INTRODUCTION

Social media is a powerful data source for many applications since it has become a significant platform for communication and information sharing. Businesses, researchers, and individuals who need to get insights into people's thoughts, attitudes, and behaviours depend on the ability to classify social media information into many categories automatically. One of the most popular uses of machine learning on social media data is emotion classification, which includes determining the sentiment or emotion portrayed in a text.

This project aims to build a Machine Learning model to identify the emotion the tweets portray on ‘Russia and Ukraine’. The very first step of the project is preparing a dataset for the same. We will use Python’s snscrape library to collect and label tweets with Hugging Face’s BERT API. The tweets will be labelled into six emotions: sadness, joy, love, anger, fear, and surprise.

After preparing the dataset, we will clean the tweets for better results and accuracy. For this purpose, we would use libraries like regex and features like stop words, stemming, etc. Once the tweets are cleaned, we use the TF-IDF (Term Frequency-Inverse Document Frequency) technique to convert the text into numerical features that our machine-learning model can use. We will be using Support Vector Machine Learning Model in our project.

LITERATURE SURVEY

1. “Emotion Classification of Social Media Texts Using Deep Learning” [LINK]:

This paper presents a deep learning approach to emotion classification of social media texts. The authors propose a model that combines a convolutional neural network (CNN) and a long short-term memory (LSTM) network to capture both local and global contextual information. The model is trained on a dataset of social media texts and evaluated using several performance metrics.

2. “LSTM, VADER and TF-IDF based Hybrid Sentiment Analysis Model for Twitter Data” [LINK]:

This paper presents a hybrid sentiment analysis model for Twitter data that combines LSTM, VADER, and TF-IDF. The authors evaluate their model on a dataset of tweets and show that it outperforms several baseline models. The paper also discusses the challenges of sentiment analysis on social media data and the potential applications of the proposed model.

3. “Emotion Detection in Text: A Review” [LINK]:

This paper provides a comprehensive review of the literature on emotion detection in text. The authors discuss various approaches to emotion detection, including rule-based methods, machine learning methods, and deep learning methods. The paper also covers various feature extraction techniques and evaluation metrics used in emotion detection research.

4. “A review on sentiment analysis and emotion detection from text” [LINK]:

This paper provides an overview of sentiment analysis and emotion detection from text. The authors discuss various levels of sentiment analysis, emotion models, and the process of sentiment analysis and emotion detection from text. The paper also covers the challenges faced during sentiment and emotion analysis and potential future directions for research.

METHODOLOGY

The methodology of this project can be divided into the steps mentioned below:

i) Data Collection and Labelling:

We used the snscrape library of Python to collect 15000 tweets on the topic 'Russia Ukraine'. After collecting the tweets, we used a pre-trained model (BERT) to label our dataset into six emotions: joy, anger, sadness, fear, surprise and love.

Colab Notebook Link for Data Collection and Labelling

ii) Data Preprocessing:

Data preprocessing is an important step in preparing data for analysis or modelling. It involves several techniques to clean, transform and organise data so that machine learning algorithms can easily use it. In the case of text data, several preprocessing steps are commonly used to clean and transform text data, such as tweet cleaning, removing emojis, changing to lowercase, removing stop words, stemming, spell check, and expanding acronyms to long text.

The data processing techniques used in this project are as follows:

a) **Tweet cleaning:** Twitter data is unique and requires specific cleaning techniques. Tweet cleaning includes removing URLs, user mentions (@username), hashtags (#RussiaUkarine), and other non-alphabetic or non-numeric characters that do not add value to the analysis. We have done tweet cleaning using the regex library of Python.

b) **Removing emojis and special symbols:** Emojis are graphic symbols commonly used in social media and messaging platforms. Emojis can convey emotions and add context to the text but can also cause problems when analysing text data. Removing emojis can simplify the text data and make it easier to analyse. It is also implemented using regex.

c) **Changing to lowercase:** Text data may contain uppercase, lowercase, or mixed-case words. Changing all text to lowercase can help reduce the vocabulary size and make comparing and analysing text easier.

d) **Remove stop words:** Stop words are commonly used words that do not contribute to the meaning of a sentence or document. Examples of stop words include "a", "an", "the", "in", "of", "to", and "is". Removing stop words can reduce the vocabulary size and improve the analysis's efficiency. It is implemented using nltk library of Python.

e) **Stemming:** Stemming is reducing words to their root form or stem. For example, the word "jumping" can be stemmed to "jump". This can help to reduce the size of the vocabulary and improve the efficiency of the analysis. It is implemented using SnowballStemmer function of the nltk library in Python.

f) **Spell check:** Text data may contain misspelt words, which can cause data analysis problems. Spell checking can correct misspelt words and improve the accuracy of the analysis. It is implemented using TextBlob library of Python.

g) **Acronyms to long text:** Text data may contain acronyms, which can be confusing or difficult to understand. Expanding acronyms to their long text form can help improve the text data's readability and understanding.

iii) Tokenization:

Tokenisation is the next step after the data is processed and cleaned. Tokenisation involves splitting a text document into individual words or sentences. In word tokenisation, a text document is split into individual words, separated by spaces, punctuation marks, or other delimiters. In sentence tokenisation, a text document is split into individual sentences, separated by punctuation marks such as periods, question marks, or exclamation marks.

In this project, we have used the nltk library of Python for tokenisation.

After data cleaning and tokenisation, the sentence:

‘What a wonderful lifee !!’ would become ‘wonder’, ‘life’.

iv) Feature Extraction:

After preprocessing the data, the next step is to extract features from the tokenised and stemmed text data. The TfidfVectorizer algorithm converts the text data into a matrix of TF-IDF features.

TF-IDF stands for "Term Frequency-Inverse Document Frequency" and is a statistical measure used to evaluate the relevance of a term in a document or a corpus of documents.

TF-IDF is calculated by multiplying two factors: the term frequency (TF) and the inverse document frequency (IDF).

i) Term Frequency (TF) measures how often a term appears in a document. It is calculated as the number of times a term appears in a document divided by the total number of terms in that document.

ii) Inverse Document Frequency (IDF) measures a term's importance in the entire corpus. It is calculated as the logarithm of the total number of documents in the corpus divided by the number of documents in which the term appears.

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

The result is a score representing a term's relevance to a specific document within a larger corpus. Higher TF-IDF scores indicate terms that are more relevant to a document.

Considering an example:

	apples	are	ate	good	red	I
I ate apples. Red Apples are good apples	3	1	1	1	1	1
Apples are red	1	1	0	0	1	0

TF(apple) = 3
IDF(apples) = log(2/2) = 0
Therefore TFIDF of apples in 1st sentence is 0

Since the TFIDF of apples has a lower value, it tells us that it is present in many documents and is not essential for our model.

Whereas for good, the TFIDF is 0.301 is useful for our analysis.

v) Train-Test Split:

The next step is to split the dataset into training and testing sets. This is done to evaluate the machine learning model's performance on unseen data. The train-test split is done using the train_test_split function from scikit-learn.

vi) Training the Model:

The next step is to train the machine learning model on the training dataset. In this case, a Support Vector Machine (SVM) classifier is used to classify the tweets into their respective emotions. SVM is a popular algorithm used for text classification tasks as it can handle high-dimensional feature spaces and effectively deals with non-linearly separable data.

We will be using RBF (Radial Basis Function) kernel of SVM. The RBF kernel is a non-linear kernel that transforms the input data into a higher-dimensional feature space to make the data more separable.

The RBF kernel is defined as:

K(x, x') = exp(-gamma ||x - x'||^2)

where x and x' are input data points, gamma is a hyperparameter that controls the width of the kernel, and ||x - x'||^2 is the squared Euclidean distance between the input data points.

For our model, we have considered the value of gamma to be 0.8

Colab Notebook Link for the ML Model

ALGORITHM

The project algorithm follows a multi-step process with the following key aspects:

1. Import and load the data into a data frame using Pandas to ensure proper formatting and readiness for processing.
2. Visualize the data using Matplotlib and Seaborn to identify trends or patterns in the data and inform subsequent steps in the process.
3. Create a mapping dictionary that maps each emotion to a numeric value to ensure that the data is in a format that can be easily processed by the algorithm. Any necessary changes are also made during this step.
4. Clean the tweets by removing URLs, mentions, retweet symbols, emojis, and other unwanted elements to ensure that the data is as clean and accurate as possible.
5. Further process the tweets by removing stop words, expanding acronyms, and correcting spelling mistakes to ensure that the data is in a format easily understood by the algorithm.
6. Perform tokenization and stemming to break the tweets down into smaller parts and help the algorithm better understand the data.
7. Split the data into training and test sets, and fit and transform both sets using the TFIDF vectorizer to help the algorithm better understand the data and identify any patterns or trends that may exist.
8. Applying RDF SVM to the training dataset allows the algorithm to learn from the data and identify any patterns or trends.
9. Evaluate the algorithm's performance using several metrics, including accuracy score, precision, recall, F1 score, and confusion matrix to ensure that the algorithm is performing as expected and that the data is being correctly analyzed.

STEPS OF ALGORITHM

Detailed step-by-step implementation of the algorithm is demonstrated in the further pages:

FLOW CHARTS

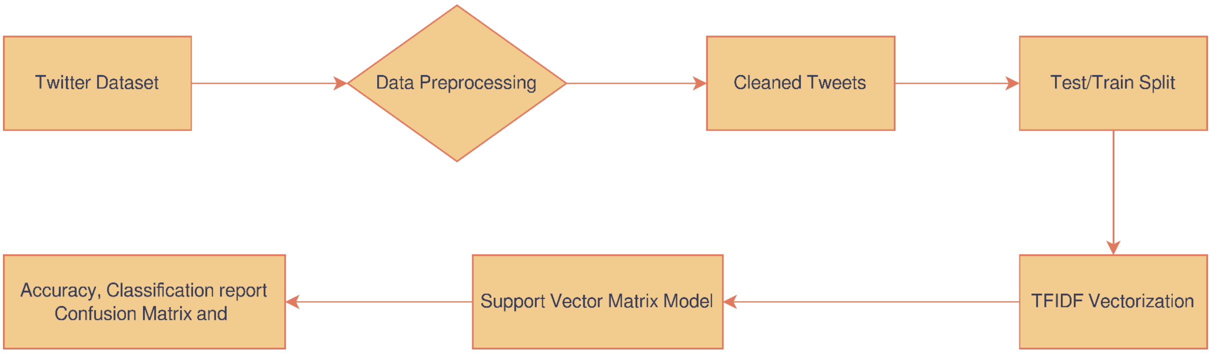

Fig 1: Basic Working of the Model

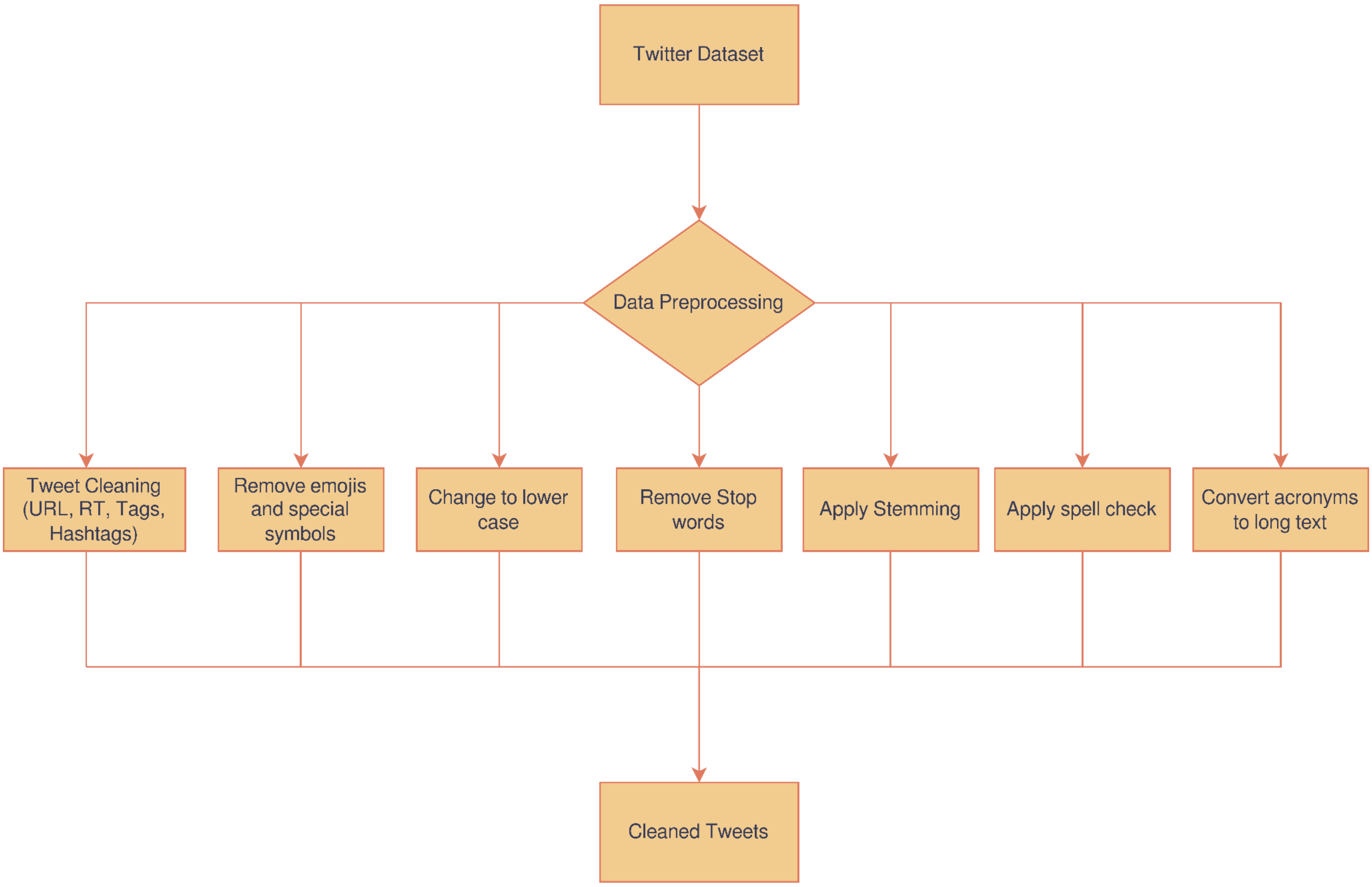

Fig 2: Data Preprocessing

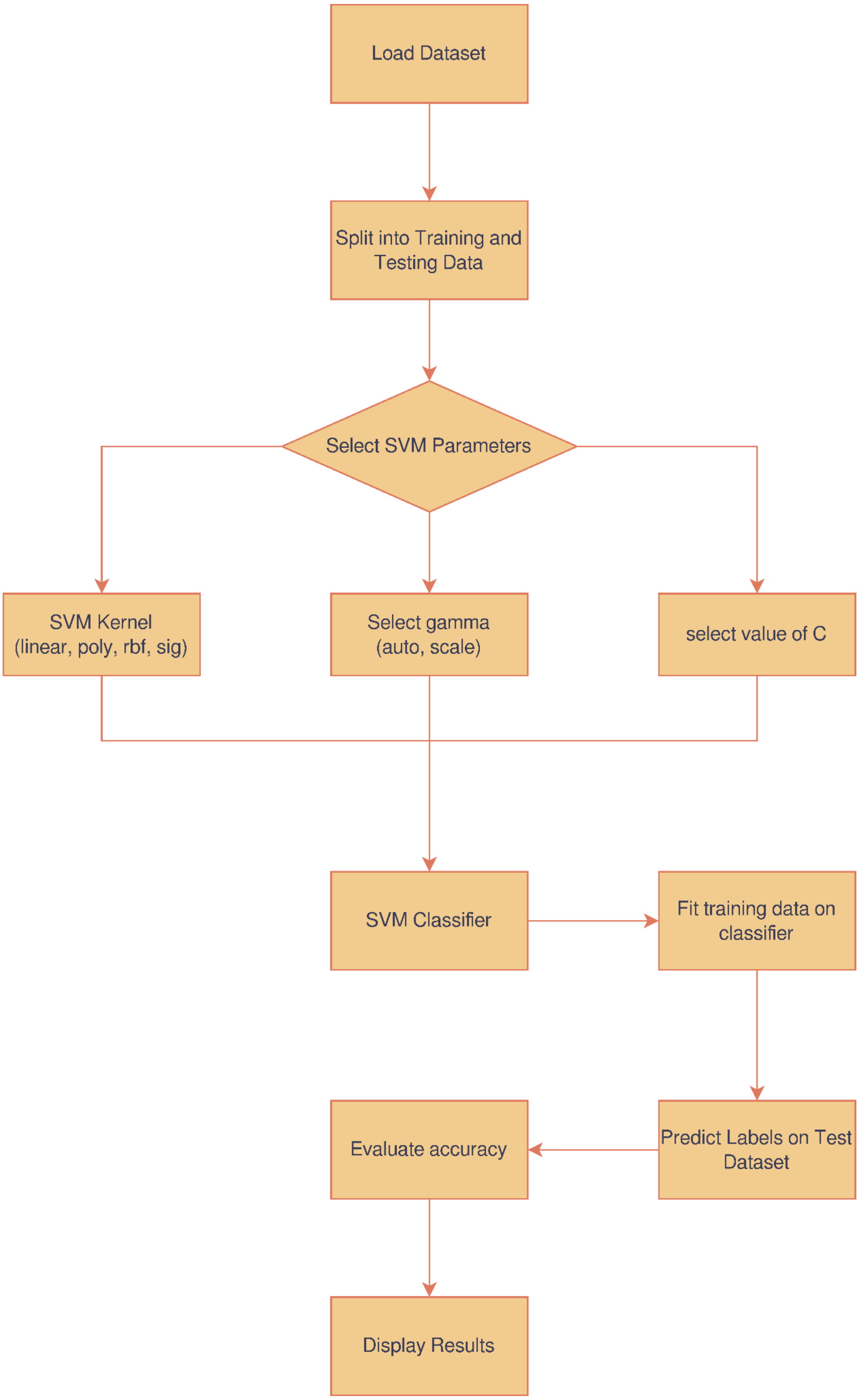
Load Dataset
Split into Training and Testing Data
Select SVM Parameters
SVM Kernel (linear, poly, rbf, sig)
Select gamma (auto, scale)
select value of C
SVM Classifier
Fit training data on classifier
Predict Labels on Test Dataset
Evaluate accuracy
Display Results

Fig 3: SVM Model

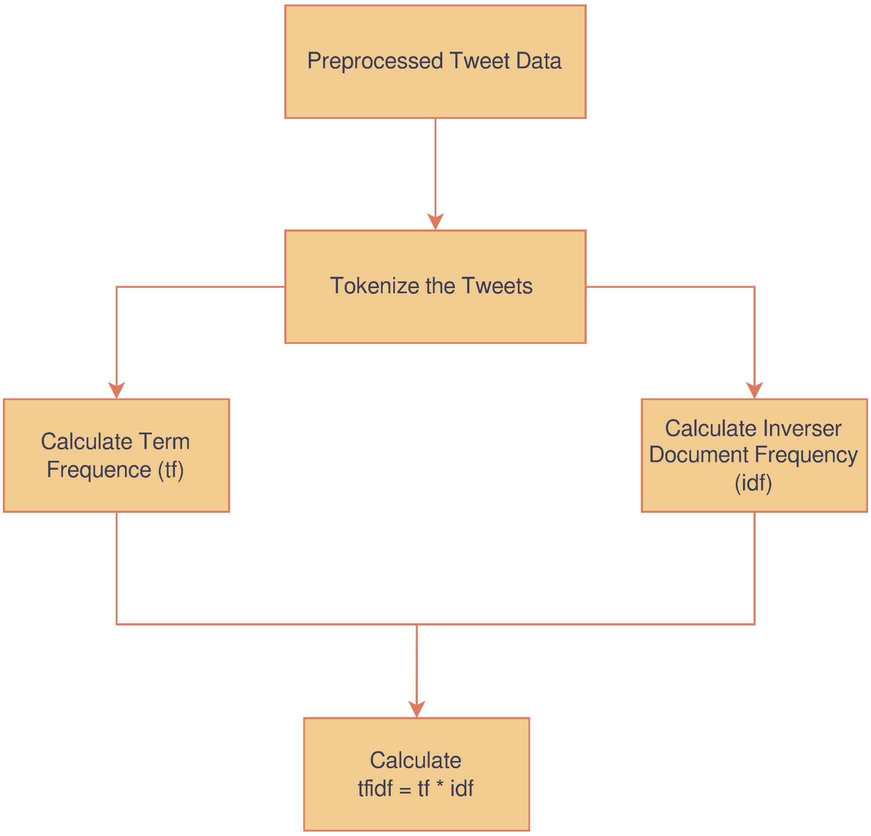
Preprocessed Tweet Data
Tokenize the Tweets
Calculate Term Frequence (tf)
Calculate Inverser Document Frequency (idf)
Calculate tfidf = tf * idf

Fig 4: TFIDF Algorithm

RESULTS AND FUTURE SCOPE

RESULT:

We have obtained an accuracy of 99% on our training data and 67% on our testing data using over 15000 tweets on the topic Russia and Ukraine.

FUTURE SCOPE:

We can use RNN (Recurrent Neural Network) using LSTM (Long Term Short Memory) to boost our accuracy. Recurrent Neural Networks (RNNs) are a type of neural network that can handle sequential data, making them well-suited for analysing natural language text data such as tweets. However, traditional RNNs have difficulty retaining long-term dependencies, which can be a problem when dealing with text data that requires an understanding of context.

This is where Long Short-Term Memory (LSTM) comes in. LSTMs are a type of RNN specifically designed to address the issue of vanishing gradients and retain long-term dependencies. By incorporating LSTMs into our neural network model, we can potentially improve the accuracy of our predictions, especially when dealing with text data that requires an understanding of context over time.