# Sentiment Analysis for Binary Reviews Classification

**Mario Sessa**
*Artificial Intelligence Project, A.A. 2020/21*
*mario.sessa@studio.unibo.it*
*0000983529*

**Abstract:** One of the most used branches of Artificial Intelligence within the social and linguistic context is Natural Language Processing. It consists in the analysis of natural languages in order to understand their meaning and, based on the information collected, make particular decisions. Within the project, we exploited the modern knowledge in Sentiment Analysis for the implementation of a binary classifier able to distinguish whether a review is positive or negative. The document will aim to provide a detailed explanation of the design phases, the obstacles faced and what are the results achieved.

## 1. Introduction

### *1.1. What is Natural Language Processing*

Natural Language Processing is a branch of artificial intelligence interested in giving automation in the texts and words analysis, trying to understand their uses and combinations given by the human being's expressions and making them understandable to a computer. This procedure is possible thanks to a combination of computational linguistics and the creation of language models using, for example, machine learning or deep learning techniques. Such models can lead a computer to process human language in textual or vocal form and understand its whole meaning. [1]

Human language is full of ambiguities that make it incredibly difficult to write software that accurately determines the meaning of the analyzed voice or textual data. Variations of sentences, expressions of sarcasm, metaphors, idioms, and other forms of irregularities make the analysis and classification of data even more complex for a machine that would take years to learn without using special techniques logic. [2]

There are many activities on which the objectives of Natural Language Processing bases; some of these tasks may include, for instance:

1. **Speech generation:** It's able to understand the human being language and generate sentences as well as humans does.
2. **Speech recognition:** It's able to analyze a speech made by a human being and understand the information he wants to transmit.
3. **Sentiment analysis:** It's able to analyze textual information in natural language and classify its emotions, qualities, sarcasm or other properties.

#### 1.1.1. Sentiment Analysis

Sentiment Analysis (also called opinion analysis) can present as the contextual extraction of a natural language text capable of extracting personal information about the corpus of origin and understanding social opinion towards a brand, a product or a service. [3] With recent advances in deep learning, it has been possible to expand sentiment analysis and improve its performance. Such artificial intelligence techniques are used for statistical company research, which can classify aspects such as, for instance, the satisfaction of a user concerning a service or a product purchased. [4]

### *1.2. Project Goals*

The project will found on applying learning methods to develop a Natural Language model that can distinguish, given an input text, whether a review is positive or negative. The training and verification datasets of correctness

will provide by Yelp, a web review platform that provides an open-source reviews data source for academic purposes.

## 2. Data Loading

### *2.1. Structure Analysis*

The structure of the Yelp dataset represents a relational database in Json format. This means that, within the attributes of each table, there are references to identifications and foreign keys typical of the relational topology. Since the goal of the project is to have to classify one or more reviews into positive or negative class, the interest is limited only to the 8,365,403 reviews without any connection to the type of company which they refer.So, we focus exclusively on the analysis of a corpus and the formalization of rules able to link the results of analysis with possible values or labels data input (in our case, a parameter generated by the stars attribute). The types of data, related to the attributes of the reviews, are known within the documentation of the dataset itself, this can help us to facilitate the loading of information through an identification of the bytes that determine its structure and speed up the loading process.

```
{
    // string, 22 character unique review id
    "review_id": "zdSx_SD6obEhz9VrW9uAWA",

    // string, 22 character unique user id, maps to the user in user.json
    "user_id": "Ha3iJu77CxlrFm-vQRs_8g",

    // string, 22 character business id, maps to business in business.json
    "business_id": "tnhfDv5Il8EaGSXZGiuQGg",

    // integer, star rating
    "stars": 4,

    // string, date formatted YYYY-MM-DD
    "date": "2016-03-09",

    // string, the review itself
    "text": "Great place to hang out after work: the prices are decent, and the ambience is…",

    // integer, number of useful votes received
    "useful": 0,

    // integer, number of funny votes received
    "funny": 0,

    // integer, number of cool votes received
    "cool": 0
}
```

Fig. 1. Json review structure types

### *2.2. Chunks Loading*

The size of the reviews file is 5.9 GB. In Python, libraries such as *pandas* provide functions that can load their content directly. This, however, has repercussions in execution time; caused by the enormous amount of data to be processed. We are able to work around this problem, using a method based on splitting into chunks which divides the file into some partitions and read them in distinct sequences. In addition, the reading of information can also be speed up using a well-knows schema of data type information which we should load on. Since the Yelp documentation provides us with this kind of metadata, through the *numpy* library, we can pass the data types to the *dtype* parameter of the *read_json* function made in *pandas* and, together with a *chunksize*, we are able to load data in low memory mode. The result of this operation consists in the same-sized disjoint loaded chunks objects represented by a subset of different data rows. Finally, we could go to recompose the various reviews in a single data block, going to merge them into a pandas dataframe. The division into blocks reduces the use of memory to the size of one chunk at a time, this allows a faster computation, a space complexity reduction and also provides

the possibility of being able to carry out data manipulation independently on each individual block. In our case, we took advantage of this property by removing some attributes that are not useful for the project goal, which are represented by all the relational attributes related to the foreign keys and the primary key of the reviews.

### *2.3. Execution Time*

The loading execution time on the JSON file is around 2 minutes. It must be considered that the operations to be carried out during the data loading include:

1. Read and put data in chunks
2. Removing columns for each loaded row *review_id business_id, user_id*
3. Inserting block in the chunks list
4. Concatenating chunks list items to form the dataframe

Removing some columns does not have any particular impact on execution time. Increasing the chunks size, we may have a reduction in overall execution time. However, if the blocks are formed by an excessive amount of elements, the low memory usage property can be lost, going to overload it, increase the complexity of space and slow down the loading process.

## 3. Data Analysis

### *3.1. Analysis Goals*

One of the most critical steps to understand the domain and data topology is Data Analysis. During the project, we mainly focused on analysing three types of information: star distribution, correlations with secondary attributes (cool, fun and useful) and properties related to the text field. We will use the information collected at this stage to be able to manipulate the data correctly. We used the following libraries during this phase: *matplotlib*, *seaborn* and *worldcloud*. They can display data in a clear and searchable way so that they can detect unknown properties.

| | stars | useful | funny | cool | text | date |
|---|---|---|---|---|---|---|
| **0** | 4.0 | 3 | 1 | 1 | Apparently Prides Osteria had a rough summer a... | 2014-10-11 03:34:02 |
| **1** | 4.0 | 1 | 0 | 0 | This store is pretty good. Not as great as Wal... | 2015-07-03 20:38:25 |
| **2** | 5.0 | 0 | 0 | 0 | I called WVM on the recommendation of a couple... | 2013-05-28 20:38:06 |
| **3** | 2.0 | 1 | 1 | 1 | I've stayed at many Marriott and Renaissance M... | 2010-01-08 02:29:15 |
| **4** | 4.0 | 0 | 0 | 0 | The food is always great here. The service fro... | 2011-07-28 18:05:01 |

Fig. 2. Review dataframe head illustration

### *3.2. Stars Analysis*

The stars field is one of the main ones in the Yelp dataset; it is necessary to match the prediction of the type of review to the numerical value (label) represented by this field. The results derive from the study on the classes of values that the real stars can assume by considering the 5-level classification given by the input file and the binary one used in the subsequent phases. As can be seen from the first graph below, there is an imbalance between the various valuation classes. The amount of reviews with star values of 4 and 5 is more significant than the remaining domain entries.

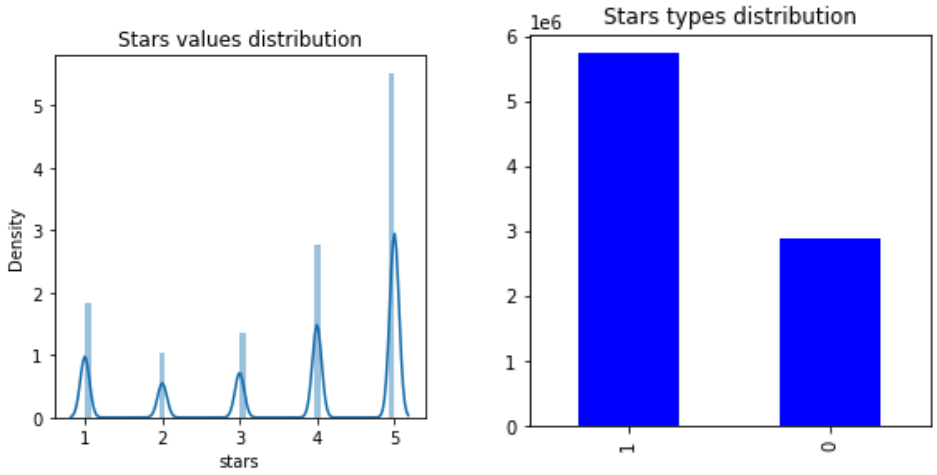


Fig. 3. Original and polarized stars distributions plots

Looking at the second graph, however, we can see how the division of the values of stars into binary classes contains a slight imbalance in favour of positive reviews. Consequently, there will be a balancing procedure to make the two types of values equipotent in the pre-processing phase. Our analysis continues to define a possible correlation with the length property of the text field with the value of the stars; results show us that there is a uniform distribution of texts length on the variation of stars values according to the number of the distribution of the stars.

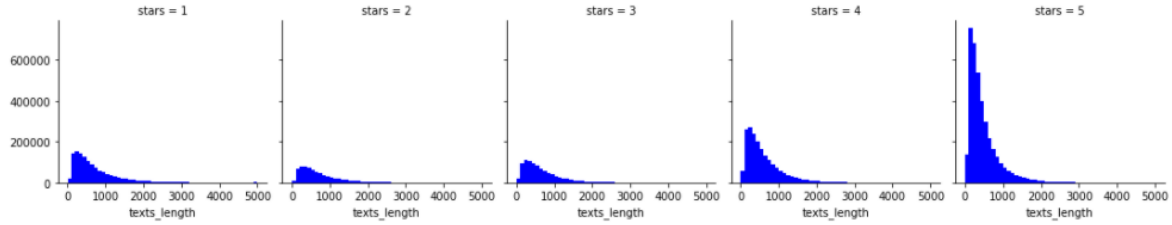


Fig. 4. Lengths in according to the different stars value

### *3.3. Secondary Features Correlations*

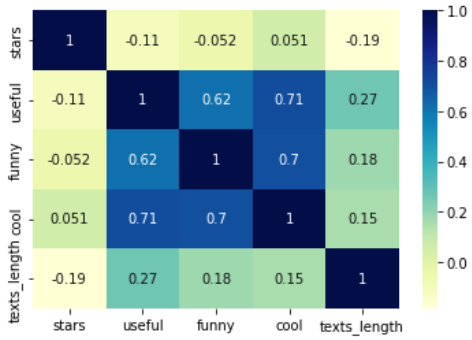


Fig. 5. Correlation Heatmap

Within the text and stars field, we also have secondary features that can evaluate a specific aspect of the main field into the initial dataset. These kinds of features are cool, funny and useful. These values will identify possible correlations between their values and the stars one. Furthermore, we will try to find also possible references with text field properties. Steps under the analysis on correlation found on the internal data functions *pandas.corr()*, and results will show on a heatmap plot. As we can see from the previous figure, compared to the original dataset, we have added a new attribute: *textLength*. This addition makes to find possible correlations between the values of the various fields and the length of the text. Consulting the heatmap, we can see no strong correlation between the secondary fields and those of text or stars. Furthermore, dependencies between secondary values alone are not crucial to the project goals and will not consider.

### 3.4. *Text Analysis*

The text field is the core of the Sentiment Analysis. It depicts a corpus on which to train the models for the generation of a performing classifier. During the analysis, we focused our interest on the frequency of terms going mainly to use a mapping through the worldcloud library.

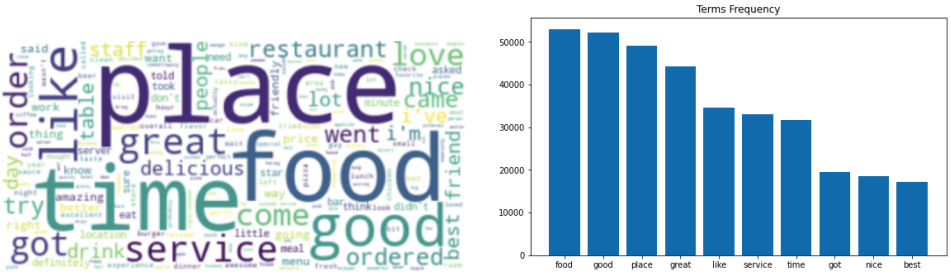


Fig. 6. Reviews frequency on a wordcloud and bar schema

As we can see, the most used terms refer to food or service. Time, orders, and other high-frequency nouns refer to quality metrics typically used in a review. Most reviews are positive cases, and it is not surprising that the most frequent adjectives represent positive judgments. Given the inaccuracy of the worldcloud, we also opted for a display of the most frequent terms using a simple bar chart. One of the positive aspects resulting from the analysis of the frequency of the terms is that there are not so high several stopwords; this can mean that in the training phase, we will have the possibility to use a word vector efficiently based on term frequency.

## 4. Data Pre-Processing

### 4.1. *Data Cleaning*

During the start of the pre-processing phase, we cleaned the dataset from useless features to improve training. Consequently, we have removed features such as ”cool”, ”funny”, ”useful”, and ”textLength” because, according to the analysis phase, we obtained negative results for the possible correlations with the main features like stars and text. In the next step, we are going to format the remaining data to optimize them for modelling.

### 4.2. *Stars Pre-Processing*

The project consists of the classification of reviews in the positive and negative classes. If we want to reach this goal, we have to transform the domain of the stars field from five classes to two ones (positive and negative). We set reviews values between zero and three to zero (negative value) while those with four or five stars to one (positive value). Finally, we obtain a dataset of positive and negative reviews. The unbalance between positive and negative reviews could be a problem during the set splitting; so, we created a new version of the dataframe with only 200,000 reviews: 100,000 positive and 100,000 negative. In conclusion, we obtain a normalized stars field labelled the model input dataset used in the training phase.

### 4.3. *Text Pre-processing*

One of the critical operations that occur when talking about Natural Language Processing is preparing data in natural language to be directly usable by a classifier or a predictor. Below, we are going to list all the procedures performed on the text field:

#### 4.3.1. Null column removals

It can happen that in many review datasets, some columns have an empty text; this is because we have the possibility of being able to do a review by selecting only one rating for the stars field, leaving the empty text field. To avoid these reviews, we have decided to remove them before balancing the polarized classes.

#### 4.3.2. Reduction of texts in lowercase

During text analysis, we need to avoid the forms of the different words for the same term. So, we transformed every character in lowercase. These changes can reduce the concepts ambiguity and models training time.

#### 4.3.3. Lemmatization

During writing text, it is possible to express various concepts using different grammar forms of the same term. Taking, for example, a verb like ”to go”, its present tense can have two forms ”goes” for the third person singular, or ”go” in the other cases. We can also have such form problems for other words such as adjectives, nouns and adverbs. The lemmatization involves a filtering procedure in which terms in different forms transform into a basic form (such as the infinitive or singular form, respectively, for verbs and nouns). We can carry out this procedure, starting with the word tagging, a function to associate a type tag for every term in the text according to their syntactic roles like nouns, verbs, adverbs and adjectives. Through a word-tag mapping, we can perform the lemmatization operation using the *WordNetLemmatizer* object from the *nltk* library; and change the word form in keeping with the associated tag.

#### 4.3.4. Stopwords removal:

Stopwords represent the most commonly used words in a particular language. In general, stopwords define as all those terms that do not have their semantic meaning but are used only for syntactic purposes: articles, prepositions, some adverbs or pronouns. In Sentiment Analysis, the removal of stopwords is crucial for understanding the dependency between sequential terms. It is necessary to pay attention to which stopwords to remove and which ones can be useful or essential for the correct learning of the models. In our case, given that we should rank whether a review is positive or not, we need to set off when certain qualities of the service we are evaluating are present or not. To do this, users express themselves using affirmative or negative forms, which may be crucial in deciding the type of reviews. For this reason, we have decided not to remove negation terms such as ”not”, ”can’t”, ”don’t”, and other forms of negations.

#### 4.3.5. Encoding

Since machine learning creation, models often elaborate mathematically. The current state of the word sequence is not able to be elaborated as an input for some of the proposed models. So, we tried to prepare input data with different numerical representations according to the nature of the model. For the Naive Bayes and Logistic Regression models, we used the bag-of-word approach with *CountVectorizer* or tf-idf properties like $TfidfVectorizer$; which transform a text in a sequence of numbers based on the counting feature or the tf-idf one. About the deep neural networks, we introduced an encoder layer that bases on a *TextVectorizer* (word2vec), which gives a unique number of each word of reviews vocabulary.

## 5. Data Modelling

### *5.1. Models Configuration*

The models used in the project are many; adopting an analysis on several alternatives allows us to understand which is the correct approach to follow. We will briefly describe which models use, which advantages they explore and how text classification can interface with such algorithms. Below are the main models used in the design phase.

#### 5.1.1. Multinomial Naive Bayes

It implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice. The distribution is parametrized by vector $\theta_y = (\theta_{y1}, ..., \theta_{yn})$ which defines the class type $y$ with $n$ features depending on the size of vocabulary. Given $\theta_{yi}$ as the probability $P(x_i|y)$ of feature $i$ appearing in a sample belonging to class $y$, we can estimate $\theta y$ vector by a smoothed version of maximum likelihood as frequency counting or tf-idf parameter. The classification aim is to define the probability to have a success assignment of a series of words $x_i, ..., x_n$ to a class $y$. In other words, we want to calculate: $y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$ which *argmax* identifies the class $y$ which gives the max value of $P(y)$ [4]

#### 5.1.2. Logistic Regression

Logistic regression uses an equation as the representation, very much like linear regression. Input values $x$ are combined linearly using weights or coefficient values to predict an output value $y$. A key difference from linear regression is that the modelled output value is binary values (0 or 1) rather than a numeric value. Below is an example logistic regression equation:$y = e^{(\beta_0+\beta_1*x)}/(1+e^{(\beta_0+\beta_1*x)})$ where y is the predicted output, $\beta_0$ is the bias or intercept term and $\beta_1$ is the coefficient for the single input value $x$. Each column in input data has an associated $\beta$ coefficient (a constant real value) that must learn from your training data. The actual representation of the model that we would store in memory or into a file is the coefficients in the equation.

#### 5.1.3. Deep Learning using biLSTM configuration

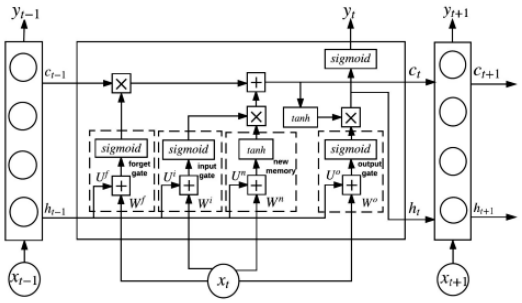


Fig. 7. Long-Short Term Memory architecture

The proposed model is the classic *Sequential* one, which has an initial encoder given by our *TextVectorizer*, which transforms a sequence of words in a sequence of numerical values according to the deep learning procedures. On the next, there is an *Embedding* layer with an input dimension equal to the sequence of word indexes and has a sequence of trainable vectors as output. We put the *mask_zero* to handle varying sequence lengths. Using *Bidirectional* will run our inputs in two ways, one from past to future and one from future to past and what differs this approach from unidirectional is that in the *LSTM* that runs backwards, we preserve information from the future and using the two hidden states combined we are able in any point in time to preserve information from both past and future. In other words, we can propagate the input in forwards and backwards directions, combine the outputs for each sentence term and train information. Finally, we have *Dense* layers, which takes values of the encoded-word vector and calculate the respective output to show in the prediction. We have used the binary $cross-entropy$, which calculate the loss distribution as differences between the true label's distribution and the predicted one. Furthermore, we used the *Adam* optimizer, which is considered the best optimizer for the gradient descent estimation. [4]

#### 5.1.4. Adding Convolutional layers to biLSTM configuration

The initial idea is to add a 'Conv1D' and a 'MaxPooling1D' between the 'Embedding' layer and the 'Bidirectional' one. The aim of the convolution, which is typically used for image and not for text, is to find some previous features before the 'Bidirectional' sequential approach based on 'LSTM'. The logic behind 'Conv1D' is to use a 'kernel_size' of 2, so we have a window of 2 cells because it is a 1-dimension convolution procedure; combining with the 'strides' at 1, the filter will move one pixel, or unit, at a time; it means that we consider a 'Linear' analysis on the embedding input much faster than a 'Linear' layer analysis with the same precision on the 2-gram analysis. We defined a 'kernel_size' at 2 because many neighbours can be associated with adjectives and nouns. Typically, using a convolutional or linear approach can reach this kind of feature. Model's problem may be only the overfitting, which can be mitigated using dropout's layers or callbacks. [5]

#### 5.1.5. Configuration with two consecutive biLSTM

The convolution layers make some overfitting but have less time to train than other models. We tried to remove the convolutional part of the model and, by adding a layer, we obtained a good model. One problem of this configuration is the enormous amount of training time caused by the bidirectional LSTM layers, which computes sequentially, increasing the *batch_size* should reduce training time, so we put in the fit method a *batch_size* of 54, which means that every computation has in input 54 samples at the same time. [3]

### *5.2. Regularization*

Regularization needs for some of the previous models. Our solution is based on adding dropout layers between the dense layers and a general callback to define if the value of loss for validation becomes worth the previous epoch. Finally, we defined models based on the reductions of neurons to a minimal form because we want to avoid a significant bias between training and validation set metrics. [3]

### *5.3. Optimizer and Loss Function*

As shown in paragraph 5.1, the typical loss function for binary classification problems is *binary_crossentropy* to define the binary difference between the real label and the predicted one. We used *adam* for the optimiser because

it is considered the best one for classification problems. It founds on a stochastic gradient descent method based on adaptive estimation of first-order and second-order moments. [4]

## 6. Data Results

### *6.1. Evaluation Parameters*

Evaluation parameters that we consider are the accuracy and F-score values. The F-score, also called the F1-score, measures a model's accuracy on a dataset. It uses to evaluate binary classification systems, classifying examples into 'positive' or 'negative'. The F-score combines the precision and recall of the model, and it defines the harmonic mean of the model's precision and recall. The F-score is commonly used for evaluating information retrieval systems such as search engines and for many kinds of machine learning models, particularly in natural language processing.

### *6.2. Results Table*

| Parameter | Accuracy | F1_score |
|---|---|---|
| Count Naive Bayes | 79.83% | 80.05% |
| Tf-idf Naive Bayes | 79.02% | 80.05% |
| Count Logistic Regression | 82.35% | 82.44% |
| Tf-idf Logistic Regression | 82.39% | 82.28% |

| Parameter | Accuracy | Loss | F1_score | Time |
|---|---|---|---|---|
| Bidirectional LSTM | 86.86% | 28.96% | 87.66 | 15min 8s |
| Convolution biLSTM | 87.30% | 28.56% | 87.84 | 9min 6s |
| Sequential biLSTM | 87.45% | 29.41% | 66.70 | 25m 11s |

Table 1. Results Tables

### *6.3. Bidirectional LSTM plots*

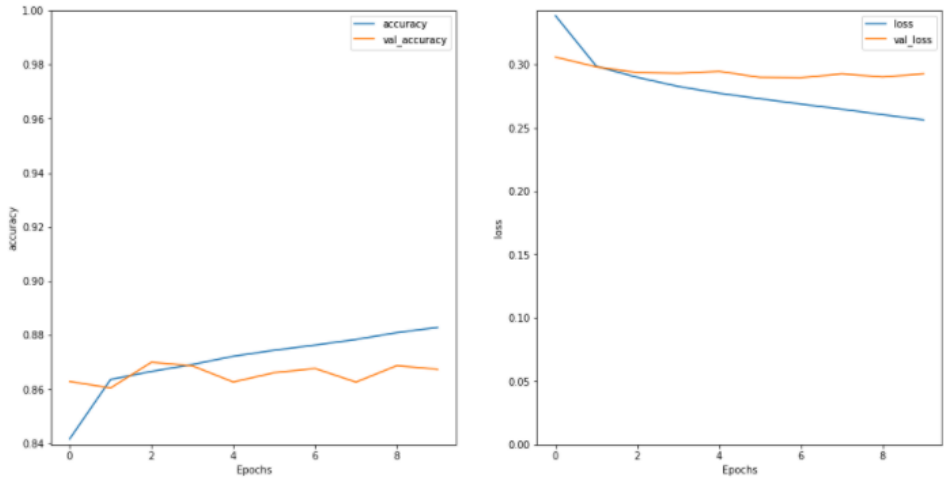


Fig. 8. biLSTM accuracy and loss values in training phase

Obtained the model, we can see the results during the training phase. Training accuracy was increasing slowly; meanwhile, validation accuracy has a mean value of around 86%. About the loss value, training loss was decreasing during the epochs successions; meanwhile, the validation loss is at least 20%, we put only the 10% of the training set as a validation set, if we change the training and validation ratio, the model may obtain better results.

### 6.4. *Convolution biLSMT plots*

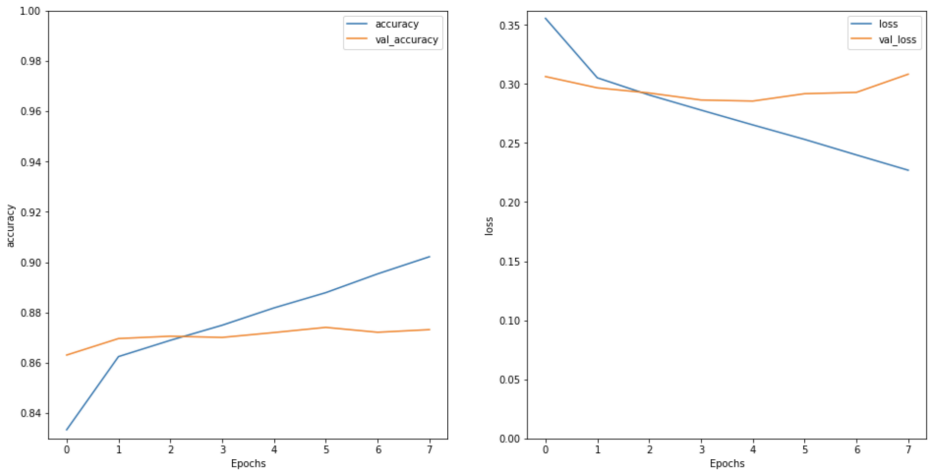


Fig. 9. Convolution biLSTM accuracy and loss values in training phase

Modified the model, we can see the results during the training phase. Training accuracy was increasing slowly; meanwhile, validation accuracy has a mean value of around 87% and is not linear compared with the training accuracy. Instead, loss validation is almost constant at 29% with a starting improvement that stops in 2 epochs later.

### 6.5. *Double biLSTM plots*

Results for the double bidirectional LSTM are good; validation and training accuracy is pretty similar until the 8th epoch, where the validation accuracy dropped down until the final increment Generally, the accuracy remains around 87% and the testing accuracy. Loss value is constant for the validation set around 29%; meanwhile, the loss function for the testing set is continuously decreasing.

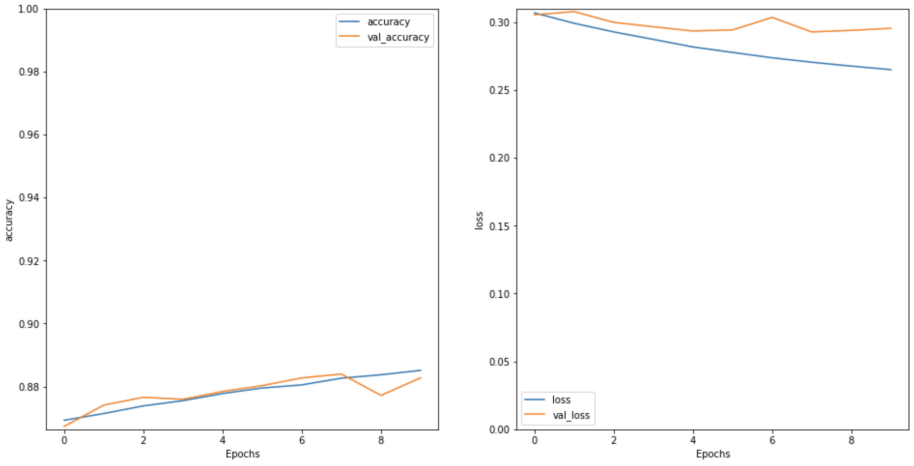


Fig. 10. Sequencial biLSTM accuracy and loss values in training phase

### 6.6. *Observations*

Improvement on the layer configuration increased accuracy and F1 score, adding a convolutional layer that can initially improve features extraction on token vectors, which gives an excellent initial increment to the validation set. However, it suffers from a margin of overfitting reduced by callbacks and dropouts. Adding more than one

bidirectional LSTM layer reduces drifts between the training and validation set, giving an optimal solution for the classification model.

## 7. Conclusions

A Deep Neural Network is an excellent solution to improve the accuracy of a sentiment prediction. Experimentation on combining more different layers like convolutional and bidirectional LSTM could be a good solution for many instances of this problem class. Our approach tested the potentiality of a convolutional neural network, typically used for image recognition, and how it works in complex systems like recurrent neural networks and, particularly, Long-Short Memory layers. Deep neural network state-of-art accuracy and f1 scores cannot reach in any case an optimal value, which is typically more than 90% of accuracy or f1 score, for instance. New models like generalized autoregressive pre-training for language understanding (XLNet) or bidirectional encoders based on transformers like BERT obtained the current state-of-art in Sentiment Analysis. So, a possible future work could be testing these 'new' approaches and comparing results with the current models one. [6]

## References


1. *Joseph, Sethunya & Sedimo, Kutlwano & Kaniwa, Freeson & Hlomani, Hlomani & Letsholo, Keletso. (2016). Natural Language Processing: A Review. Natural Language Processing: A Review.*
2. *Khurana, Diksha & Koli, Aditya & Khatter, Kiran & Singh, Sukhdev. (2017). Natural Language Processing: State of The Art, Current Trends and Challenges.*
3. *Luo, Tiejian & Chen, Su & Xu, Guandong & Zhou, Jia. (2013). Sentiment Analysis.*
4. *Lei Zhang, Shuai Wang, Bing Liu. Deep Learning for Sentiment Analysis: A Survey*
5. *W. Yue and L. Li, "Sentiment Analysis using Word2vec-CNN-BiLSTM Classification," 2020 Seventh International Conference on Social Networks Analysis, Management and Security (SNAMS), 2020, pp. 1-5*
6. *Lan, Zhenzhong & Chen, Mingda & Goodman, Sebastian & Gimpel, Kevin & Sharma, Piyush & Soricut, Radu. (2019). ALBERT: A Lite BERT for Self-supervised Learning of Language Representations.*