# REPORT

ARSEN KUZMIN

## OVERVIEW

I evaluated LSTM model on twitter dataset and compared it with the Naïve Bayes.

## DATASET

I have chosen twitter dataset: link and tried to solve sentiment classification problem using LSTM.

## DATASET EXAMINING

The sentiment classification is quite challenging  problem due to following reasons:

1) Tweets are quite short.
   Examples:
   a. oh no!
   b. Ok then!
2) Sometimes it is impossible to derive sentiment from a tweet without knowing context (Replies).
   Examples:
   a. no it doesn't
   b. not yet!!
3) Users like to repeat letters to express more feeling
   Examples:
   a) finished watching FRIENDS so sad, WHY WHYYYYYY DID IT HAD TO ENNNDDDD
   b) its going to be a good day, yes sir a gooooooooooooooooood day!
4) People use slangs and words shortening
   a. Aw babe i kinda thot u'd b told that  but it will get better if u rest it
   b. Nice pic boo OOOw

## DATASET PREPROCESSING

1. Since some tweets contain redundant letters like in "gooooooooooooooooood day" I decided to transform 3 or more repeated letter to 2 ones.
2. Also tweets contain the user names like @AliiWynn. It does not give any information for sentiment classification so I removed such substrings.
3. I used TweetTokenizer for words tokenisation. (The name speaks for itself).
4. I removed all words which contain non alphanumeric symbols and converted them to lower case.
5. Finally, I used PorterStemmer to normalize words.

## FEATURES EXTRACTION

I used **Word2Vec** as features extractor for the words. Genism library was used in order to build Word2Vec. Model was learning 50 epochs with following parameters:

- Size = 32
- Window =10
- Min count =2

## RESULTS OF WORD2VEC MODEL

The picture below shows the closest words to “bad”. (Words have such strange forms due to stemming.)

```
[('horribl', 0.7160726189613342),
 ('wors', 0.6436370015144348),
 ('worst', 0.6216478943824768),
 ('unlov', 0.6133800148963928),
 ('good', 0.6130462288856506),
 ('crap', 0.5749754905700684),
 ('nasti', 0.5736783742904663),
 ('sunbath', 0.5698460340499878),
 ('exactli', 0.5638651847839355),
 ('ugh', 0.5378111600875854)]
```

## CONVERTING SENTENCES TO VECTOR

In order to use Embedding layer as first layer of my model, I converted each word to integer number from -1 to number of words in word2vec. The number corresponds to index of the word in word2vec dictionary if word in this vocabulary otherwise -1 is assigned to the word. This trick is done in order to use weights of word2vec in Embedding layers and make it trainable to add flexibility to the model. Also it to makes good weights initialisation of Embedding layer.

To represent sentence as vector I converted each word to integer number and restricted length of vector (by maxlen constant which in my case equals to 80). If there are not enough words to make vector I pad the vector with 0s.

## MODEL

**1st Layer** as was mentioned is Embedding Layer Turns positive integers (indexes of neurons that are activated) into dense vectors of fixed size. The weights of this layer were initialized using weights derived from Word2Vec model.

**2nd and 3rd Layers** are 1D convolution and 1D MaxPooling. The data has a one-dimensional spatial structure in the sequence of words in tweets and the CNN may be able to pick out invariant features for good and bad sentiment. This learned spatial features may then be learned as sequences by an LSTM layer.

**4th layer** is LSTM. LSTM. Our data is sequence of word; we need to remember previous context of the sentence in order to make sentiment classification

**5th Layer** is Dense Layer with 1024 neurons. This layer is added to make the model more flexible.

**6th Layer** is also Dense with 1 neuron and with sigmoid activation function. This neuron says whether the tweet is positive or negative.

## REGULARISATION

To make model not overfit training data I used Dropout. Dropouts are added between each layer and their probabilities equal to 0.25. Also there is recurrent dropout in LSTM and its probability also 0.25

## SUMMARY OF THE MODEL

| Layer (type) | Output Shape | Param # |
|---|---|---|
| embedding_11 (Embedding) | (None, None, 32) | 485376 |
| dropout_20 (Dropout) | (None, None, 32) | 0 |
| conv1d_11 (Conv1D) | (None, None, 32) | 5152 |
| max_pooling1d_1 (MaxPooling1 | (None, None, 32) | 0 |
| lstm_10 (LSTM) | (None, 32) | 8320 |
| dense_19 (Dense) | (None, 1024) | 33792 |
| dropout_21 (Dropout) | (None, 1024) | 0 |
| dense_20 (Dense) | (None, 1) | 1025 |

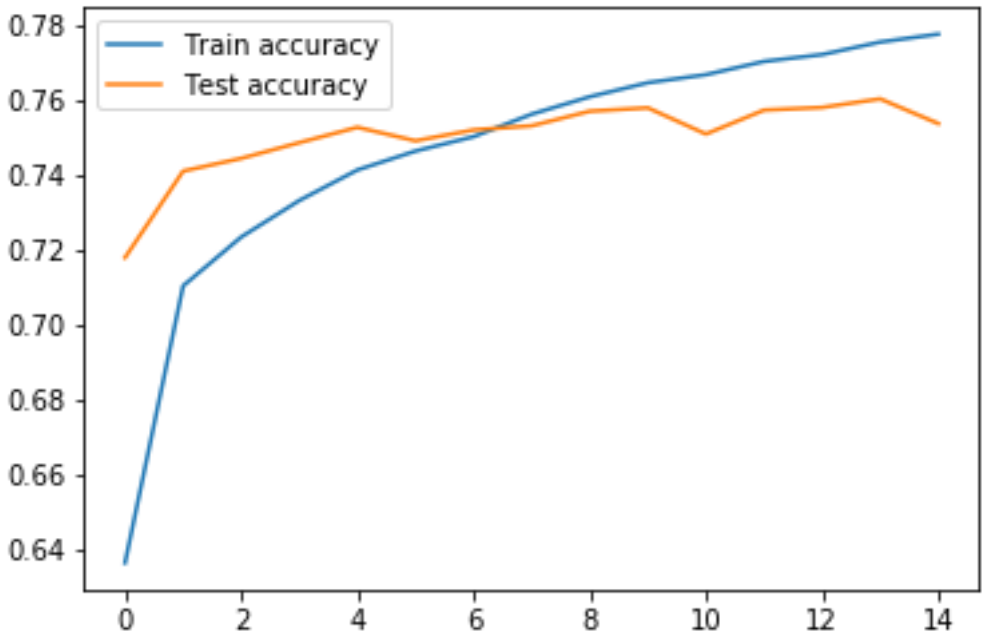
```
Total params: 533,665
Trainable params: 533,665
Non-trainable params: 0
```

## RESULTS

On test set (10% of whole data) final model shows 75.38% accuracy.

During training model showed maximum 76.07% on validation set.

Picture shows train and test accuracies during 15 epochs



## BASELINE

I applied same model as in: link. The results are quite fair because model is applied for the same preprocessed data as for my model.

Accuracy of Naïve Bayes: 60.6%

## PREVIOUS TRIES

1) I tried to represent sentence as average of its words. This showed 61.1% accuracy
2) I tried to represent sentence as concatenation of average, max, min of its words. This showed 68.2 % accuracy