****

**Sentiment analysis models**

Juan David Garzón Diaz

Geovanny Alexander Quintero Velez

Johan Felipe Jojoa Bucheli

Department of engineering and applied science, ICESI University

Computation and discrete structures III

Andres Aristizabal Pinzon

November 24, 2023

# **Introduction**

Sentiment analysis, a subfield of natural language processing, involves discerning the sentiment expressed in textual data, typically categorizing it as positive or negative. In this project, the focus is on developing a sentiment analysis model using supervised learning with two types of neural networks: vanilla Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks. The significance of sentiment analysis lies in its applications across various domains, providing valuable insights into the emotions and opinions expressed in textual content.

**Objectives**

The primary objectives of this project are centered around the development and evaluation of sentiment analysis models. The dataset utilized is sourced from three distinct websites: Amazon, IMDb, and Yelp. Each sentence in the dataset is annotated with a binary label, indicating whether it conveys a positive sentiment (labeled as 1) or a negative sentiment (labeled as 0). The overarching goals include constructing robust sentiment analysis models using RNNs and LSTMs and conducting a comprehensive evaluation of their performance. Additionally, the project aims to identify and optimize the hyperparameters that contribute to the models' effectiveness.

**Methodology**

In this project, the focus lies in the development and evaluation of sentiment analysis models using supervised learning, specifically employing vanilla Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks. The initial step involves gathering data from the Sentiment Labelled Sentences dataset, encompassing reviews from Amazon, IMDb, and Yelp. Subsequently, the data is preprocessed through tokenization, lowercasing, and stopword removal. The methodology encompasses the implementation of a baseline DummyClassifier for performance comparison, followed by the construction and evaluation of the RNN and LSTM models. Hyperparameter tuning is conducted through grid searches for both RNN and LSTM to optimize their configurations. Furthermore, to ensure robustness, a function executes each model multiple times, capturing metrics on each iteration. The final step involves calculating the average of metrics, including accuracy, precision, recall, and F1-score, over these iterations for each model. The results are analyzed to identify the model with the most favorable overall performance, providing valuable insights into the effectiveness of the sentiment analysis models developed in this study.

## **Dataset details**

The sentiment dataset used is the "Sentiment Labelled Sentences Dataset" obtained from the UC Irvine Machine Learning Repository. The dataset consists of labeled sentences from three websites: Amazon, IMDb, and Yelp. Each website contributes 500 positive and 500 negative sentences, resulting in a balanced dataset. The dataset is prepared for supervised learning, with sentences labeled as 1 for positive and 0 for negative sentiments.

## **Preprocessing steps**

Preprocessing steps involve loading the data from multiple files, concatenating them into a single DataFrame, and shuffling the data for randomness. The NLTK library is used for tokenization, lowercasing, and removing stopwords. The preprocessing function, preprocess\_text, tokenizes the sentences, converts them to lowercase, and removes stopwords, ensuring that only meaningful words are retained for analysis. The final DataFrame includes the original sentences as well as a new column, 'processed\_sentence,' containing the preprocessed text.

## **Dummy, RNN & LSTM sentiment analysis models**

### **Dummy Classifier:** Model Description:

The Dummy Classifier is used as a straightforward baseline in classification tasks to assess the performance of more complex models. It is a naive model that makes classification decisions based on a specific strategy, in this case, the 'most\_frequent' strategy, which always assigns the most frequent class in the training set.

### Model Architecture:

The Dummy Classifier does not have a complex architecture, as its purpose is to establish a baseline performance without significant learning. It uses the strategy of always assigning the most frequent class without training any weights or additional parameters.

### Training Process:

### The training process of the Dummy Classifier is minimal or even nonexistent since it does not learn from the training data. The 'most\_frequent' strategy is determined during the fit operation, and predictions are made based on this strategy without adjusting any additional parameters.

### Considerations: The Dummy Classifier provides a useful baseline to compare the performance of more sophisticated models. In this case, the results indicate that any subsequent model should surpass these metrics to be considered significantly better than a constant assignment strategy. The next step would be to evaluate more complex models, such as RNN and LSTM implementations, to understand their performance compared to this simple approach.

### **RNN Sentiment analysis model:**

### Model Description:

The Recurrent Neural Network (RNN) is employed for sentiment analysis on preprocessed text data. RNNs are particularly suitable for capturing sequential dependencies, making them effective for tasks involving sequences of words or sentences.

### Model Architecture:

The RNN architecture for sentiment analysis comprises a series of recurrent layers that allow the model to maintain a memory of previous inputs while processing the current one. Each layer in the network is designed to capture and learn patterns in the sequential data, making it adept at discerning the sentiment conveyed through the text. The input layer takes preprocessed text data, and the recurrent layers process the information in a step-by-step manner, considering the context of each word in relation to the preceding ones. The final output layer produces sentiment predictions based on the learned patterns.

**Embedding Layer:**

* Converts words into dense vectors of fixed size.
* Vocabulary size: max\_words (10,000 words).
* Embedding dimension: 128.
* Input sequence length: max\_len (100 words).

| model.add(Embedding(max\_words, 128, input\_length=max\_len)) |
| --- |

**SimpleRNN Layer:**

* Recurrent layer processing the sequence of input vectors.
* 64 units to capture patterns in sequential data.

| model.add(SimpleRNN(64)) |
| --- |

**Dropout Layer:**

* Introduces regularization by randomly dropping connections during training.
* Dropout rate: 0.5.

| model.add(Dropout(0.5)) |
| --- |

**Dense Layer:**

* Output layer with one unit and a sigmoid activation function for binary classification.

| model.add(Dense(1, activation='sigmoid')) |
| --- |

### Training Process:

### **Compilation of the Model:**

### Optimizer: Adam.

### Loss function: Binary Crossentropy.

### Metrics: Accuracy.

| model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy']) |
| --- |

**Tokenization and Preprocessing:**

* Tokenization of texts using Tokenizer and fitting the model to the training set.

| tokenizer = Tokenizer(num\_words=max\_words)  tokenizer.fit\_on\_texts(X\_train) |
| --- |

**Sequences and Padding:**

* Transformation of texts into sequences and subsequent padding to equalize length.

| X\_train\_seq = tokenizer.texts\_to\_sequences(X\_train)  X\_train\_pad = pad\_sequences(X\_train\_seq, maxlen=max\_len) |
| --- |

**Model Training:**

* Training the model on the preprocessed data with a 20% validation split.
* 25 epochs and a batch size of 128.

| history = model.fit(X\_train\_pad, y\_train\_encoded, validation\_split=0.2, epochs=25, batch\_size=128) |
| --- |

**Predictions and Evaluation:**

* Utilizing the trained model to make predictions on the test set.
* Calculating performance metrics, including accuracy, precision, recall, and F1-score.

| y\_pred = model.predict(X\_test\_pad)  y\_pred\_binary = (y\_pred > 0.5).astype(int) |
| --- |

### Considerations: The RNN model has the assumption of reasonably improving the performance in sentiment analysis compared with the Dummy classifier. The architecture, comprising embedding, recurrent, and dense layers, allows the model to capture sequential dependencies. The dropout layer helps prevent overfitting. The training process involves tokenization, padding, and fitting the model to the training data. The achieved results provide a baseline for comparison with more advanced models, and further hyperparameter tuning or the exploration of more sophisticated architectures, such as LSTM, may lead to improved performance.

### **LSTM Sentiment analysis model:** Model Description:

The Long Short-Term Memory (LSTM) model is employed for sentiment analysis on preprocessed text data. LSTMs are a type of recurrent neural network (RNN) designed to capture long-term dependencies in sequential data, making them well-suited for tasks involving sequences of words or sentences.

### Model Architecture:

**Embedding Layer:**

* Converts words into dense vectors of fixed size.
* Vocabulary size: max\_words (10,000 words).
* Embedding dimension: 128.
* Input sequence length: max\_len (100 words).

| model\_lstm.add(Embedding(max\_words, 128, input\_length=max\_len)) |
| --- |

**LSTM Layer:**

* A type of recurrent layer that is capable of capturing long-term dependencies.
* 64 LSTM units are used to learn patterns in sequential data.

| model\_lstm.add(LSTM(64)) |
| --- |

**Dropout Layer:**

* Introduces regularization by randomly dropping connections during training.
* Dropout rate: 0.5.

| model\_lstm.add(Dropout(0.5)) |
| --- |

**Dense Layer:**

* Output layer with one unit and a sigmoid activation function for binary classification.

| model\_lstm.add(Dense(1, activation='sigmoid')) |
| --- |

### Training Process:

### **Compilation of the Model:**

### Optimizer: Adam.

### Loss function: Binary Crossentropy.

### Metrics: Accuracy.

| model\_lstm.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy']) |
| --- |

**Tokenization and Preprocessing:**

* Tokenization of texts using Tokenizer and fitting the model to the training set.

| tokenizer = Tokenizer(num\_words=max\_words)  tokenizer.fit\_on\_texts(X\_train) |
| --- |

**Sequences and Padding:**

* Transformation of texts into sequences and subsequent padding to equalize length.

| X\_train\_seq = tokenizer.texts\_to\_sequences(X\_train)  X\_train\_pad = pad\_sequences(X\_train\_seq, maxlen=max\_len) |
| --- |

**Model Training:**

* Training the model on the preprocessed data with a 20% validation split.
* 25 epochs and a batch size of 128.

| history = model\_lstm.fit(X\_train\_pad, y\_train\_encoded, validation\_split=0.2, epochs=25, batch\_size=128) |
| --- |

**Predictions and Evaluation:**

* Utilizing the trained model to make predictions on the test set.
* Calculating performance metrics, including accuracy, precision, recall, F1-score, and Cohen's Kappa.

| y\_pred = model\_lstm.predict(X\_test\_pad)  y\_pred\_binary = (y\_pred > 0.5).astype(int) |
| --- |

### Considerations: The LSTM model has the assumption of reasonably improving the performance in sentiment analysis compared with the simple RNN model. The architecture, with the addition of LSTM layers, allows the model to capture long-term dependencies more effectively. The dropout layer helps prevent overfitting. The training process involves tokenization, padding, and fitting the model to the training data. The achieved results provide evidence that the LSTM model outperforms the basic RNN for sentiment analysis in this context.

## **Performance evaluation**

Each model was run a total of 100 times and the average of each metric per model was taken. This was done in order to obtain more accurate results of the effectiveness of the models and to better demonstrate the improvement in the results with more complex models.

Before showing the results, discuss the relevance and significance of each metric in the context of binary classification models such as this sentiment analysis. The resulting metrics have the following meaning at the outcome level:

**Accuracy:**

* Represents the proportion of instances correctly classified out of the total instances.
* In this case, 77.45% of predictions are correct.

**Precision:**

* Indicates the proportion of true positive predictions out of the total instances predicted as positive.
* A value of 77.74% means that 77.74% of instances predicted as positive are truly positive.

**Recall:**

* Represents the proportion of true positive predictions out of the total actual positive instances.
* A value of 77.17% indicates that 77.17% of actual positive instances are captured.

**F1-score:**

* A metric that combines precision and recall into a single value, useful when there is an imbalance between classes.
* A value of 77.45% suggests a good balance between precision and recall.

**Kappa-score:**

* Measures the agreement between model predictions and actual classes, adjusting for chance agreement.
* A value of 54.91% indicates a moderate level of agreement beyond chance.

**Dummy Classifier Performance:**

Accuracy: 0.501818

Precision: 0.501818

Recall: 1.0

F1-score: 0.668281

**Analysis:**  
**Accuracy:** 50.18% of predictions are correct.

**Precision:** When predicting positive, it's correct 50.18% of the time.

**Recall:** Successfully captures all actual positive instances (100%).

**F1-score:** A balanced measure considering precision and recall, resulting in 66.83%.

This set of metrics indicates that the model has a high recall, capturing all actual positive instances, but the precision is relatively low, leading to a moderate F1-score. The overall accuracy is 50.18%, suggesting a limited overall predictive performance.

**Recurrent Neural Network Performance:**

Accuracy: 0.725545

Precision: 0.711701

Recall: 0.762065

F1-score: 0.735756

**Analysis:**  
**Accuracy:** 74.91% of predictions are correct.

**Precision:** When predicting positive, it's correct 75% of the time.

**Recall:** Successfully captures 75% of actual positive instances.

**F1-score:** A balanced measure considering precision and recall, resulting in 75%.

This set of metrics suggests a well-balanced performance, with equal values for precision, recall, and F1-score. The Kappa index further indicates a moderate agreement beyond what would be expected by chance. Overall, the model demonstrates a relatively robust predictive performance.

**RNN With Best Params Performance:**

Accuracy: 0.727855

Precision: 0.7279

Recall: 0.746775

F1-score: 0.729427

Kappa Cohen: 0.455644

**Analysis:**  
**Accuracy:** 74.91% of predictions are correct.

**Precision:** When predicting positive, it's correct 75% of the time.

**Recall:** Successfully captures 75% of actual positive instances.

**F1-score:** A balanced measure considering precision and recall, resulting in 75%.

**LSTM Performance:**

Accuracy: 0.762927

Precision: 0.75935

Recall: 0.772971

F1-score: 0.765866

Kappa-score: 0.525816

**Analysis:**  
**Accuracy:** 77.45% of predictions are correct.

**Precision:** When predicting positive, it's correct 77.74% of the time.

**Recall:** Successfully captures 77.17% of actual positive instances.

**F1-score:** A balanced measure considering precision and recall, resulting in 77.45%.

**Kappa-score:** Indicates a moderate level of agreement beyond chance (54.91%).

These metrics collectively suggest a well-balanced performance, with a high accuracy and a good balance between precision and recall. The Kappa score provides an additional measure of agreement adjusted for chance, indicating a moderate level of concordance.

**LSTM With Best Params Performance:**

Accuracy: 0.762909

Precision: 0.761456

Recall: 0.768949

F1-score: 0.764921

Kappa-score: 0.525794

**Analysis:**  
**Accuracy:** 77.45% of predictions are correct.

**Precision:** When predicting positive, it's correct 77.74% of the time.

**Recall:** Successfully captures 77.17% of actual positive instances.

**F1-score:** A balanced measure considering precision and recall, resulting in 77.45%.

**Kappa-score:** Indicates a moderate level of agreement beyond chance (54.91%).

## **Comparative analysis**

As we mentioned earlier, we decided to run their training, tuning, and testing processes 100 times, thus obtaining an average of their metrics. This approach allows us to mitigate the potential impact of variability in the training process and provides a more robust evaluation of the models' capabilities. By repeating the experiments multiple times, we aim to capture the inherent variability in model performance and ensure a comprehensive assessment of their generalization ability. This iterative approach contributes to a more reliable understanding of how well the models are likely to perform on unseen data and enhances the overall validity of our evaluation.

| Model | Accuracy | Precision | Recall | F1-score | Kappa-score |
| --- | --- | --- | --- | --- | --- |
| Dummy Classifier | 0.50 | 0.50 | 1 | 0.66 | - |
| RNN | 0.73 | 0.71 | 0.76 | 0.73 | 0.45 |
| LSTM | 0.76 | 0.76 | 0.77 | 0.76 | 0.52 |

In this comparative analysis, we examine the performance of three models: the Dummy Classifier, a Recurrent Neural Network (RNN), and a Long Short-Term Memory Neural Network (LSTM). The Dummy Classifier, by simply predicting the majority class, shows low accuracy and precision, although its recall is perfect. The RNN exhibits balanced performance with consistent precision, recall, and F1-score, reflecting a robust model. On the other hand, the LSTM further improves with slightly higher precision, a significant Kappa score, and overall enhanced performance. In summary, the LSTM stands out as the most effective model in this comparison, surpassing both the Dummy Classifier and the RNN in terms of precision and predictive capability.

The improvement in the performance of the Long Short-Term Memory Neural Network (LSTM) compared to the Dummy Classifier and the Recurrent Neural Network (RNN) can be attributed to the unique ability of LSTMs to retain information over long periods and effectively handle temporal dependencies in sequential data. While the Dummy Classifier simply makes predictions based on class frequencies, and the RNN may struggle to capture long-term dependencies due to vanishing or exploding gradient issues, LSTMs can maintain and utilize relevant information across extensive sequences, enabling them to learn more complex patterns and enhance classification accuracy.

## **Conclusion**

In conclusion, from this comparative analysis of models, it is evident that the Long Short-Term Memory Neural Network (LSTM) significantly outperforms the Dummy Classifier and the Recurrent Neural Network (RNN) in terms of classification performance. The LSTM demonstrates improved precision, a significant Kappa score, and overall stronger performance. This is attributed to the unique capability of LSTMs to handle long-term temporal dependencies in sequential data, enabling them to capture more complex patterns and enhance predictive capability. In contrast, the Dummy Classifier, relying solely on class frequencies, exhibits limited performance, while the RNN, although balanced, falls short of matching the LSTM's ability to retain long-term information. In summary, the choice of the LSTM is justified by its enhanced capacity to address inherent complexities in sequential data and its ability to provide more accurate results compared to the other evaluated models.