Capstone Project

Comment Toxicity Classification Using Neural Networks and Machine Learning Techniques

Machine Learning Engineer Nanodegree

Fabio STEFFENINO
October 4th, 2019

I. Definition

Project Overview

The argument of the research I would like to tackle and develop as final capstone project for my Machine Learning Engineer Nanodegree, it is related to the usage of Supervised Learning, Neural Networks and NLP techniques in comment toxicity classification.

As a future machine learning engineer, I am really interested in understanding how NLP works and how to solve related problems, as a father, I am concerned about online comment cruelty and cyber-bullying.

The online world has become entirely part of our lives. One of the biggest problem of the cyberspace is the one identified with the term **Online Disinhibition Effect**, in other words the lack of restraint one person feels when communicating online in comparison to communicating in-person (Wikipedia definition). There are two Online Disinhibition categories: *Benign* and *Toxic*. The Toxic Online Disinhibition represent a situation where people, with the help on anonymity in certain cases, have an inappropriate behaviour on platforms like blogs or social networks where they feel free to comment using hostile or derisive language.

Google and Jigsaw co-founded a research initiative, called Conversation AI, that is working on tools to help improve online conversation. One area of focus is the one highlighted before: the study of online negative behaviour and toxic comment identification. In 2017 they release a free tool, Perspective API, that use machine learning to score the perceived impact a comment might have on a conversation. Their first version of the model identifies whether it could be perceived as "toxic" or not to a discussion. In order to improve their model performances, they hosted a Kaggle competition detailed at Toxic comment classification challenge.

Google and Jigsaw are providing on kaggle a large dataset of Wikipedia comments, manually labeled by human raters for toxic behaviour, to be used for this challenge and which we will analyse later in this paper.

Researchers spent a lot of time in the last years working on Sentiment Analysis and Comment Toxicity classification and Natural language Processing with Neural Networks appear to be the most used technique when solving this kind of problems.

This will be my first approach to NLP problems and my goal is, first, to learn how to design a Neural Network for text processing and, second, how to apply what I learned to the problem.

Problem Statement

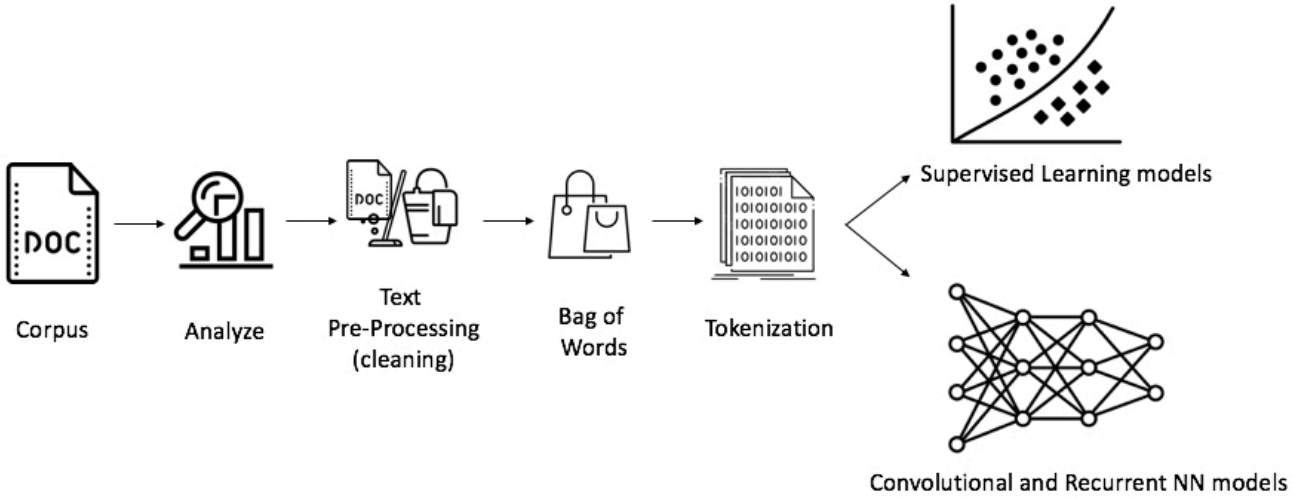
Given the wikipedia comment dataset provided on kaggle, the scope of this competition is to build a multi-headed model that is capable of detecting different types of toxicity like *threats, obscenity, insults, and identity-based hate* better than Perspective's current models and providing for each comment the probability that it falls under each class (*Multi-Label classification*).

The very first objective of our work is to be able to beat the random prediction with a score higher than 0.5. We will then start by creating baseline models using supervised machine learning techniques. During the course we saw that **Multinomial Naive Bayes** is a good candidate for text classification (ex: spam / not-spam). Lot of researcher obtained good result using as well a **Logistic Regression** model. So we will compare those two along to a tree based model (**Random Forest Classifier**).

We will start by analyzing the dataset exploring its charateristics and what are the most used words in a given class.

Text pre-processing is a very important phase of NLP problem so after the first dataset analysis we will build a function to clean the text corpus removing STOPWORDS, numbers and correcting the abbreviated form with the extended one (ex: *what's* is becoming *what is*)

As we know well from the course, in machine learning the input for the models should be numerical and not textual. Our text data requires special preparation before usage and it needs to be converted into numbers. The **bag-of-words** (BoW) model + tokenization will helps us achieving this goal.



Once we have analysed the data, preprocessed the texts and evaluated our baseline models, we will try to obtain the same performances by making use of **Convolutional Neural Networks** and **Recurrent Neural Networks**.

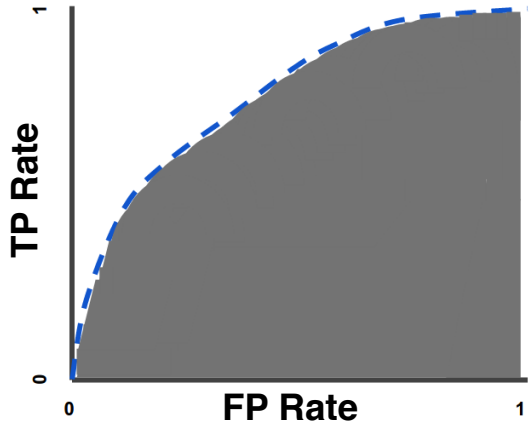
We will also compare the results in case of usage of simple word tokenization and tokenization plus pre-trained embedding with GloVe (Global Vectors). We will see more in details later what is GloVe and how to use it.

Metrics

The initial metric proposed by Kaggle was the **LogLoss** function but it has been later changed into **AUC-ROC**. So we will be using this last one in order to evaluate our model.

An **ROC curve** (**receiver operating characteristic curve**) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- TP rate (True Positive Rate)
  - TPR = TP/(TP+FN)
- FP rate (False Positive Rate)
  - FPR = FP/(FP+TN)



AUC-ROC is the **Area Under the Receiver Operating Characteristics Curve**. It is a particular good metrics because it tells how much the model is capable of distinguish between the different classes. In a multi-label classification problem the global AUC-ROC score is calculated by taking the average of the individual AUCs of each predicted category column.

AUC can have value between 0 and 1. A model with 100% of correct prediction has a AUC score value of 1, a model with 100% of wrong prediction has a AUC score value of 0.

II. Analysis

The code for the data anaysis can be found in the notebook Toxic_comment_classification_Data_Analysis

Data Exploration

As we said before, the dataset is provided by Kaggle for both training and testing in two different files in CSV format. We can see a snapshot of the dataset hereafter:

```
train_set.drop( ['id'], axis=1).head()
```

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

The dataset called "Wikipedia Talk Page Comments annotated with toxicity reasons" can be found at the following link: [Toxic comment classification challenge dataset](#).

Both Train and Test dataset are CSV files containing approximately 160,000 comments each (train set 159571 - test set 153164) and they have been all manually labeled by human raters for toxic behaviour.

```
train_set.shape
```

```
(159571, 8)
```

```
test_set.shape
```

```
(153164, 2)
```

The given toxicity categories are the following:

- *Toxic*
- *Severe Toxic*
- *Obscene*
- *Threat*
- *Insult*
- *Identity Hate*

There are 8 columns in the train dataset. ID, COMMENT_TEXT and one column for each category with value 0;1 (1 means that the comment falls under that category, each comment can be part of more than one category). The ID column is not usefull for training so we will get rid of it. Comment_text will need to be pre-processed and tokenized in order to be used in our model.

The test dataset contains instead only ID and COMMENT_TEXT. In this case we cannot get rid of ID column because it will be used for our submission to Kaggle.

Here are some example of comments:

```
train_set['comment_text'][290]
```

```
"Another question, does this style information stripping occur only whene
ver a shape itself is edited, or for less intrusive edits, such as moving
a shape around? If even highlighting and selecting the shape in Inkscape
makes it unusable, then I shouldn't even be messing with Inkscape at all.
"
```

```
train_set['comment_text'][6]
```

```
'COCKSUCKER BEFORE YOU PISS AROUND ON MY WORK'
```

The first one is classified as clean and the second one is classified as toxic, severe_toxic, obscene and insult. One of the first things we can notice is that the comments vary a lot in term of size (number of characters and words).

```
temp_words = train_set.drop( toxic_columns, axis=1 )
temp_words = temp_words.drop( ['id'], axis=1 )
temp_words['count_word'] = temp_words["comment_text"].apply(lambda x: len(str(x).split()))
temp_words['count_char'] = temp_words["comment_text"].apply(lambda x: len(str(x)))
temp_words.describe()
```

	count_word	count_char
count	159571.000000	159571.000000
mean	67.273527	394.073221
std	99.230702	590.720282
min	1.000000	6.000000
25%	17.000000	96.000000
50%	36.000000	205.000000
75%	75.000000	435.000000
max	1411.000000	5000.000000

At this step we didn't clean yet the corpus but we can start having a roughly idea about the number of words and char we have by comments in our dataset. We can see from the data printed above that the majority of comments has an average of 67 words and 394 characters, the shortest has 1 word and the longest 1411. In terms of characters, the shortest has 6 chars and the longest has 5000.

There are clearly few comments having an unusual number of words but we do not worry too much at this step cause this is going to change after text cleaning phase and tokenization.

On last obeservation about the data is regarding the class tagging. Very few comments are flagged as "toxic" and we can see it in the following table:

```
main_set = train_set.drop(['id'], axis=1)
main_set['not_toxic'] = 1-main_set[toxic_columns].max(axis=1)
main_set.drop(['comment_text'], axis=1).sum()
```

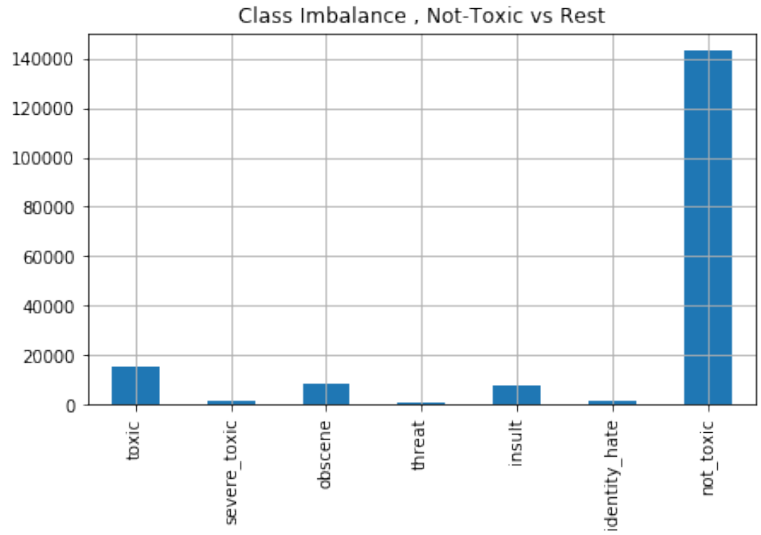
```
toxic             15294
severe_toxic       1595
obscene            8449
threat              478
insult             7877
identity_hate      1405
not_toxic        143346
```

We computed a new column called *not_toxic* that is equal to 1 in case of comment not flagged as any other category. We have 143346 clean data over 159571 total comments. It is a clear case of **class imbalance**. A model classifying every comments as clean would reach an accuracy of over 0.8 so we need to be very carefull during the evaluation. We learnt during the course, when talking about spam detection, that If we have an imbalanced dataset **accuracy** can give us false assumptions regarding the classifier's performance and it's better to rely on **precision** and **recall** (this particular case is a high recall problem). That's why instead of using accuracy we will use **ROC-AUC** (Area Under the Receiver Operating Characteristics Curve) mentioned above that also is an excellent metric for a Multi-Label classification problem.

Usually, with neural networks, class imbalance can be sorted out by the model given enough passes through the training data (epochs).

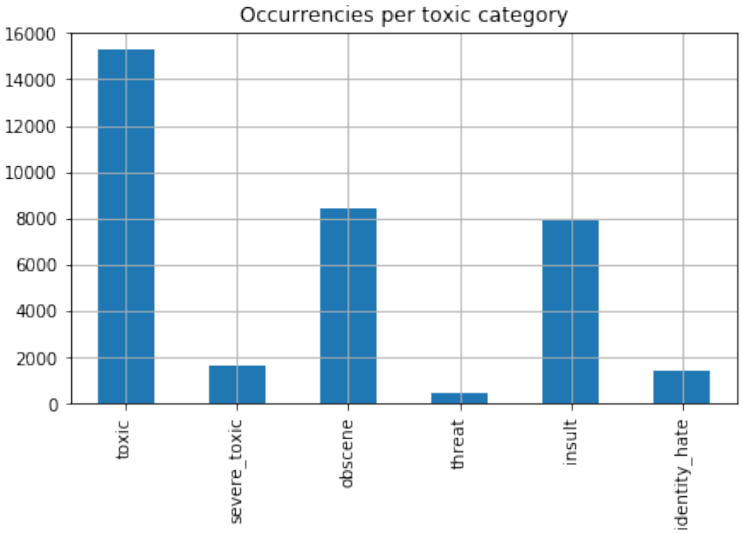
Exploratory Visualization

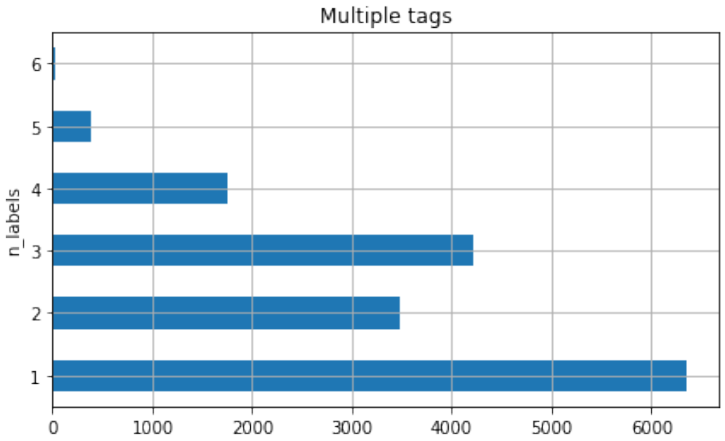
Let's plot now some information related to the labels.



We already discussed about **class imbalance** in the previous session but it is important, for the visula effect, to plot it using histograms. We can see it in the previous image and we can easily conclude that there are over 140k clean comments against less than 20k toxic ones. Removing the not_toxic column we can see more closely the number of comments classified in each toxic category. Most of them are simply *Toxic* while Very few are classified as *Threat*, *Identity_Hate* or *severe_toxic*.

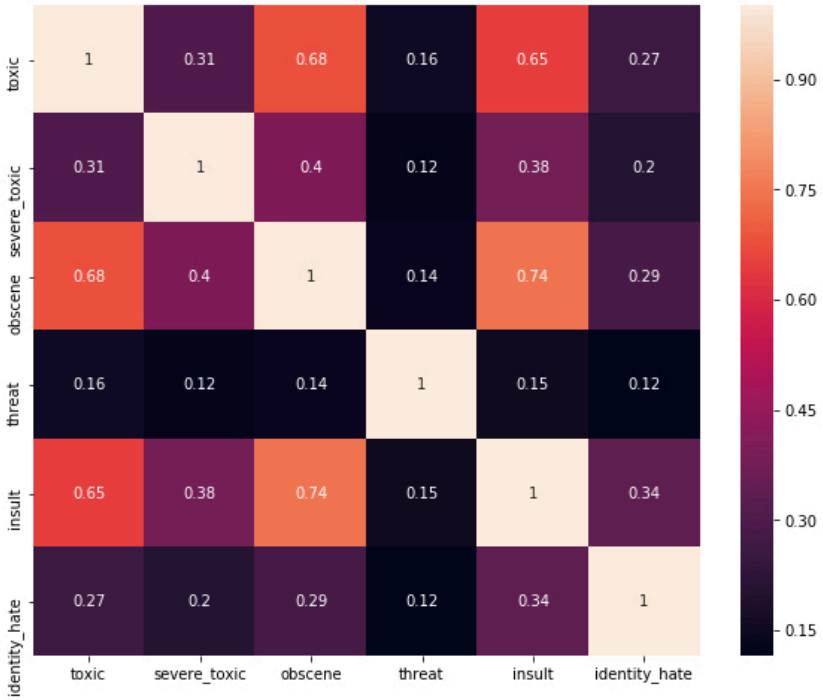
As we have a total of 159k comments in our dataset and 140k are clean, looking the figure below we can clearly deduct that most of the toxic comments have more than one category and they are **multi-tagged**.





Apart from the comments having only one toxic category, it is interesting to see in the previous plot that over 4000 have 3 tag and also that there are few with 6 tags.

The matrix below show the correlation between labels. We can deduct from this image that if a comment is Toxic, it is likely to be Obscene and Insult as well. And if a comment is classified as Obscene there are high chances it is classified as Insult as well.



Algorithms and Techniques

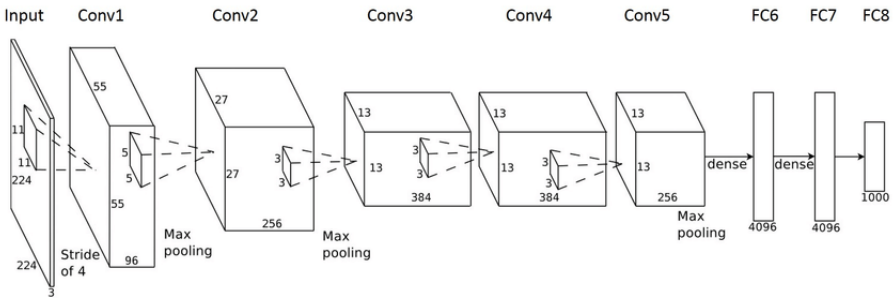
In order to solve this classification problem we will make use of two Deep Learning models: Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN).

We are in a natural language processing (NLP) problem and the very first step will be the text pre-processing. We will need to transform the raw corpus text into numerical vectors that can be given in input to our models. We will use the bag of words technique in conjunction of the tokenizer and padding classes provided by keras. This will allow us to create a dictionary of used words, to assign an index to each of them and finally pad the sequences of indexes by adding zeros at the end in order to have vectors of the same size (we will see the output of this step in the data pre-processing session).

In an attempt to improve our model and get the extra edge(after reading this article), we tried to use **Glove** pre-trained vectors that allows us to build a word co-occurrence matrix that basically count how frequently a word appears in a context. We used the pre-trained word vectors glove.6B.50d.txt that can be found on Glove website.

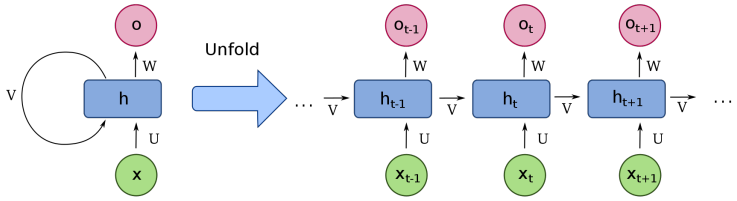
After text vectorization we build 3 neural networks architectures and evaluated their output: Standard Convolutional Neural Network, Recurrent Neural Network with LSTM cell and Recurrent Neural network With LSTM and Glove embedding.

- Convolutional Neural Network



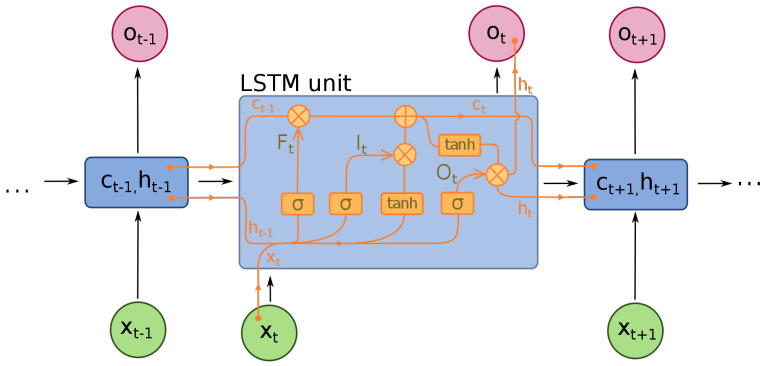
A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input, assign importance (learnable weights and biases) to various aspects/objects of it and be able to differentiate one from the other. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. CNN are mostly used for Image and Video classification but we will see how they behave when they need to classify text. *ref*

- Recurrent Neural Network



CNN have been designed to simulate the neuron of humain brain but they have an issue. Humans don't start their thinking from scratch every second. Our thoughts have persistence. Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

- Long Short-Term Memory Cell



Long Short Term Memory networks are a special kind of RNN capable of learning long-term dependencies. In LSTM the hidden layer receive the input from the input layers and also from the other hidden layers making them able to remember information for long periods of time.

All the models above have been widely used by researcher in solving NLP classification problem, especially LSTM. We will try to build our own model and get some good results.

Benchmark

The code for the benchmark models can be found in the notebook Toxic_comment_classification_NB_LR_RF

On the net we can find a lot of research papers and even on kaggle the top kernels have been made public. Even if I could use one of those as benchmark model I decided to build my own.

I was actually curious to compare the results of different supervised learning models and i selected the following ones:

- Naive Bayes
- Logistic regression
- Random Forest classifier

I decided to use Naive Bayes for its semplicity and because it has been used during the course when discussing the spam email classification (similar to our problem). Logistic regression because it is one of the most used on kaggle for this competition. Finally Random Forest because i wanted to see how a tree based model would perform on such problem.

So, I first cleaned the data, then I encoded the text into vectors of numerical values using the tf-idf vectorizer and finally ran the benchmark models on the transformed data.

I used MultinomialNB, LogisticRegression and RandomForestClassifier from sklearn library with the following parameters:

```
'NaiveBayes': MultinomialNB(),
'LogisticRegression': LogisticRegression(solver='sag'),
'RandomForest': RandomForestClassifier(n_estimators=100, max_depth=3)
```

Hereafter we can see the ROC-AUC score of each model.

train dataset	**MultinomialNB**	**LogisticRegression**	**RandomForestClassifier**
toxic	0.87	0.97	0.87
severe_toxic	0.88	0.98	0.89
obscene	0.88	0.98	0.87
threat	0.76	0.98	0.75
insult	0.87	0.97	0.88
identity_hate	0.82	0.97	0.88
AVG	0.84	0.975	0.85

I also performed the same pre-process on the test dataset and generated the predictions for each of the three models. As kaggle didn't provide the label file for the test dataset I have submitted the prediction on the challenge page and I got the following scores:

test dataset	**MultinomialNB**	**LogisticRegression**	**RandomForestClassifier**
Kaggle score	0.85244	0.97424	0.82695

Knowing the class imbalance we can say the NaiveBayes and RandomForest didn't perform well while LogisticRegression confirmed itself as the supervised model to be used in this kind of problems.

We will try to surpass these score with our CNN and RNN models.

III. Methodology

The code for the following sessions can be found in the notebooks Toxic_comment_classification_NN, Toxic_comment_classification_RNN_LSTM and Toxic_comment_classification_RNN_Glove

Data Preprocessing

We are in an NLP problem and data pre-processing for us means text preprocessing. We performed the following actions:

- Text String cleaning

```
def normalize_text(comment):
    comment = comment.lower()
    comment = re.sub(r"i'm", "i am ", comment)
    comment = re.sub(r"\'s", " ", comment)
    comment = re.sub(r"\'ve", " have ", comment)
    comment = re.sub(r"can't", "can not ", comment)
    comment = re.sub(r"n't", " not ", comment)
    comment = re.sub(r"\'ll", " will ", comment)
    comment = re.sub(r"\'re", " are ", comment)
    comment = re.sub(r"\'d", " would ", comment)
    comment = re.sub(r"what's", "what is ", comment)
    comment = re.sub(r"\'scuse", " excuse ", comment)
    comment = re.sub("[^a-z]", " ", comment)
    comment = comment.strip(' ')
```

  First we lowered the text. Then we realized there were a lot of short forms of the verbs so we changed it into the long form and finally we kept only the characters.

- Stopwords removal

```
stop_words = set(stopwords.words('english'))
for word in stop_words: #removing stopwords
    token = " " + word + " "
    comment = comment.replace(token, " ")
    comment = comment.replace("  ", " ")
```

  Then we removed the stopwords. Stopwords are words commonly used in verbal and written communication and do not carry any usefull information for the classification task. So we get rid of them.

- Dictionary creation

```
#create dictionary text
tokenizer_train = Tokenizer( num_words=max_words, oov_token=oov_tok )
tokenizer_train.fit_on_texts( train_corpus['comment_text'] )

#indexes of our bag of words
dict_index = tokenizer_train.word_index

#bag of words reverted
reverse_index = dict()
for key in dict_index:
    reverse_index[dict_index[key]]=key
```

  We created then our bag of words using Tokenizer by keras with a maximum number to words equal to 50000.

- Tokenization and Padding

```
#create padded sequences train  set
sequences = tokenizer_train.texts_to_sequences(train_corpus['comment_text'])
padded_train_corpus = pad_sequences(sequences, maxlen=max_length, padding=pad_type, truncating=trunc_type)
```

  And finally we tokenized every sentence using the dictionary created before and padded them in order to have vectors of the same size. We chose padding and truncating type "post" and maximum size of 200

Hereafter we can see an example of Tokenization - Padding and decoding.

```
sequences[1]
```

```
[2308, 15862, 2427, 427, 3541, 4260, 2514, 22, 5, 832, 88]
```

```
padded_train_corpus[1]
```

```
array([ 2308, 15862,  2427,   427,  3541,  4260,  2514,    22,     5,
         832,    88,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0,     0,     0,     0,     0,     0,     0,     0,
           0,     0], dtype=int32)
```

```
train_corpus.iloc[1].comment_text
```

```
'd aww matches background colour seemingly stuck thanks talk january utc'
```

```
#function decoding a list of word indexes
def decode(sentence):
    comment = ""
    for el in sentence:
        comment=comment+" "+str(reverse_index[el])
    return comment
```

```
print(decode(sequences[1]))
```

```
 d aww matches background colour seemingly stuck thanks talk january utc
```

Implementation

We already discussed the text-preprocessing phase and the results obtained with our baseline benchmarking supervised lerning models so It's finally time to code our deep learning neural networks.

We used keras from tensorflow library.

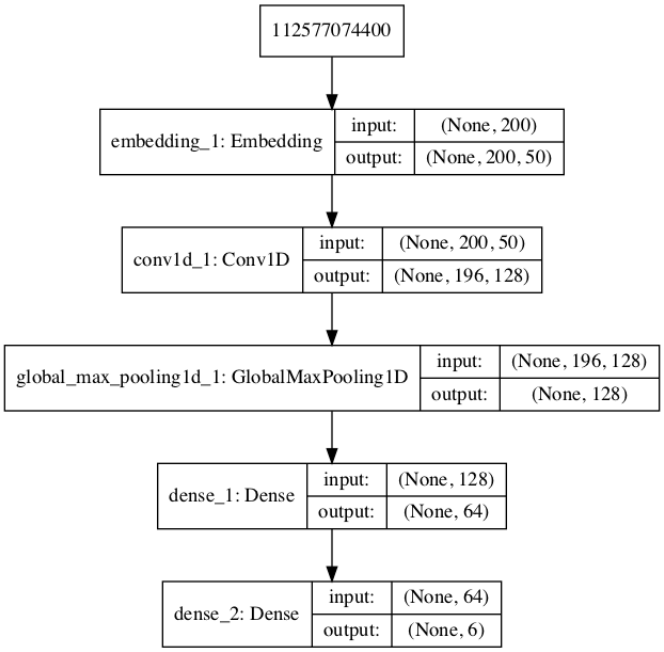
As said before, we tested 3 different architecture: ConvolutionalNeuralNetwork, RNN with LSTM and finally RNN with LSTM with Glove pre-trained embedding.

Talking about the first one, we can see hereafter how we defined the layers.

```
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=max_length))
model.add(Conv1D(128, 5, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(64, activation='relu'))
model.add(Dense(6, activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```

We can see that we have an Embedding layer getting the tokenized-padded vectors in input and passing the output to a **Conv1D** layer with 128 filters, convolution window of size 5 and a relu activation function. Activation functions are used to transform the weighted sums of inputs of a layer into an output value. *Relu* is the most used activation function. It produce 0 if the input x < 0 and x if the input x >= 0. One Characteristic of Relu is that it help fighting the vanishing gradient problem. Then the ouput goes into a *GlobalMaxPooling1D* layer in order to reduce the dimensionality. Finally we have two *Dense* fully connected layers with the last one having 6 nodes (one for each toxicity class). Note that for the last Dense Layer we are not using Relu as activation function but *Sigmoid*. Sigmoid takes a real value as input and outputs another value between 0 and 1 that in case of classification problem can be interpreted as the probability of the input to be classified as class X.

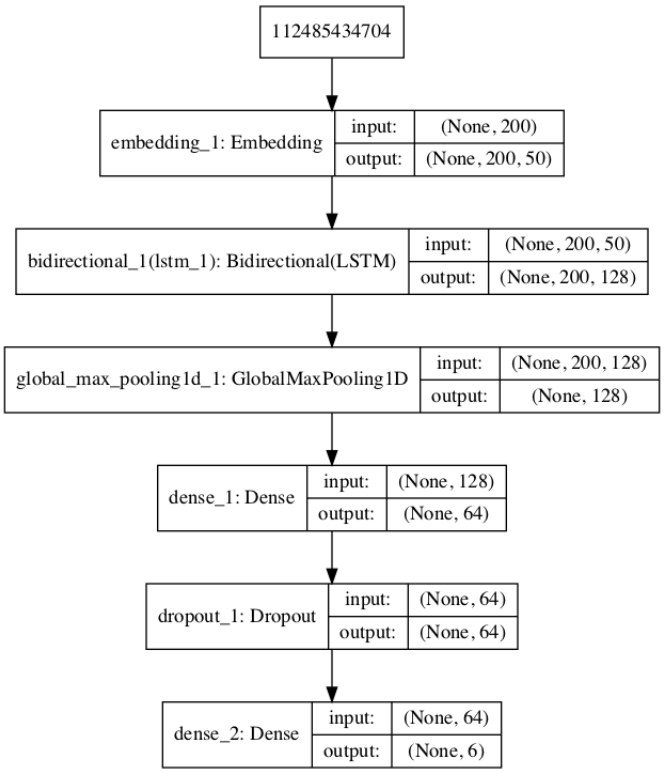


We then finally compiled the model using *binary_crossentropy* as loss function because we are in a multi-label classification problem (if we were in multi-class we would have been using categorical_crossentropy) and adam algorithm as optimizer.

The second model we built was a Recurrent Neural Network with Bidirectional LSTM cell. We can see hereafter the architecture that is very similar to the previous with the exception for two hidden layers.

```
model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=max_length))
model.add(Bidirectional(LSTM(64, return_sequences=True, recurrent_dropout=0.2)))
model.add(GlobalMaxPooling1D())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(6, activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```



Instead of the Conv1D layer we have an **LSTM** cell with 64 units (that result into 128 nodes in total as it is bidirectional) and recurrent_dropout set to 0.2. And Finally we added a **Dropout** layer between the two fully connected Dense layers. Dropout consists in literally ignoring units/neurons (chosen randomly) during the training phase. The usage of Dropout in Neural Networs allow us to build a model avoiding overfitting. With dropout set to 0.2 it means we are ignoring 20% of the units at each forward or backward pass.

Refinement

```
glove_file='./glove.6B/glove.6B.50d.txt'
```

```
embed_weights = dict()

f = open(glove_file)
for line in f:
    el = line.split()
    # at position 0 we have the word and then all the coefficients
    embed_weights[el[0]] = np.asarray(el[1:], dtype='float32')
f.close()
```

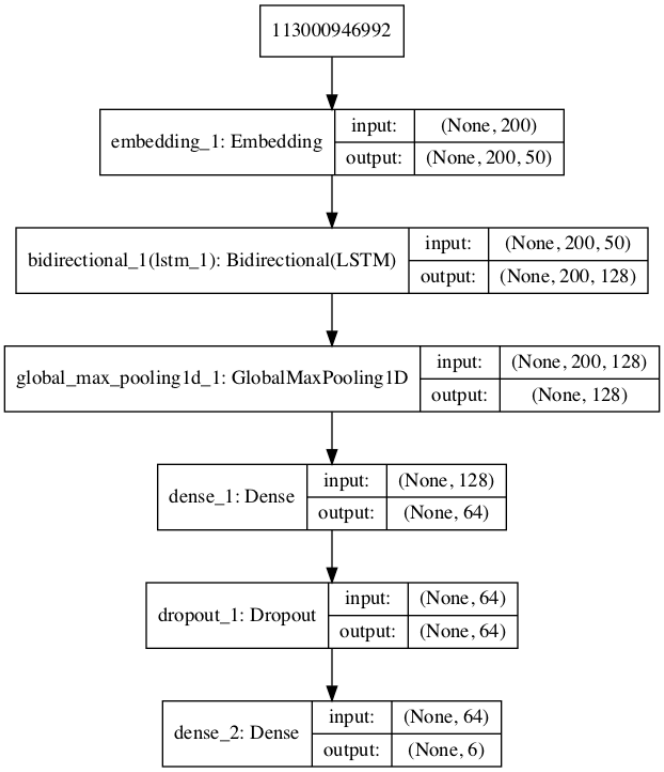
```
weights_matrix = np.random.random((len(dict_index) + 1, embedding_dim))
for word, i in dict_index.items():
    weights_vector = embed_weights.get(word)
    if weights_vector is not None:
        weights_matrix[i] = weights_vector
```

The third and final model has been explored in a tentative to get that extra edge needed in order to beat our best benchmarking model.

We tried to used Glove pre-trained embedding with the previous RNN model with bidirectional LSTM cell. The implementation is exactly the same as before, exception made for the embedding layer.

```
model = Sequential()
model.add(Embedding(weights_matrix.shape[0], embedding_dim, weights = [weights_matrix],
                    input_length=max_length, trainable=False))
model.add(Bidirectional(LSTM(64, return_sequences=True, recurrent_dropout=0.2)))
model.add(GlobalMaxPooling1D())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(6, activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.summary()
```



More informations about GloVe can be found here.

IV. Results

Model Evaluation and Validation

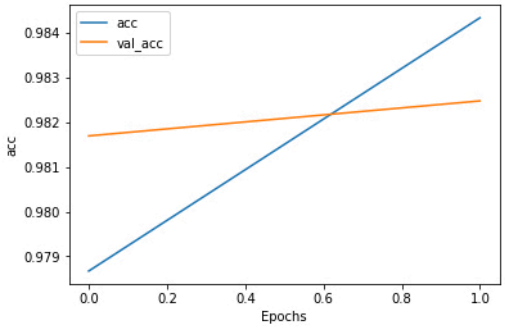
Before fitting our DNN models over 2 epochs, we split the training dataset into training and validation (we reserved 20% of the dataset for validation).

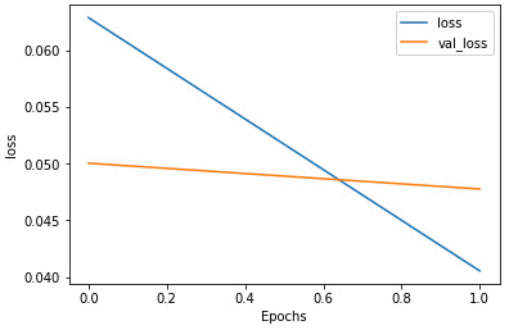
```
idx = int(0.2*padded_train_corpus.shape[0])
x_train = padded_train_corpus[: -idx]
y_train = train_labels[: -idx]
x_val = padded_train_corpus[-idx: ]
y_val = train_labels[-idx: ]
```

```
num_epochs = 2
history = model.fit(x_train,
                    y_train,
                    validation_data=(x_val,y_val),
                    epochs=num_epochs,
                    batch_size=32)
```

It is now time to explore the results of the models over the training dataset. They all took quite a lot of time in order to complete training.

- Conv1D

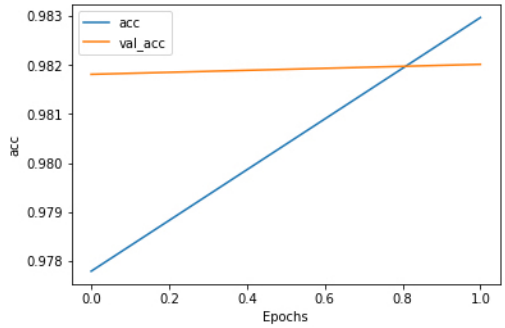


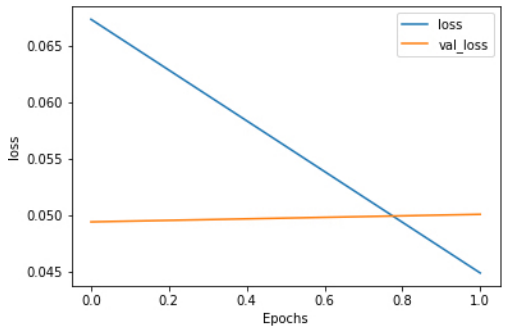


  After 2 epochs:

  loss: 0.0405 - acc: 0.9843 - val_loss: 0.0478 - val_acc: 0.9825

- LSTM

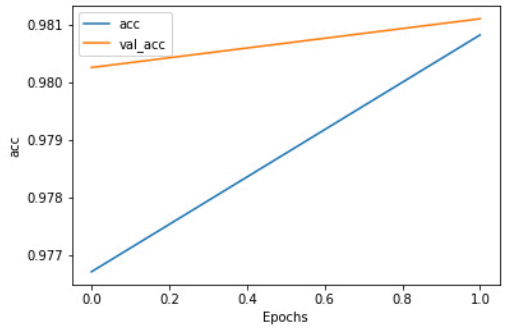


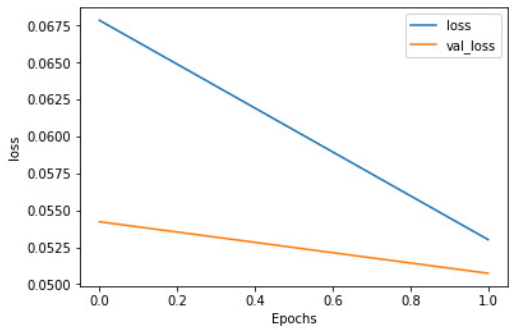


After 2 epochs:

loss: 0.0449 - acc: 0.9830 - val_loss: 0.0501 - val_acc: 0.9820

- LSTM with Glove





After 2 epochs:

loss: 0.0530 - acc: 0.9808 - val_loss: 0.0507 - val_acc: 0.9811

From the above plot we can deduct that after only 2 epochs we are already overfitting our model. This is due for sure to the very basic architecture we have chosen. One observation we have to make is regarding the LSTM with GloVe plot. We can see that validation accuracy is higher than training accuracy. This is linked to the usage of dropouts in the model during training.

Keras is not providing roc_auc in the list of usable metrics for the neural network model so in order to compare the performances of the CNN and RNN we built against the performances of the benchmark ones, we generated the prediction for the test dataset and we submitted them to kaggle.

Hereafter we can see the results:

9 submissions for **fabiosteffenino** Sort by Private Score

All Successful Selected

Submission and Description	Private Score	Public Score	Use for Final Score
submission_LR.csv 18 days ago by fabiosteffenino logistic regression	0.97379	0.97424	☐
submission_LSTM_glove.csv 12 days ago by fabiosteffenino LSTM with glove weights 50 dimensions	0.97204	0.97062	☐
submission_CONV1D.csv 13 days ago by fabiosteffenino Baseline Convolutional neural network Conv1D	0.96982	0.97080	☐
submission_LSTM.csv 12 days ago by fabiosteffenino Basic LSTM no pre train embeddings	0.96797	0.96506	☐
submission_NB.csv 18 days ago by fabiosteffenino MultinomialNB	0.84484	0.85244	☐
submissionNB2.csv 18 days ago by fabiosteffenino naive bayes 2	0.84475	0.85238	☐
submission_RF.csv 18 days ago by fabiosteffenino random forest	0.82409	0.82695	☐

We can see from the score that even if our model were quite simple, we obtained a quite good result going above 0.97.

MultinomialNB and RandomForest, as we have already seen in the benchmarking session, they have the worse performances with a score of 0.844 and 0.824 respectively. Logistic Regression confirmed himself as the supervised learning model to be used with a score of 0.973.

Our Neural Networks model are not performing bad. The basic Conv1D and LSTM models have more or less the same score around 0.969. By using the pre-trained GloVe embedding with LSTM we can see we found that extra edge we were looking for by reaching more or less the same score of our best benchmark model (0.972).

V. Conclusion

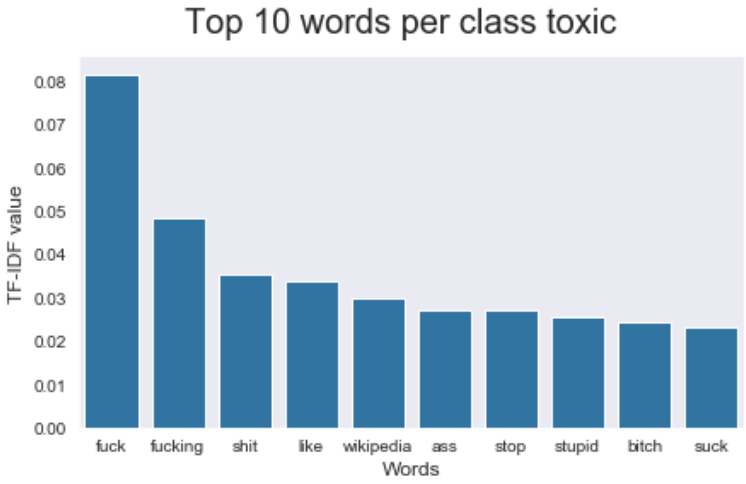
Free-Form Visualization (word analysis)

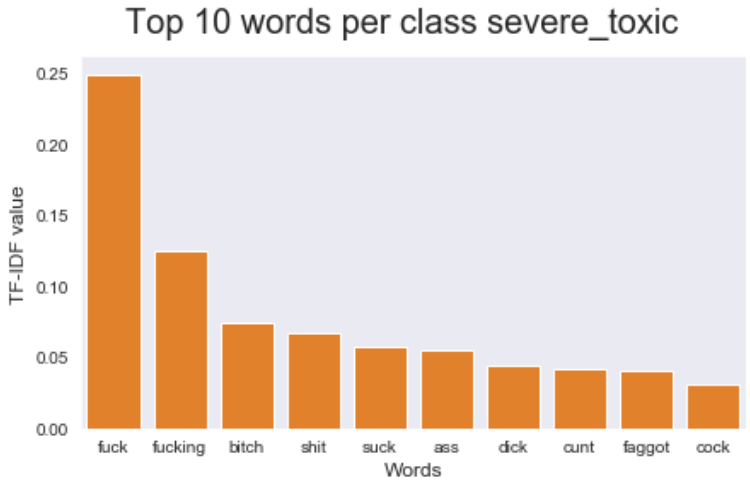
Disclaimer: the dataset for this competition contains text that may be considered profane, vulgar, or offensive

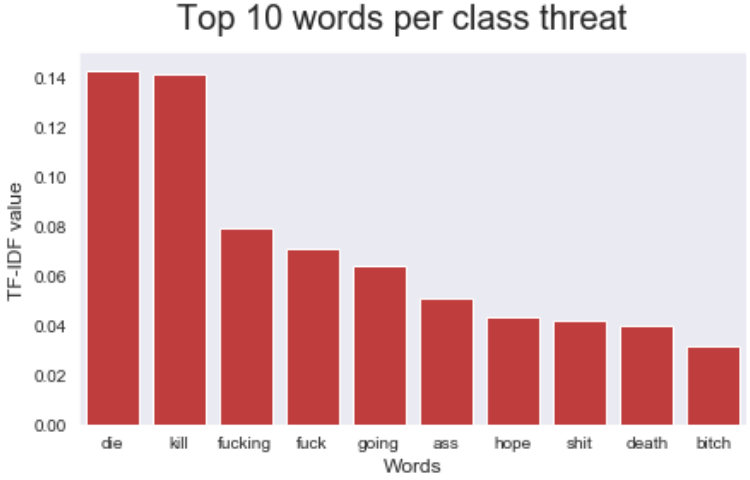
A quite interesting visualization we performed during the data analysis and that we didn't descuss previously is regarding the top words per Toxic category. In order to achieve this we used the following technique.

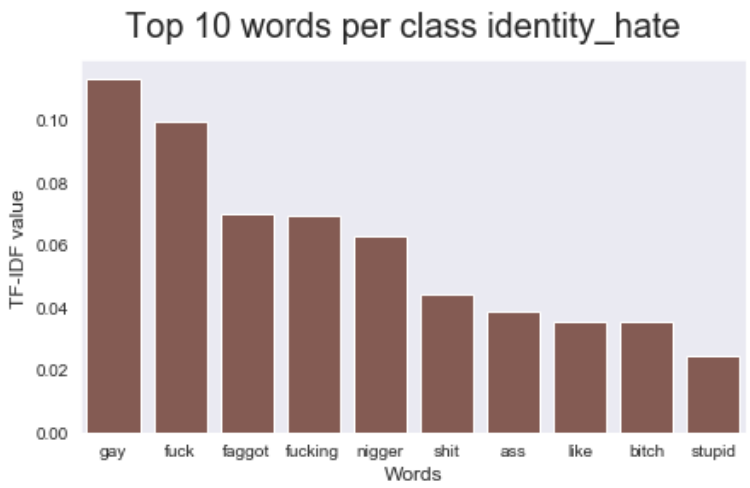
TF-IDF unigram. We used TD-IDF vectorizer as a vectorization techniques for the corpus text to be given in input to out benchmark models (for the neural network models we preferred a different method). TF means Term Frequency and is related to the phase of counting the words in the text corpus. IDF means Inverse Document Frequency and is related to the phase of penalization of words that are too frequent. TF-IDF combined provide word frequency scores that try to highlight words that are more interesting. When encoding the text corpus using TF-IDF, it is really easy to extract the top 10 words.

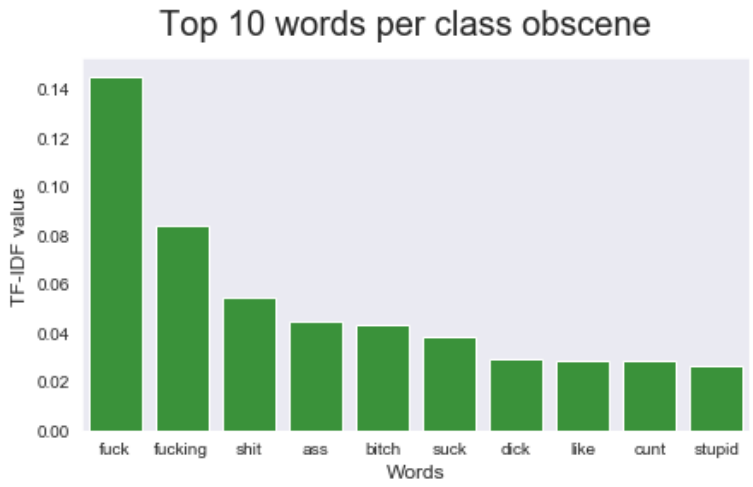
Hereafter we can see for each class, the plot of the top 10 words with the relative tf-idf score and the relative word cloud.

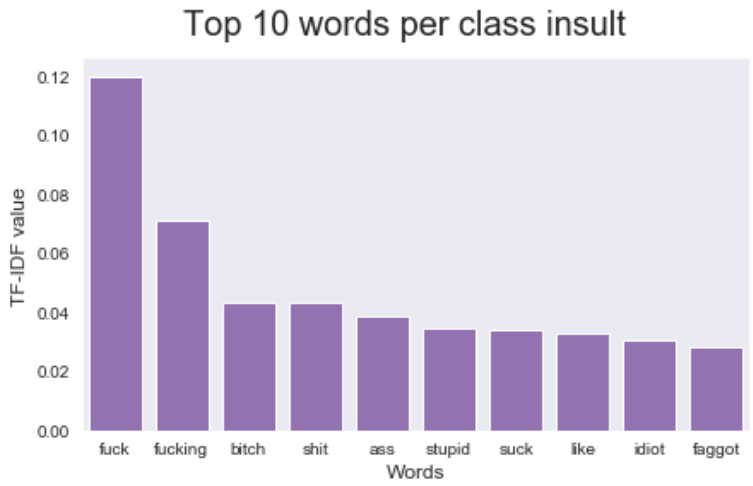



Top 10 words per class severe_toxic
TF-IDF value
0.25
0.20
0.15
0.10
0.05
0.00
fuck
fucking
bitch
shit
suck
ass
dick
cunt
faggot
cock
Words


Top 10 words per class threat
TF-IDF value
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
die
kill
fucking
fuck
going
ass
hope
shit
death
bitch
Words


Top 10 words per class identity_hate
TF-IDF value
0.10
0.08
0.06
0.04
0.02
0.00
gay
fuck
faggot
fucking
nigger
shit
ass
like
bitch
stupid
Words



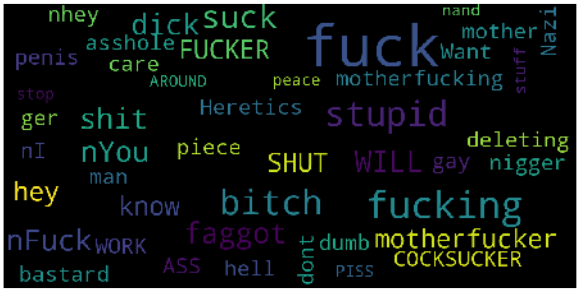


It is really interesting to see that just by looking at the top words we can easily deduct the category. Die and Kill are top words for Threat category. Nigga and Faggot are in the top for Identity_Hate and Stupid and Idiot are in the top for Insult category. Words like Fuck, Bitch and Shit are appearing multiple times and this is linked to the multi-tagging.

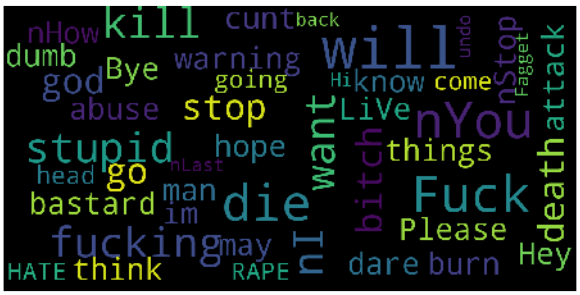
Word Clouds is another fancy way to visualize frequent words in a corpus. The higher the frequency of a word the higher the size of the words in the cloud. It provides a rough idea about Top words in a corpus.

toxic

obscene



severe_toxic



threat

insult

identity_hate

Reflection

This has been my first attemp to NLP problems. Even if I implemented not too complex supervised learning and neural network models, I already obtained some quite good results. I successfully deployed GloVe embeddings and recurrent neural network in order to build a toxic comment classification model and achieved a good accuracy with low cost. Our baseline model using LogisticRegression is still performing slightly better but I had fun trying to understand how convolutional/recurrent neural networks and word embedding works.

GloVe gave me that extra edge i was looking for when trying to improve my baseline LSTM model, but i am sure that with some other basic twiks we can easily outperform the LogisticRegression model (we will discuss the possible improvements in the next session).

I can consider myself quite lucky because kaggle provided a really good dataset for this problem. It was complete and it required a very little pre-processing work. This allowed me to take some time to discover different way of vector representation for the text corpus, from the bag of words method to tf-idf vectorizer and tokenization and padding.

The most difficult part of the project has been to understand how to proceed when tackling NLP problems like this one. As NLP was not part of the course, I had to go throuth some external resources and courses (I completed the TensorFlow Specialization by Andrew NG) . Also, lucky me, the research in this field is really advanced and we can find all different kind of documetation on the net.

The developed models matches my expectations. I think they provide, for a beginner in this field, a good idea about all the different techniques that need to used in those kind of classification problems and they could be easily used, maybe with some improvements, in a generic settings.

Improvement

As we said before, we obtained good results but with basic models and there is still space for improvements.

One of the first things to try is a different embedding method like Word2Vec of FastText instead of GloVe.

We could also try to use a bigger embedding (we used a 50 dimentions one in our model) with 100 or 200 dimentions.

Also in our models we used only one LSTM layer and we could possibly think about a more efcient pipeline with more than one LSTM or Convolutional hidden layer.

And finally we could explore some state of the art architecture like BERT or ELMo.

I am still scratching the surface of NLP related problems but thanks to this project I decided that i want to keep working on this field and maybe continue my study by doing the NLP nanodegree as well.