Sentiment Analysis on Tweets with LSTM

Karina Reyes Santiago
May 27, 2020

Abstract

This report presents the implementation of sentiment analysis on tweets using a Long Short-Term Memory network.

Introduction

Sentiment analysis is the process of interpretation and classification of emotions within text data. [1] This field has acquired huge importance in social media analytics, because it allows businesses to identify their customer sentiments towards its brands and services. As Twitter is one of the most used social networks nowadays, Twitter sentiment analysis has become an important task to classify tweets as negative, neutral or positive.

Dataset

The dataset used in this project was obtained from Sentiment140 [2] and it contains 1.6 million tweets in English with no emoticons. It is important to mention that the polarities in this tweets' dataset only include negative and positive labels, so this project is attached to this binary classification.

Data preprocessing

Training a classifier using tweets is difficult due to the difference of length and the use of emoticons, usernames and links in the text. That is why the data needs to be preprocessed and cleaned before the next step.

As the dataset in this project has already no emoticons in it, the cleaning part just focus on converting every tweet text into lowercase and removing links and usernames. Also, as stop words are very commonly used in English and have no contextual meaning, they are also removed to avoid a bias caused by them in the training.

In the other side, the labels for each tweet in the dataset are annotated as 0 for negative and 4 for positive. So, in order to keep a common binary classification model, a label encoder is used to transform the labels into 0 for negative and 1 for positive.

In this project, the splitting of data is set to 80% for the training dataset and 20% for the test dataset.

Tokenization

By this point, the tweets are still normal sentences with punctuation. The tokenization process changes that sequence of characters into tokens. This means, "chopping" the sentences into pieces, convert each word into an index and throwing away characters as punctuation. [3]

Also, as tweets could have different lengths and the LSTM model needs inputs of the same length, a pad function is used to ensure the same length in all the input texts. In this case, the maximum length from the training dataset is taken to pad all the text sequences, both in training and test datasets.

Word embedding

A word embedding is a representation for a text vocabulary, where words that have similar meaning have similar representations. [4] This allows to capture the context and semantic of a word in relation with other words.

There are pre-trained word embeddings that can be used to avoid training an embedding from scratch. This project uses the pre-

trained Twitter GloVe [5] embedding with 27 billion tokens and 200d vectors.

Model training

The model architecture proposed for solve this task consists of a Sequential model with Embedding, Dropout, LSTM and Dense layers, as it shown in Figure 1.

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 50, 200)	58142800
dropout_19 (Dropout)	(None, 50, 200)	0
lstm_21 (LSTM)	(None, 200)	320800
dense_37 (Dense)	(None, 64)	12864
dense_38 (Dense)	(None, 1)	65

Figure 1. Model architecture

The embedding layer turns the input sequences into dense vectors. To configure this layer, the vocabulary size was set to 290714, the embedding dimension to 200 (as the 200d vectors from GloVe was used) and the input length to 50.

After the embedding layer, a dropout layer was used to prevent overfitting. The rate for this layer was set to 0.2.

The LSTM layer works with a mechanism known as cell states, which allows it to learn the context of the words selectively forgetting and remembering things. [6] The set up for this layer was 200 units and a dropout rate of 0.2.

Finally, two dense layers are added to the model. The first one with a set up of 64 units and ReLU as the activation function. The second dense layer, and last of the model, was set to 1 unit and a sigmoid function for the binary classification.

The model uses Adam as optimization algorithm, binary cross entropy as loss function and accuracy to measure the performance of the model.

Also, to help the performance during learning process, a callback is used to reduce the learning rate at a factor of 0.1 when the validation loss has stopped improving. The model was trained for 15 epochs, with a batch size of 1024 and a validation split of 0.1.

Model evaluation

The results of the training process were the following:

```
Train loss: 0.4144
Train accuracy: 0.8060

Validation loss: 0.4240
Validation accuracy: 0.8021
```

Figure 2 shows the behavior of the train and validation accuracy through the epochs. And Figure 3 shows the behavior of the loss.

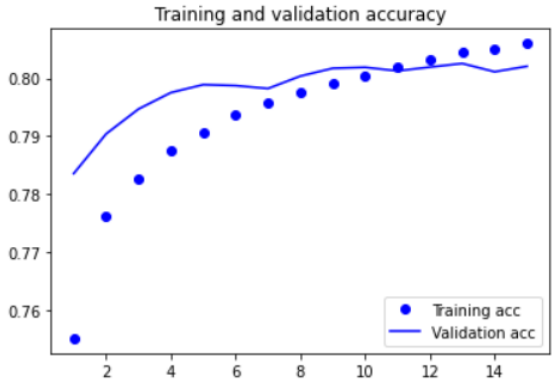

Figure 2. Train and validation accuracy

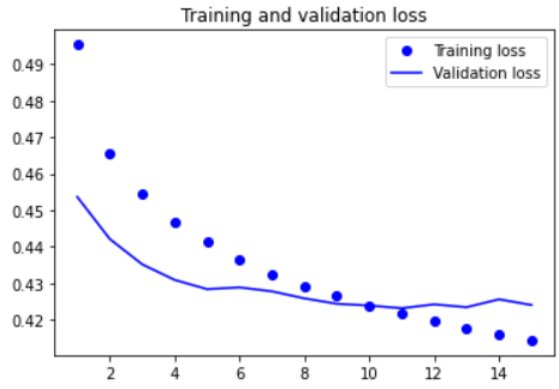

Figure 3. Train and validation loss

Also, the results for the test dataset were:

```
Test loss: 0.4292
Test accuracy: 0.8007
```

Model prediction

Ideally, the model is trained to predict negative and positive sentiment from tweets. So, the input queries used to test the model predictions were tweets taken from real Twitter accounts.

Two prediction examples are shown in Figure 4 and Figure 5. The outputs show the score obtained from the model and the decoded prediction. If the score is less than 0.5, the tweet sentiment is labeled as negative and if the score is greater or equal than 0.5, as positive.

```
Tweet:
I am selling almost all physical possessions. Will own no house.
Score: [0.39117682] Label: Negative
```

Figure 4. Tweet taken from @elonmusk account.

```
Tweet:
ICYMI: Earth is beautiful.
Score: [0.90122443] Label: Positive
```

Figure 5. Tweet taken from @NASA account.

Other model approaches

It is important to mention that the model architecture presented in this report was not the first model approach. Other models were used with different training configurations, but the results were not the best.

One of the first approaches was using the Wikipedia 2014 GloVe embedding. The model had the architecture shown in Figure 6. The results for this model within 10 epochs were 0.7680 for train accuracy and 0.7850 for validation accuracy. But these values improved a lot when the Twitter GloVe embedding started to be used instead.

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 50, 300)	87214200
dropout_1 (Dropout)	(None, 50, 300)	0
lstm_1 (LSTM)	(None, 100)	160400
dense_1 (Dense)	(None, 1)	101

Figure 6. Model architecture using Wikipedia GloVe

Another approach, now using Twitter GloVe embedding, had the architecture shown in Figure 7, using a 1D convolution layer before the LSTM one. Although the train accuracy with this model was 0.8274 in 20 epochs, the validation accuracy reduced to 0.7883 and the test accuracy to 0.7859, meaning that our model was probably overfitting.

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 50, 200)	58142800
conv1d_1 (Conv1D)	(None, 46, 128)	128128
lstm_20 (LSTM)	(None, 100)	91600
dense_35 (Dense)	(None, 30)	3030
dense_36 (Dense)	(None, 1)	31

Figure 7. Model architecture using 1D conv

Conclusions

Sentiment analysis is not an easy task as is related with Natural Language Processing. Tweets tend to be noisy and to use many different forms for words, which can be a problem for its analysis. However, the amount of data and the preprocessing of the data helped a lot in this project to achieve good results.

The pre-trained embedding seems to be an important decision as it showed better results when it was pre-trained specially with tweets than with Wikipedia articles.

The predictions obtained with new data are accurate to the content of the tweets, which

means that the model is working well with a ~80% accuracy.

The model could be improved adding new preprocessing data techniques and with other model architectures recommended for sentiment analysis as Bidirectional LSTMs.

References

[1] MonkeyLearn. (2020). Sentiment Analysis. Retrieved from: https://monkeylearn.com/sentiment-analysis/

[2] Sentiment140. (n.d.) For Academics. Retrieved from: http://help.sentiment140.com/for-students

[3] Stanford University. Tokenization. Retrieved from: https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html

[4] Brownlee, J. (2017). What are Word Embeddings for Text? Retrieved from: https://machinelearningmastery.com/what-are-word-embeddings/

[5] Stanford University. GloVe: Global Vectors for Word Representation. Retrieved from: https://nlp.stanford.edu/projects/glove/

[6] Srivastava, P. (2017). Essentials of Deep Learning: Introduction to Long Short Term Memory. Retrieved from: https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/

Code references

[7] Brownlee, J. (2017). How to Develop a Deep Convolutional Neural Network for Sentiment Analysis (Text Classification). Retrieved from: https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/

[8] Brownlee, J. (2016). How to Predict Sentiment Analysis From Movie Reviews Using Deep Learning (Text Classification). Retrieved from: https://machinelearningmastery.com/predict-sentiment-movie-reviews-using-deep-learning/

Project code available at: https://github.com/karyrs15/sentiment_analysis_project