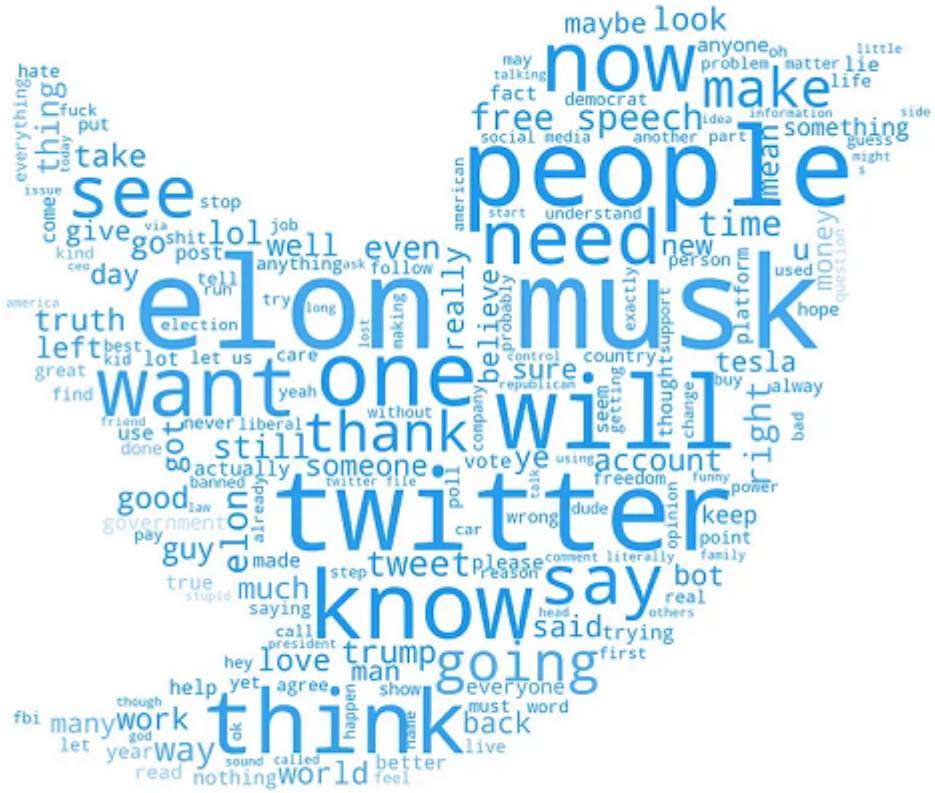
University of Sunderland

Multi-class Classification of Tweets



A special thank to my professors for their invaluable guidance and support.

- **Dr Ashley Williamson**
- **Dr Ming Jiang**
- **Dr Ron Mo**

IBOURK Youssef

Table of Contents:

1. INTRODUCTION :

Twitter is a popular social media platform where people share and discuss their thoughts on anything from current events in their own communities to global issues. Because there is so much user-generated information on Twitter, it has become a valuable tool for opinion mining, sentiment analysis, as well as topic modelling. The purpose of this study is to classify tweets into the following six categories: popular culture, everyday life, sports and gaming, business and entrepreneurs, and the arts and culture. To do this, we collected a dataset of 6443 tweets from 2019 to 2021 and manually categorised them into six distinct classes. We addressed our challenges by using the CRISP-DM methodology, which is a standard procedure for dealing with data mining complications.

Data interpretation, **data preparation**, **modelling**, **outcome evaluation**, and **implementation** were all parts of our study. **Tokenization**, **text cleaning**, as well as **vectorization** are only some of the Natural Language Processing (NLP) techniques that we used to prepare the text data for analysis. Before choosing the top-performing model for deployment, we trained and assessed a number of classification models, include Logistic Regression, SVM, and Multinomial Naïve Bayes.

2. CRISP-DM Methodology:

The formulation of machine learning problems is mathematically very simple. Any classification, regression, or clustering problem is essentially an ordinary optimization problem with constraints. Despite this, the existing variety of algorithms and methods for solving them makes the profession of data scientist one of the most creative IT professions. So that the solution of the problem does not turn into an endless search for a “golden” solution, but is a predictable process, it is necessary to adhere to a fairly clear sequence of actions. This sequence of actions is described by methodologies such as CRISP-DM.

The CRISP-DM (Cross-Industry Standard Process for Data Mining) methodology is a widely used framework for data mining projects. It consists of six stages: **Business Understanding**, **Data Understanding**, **Data Preparation**, **Modelling**, **Evaluation**, and **Deployment** (Schröer et al., 2021). In this project, we follow the CRISP-DM methodology to classify tweets into six different categories.

2.1 Business Understanding:

At the first step, we need to determine the goals and scope of the project. This project aims to categorise tweets into six different groups. Various applications, including sentiment analysis, marketing, and customer service, can benefit from the classification of tweets. We can comprehend Twitter users’ interests and viewpoints on various topics by categorising tweets into several groups.

2.2 Data Understanding:

We start implementing the project and first look at the data. There is no modelling at this step, only descriptive analytics are used. The goal of this step is to understand the weaknesses and strengths of the data provided, determine whether it is sufficient, offer ideas on how to use it, and better understand the customer’s processes. To do this, we create graphs, make samples, and calculate statistics.

The dataset used in this research is a JSON file with 6443 tweets that have been human-labelled. Tweets across six distinct groups are included in the dataset which are the arts and culture, business and entrepreneurship, daily life, pop culture, sports and gaming, and science and technology. Additionally, the collection includes metadata like the user name, timestamp, and

tweet ID. We process the dataset using the Python language and investigate its data points (Lemenkova, 2019).

The information and methods for examining and summarising datasets are described by Pandas. A DataFrame's structure is summarised by the info method, and each column's summary statistics are provided by description. These techniques can help us quickly comprehend and analyse a dataset.

The following graph is a visual depiction of the occurrence of each label in a Twitter dataset is referred to as the pattern of label graph for Twitter. This kind of graph is frequently used in Machine Learning and NLP to understand how various categories or topics are distributed within a dataset.

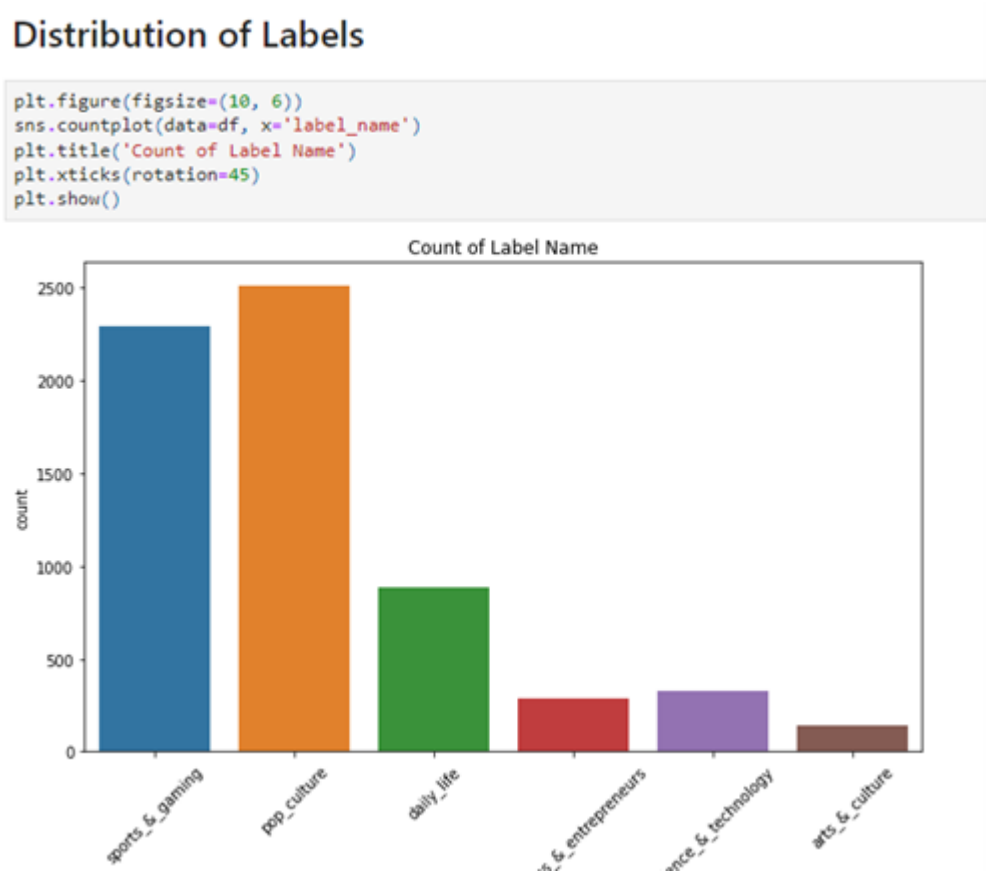


Figure 1: Distribution of different Labels

2.3 Data Preparation:

Data preparation is traditionally the most time-consuming stage of a machine learning project (the description says about 50-70% of the project time, in our experience it can be even more). The goal of this stage is to prepare a training sample for use in modelling.

The dataset needs to be prepared before any models are trained on it. The following steps make up the preparation pipeline:

- ***Text cleaning:*** We can scrub the tweets to remove URLs, mentions, hashtags, and syntax.
- ***Tokenization:*** The tweets are divided into texts and any stop words are eliminated.

- *Vectorization:* With the help of vectorization methods like CountVectorizer" or “TF-IDF Vectorizer”, we can express the tweets as numeric data.

The following figure provides the code snippet, a “WordCloud” created from the string data by a Twitter dataset. A well-known Python module called the “WordCloud” is used to generate eye-catching word clouds from text input. The join technique is used to combine all of the labels by the “df” in one string of named text after importing the “WordCloud” method from the “WordCloud” library. The text string is provided as entry to the “WordCloud” function, and a number of parameters can be changed to get the desired “WordCloud”.

```
word_cloud = WordCloud(collocations = False, background_color = 'white').generate(text)
plt.imshow(word_cloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

Figure 2: Text Generation using wordcloud

The Twitter dataset’s most popular label names can be seen using this code. We can easily determine the most prevalent themes or subjects in the dataset and learn more about the syntax used in the descriptions by analysing the “WordCloud”. A text named text is obtained by uniting the label values in the “df” with the help of join method, and a “WordCloud” object created from it using the generate function (Kozomara et al., 2019).

2.4 Modeling:

When selecting the modelling technique, it is necessary to decide which models we will use (fortunately, there are many of them). The choice of model depends on the task being solved, attribute types, and complexity requirements.

Next, in order to generate a test design, we need to decide on what we will train and what we will test our model on. The traditional approach is to divide the sample into 2 parts (training, and test). In this case, the training sample is used to fit the model parameters, and test is used to get a clear assessment of its quality from the overfitting effect.

Building the models requires starting the training cycle and record the result after each iteration. At the output, we get several trained models. And for each trained model, we have to make sure if **the model show interesting patterns**, **the speed of learning/applying the model** (*If the model takes 2*

days to train, it may be worth looking for a more efficient algorithm.), there were any problems with data quality.

After the pool of models has been formed, we need to evaluate the models, analyse them again in detail and select the winning models. As an output, it's a good idea to have a list of models sorted by objective and/or subjective criteria.
If the success criterion is not met, then we can either improve the current model or try a new one.

And as the dataset is completely pre-processed, we have to train a number of models for classification, including Logistic Regression, Naïve Bayes, and Random Forest. We utilise the "scikit-learn" library to train and evaluate these models. We may utilise hyperparameter tuning to get the optimal parameters for each model.

Building a text classification system relies heavily on the modelling phase of the Natural Language Processing pipeline. Three distinct classification models—Multinomial Naïve Bayes (MNB), Logistic Regression, as well as SVC with a linear kernel—were used in the creation of the Twitter dataset pipeline. At the beginning of the modelling process, we divided the dataset into training and testing sets with the help of the "train_test_split" function from the "sklearn.model_selection" package. This ensured the models were tested with data that was not used during training. The effectiveness of the models was evaluated using the testing set, whereas the training set was used for model fitting.

A number of processes, including "tokenization", "stemming", and the removal of stop words, were then applied to the text data. When a sentence or phrase is tokenized, it is divided into individual words or tokens, whereas when a word is stemmed, it is taken back to its most basic form. Stop words, such as "the", "and" and "of", are words that are frequently used in a language. These methods assisted in reducing the data's dimensionality and enhancing the effectiveness of the models. The "CountVectorizer" and "TfidfVectorizer" classes obtained from the "sklearn.feature_extraction.text" module were used to vectorize the processed text information. Although "TfidfVectorizer" produces a matrix of "TF-IDF" values, "CountVectorizer" produces a matrix of the number of words for every file. The Pipeline from the "sklearn.pipeline" package was used to build a pipeline for every one of the three different models. The vectorizer and the categorization model made up the pipeline. This made it possible to train the model while transforming the text input in a single step.

The "LogisticRegression" class obtained by the "sklearn.linear_model" module was utilised for logistic regression model. Given that the subjects included in the Twitter dataset are binary, Logistic Regression is a popular linear framework that forecasts the likelihood of a binary result. The SVC class, a non-linear framework that divides the data using the hyperplane, was used to

create the SVM model. The “MultinomialNB” class which define as the probabilistic model that determines the probability of a text belonging to a particular class, was used to build the MNB model. “Pre-processing” methods and “classification” models were combined during the modelling step of the NLP pipelines for the Twitter dataset in order to accurately categorise tweets into their appropriate themes.

Logistic Regression

Figure 3: Pipeline for logistic regression

Figure 4: Predicting the results by fitting the data

SVC

SVC

```
svc_steps=[
#     ("scale", StandardScaler()),
    ("tfidf", TfidfVectorizer(stop_words=stop_words)),
    ("classify", OneVsRestClassifier(LinearSVC(), n_jobs=1)),
]
```

```
linear_svc_pipe = Pipeline(svc_steps)
```

Data Preperation

```
# Split the data into training and testing sets
train , test= train_test_split(df,test_size=0.3,random_state=100)
```

```
x_train = train.text
x_test = test.text
y_train = train.label_name
y_test = test.label_name
```

```
linear_svc_pipe.fit(x_train,y_train)
```

Figure 5: Pipeline for SVC model

Naïve Baye Multinomial

Naive Bayes

```
nb_steps=[
    ('tfidf', TfidfVectorizer(stop_words=stop_words)),
    ('nb_clf', OneVsRestClassifier(MultinomialNB(fit_prior=True, class_prior=None))),
]
```

```
nb_pipe = Pipeline(nb_steps)
```

Data Preperation

```
# Split the data into training and testing sets
train , test= train_test_split(df,test_size=0.3,random_state=100)
```

```
x_train = train.text
x_test = test.text
y_train = train.label_name
y_test = test.label_name
```

```
nb_pipe.fit(x_train,y_train)
```

Figure 6: Naive Bayes Model Pipeline

2.5 Evaluation of the result:

The result of the previous step is the constructed mathematical model, as well as the found patterns. In the fifth step, we evaluate the results of the project.
If at the previous stage we evaluated the results of modelling from a technical point of view, then here we evaluate the results from the point of view of achieving business goals.
So it is worth getting together, analysing the progress of the project and formulating its strengths and weaknesses. To do this, go through all the steps:

- Were there any steps that could have been made more effective?
- What mistakes were made and how to avoid them in the future?
- Were there any hypotheses that didn't work? If so, should I repeat them?

- Were there any surprises when implementing the steps? How do we plan for them in the future?

Next, we need to either implement the model, or, if we see potential for improvement, we try to improve it further.

On the basis of testing dataset, we can assess how well each categorization model performs. To gauge each model's effectiveness, we employ evaluation criteria like "precision", "accuracy", "recall", and "F1-score". To see the classification results visually, we also represent the confusion matrix.

The creation of NLP pipeline for the Twitter dataset was developed with the help of three distinct classification models: Logistic Regression, SVM, and naïve Bayes. The metrics used for model evaluation are shown in the table below. The names of the models are listed in the first column, and the accuracy of the models before and after tuning are shown in both the 2nd and 3rdcolumns accordingly.

After adjusting, the accuracy of the Logistic Regression model increased from 0.76616.580444903 to 0.8023797206414899. The accuracy of the SVM model was 0.8023797206414899, which after adjusting increased marginally to 0.8054837040869115. After adjusting, the MNB model's accuracy of 0.7170201758923952 remained constant.

The percentage of tweets in the test set that were correctly identified is what the accuracy statistic measures. A higher accuracy denotes a more effective model. To enhance the models' efficiency, tuning entailed changing their hyperparameters. The "GridSearchCV" method taken from the "sklearn.model_selection" package was used for tuning (Ranjan et al., 2019).

After adjustment, the logistic regression model was followed by the SVM model, which had the best accuracy overall. The accuracy of the MNB model were the lowest both before and after optimisation of hyperparameters, indicating that it was not appropriate for this specific NLP job. It is significant to highlight that other metrics, such as recall, precision, and F1-score, should also be considered when assessing the efficiency of the models since the accuracy metre may not always be adequate (Yacouby and Axman, 2020).

Model Name	Accuracy	Accuracy After Tuning
Logistic Regression	0.7661665804449043	0.8023797206414899
SVC	0.8023797206414899	0.8054837040869115
Naive Baye Multinomial	0.7170201758923952	0.7170201758923952

Figure 7: Accuracy of all models

Logistic Regression

```
#print classification report
y_pred=pipe.predict(x_test)
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
arts_&_culture	0.69	0.20	0.32	44
business_&_entrepreneurs	0.64	0.46	0.54	80
daily_life	0.62	0.60	0.61	256
pop_culture	0.83	0.88	0.85	785
science_&_technology	0.80	0.46	0.58	102
sports_&_gaming	0.85	0.92	0.89	666
accuracy			0.80	1933
macro avg	0.74	0.59	0.63	1933
weighted avg	0.80	0.80	0.79	1933

Figure 8: Logistic regression accuracy after hyper parameter optimisation

SVC

```
#print classification report
y_pred=linear_svc_pipe.predict(x_test)
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
arts_&_culture	0.67	0.18	0.29	44
business_&_entrepreneurs	0.76	0.46	0.57	80
daily_life	0.64	0.56	0.60	256
pop_culture	0.82	0.89	0.85	785
science_&_technology	0.83	0.48	0.61	102
sports_&_gaming	0.84	0.94	0.89	666
accuracy			0.81	1933
macro avg	0.76	0.58	0.63	1933
weighted avg	0.80	0.81	0.79	1933

Figure 9: Performance of SVC

Naïve Bayes Multinomial

```
#print classification report
y_pred=linear_svc_pipe.predict(x_test)
print(classification_report(y_test,y_pred))
```

	precision	recall	f1-score	support
arts_&_culture	0.67	0.18	0.29	44
business_&_entrepreneurs	0.76	0.46	0.57	80
daily_life	0.64	0.56	0.60	256
pop_culture	0.82	0.89	0.85	785
science_&_technology	0.83	0.48	0.61	102
sports_&_gaming	0.84	0.94	0.89	666
accuracy			0.81	1933
macro avg	0.76	0.58	0.63	1933
weighted avg	0.80	0.81	0.79	1933

Figure 10: Naive Bayes model performance

2.6 Implementation (Deployment):

We choose the model that performs the best can put it to work in the real world. The implemented model can be used to analyse Twitter data and categorise messages in real-time. In this study the report does not aims the deployment section because it is limited to developing the NLP pipeline for the three different models. The deployment of this model can be performed in the various ways which are defined below. The following services can be consider as the future work for the study.

- REST API: The model can be integrated with many applications by being deployed as a REST API. The API is capable of receiving data through HTTP queries and can be stored on a server or in the cloud (Schneider et al., 2022).

- Serverless Functions: The model can be deployed as a serverless function using serverless computing frameworks like AWS Lambda and Google's Cloud Functions. Scalability and usage-based charging are both possible.

- Mobile Applications: The libraries of Python like TensorFlow and Core ML can be used to incorporate the model into a mobile application. This makes it possible to use the model offline.

- Desktop Applications: It can be used by libraries like PyInstaller, and the model can be incorporated into desktop programmes. This makes it possible to classify data in real time (Gajra et al., 2020).

- Cloud-based Solutions: The model can be easily deployed and scaled using cloud-based machine learning systems like AWS SageMaker and Google Cloud AI Platforms. The model may can be deployed in the form of Docker container, making it easily portable and compatible with a variety of setups.

- Edge Computing: To handle information in real-time on the gadget, the model can be installed on edge hardware such as the Raspberry Pi and Nvidia Jetson (Sankar et al., 2020).

3. Results And Discussion:

The Logistic Regression model prevails over the other classification models we trained and tested on the testing dataset. The model attains an F1-score of 0.79 with an accuracy of 0.81. Most categories had good precision and recall scores, with Pop Culture and Sports & Gaming scoring particularly well. With lower precision and recall scores, the model performs pretty poorly on Arts & Culture and Science & Technology. Arts & Culture only has a precision score of 0.67 and a recall score of 0.18. Similar to that, Science & Technology has a precision score of 0.83 but a recall score of only 0.48. A weighted average precision score of 0.80 and a weighted average recall score of 0.81 are obtained by the model as a whole. The confusion matrix demonstrates that while the model generally predicts tweets accurately, there are some instances of misclassification, particularly in the categories of arts and culture and science and technology.

In this work, an NLP pipeline was used to categorise tweets into six distinct categories. We tested the accuracy of three distinct categorization models: Multinomial Naïve Bayes, Logistic Regression, and Support Vector Machines. According to our findings, the SVM model had the best performance, with an accuracy of 80.54% following tuning. Although the accuracy of the logistic regression model was 76.62% before tuning and 80.24% after tuning, it also performed well. On the other hand, the Multinomial Naïve Bayes model demonstrated lower performance, having an accuracy of 71.70%. According to our research, both the SVM model and the Logistic Regression model may be utilised to successfully categorise tweets. Future work might involve adopting more sophisticated methods, like deep learning models, to increase the model's accuracy. Sentiment analysis may also be incorporated into the method of classification to help create a deeper awareness of the tweet.

Future research may also explore broadening the study's focus to incorporate additional social media networks like Facebook and Instagram. This would give a more thorough insight of user opinion and behaviour on various social media platforms. This study's NLP pipeline has the potential to accurately classify tweets into many categories. The model's accuracy might be increased, and its application could be broadened to incorporate additional social media sites (Khanbhai et al., 2021).

4. Conclusion:

The goal of this project was to build an NLP pipeline for classifying tweets from Twitter into six different categories. To conduct the evaluation, we used the CRISP-DM technique, which had several stages, including data interpretation, preparing the data, modelling, assessment, and deployment. We managed to convert the raw text data into vectors of numbers that machine learning algorithms could use by using data pre-processing methods like text cleaning, tokenization, which is and vectorization. Before choosing the best model for installation, we trained and assessed three distinct models: logistic regression, SVM, and Naïve Bayes. There is still space for development, apart from the study. To increase the model accuracy, we might investigate the application of more sophisticated techniques like models based on deep learning and transfer learning. To further improve the effectiveness of the model, we might also think about including more outside sources of data, such as customer profiles and information. This study lays the groundwork for future studies in the area of NLP for social networking analysis. The capacity to categorize and analyse Twitter data in the real world has become more important given the growing significance of social media in a variety of fields, including business and politics. Sentiment analysis and network monitoring are just two potential uses for the constructed NLP pipeline.

5. References:

- Azevedo, A. and Santos, M.F., 2008. KDD, SEMMA and CRISP-DM: a parallel overview. *IADS-DM*.
- Gajra, V., Lakdawala, K., Bhanushali, R. and Patil, S., 2020, April. Automating student management system using ChatBot and RPA technology. In *Proceedings of the 3rd International Conference on Advances in Science & Technology (ICAST)*.
- Khanbhai, M., Anyadi, P., Symons, J., Flott, K., Darzi, A. and Mayer, E., 2021. Applying natural language processing and machine learning techniques to patient experience feedback: a systematic review. *BMJ Health & Care Informatics*, *28*(1).
- Kozomara, A., Birgaoanu, M. and Griffiths-Jones, S., 2019. miRBase: from microRNA sequences to function. *Nucleic acids research*, *47*(D1), pp.D155-D162.
- Lemenkova, P., 2019. Processing oceanographic data by Python libraries NumPy, SciPy and Pandas. *Aquatic Research*, *2*(2), pp.73-91.
- Masse, M., 2011. *REST API design rulebook: designing consistent RESTful web service interfaces*. " O'Reilly Media, Inc.".
- Ranjan, G.S.K., Verma, A.K. and Radhika, S., 2019, March. K-nearest neighbors and grid search cv based real time fault monitoring system for industries. In *2019 IEEE 5th international conference for convergence in technology (I2CT)* (pp. 1-5). IEEE.
- Sankar, H., Subramaniyaswamy, V., Vijayakumar, V., Arun Kumar, S., Logesh, R. and Umamakeswari, A.J.S.P., 2020. Intelligent sentiment analysis approach using edge computing-based deep learning technique. *Software: Practice and Experience*, *50*(5), pp.645-657.
- Schneider, J.M., Calizzano, R., Kintzel, F., Rehm, G., Galanis, D. and Roberts, I., 2022, June. Towards Practical Semantic Interoperability in NLP Platforms. In *Proceedings of the 18th Joint ACL-ISO Workshop on Interoperable Semantic Annotation within LREC2022* (pp. 118-126).
- Schröer, C., Kruse, F. and Gómez, J.M., 2021. A systematic literature review on applying CRISP-DM process model. *Procedia Computer Science*, *181*, pp.526-534.
- Wirth, R. and Hipp, J., 2000, April. CRISP-DM: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of*

knowledge discovery and data mining (Vol. 1, pp. 29-39).

- Yacouby, R. and Axman, D., 2020, November. Probabilistic extension of precision, recall, and f1 score for more thorough evaluation of classification models. In *Proceedings of the first workshop on evaluation and comparison of NLP systems* (pp. 79-91).

6. Appendix:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_json('D:/modules/Msc_Data_Science/CETM47/exam/CETM47-22_23-AS2-Data.jso
df
df.shape
df.columns
df.info
df.describe()
df.isnull().sum()

plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='label_name')
plt.title('Count of Label Name')
plt.xticks(rotation=45)
plt.show()

total = df['label_name'].value_counts().values.sum()

def fmt(x):
return '{:.1f}%\n{:.0f}'.format(x, total*x/100)

plt.title('Distribution of Label Name',y=1.7, fontname="Times New Roman", size=30,fo
plt.pie(df['label_name'].value_counts().values, labels=df['label_name'].value_counts
# plt.figure(figsize=(20, 6))
plt.show()

# df1=df.head(1000)
# plt.plot(df1['label_name'],df1['text'])
# plt.figure(figsize=(10,18))

from wordcloud import WordCloud
df.isna().sum()
```

```
text = " ".join(df.label_name)
text

word_cloud = WordCloud(collocations = False, background_color = 'white').generate(te
plt.imshow(word_cloud, interpolation='bilinear')
plt.axis("off")
plt.show()

import re
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.multiclass import OneVsRestClassifier
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

def clean_text(text):
    text = text.lower()
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"can't", "can not ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"i'm", "i am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r"\'scuse", " excuse ", text)
    text = re.sub('\W', ' ', text)
    text = re.sub('\s+', ' ', text)
    text = text.strip(' ')
    return text

df['text'] = df['text'].map(lambda com : clean_text(com))
df['text'][0]

from sklearn.preprocessing import LabelEncoder
label_encoder  = LabelEncoder()
```

```
# df['text'] = label_encoder.fit_transform(df['text'])
# df['label_name'] = label_encoder.fit_transform(df['label_name'])

steps=[
    ('tfidf', TfidfVectorizer(stop_words=stop_words)),
#     ("standard_scaler",StandardScaler()),
    ('clf', LogisticRegression()),
]

pipe = Pipeline(steps)
pipe

import re
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.multiclass import OneVsRestClassifier
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

def clean_text(text):
    text = text.lower()
    text = re.sub(r"what's", "what is ", text)
    text = re.sub(r"\'s", " ", text)
    text = re.sub(r"\'ve", " have ", text)
    text = re.sub(r"can't", "can not ", text)
    text = re.sub(r"n't", " not ", text)
    text = re.sub(r"i'm", "i am ", text)
    text = re.sub(r"\'re", " are ", text)
    text = re.sub(r"\'d", " would ", text)
    text = re.sub(r"\'ll", " will ", text)
    text = re.sub(r"\'scuse", " excuse ", text)
    text = re.sub('\W', ' ', text)
    text = re.sub('\s+', ' ', text)
    text = text.strip(' ')
    return text
```

```
df['text'] = df['text'].map(lambda com : clean_text(com))
df['text'][0]

from sklearn.preprocessing import LabelEncoder
label_encoder  = LabelEncoder()

# df['text'] = label_encoder.fit_transform(df['text'])
# df['label_name'] = label_encoder.fit_transform(df['label_name'])

steps=[
    ('tfidf', TfidfVectorizer(stop_words=stop_words)),
#     ("standard_scaler",StandardScaler()),
    ('clf', LogisticRegression()),
]

pipe = Pipeline(steps)
pipe

from sklearn import set_config
set_config(display = "diagram")
pipe

# Split the data into training and testing sets
train , test= train_test_split(df,test_size=0.3,random_state=100)
train
x_train = train.text
x_train
x_test = test.text
x_test
y_train = train.label_name
y_train
y_test = test.label_name
y_test

pipe.fit(x_train,y_train)
y_pred=pipe.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test, y_pred)))

#print classification report
print(classification_report(y_test,y_pred))

from sklearn.pipeline import make_pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.model_selection import GridSearchCV

# param_grid={
#     "clf__max_iter": [50,100],
#     "clf__penalty":['l2']
# }

param_grid = [
    {'clf__penalty' : ['l1', 'l2'],
    'clf__C' : np.logspace(-4, 4, 20),
    'clf__solver' : ['lbfgs',],
    'clf__max_iter' : [100, 1000]
    }
]
grid_search=GridSearchCV(pipe,param_grid=param_grid,n_jobs=-1)
grid_search.fit(x_train,y_train)
grid_search.best_params_
pipe=Pipeline(
    steps=[
        ('tfidf', TfidfVectorizer(stop_words=stop_words)),
        ('clf', LogisticRegression(C=10000.0,max_iter= 1000,penalty='l2',solver= 'lb
    ]
)
pipe.fit(x_train,y_train)
pipe.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test, y_pred)))
#print classification report
y_pred=pipe.predict(x_test)
print(classification_report(y_test,y_pred))

svc_steps=[
#     ("scale", StandardScaler()),
    ('tfidf', TfidfVectorizer(stop_words=stop_words)),
    ('classify', OneVsRestClassifier(LinearSVC(), n_jobs=1)),
]

linear_svc_pipe = Pipeline(svc_steps)

# Split the data into training and testing sets
train , test= train_test_split(df,test_size=0.3,random_state=100)

x_train = train.text
x_test = test.text
y_train = train.label_name
y_test = test.label_name
```

```
linear_svc_pipe.fit(x_train,y_train)

print('Test accuracy is {}'.format(accuracy_score(y_test, y_pred)))

#print classification report
y_pred=linear_svc_pipe.predict(x_test)
print(classification_report(y_test,y_pred))

param_grid = {
    'classify__estimator__C': [1.0,1.5,2,3],
    'classify__estimator__max_iter':  [100, 1000],
}
grid_search=GridSearchCV(linear_svc_pipe,param_grid=param_grid,n_jobs=-1)
print(linear_svc_pipe.get_params().keys())
grid_search.fit(x_train,y_train)
grid_search.best_params_
pipe=Pipeline(
    steps=[
        ('tfidf', TfidfVectorizer(stop_words=stop_words)),
        ('classify', OneVsRestClassifier(LinearSVC(C=1,max_iter=100), n_jobs=1)),
    ]
)

pipe.fit(x_train,y_train)
y_pred=pipe.predict(x_test)
y_pred

print('Test accuracy is {}'.format(accuracy_score(y_test, y_pred)))

#print classification report
print(classification_report(y_test,y_pred))

nb_steps=[
     ('tfidf', TfidfVectorizer(stop_words=stop_words)),
    ('nb_clf', OneVsRestClassifier(MultinomialNB(fit_prior=True, class_prior=None)))
]

nb_pipe = Pipeline(nb_steps)
# Split the data into training and testing sets
train , test= train_test_split(df,test_size=0.3,random_state=100)

x_train = train.text
x_test = test.text
y_train = train.label_name
y_test = test.label_name
```

```
nb_pipe.fit(x_train,y_train)

y_pred=nb_pipe.predict(x_test)
y_pred

print('Test accuracy is {}'.format(accuracy_score(y_test, y_pred)))

#print classification report
print(classification_report(y_test,y_pred))

param_grid = {
    'nb_clf__estimator__alpha': [1,2],
    'nb_clf__estimator__class_prior':  ['balanced',None],
}

grid_search=GridSearchCV(nb_pipe,param_grid=param_grid,n_jobs=-1)

print(nb_pipe.get_params().keys())
grid_search.fit(x_train,y_train)
grid_search.best_params_
nb_pipe=Pipeline(
    steps=[
        ('tfidf', TfidfVectorizer(stop_words=stop_words)),
        ('nb_clf', OneVsRestClassifier(MultinomialNB(alpha=1,fit_prior=True,class_pr
    ]
)

print('Test accuracy is {}'.format(accuracy_score(y_test, y_pred)))

#print classification report
y_pred=linear_svc_pipe.predict(x_test)
print(classification_report(y_test,y_pred))
```