DEMOKRITOS
NATIONAL CENTER FOR SCIENTIFIC RESEARCH

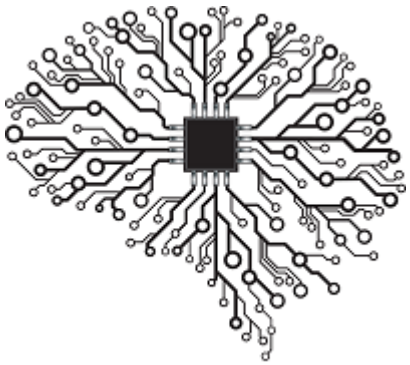
UNIVERSITY of the PELOPONNESE

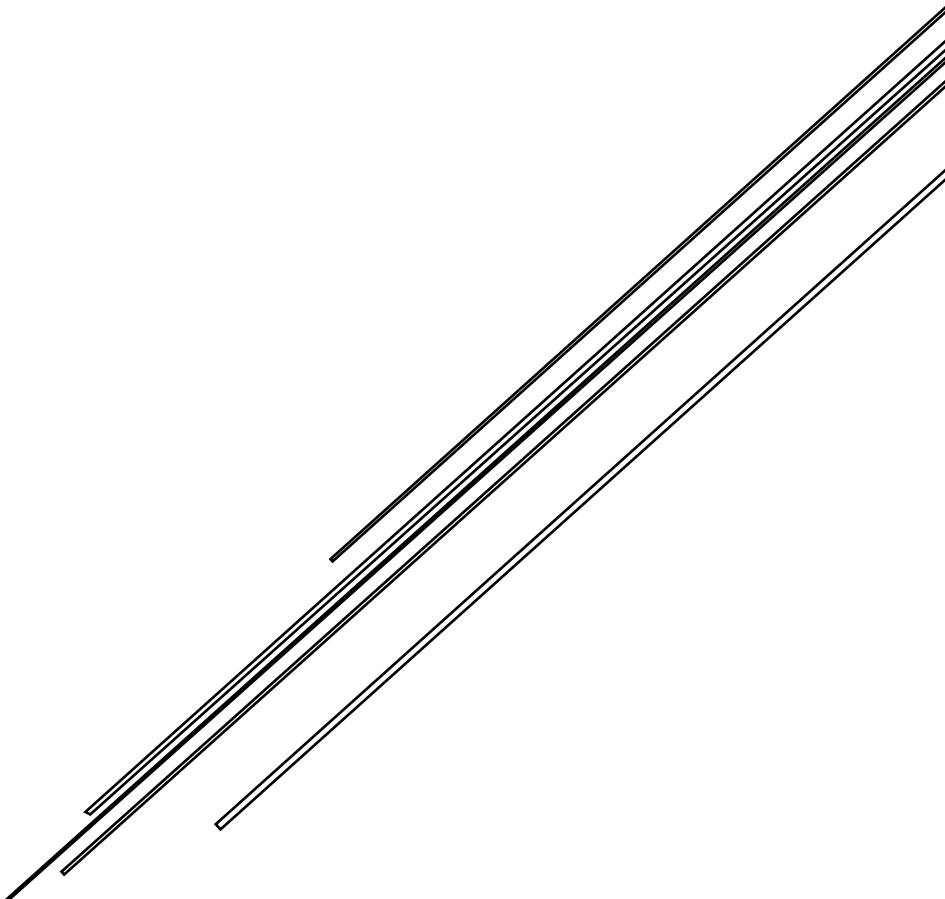
MSc in Data Science

# Deep Learning

## Text-based sentiment analysis

Sykallou Sofia - 2022202204009
Pouli Stavroula - 2022202204008





# Table of Contents

Introduction ..... 2
Data & Methodology ..... 2
Data ..... 2
Features ..... 3
Problem Statement ..... 3
Methodology ..... 4
Data Cleaning & Preprocessing ..... 5
Duplicate values ..... 5
Sentiment label transformation ..... 6
Stemming ..... 6
Transformations and removals ..... 7
Exploratory Analysis ..... 9
Top 20 most common words and bigrams ..... 9
Distribution of the lengths of the reviews ..... 9
Word Clouds for Positive and Negative Reviews ..... 10
Number of Characters and Words in Positive and Negative Reviews ..... 11
Split the dataset (train and test) ..... 11
Word embedding using Word2Vec model ..... 11
Tokenization and Padding ..... 13
Tokenization ..... 13
Padding ..... 13
Embedding matrix ..... 13
Model selection ..... 14
RNN (Recurrent Neural Network) ..... 14
LSTM (Long Short-Term Memory) ..... 16
Bidirectional LSTM ..... 19
Model evaluation ..... 23
Confusion matrix ..... 23
Classification report ..... 24
ROC Curves ..... 25
Final model selection ..... 26
Conclusion ..... 26

# Introduction

The objective of this assignment is to build a deep learning model that performs text-based sentiment analysis of movie reviews from IMDb.

Sentiment analysis, commonly referred to as opinion mining, is a branch of natural language processing (NLP) that focuses on figuring out whether a text has positive or negative sentiment.

Recurrent neural networks (RNNs), which have demonstrated to be particularly successful at collecting sequential dependencies and contextual information in text input, will be used in the proposed deep learning model.

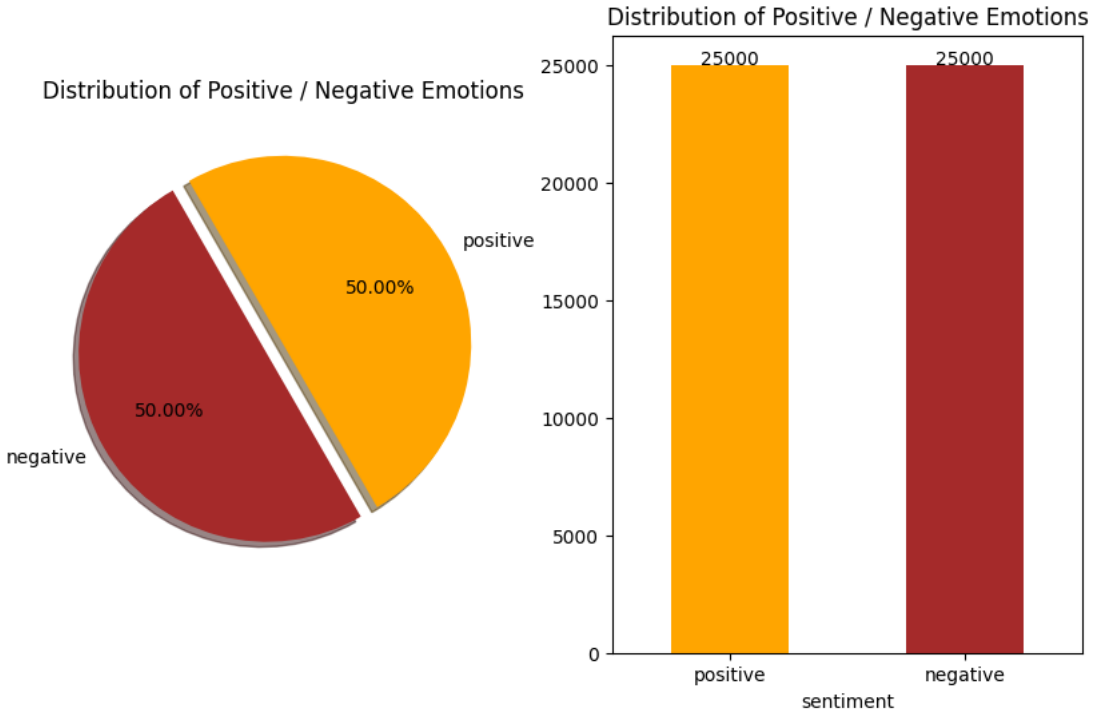
The model will develop the ability to classify reviews based on sentiment by training on the dataset of IMDb movie reviews.

# Data & Methodology

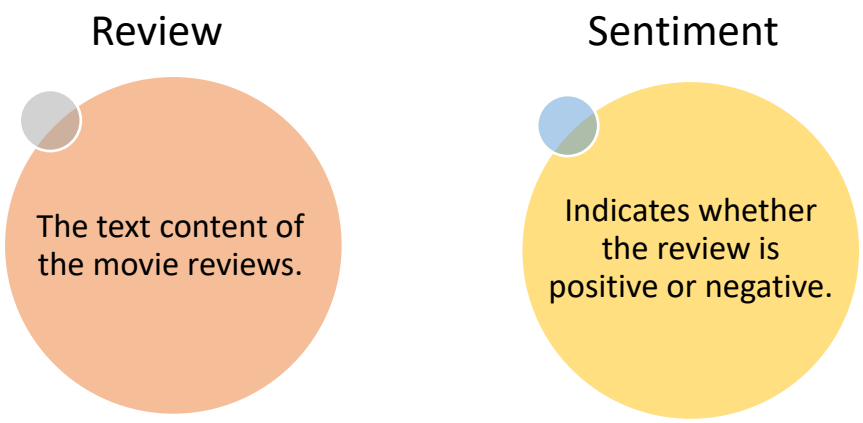
## Data

Dataset Source: [Kaggle]

The IMDb Dataset of 50K Movie Reviews is a popular dataset, commonly used for sentiment analysis and natural language processing. The dataset consists of 50.000 movie reviews, evenly divided into 25.000 positive and 25.000 negative labeled reviews (balanced dataset).



## Features




## Problem Statement

Utilize deep learning techniques to predict the sentiment (positive or negative) of movie reviews.

## Methodology

i. **Data Preprocessing**
   Perform preprocessing steps to clean and prepare the text data for analysis. Detailed steps can be found in the "Data_preprocessing_notebook.ipynb".

ii. **Padding**
   Pad the preprocessed text sequences to ensure they have a consistent length. This is necessary as deep learning models typically require fixed-length inputs.

iii. **Word Embedding**
   Convert the preprocessed text into a numerical representation using word embeddings, such as Word2Vec. Word embeddings map words to dense vectors in a continuous vector space, capturing semantic relationships between words.

iv. **Model Training**
   Split the dataset into training and test sets and train deep learning models using the training data.

v. **Model Selection**
   Choose an appropriate deep learning model architecture for sentiment analysis, such as recurrent neural networks (RNNs), long short-term memory (LSTM) networks, or Bidirectional models.
   Experiment with different architectures in a separate file ("Initial_model_selection.ipynb"), to identify the most effective ones and use them in the "Working_notebook.ipynb".

vi. **Model Evaluation**
   Assess the trained model's performance on the test set, considering metrics such as accuracy, precision, recall, F1 score, and ROC-AUC curves. These metrics provide insights into the model's ability to correctly classify movie reviews based on sentiment.

By following this methodology, we aim to build an accurate and robust deep learning model that can predict the sentiment of IMDb movie reviews.

# Data Cleaning & Preprocessing

The first step is to load and preview a few reviews to get a glimpse of the data.

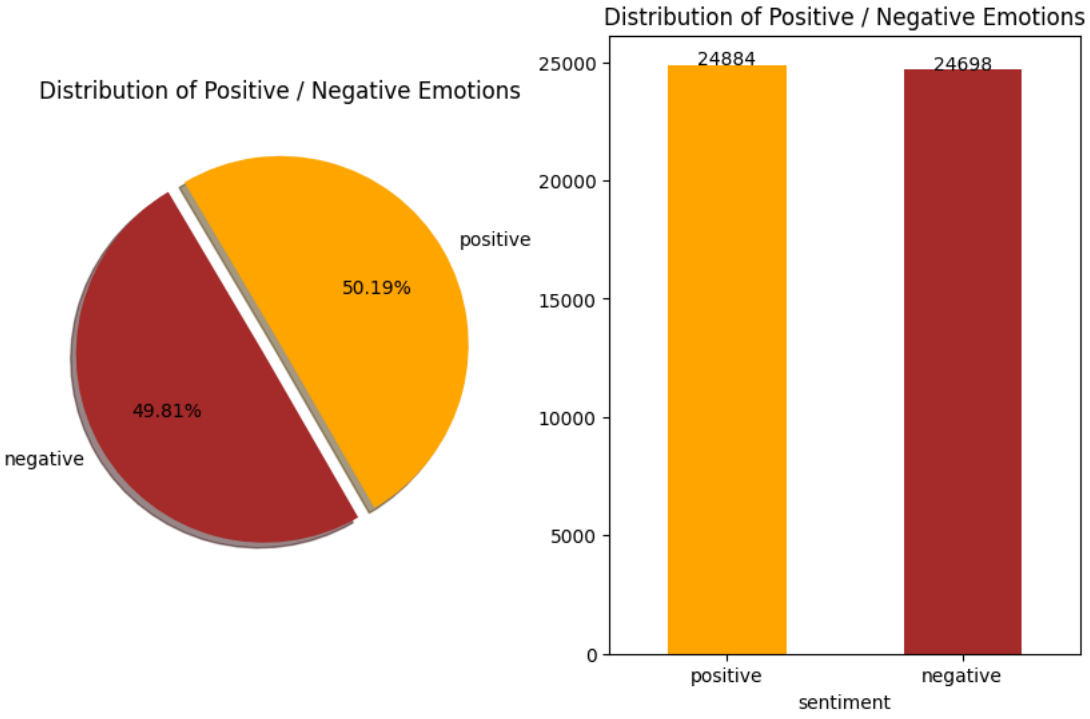
| | review | sentiment |
|---|---|---|
| 0 | One of the other reviewers has mentioned that ... | positive |
| 1 | A wonderful little production. <br /><br />The... | positive |
| 2 | I thought this was a wonderful way to spend ti... | positive |
| 3 | Basically there's a family where a little boy ... | negative |
| 4 | Petter Mattei's "Love in the Time of Money" is... | positive |

It is obvious that we must clean the data in order to be able to complete the sentiment analysis.

## Duplicate values

We searched for duplicate values, and we found that there are 418 duplicate values in our dataset. After deleting them, the dataset consisted of 49.582 instances.

After this step, we wanted to check if the balance would be affected by the deletion of the duplicate values. Based on the graph below, we can surely state that the dataset remained balanced.



## Sentiment label transformation

Another crucial step is to transform the feature **Sentiment**, since the labels must be defined as binary numbers.


After this transformation, the data looked like this:

## Stemming

Stemming is a process that reduces words to their base or root form, known as a stem. It involves removing suffixes or affixes from words to normalize them and group similar words together.

We experimented with three stemming algorithms such as Porter, Snowball, and Lancaster algorithms. The one with the highest performance in our data, was the Porter stemming algorithm.

## Transformations and removals

Firstly, we converted all the letter in lowercase and then we performed the following:

i. Definition of a word mapping dictionary, where the selected informal contractions or abbreviations are transformed to their corresponding expanded form.
ii. Removal of
    - Hyperlinks
    - HTML tags
    - Numbers
    - Emoticons and emojis
    - Symbols and pictographs
    - Transport and map symbols
    - Flags
    - Punctuations such as ['!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~]+
    - Whitespaces
iii. Separate the text into individual words
iv. Application of the Porter stemmer algorithm

Here is a review before applying the above tasks:

*One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened with me.<br /><br />The first thing that struck me about Oz was its brutality and unflinching scenes of violence, which set in right from the word GO. Trust me, this is not a show for the faint hearted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is hardcore, in the classic use of the word.<br /><br />It is called OZ as that is the nickname given to the Oswald Maximum Security State Penitentary. It focuses mainly on Emerald City, an experimental section of the prison where all the cells have glass fronts and face inwards, so privacy is not high on the agenda. Em City is home to many..Aryans, Muslims, gangstas, Latinos, Christians, Italians, Irish and more....so scuffles, death stares, dodgy dealings and shady agreements are never far away.<br /><br />I would say the main appeal of the show is due to the fact that it goes where other shows wouldn't dare. Forget pretty pictures painted for mainstream audiences, forget charm, forget romance...OZ doesn't mess around. The first episode I ever saw struck me as so nasty it was surreal, I couldn't say I was ready for it, but as I watched more, I developed a taste for Oz, and got accustomed to the high levels of graphic violence. Not just violence, but injustice (crooked guards who'll be sold out for a nickel, inmates who'll kill on order and get away with it, well mannered, middle class inmates being turned into prison bitches due to their lack of street skills or prison experience) Watching Oz, you may become comfortable with what is uncomfortable viewing....thats if you can get in touch with your darker side.*

After applying the above tasks, the reviews looked like this:

*one review mention watch oz episod hook right exactli happen first thing struck oz brutal unflinch scene violenc set right word go trust show faint heart timid show pull punch regard drug sex violenc hardcor classic use word call oz nicknam given oswald maximum secur state penitentari focus mainli emerald citi experiment section prison cell glass front face inward privaci high agenda em citi home mani aryan muslim gangsta latino christian italian irish scuffl death stare dodgi deal shadi agreement never far away would say main appeal show due fact goe show dare forget pretti pictur paint mainstream audienc forget charm forget romanc oz mess around first episod ever saw struck nasti surreal say readi watch develop tast oz got accustom high level graphic violenc violenc injustic crook guard sold nickel inmat kill order get away well manner middl class inmat turn prison bitch due lack street skill prison experi watch oz may becom comfort uncomfort view that get touch darker side*

**Note:** *We also tried to apply lemmatization, but the results did not help improve the overall performance of the sentiment analysis task. Despite the benefit of obtaining meaningful word forms, the impact on the final sentiment classification was not significant enough to outweigh the computational costs involved, so we decided to exclude it from our implementation.*

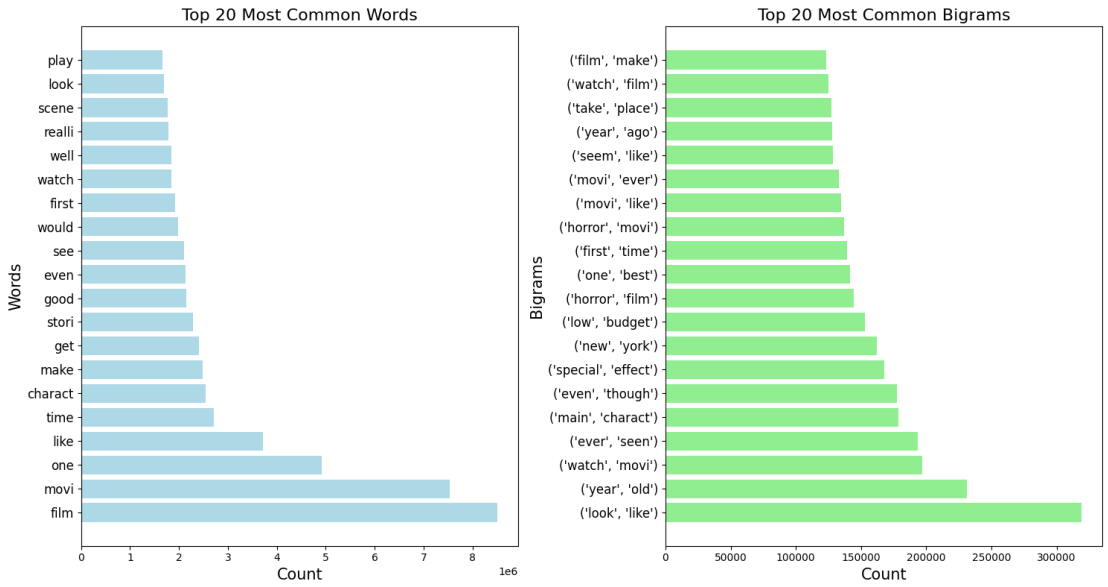
# Exploratory Analysis

## Top 20 most common words and bigrams

This visualization shows the top 20 most common words (left subplot) and bigrams (right subplot) in a collection of text data.

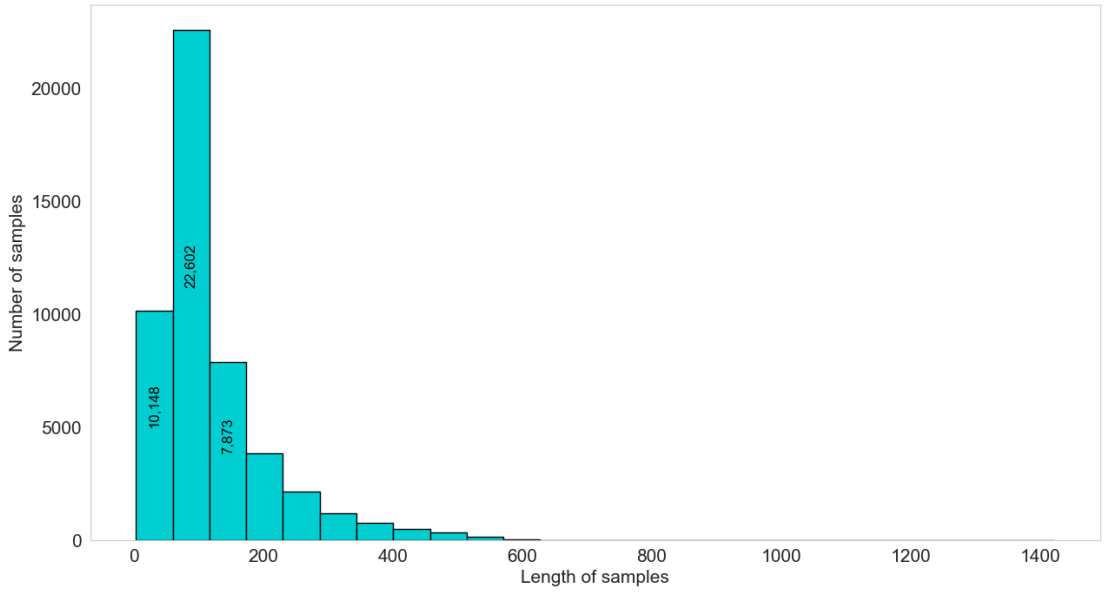
We used the Counter class to count the frequency of each word of our collection of text data.

As shown below, the most common word is “film” and the most common bigram is {‘look’,’like’}.




## Distribution of the lengths of the reviews

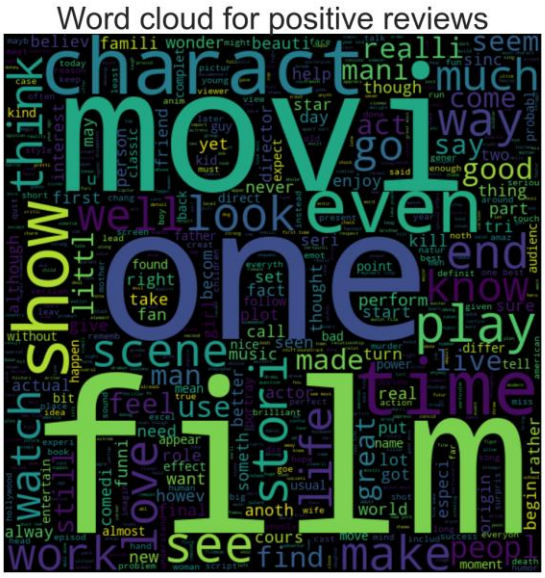
This histogram shows the distribution of the lengths of the reviews. It’s generated by using the “lengths” list as an input with 25 bins.



## Word Clouds for Positive and Negative Reviews

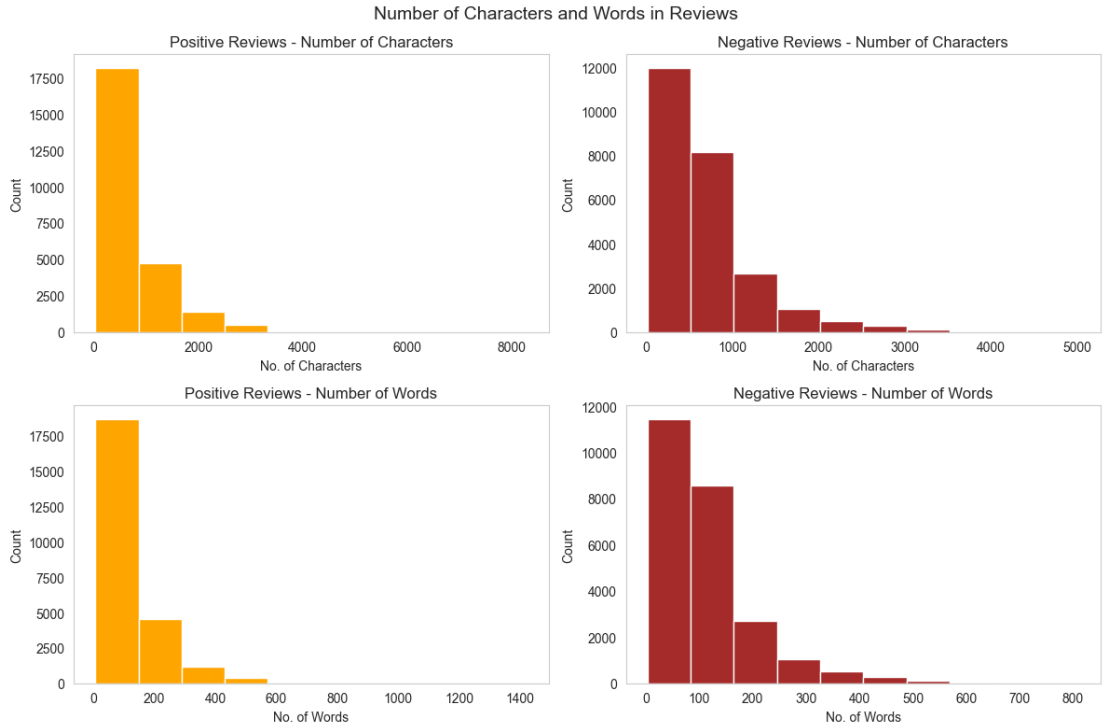
Another interesting visualizations, are word clouds. Below is a word cloud for the positive and negative reviews.

Mechanism: We extract the positive reviews from the 'review' field where the sentiment is 1 (positive). Then we perform the same logic for the negative reviews where the sentiment is 0 (negative).



## Number of Characters and Words in Positive and Negative Reviews

In the below histograms, we can see the number of characters and words in positive and negative reviews. We start by calculating the number of words in each positive review by splitting in into a list of words and the checking the length of that list. The same logic applies for the negative reviews.




After the data cleaning and preprocessing, among with the exploratory analysis phase, we saved the clean dataset in the 'imdb_clean_dataset.csv' file, to continue with our implementation.

# Split the dataset (train and test)

To be able to dive deep into the models, we must split the dataset into a training and a testing set. The training set will be the 80% (39665) of our data, while the testing set will be the 20% (9917) of our data.

# Word embedding using Word2Vec model

Word embeddings is a technique to represent words as dense vectors (used in NLP). We used Word2Vec, a tool to discover the semantic relationships between words.

The Word2Vec, will help us analyze the context of each word to generate word embeddings, and then, the model will learn the relationships between words based on their positions.

The process:

i. Create a list that will hold the training data for Word2Vec.
ii. Apply a lambda function to split each sentence into a list of words based on the whitespace.

After those steps, we will train the Word2Vec model on the Word2Vec training data we created. The Word2Vec training data contains the tokenized reviews from the training set. We decided to use 100 dimensions for the word embeddings.

Then, we define the model, by tuning its parameters to our desire.

Word2Vec model parameters:

i. Input: the Word2Vec training data.
ii. Vector size: the dimensions for the word embeddings (100).
iii. Workers: the worker threads that should be used for training (8).
iv. Min_count: the minimum allowed frequency threshold for a word to be in the vocabulary (5). In our case, every word that will occur less than 5 times, will be ignored.

After executing the above process, the vocabulary's length is equal to 24221 words.

# Tokenization and Padding

## Tokenization

In this part, we will perform text tokenization with the Tokenizer class from the Keras library.

At first, we parameterize the tokenizer:

i. create a tokenizer object where no characters will be filtered out during tokenization (filters="").
ii. the text will not be converted to lowercase (lower=False).
iii. whenever the model will have to encounter unseen words, it will use the oov (out of vocabulary) token "<oov>" (oov_token="<oov>")

Next, we fit the tokenizer on the input data to learn the vocabulary based on the texts.

Lastly, we limit the number of words considered during tokenization to 35.000. Note that in our code, we saved in the vocab_length to 35.000. This indicated the maximum number of words to keep based on their frequency.

## Padding

We apply padding, to ensure that all sequences have the same length. We decided to set the maximum length of the input sequences to 750.

Based on the tokenizer's vocabulary, we convert the texts in X_train/X_test into sequences of integers. These sequences are then padded using pad_sequences to have the maximum length (750).

For both X_train and X_test, we pad and then truncate the sequences.

# Embedding matrix

For each word in the tokenizer's vocabulary, we store the embedding vectors in a matrix.

If the Word2Vec model contains the word, then we retrieve the embedding vector for the word from the Word2Vec model.

The embedding matrix should look like this:

```
Embedding Matrix Shape: (35000, 100)

array([[ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.25320712, -0.32531893, -2.18189287, ...,  0.63631606,
         0.05103426, -0.31895107],
       ...,
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ]])
```

# Model selection

Based on the type of our problem, we decided to use 3 models, the RNN, the LSTM and the Bidirectional LSTM model.

## RNN (Recurrent Neural Network)

Recurrent Neural Networks process sequential data, such as our reviews.
They can handle dependencies and capture temporal information in the data. Due to recurrency, RNNs pass information from previous steps to the current one.

We create our first model by creating an embedding layer that maps the input words to dense vectors.

Its parameters are:

i. `input dim`: The size of the vocabulary.
ii. `output_dim`: The dimensionality of the word embeddings.
iii. `weights`: The initial weights for the embedding layer.
iv. `input_length`: The length of the input sequences.

Now, it's time to tune our model that consists of many layers.

At first, there is the embedding layer that we discussed above.

Then a SimpleRNN layer with 100 units for the sequential input data. This will return an output at the final time step.

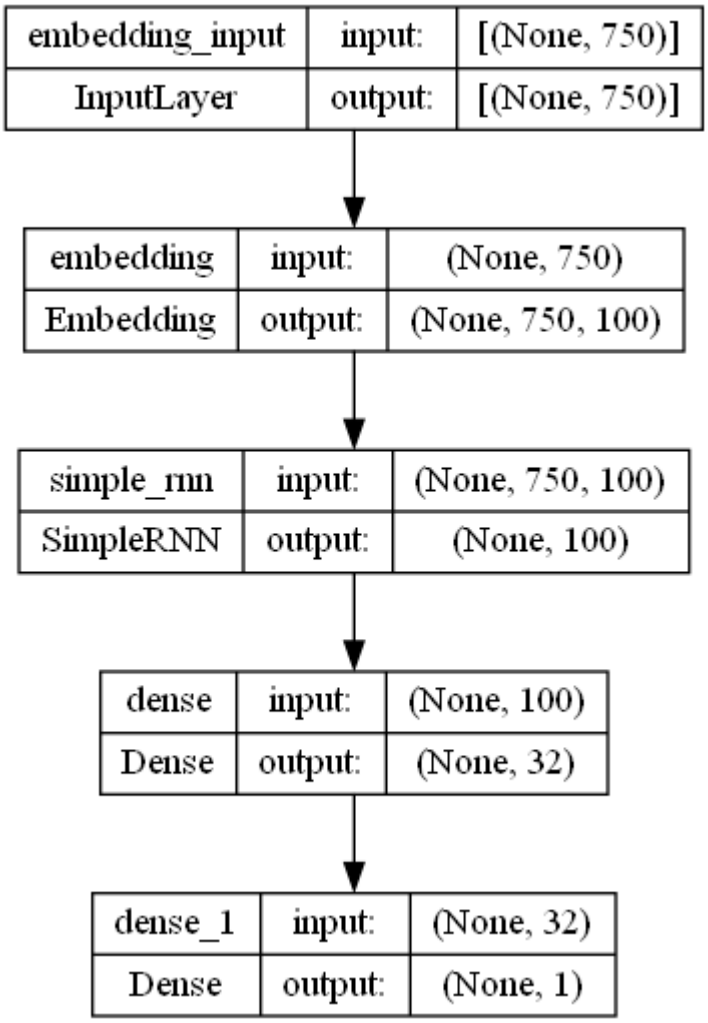
Moving on, we need a dense layer with 32 units and an activation function called "relu".

Finally, we used a dense layer with a single unit and the activation function called "sigmoid". This layer maps the output of the previous layrer to a probability value between 0 and 1, which will help us with the prediction.

A summary of the above is presented in this array:

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 750, 100)          3500000

 simple_rnn (SimpleRNN)      (None, 100)               20100

 dense (Dense)               (None, 32)                3232

 dense_1 (Dense)             (None, 1)                 33

=================================================================
Total params: 3,523,365
Trainable params: 23,365
Non-trainable params: 3,500,000
_________________________________________________________________
```

With the use of the plot_model function, we can represent our training model:




The model is ready to be trained. We need to set a specified loss function, an optimizer and at least one evaluation metric.

Loss function: We chose 'binary_crossentropy'`, to measure the difference between the predicted probabilities and the true labels.

Optimizer: The “adam” optimizer uses adaptive learning rates and momentum to efficiently update the model weights and improve convergence.

Evaluation metrics: “accuracy” calculates the proportion of correct predictions over the total number of samples, providing an indication of the model's accuracy.

We trained the model and when evaluating it, we found out that its loss is about 35% and its accuracy is about 84%

```
Test loss: 0.3503372371196747
Test accuracy: 0.8472320437431335
```

## LSTM (Long Short-Term Memory)

LSTM is a type of RNN that addresses the vanishing gradient problem. LSTMs selectively retain or discard information over time with memory cells and can capture long-term dependencies and handle sequences with varying time intervals effectively.

We create our second model by once again creating an embedding layer that maps the input words to dense vectors.

Its parameters are:

i. `input dim`: The size of the vocabulary.
ii. `output_dim`: The dimensionality of the word embeddings.
iii. `weights`: The initial weights for the embedding layer.
iv. `input_length`: The length of the input sequences.
v. Trainable=False

With the help of the Keras library we build our sequential model.

Our model has the following layers:

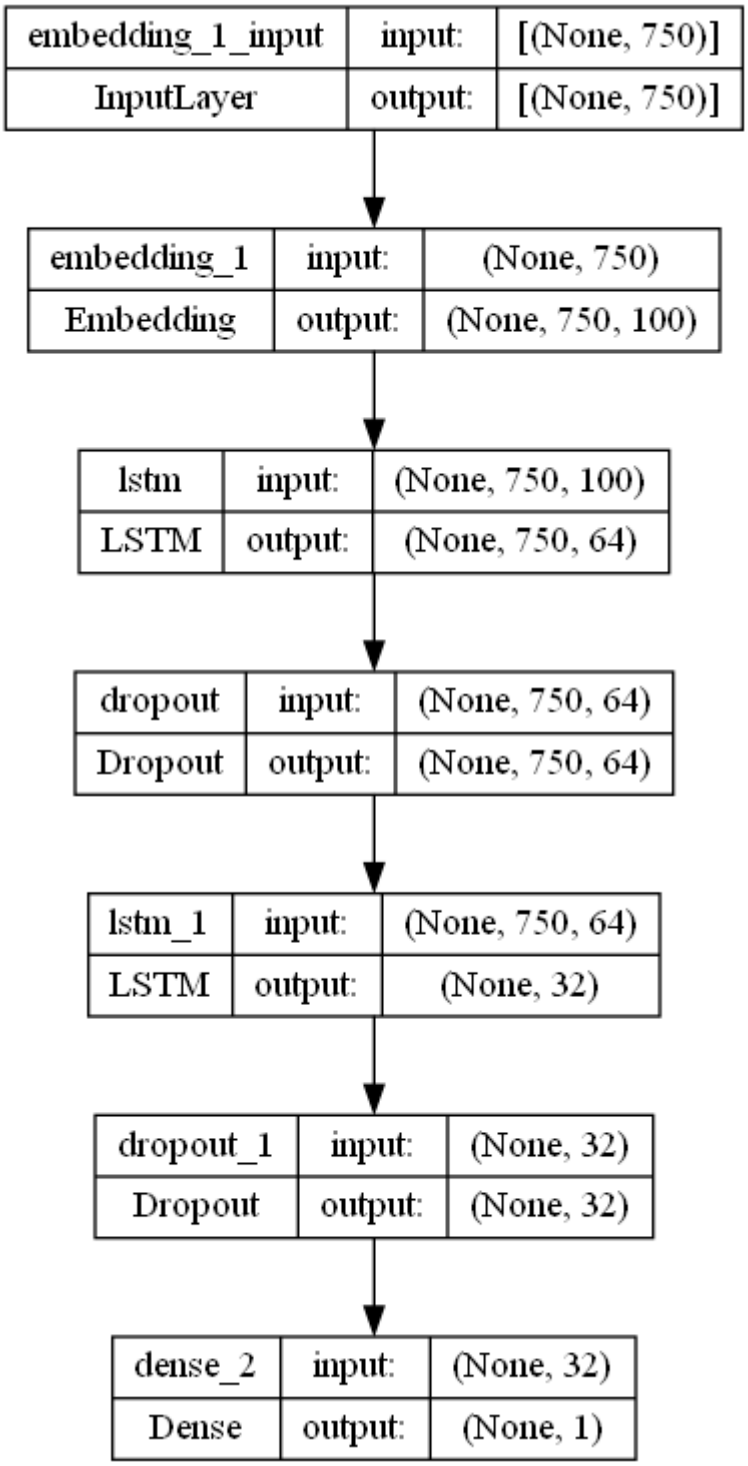
i. Embedding layer: To map the input words to dense vectors
ii. LSTM layer: with 64 units, to get the sequence of hidden states for each step
iii. Dropout layer: A dropout layer with a (dropout) rate of 0.2, to prevent overfitting. We manage to prevent overfitting because we randomly set a fraction of the input units to 0 during training.
iv. LSTM layer: Another LSTM layer, but this time with 32 units, but this one does not return sequences.
v. Dropout layer: Another Dropout layer with a (dropout) rate of 0.2.
vi. Dense layer: A layer with a single unit and a sigmoid activation function.

*The final layer helps us with the prediction, as it maps the output of the previous layer to a probability value between 0 and 1.*

A summary of the above is presented in this array:

```
Model: "Sentiment_Model_LSTM"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 750, 100)          3500000

 lstm (LSTM)                 (None, 750, 64)           42240

 dropout (Dropout)           (None, 750, 64)           0

 lstm_1 (LSTM)               (None, 32)                12416

 dropout_1 (Dropout)         (None, 32)                0

 dense_2 (Dense)             (None, 1)                 33

=================================================================
Total params: 3,554,689
Trainable params: 54,689
Non-trainable params: 3,500,000
_________________________________________________________________
```

With the use of the plot_model function, we can represent our training model:



The model is ready to be trained. We need to set a specified loss function, an optimizer and at least one evaluation metric.

Loss function: We chose 'binary_crossentropy'`, to measure the difference between the predicted probabilities and the true labels.

Optimizer: The “adam” optimizer uses adaptive learning rates and momentum to efficiently update the model weights and improve convergence.

Evaluation metrics: “accuracy” calculates the proportion of correct predictions over the total number of samples, providing an indication of the model's accuracy.

We trained the model and when evaluating it, we found out that its loss is about 30% and its accuracy is about 87%

```
Test loss: 0.308144748210907
Test accuracy: 0.8748613595962524
```

## Bidirectional LSTM

Our third and final model is the Bidirectional LSTM. Bidirectional LSTM is an extension of the LSTM model that processes the input sequence in both forward and backward directions. It combines the information from past and future contexts, allowing the model to capture all the dependencies.

We create an embedding layer that maps the input words to dense vectors.

Its parameters are:

i. `input dim`: The size of the vocabulary.
ii. `output_dim`: The dimensionality of the word embeddings.
iii. `weights`: The initial weights for the embedding layer.
iv. `input_length`: The length of the input sequences.
v. Trainable=False

With the help of the Keras library we build our sequential model.

Our model has the following layers:

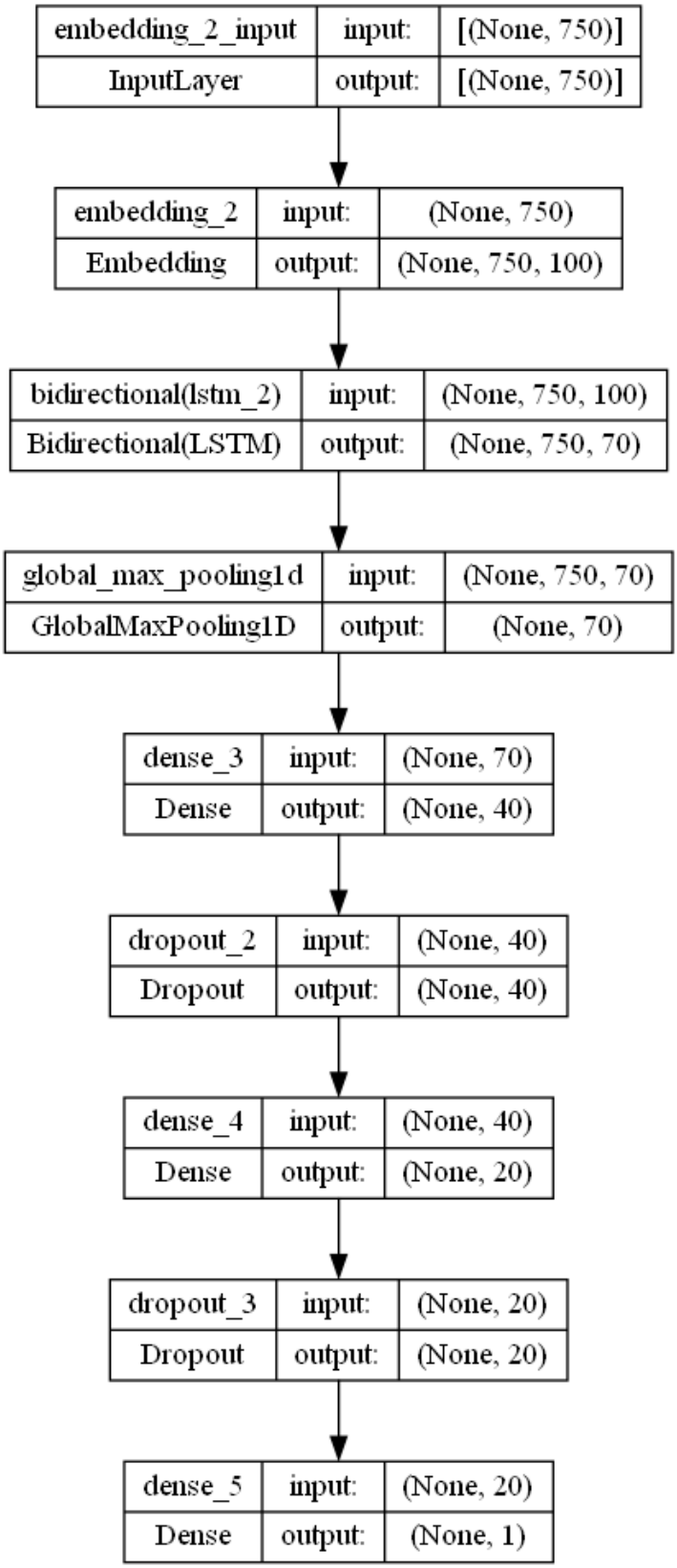
i. Embedding layer: To map the input words to dense vectors
ii. Bidirectional LSTM layer: A layer with 35 units, to get the sequence of hidden states for each step
iii. Global Max Pooling 1D layer: A layer to perform max pooling operation in the dimension of the sequence.
iv. Dense layer: A layer with 40 units and a relu activation function. This layer will help us with the linear transformation of the data.
v. Dropout layer: A dropout layer with a (dropout) rate of 0.5, to prevent overfitting. We manage to prevent overfitting because we randomly set a fraction of the input units to 0 during training.
vi. Dense layer: A layer with 20 units and a relu activation function.
vii. Dropout layer: A dropout layer with a (dropout) rate of 0.5
*viii.* Dense layer: The final layer, with a single unit and a sigmoid activation function.

*The final layer helps us with the prediction, as it maps the output of the previous layer to a probability value between 0 and 1.*

A summary of the above is presented in this array:

```
Model: "Sentiment_Model_LSTM_Bidirectional"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, 750, 100)          3500000

 bidirectional (Bidirectiona  (None, 750, 70)          38080
 l)

 global_max_pooling1d (Globa  (None, 70)               0
 lMaxPooling1D)

 dense_3 (Dense)             (None, 40)                2840

 dropout_2 (Dropout)         (None, 40)                0

 dense_4 (Dense)             (None, 20)                820

 dropout_3 (Dropout)         (None, 20)                0

 dense_5 (Dense)             (None, 1)                 21

=================================================================
Total params: 3,541,761
Trainable params: 41,761
Non-trainable params: 3,500,000
_________________________________________________________________
```

With the use of the plot_model function, we can represent our training model:



The model is ready to be trained. We need to set a specified loss function, an optimizer and at least one evaluation metric.

Loss function: We chose 'binary_crossentropy'`, to measure the difference between the predicted probabilities and the true labels.

Optimizer: The “adam” optimizer uses adaptive learning rates and momentum to efficiently update the model weights and improve convergence.

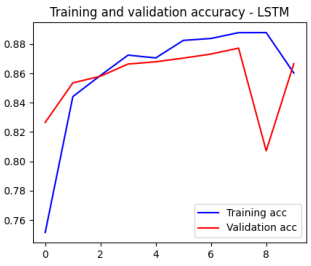
Evaluation metrics: “accuracy” calculates the proportion of correct predictions over the total number of samples, providing an indication of the model's accuracy.

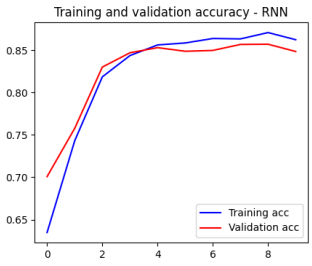
We trained the model and when evaluating it, we found out that its loss is about 28% and its accuracy is about 88%

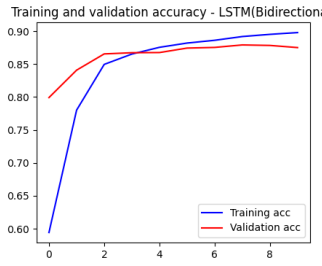
```
Test loss: 0.2860482335090637
Test accuracy: 0.8803065419197083
```

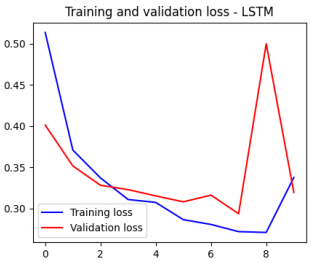
# Model evaluation

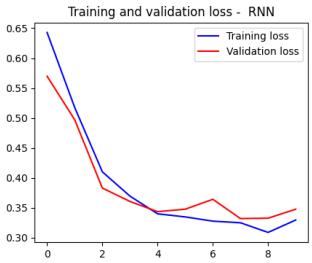
Now that we are done with training and validating of our models, we can use a plot to observe the accuracy and loss for each model (for both training and validation):

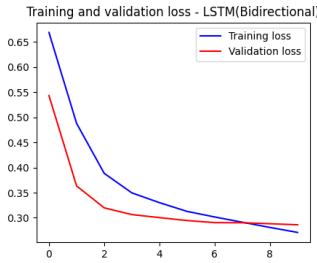


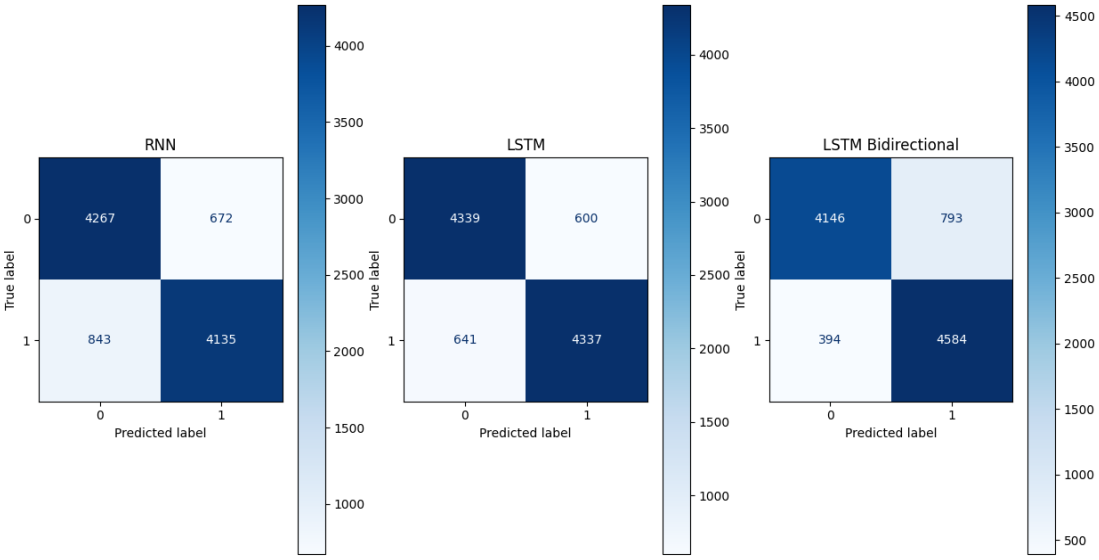



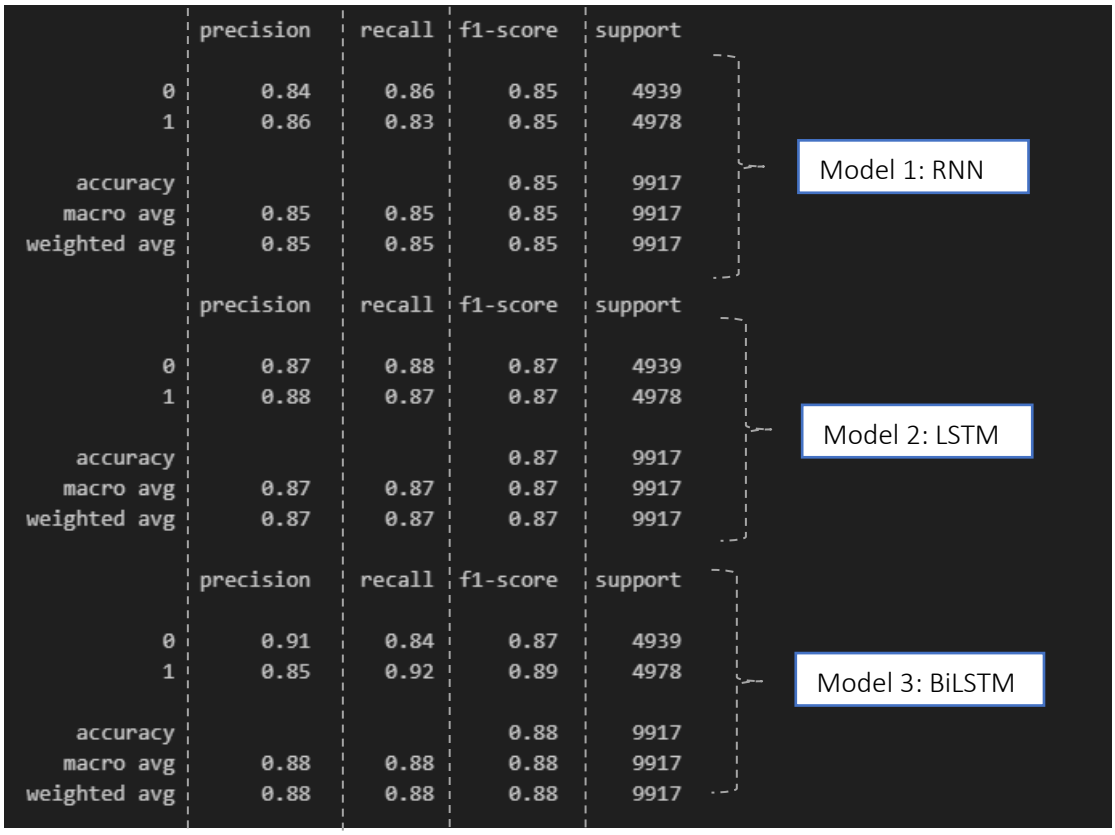

## Confusion matrix

Another very useful tool, is the confusion matrix:



## Classification report

The following array is generated with the help of the classification report. It shows all the evaluation metrics for our (classification) problem. It works by using the true labels (y_test) and the predicted labels (y_pred) and using them to create a report with the precision, recall, f1-score and support.



| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.84 | 0.86 | 0.85 | 4939 |
| 1 | 0.86 | 0.83 | 0.85 | 4978 |
| accuracy | | | 0.85 | 9917 |
| macro avg | 0.85 | 0.85 | 0.85 | 9917 |
| weighted avg | 0.85 | 0.85 | 0.85 | 9917 |

Model 1: RNN

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.87 | 0.88 | 0.87 | 4939 |
| 1 | 0.88 | 0.87 | 0.87 | 4978 |
| accuracy | | | 0.87 | 9917 |
| macro avg | 0.87 | 0.87 | 0.87 | 9917 |
| weighted avg | 0.87 | 0.87 | 0.87 | 9917 |

Model 2: LSTM

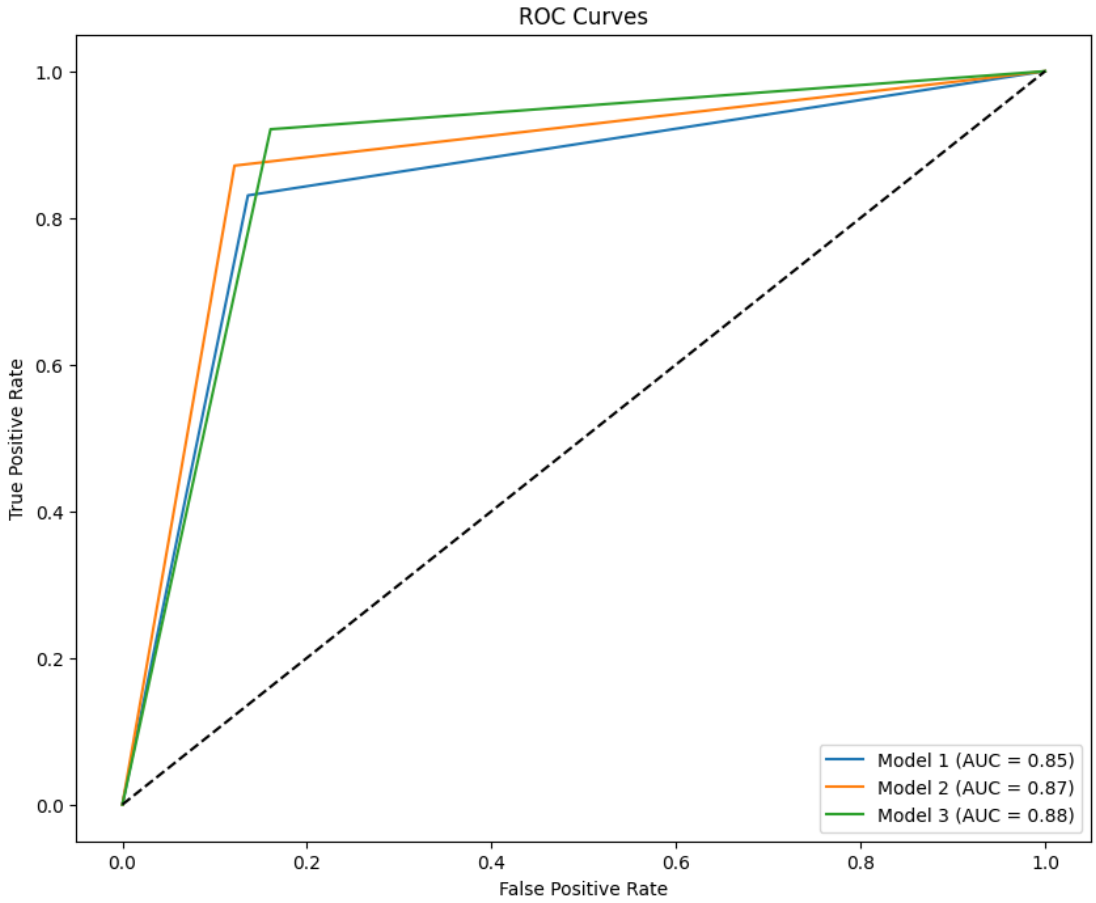
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.91 | 0.84 | 0.87 | 4939 |
| 1 | 0.85 | 0.92 | 0.89 | 4978 |
| accuracy | | | 0.88 | 9917 |
| macro avg | 0.88 | 0.88 | 0.88 | 9917 |
| weighted avg | 0.88 | 0.88 | 0.88 | 9917 |

Model 3: BiLSTM

## ROC Curves

The ROC (Receiver Operating Characteristic) curve is a representation of the performance of a binary classification model. We start by computing the false positive rate (FPR), the true positive rate (TPR) and thresholds for each model.

The AUC (Area Under the Curve) of the ROC curve is a value that represents the overall performance of the classification model. An AUC of 0.5 represents a random classifier, while an AUC of 1.0 represents a perfect classifier.



## Final model selection

Based on our observations, we conclude that the best model for our problem is the Bidirectional LSTM model, since it was the model with the highest accuracy on the test set, in comparison to the other two models.

```
310/310 [==============================] - 15s 48ms/step - loss: 0.3503 - accuracy: 0.8472
310/310 [==============================] - 53s 171ms/step - loss: 0.3081 - accuracy: 0.8749
310/310 [==============================] - 29s 93ms/step - loss: 0.2860 - accuracy: 0.8803
The best model is model 3 (LSTM Bidirectional).
```

## Conclusion

In this project were called to perform text based sentiment analysis on an IMDb dataset.

After data cleaning, data preprocessing and an exploratory analysis, we were able to train and test our models.

The models we used were:

i. RNN (Recurrent Neural Networks)
ii. LSTM (Long Short-Term Memory)
iii. BiLSTM (Bidirectional Long Short-Term Memory)

In the model evaluation phase, the “winner” model was the BiLSTM model. It scored a loss around 28% and an accuracy around 88%.

In general, when having to deal with problems such as text based sentiment analysis the BiLSTM model seems to be the most robust. This is because the BiLSTM model process the input sequence both forward and backward. As a result, the model can better interpret the overall sentiment of the text by capturing contextual information from both past and future situations.