Capstone Project Proposal. Machine Learning Engineering Nanodegree

Text Classification: Toxic Comment Analysis

Stefano Paccagnella

6 October 2021

Domain Background

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 recognise 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.

Problem 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 platform.

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 analysed. 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 depends 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, it is quite appealing.

Datasets and Inputs

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. The level of toxicity defined/labelled is "toxic", "severe toxic", "obscene", "threat", "insult" and "identity hate".

The file provided are:

- ***train.csv*** *- the training set, containing comments with their binary labels*
- ***test.csv*** *- the test set, on which prediction on toxicity probabilities must be determined. To deter hand labelling, the test set contains some comments which are not included in scoring.*

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

All data are under CC0 license, with the underlying comment text being governed by Wikipedia's CC-SA-3.0.

Solution Statement

The final goal of this project is to develop a model that can classify with accuracy the level of toxicity of the text provided in the Kaggle competition. It is also scope of this project to compare performance of the different models that will be used to achieve this final aim.

At this stage there is not a clear picture on which model will be better performing on the provided dataset. The logic will be to apply several ML models, by increasing their complexity, and mainly to apply the "Bag of Words" model, "Word2Vec Embeddings", "FastText Embeddings", "Convolutional Neural Networks (CNN)" and possibly other two NLP models like "Long Short-Term Memory (LSTM)" and "Bidirectional Encoder Representation from Transformers (BERT)".

Benchmark Model

The benchmark model that will be used is the simpler model at the base of Text Classification, i.e. "Bag of Words". In this model, a text is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity.

We will use a sklearn implementation of "Bag of Words", *CounterVectorizer*, to convert the provided text into a numerical matrix that can be then mapped on the pre-defined set of label provided (i.e. "toxic", "severe toxic", "obscene", "threat", "insult" and "identity hate"). We will then use a Naive Bayes and Logistic Regression algorithms on data created by the CountVectorizer and will keep as benchmark the

one which is better performing. The prediction will be then realizes by using a Multi-Output Classifier from sklearn to predict all 6 categories.

Evaluation Metrics

To evaluate the performance of our models, we will use the AUC-ROC Curve. This is one of the ideal evaluation metrics when it is matter of multi-class classification problem. AUC stands for "Area under the Curve", while ROC stands for "Receiver Operating Characteristics".

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).

Project Design

The main steps in the project implementation are listed below and are depending on the model used:

- **Data pre-processing**: pre-processing data is one of the vital parts for any NLP project. This is because the type of data used in NLP, unstructured, contains a lot of noise. It will than be needed to reduce this noise, by converting all characters to lowercase, removing punctuation, removing stop words and typos.

Bag of Word model

After pre-processing the data, we will:

- Create a Bag of Words vector, by including the most -frequent words. The matrix should look like something similar to:

w1	w2	w3	w4	w5	w6	wn
0	0	0	0	0	1	0
0	1	0	0	0	0	0
1	0	0	0	0	1	0
0	0	1	0	0	0	0
0	0	0	1	0	0	0

- Create the Multi-Output Classifier: as we have to predict if a word is “toxic” or not “severe toxic” or not, etc. , we will need to classify the text against 6 output variables, which is a Multi Label Classification problem. The strategy will be to fit one classifier for target.

- Evaluation: last step will be to measure the performance of the models.

Word2Vec

After pre-processing the data, we will:

- Load and use a pre-trained embeddings vector (e.g. Google’s Word2vec). This will enable to “capture the semantic and syntactic meaning of a word as they are trained on large datasets” (see Aravind Pai, March 16,2020);
- Convert target text to embeddings by using the pre-trained vector;
- Train a Multi-Output Classifier ;
- Evaluate the model on the AUC-ROC metrics.

fastText

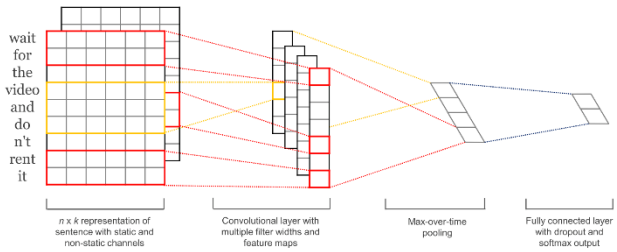
After pre-processing the data (in this case we will need also to use a particular label semantic __label 1__, __label 2__, etc.), we will:

- Train the model
- Evaluate the model

CNN:

After pre-processing the data (we will be using Keras libraries, i.e. initializing the tokenizer class, calling the fit on text function and using the calling the texts to sequence function), we will:

- Define a 1D CNN model: embedding layer as input, followed by a 1D convolutional neural network, pooling layer, and then a prediction output layer (see img. below):

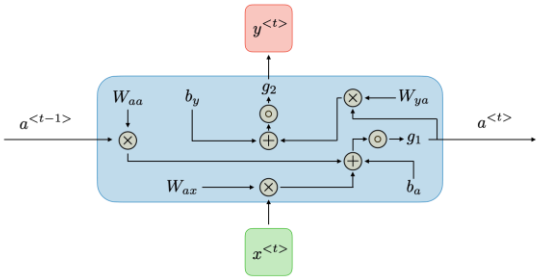


- Compile and fit the CNN model
- Evaluate the model

Long Short-Term Memory (LSTM):

After pre-processing the data (similar to what we will be doing in CNN), we will:

- Define a Multi-Label LSTM model: we will be using a dropout parameter.

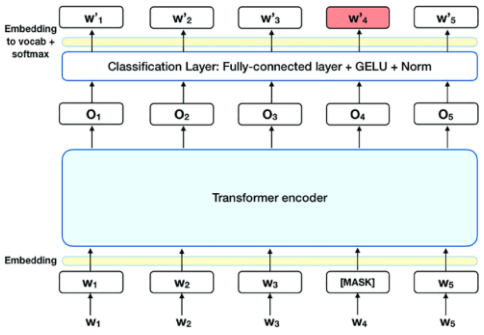


- Compile and Train the model
- Evaluate the model

BERT:

After pre-processing the data, we will:

- Create Train and Validate a Data loaders
- Load a pre-trained BERT and setting fine-tuning parameters
- Tune the BERT model



- Evaluate the model

At the end of the process we will than pick the model that is better performing and we will evaluate if a more complex model is always the best solution or if even a basic model can perform well.