Report

September 18, 2018

1 Introduction

On Internet, often, we can find posts cointaining hate speech. The aim of the social media sites is to identify these posts and censor them, but without comprimise the right to freedom of speech and expression.
The goal of this analysis is to detect hate speech in tweets, predicting label based on the words used. The tweet is consider hate speech when it contains a racist or sexist sentiment. The training dataset has a feature named *label*; the label '1' indicates the tweet is racist/sexist and label '0' denotes the tweet is not racist/sexist.

The data, that I have, are split in training and testing data:

- 'train.csv'contains 31962 tweets (rows) and 3 feature variables (tweet id, its label and the tweet);
- 'test.csv' contains only tweet ids and the tweet text.

2 Analysis

2.1 Basic pre-processing

My first step is to import relevant libraries and them load the data files into a DataFrame (*train_tweets* and *test_tweets*) and inspect them using **.head()** method. After that, I need to clean the text data in order to obtain better features and, also, to apply machine learning algorithms to it. I achieve this by doing some basic pre-processing steps on our data. For this reason, I merge train and test dataset (*tot*) to can preprocess the tweet text together.

First, I remove the pattern *@user* (in this dataset, due to the privacy, the twitter handles are masked as '@user'), because it doesn't add any information, and, if I don't remove it, it invalidates the result. Second, I remove special characters, numbers, punctuations and all the words with length 3 or less. To finish, I remove stop words (the most common words).

Then, I tokenize all the cleaned tweets, so I divide the text into a sequence of words (tokens), and I lemmatize them. Lemmatization is the process of converting the word into its root word: it uses of a vocabulary and morphological analysis of words and returns the base or dictionary form of a word, which is known as the lemma.

2.2 The most words used

After cleaning the text data, I start to study the most frequent words for racist/sexist tweets and non racist/sexist tweets, also using the **TF-IDF** (Term Frequency and Inverse Document Frequency).

The Term Frequency (tf) measures how frequently a word occurs in a document; however, there are words in a document that occur many times but may not be important. An approach is to look at a term's inverse document frequency (idf), which decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents. This can be combined with term frequency to calculate a term's tf-idf, the frequency of a term adjusted for how rarely it is used. It is intended to measure how important a word is to a document in a collection (or corpus) of documents computing a weight. The importance increases proportionally to the number of times a word appears in the document.

A TfidfVectorizer can be accessed through the attribute idf_, which will return an array of length equal to the feature dimension. I sort the features by this weighting to get the top weighted features and I determine the 20 words with highest TF-IDF scores for each label.

2.3 Machine Learning model

At this point, I divide the dataset in *train* and *test_set*. Using the *train* dataset, I run different machine learning algorithms to predict label based on the words used in the tweet. First, I split the dataset into a train and validation sets. The train set is used to build the predictive model, while the validation set is used to evaluate the model (having the label I can really check the result). The train set contains 90% of the total data, while the validation data contains 10% of the total data. After that, I run different algorithms to choose the best one to do label prediction calculating the accuracy (the F1-score) using cross-validation.

Cross-validation is a vital step in evaluating a model. It maximizes the amount of data that is used to train the model, as during the course of training, the model is not only trained, but also tested on all of the available data. I practice 5-fold cross validation on the data: the dataset is divided into 5 equal partitions and it uses the fold 1 as the testing set and the union of the other folds as the training set; then the accuracy is calculated. This process is repeated 5 times, therefore the function returns 5 scores. For this reason I take the average of these 5 scores.

I run KNeighbors classifier, random forest classifier, logistic regression, linear support vector machine and multinomial naive bayes. LinearSVC have a slight advantage with a weigthed F1 score of around 96% and therefore, it's the best algorithm to predict the label.

Finding the best algorithm, now, I can apply it to the split dataset, so I can check the result. I also print the confusion matrix (a table that describes the performance of a classification model) to see the number of true positive, false positive, false negative and true negative. True positive and true negatives are the observations that are correctly predicted, while false positives and false negatives occur when your actual class contradicts with the predicted class. Checking 20 prediction, I can see that there are two tweets over 20 where the predict label results 0 but the true label is 1: they are False Negative, but we can know that the accuracy of the model is around 96%.

Finally, I predict the label for my test dataset and I save the result in *test_set*.

2.4 Deep Learning model

I decide to do a model using Keras to run a neural network model. Neural networks are collections of nodes that apply transformations to data. From training data the model generates an output, compares that output to the actual result and adjuts the weights on its nodes.

First, I organize my data; also here I divided into train and validation dataset. Each token is converted into a matrix and create a dictionary of all the words we keep track of; I decide to use the top 3000 most-commonly occuring words in the training corpus. The label is trasformed to categorical, to have two columns: outcome 0 and outcome 1.

After that, I make the model: I use Keras' Sequential(), a simple type of neural network, and I add 2 hidden layers and an output layer. I also define the Embedding layer, that is a simple matrix multiplication that trasforms words into their corresponding word embeddings, with *output_dim* set to 100 (I choose the 100-dimensional version), *input_dim* set to 3000 and the *input_length* as maxlen set to 100. A word embedding is a mathematical representation of a word and this is needed since I can't work with text as plain input. Then, I flatten the input.

Next, I use Dense() (the 'standard', linear neural net layer of inputs, weights, and outputs) to implement the operation: first, I want to come out of first layer 500 outputs, then 250 outputs and of the output layer 2 outputs (because it's a classification problem). The activation functions used when training the network in the first and second layers are 'relu' (Rectified Linear Activation) and in the last 'softmax'. I use also two Dropout layers to randomly remove data, which can help avoid overfitting. As the last step, the model is compiled: I need to specify the optimizer (I choose 'adam'), the loss function to use and the accuracy metric.

After that, I fit that model and see Keras do the optimization so my model continually gets better. The model evaluates data in group of *batch_size* and the *epochs* is how many times it does this batch-by-batch splitting. My model has an accuracy of around 98%. After, I predict the label using that model and I test the result with the validation set. The result seems good. To finish I run the model and predict the label using the test dataset.