Sentiment Analysis of Black Lives Matter movement tweets

Cota Davide Antonio Maria
Politecnico di Torino

1 Data exploration

The dataset contains 100000 tweets in english regarding the *Black Lives Matter* movement, divided in 80000 for the training set and 20000 for the test set. The training set is balanced, being composed of 49.91% of positive tweets (in favor of the movement) and 50.09% of negative ones.

- Positive: `"RT @jeremycorbyn: It is shameful the UK Government won't condemn Trump. Now is the time to speak up for justice and equality. #BlackLives"`
- Negative: `"RT @larryelder: How many unarmed blacks were killed by cops last year? 9. How many unarmed whites were killed by cops last year? 19. More"`

We can see that the text contains not only plain text but also usernames, hashtags, emojis and retweets.

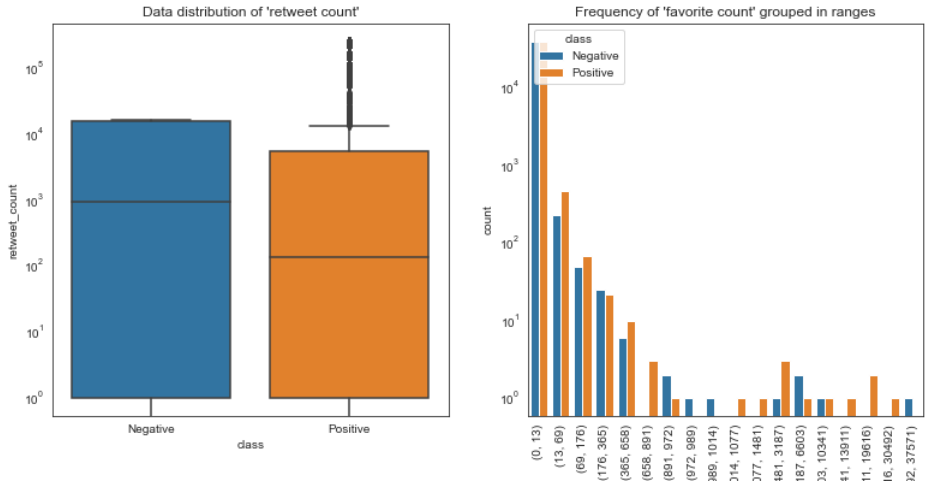

Figure 1: `'retweet count'` data distribution and `'favorite count'` frequencies

Fig.1 shows, on the left, the data distribution for the `'retweet count'` value associated at each tweet. We can appreciate the fact that the two classes have a different distribution and the medians differs almost of an order of magnitude. The plot on the right is related to `'favorite count'` and explains the frequency for each range compared to each class. The bins are obtained by using the *KMeans* algorithm with 18 centroids. The graph doesn't shows big differences in the two distributions so we can assume that this feature won't be so helpful in discriminating the two classes.
Instead, what can help our analysis is considering the `tweet length` for each class: negative comments are often associated to short tweets (up to 120 characters), while positive ones are longer (Fig.2). This can be due to the fact that positive tweets contains a more specific description of the problem.

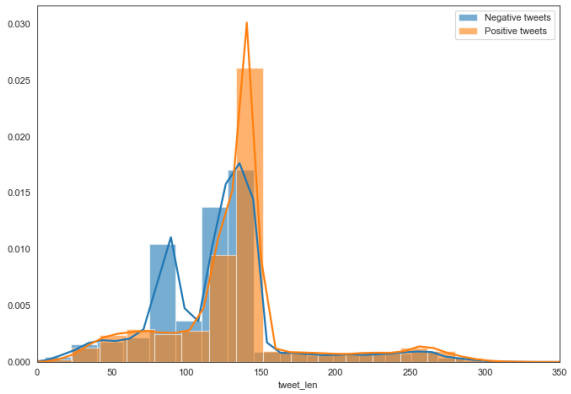

Figure 2: Length distribution

It could have been interesting taking into account for our analysis some geolocalization features like `'coordinates'` and `'places'`, but this hasn't been possible due to high number of `Null` values (99%).
A *Wordcloud* is a visual representation that shows in which way the most frequent words are grouped together: Fig.3 contains two wordclouds for the 100 most frequent terms in the dataset (excluding punctuation and stopwords). The first cluster can be associated to words related to negative tweets, such as racist, white, right, protest and the hashtag #alllivesmatter. K-pop fans worldwide drowned out hashtags associated with right-wing causes with K-pop videos, that's why this term is present in this particular cluster. The second cluster is associated to positive tweets terms such as unarmed, killed, black, shot and the popular black activist Larry Elder.

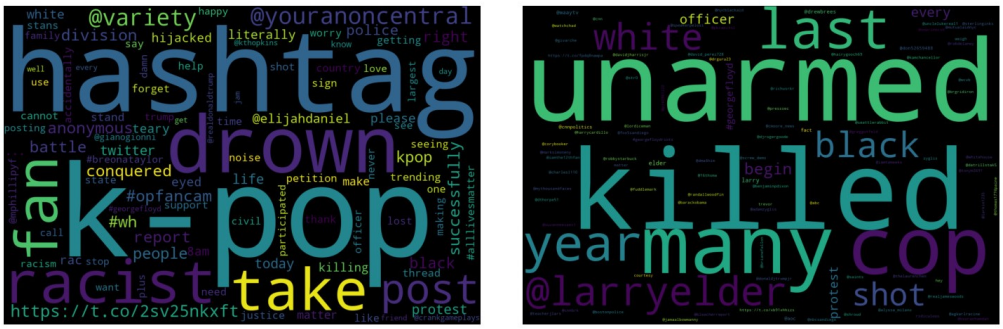

Figure 3: Wordclouds

2 Preprocessing

Not all the features are relevant in our analysis, some of them can be redundant. To check the correlation among the ordinal features a correlation matrix can be helpful. Fig.4 shows that all the ordinal features are independents from each other.

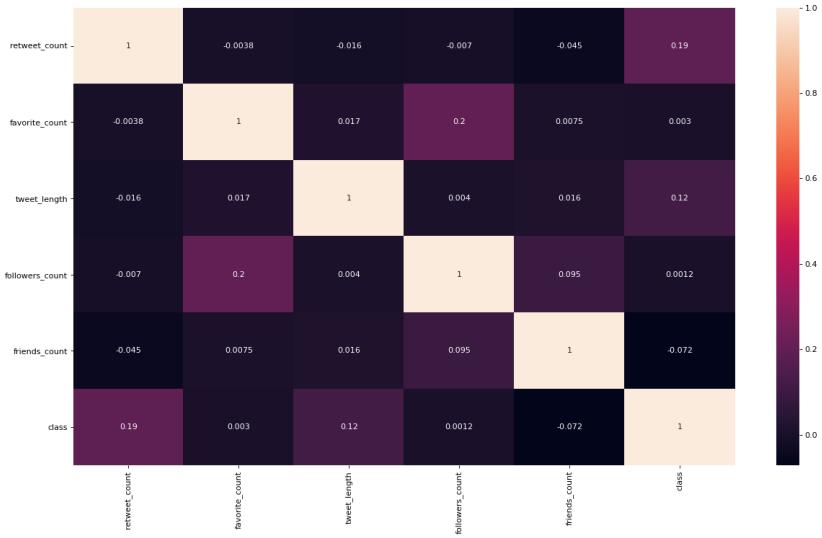

Figure 4: Correlation matrix

To check how much is strong the relationship between each feature and the class label we use the *Chi-Squared test*. It's clear in Fig.5 that the most significant feature is the `'retweet count'`, while all the others are poorly related with the class label. It is also telling us that we made a correct hypothesis when supposing that the `'favorite count'` feature wouldn't have brought relevant insights. We can proceed our analysis by considering only the `'retweet count'` (encoded in bins using a

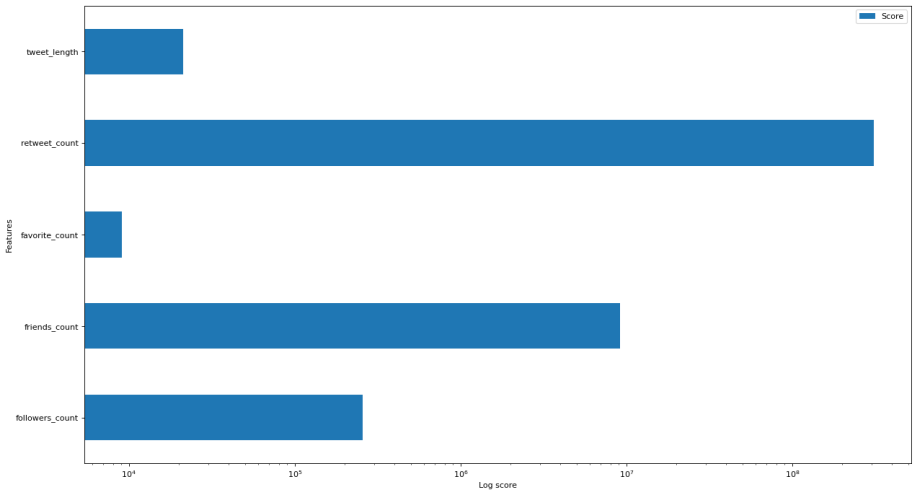

Figure 5: *Chi-Squared test*

KMeans strategy), the `'full text'` of the tweet and the `'user description'`.
In order to make understandable the text by our model we need to convert it in a readable format. We are proposing two ways:

- *TF-IDF vectorization*
- *Word embeddings*

The *'Term Frequency - Inverse Document Frequency'* is a way of text transforming in which frequent terms assumes a lower value respect to the ones less frequent. In order to consider the whole meaning of the sentence we do not limit to convert only one word at a time, but groups of close terms, called *n-grams*. Removing the *stopwords* with this kind of approach resulted always in lower performances, probably because they bring relevant informations when grouped in n-grams. All the terms that appeared at the least once in the whole training set has been considered. Each tweet has been converted to lowercase, only the punctuation has been removed and no others dimensionality reduction techniques has been used.
Word embeddings is a technique in which words are mapped to n-dimensional vector. The words that are similar will have similar vector. This mapping can be obtained dynamically by training an Embedding layer on the whole set of documents in exam or it can be used some kind of pre-trained word vectors, such as *GloVe* [4].
For both approaches the *TwitterTokenizer* has been used alongside the *WordNetLemmatizer*. In this way hashtags and usernames has been preserved.
An out-of-fashion methodology that gave us a boost in the performance was concatenating the `'user description'` with the `'full text'` of each tweet. This method permits us to maintain a single vocabulary *term - frequency*, instead having one for each type of text (`'full text'` and `'user description'`).

3 Algorithm choice

In order to choose the appropriate model for this kind of task some different classifiers has been evaluated, some of them already analyzed in a similar work from *Tripathy et al.* [6]: *Linear SVC* (*SVM* with linear kernel), *Random Forest*, *Multinomial Naive Bayes*, *KNeighbours classifier* and *SGD classifier*. Fig.6 shows accuracy scores for each classifier with default hyperparameters evaluated through *cross-validation* technique using 15 folds. It is clear that SGD classifier outperforms all the other ones in terms of accuracy and training time, that's why we are going to focus on this particular model. *Stochastic Gradient Descent (SGD)* [5] corresponds to an optimization technique,

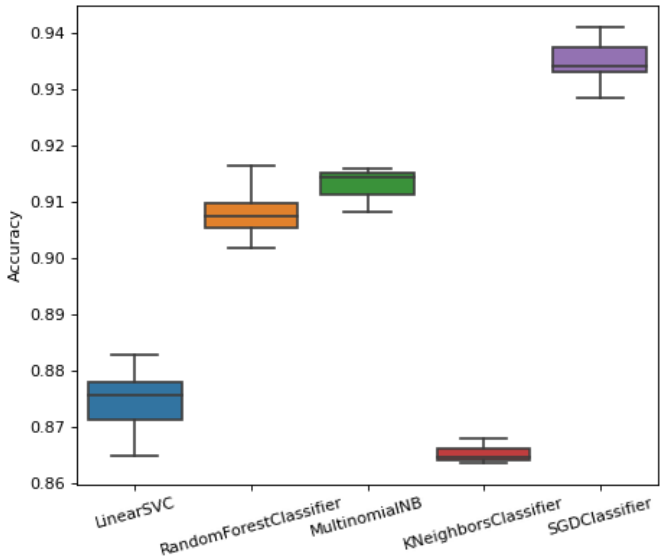

Figure 6: Classifier performances

i.e. an efficient way to train the model. The SGD classifier supports different loss functions that permits the model to fit different kind of linear classifiers (or regressors), such as *Support Vector Machines*. The input values for this estimator are the n-grams resulting from the *TF-IDF* transformation.

Our second approach consists in building a neural network for the sentiment analysis. We use an *LSTM (Long Short Term Memory Network)*, that is a variant of *Recurrent Neural Network* that best fits *NLP* problems. Our preprocessing is now based on *Word embeddings*. The model described in Table 1 has been set up by looking at a previous work from a Kaggle notebook [1] and slightly modified. The loss used is the *binary crossentropy* and the optimizer is *Adam* [3].

After the first tries we can notice that this model suffers of a well known problem with *LSTMs*: *overfitting* in the early steps of training. Despite trying to decrease the *LSTM* units and increasing the *Dropout* rate, the problem of overfitting only relieves a bit.

To efficiently counteract this issue we build a neural network composed of `n` parallel branches of both *LSTM* and *Convolutional* layers. The model has first been proposed by *Yenter et al.* [7], and we take their best performing configuration described in Fig.7. The loss used is again the *binary*

Layer	Units	Rate	Activation
Embedding			
LSTM	128		
MaxPooling			
Dropout		0.5	
Dense	50		ReLu
Dropout		0.5	
Dense	1		sigmoid

Table 1: Neural network #1

crossentropy and the optimizer is the *RMSprop* [2]. Layers like *Dropout* and *Max Pooling* help preventing overfitting, while *Batch Normalization* leads to convergence at a faster rate. The *Convolutional* layers permits the model to view `c` words at a time (where `c` is the kernel size), mimicking the n-gram behaviour.

Figure 7: Neural network #2

4 Tuning and validation

For the *SGD classifier* a `3 folds grid search cross validation` has been used to fine tune the following hyperparameters:

- `loss`: 'hinge' (gives a linear SVM), 'squared hinge' and 'modified huber'
- `penalty`: 'l1' or 'l2'
- `alpha`: from 1^{-1} to 1^{-6}
- `tolerance`: from 1^{-2} to 1^{-5}
- `number of bins`[1]: from 2 to 6
- `n-gram range`: (1,1), (1,2), (1,3), (1,4), (1,5)

Table 2 shows the validation mean accuracy for the `top 5` *SGD classifiers* and their relative hyperparameters: The best performing classifier has been refitted on the whole training set after the cross

loss	penalty	alpha	tolerance	n. bins	n-gram range	accuracy
hinge	l2	1^{-5}	1^{-3}	4	(1,4)	**0.9322**
hinge	l2	1^{-5}	1^{-3}	4	(1,3)	0.9321
hinge	l2	1^{-5}	1^{-2}	2	(1,4)	0.9321
modified huber	l2	1^{-5}	1^{-3}	4	(1,4)	0.9313
hinge	l2	1^{-6}	1^{-3}	4	(1,3)	0.9307

Table 2: Top-5 *SGD classifiers*

validation, giving an accuracy on the test set of **0.9387**.

For what concerns our first neural network, Fig.8 shows the trend of accuracy and loss over 4 epochs. The clear indicator of overfitting is that, just after the first epoch, the accuracy on the training set maintains increasing while the accuracy on the validation set starts decreasing. Our second neural network, instead, successfully eliminates the overfitting problem as the validation loss stops decreasing only after 20 epochs. Both the neural networks have been trained using 20% of the dataset as validation set. The hyperparameters to choose for this model are the following:

- `Vector dimension`
- `Batch size`
- `LSTM units`
- `Dropout rate`

The network has been trained using the *Early Stopping criterion*: the train will stop when the validation loss doesn't decrease for 2 following epochs. No type of grid search has been implemented due to the high training time, but the model has been manually fine tuned. The gain in terms of accuracy exists but it isn't much higher respect to the previous one. No improvements are noticed when using *GloVe* pretrained weights on Twitter corpus. The best hyperparameters are showed in Table 3. The model has been retrained on the whole training set and gave a test accuracy of **0.9315**.

[1]Number of bins in the discretization step for the 'retweet count' feature

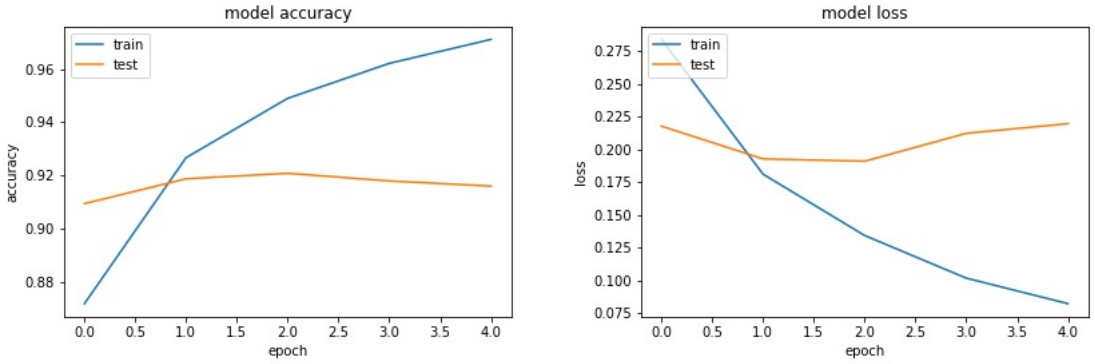

Figure 8: Accuracy and loss over 4 epochs

Vector dimension	Batch size	LSTM units	Dropout rate	accuracy
64	64	128	0.5	**0.9213**

Table 3: Neural network accuracy

5 Code

The code is available here.

References

[1] Bongo. Tackling toxic using keras.

[2] Geoffrey Hinton. Rmsprop: a mini-batch version of rprop.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[5] Scikit-learn. Stochastic gradient descent.

[6] Abinash Tripathy, Ankit Agrawal, and Santanu Rath. Classification of sentiment reviews using n-gram machine learning approach. *Expert Systems with Applications*, 57, 03 2016.

[7] Alec Yenter and Abhishek Verma. Deep cnn-lstm with combined kernels from multiple branches for imdb review sentiment analysis. pages 540–546, 10 2017.