**PROJECT REPORT**

**COLLAGE NAME: GRT INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**COURSE NAME:** ARTIFICILAL INTELLIGENCE.

**PROJECT NAME:** SENTIMENT ANALYSIS FOR MARKETING

**Submitted By: KEERTHANA .G**

**Mail Id:** gkeerthana793@gmail.com

TABLE OF CONTENT

|  |  |  |
| --- | --- | --- |
| SI.NO | TITLE | PG.NO |
| 01. | AIM AND ABSTRACT | 2 |
| 02. | INTRODUCTION | 3 |
| 03. | METHODS THAT ARE IMPLEMENTED | 4 |
| 04. | INSIGHTS AND IMPLICATION | 5 |
| 05. | DESIGN | 6 |
| 06. | DATA PREPROCESSING | 9 |
| 07. | NLP TECHNIQUE | 19 |
| 08. | RESULT | 33 |
| 09. | FUTURE SCOPE | 37 |
| 10. | CONCLUSION | 38 |

**AIM:**

* **Identify and analyze the sentiment of marketing messages in IMDb movie reviews.**
* **Understand how marketing messages are perceived by different audiences.**
* **Evaluate the effectiveness of marketing campaigns.**
* **Improve the targeting and personalization of marketing messages.**

**ABSTRACT:**

Sentiment Analysis of IMDB Movie Reviews by the use of Natural Language Processing Techniques and analyzing the performance of various Machine Learning Algorithms on movie reviews. After converting unstructured data into structured data for the ease of analysis. Movie Reviews are classified into binary categories i.e. Positive reviews or Negative reviews on the basis of words used in the reviews. Machine Learning classifiers are used to categorize these reviews to its maximum accuracy and comparing the performance of the classifiers with each other on the same dataset.

**Keywords:**

Sentiment Analysis, Movie, Reviews, Machine Learning, NLP

**INTRODUCTION:**

The inflow of unstructured data is continually expanding. It must be classified in order to provide significant information. Sentiment analysis can be utilized in a variety of sectors, including market product performance analysis, training chatter bots with certain sentiments to respond, content ratings for various blogs, posts, and videos, and tale summary. Sentiment Analysis is also employed in several search engine page ranking systems. IMDB A dataset of negative and favorable movie reviews is taken. Each critical and positive review has a thousand reviews in the dataset. These unstructured reviews are vectorized and translated into structured data. These negative and positive review vectors teach the algorithm to categorize test data reviews as positive or negative.

This research explores the complex terrain of customer attitudes, opinions, and preferences, building on the four phases in which we have previously demonstrated how to leverage advanced natural language processing techniques. Through creative approaches, data preprocessing, and deep insights, we were able to determine how to conduct sentiment analysis on user reviews of movies on IMDb to determine if a comment is neutral, positive, or negative. Businesses can improve product offerings, strengthen customer connections, and improve marketing strategies by gleaning subtle insights from the vast ocean of customer sentiments. This approach will ultimately lead to long-term success in the market.

**With reference to the link below :**

https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews?resource=download

**THE METHODS THAT ARE IMPLEMENTED:**

* Data Collection
* Text Preprocessing
* Tokenization
* Sentiment Analysis Techniques

1.Data Collection:

•Gather diverse sources of customer feedback, including online reviews, social media posts, and customer surveys, related to competing products.

2. Text Preprocessing:

•Cleanse and preprocess the textual data by removing irrelevant characters, stopwords , and special symbols to enhance the quality of analysis.

3. Tokenization:

•Break down the preprocessed text into individual words or tokens, enabling further analysis at a granular level.

4. Sentiment Analysis Techniques:

•Utilize sentiment analysis algorithms like VADER (Valence Aware Dictionary and Entiment Reasoner) and machine learning models such as Support Vector Machines (SVM) and Recurrent Neural Networks (RNN) to classify sentiments as positive, negative, or neutral.

**INSIGHTS AND IMPLICATIONS** :

1.Identify Product Strengths and Weaknesses:

•Determine which features or aspects of competing products are praised or criticized most frequently, helping businesses enhance their strengths and address weaknesses.

2. Customer Preferences and Expectations:

•Analyze sentiments to understand customer expectations, preferences, and desires, guiding product development and marketing strategies.

3. Brand Perception:

•Evaluate how customers perceive different brands in the market, allowing businesses to position their products effectively and build a strong brand image.

4. Competitor Analysis:

•Compare sentiments across competing products to gain insights into competitors' strengths and weaknesses, enabling strategic decision-making.

**BENEFITS OF NLP –BASED SENTIMENT ANALYSIS IN MARKETING:**

1. Data-Driven Decision Making
2. Enhanced Customer Engagement
3. Proactive Issue Resolution
4. Improved Product Development

**DESIGN:**

Sentiment Analysis

Techniques

Data Collection

Data Preprocessing

Feature Extraction

Insights Generation

Visualization

Here are some significant advances and approaches we utilized BERT language model to estimate IMDb reviews and values in our model:

1. Choose pretrained model
2. Tokenization
3. Model architecture
4. Fine tuning process

**1. CHOOSING PRETRAINED MODEL:** Choosing a pre-trained model in sentiment analysis for IMDb reviews refers to selecting a pre-existing machine learning or deep learning model that has already been trained on a large dataset for a related natural language processing (NLP) task. This pre-trained model serves as the foundation for your sentiment analysis task on IMDb reviews. By using a pre-trained model, you can take use of the substantial knowledge already inherent in these models while saving time and effort when compared to building a sentiment analysis model from scratch. Because the model has a strong foundation in comprehending English language, this approach frequently results in more accurate sentiment analysis for IMDb reviews.

**2.TOKENIZATION:** Tokenization is the process of dividing a review into tokens or words. Tokenization would produce the following: ["This", "movie", "was", "great", "!", "I", "loved", "the", "acting", "and", "the", "plot", "."]. Tokenization is an important preprocessing step in natural language processing applications such as sentiment analysis since it enables the algorithm to deal with structured input that can then be fed into machine learning models for sentiment classification.

**3.MODEL ARCHITECTURE :**

• Input Layer: The input layer represents the text data from IMDb reviews, usually encoded as sequences of words or word embeddings.

• Word Embeddings: Pre-trained word embeddings like Word2Vec, Glo Ve, or fast Text are often used to convert words into dense vector representations, capturing semantic relationships.

• Convolutional Layers: In CNN-based architectures, convolutional layers are used to detect patterns and features in the word embeddings. These layers slide filters over the text data to identify relevant features.

• Pooling Layers: Max-pooling or average-pooling layers are applied to reduce the dimensionality and capture the most important information from the convolutional layers.

• Fully Connected Layers: These layers process the extracted features and feed them into a neural network that maps them to a binary sentiment prediction (positive or negative).

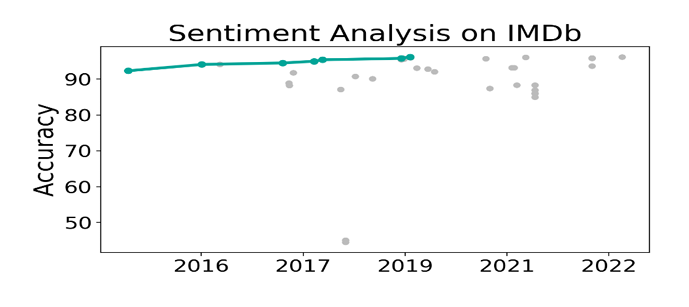
• Output Layer: The output layer typically consists of a single neuron with a sigmoid activation function, which outputs a probability score between 0 and 1. A threshold (e.g., 0.5) is used to classify the sentiment.

• Training: The model is trained on a labeled dataset of IMDb reviews with their corresponding sentiments. The loss function, such as binary cross-entropy, measures the prediction's error, and optimization techniques like gradient descent are used to update the model's weights.

**4.FINE TUNING PROCESS:** In the context of sentiment analysis for IMDb reviews, fine-tuning often refers to the process of adjusting a pre trained model to perform better on IMDb movie reviews. Here's a quick rundown of the fine-tuning procedure:

* Pre-trained Model: Begin using a model that has already been trained on a big dataset, such as BERT. From large amounts of text data, these models have learnt a good representation of the language.
* IMDb Dataset: Compile a dataset of IMDb reviews that have been labeled with their feelings (e.g., positive or negative).
* Fine-tuning: Train the pre-trained model using the IMDb dataset. The model's weights are incrementally changed during this process to make it better at predicting sentiment in movie reviews.
* Divide the dataset into two parts: Feed the training set reviews into the pre-trained model. Adjust the weights of the model based on the difference between its predictions and the actual sentiments. To avoid overfitting, keep an eye on performance on the validation set. After fine-tuning, assess the model's performance on a test set of IMDb reviews that it hasn't seen previously. This indicates how well the model will function in realistic settings.
* Deployment: Once the model's performance has been validated, it may be used to forecast the sentiment of fresh IMDb reviews.

Remember that fine-tuning must be handled carefully to avoid overfitting, especially when the fine-tuning dataset (IMDb reviews in this case) is significantly smaller than the original dataset on which the model was pre-trained.



Now we are moving to see how data preprocessing method and some of the NLP methods in sentiment analysis is working …

**DATA PREPROCESSING:**

Once you have collected the data, you'll need to preprocess it to ensure it's clean and ready for analysis. This involves:

Handling missing data: You may need to deal with missing values in any of the fields, possibly by imputing them or removing the corresponding entries.

Data encoding: Categorical data like genre and languages need to be encoded into a numerical format. This can be done using techniques like one-hot encoding or label encoding.

Feature scaling: You might need to scale numeric features like runtime to have similar ranges, especially if you plan to use algorithms that are sensitive to feature scales, such as gradient descent-based methods.

Outlier detection: Identify and handle outliers in IMDb scores or runtime that could skew the predictions.

**THE ESSENTIAL LIBRARIES:**

We start by importing the main libraires that we will use:

(1) the re module (for regular expression matching operations)

(2) the nltk toolkit (for natural language operations)

(3) the random module (for random number generation)

(4) the numpy library (for arrays operations)

(5) the pandas library (for data analysis)

(6) the scipy. stats module (for statistics)

(7) the seaborn library (for statistical data visualization)

(8) the matplotlib .pyplot interface (for MATLAB-like plots) We also download the stopwords and punkt data packages from the nltk toolkit.

In [1] :

import re

import nltk

import random

import numpy as np

import pandas as pd

import seaborn as sns

from scipy import stats

import matplotlib.pyplot as plt

#nltk.download('punkt')

#nltk.download('stopwords')

# Setting as large the xtick and ytick font sizes in graphs

plt.rcParams['xtick.labelsize'] = 'large'

plt.rcParams['ytick.labelsize'] = 'large'

IMDB DATASET: We obtain the csv dataset "IMDB Dataset.csv" from Kaggle, which contains 50'000 IMDB movie and TV show reviews with their positive or negative sentiment classification.

In [2]:

# Storing the csv file into a DataFrame "df"

df = pd.read\_csv('../input/imdb-dataset-of-50k-movie-reviews/IMDB Dataset.csv')

df

Out[2]:

|  | 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 |
| ... | ... | ... |
| 49995 | I thought this movie did a down right good job... | positive |
| 49996 | Bad plot, bad dialogue, bad acting, idiotic di... | negative |
| 49997 | I am a Catholic taught in parochial elementary... | negative |
| 49998 | I'm going to have to disagree with the previou... | negative |
| 49999 | No one expects the Star Trek movies to be high... | negative |

50000 rows × 2 columns

We print the DataFrame's fundamental attributes. We especially notice that there are no null values in the DataFrame.

In [3]:

print('\033[1m' + 'df.shape:' + '\033[0m', df.shape)

print('\033[1m' + 'df.columns:' + '\033[0m', df.columns, '\n')

print('\033[1m' + 'df.sentiment.value\_counts():' + '\033[0m')

print(df.sentiment.value\_counts(), '\n')

with sns.axes\_style("darkgrid"):

df['sentiment'].value\_counts().plot.bar(color=['darkblue', 'r'], rot=0, fontsize='large')

plt.show()

print('\033[1m' + 'df.info:' + '\033[0m')

df.info()

df.shape: (50000, 2)

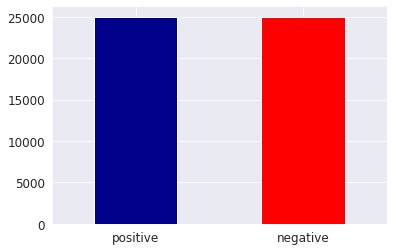
df.columns: Index(['review', 'sentiment'], dtype='object')

df.sentiment.value\_counts():

positive 25000

negative 25000

Name: sentiment, dtype: int64



**df.info**

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 50000 entries, 0 to 49999

Data columns (total 2 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 review 50000 non-null object

1 sentiment 50000 non-null object

dtypes: object(2)

memory usage: 781.4+ KB

To prepare the DataFrame for analysis, transform its sentiment values to integers:

👉 positive → 1

👉 negative → 0

In [4]:

df.sentiment = [1 if s == 'positive' else 0 for s in df.sentiment]

df

Out[4]:

|  | review | sentiment |
| --- | --- | --- |
| 0 | One of the other reviewers has mentioned that ... | 1 |
| 1 | A wonderful little production. <br /><br />The... | 1 |
| 2 | I thought this was a wonderful way to spend ti... | 1 |
| 3 | Basically there's a family where a little boy ... | 0 |
| 4 | Petter Mattei's "Love in the Time of Money" is... | 1 |
| ... | ... | ... |
| 49995 | I thought this movie did a down right good job... | 1 |
| 49996 | Bad plot, bad dialogue, bad acting, idiotic di... | 0 |
| 49997 | I am a Catholic taught in parochial elementary... | 0 |
| 49998 | I'm going to have to disagree with the previou... | 0 |
| 49999 | No one expects the Star Trek movies to be high... | 0 |

50000 rows × 2 columns

Data preprocessing: In this we are going to preprocess the above data ,First, we utilize regular expressions to modify the reviews as follows:

* remove punctuation marks
* remove HTML tags
* remove URL's
* remove characters which are not letters or digits
* remove successive whitespaces
* convert the text to lower case
* strip whitespaces from the beginning and the end of the reviews

In [5]:

# Storing in "before\_process" a random example of review before preprocessing

# Defining and applying the function "process" performing the transformations of the reviews

# Storing in "after\_process" the example of review after preprocessing

idx = random.randint(0, len(df)-1)

before\_process = df.iloc[idx][0]

def process(x):

x = re.sub('[,\.!?:()"]', '', x)

x = re.sub('<.\*?>', ' ', x)

x = re.sub('http\S+', ' ', x)

x = re.sub('[^a-zA-Z0-9]', ' ', x)

x = re.sub('\s+', ' ', x)

return x.lower().strip()

df['review'] = df['review'].apply(lambda x: process(x))

after\_process = df.iloc[idx][0]

Then, we use the word\_tokenize() method from the nltk.tokenize package to remove stopwords from the reviews.

In [6]:

# Storing in "sw\_set" the set of English stopwords provided by nltk

# Defining and applying the function "sw\_remove" which remove stopwords from reviews

# Storing in "after\_removal" the example of review after removal of the stopwords

sw\_set = set(nltk.corpus.stopwords.words('english'))

def sw\_remove(x):

words = nltk.tokenize.word\_tokenize(x)

filtered\_list = [word for word in words if word not in sw\_set]

return ' '.join(filtered\_list)

df['review'] = df['review'].apply(lambda x: sw\_remove(x))

after\_removal = sw\_remove(after\_process)

As an example, we print the review before preprocessing, after preprocessing, and after stopwords removal.

In [7]:

print('\033[1m' + 'Review #%d before preprocessing:' % idx + '\033[0m' + '\n', before\_process, '\n')

print('\033[1m' + 'Review #%d after preprocessing:' % idx + '\033[0m' + '\n', after\_process, '\n')

print('\033[1m' + 'Review #%d after preprocessing and stopwords removal:' % idx + '\033[0m' + '\n', after\_removal)

out:

Review #49495 before preprocessing:

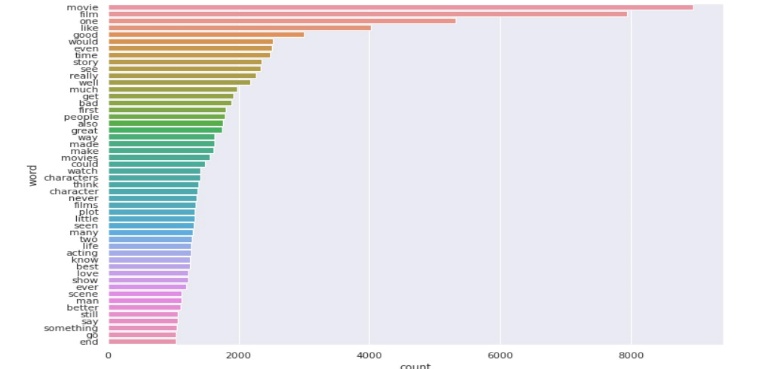
Personally, I think that the film was done very professionally, I loved the choreography and the acting. The plot is also gripping and mysterious. The film itself is very emotional, and what I liked about it most is that it makes you think afterwards. Antonio Gades has absolutely lived his role to the end, and I must say that it's one of my favourite pictures and Saura is a wonderful director.

Review #49495 after preprocessing:

personally i think that the film was done very professionally i loved the choreography and the acting the plot is also gripping and mysterious the film itself is very emotional and what i liked about it most is that it makes you think afterwards antonio gades has absolutely lived his role to the end and i must say that it s one of my favourite pictures and saura is a wonderful director

Review #49495 after preprocessing and stopwords removal:

personally think film done professionally loved choreography acting plot also gripping mysterious film emotional liked makes think afterwards antonio gades absolutely lived role end must say one favourite pictures saura wonderful director



**Now we are going to employ NLP techniques:**

i) Data splitting

ii) Tokenization

iii) LSTM model

Data splitting: When data is divided into two or more subgroups, this is known as data splitting. When using a two-part split, the data is usually evaluated or tested in part, while the model is trained in the other half. A crucial component of data science is data splitting, especially when building models with data.

Tokenization: Tokenization involves cutting the raw text into manageable pieces. Tokenization divides the original text into tokens, which are words or sentences. These tokens are useful for building the NLP model or comprehending the context. By examining the word order, tokenization assists in deciphering the text's meaning.

**Data splitting and tokenization:**

We start by splitting our DataFrame into a training and test lists. We use the [train\_test\_split()](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) function from the [sklearn.model\_selection](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection) module which allow to perform the splitting randomly with respect to the index of the DataFrame.

In [8]:

from sklearn.model\_selection import train\_test\_split

train\_rev, test\_rev, train\_sent, test\_sent = train\_test\_split(df['review'], df['sentiment'], test\_size=0.1, random\_state=42)

print('**\033**[1m' + 'train\_rev.shape:' + '**\033**[0m', train\_rev.shape)

print('**\033**[1m' + 'test\_rev.shape:' + '**\033**[0m', test\_rev.shape)

print('**\033**[1m' + 'train\_sent.shape:' + '**\033**[0m', train\_sent.shape)

print('**\033**[1m' + 'test\_sent.shape:' + '**\033**[0m', test\_sent.shape)

**train\_rev.shape:** (45000,)

**test\_rev.shape:** (5000,)

**train\_sent.shape:** (45000,)

**test\_sent.shape:** (5000,)

Next, we use the Tokenizer class from keras.preprocessing.text module to create a dictionary of the "dict\_size" most frequent words present in the reviews (a unique integer is assigned to each word), and we print some of its attributes. The index of the Tokenizer is computed the same way no matter how many most frequent words we use later, see this post.

In [9]:

from keras.preprocessing.text import Tokenizer

dict\_size = 35000

tokenizer = Tokenizer(num\_words=dict\_size)

tokenizer.fit\_on\_texts(df['review'])

print('**\033**[1m' + 'Dictionary size:' + '**\033**[0m', dict\_size)

print('**\033**[1m' + 'Length of the tokenizer index:' + '**\033**[0m', len(tokenizer.word\_index))

print('**\033**[1m' + 'Number of documents the tokenizer was trained on:' + '**\033**[0m', tokenizer.document\_count, '**\n**')

print('**\033**[1m' + 'First 20 entries of the tokenizer index:' + '**\033**[0m')

print(\*list(tokenizer.word\_index.items())[:20])

**Dictionary size:** 35000

**Length of the tokenizer index:** 125791

**Number of documents the tokenizer was trained on:** 50000

**First 20 entries of the tokenizer index:**

('movie', 1) ('film', 2) ('one', 3) ('like', 4) ('good', 5) ('time', 6) ('even', 7) ('would', 8) ('really', 9) ('story', 10) ('see', 11) ('well', 12) ('much', 13) ('get', 14) ('bad', 15) ('people', 16) ('great', 17) ('also', 18) ('first', 19) ('made', 20)

We use the texts\_to\_sequences() function of the Tokenizer class to convert the training reviews and test reviews to lists of sequences of integers (tokens) "train\_rev\_tokens" and "test\_rev\_tokens", and we store in the numpy array "seq\_lengths" the lengths of the sequences included in "train\_rev\_tokens".

In [10]:

train\_rev\_tokens = tokenizer.texts\_to\_sequences(train\_rev)

test\_rev\_tokens = tokenizer.texts\_to\_sequences(test\_rev)

seq\_lengths = np.array([len(sequence) for sequence **in** train\_rev\_tokens])

If the lengths of the sequences were normally distributed, then a given length could be considered small or large when outside the interval

mean value of seq\_lengths

±

2 standard deviations of seq\_lengths,

and lengths not belonging to this interval would only represent 5% of the elements of seq\_lengths (see the 68–95–99.7 rule in statistics). Here, we follow this heuristics, and thus define an upper bound for the length of sequences accordingly.

(note that we use only the training set to define this upper bound, in order to avoid any data leakage or look-ahead bias)

In [11]:

*# Storing in "upper\_bound" our chosen upper bound for the length of sequences*

*# Computing the percentage of lengths smaller or equal than "upper\_bound"*

upper\_bound = int(np.mean(seq\_lengths) + 2 \* np.std(seq\_lengths))

percentage = stats.percentileofscore(seq\_lengths, upper\_bound)

print('The value of upper\_bound is **%d** and the percentage of sequences in "train\_rev\_tokens" **\**

of length smaller or equal than upper\_bound is **%.2f%%**.' % (upper\_bound, round(percentage, 2)))

*# Histogram plot of the lengths of the sequences in "train\_rev\_tokens"*

with sns.axes\_style("darkgrid"):

\_, hist = plt.subplots(figsize=(10,6))

hist.hist(seq\_lengths[seq\_lengths < 2\*upper\_bound], color='darkblue', bins=40, rwidth=0.7)

hist.axvline(np.mean(seq\_lengths), color='darkorange', linestyle='--', label='Mean value')

hist.axvline(upper\_bound, color='r', linestyle='--', label='Upper bound')

plt.xlabel('Length of sequences in "train\_rev\_tokens"', size='large')

plt.ylabel('Number of samples', size='large')

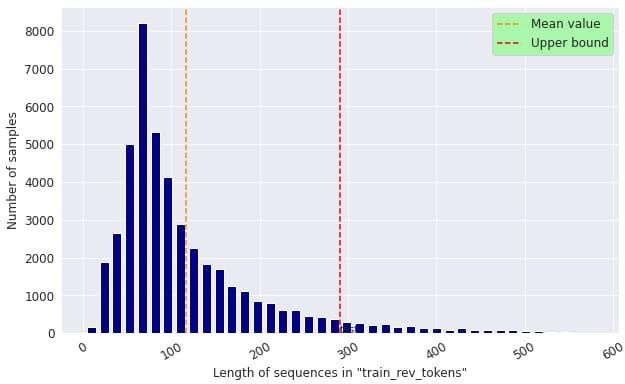
plt.text(upper\_bound, 0, 'test')

plt.legend(fontsize='large', facecolor='palegreen')

plt.xticks(rotation=30)

plt.show()

The value of upper\_bound is 291 and the percentage of sequences in "train\_rev\_tokens" of length smaller or equal than upper\_bound is 94.56%.



Using the pad sequences() function from keras preprocessing sequence module, we transform “train rev tokens” and “test rev tokens” into 2D numpy arrays of shape (number of sequences, upper bound). Sequences of length smaller (resp. larger) than “upper bound” are extended (resp. truncated) to get a length equal to “upper bound”.

# In [12]:

From keras.preprocessing.sequence import pad\_sequences

Train\_rev\_pad = pad\_sequences(train\_rev\_tokens, maxlen=upper\_bound) Test\_rev\_pad = pad\_sequences(test\_rev\_tokens, maxlen=upper\_bound)

Print(‘\033[1m’ + ‘train\_rev\_pad.shape:’ + ‘\033[0m’, train\_rev\_pad.shape) Print(‘\033[1m’ + ‘test\_rev\_pad.shape:’ + ‘\033[0m’, test\_rev\_pad.shape, ‘\n’)

# Printing an example of review after padding

Idx\_pad = random.randint(0, len(train\_rev\_pad)-1)

Print(‘\033[1m’ + ‘Review #%d after padding:’ %idx\_pad + ‘\033[0m’ + ‘\n’, train\_rev\_pad[idx\_pad])

Train\_rev\_pad.shape: (45000, 291)

Test\_rev\_pad.shape: (5000, 291)

**Review #2446 after padding:**

[ 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 297 2521 2 64 1389 17

2593 11 83 29 108 2139 578 150 1748 406 2526 1567

7 9163 64 5 330 1144 3 5 535 880 2529 2

1072 31303 783 2 1240 187 308 202 308 270 56 2062

216 2 81 240 40 2 173 666 37 332 375 2

10 5 2593 17 899 995 4010 2593 286 690 10 28

837 30 78 228 1489 2593 14333 1628 10076 96 40 28

10 517 3188 113 333 25 40 25 1414 1690 2288 54

83 29 1975 1578 795 1273 14132 21 69 330 1144 3

5110 677 2]

LSTM: In the realm of deep learning, long short-term memory is an artificial recurrent neural network design. LSTM has feedback connections, in contrast to conventional feedforward neural networks.

Unlike a conventional recurrent neural network, which retains all of the data, an LSM just stores the data in its short-term memory.

# We start by importing some classes from Keras:

* The Sequential class from the keras.models API (to group a linear stack of layers into a model)
* The Embedding class from the keras.layers API (to turn positive integers (indexes) into dense vectors of fixed size)
* The LSTM class from the keras.layers API (to apply a long short-term memory layer to an input)
* The Dropout class from the keras.layers API (to apply dropout to an input .
* The Dense class from the keras.layers API (to apply a regular densely-connected NN layer to an input)

In [13]:

From keras.models import Sequential

From keras.layers import Embedding, LSTM, Dropout, Dense

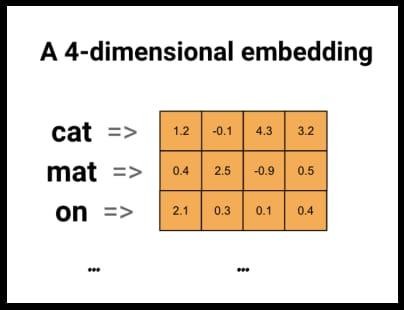
In [14]:

# Importing the “imageio.v3” library (for reading and writing images) # See <https://imageio.readthedocs.io/en/stable/>

# Import imageio.v3 as iio

Image = iio.imread(<https://www.tensorflow.org/text/guide/images/embedding2.png>) Plt.figure(figsize = (7, 7))

# Plt.imshow(image) Plt.axis(‘off’);



In the LSTM model, we set the following parameters:

* The output dimension of the Embedding layer (dimension of the vector space containing the word embeddings) is “output\_dim”
* The number of units of the LSTM layer is “units\_lstm”
* The dropout rate of the Dropout layer is “r”
* The activation function of the final Dense layer is sigmoid (this is a natural choice since the output of the model should be a number between 0, for negative reviews, and 1, for positive reviews)

# In [15]:

Output\_dim = 14

Units\_lstm = 16

R = 0.8

Model = Sequential()

Model.add(Embedding(input\_dim=dict\_size, output\_dim=output\_dim,

input\_length=upper\_bound)) Model.add(LSTM(units\_lstm))

Model.add(Dropout®)

Model.add(Dense(1, activation=’sigmoid’))

We give a summary of the model using the [summary](https://keras.io/api/models/model/#summary-method) method of the [model class](https://keras.io/api/models/model/) of Keras. The "None" value stands for the (not yet defined) value of the batch size.

In [16]:

model.summary()

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding (Embedding) (None, 291, 14) 490000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm (LSTM) (None, 16) 1984

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout (Dropout) (None, 16) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 1) 17

=================================================================

Total params: 492,001

Trainable params: 492,001

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Non-trainable params: 0

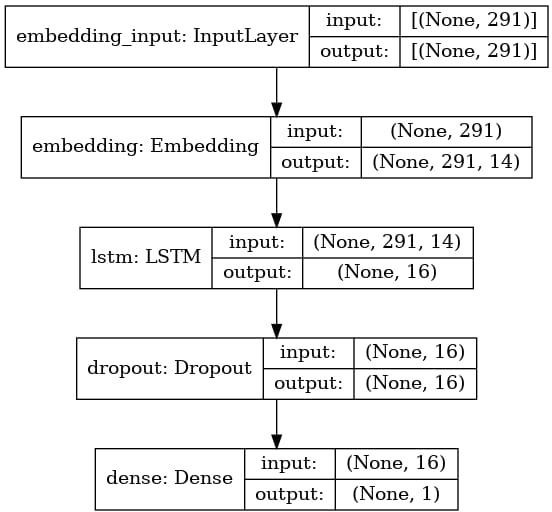
We import the plot\_model function from the keras.utils.vis\_utils module to plot a schema of the model.

# In [17]:

From keras.utils.vis\_utils import plot\_model

Plot\_model(model, show\_shapes=True

Out[17]:



We compile the model for training with the following parameters:

* Adam as optimizer to use during training process (a combination of gradient descent with momentum and RMSP)
* Binary cross-entropy (bce) between true labels and predicted labels as loss to minimise during training process
* Accuracy as metric to display during training process (how often predicted labels equal true labels)

# In[18]:

Model.compile(optimizer=’adam’, loss=’bce’, metrics=’accuracy’)

We train the model with “train\_rev\_pad” as input array, “train\_sent” as output array, validation split, batch size, number of epochs, and the option “shuffle=True” to shuffle the training data before each epoch. An epoch is a pass of the neural network over the entire training set and the batch size is the number of samples that are passed to the network at once. For each epoch, we thus have

Number of training steps

=

Length of training set – length of validation set Batch size

.

# In [19]:

Validation\_split = 0.1

Batch\_size = 384

Epochs = 3

Fitted = model.fit(train\_rev\_pad, train\_sent, validation\_split=validation\_split, Batch\_size=batch\_size, epochs=epochs, shuffle=True)

I tensorflow/compiler/mlir/mlir\_graph\_optimization\_pass.cc:185] None of the MLIR Optimization Passes are enabled (registered 2)

Epoch 1/3

106/106 [==============================] - 32s 280ms/step - loss: 0.6357 - accuracy: 0.6617 - val\_loss: 0.4595 - val\_accuracy: 0.8480

Epoch 2/3

106/106 [==============================] - 29s 276ms/step - loss: 0.3650 - accuracy: 0.8758 - val\_loss: 0.2707 - val\_accuracy: 0.8936

Epoch 3/3

106/106 [==============================] - 29s 275ms/step - loss: 0.2579 - accuracy: 0.9200 - val\_loss: 0.2615 - val\_accuracy: 0.8971

# In [20]:

# Storing in “ep\_values” the values of the epochs

Ep\_values = range(1, epochs+1)

# Plot of the training loss and validation loss (binary cross-entropy)

With sns.axes\_style(“darkgrid”):

\_, (loss, acc) = plt.subplots(1, 2, figsize=(15, 6))

Loss.plot(ep\_values, fitted.history[‘loss’], color=’darkblue’, linestyle=’dotted’, Marker=’o’, label=’Training loss (binary cross-entropy)’)

Loss.plot(ep\_values, fitted.history[‘val\_loss’], color=’r’, linestyle=’dotted’, Marker=’o’, label=’Validation loss (binary cross-entropy)’)

Loss.set\_xlabel(‘Epoch’, size=’large’) Loss.legend(fontsize=’large’, facecolor=’palegreen’)

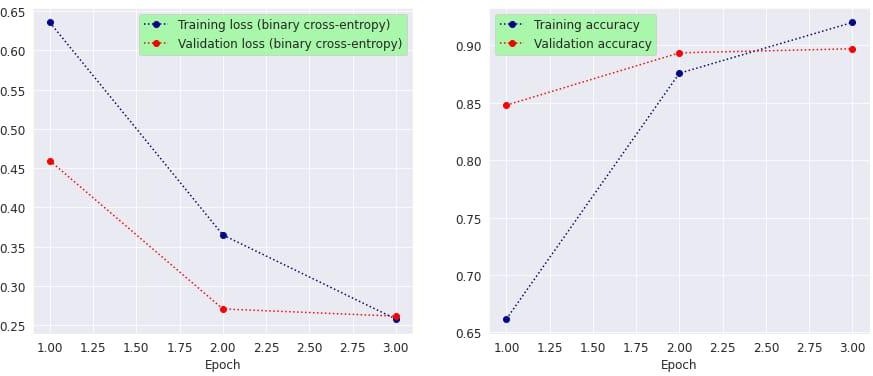
Acc.plot(ep\_values, fitted.history[‘accuracy’], color=’darkblue’, linestyle=’dotted’, Marker=’o’, label=’Training accuracy’)

Acc.plot(ep\_values, fitted.history[‘val\_accuracy’], color=’r’, linestyle=’dotted’, Marker=’o’, label=’Validation accuracy’)

Acc.set\_xlabel(‘Epoch’, size=’large’) Acc.legend(fontsize=’large’,

facecolor=’palegreen’)

Plt.show()



# **Results**

First, we [evaluate](https://keras.io/api/models/model_training_apis/#evaluate-method) the loss and accuracy of the trained model on the test set.

In [21]:

result= model.evaluate(test\_rev\_pad, test\_sent)

157/157 [==============================] - 2s 14ms/step - loss: 0.2522 - accuracy: 0.9028

Next, we use the [confusion\_matrix()](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html) function from the [sklearn.metrics](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics) module to compute the [confusion matrix](https://en.wikipedia.org/wiki/Confusion_matrix) for the predictions of the trained model, and we use the [heatmap](https://seaborn.pydata.org/generated/seaborn.heatmap.html) method from seaborn to plot the confusion matrix.

In [22]:

linkcode

from sklearn.metrics import confusion\_matrix

predictions = np.round(model.predict(test\_rev\_pad))

cf\_matrix = confusion\_matrix(test\_sent, predictions)

# In [23]:

# Storing in “legends” the legends of each entry of the confusion matrix

# Storing in “percentages” the percentages of each entry of the confusion matrix

# Storing in “labels” the grouped values (legend + percentage) of each entry of the confusion matrix

Legends = [‘True negatives’, ‘False positives’, ‘False negatives’, ‘True positives’] Percentages = [round(100\*num, 2) for num in cf\_matrix.flatten()/np.sum(cf\_matrix)]

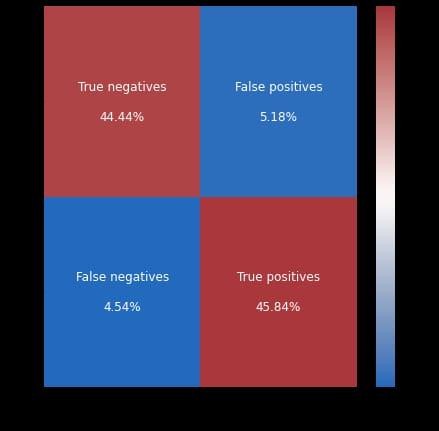
Labels = [f’{v1}\n\n{v2}%’ for v1, v2 in zip(legends, percentages)] Labels = np.asarray(labels).reshape(2, 2)

# Heatmap plot of the confusion matrix

Plt.figure(figsize = (7, 7))

Cm = sns.heatmap(cf\_matrix, annot=labels, fmt=’’, cmap=’vlag’, annot\_kws={‘fontsize’: ‘large’}) cm.set\_xlabel(‘Predicted sentiments’, size=’large’)

cm.set\_ylabel(‘Actual sentiments’, size=’large’) cm.xaxis.set\_ticklabels([‘Negative’, ‘Positive’]) cm.yaxis.set\_ticklabels([‘Negative’, ‘Positive’]) plt show()



Finally, we test the trained model on a randomly chosen review from the test set. We display the original review, the sentiment predicted by the model with its probability, and the actual (correct) sentiment.

# In[24]:

# Storing in DataFrame “df\_original” the original reviews and sentiments

Df\_original = pd.read\_csv(‘../input/imdb-dataset-of-50k-movie-reviews/IMDB Dataset.csv’)

# Choosing randomly a review and its sentiment in the test data

Idx\_test = random.randint(0, len(test\_sent)-1) Idx\_original = test\_rev.index[idx\_test]

(actual\_rev, actual\_sent) = df\_original.iloc[idx\_original]

# Storing in “prediction\_sent” the predicted sentiment of the chosen review

# Storing in “probability” the probability of the predicted sentiment of the chosen review

Prediction = model.predict(test\_rev\_pad)[idx\_test][0] Prediction\_sent = ‘positive’ if prediction >= 0.5 else ‘negative’

Probability = round(prediction if prediction >= 0.5 else 1-prediction, 2)

# Printing the original review, its predicted sentiment and probability, and original sentiment

Print(‘\033[1m’ + ‘Review #%d:’ % idx\_original + ‘\033[0m’ + ‘\n’, actual\_rev, ‘\n’)

Print(‘\033[1m’ + ‘Predicted sentiment:’ + ‘\033[0m’, prediction\_sent, ‘(with probability %.2f)’ % probability, ‘\n’)

Print(‘\033[1m’ + ‘Actual sentiment:’ + ‘\033[0m’, actual\_sent) Review #2491

1) Bad acting.<br /><br />2) For a bunch of castaways on an alien planet, it sure looked like home, especially with the houses and roads you can glimpse in the background.<br /><br />3) Terrible plot with stupid caracters making idiotic decisions and blithely losing precious survival equipment and clothing left, right and center.<br /><br />4) Cool 70’s scifi jumpsuits (possibly the only good thing about this movie)<br /><br />5) Interesting ship at the beginning (this crew must have been watching Space 1999 a lot). Too bad it blows up so early. The escape ship also got sunk too fast.

\*sigh\*<br /><br />6) Anthropologists might find some aspects of the movie interesting in terms of primate group behavior.

Predicted sentiment: negative (with probability 0.98)

Actual sentiment: negative

**FUTURE SCOPE:**

The potential of machine learning is too much, overtaking some of the human labor of some lexicon based tasks that requires intensive human labor. This is where machine learning comes into play. Such algorithms will also have to perceive and examine natural text context-wise and concept-wise. Time will also play a crucial part look in gat the amount of data which is being generated on the Web day-to-day.Now a days everyday people use Social Media platforms to show their sentiments inthe form of text, videos, images etc. So Sentiment Analysis plays a crucial role inmaking business strategy.Growth of Artificial Intelligence and Machine Learning ishighly reliable on how machines validate and match to different sentiments shown byhumans in the form of speech, text, videos, body language etc.

**CONCLUSION:**

Marketers can gain valuable insights into consumer preferences and opinions by employing natural language processing (NLP) to analyze the sentiment of IMDB movie reviews. Marketers can gain insights into audience sentiment by using NLP models like LSTM to gather, preprocess, and extract characteristics from movie reviews. The decisions made in marketing, such as creating customized campaigns, enhancing product development, and keeping an eye on a brand's reputation, can then be informed by these data.

One of the key benefits of using NLP for sentiment analysis is that it can be used to automate the process of analyzing large volumes of data. This is important because it allows marketers to quickly and easily get insights into audience sentiment, even for popular movies with millions of reviews.

Finding particular elements of a film that are evoking good or negative sentiment is another advantage of employing natural language processing (NLP) for sentiment analysis. A marketer could utilize NLP, for instance, to determine that while viewers are applauding a movie's visual effects, they are disliking the plot. Afterwards, marketing efforts that highlight the film's positive aspects and mitigate its negative aspects might be created using this information.

All things considered, marketers can learn a lot about consumer preferences and opinions by employing natural language processing (NLP) to analyze the sentiment of IMDB movie reviews. By automating the process of evaluating massive amounts of data and identifying particular elements of a film that are generating favorable or negative sentiment, marketers can gain insights that can be used to inform a variety of marketing decisions.

Here is a specific example of how sentiment analysis of IMDB movie reviews can be used in marketing:

A movie studio is releasing a new superhero movie. The studio wants to understand how audiences are reacting to the movie in order to develop targeted marketing campaigns. The studio collects a dataset of IMDB movie reviews for the movie and uses NLP techniques to preprocess the data and extract features. The studio then uses an LSTM model to generate insights into audience sentiment.

The LSTM model identifies that the movie is receiving positive reviews overall, but that some audiences are criticizing the movie's villain as being underdeveloped. The studio uses this information to develop a marketing campaign that highlights the movie's other strengths, such as its action sequences and special effects. The studio also uses this information to develop a post-release marketing campaign that addresses the criticism of the villain.

By using NLP to analyze IMDB movie reviews, the movie studio is able to gain insights into audience opinion that it can use to inform its marketing decisions.

THANK YOU