Inferring the actitude toward self-driving cars from tweets using an NN-based supervised classification algorithm

Asier Barrasa, Iván Valdés

November 2018

Abstract

Sentiment analysis is a well-known problem in the machine learning community. A lot of companies are interested in the multiple context sentiment analysis can be used in. The use of Neuronal Networks for this task made a big improvement on the results. In our approach we will use all the knowledge we learned about this to infer an attitude toward different tweets about self-driving cars.

1 Description of the problem

- 1 dataset containing tweets about self driving-cars.
- 5 classes representing the attitude toward the self-driving cars. labeled from 1 to 5 they represents very positive, slightly positive, neutral, slightly negative, or very negative respectively.
- The data is unbalanced: Class 1 : 110, Class 2 : 685, Class 3 , 4227, Class 4 : 441, Class 5 : 458 [1]

2 Description of our approach

2.1 Research

The main problem at the start of the project was to find a good representation for our data.We cannot feed a NN with the tweets directly, so we had to find a way to transform words into a valid input for the net. We stared to search information about this and we found a lot of papers that come up with different approaches. In fact, we realized that there are several methods that are, in some way, standardized witch are already implemented.

[1] The process of counting is shown in the notebook

2.2 Preprocessing

1. Clean non-ASCII characters
2. Clean stop words
3. Clean too short words
4. Extract the text and the sentiment from cleaned csv

The libraries used are "gensim" and "nltk" We implement a function to "clean" the tweets. First, we eliminate all the non-ASCII characters. Then using the tools provided by "nltk" and "genism, we removed irrelevant characters, english "stopwords" ("I","a","for") and the words with less than 3 characters.We also remove some empty rows of the data-.set At this point the tweets are ready to extract features from them.

2.2.1 Feature extraction

For feature extraction we have used three different techniques, in order to compare the results and decide with one gives better results.

1. Count Vectorizer
2. TF-IDF
3. Doc2Vec

2.2.2 Count Vectorizer

Probably the most common technique. Count vectorizer count the appearance of the words in each text. The vector length would be the number of different words on the document

2.2.3 TF-IDF

TFIDF is another way to convert textual data to numeric form, it gives better results than Count Vectorizer.

$$TF(t,d) = \frac{number\ of\ times\ term(t)\ appears\ in\ document(d)}{total\ number\ of\ terms\ in\ document(d)}$$

$$IDF(t,D) = \log(\frac{total\ number\ of\ documents(D)}{number\ of\ documents\ with\ the\ term(t)\ in\ it}\ )$$

$$TFID(t,d,D) = TF(t,d) \times IDF(t,D)$$

2.2.4 doc2vec

Doc2Vec works on a similar way that word2Vec[2]. does. It takes as input a vector of words, in our case a tweet, and its returns a fixed length vector which contains the features of the vector.

While the word vectors represent the concept of a word, the document vector intends to represent the concept of a document.

We have used Distributed Bag Of Words (DBOW), in which the paragraph vectors are obtained by training a neural network on the task of predicting a probability distribution of words in a paragraph given a randomly-sampled word from the paragraph,this model is analogous to Skip-gram model in Word2Vec,combined with DMC (Distributed Memory Concatenated), in which the paragraph vectors are obtained by training a neural network on the task of inferring a center word based on context words and a context paragraph.

Unlike DBOW model, DM model can learn meaningful semantic vector representation of each word as well.

2.3 Classifier

According to the characteristics of our problem, we decided using a multi-layer perceptron to classify the data. In fact, we have implement two multilayer perceptron. One to satisfy the input size of the doc2vec generated vectors, and another one for Count Vectorizer and TF-IDF.

3 Implementation

3.1 Data Handling

To make easier the handling of the data-set in the project, the first thing we made was to clean the data-set. From the first one, we made a second data-set called *"cleaned-data-set.csv"* that contains the tweets with the preprocessing we mentioned before and the sentiment for the tweet.

3.2 Feature Extraction and Selection

After all the prepossessing, we are ready to extract features from the tokenized words of the tweets. At this point we have 6129 tweets. To extract the features we use all of this examples to make our vocabulary[2] and put those words in context. Then, after making some tests, we decided to choose 5000 of then for test and 1629 to test it.

[2]https://medium.com/scaleabout/a-gentle-introduction-to-doc2vec-db3e8c0cce5e

As we mentioned before, we are using three different ways of extract features. Doc2Vec generates vectors of 800 and the other of 8129.After computing this vectors the inputs are ready to train and test the NN.

4 Classification

NN	Input Layer Size	Number of Hidden Layers	Output Layer Size	Activation Function
Multilayer Perceptron for Dosc2Vec	800	2	5	Softmax
Multilayer Perceptron for CountVectorizer and _TF-IDF Vectorizer	8129	2	5	Softmax

Figure 1: Description of the used NN's

For the classification of the tweets we implemented 2 multi-layer perceptrons with 2 hidden layers and 5 outputs.

The input of the first layers depends on the technique, with Count vectorized and TD-IDF vectorized the number of the input layer was 8129, one per different word on the corpus. Instead, and after several tries, we have chosen 800 for doc2vec. We have used 2 hidden layer, each of 250 neurons.

The input of the first layers depends on the technique, with Count vectorized and TD-IDF vectorized the number of the input layer was 8129, one per different word on the corpus. Instead, and after several tries, we have chosen 800 for doc2vec. We have used 2 hidden layer, each of 250 neurons. Firstly, we tried with DBOW, but after reading Quoc Le and Tomas Mikolovs paper [1], we decided to implement DMC (Distributed Memory Concatenation) in order to get better results. The results were not as good as we expected, the networks, improve but not significantly.

We tried with different activation function like sigmoid on RELU but, we obtained the best results using linear activation in the hidden layers and then using softmax in the output layer.

5 Results

According to the different these are the different accuracies we get using the diferent models

Model	Accuracy
Doc2vec	59.24%
TD-IDF vectorized	
Unigram	60,35%
Bigram	59,41%
Trigram	58,78%
CountVectorizer	
Unigram	60,65%
Bigram	61,41%
Trigram	60,69%

Figure 2: Results obtained using Logistic Regression

NN	Accuracy
Doc2vec	56.55%
TD-IDF Vectorized	56.63%
CountVectorizer	56.53%

Figure 3: Results obtained using the multi-layer perceptrons

6 Conclusion

Throughout the project, we have taken into account how complex it is to extract features from text, and how word2vec and doc2vec works, and how they transform words and they meanings in numbers.
Although we have not achieved good results, we have learned a lot about NPL and how a computer understand the meaning of the words.

References

[1] Quoc Le and Tomas Mikolov [*Distributed Representations of Sentences and Documents*]. http://proceedings.mlr.press/v32/le14.pdf

[2] Jey Han Lau and Timothy Baldwin [*An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation*]. https://arxiv.org/pdf/1607.05368.pdf