Sentiment Analysis assignment - Carlos Azevedo

The challenge consists of a sentiment analysis problem where we want to identify tweets which are hate tweets and which are not (non-hate tweets).

The approach consists of the following steps:

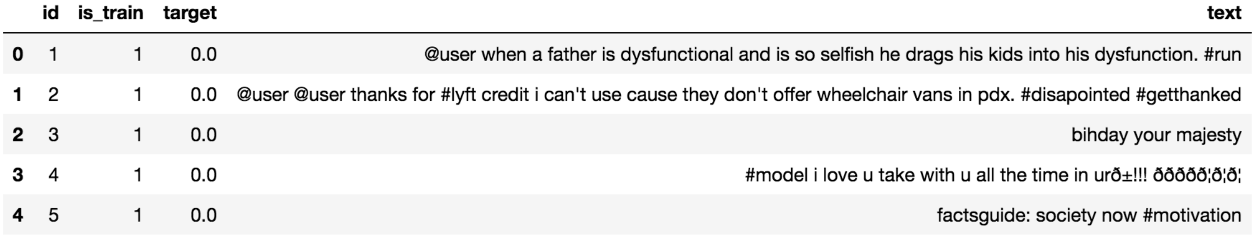
1. Data cleaning;
2. Data exploration;
3. Modeling with Logistic Regression and Random Forests;
4. Modeling with simple NN and NN with LSTM.

Data cleaning

The data is provided in two separate .csv files that were merged to avoid preprocessing twice:

- training set with 31962 samples (29530 after dropping duplicated tweets);
- Testing set with 17197 samples (15447 after dropping duplicated tweets).

Follows the first five rows of the dataset:



	id	is_train	target	text
0	1	1	0.0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run
1	2	1	0.0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked
2	3	1	0.0	bihday your majesty
3	4	1	0.0	#model i love u take with u all the time in urð±!!! ðððð¦ð¦ð¦
4	5	1	0.0	factsguide: society now #motivation

Looking at it we can observe some encoding problems and particularities about the twitter messages that need to be cleaned. If we look into it with more detail we can find the following irregularities that we will normalise:

1. @user - masked user names for anonymity. We'll consider it irrelevant for the problem at hands and thus we will remove it;

2. \#getthanked - hashtags contain relevant information (it can be seen here https://www.analyticsvidhya.com/blog/2018/07/hands-on-sentiment-analysis-dataset-python/) so we'll keep them and remove only the \# symbol;

3. Abbreviations - in a more in-depth study it should be normalised to one form, in this case we will do it for a short list of abbreviations;

4. Contractions - Forms as (can't) vs (can not) vs (cannot) - will be converted to a common format;

5. Stop words - words like 'the', 'to', etc do not add information to the approach that will be used and will end up contributing just as noise for the final model;

6. Stemming will be applied with the same purpose of finding a common form for a common meaning. This reduces the vocabulary by reducing words to its root / stem;

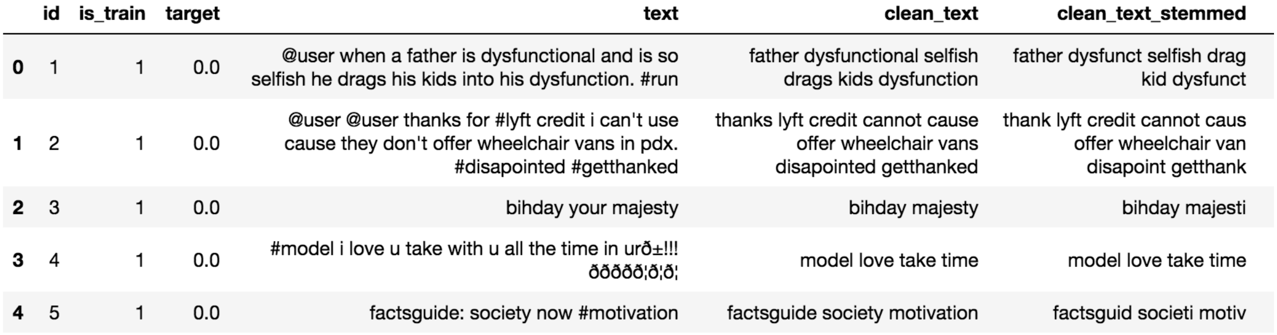
7. All the text will be transformed into lowercase;

8. Remove HTML encoding - Example: '&amp', …;

9. Remove URLs;

10. Correct words as 'sooo goooood' to 'soo good' - only allows to repeat the same character twice. We will still have 'so good' and 'soo good' but it reduces the number of possibilities.

After this preprocessing we get:

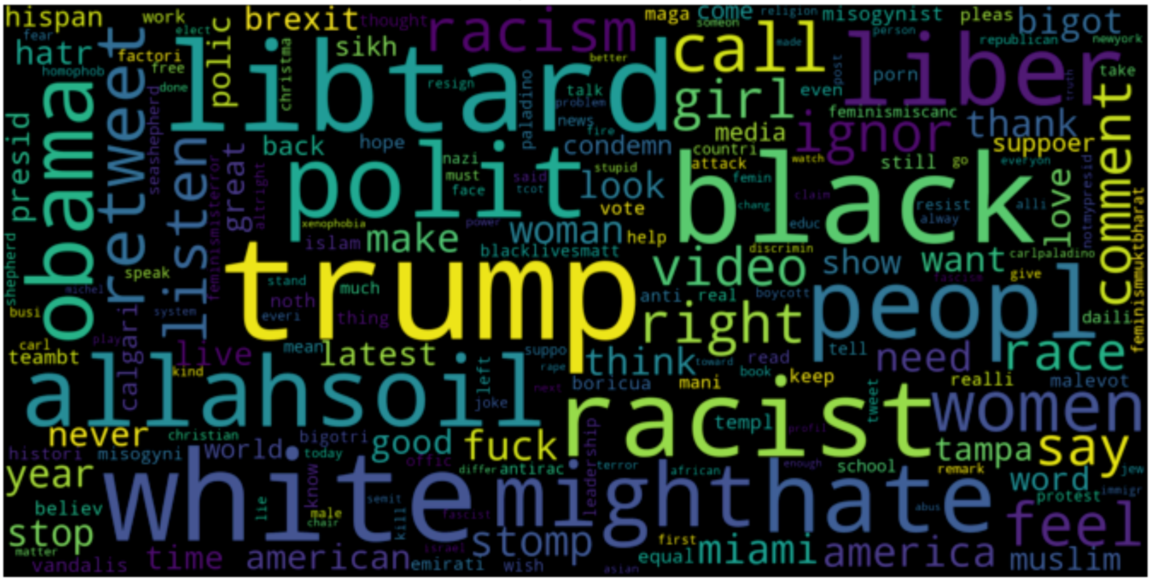


	id	is_train	target	text	clean_text	clean_text_stemmed
0	1	1	0.0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run	father dysfunctional selfish drags kids dysfunction	father dysfunct selfish drag kid dysfunct
1	2	1	0.0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked	thanks lyft credit cannot cause offer wheelchair vans disapointed getthanked	thank lyft credit cannot caus offer wheelchair van disapoint getthank
2	3	1	0.0	bihday your majesty	bihday majesty	bihday majesti
3	4	1	0.0	#model i love u take with u all the time in urð±!!! ðððð¦ð¦ð¦	model love take time	model love take time
4	5	1	0.0	factsguide: society now #motivation	factsguide society motivation	factsguid societi motiv

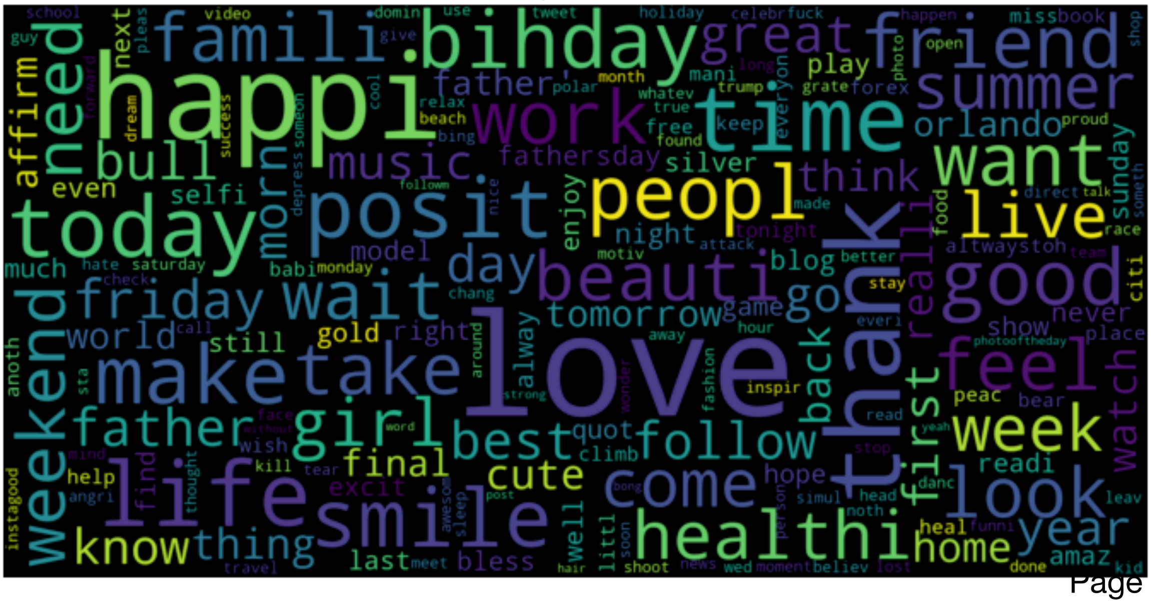
Although the *WordCloud* is not a scientific method it does help to visually identify which words are more prevalent.

By using this method I ended up finding some words that were contaminating the text (example: &amp);

Hate speech WordCloud



Non-hate speech WordCloud

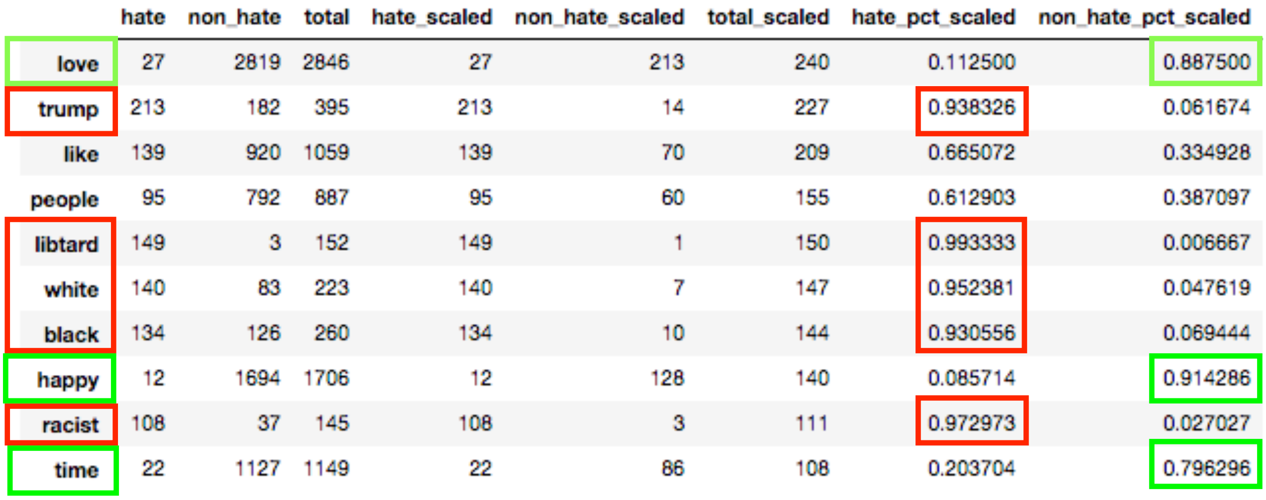


From the images above we can see that the ‘hate speech’ words that are drawn make sense - usually related with politics, races, gender, negative words, swearing, etc. In the ‘non-hate speech’ *WordCloud* we can find a lot of positive words. It is important to notice that we are considering only 1-gram here, which does not capture the context in which the word is inserted - ‘happy’ and ‘not happy’ will contribute equally for the word ‘happy’.

In both cases there's a lot of noisy words that could be excluded in a more detailed analysis.

Data exploration - comparison between classes

The training set is imbalanced having 2242 (0.07%) samples of hate speech and 29720 (0.93%) samples of non-hate speech. In order to compare the percentage that a certain word shows up in one class vs the other we will re-scale to be as if there are the same number of samples for each class. That will be shown by the columns 'hate_pct_scaled' and ‘non_hate_pct_scaled'.



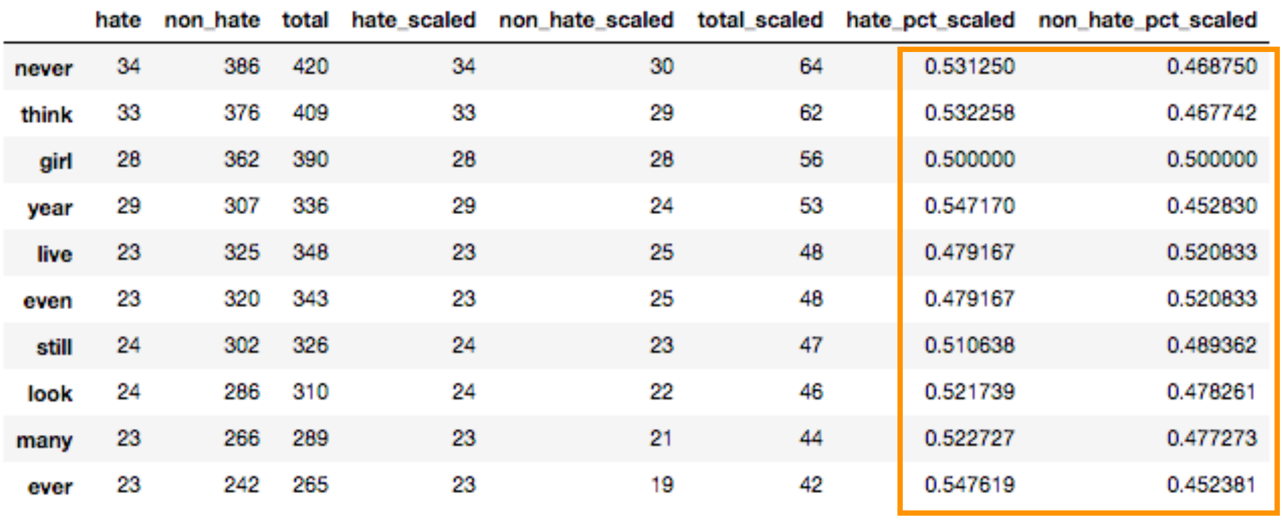
	hate	non_hate	total	hate_scaled	non_hate_scaled	total_scaled	hate_pct_scaled	non_hate_pct_scaled
love	27	2819	2846	27	213	240	0.112500	0.887500
trump	213	182	395	213	14	227	0.938326	0.061674
like	139	920	1059	139	70	209	0.665072	0.334928
people	95	792	887	95	60	155	0.612903	0.387097
libtard	149	3	152	149	1	150	0.993333	0.006667
white	140	83	223	140	7	147	0.952381	0.047619
black	134	126	260	134	10	144	0.930556	0.069444
happy	12	1694	1706	12	128	140	0.085714	0.914286
racist	108	37	145	108	3	111	0.972973	0.027027
time	22	1127	1149	22	86	108	0.203704	0.796296

The image above shows the ten words that show up most often (after scaling, based on total_scaled column). In red words that show up mostly in tweets that are classified as hate speech and in green non-hate speech.

From the table above, looking into the last two columns we can observe three scenarions:

- words that show up more often for neutral class (0): 'love', 'happy';
- words that show up more often for hate speech class (1): 'trump', 'libtard', 'white', 'black', etc...
- words that are more balanced regarding their frequencies in both class: 'like', 'people';

The last scenario will contain noisy words that will affect negatively the performance of our algorithms. We will now look into more of these examples.



	hate	non_hate	total	hate_scaled	non_hate_scaled	total_scaled	hate_pct_scaled	non_hate_pct_scaled
never	34	386	420	34	30	64	0.531250	0.468750
think	33	376	409	33	29	62	0.532258	0.467742
girl	28	362	390	28	28	56	0.500000	0.500000
year	29	307	336	29	24	53	0.547170	0.452830
live	23	325	348	23	25	48	0.479167	0.520833
even	23	320	343	23	25	48	0.479167	0.520833
still	24	302	326	24	23	47	0.510638	0.489362
look	24	286	310	24	22	46	0.521739	0.478261
many	23	266	289	23	21	44	0.522727	0.477273
ever	23	242	265	23	19	42	0.547619	0.452381

From this table we can see that words as 'look', 'many', 'still', 'even' do not carry meaning by themselves and it is worth trying to model after excluding them. In the current solution these words were not removed. The reason is that we will work with n-grams and they might still make sense in the context of more than one word. A more in-depth study is needed in order to understand the relevance of these words.

Data split

The initial training dataset (labeled data) was split into two datasets: training (90%) and test (10%). The former will then be split into training and validation in order to perform cross-validation and the latter will be used to evaluate results / simulate a real case scenario with data that the model has not seen yet.
The process of cross-validation in this case is being used only to determine the threshold that maximises f1 score. In a more in-depth study it should be used to tune the hyper parameters of the model.

Modeling with Logistic Regression and Random Forests

When modeling, the first step we want to take is to define the performance metrics that we will use. The problem states that it will use f1 score to evaluate the results, nevertheless we will also look into recall and precision to better understand where our model is misclassifying (hate vs non-hate speech). Accuracy will not be very informative in this context due to the data imbalance.

Now we will create some models (logistic regression and random forests) based on word of bags (bow) and term frequency–inverse document frequency (tf-idf) and explain its predictions by evaluating what features have more weight (both positive and negative).

Regarding feature engineering only two approaches were taken:
- Word of bags - which consists in the word count of each tweet;
- Tf-idf - which is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.

From the perspective of tf-idf, if a word shows up only in one tweet and it does not show up anywhere else it means that this word is extremely relevant for that tweet. This case is not what we want because it will also only be valid for that one tweet and we want to create a model that is able to generalise: this is why we will exclude words that show up too often as well as words that show up only in very few samples. This also applies to the word of bags approach.

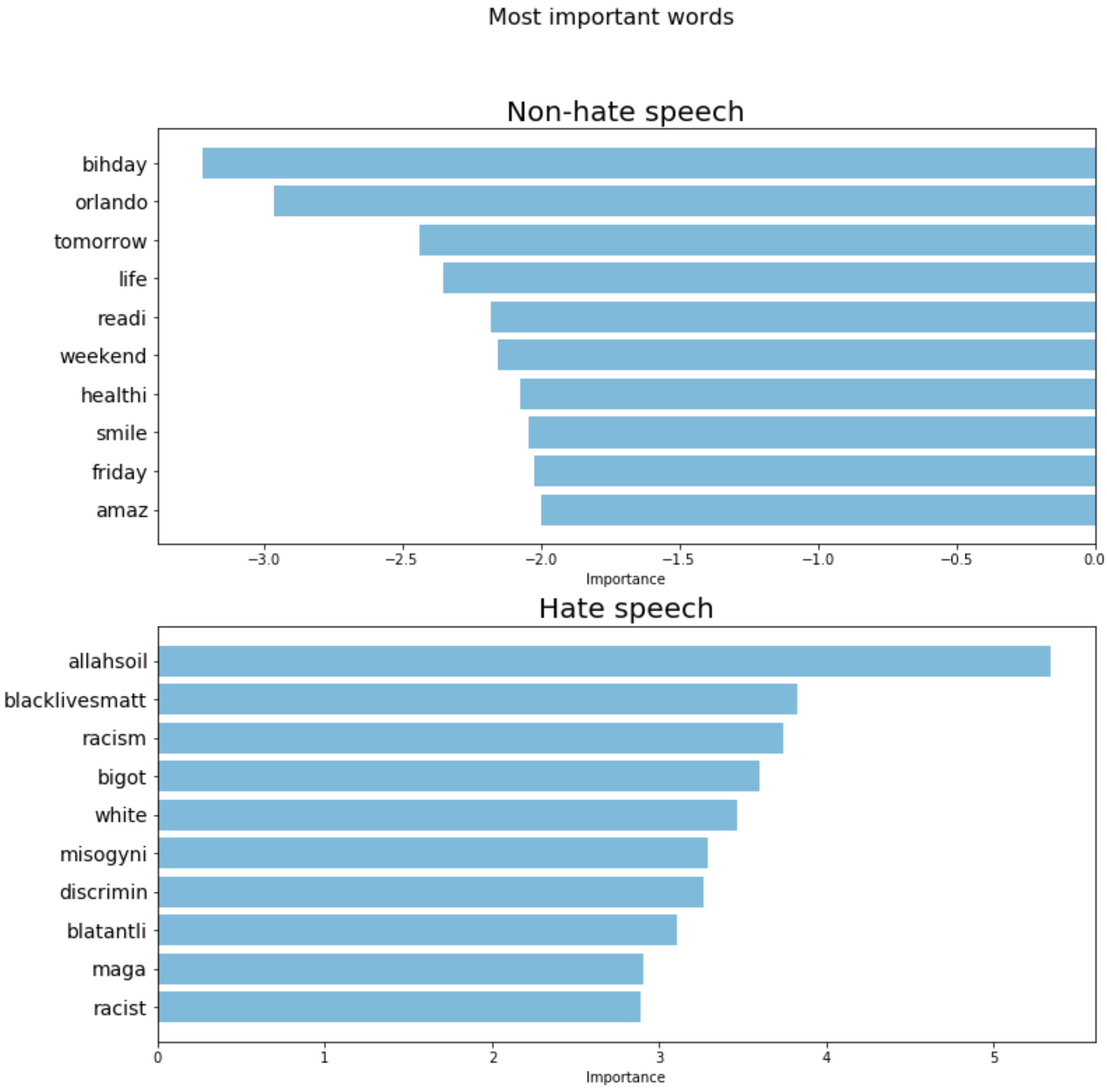
In both cases it was used 1-gram, 2-gram and 3-gram of the tweets.

Logistic Regression

For this model the only non-default parameter that was used was the 'class_weight' that was set to balanced. This allows the model to deal with the data imbalance and give higher weight to the 'hate speech' class samples (class fewer samples).

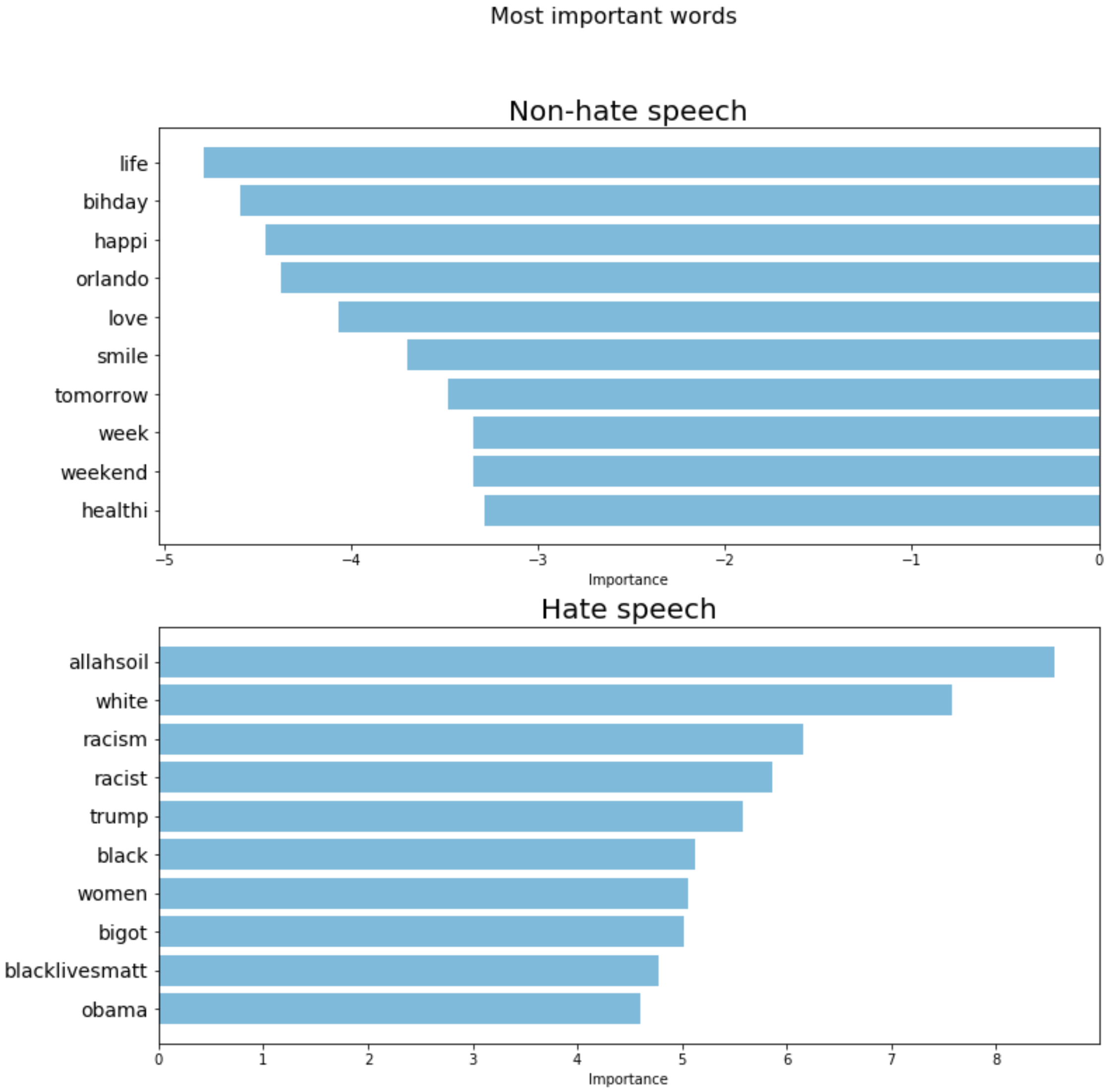
- Bag of words - The logistic regression with the bag of words features output the following results: accuracy = 0.956, precision = 0.726, recall = 0.603, f1 = 0.659

Below we can see the feature importance after fitting the model.



- Tf-idf - The logistic regression with the tf-idf features output the following results: accuracy = 0.952, precision = 0.686, recall = 0.585, f1 = 0.631

Below we can see the feature importance after fitting the model.



Looking at the results/metrics we have a scenario of low recall and high precision. This means that our classifier is conservative: the tweets it classifies as 'hate speech' are actually correct but it is missing a lot of them as well.

Random Forests

For this model the only non-default parameter that was used was the ‘class_weight’ that was set to balanced. This allows the model to deal with the data imbalance and give higher weight to the ‘hate speech’ class samples (class fewer samples).

- Bag of words The random forest with the bag of words features output the following results: accuracy = 0.952, precision = 0.698, recall = 0.558, f1 = 0.620

- Tf-idf - The random forest with the tf-idf features output the following results: accuracy = 0.931, precision = 0.503, recall = 0.647, f1 = 0.566

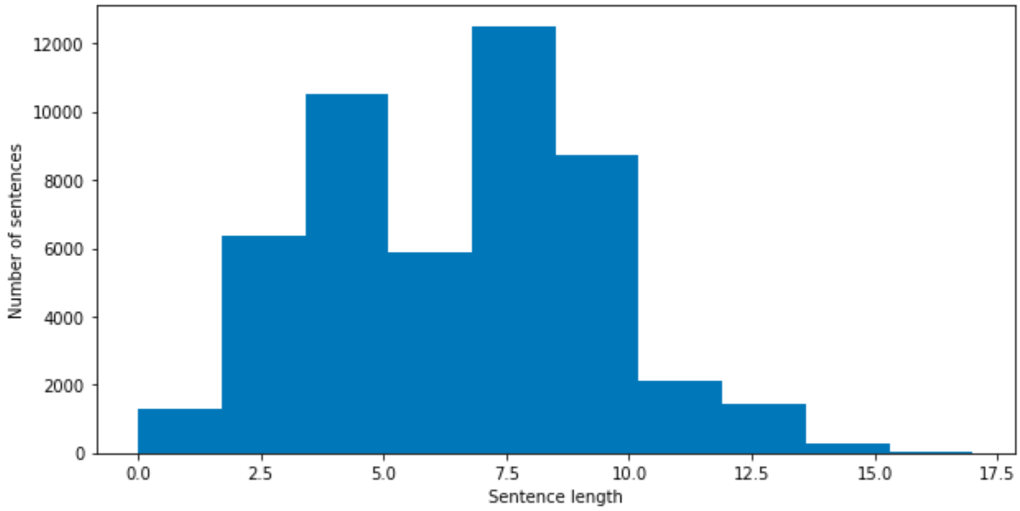
Looking at the last result we can say that we have a ‘high’ recall and low precision. This means that our classifier thinks there's a lot of tweets that are 'hate speech’: it did classify correctly a lot of ‘hate speech’ category but there’s a lot of positive classifications that should be ‘non-hate speech’ (non-conservative).

Modeling with naive NN and NN with LSTM

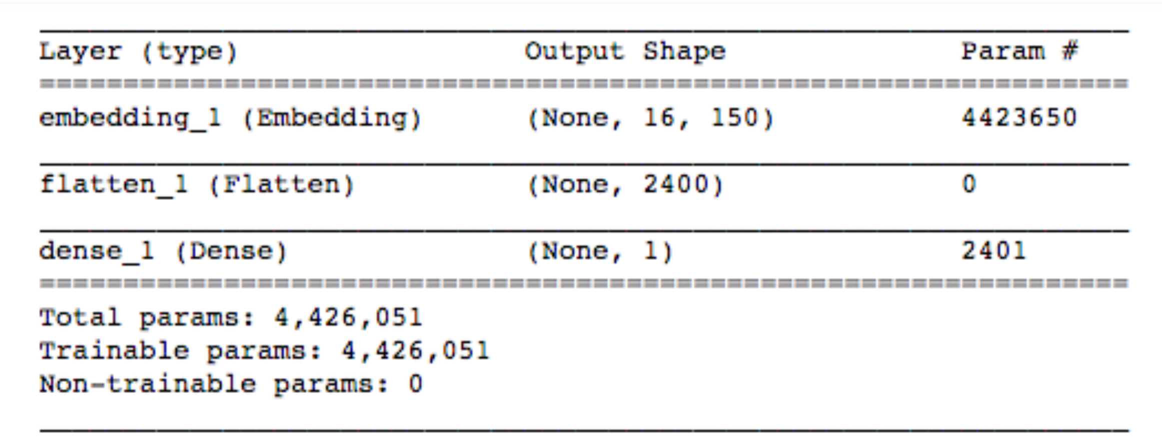
The NN approach that will be used here is simple: input layer that will receive the whole sentences/tweets. Each tweet is converted into a vector (where each entry is a word) that will be fed to the NN, then each of the entries of that vector (each word) is mapped to it's representation in the vector space - word embedding. After having all the words represented by a vector, we have a flatten layer that will then feed a sigmoid in order to have an output between 0 and 1.

It could have been more elaborated by using a CNN or RNN (that are used to take into account sequences) with pre-trained word embeddings that would map words into a vector space where closely related words in terms of meaning would be grouped together. This means that words like 'black' and 'white' would most probably be similar with other colours like 'yellow' or 'green' but it could also relate with the word 'racism', specially if we retrain the word embedding to our particular case / subject.

In order to use as input the whole sentence we need to determine the maximum size (number of words) of our tweets. Below is an histogram with word counts of each tweet in our dataset.



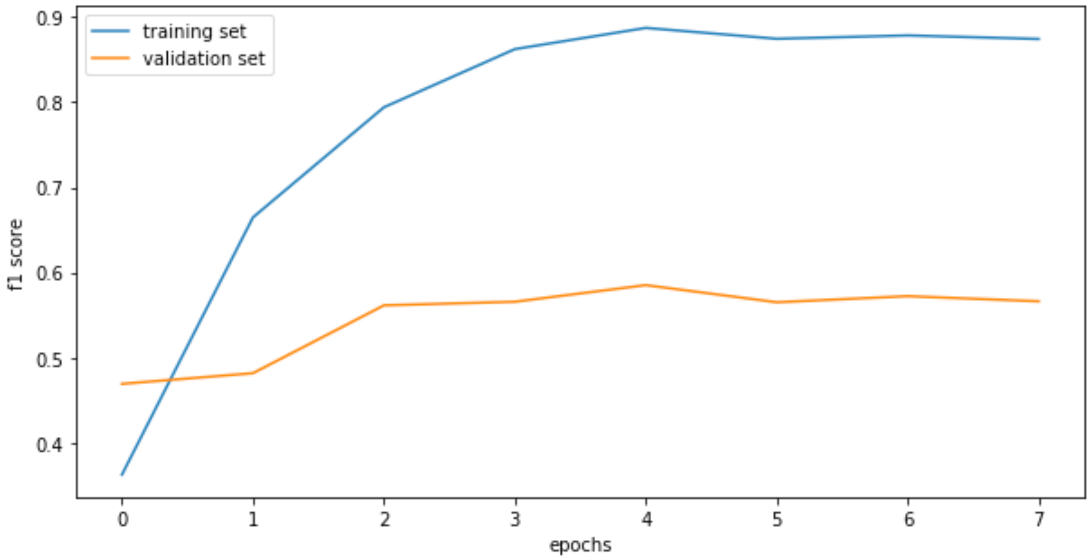
The architecture of our simple NN is the following:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 16, 150)           4423650   
_________________________________________________________________
flatten_1 (Flatten)          (None, 2400)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 2401      
=================================================================
Total params: 4,426,051
Trainable params: 4,426,051
Non-trainable params: 0
_________________________________________________________________
```

Which gave us the following results:
accuracy = 0.951, precision = 0.682, recall = 0.667, f1 = 0.674

If we look at the f1 score during training we can see that there's overfit to the training data making this model unreliable.



It is important to state that the approach is completely different and it lacks a fundamental step when we talk about NN: hyper parameter tuning: number of layers, number of neurons, dropout, learning rate, activation functions, etc.

So in the end, this is just a dummy/naive example to allow for further discussion if I step into the next stage of the recruitment process.

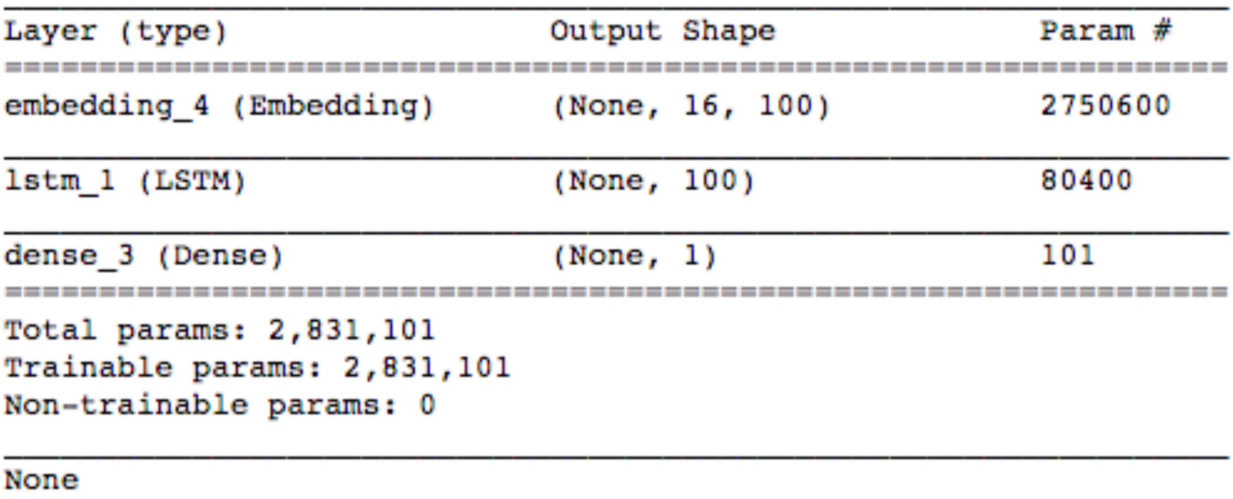
It is also important to mention that this NN does not share features learned across different positions of text.

Example:
- sentence 1: 'happy people care about you';
- sentence 2: 'this makes me incredibly happy';

If the NN learns that the word 'happy' in the first sentence (first word) contributes to the class 'non-hate speech' it will not transfer this knowledge to the second sentence because the word shows up in a different position of the sentence (last word);

One analogy that can be done with images (using Convolutional NN): if the CNN learns how to identify dogs in a image it will know how identify dogs in any part of the image: bottom left, top right, center, etc…

Another NN architecture that was tested was and NN with LSTM:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 16, 100)           2750600
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               80400
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 101
=================================================================
Total params: 2,831,101
Trainable params: 2,831,101
Non-trainable params: 0
_________________________________________________________________
None
```

This architecture allows to take into account sequences of words taking into account their position in the sentence. Although this method was not fully covered, theoretically it should be the method that best fits the purpose.

That resulted in the following scores:
accuracy = 0.915, precision = 0.463, recall = 0.759, f1 = 0.575

Summary of results

- LR_bow: accuracy = 0.956, precision = 0.726, recall = 0.603, f1 = 0.659
- LR_tfidf: accuracy = 0.952, precision = 0.686, recall = 0.585, f1 = 0.631
- RF_bow: accuracy = 0.952, precision = 0.698, recall = 0.558, f1 = 0.620
- RF_tfidf: accuracy = 0.931, precision = 0.503, recall = 0.647, f1 = 0.566
- NN: accuracy = 0.951, precision = 0.682, recall = 0.667, f1 = 0.674
- NN_LSTM: accuracy = 0.915, precision = 0.463, recall = 0.759, f1 = 0.575

Note: cross-validation was not performed at all for NN and NN_LSTM which means that the threshold was not optimised for the metric f1_score.

Looking at the results the best f1_score was obtained using NN.

Comments

Follow some ideas that would improve the results:

- Cross-validation in order to tune the hyper parameters of the models (for instance, through grid search);
- Convert more ‘slang’ from the twitter to a common form;
- Ensemble modeling - example: since logistic regression proved to be best at ‘recall’ and random forest at ‘precision’ both could be combined to get a better overall output;
- In the case of NN use pre-trained word embedding and more suited architecture for this purpose - https://arxiv.org/abs/1408.5882 ;
- For NN try different optimisation algorithms and loss functions;