**University of Sunderland School of Computer Science**

**STUDENT NAME: ISAAC AKINWUMI**

**STUDENT ID: BI54UP**

**MODULE CODE: CETM47**

**MODULE TITLE: Machine Learning and Data Analytics**

**MODULE ASSESSOR: Ashley Williamson ASSESSMENT: 2 of 2**

**TITLE: Multi-class Classification of Tweets**

# Introduction

Nowadays social networks such as X (Twitter) is most indispensable in people’s daily lives, and thus it is important to keep the social community engaging (Ibrahim, Wang and Bourne, 2017). Establishing an interest assessment mechanism is very helpful for maintaining an engaging environment, which could be used for applications like a recommendation system X space and tweet. Twitter, in particular, stands out as a rich source of diverse textual content, capturing a wide range of topics that influence public dialogue (Bruns and Burgess, 2015). According to Hemmatian and Sohrabi (2019), Sentiment analysis is formally defined as the task of identifying and analysing subjective information about people’s opinions in social media sources. These sentiments are analysed through Natural Language Processing (NLP) either as binary task or multi-class text task. Multi-class text classification is the process of classifying text into more than two classes or categories.

This report aims to explore and compare various solutions for sentiment analysis and multi-class text classification on Twitter data. The focus will be on evaluating different methodologies and techniques, such as vectorization, stop-word removal, regex filtering, and various classification models. Through a series of experiments, the report will present the rationale behind each hypothesis, the evaluative methodology used, and the results obtained. Additionally, the evaluation section will assess the overall effectiveness of the solutions, concluding the best pipeline choices and classification models, and providing commentary on the success of preprocessing steps taken.

# Methodology

The project's task organization and execution follow the guidelines of the Cross-Industry Standard Process for Data Mining (CRISP-DM) methodology. The CRISP-DM methodology ensures that each phase of data mining and machine learning projects makes a significant contribution to the overall goal of building a strong Natural Language Processing (NLP) pipeline for Twitter topic classification (Schröer, Kruse and Gómez, 2021). This project benefits from CRISP-DM's versatility and comprehensiveness. The methodology is made up of six unique stages, business understanding, data understanding, data preparation, modelling, evaluation, and deployment. These steps work together to create a comprehensive. CRISP-DM is widely accepted, and it is the de facto standard in data analytics applications.

## Business Understanding

In this initial stage, the project's objectives are defined, the problem to be solved is identified, and the requirements for the task are determined. A clear understanding of the project needs and expectations is established, which helps to focus the project's efforts on delivering a solution that meets the stakeholders' requirements. This stage sets the foundation for the entire project, and on which the other stage relies.

The objective of this project is to introduce a reusable process model for tweet subject classification, to address the need for effective classification of tweets across various topics. This could improve understanding and insight into the content of tweets across different categories, enabling more effective analysis of user interests and engagement within the Twitter platform to support decision-making processes and enhance recommendation systems within the Twitter ecosystem. To achieve this data is collected, pre-processed and analysed based on this objective.

## Data Collection and understanding

In this stage, Twitter data is collected and explored to gain insights into its characteristics, quality, and relevance to the project's objectives. The data is analysed to identify patterns, trends, and relationships, which help to inform the subsequent stages of the project. This stage is critical in understanding the data's limitations and potential biases, which can impact the accuracy of the NLP pipeline.

Tweet from year 2019 to 2021 was utilized for this study which consists of 6,443 entries, representing tweets from the social media platform Twitter, covering six distinct topics: Arts & Culture, Business & Entrepreneurs, Pop Culture, Daily Life, Sports & Gaming, and Science & Technology. The dataset contains the following columns: 'text', 'date', 'label', and 'id'. The 'text' column stores the raw content of the tweet, including text, emojis, hashtags, and user mentions. To get a better understanding of the data word cloud for each class and target (label) was visualized as shown in figure 1 to 7 below. Findings show that the dataset is imbalanced, with Pop Culture being the most prevalent category, accounting for 2,512 tweets, followed by Sports & Gaming with 2,291 tweets, Daily Life with 883 tweets, Science & Technology with 326 tweets, Business & Entrepreneurs with 287 tweets, and Arts & Culture being the least represented category with 144 tweets.

## Data preparation

The collected Twitter data is cleaned, transformed, and prepared for modelling in this phase. This stage involves data preprocessing, feature engineering, and data transformation to create a dataset that is suitable for modelling. Missing values, outliers, and noisy data are handled to ensure that the dataset is reliable and consistent. Other preprocessing steps taken are removing punctuations like., ,, !, $, (, ), \\*, %, @, removing URLs, removing stop words, lower casing, tokenization, stemming, and lemmatization.

Preprocessing tasks like noise reduction, normalization, tokenization, stop word removal, and feature extraction are important (Mhatre et al., 2017). Stop word removal eliminates common words like "the," "is," and "and," which occur frequently but convey little semantic meaning, improving the efficiency of text analysis by reducing noise. Noise reduction helps remove punctuation, special characters, and irrelevant symbols, making the text cleaner and easier to analyse (Javed and Kamal, 2018). Normalization techniques like stemming and lemmatization help standardize different forms of words that convey the same meaning but appear in different forms. Tokenization divides text into meaningful units, such as words or phrases, facilitating subsequent processing steps like feature extraction (Arumugam, 2019). Preprocessing can also involve extracting features from text, such as word frequencies, n-grams, or word embeddings, which are essential for building machine learning models. Additionally, text data often has a high dimensionality due to the presence of a large vocabulary, and preprocessing techniques like term frequency-inverse document frequency (TF-IDF), Bag of words, embedding and dimensionality reduction methods can help reduce dimensionality (Alkhatib, Rensing and Silberbauer, 2017). This ensures that at the end data is ready to be feed in the machine learning model.

## Modelling

With the clean and pre-processed text data, the next step is to build models that can extract insights and meaning from the tweet. This involves selecting and training machine learning algorithms that can learn patterns and relationships in the data. Logistic regression, linear support vector machine, Navie Bayes and Random Forest were selected. Each was trained using Bag of words and Tfidf methods and Grid search was used to search for the optimal parameter of the model for better performance. To further obtain a better performance ensemble classifier was considered using the selected model. Additionally, deep learning model was explored, a tokenizer was used for the vectorisation and LSTM architecture was used. Long Short-Term Memory Networks is a deep learning, sequential neural network that allows information to persist (Balderas, Ponce and Molina, 2019). It is a special type of Recurrent Neural Network which is capable of handling the vanishing gradient problem faced by RNN. It excels at capturing long-term dependencies, making it ideal for sequence prediction tasks making it suitable for this tweet classification. With the train, validation and test data the model where trained and evaluated using their accuracy of classification, F1 score, Recall and precision as further discussed in the next section.

# Results

The result obtain are shown below, other results are shown in appendix B

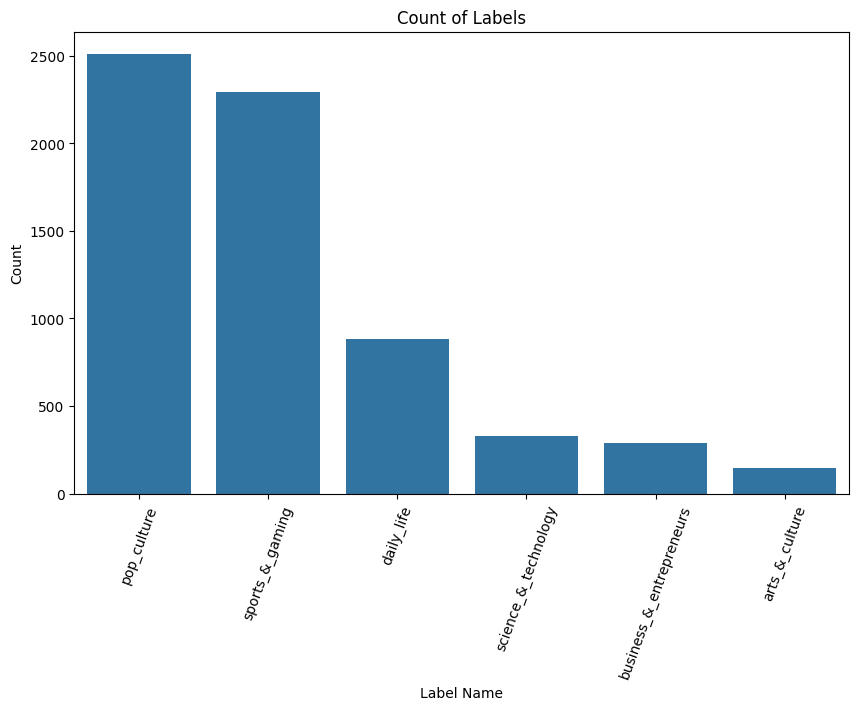


Figure 1 target varible distribution

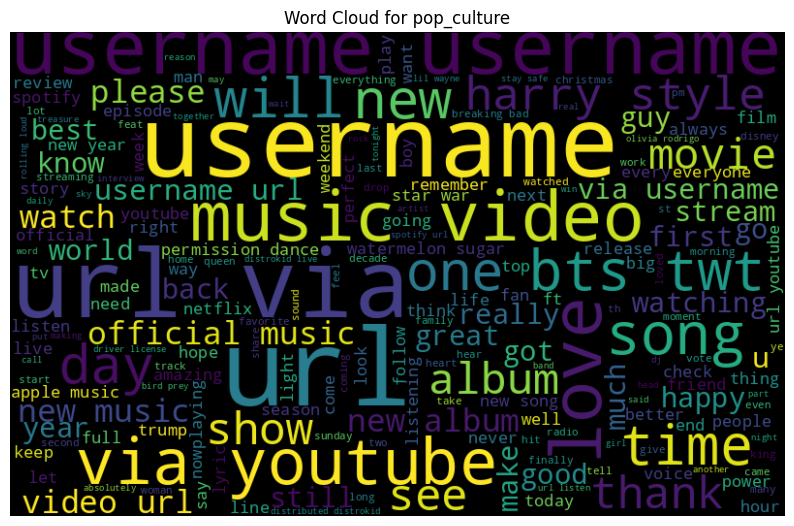
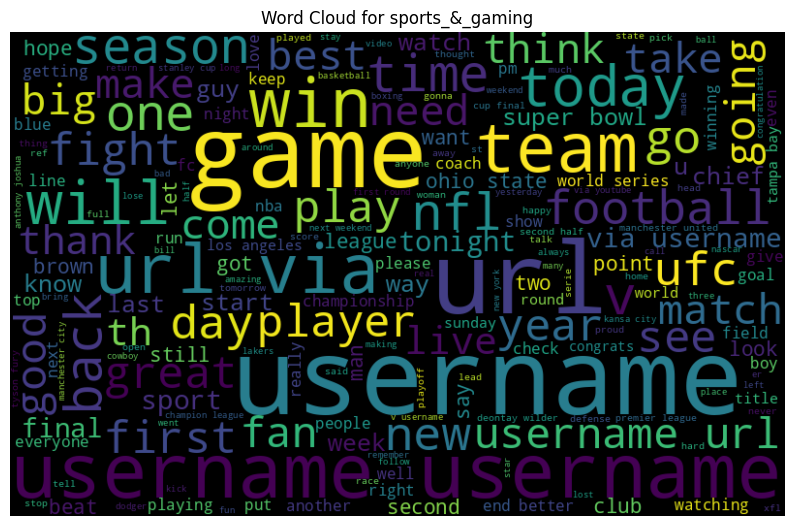


Figure 2 word cloud

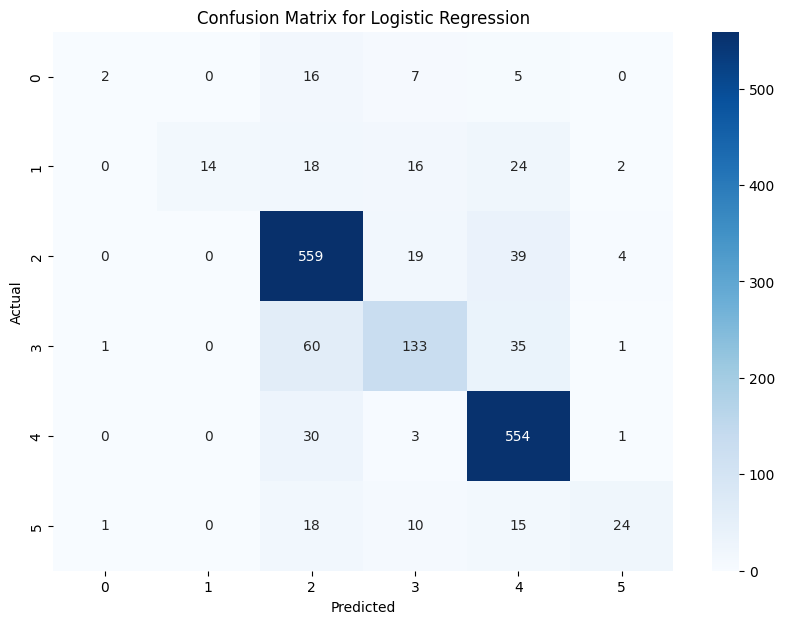
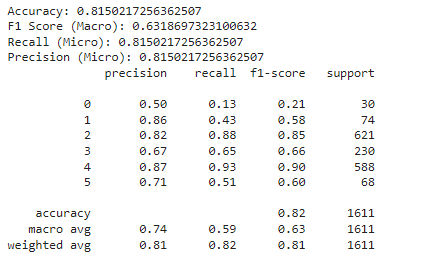
 

Figure 3 Best model

# Evaluation and Discussions

To evaluate the performance of the text classification models, accuracy, F1 score, precision and Roc evaluative methodology was employed. Accuracy measures the overall correctness of the model's predictions, while precision and recall capture the model's ability to correctly identify positive instances and avoid false positives and false negatives, respectively (Juba and Le, 2019). The F1-score provides a harmonic mean of precision and recall, offering a balanced measure of a model's performance.

The dataset was split into training (80%) and test (20%) sets, ensuring that the test set remained unseen during model training and parameter tuning. This allowed for assessing the models' ability to generalize to new, unseen data, which is crucial for real-world applications.

Several machine learning models were trained and evaluated, including Logistic Regression, Naive Bayes, Random Forest, Support Vector Machines (SVMs) and an Ensemble Voting Classifier combining this model. The models were trained on the pre-processed text data, which underwent various preprocessing steps such as TFIDF and Bag of words.

As shown in Figure 3, the Ensemble Voting Classifier achieved the highest accuracy of 82% using Tfidf vectorisation method on the test set, outperforming the other models whether with or with hyperparameter tuning using Grid search. Furthermore, the Ensemble Voting Classifier exhibited a high F1-score of 0.83, indicating a good balance between precision and recall. These results suggest that the Ensemble Voting Classifier is the best-performing model for the tweet classification task.

The Ensemble Voting Classifier was selected due to its ability to leverage the strengths of multiple models, thus improving performance over individual classifiers. Ensemble methods are known for their robustness to overfitting and their capacity to handle high-dimensional and non-linear data, which is often the case in text classification problems (Abiodun et al., 2021).

During the evaluation process, different preprocessing techniques and hyperparameter tuning were experimented with. For instance, stemming and lemmatization were tried to reduce word variations, and their impact on model performance was compared.

While the current methodology yielded promising results, there are potential areas for improvement. One avenue for exploration could be the incorporation of advanced techniques such as transformer-based models (e.g., BERT or GPT) for text representation. These techniques have shown remarkable success in capturing semantic and contextual information, which could further enhance the model's performance.

In summary, the evaluative methodology involved rigorous model training, hyperparameter tuning, and performance assessment using various metrics. The Ensemble Voting Classifier emerged as the best-performing model for the text classification task, exhibiting high accuracy and a good balance between precision and recall. While the current approach yielded promising results, there is room for improvement by incorporating advanced text representation techniques, which could further boost the model's performance and generalization capabilities.

# Future Work

There are several potential areas for further investigation such as

1. Incorporation of Advanced Text Representation Techniques:

The current approach utilized traditional feature extraction methods like TF-IDF vectorization. However, recent advancements in natural language processing have introduced powerful text representation techniques, such as word embeddings (e.g., Word2Vec, GloVe) and transformer-based models (e.g., BERT, GPT). These techniques have demonstrated remarkable success in capturing semantic and contextual information, leading to improved performance in various NLP tasks (Worth, 2023). Incorporating these advanced techniques could potentially enhance the model's ability to capture the nuances and complexities of the text data, leading to better classification accuracy and generalization.

1. Domain-specific Language Model Fine-tuning:

Pre-trained language models like BERT and GPT have been trained on large general-purpose corpora, which may not fully capture the domain-specific relationship and terminology present in the text data. Fine-tuning these pre-trained models on domain-specific corpora or incorporating domain-specific knowledge could potentially enhance their performance on the target task (Myers et al., 2023). This approach has shown promising results in various domains, such as biomedical text analysis and legal document processing.

1. Exploration of Multi-task and Transfer Learning Approaches:

Multi-task learning and transfer learning have gained significant attention in recent years for their ability to leverage knowledge from related tasks or domains, potentially improving performance and generalization (Zhou et al., 2021). Investigating the application of these techniques to the text classification task could be a promising direction, as they may enable the model to capture more robust and transferable representations, leading to improved performance and adaptability to new domains or tasks.

# Reference

Abiodun, E.O., Alabdulatif, A., Abiodun, O.I., Alawida, M., Alabdulatif, A. and Alkhawaldeh, R.S. (2021). Correction to: A systematic review of emerging feature selection optimization methods for optimal text classification: the present state and prospective opportunities. *Neural Computing and Applications*, 33(22), pp.15119–15119. doi:<https://doi.org/10.1007/s00521-021-06561-y>.

Alkhatib, W., Rensing, C. and Johannes Silberbauer (2017). Multi-label Text Classification Using Semantic Features and Dimensionality Reduction with Autoencoders. *Lecture notes in computer science*, pp.380–394. doi:<https://doi.org/10.1007/978-3-319-59888-8_32>.

Arumugam, M. (2019). Processing the Textual Information Using Open Natural Language Processing (NLP). *SSRN Electronic Journal*. doi:<https://doi.org/10.2139/ssrn.3361108>.

Balderas, D., Ponce, P. and Molina, A. (2019). Convolutional long short term memory deep neural networks for image sequence prediction. *Expert Systems with Applications*, 122, pp.152–162. doi:<https://doi.org/10.1016/j.eswa.2018.12.055>.

Bruns, A. and Burgess, J., (2015). Twitter hashtags from ad hoc to calculated publics. *Hashtag publics: The power and politics of discursive networks*, pp.13-28. <https://snurb.info/files/2015/Twitter%20Hashtags%20from%20Ad%20Hoc%20to%20Calculated%20Publics.pdf>

Ganaie, M.A., Hu, M., Malik, A.K., Tanveer, M. and Suganthan, P.N. (2022). Ensemble deep learning: A review. *Engineering Applications of Artificial Intelligence*, 115, p.105151. doi:<https://doi.org/10.1016/j.engappai.2022.105151>.

Hasan, A., Li, J., Reynolds, M., Sellis, T. and Jeffrey Xu Yu (2019). Location prediction in large-scale social networks: an in-depth benchmarking study. *The VLDB journal*, 28(5), pp.623–648. doi:<https://doi.org/10.1007/s00778-019-00553-0>.

Hemmatian, F. and Sohrabi, M.K. (2019). A survey on classification techniques for opinion mining and sentiment analysis. *Artificial Intelligence Review*, 52(3). doi:<https://doi.org/10.1007/s10462-017-9599-6>.

Ibrahim, N.F., Wang, X. and Bourne, H. (2017). Exploring the effect of user engagement in online brand communities: Evidence from Twitter. *Computers in Human Behavior*, 72, pp.321–338. doi:<https://doi.org/10.1016/j.chb.2017.03.005>.

Javed, M. and Kamal, S. (2018). Normalization of Unstructured and Informal Text in Sentiment Analysis. *International Journal of Advanced Computer Science and Applications*, 9(10). doi:<https://doi.org/10.14569/ijacsa.2018.091011>.

Juba, B. and Le, H.S. (2019). Precision-Recall versus Accuracy and the Role of Large Data Sets. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(1), pp.4039–4048. doi:<https://doi.org/10.1609/aaai.v33i01.33014039>.

Mhatre, M., Phondekar, D., Kadam, P., Chawathe, A. and Ghag, K. (2017). Dimensionality reduction for sentiment analysis using pre-processing techniques. *2017 International Conference on Computing Methodologies and Communication (ICCMC)*. doi:<https://doi.org/10.1109/iccmc.2017.8282676>.

Myers, D., Rami Mohawesh, Venkata Ishwarya Chellaboina, Anantha Lakshmi Sathvik, Venkatesh, P., Ho, Y.-H., Henshaw, H., Muna Alhawawreh, Berdik, D. and Yaser Jararweh (2023). Foundation and large language models: fundamentals, challenges, opportunities, and social impacts. *Cluster Computing*. doi:<https://doi.org/10.1007/s10586-023-04203-7>.

Schröer, C., Kruse, F. and Gómez, J.M. (2021). A Systematic Literature Review on Applying CRISP-DM Process Model. *Procedia Computer Science*, 181(1), pp.526–534. doi:<https://doi.org/10.1016/j.procs.2021.01.199>.

Worth, P.J. (2023). Word Embeddings and Semantic Spaces in Natural Language Processing. *International Journal of Intelligence Science*, 13(01), pp.1–21. doi:<https://doi.org/10.4236/ijis.2023.131001>.

Zhou, Y., Zhang, X., Wang, Y. and Zhang, B. (2021). Transfer learning and its application research. *Journal of physics*, 1920(1), pp.012058–012058. doi:<https://doi.org/10.1088/1742-6596/1920/1/012058>.

# Appendix Code

# \*\*Import librares\*\*

! pip install Keras-Preprocessing

! pip install textblob

import numpy as np

import pandas as pd

import nltk

import tensorflow as tf

from nltk.stem import SnowballStemmer

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

import pickle

import joblib

from collections import Counter

from textblob import Word

from wordcloud import WordCloud

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

from sklearn.naive\_bayes import MultinomialNB

from sklearn.svm import LinearSVC, SVC

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier, AdaBoostClassifier, VotingClassifier

from sklearn.model\_selection import KFold

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, precision\_score, f1\_score, recall\_score

from sklearn.model\_selection import cross\_val\_score, cross\_val\_predict

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

from keras.preprocessing.text import Tokenizer

from keras.models import Sequential, load\_model

from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

from keras\_preprocessing.sequence import pad\_sequences

# from keras.utils.np\_utils import to\_categorical

from keras.layers import Activation, Dense, Embedding, LSTM, SpatialDropout1D, Dropout, Flatten, GRU, Conv1D, MaxPooling1D, Bidirectional

from wordcloud import WordCloud,ImageColorGenerator

from PIL import Image

import urllib

import requests

import re

! pip install ktrain

import ktrain

from ktrain import text

import pandas as pd

import re

from nltk.corpus import stopwords

from nltk.tokenize import RegexpTokenizer

from textblob import Word

%matplotlib inline

nltk.download('stopwords')

nltk.download('wordnet')

nltk.download('omw-1.4')

nltk.download('gutenberg')

nltk.download('brown')

nltk.download("reuters")

nltk.download('words')

### \*\*Load Dataset\*\*

df\_tweet\_multi=pd.read\_json('/content/CETM47-23\_24-AS2-Data.json', encoding='latin-1')

df\_tweet\_multi['label\_name'].value\_counts()

df\_tweet\_multi

# \*\*Data Cleaning\*\*

df\_tweet\_multi['text']=df\_tweet\_multi['text'].fillna("")

df\_tweet\_multi.isna().sum()

# \*\*Preprocessing\*\*

def preprocess\_text(df\_tweet\_multi, text\_column):

"""

Preprocesses the text data in the specified column of the DataFrame.

This function performs several preprocessing steps:

1. Converts text to lowercase, strips leading/trailing spaces, and replaces newlines with spaces.

2. Removes non-alphabetic characters and non-ASCII characters.

3. Removes links from the text.

4. Tokenizes the text into words.

5. Removes stopwords, with some exceptions.

6. Filters out words that are shorter than 2 characters.

7. Lemmatizes the words.

Parameters:

df\_tweet\_multi (pd.DataFrame): The DataFrame containing the text data.

text\_column (str): The name of the column containing the text to preprocess.

Returns:

pd.DataFrame: A DataFrame with a new column 'processed\_text' containing the preprocessed text.

"""

# Convert text to lowercase and clean spaces and newlines

df\_tweet\_multi['lower\_case'] = df\_tweet\_multi[text\_column].apply(lambda x: x.lower().strip().replace('\n', ' ').replace('\r', ' '))

# Remove non-alphabetic characters and non-ASCII characters

df\_tweet\_multi['alphabatic'] = df\_tweet\_multi['lower\_case'].apply(lambda x: re.sub(r'[^a-zA-Z\']', ' ', x))

df\_tweet\_multi['alphabatic'] = df\_tweet\_multi['alphabatic'].apply(lambda x: re.sub(r'[^\x00-\x7F]+', '', x))

# Remove links from the text

df\_tweet\_multi['without\_link'] = df\_tweet\_multi['alphabatic'].apply(lambda x: re.sub(r'http\S+', '', x))

# Tokenize the text into words

tokenizer = RegexpTokenizer(r'\w+')

df\_tweet\_multi['Special\_word'] = df\_tweet\_multi['without\_link'].apply(lambda x: tokenizer.tokenize(x))

# Remove stopwords, with some exceptions

stop = [word for word in stopwords.words('english') if word not in [

"my", "haven't", "aren't", "can", "no", "why", "through", "herself",

"she", "he", "himself", "you", "you're", "myself", "not", "here",

"some", "do", "does", "did", "will", "don't", "doesn't", "didn't",

"won't", "should", "should've", "couldn't", "mightn't", "mustn't",

"shouldn't", "hadn't", "wasn't", "wouldn't", 'url', 'username',

]]

df\_tweet\_multi['stop\_words'] = df\_tweet\_multi['Special\_word'].apply(lambda x: [item for item in x if item not in stop])

# Filter out words that are shorter than 2 characters

df\_tweet\_multi['short\_word'] = df\_tweet\_multi['stop\_words'].apply(lambda x: [word for word in x if len(word) >= 2])

# Join the words back into a single string

df\_tweet\_multi['string'] = df\_tweet\_multi['short\_word'].apply(lambda x: ' '.join(x))

# Lemmatize the words

df\_tweet\_multi['processed\_text'] = df\_tweet\_multi['string'].apply(lambda x: " ".join([Word(word).lemmatize() for word in x.split()]))

# Drop the intermediate columns

df\_tweet\_multi.drop(columns=['lower\_case', 'alphabatic', 'without\_link', 'Special\_word', 'stop\_words', 'short\_word', 'string'], inplace=True)

return df\_tweet\_multi

df\_tweet\_multi = preprocess\_text(df\_tweet\_multi, 'text')

df\_tweet\_multi

## Visualization

fig = plt.figure(figsize=(14,7))

df\_tweet\_multi['length'] = df\_tweet\_multi.processed\_text.str.split().apply(len)

ax1 = fig.add\_subplot(122)

sns.histplot(df\_tweet\_multi['length'], ax=ax1,color='green')

describe = df\_tweet\_multi.length.describe().to\_frame().round(2)

ax2 = fig.add\_subplot(121)

ax2.axis('off')

font\_size = 14

bbox = [0, 0, 1, 1]

table = ax2.table(cellText = describe.values, rowLabels = describe.index, bbox=bbox, colLabels=describe.columns)

table.set\_fontsize(font\_size)

fig.suptitle('Distribution of text length for text.', fontsize=16)

plt.show()

# Set up the figure and axis

plt.figure(figsize=(10, 6))

# Create the bar plot using value counts

label\_counts = df\_tweet\_multi['label\_name'].value\_counts()

sns.barplot(x=label\_counts.index, y=label\_counts.values)

# Rotate the x-axis labels to be vertical

plt.xticks(rotation=70)

# Optionally, add a title and labels

plt.title('Count of Labels')

plt.xlabel('Label Name')

plt.ylabel('Count')

# Show the plot

plt.show()

import matplotlib.pyplot as plt

from wordcloud import WordCloud

def generate\_wordcloud(df\_tweet\_multi, label\_column, text\_column, label\_value):

"""

Generates and displays a word cloud for a specific label value.

Parameters:

df\_tweet\_multi (pd.DataFrame): The DataFrame containing the data.

label\_column (str): The name of the column containing the labels.

text\_column (str): The name of the column containing the text data.

label\_value (str): The specific label value to filter the text data.

Returns:

None

"""

# Join all the text entries for the specified label

normal\_words = ' '.join(df\_tweet\_multi[text\_column][df\_tweet\_multi[label\_column] == label\_value])

# Generate the word cloud

wordcloud = WordCloud(width=800, height=500, random\_state=21, max\_font\_size=110).generate(normal\_words)

# Plot the word cloud

plt.figure(figsize=(10, 7))

plt.imshow(wordcloud, interpolation="bilinear")

plt.axis('off')

plt.title(f'Word Cloud for {label\_value}')

plt.show()

normal\_words =' '.join([text for text in df\_tweet\_multi['processed\_text']])

wordcloud = WordCloud(width=800, height=500, random\_state=21, max\_font\_size=110).generate(normal\_words)

plt.figure(figsize=(10, 7))

plt.imshow(wordcloud, interpolation="bilinear")

plt.axis('off')

plt.show()

generate\_wordcloud(df\_tweet\_multi, 'label\_name', 'processed\_text', 'pop\_culture')

generate\_wordcloud(df\_tweet\_multi, 'label\_name', 'processed\_text', 'sports\_&\_gaming')

generate\_wordcloud(df\_tweet\_multi, 'label\_name', 'processed\_text', 'daily\_life')

generate\_wordcloud(df\_tweet\_multi, 'label\_name', 'processed\_text', 'science\_&\_technology')

generate\_wordcloud(df\_tweet\_multi, 'label\_name', 'processed\_text', 'business\_&\_entrepreneurs')

generate\_wordcloud(df\_tweet\_multi, 'label\_name', 'processed\_text', 'arts\_&\_culture')

## Data Splitting

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df\_tweet\_multi["processed\_text"],df\_tweet\_multi["label"], test\_size = 0.25, random\_state = 42)

print(f'x\_train shape: {x\_train.shape}')

print(f'y\_train shape: {y\_train.shape}')

print(f'x\_test shape: {x\_test.shape}')

print(f'y\_test shape: {y\_test.shape}')

# Modelling

### Helper functions for plots

import warnings

from sklearn.exceptions import UndefinedMetricWarning

# Suppress UndefinedMetricWarning

warnings.filterwarnings("ignore", category=UndefinedMetricWarning)

# Your model evaluation code here

from sklearn.preprocessing import label\_binarize

from sklearn.metrics import roc\_curve, auc, precision\_recall\_curve, average\_precision\_score

def plot\_multiclass\_roc\_curve(y\_true, y\_scores, model\_name, n\_classes):

"""

Plots the ROC curve for a multiclass problem.

Parameters:

y\_true: array-like of shape (n\_samples,)

True labels.

y\_scores: array-like of shape (n\_samples, n\_classes)

Predicted scores/probabilities for each class.

model\_name: str

Name of the model (for the plot title).

n\_classes: int

Number of classes.

"""

# Binarize the output

y\_true\_bin = label\_binarize(y\_true, classes=[i for i in range(n\_classes)])

# Compute ROC curve and ROC area for each class

fpr = dict()

tpr = dict()

roc\_auc = dict()

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y\_true\_bin[:, i], y\_scores[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area

fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_true\_bin.ravel(), y\_scores.ravel())

roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])

plt.figure()

plt.plot(fpr["micro"], tpr["micro"], color='deeppink', linestyle=':', linewidth=4, label=f'micro-average ROC curve (AUC = {roc\_auc["micro"]:.2f})')

for i in range(n\_classes):

plt.plot(fpr[i], tpr[i], lw=2, label=f'ROC curve of class {i} (AUC = {roc\_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title(f'Receiver Operating Characteristic for {model\_name}')

plt.legend(loc='lower right')

plt.show()

def plot\_multiclass\_precision\_recall\_curve(y\_true, y\_scores, model\_name, n\_classes):

"""

Plots the Precision-Recall curve for a multiclass problem.

Parameters:

y\_true: array-like of shape (n\_samples,)

True labels.

y\_scores: array-like of shape (n\_samples, n\_classes)

Predicted scores/probabilities for each class.

model\_name: str

Name of the model (for the plot title).

n\_classes: int

Number of classes.

"""

# Binarize the output

y\_true\_bin = label\_binarize(y\_true, classes=[i for i in range(n\_classes)])

# Compute Precision-Recall and plot curve for each class

precision = dict()

recall = dict()

average\_precision = dict()

for i in range(n\_classes):

precision[i], recall[i], \_ = precision\_recall\_curve(y\_true\_bin[:, i], y\_scores[:, i])

average\_precision[i] = average\_precision\_score(y\_true\_bin[:, i], y\_scores[:, i])

# Compute micro-average Precision-Recall curve and area

precision["micro"], recall["micro"], \_ = precision\_recall\_curve(y\_true\_bin.ravel(), y\_scores.ravel())

average\_precision["micro"] = average\_precision\_score(y\_true\_bin, y\_scores, average="micro")

plt.figure()

plt.plot(recall["micro"], precision["micro"], color='gold', linestyle=':', linewidth=4, label=f'micro-average Precision-Recall curve (AP = {average\_precision["micro"]:.2f})')

for i in range(n\_classes):

plt.plot(recall[i], precision[i], lw=2, label=f'Precision-Recall curve of class {i} (AP = {average\_precision[i]:.2f})')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title(f'Precision-Recall Curve for {model\_name}')

plt.legend(loc='lower right')

plt.show()

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import roc\_curve, auc, precision\_recall\_curve, average\_precision\_score

def plot\_confusion\_matrix(y\_true, y\_pred, model\_name):

"""

Plots the confusion matrix for the given true and predicted labels.

Parameters:

y\_true: array-like of shape (n\_samples,)

True labels.

y\_pred: array-like of shape (n\_samples,)

Predicted labels.

model\_name: str

Name of the model (for the plot title).

"""

cm = confusion\_matrix(y\_true, y\_pred)

plt.figure(figsize=(10, 7))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title(f'Confusion Matrix for {model\_name}')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

## \*\*Training Base model\*\*

from sklearn.pipeline import Pipeline

def train\_and\_evaluate\_models(x\_train, y\_train, x\_test, y\_test):

"""

Train and evaluate multiple classifiers without hyperparameters.

Parameters:

x\_train: Training feature set

y\_train: Training labels

x\_test: Testing feature set

y\_test: Testing labels

Returns:

dict: A dictionary containing evaluation metrics for each model.

"""

# Print shapes to debug the issue

print(f'x\_train shape: {x\_train.shape}')

print(f'y\_train shape: {y\_train.shape}')

print(f'x\_test shape: {x\_test.shape}')

print(f'y\_test shape: {y\_test.shape}')

models = {

'Logistic Regression': LogisticRegression(),

'Linear SVC': LinearSVC(),

'Multinomial NB': MultinomialNB(),

'Random Forest': RandomForestClassifier()

}

metrics = {

'Model': [],

'Accuracy': [],

'F1 Score (Macro)': [],

'Recall (Micro)': [],

'Precision (Micro)': [],

'ROC AUC': [],

'PR AUC': []

}

for name, model in models.items():

print(f'Starting Training for {name} classifier')

m = Pipeline([('vect', CountVectorizer(min\_df=5, ngram\_range=(1,2))),

('tfidf\_tweet\_multi', TfidfTransformer(norm='l2', sublinear\_tf=True)),

('model', model),

])

m.fit(x\_train, y\_train)

y\_pred = m.predict(x\_test)

if hasattr(m, "predict\_proba"):

y\_scores = m.predict\_proba(x\_test)

else: # use decision function for estimators that do not have predict\_proba

y\_scores = m.decision\_function(x\_test)

print(f"Model: {name}")

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred)}")

print(classification\_report(y\_test, y\_pred))

print("\n")

# Plot confusion matrix

plot\_confusion\_matrix(y\_test, y\_pred, name)

# Plot ROC curve and compute AUC

n\_classes = len(set(y\_train))

plot\_multiclass\_roc\_curve(y\_test, y\_scores, name, n\_classes)

roc\_auc = auc(\*roc\_curve(label\_binarize(y\_test, classes=[i for i in range(n\_classes)]).ravel(), y\_scores.ravel())[:2])

# Plot Precision-Recall curve and compute AUC

plot\_multiclass\_precision\_recall\_curve(y\_test, y\_scores, name, n\_classes)

pr\_auc = average\_precision\_score(label\_binarize(y\_test, classes=[i for i in range(n\_classes)]), y\_scores, average="micro")

metrics['Model'].append(name)

metrics['Accuracy'].append(accuracy\_score(y\_test, y\_pred) \* 100)

metrics['F1 Score (Macro)'].append(f1\_score(y\_test, y\_pred, average='macro') \* 100)

metrics['Recall (Micro)'].append(recall\_score(y\_test, y\_pred, average='micro') \* 100)

metrics['Precision (Micro)'].append(precision\_score(y\_test, y\_pred, average='micro') \* 100)

metrics['ROC AUC'].append(roc\_auc)

metrics['PR AUC'].append(pr\_auc)

return metrics

metric\_df\_tweet\_multi = train\_and\_evaluate\_models(x\_train, y\_train, x\_test, y\_test)

metric\_df\_tweet\_multi = pd.DataFrame(metric\_df\_tweet\_multi)

metric\_df\_tweet\_multi

# Pipeline and GridSearchCV with Hyperparameter

import pandas as pd

from sklearn.pipeline import Pipeline

from sklearn.model\_selection import GridSearchCV

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import LinearSVC

from sklearn.naive\_bayes import MultinomialNB

from sklearn.ensemble import RandomForestClassifier

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics import accuracy\_score, f1\_score, recall\_score, precision\_score, classification\_report

def train\_and\_evaluate\_models\_with\_pipeline\_hyper(x\_train, y\_train, x\_test, y\_test):

"""

Train and evaluate multiple classifiers using Pipeline and GridSearchCV to find the best hyperparameters.

Collect performance metrics for comparison.

Parameters:

x\_train: Training feature set

y\_train: Training labels

x\_test: Testing feature set

y\_test: Testing labels

Returns:

pd.DataFrame: DataFrame containing the performance metrics for each model.

"""

# Define the models and their hyperparameter grids

models = {

'Logistic Regression': (LogisticRegression(), {'clf\_\_C': [0.1, 1, 2, 10], 'clf\_\_max\_iter': [100, 500, 1000]}),

'Linear SVC': (LinearSVC(), {'clf\_\_C': [0.1, 1, 10]}),

'Multinomial NB': (MultinomialNB(), {'clf\_\_alpha': [0.1, 1, 10]}),

'Random Forest': (RandomForestClassifier(), {'clf\_\_n\_estimators': [100, 200, 300], 'clf\_\_max\_depth': [10, 15, 20]})

}

# Dictionary to store the metrics

metrics = {

'Model': [],

'Accuracy': [],

'F1 Score (Macro)': [],

'Recall (Micro)': [],

'Precision (Micro)': []

}

for name, (model, params) in models.items():

print(f'Starting Training for {name} classifier')

# Create a pipeline with the TF-Idf\_tweet\_multi Vectorizer and the classifier

pipeline = Pipeline([

('vect', CountVectorizer(min\_df=5, ngram\_range=(1,2))),

('tfidf\_tweet\_multi', TfidfTransformer(norm='l2',sublinear\_tf=True)),

('clf', model)

])

# Perform GridSearchCV with the pipeline

grid\_search = GridSearchCV(pipeline, param\_grid=params, cv=5, n\_jobs=-1)

grid\_search.fit(x\_train, y\_train)

best\_model = grid\_search.best\_estimator\_

y\_pred = best\_model.predict(x\_test)

# Plot confusion matrix

plot\_confusion\_matrix(y\_test, y\_pred, name)

metrics['Model'].append(name)

metrics['Accuracy'].append(accuracy\_score(y\_test, y\_pred) \* 100)

metrics['F1 Score (Macro)'].append(f1\_score(y\_test, y\_pred, average='macro') \* 100)

metrics['Recall (Micro)'].append(recall\_score(y\_test, y\_pred, average='micro') \* 100)

metrics['Precision (Micro)'].append(precision\_score(y\_test, y\_pred, average='micro') \* 100)

# Convert the metrics dictionary to a DataFrame

comparison\_df\_tweet\_multi = pd.DataFrame(metrics)

return comparison\_df\_tweet\_multi

comparison\_df\_tweet\_multi = train\_and\_evaluate\_models\_with\_pipeline\_hyper(x\_train, y\_train, x\_test, y\_test)

metric\_df\_tweet\_mult\_hyper = pd.DataFrame(comparison\_df\_tweet\_multi)

metric\_df\_tweet\_mult\_hyper

## \*Using Best model for Ensemble classifier\*

from sklearn.pipeline import Pipeline

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.ensemble import RandomForestClassifier, VotingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, f1\_score, recall\_score, precision\_score, classification\_report

def create\_and\_evaluate\_ensemble\_pipeline(x\_train, y\_train, x\_test, y\_test):

"""

This function creates a text processing and ensemble classification pipeline,

fits it to the training data, and evaluates it on the test data.

The pipeline performs the following steps:

1. Vectorizes the text data using CountVectorizer with n-grams (1,2) and min\_df\_tweet\_multi=5.

2. Transforms the count matrix into a TF-Idf\_tweet\_multi representation.

3. Applies an ensemble classifier consisting of:

- Multinomial Naive Bayes

- Random Forest Classifier

- Logistic Regression

- Support Vector Machine

The ensemble classifier uses soft voting and weighted averaging with specified weights.

Parameters:

x\_train (array-like or sparse matrix): Training data features (text).

y\_train (array-like): Training data labels.

x\_test (array-like or sparse matrix): Test data features (text).

y\_test (array-like): Test data labels.

Returns:

dict: Dictionary containing model performance metrics.

"""

# Define individual components of the ensemble

mnb = MultinomialNB()

rfc = RandomForestClassifier(n\_estimators=1000, max\_depth=12, random\_state=42)

lr = LogisticRegression(C=2, max\_iter=1000, n\_jobs=-1)

svc = SVC(probability=True)

# Create the VotingClassifier

ec = VotingClassifier(

estimators=[

('Multinomial NB', mnb),

('Random Forest', rfc),

('Logistic Regression', lr),

('Support Vector Machine', svc)

],

voting='soft',

weights=[1, 2, 3, 4]

)

# Create the pipeline

pipeline = Pipeline([

('vect', CountVectorizer(min\_df=5, ngram\_range=(1, 2))),

('tfidf\_tweet\_multi', TfidfTransformer()),

('model', ec)

])

# Fit the pipeline to the training data

pipeline.fit(x\_train, y\_train)

# Predict on the test data

y\_pred = pipeline.predict(x\_test)

# Calculate metrics

metrics = {

'Model': 'Ensemble Voting Classifier',

'Accuracy': accuracy\_score(y\_test, y\_pred),

'F1 Score (Macro)': f1\_score(y\_test, y\_pred, average='macro'),

'Recall (Micro)': recall\_score(y\_test, y\_pred, average='micro'),

'Precision (Micro)': precision\_score(y\_test, y\_pred, average='micro')

}

# Print accuracy and classification report

print(f"Accuracy: {metrics['Accuracy']}")

print(f"F1 Score (Macro): {metrics['F1 Score (Macro)']}")

print(f"Recall (Micro): {metrics['Recall (Micro)']}")

print(f"Precision (Micro): {metrics['Precision (Micro)']}")

print(classification\_report(y\_test, y\_pred))

return metrics

metric\_df\_tweet\_mult\_ensemble = pd.DataFrame([metrics\_ese])

metric\_df\_tweet\_mult\_ensemble

# \*\*Deep Learning Model\*\*

vocabulary\_size = 15000

max\_text\_len = 768

stemmer = SnowballStemmer('english')

stop\_words = [word for word in stopwords.words('english') if word not in ["my","haven't","aren't","can","no", "why", "through", "herself", "she", "he", "himself", "you", "you're", "myself", "not", "here", "some", "do", "does", "did", "will", "don't", "doesn't", "didn't", "won't", "should", "should've", "couldn't", "mightn't", "mustn't", "shouldn't", "hadn't", "wasn't", "wouldn't"]]

def preprocess\_text(text):

text = re.sub('[^a-zA-Z]', ' ', text)

words = text.lower().split()

words = [stemmer.stem(word) for word in words if not word in stop\_words]

cleaned\_text = ' '.join(words)

return cleaned\_text

df\_tweet\_multi['cleaned\_text'] = df\_tweet\_multi['text'].apply(preprocess\_text)

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

tokenizer.fit\_on\_texts(df\_tweet\_multi['processed\_text'].values)

le = len(tokenizer.word\_index) + 1

print(le)

sequences = tokenizer.texts\_to\_sequences(df\_tweet\_multi['processed\_text'].values)

X\_DeepLearning = pad\_sequences(sequences, maxlen=max\_text\_len)

XX\_train, XX\_test, y\_train, y\_test = train\_test\_split(X\_DeepLearning , df\_tweet\_multi['label'], test\_size=0.25, random\_state=42)

print((XX\_train.shape, y\_train.shape, XX\_test.shape, y\_test.shape))

# \*\*LSTM 1-Layer\*\*

epochs = 25

emb\_dim = 256

batch\_size = 50

model\_lstm1 = Sequential()

model\_lstm1.add(Embedding(vocabulary\_size,emb\_dim, input\_length=X\_DeepLearning.shape[1]))

model\_lstm1.add(SpatialDropout1D(0.8))

model\_lstm1.add(Bidirectional(LSTM(300, dropout=0.5, recurrent\_dropout=0.5)))

model\_lstm1.add(Dropout(0.5))

model\_lstm1.add(Flatten())

model\_lstm1.add(Dense(64, activation='relu'))

model\_lstm1.add(Dropout(0.5))

model\_lstm1.add(Dense(6, activation='softmax'))

model\_lstm1.compile(optimizer=tf.optimizers.Adam(),loss='categorical\_crossentropy', metrics=['acc'])

print(model\_lstm1.summary())

checkpoint\_callback = ModelCheckpoint(filepath="lastm-1-layer-best\_model.h5", save\_best\_only=True, monitor="val\_acc", mode="max", verbose=1)

early\_stopping\_callback = EarlyStopping(monitor="val\_acc", mode="max", patience=10, verbose=1, restore\_best\_weights=True)

reduce\_lr\_callback = ReduceLROnPlateau(monitor="val\_loss", factor=0.1, patience=5, verbose=1, mode="min", min\_delta=0.0001, cooldown=0, min\_lr=0)

callbacks=[checkpoint\_callback, early\_stopping\_callback, reduce\_lr\_callback]

from keras.utils import to\_categorical

# Convert target data to one-hot encoded format

y\_train\_encoded = to\_categorical(y\_train)

y\_test\_encoded = to\_categorical(y\_test)

from tensorflow.keras.backend import clear\_session

for \_ in range(3):

clear\_session()

history\_lstm1 = model\_lstm1.fit(XX\_train, y\_train\_encoded, epochs = epochs, batch\_size = batch\_size, validation\_data=(XX\_test,y\_test\_encoded), callbacks=callbacks)

results\_1 = model\_lstm1.evaluate(XX\_test, y\_test, verbose=False)

print(f'Test results - Loss: {results\_1[0]} - Accuracy: {100\*results\_1[1]}%')

acc = history\_lstm1.history['acc']

val\_acc = history\_lstm1.history['val\_acc']

loss = history\_lstm1.history['loss']

val\_loss = history\_lstm1.history['val\_loss']

plt.plot( acc, 'go', label='Train accuracy')

plt.plot( val\_acc, 'g', label='Validate accuracy')

plt.title('Train and validate accuracy')

plt.legend()

plt.figure()

plt.plot( loss, 'go', label='Train loss')

plt.plot( val\_loss, 'g', label='Validate loss')

plt.title('Train and validate loss')

plt.legend()

plt.show()

# Appendix B:

