



République Tunisienne
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Tunis El Manar École Nationale d'Ingénieurs de Tunis



Information and Communication Technologies

Sentiment analysis

Twitter Sentiment Analysis

Realized by:

MEFTAH GHAZI AND AMAMOU HAZEM

Classe : 3ATEL2

Supervised by

Sabrine Krichen and Fares EL Kahla

Academic year 2022/2023

Introduction

Natural Language Processing (NLP) is a field of artificial intelligence (AI) and computer science that focuses on the interaction between computers and humans using natural language. The goal of NLP is to enable computers to understand, interpret, and generate human language in a way that is both efficient and accurate.

NLP is widely used in many fields and applications such as chatbots, automated customer service, speech recognition, language translation, text summarization, social media monitoring, and many more. The field of NLP is constantly evolving with the emergence of new techniques and models, such as deep learning (DL) and transformer models.

Sentiment classification is a task in natural language processing (NLP) that involves determining the sentiment or emotional tone of a piece of text. The text can be a sentence, a paragraph, or even a full document, and the sentiment can be positive, negative, or neutral. Sentiment classification is used in many applications such as social media monitoring, customer feedback analysis, and brand reputation management.

We will go through all the key and fundamental concepts of NLP and Sequence Models in this work.

0.1 Data Preprocessing :

In this project, we are using Sentiment-140 from Kaggle. It contains 1,600,000 tweets extracted using the Twitter API. The tweets have been annotated (0 = Negative, 4 = Positive) and they can be used to detect sentiment.

0.1.1 Loading data :

First of all, we load our data and convert it into DataFrame using pandas.

As we can observe, the columns in the dataset do not have clear or descriptive names. To make it easier to understand and refer to, we should give them appropriate names. Additionally, since we will only be using the text to classify the sentiment, we can eliminate any columns that are not relevant to this task by removing them.

0.1.2 Data Distribution :

We're plotting the distribution for the dataset to see whether we have equal number of positive and negatives tweets or not.

```
[ ] # Load the Sentiment140 dataset into a Pandas dataframe
df = pd.read_csv('/content/drive/MyDrive/Sentiment-analysis/training.1600000.processed.noemoticon.csv',encoding='latin-1',header = None)
```

```
[ ] df.sample(5)
```

	0	1	2	3	4	5
967012	4	1827740859	Sun May 17 11:20:51 PDT 2009	NO_QUERY	Tweetygblue	's hair is frizzy cause of the rain... But she...
727811	0	2262911850	Sat Jun 20 23:30:41 PDT 2009	NO_QUERY	Trammy	@jayGREGO I know isn't that so sad? I'm an ol...
197858	0	1971090796	Sat May 30 06:11:09 PDT 2009	NO_QUERY	itselise	11:11 pm - I wish Shane didn't have to go home
1223803	4	1990459897	Mon Jun 01 04:55:51 PDT 2009	NO_QUERY	SWMaina	time to go offline. will tweet you latter toda...
601627	0	2221188689	Thu Jun 18 04:53:16 PDT 2009	NO_QUERY	rodiart	@annejuleart such a shame you have to spend t...

Figure 1: Raw data

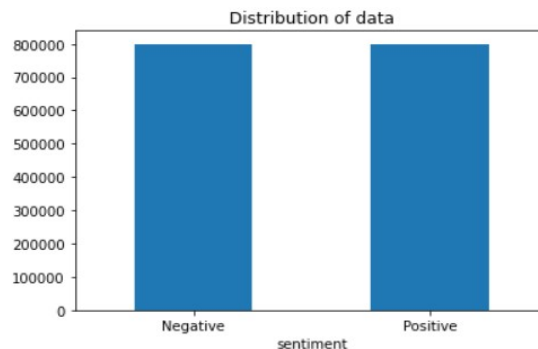


Figure 2: Data distribution

As we can see from the graphs, we have equal number of Positive/Negative tweets. Both equaling to 800,000 tweets. This means our dataset is not skewed and it is balanced, which makes working on the dataset easier for us.

0.2 Preprocessing the text :

Text Preprocessing is traditionally an important step for Natural Language Processing (NLP) tasks. It transforms text into a more digestible form so that deep learning algorithms can perform better.

Tweets usually contains a lot of information apart from the text, like mentions, hashtags, urls, emojis or symbols. Since normally, NLP models cannot parse those data, we need to clean up the tweet and replace tokens that actually contains meaningful information for the model.

The Preprocessing steps taken are:

- 1 Lower Casing:** Each text is converted to lowercase.
- 2 Replacing URLs:** Links starting with 'http' or 'https' or 'www' are replaced by '<url>'.
- 3 Replacing Usernames:** Replace Usernames with word '<user>'. [eg: 'Kaggle' to '<user>'].
- 4 Replacing Consecutive letters:** 3 or more consecutive letters are replaced by 2 letters. [eg: 'Heyyyy' to 'Heyy'].
- 5 Replacing Emojis:** Replace emojis by using a regex expression. [eg: ':)' to '<smile>'].
- 6 Replacing Contractions:** Replacing contractions with their meanings. [eg: "can't" to 'can not'].
- 7 Removing Non-Alphabets:** Replacing characters except Digits, Alphabets and pre-defined Symbols with a space.
- 8 Removing Stopwords:** stopwords are often removed from text data in order to focus on the important words that carry the most meaning such as "the," "and," and "is."
- 9 Stemming:** Normalize text by reducing words to their most basic form, so that words with the same meaning but different variations can be identified. For example, the words "running," "runner," and "ran" can all be reduced to the stem "run".

0.3 Analysing the data:

Now we are going to analyse the preprocessed data to get an understanding of it. We plotted Word Clouds for Positive and Negative tweets from our dataset and see which words occur the most.



Figure 3: Word cloud visualization for positive words

word embeddings using shallow neural network. Word2Vec can create word embeddings using two methods (both involving Neural Networks): Skip Gram and Common Bag Of Words (CBOW).

Training parameters:

- **Size:** The number of dimensions (N) that the Word2Vec maps the words onto. Bigger size values require more training data, but can lead to better (more accurate) models.

- **Workers:** Specifies the number of worker threads for training parallelization, to speed up training.

- **min_count:** min_count is for pruning the internal dictionary. Words that appear only once or twice in a billion-word corpus are probably uninteresting typos and garbage. In addition, there is not enough data to make any meaningful training on those words, so it's best to ignore them.

0.6 Tokenization and Padding datasets :

Tokenization is a common task in Natural Language Processing (NLP). It's a fundamental step in both traditional NLP methods like Count Vectorizer and Advanced Deep Learning-based architectures like Transformers.

Tokenization is a way of separating a piece of text into smaller units called tokens. Here, tokens can be either words, characters, or subwords. Hence, tokenization can be broadly classified into 3 types – word, character, and subword (n-gram characters) tokenization.

All the neural networks require to have inputs that have the same shape and size. However, when we pre-process and use the texts as inputs for our model e.g. LSTM, not all the sentences have the same length. We need to have the inputs with the same size, this is where the padding is necessary

Padding is the process by which we can add padding tokens at the start or end of a sentence to increase its length upto the required size. If required, we can also drop some words to reduce to the specified length.

- **Tokenizer:** Tokenizes the dataset into a list of tokens.

- **pad_sequences:** Pads the tokenized data to a certain length

Args in Tokenizer():

- 1 filters:** Characters to filter out from the sentences to tokenize.

- 2 lower:** True/False. Whether to lowerCase the sentence or not.

- 3 oov_token:** Out of Vocabulary token to put in for words which aren't in the tokenizer vocab.

Filters and lower has been turned off because we've already done those steps during the preprocessing step.

0.7 Creating Embedding Matrix

Embedding Matrix is a matrix of all words and their corresponding embeddings. We use embedding matrix in an Embedding layer in our model to embed a token into its vector representation, that contains information regarding that token or word. We get the embedding vocabulary from the tokenizer and the corresponding vectors from

the Embedding Model, which in this case is the Word2Vec model. Shape of Embedding matrix is usually the Vocab Length * Embedding Dimension.

0.8 Creating the Model

There are different approaches which we can use to build our Sentiment analysis model. We're going to build a deeplearning Sequence model. Sequence model are very good at getting the context of a sentence, since it can understand the meaning rather than employ techniques like counting positive or negative words like in a Bag-of-Words model.

0.8.1 Model Architecture

- 1. Embedding Layer:** Layer responsible for converting the tokens into their vector representation that is generated by Word2Vec model. We're using the predefined layer from Tensorflow in our model.
- 2. Bidirectional:** Bidirectional wrapper for RNNs. It means the context are carried from both left to right and right to left in the wrapped RNN layer.
- 3. LSTM:** Long Short Term Memory, its a variant of RNN which has memory state cell to learn the context of words which are at further along the text to carry contextual meaning rather than just neighbouring words as in case of RNN.
- 4. Conv1D:** This layer creates a convolution kernel that is convolved with the layer input over a single dimension to produce a tensor of outputs.
- 5. GlobalMaxPool1D:** Downsamples the input representation by taking the maximum value over the different dimensions.
- 6. Dense:** Dense layer adds a fully connected layer in the model. The argument passed specifies the number of nodes in that layer.

The last dense layer has the activation "Sigmoid", which is used to transform the input to a number between 0 and 1. Sigmoid activations are generally used when we have 2 categories to output in.

Model: "Sentiment_Model"		
Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 60, 100)	6000000
bidirectional (Bidirectional)	(None, 60, 200)	160800
bidirectional_1 (Bidirectional)	(None, 60, 200)	240800
conv1d (Conv1D)	(None, 56, 100)	100100
global_max_pooling1d (Global)	(None, 100)	0
dense (Dense)	(None, 16)	1616
dense_1 (Dense)	(None, 1)	17
Total params: 6,503,333		
Trainable params: 503,333		
Non-trainable params: 6,000,000		

Figure 6: Model summary

0.9 Training the Model

0.9.1 Model Callbacks

Callbacks are objects that can perform actions at various stages of training (e.g. at the start or end of an epoch, before or after a single batch, etc). We can use callbacks to write TensorBoard logs after every batch of training, periodically save our model, stop training early or even to get a view on internal states and statistics during training.

ReduceLROnPlateau: Reduces Learning Rate whenever the gain in performance metric specified stops improving.

monitor: quantity to be monitored.

patience: number of epochs with no improvement after which learning rate will be reduced.

cooldown: number of epochs to wait before resuming normal operation after lr has been reduced.

EarlyStopping: Stop training when a monitored metric has stopped improving.

monitor: Quantity to be monitored. **min-delta:** Minimum change in the monitored quantity to qualify as an improvement, i.e. an absolute change of less than min-delta, will count as no improvement.

patience: Number of epochs with no improvement after which training will be stopped.

0.9.2 Model Compile

The Model must be compiled to define the loss, metrics and optimizer. Defining the proper loss and metric is essential while training the model.

Loss: We're using Binary Crossentropy. It is used when we have binary output categories. Check out this article on losses.

Metric: We've selected Accuracy as it is one of the common evaluation metrics in classification problems when the category data is equal. Learn more about metrics here.

Optimizer: We're using Adam, optimization algorithm for Gradient Descent.

0.10 Model Evaluation

Since our dataset is not skewed, i.e. it has equal number of Positive and Negative Predictions. We're choosing Accuracy as our evaluation metric. Furthermore, we're plotting the Confusion Matrix to get an understanding of how our model is performing on both classification types.

0.10.1 Printing out the Learning curve

Learning curves show the relationship between training set size and your chosen evaluation metric (e.g. RMSE, accuracy, etc.) on your training and validation sets. They can be an extremely useful tool when diagnosing your model performance, as they can tell you whether your model is suffering from bias or variance.

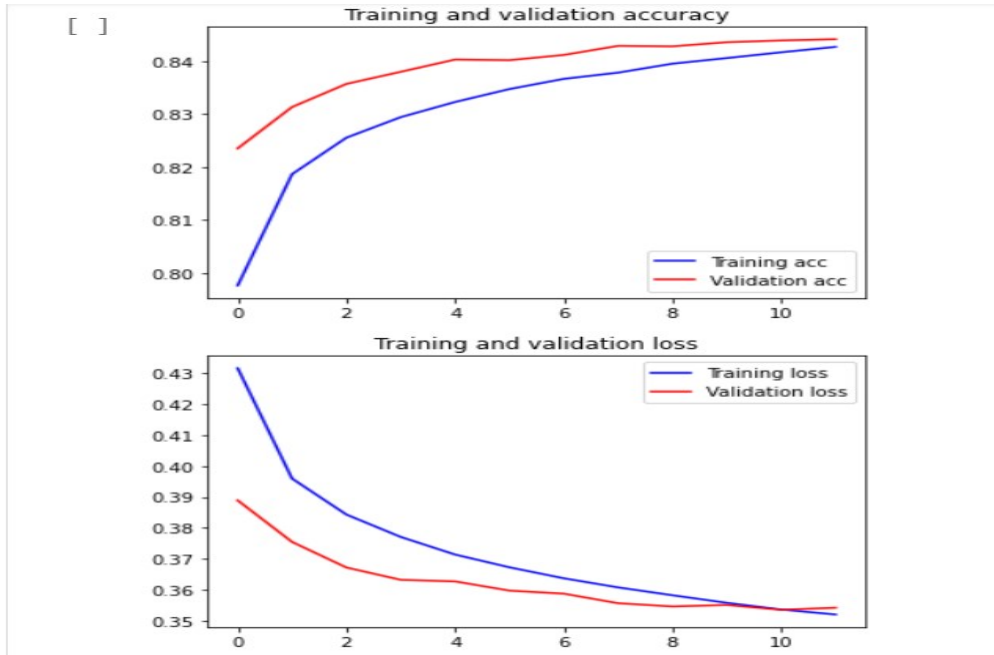


Figure 7: Accuracy and loss for training and validation

From the training curve we can conclude that our model doesn't have bias nor is it overfitting. The accuracy curve has flattened but is still rising, which means training for more epochs can yield better results. The Validation loss is lower than the training loss because the dropouts in LSTM aren't active while evaluating the model.

0.10.2 Confusion Matrix

From the confusion matrix, it can be concluded that the model makes more False Negative predictions than positive. This means that the model is somewhat biased towards predicting negative sentiment.

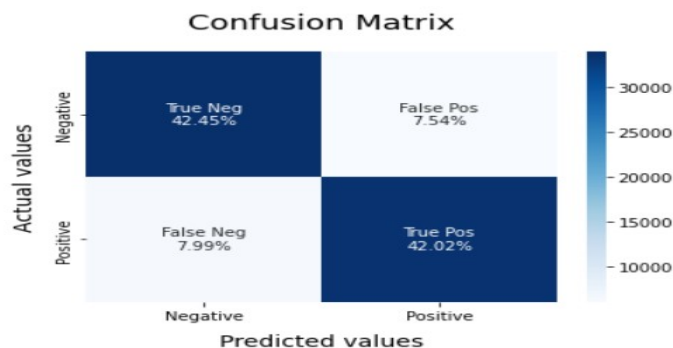
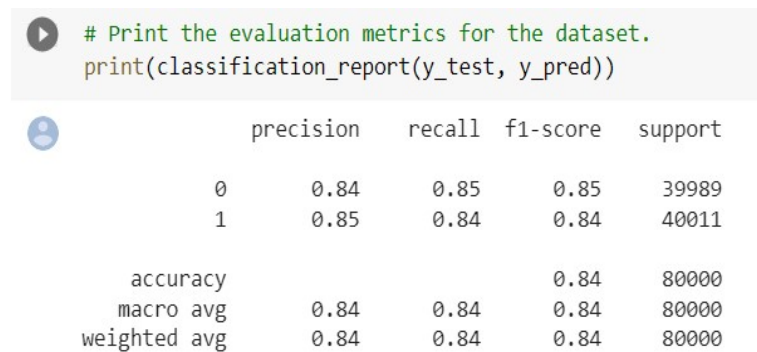


Figure 8: Confusion matrix

0.10.3 Classification Report

Using the classification report, we can see that the model achieves nearly 85% Accuracy after training for just 12 epochs. This is really good and better than most other models achieve.



```
# Print the evaluation metrics for the dataset.  
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.84	0.85	0.85	39989
1	0.85	0.84	0.84	40011
accuracy			0.84	80000
macro avg	0.84	0.84	0.84	80000
weighted avg	0.84	0.84	0.84	80000

Figure 9: Results

Conclusion

In conclusion, the use of a sequential model for sentiment analysis has been shown to be effective in accurately classifying text data into different sentiment categories. The model was trained on a large dataset of labeled text data and was able to achieve a high level of accuracy in both training and testing sets. Additionally, the use of pre-trained embeddings and various techniques such as dropout and LSTM layers helped to improve the performance of the model. However, it is important to note that there are limitations to the model, such as its inability to accurately classify sarcasm and irony, and that further research is needed to improve its performance in these areas. Overall, the use of a sequential model is a promising approach for sentiment analysis and can be useful in various applications such as social media monitoring and customer feedback analysis.