.484232	0.532400	0.540578
4	I don't anonymously edit articles at all.	0.559679	0.590841	0.560179	0.498292	0.555667	0.559693

Per each row in comment_text, we have been able to extract the probability that one comment is to be classified in one class compared to another.

Embeddings models:

In this section, we will go behind the scope of a simple "Bag of Words" representation to use models that are built around word embeddings.

Word Embeddings are text representation that allow similar words to be represented similarly. More specifically, they can be considered as a distribution representation for text. Each word is represented by a vector which is than place in a predefined vector space. An example of this representation can be summarized by the picture below, where word/vector are represented into a vector-space framework based on their characteristics (e.g., similarity):

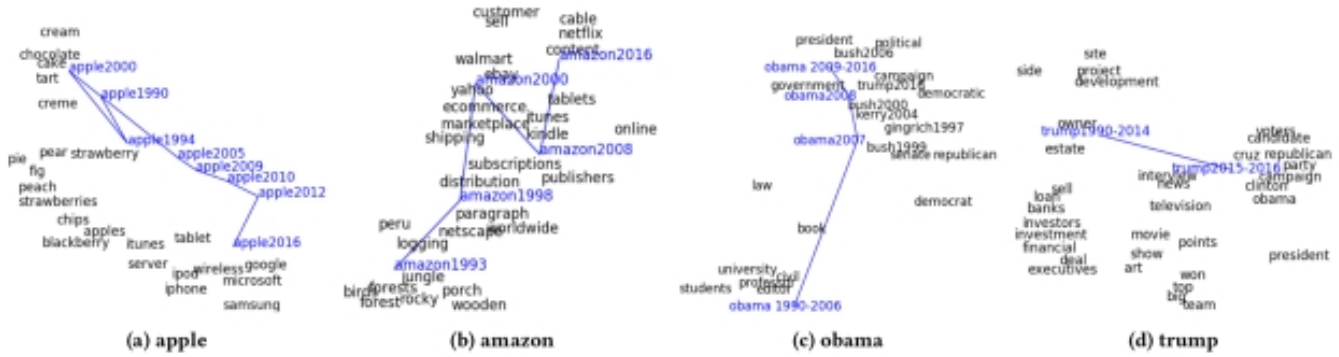


Figure 1: Trajectories of brand names and people through time: `apple`, `amazon`, `obama`, and `trump`.

The mapping word-vector is realized through learning technique that resembles a neural network. Compared to one-hot encoding, the embedding technique has the advantage to represent word within ten/hundreds of dimensions Vs. the millions required for the Bag of Words.

Compared to the project proposal, where we were assuming of using a Word2Vec model, we have decided to select other two embedding algorithms that we believe will lead to better performance:

1. Glove
2. FastText

FastText:

FastText is a word embedding technique that is an extension of Word2vec. Instead of learning vectors for words directly, FastText represents each word as an n-gram of characters. So, for example if we take the word “artifact” with n= 3, the FastText representation will be “ar” “art” “rti” “tif” etc. It enables to better understand the meaning of shorten word and to understand prefix- and -suffix. One of the added values of this model is that is treating well the OOV - Out of vocabulary words. We will see that by using Glove, we will recall a pre-trained dictionary of embeddings, which of course can lead to the problem – what happen if an embedding has not been trained? The model will probably fail in the prediction, while the FastText structure will also cover this issue.

Glove:

Overall, it is an approach that is considering both the statistics of matrix factorization (like frequency), but also that is including a more meaning-specific technique. The idea is not only to consider how a word-vector is represented and where, but also the distance between other vectors which at the end represents a distance in meaning of the word as such.

In our project, we will be using a embedding vocabulary trained on Wikipedia 2014 + Gigaword 5. A model composed of 5 billion tokens, 400K words, and 300d vectors.

As mentioned before, the structure of word-vector embeddings learning is like the one of Neural Network. This is the reason why we have decided of using deep learning algorithms to work with the pre-trained dictionary of embedding provided by Standford.

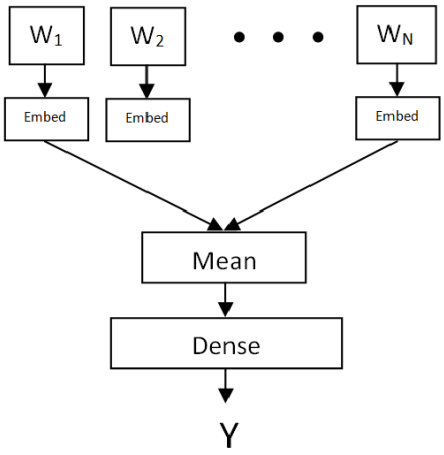
FastText:

FastText is an algorithm developed by Facebook engineering in 2016. As mentioned, before it takes the logic behind Word2vec representation a step ahead and learns representations for character n-grams.

The main benefits of this model are:

- Avoiding the problem of Out of vocabulary (OOV)
- model ignores the internal structure of words as it is using distinct vector representation.

An illustrative representation of FastText word embedding can be found below:



In order to apply FastText to our project, we will:

1. Pre-process our dataset to clean the main data text issues and prepare the classes in the FastText format.
2. Split our Train.csv dataset into train and test (validation) data
3. Train the model
4. Evaluate the model

Data pre-processing:

In this part of the notebook, we will mainly apply the same cleaning function as the one we have used in the Bag of Words model.

We will have also to transform the hot encoding of each class 0 and 1 to __class__0 and __class__1. This is the vocabulary accepted by FastText algorithm.

Data Split:

As before in Bag of Words, we will split our Train dataset into Train and Test data (validation). We have decided to use 33% of our original dataset to be used to validate our final model.

```
#Prepare the dataset to be splitted into Train and Test data
classes = ['toxic', 'severe_toxic', 'obscene', 'threat','insult', 'identity_hate']

X = df_train.comment_text  #predicotors
y = df_train[classes] # independent variable

# Split of the dataset. Test size 33% and random seed = 42
X_train, X_test = train_test_split(df_train, test_size=0.33, shuffle = True, random_state=42)
```

As in the Bag of Words model, it is important to remember that the Test dataset is not the one that will be used for the final exercise, i.e., Test.csv that will be the one used to scale our model.

FastText model and Evaluation:

In order to predict the probability of each class, we have decided to use the "train supervised" function of FastText, which is enabling us to predict a text based on supervised learning method.

We have first to save our data frame into a .csv file as a FastText model receive as input only .csv file.

We have then loop over the test dataset (validation) to measure the accuracy of the model.

At the end we have associate each probability to the right row/column, by transposing the results of our model and we have calculated the mean AUC-ROC over each class to evaluate our model. The result is a mean AUC-ROC of 77% probability to evaluate our model, which is slightly above the 72% performed by the Random Forest model.

The embedding used by FastText is therefore a step forward into increasing the model performance. Nevertheless, it is still underperforming from a business point of view, and it will require further improvement to be scaled to other classification problems.

Glove and Bidirectional LSTM model:

In this section of the project, we have decided to use the Standford algorithm for word embedding, Glove.

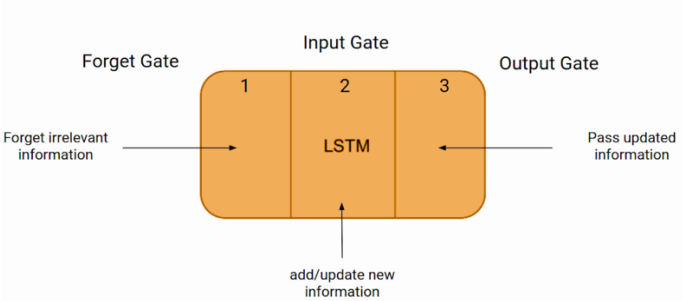
Basically, we have decided to import the word embedding trained by the Standford university, which include over 400K words and 5 billion tokens that will be at the basis of our learning model.

We will be using a deep learning algorithm, bidirectional Long Short-Term Memory Networks (LSTM) to re-train our model over the pre-trained embedding imported from Standford University.

Before implementing the project, it will be needed to give an overview of what is a LSTM. LSTM is a type of RNN, or i.e., Recurrent Neural Network. We can think a RNN as a sort of memory storage. In order to predict a word, the RNN is remembering what word appeared in the previous time step. LSTM overcome the problem of memory storage, by adding the concept of short-term memory.

The LSTM consists of three parts, as shown in the image below and each part performs an individual function. The first part (Forget Gate) chooses whether the information coming from the previous timestamp is to be remembered or is irrelevant and can be forgotten. In the second part (Input Gate), the cell tries to learn new information from the input to this cell. At last, in the third

part (Output), the cell passes the updated information from the current timestamp to the next timestamp.



The idea of using a Bidirectional LSTMs, it is quite simple. In problems where all timesteps of the input sequence are available, Bidirectional LSTMs train two instead of one LSTMs on the input sequence. It basically involves duplicating the first recurrent layer in the network so that there are now two layers side-by-side, then providing the input sequence as-is as input to the first layer and providing a reversed copy of the input sequence to the second.

In this section of the project, we will therefore apply a Dee Learning technique, used to analyze advance problems like speech recognition, to perform our classification.

In order to do that, we will:

1. Pre-process our dataset: we will mainly tokenize our text
2. Load our pre-trained set of embeddings
3. Train the model
4. Evaluate the model

Bidirectional LSTM:

As the part of processing the text and load the pre-trained embedding is quite straight forward, we will analyze directly our bidirectional LSTM models. The structure of the model look like the below:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 1000, 300)         63101400
_________________________________________________________________
bidirectional (Bidirectional (None, 1000, 100)         140400
_________________________________________________________________
global_max_pooling1d (Global (None, 100)               0
_________________________________________________________________
dense (Dense)                (None, 50)                5050
_________________________________________________________________
dropout (Dropout)            (None, 50)                0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 306
=================================================================
Total params: 63,247,156
Trainable params: 63,247,156
Non-trainable params: 0
```

We have basically 6 layers:

1. The first one is the embedding layers, which is including all the pre-trained vectors from the Standford University.
2. The second one is applying a bidirectional LSTM to our model. It is basically creating two side-by side layers. To the first one we will inject the embedding input, while to the second we will provide a reverse copy of the Standford embeddings.
3. The third layer is composed by a Global Max Pooling. This layer is needed to reduce dimensionality. We see in fact that the shape of our 3-D array has changed to a 2-D array.
4. A dense layer: which is a set of neurons receiving inputs from all other neuron in the previous layer.
5. A drop-out layer: where we randomly select some neuron information and discard other.
6. And again, another dense layer.

LSTM Evaluation:

The results of our model, only with 2 epoch is quite impressive as we obtain a total AUC-ROC of 98% within the best model performance. This is way above the performance of the Random Forest or of the FastText model.

```
Epoch 1/2
752/752 [==============================] - 4410s 6s/step - loss: 0.0745 - auc: 0.9541 - val_loss: 0.0512 - val_auc:
0.9793
Epoch 2/2
752/752 [==============================] - 4214s 6s/step - loss: 0.0459 - auc: 0.9826 - val_loss: 0.0497 - val_auc:
0.9827
```

It is also a result that give us enough confidence to scale the model to other problem that may be different form the toxicity classification that we have experimented up to now.

Conclusion

In this project we have applied some NLP algorithms to our dataset to analyze the level of toxicity of the text comments extracted from Wikipedia.

We have applied the simplest NLP model, Bag of Words, that is at the basis of any text classification problem. The algorithm used has performed on average, with a total mean AUC-ROC of 72%; followed by the 77% of FastText. The best one has been so far the bidirectional LSTM, where the deep learning technique has scored a very high 98%.

I am sure that also the less performing model could have been better trained with some hyperparameter tunning or with ad hoc data pre-processing based on the characteristics of the model used.

Nevertheless, the LSTM has surely meet the expectation and it can be used not only for our classification problem, but also for future more complex problems. We are therefore more that satisfy with it.