BMCS2123 NATURAL LANGUAGE PROCESSING

202311 Session, Year 2023/24

Assignment Documentation

Project Title: Sentiment Analysis on Out-Of-Vocabulary (OOV) Malaysia Rojak Language
Programme: RDS Y2S2
Tutorial Group: 2
Tutor: Ms. Fatin Izzati binti Ramli

Team members' data

No	Student Name	Student ID	Module In Charge
1	TAY WEI RONG	22WMR03594	LSTM Model Multinomial NB
2	LIM JO SUN	22WMR01705	TF-IDF SVM Truncated SVM

Table of Contents

1.0 Introduction

1.1 Problem Statement

In the diverse linguistic landscape of Malaysia, the Rojak language, also known as the Malaysian mixed language, emerges as a captivating expression of cultural amalgamation (Scaraceni, 2010). This linguistic fusion is not only a product of language dynamics but also reflects the amalgamation of Malaysia's diverse populace, including Malays, Chinese, Indians, and other minority groups, but subtly influences the language choices and communication styles of the citizens (Salinaamran, 2022). Rojak uniquely expresses Malaysia's rich linguistic landscape as a trade language, embodying cultural diversity and fluidity. It serves as a medium through which the nuances of Malaysia's multicultural identity are communicated—a linguistic bridge transcending individual language preferences.

Linguistically, the Rojak language integrates two pivotal components: code-switching and code-mixing. These terms denote common speech patterns observed among bilingual individuals, especially in multilingual and multicultural environments (Waris, 2012). Code-switching, the natural alternation between two or more languages within a bilingual context, allows for seamless transitions and mutual comprehension among listeners. For instance, in casual conversation: "Apa khabar? Did you catch the latest?" demonstrated a smooth switch between Malay and English words for a natural flow. Conversely, code-mixing occurs when bilingual individuals incorporate words, phrases, or sentences from different languages within the same conversation, both in speech and writing. Consider a tweet: "Had a super productive meeting sama colleagues. Semangat dengan project baru!" showcases a blend of English and Malay, conveying a message with linguistic diversity.

The convergence of different linguistic elements in the Rojak language creates an exciting landscape, especially in the digital world of social media. The use of online platforms was widespread after the emergence of the Internet, providing a unique opportunity to explore the feelings conveyed through Malaysian Rojak Language expressions. The fluid interplay between code-switching and code-mixing, evident in the vast expanse of online communication, adds complexity to our understanding of emotions embedded in this unique linguistic blend. As we navigate social media's digital intricacies of the Rojak language, it becomes increasingly clear that there is a need to decode and analyse the sentiments it encapsulates.

1.2 Objectives

In the evolving landscape of social media, the Malaysian Rojak Language presents a unique linguistic tapestry, blending various dialects, colloquialisms, and expressions. This study, drawing from extensive social media data from platforms like Twitter and Reddit, aims to delve into the intricacies of this language, particularly in the context of sentiment analysis.

1. Comprehensive Analysis of Sentiment Expressions

The primary objective is to conduct an exhaustive analysis of sentiment expressions found in the Malaysian Rojak Language dataset. This involves a detailed examination of the various shades and intensities of sentiments—positive, negative, and neutral. By scrutinizing the dataset's rich collection of linguistic expressions, the study seeks to identify contextual triggers that elicit different emotions, interpret subtle shifts in tone, and unearth underlying patterns within these sentiments. The intention is to paint a comprehensive picture of sentiment manifestation in the Rojak Language, revealing not just the overt expressions but also the hidden nuances that influence the emotional landscape of these communications.

2. Enhancing Cultural Understanding

A crucial aspect of this research is to move beyond mere sentiment classification and delve into the cultural nuances embedded within the Rojak Language. By understanding the emotions and expressions unique to this linguistic blend, the study aims to foster a deeper comprehension of the values and social dynamics inherent in the language. This exploration is not limited to linguistic analysis; it also seeks to preserve and highlight the cultural identity that the Rojak Language represents. In doing so, the study aspires to contribute to the preservation of Malaysia's rich cultural heritage, ensuring that the language is not only analyzed but also appreciated in its cultural context.

3. Improving Model Robustness

Another critical aim is to enhance the robustness of sentiment analysis models, especially in dealing with out-of-vocabulary linguistic forms characteristic of the Rojak Language. This language, with its eclectic mix of regional expressions and colloquialisms, poses unique challenges for existing sentiment analysis models. Therefore, the study focuses on developing and refining methodologies capable of navigating the complexities of the Rojak Language. This objective extends beyond improving sentiment analysis for this particular language; it contributes to the broader field of natural language processing. By tackling the unique challenges presented by the Rojak Language, the study aims to advance our ability to handle diverse linguistic variations and, in turn, develop more accurate and inclusive sentiment analysis systems.

In summary, this study is poised at the intersection of linguistic analysis and cultural understanding. By exploring the sentiment dynamics within the Malaysian Rojak Language, it aims to shed light on the emotional undercurrents of this linguistic phenomenon. Furthermore, it endeavors to enhance the cultural appreciation of this language and improve the robustness of sentiment analysis models in dealing with its complexities. This comprehensive approach promises not only to enrich our understanding of the Rojak Language but also to contribute significantly to the field of natural language processing.

1.3 Motivations

The motivation behind conducting this sentimental analysis can be encompassing both cultural relevance and market insight.

1.3.1 Cultural Relevance and Diversity

This sentiment analysis endeavor explores the diverse array of cultural subtleties ingrained in Malaysian society. This endeavour seeks to provide an immersive trip into the hearts and thoughts of people expressing themselves through the Rojak Language, giong beyond a simple sentiment analysis. Our goal in deciphering these language expressions is to reveal the complex web of value, emotions and societal nuances present in this milieu rich in culture. This is a chance to delve deeply into the core of Malaysian cultural identity as it is encapsulated in language, rather than just skim the surface. By means of this investigation, we hope to not only comprehend feelings but also to conserve the rich cultural legacy embedded in the Rojak Language, cultivating a deeper appreciation.

1.3.2 Market and Community Insights

Beyond examining cultural differences, the analysis of attitudes conveyed in the Rojak Language has the ability to shed light on consumer behaviour, public opinion and societal trends. Comprehending the feelings and ideas expressed in this combination of languages provides a distinct advantage for companies looking to enter the Malaysian market. Businesses can use it to identify customer emotions, preferences, and behavioural patterns, which gives them an advantage in customising their goods and marketing tactics to appeal to their target market more successfully. Furthermore, this analysis provides a deeper understanding of public perspectives on a range of societal concerns for policymakers, scholars and community leaders. It acts as a guide for well-informed decision-making, making it easier to develop communication plans and policies that are consistent with the ideas and principles conveyed in the Malaysian Rojak Language. This promotes more inclusivity and community involvement.

2.0 Literature Review

2.1 Sentiment Analysis

According to Tonkin (2016), sentiment analysis, also known as opinion mining, is a specific task in the field of opinion extraction, which aims to distinguish sentiments and emotions expressed in textual data. In other words, sentiment analysis is a process conducted to identify people's feelings towards an entity, including products, services, organisations, individuals, issues, events, topics and their attributes, through a piece of text (Liu, 2012; Henrickson et al., 2019). Similarly, according to Onyenwe et al. (2020), sentiment analysis refers to a process of extracting information and semantics from text and determining the writer's attitude. It usually relies on applying machine learning algorithms to classify documents based on a collection of features extracted from the text using other Natural Language Processing (NLP) techniques (Henrickson et al., 2019). The concept of opinion itself is broad, but typically, sentiment analysis will categorise texts according to 3 polarity classes: positive, negative and neutral (Tonkin, 2016). Notably, the sentiment level in the analysis is currently limited to three classes. However, there's potential for diversification in the dataset, incorporating nuances like agreement or disagreement, as well as providing a more detailed range of sentiments such as good or bad, along with a rating scale, which could provide richer meanings, capturing a deeper understanding in the writer's intention (Prabowo & Thelwall, 2009; Nandwani & Verma, 2021). Moreover, beyond the polarity level analysis, there is also emotional detection which measures emotional states from the content and basically covers 6 states: pleasure, anger, disgust, sadness, fear and surprise (Ho, 2020).

2.2 Level of Sentiment Analysis

Sentiment analysis can be approached at different levels of granularity, each providing a varying depth of understanding about the sentiments expressed in the textual content. There are three levels of analysis: Document-level, Sentence-level and Aspect or Entity-based Sentiment Analysis (Su et al., 2023).

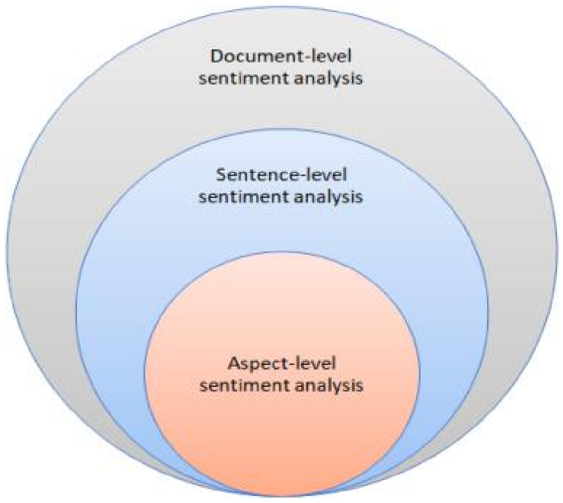

Document-level sentiment analysis
Sentence-level sentiment analysis
Aspect-level sentiment analysis

Figure 1. Levels of sentiment analysis. Reprinted from "A comprehensive survey on sentiment analysis: Approaches, challenges and trends" by M. Birjali, M. Kasri, and A. Beni-Hssane, 2021, *Knowledge-Based System*, Levels and applications of sentiment analysis section, Figure 2 (https://doi.org/10.1016/j.knosys.2021.107134)

Document-level Sentiment Analysis

According to Pang et al. (2002), at this stage, classification will be employed for the purpose of analysing the overall sentiment or single opinion within a document. It specifically examines indications of favourable or unfavourable sentiments toward a single product (Liu, 2012). Also, the analysis assumes that each document should express only an opinion with respect to a single entity, excluding coverage for documents assessing or comparing multiple entities. However, it could be beneficial in understanding the general sentiment of articles, reviews or long-form content (Balaji et al, 2017).

Sentence-level Sentiment Analysis

At the sentence level, also known as the pharisee level, each document is broken down into sentences in order to determine the general polarity expressed, whether positive or negative (Liu, 2012; Su et al., 2023). While none of them, then it is neutral, which usually is not an opinion at all. However, according to Hamed et al. (2016), the neutral class in sentiment analysis reflects ambiguity by capturing instances where the sentiment analysis expressed is neither clearly positive nor negative and enhances model accuracy by providing a better distinction between positive and negative. Also, neutral feedback indicates that it highlights the improvement area when it comes to customer analysis. According to Wiebe et al. (1999, as cited in Liu, 2012), this level of analysis is related to subjectivities classification, which makes it possible to distinguish between sentences expressing factual information from those in which objective views and opinions are expressed. However, it is essential to note that subjectivity is not equivalent to sentiment, as many objective sentences can be interpreted as opinions. In short, this analysis stage will allow for a more nuanced interpretation of sentiments within a document.

Entity and Aspect Level Sentiment Analysis

As stated by Nandwani & Verma (2021), opinion about a specific feature is predicted at the aspect stage of sentiment analysis. In the same vein, Liu (2012), in his book Sentiment Analysis and Opinion Mining, noted that instead of looking at language constructs such as documents, paragraphs, sentences, clauses or phrases, the aspect level will directly examine the opinion itself. It is based on the concept that an opinion will consist of sentiment and a target. According to theory, the understanding of the problem will be useful in analysis when people realise that opinion objectives are important. Thus, the goal of aspect-level analysis is to identify sentiments on entities which are explicit and implicit (Liu, 2012; Nandwani & Verma, 2021). It is valuable for product reviews, where sentiments about different product features are analysed separately. In an example provided by Nandwani & Verma (2021), aspect-level sentiment analysis evaluates different viewpoints of a product, for instance, speed and cost. Speed, if explicitly mentioned in the text, is an explicit aspect, while cost consumed occasionally is implicit and harder to identify. Therefore, analysing implicit features might add complexity to aspect-level sentiment analysis.

2.3 Sentiment Analysis Approaches

In the paper of Nandwani and Verma (2021), techniques for sentiment analysis can be classified into 5 types: Lexicon-based approach, Machine Learning-based approach, Deep Learning-based approach, Transfer Learning, and Hybrid Approach.

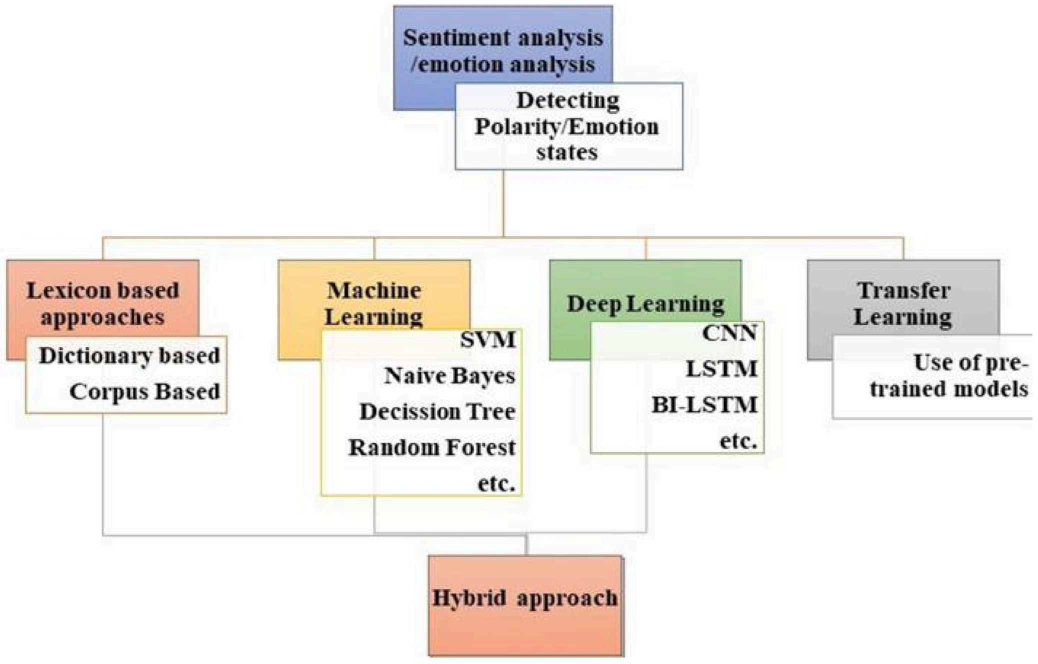


Figure 2. Techniques for Sentiment Analysis. From "A review on sentiment analysis and emotion detection from text" by P. Nandwani and R. Verma, 2021, *Social network analysis and mining, 11*(1), Background section, Figure 4 (https://doi.org/10.1007/s13278-021-00776-6)

2.3.1 Lexicon-based Approach

In this approach, a dictionary of positive and negative words will be maintained, which assigns a sentiment value to each positive and negative word (Nandwani & Verma, 2021). In order to calculate the overall sentiment of a sentence or document, it must then be derived from the sum or mean of these values. The statement is supported by Birjali et al. (2021), who stated that the lexicon approach relies on an opinion lexicon, which is a predefined set of words associated with positive or negative orientations through scores like +1, -1 or 0, in representing positive, negative or neutral. By tokenising the document and assigning lexicon-based sentiment values, algorithms like summation and averaging can be performed to determine the overall sentiment of the document. Regarding the strengths, the lexicon-based approach is straightforward and practical at the sentence and phrase level of sentiment analysis (Nandwani & Verma, 2021). Moreover, it is resource-effective without requiring extensive training datasets.

Dictionary-based and corpus-based approaches are two forms of lexicon-based methods in sentiment analysis that rely on sentiment lexicons. The dictionary-based approach presumes that synonymous words have the same sentiment polarities as antonyms, whereas antonyms have the opposite polarities (Birjali et al., 2021; Nandwani & Verma, 2021). In other words, this method relies on the assumption that words with related meanings convey similar sentiments while opposite meanings imply opposing sentiments. However, it lacks domain specificity. The corpus-based approach overcomes this by incorporating domain-specific sentiment words and assigning polarity labels based on context and domain. However, it poses a challenge in generalisation; it heavily relies on domain-specific data, and its performance might not extend well beyond the specific context it was trained on.

2.3.2 Machine Learning-based Approach

Machine learning-based approaches involve training models to predict sentiment based on features extracted from textual data. For the purpose of training and evaluating models, the entire dataset is typically divided into two parts: a training set and a testing set (Nandwani & Verma, 2021). The training dataset serves as the information source for training the model, providing detailed information about the instances of an item (Shobha & Rangaswamy, 2018). Subsequently, the testing dataset is employed to assess the effectiveness of the model trained on the training dataset (Nandwani & Verma, 2021). In the realm of sentiment analysis, machine learning algorithms commonly follow a supervised classification approach. Various algorithms used for sentiment classification encompass Naïve Bayes, support vector machine (SVM), and decision trees, among others, each with its own set of advantages and drawbacks.

In machine learning-based approaches, they can typically be classified into two learning methods: supervised machine learning and unsupervised machine learning. According to Bangert (2021), supervised learning is a machine learning method characterised by the utilisation of a labelled dataset. This approach involves training algorithms with labelled inputs and outputs, enabling the model to accurately classify data or predict outcomes. On the other hand, unsupervised machine learning employs algorithms to analyse and cluster an unlabeled dataset, discovering hidden patterns without human intervention. In sentiment analysis, labelled training data is used to train a supervised learning model to classify text into predefined sentiment categories such as positive, negative, or neutral (Hauthal et al., 2020). Common supervised learning algorithms used in sentiment analysis include Support Vector Machines (SVM), Naive Bayes, and Decision Trees. In contrast, in unsupervised learning, as with the general model, patterns or groupings within the data emerge without predefined categories. Clustering methods such as k-means cluster analysis can group similar text segments based on their features, identifying different emotional patterns within the data. It is important to note that both supervised and unsupervised learning methods have their own advantages and limitations. Supervised learning requires labelled training data, which can be time-consuming and costly to obtain, but it often yields more accurate results. Unsupervised learning, while not requiring labelled data, may be more prone to ambiguity and may require more manual interpretation of the results. In text classification, sometimes it is difficult to create labelled training documents, but it is easy to collect unlabeled documents.

2.3.3 Deep Learning-based Approach

Deep learning, a subset of machine learning, operates through neural networks with a multi-layered structure that comprehends data at multiple levels of abstraction (LeCun et al., 2015). Initially, neural networks comprised three layers: input, hidden, and output. However, incorporating multiple hidden layers renders the network 'deep,' empowering it to capture intricate relationships. The depth of these networks is defined by the number of hidden layers, signifying deep learning as a specialised subset within machine learning. Unlike traditional methods reliant on manual feature selection, deep learning models autonomously extract features, notably enhancing accuracy and performance (Dang et al., 2020). Recent advancements have propelled deep learning methodologies ahead of established machine learning techniques, especially within various Natural Language Processing (NLP) domains like computer vision, entity recognition, and speech recognition (Rani & Kumar, 2018).

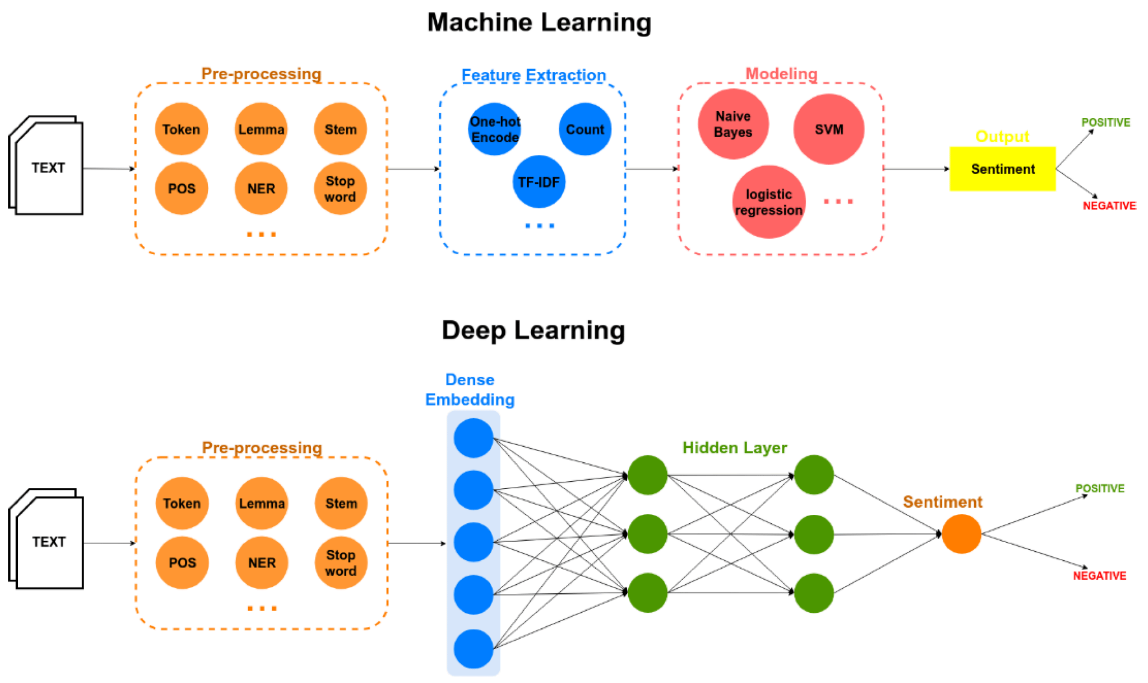


Figure 3. Differences between Machine Learning and Deep Learning. From "Sentiment Analysis Based on Deep Learning: A Comparative Study" by N. C. Dang, M. N. Moreno-García, and F. De la Prieta, 2020, *Electronics, 9*(3), Background section, Figure 1 (https://doi.org/10.3390/electronics9030483). CC BY.

In the context of sentiment analysis, deep learning algorithms can automatically learn to represent the complex and subtle features of the language, allowing them to capture the nuances of sentiment in text. Recently, deep learning has gained popularity as a practical approach to sentiment analysis (Unat & Aktaş, 2018). It has been shown through experiments that sentiment classification models based on deep learning algorithms have achieved excellent classification performance (Unat & Aktaş, 2018; Dang et al., 2020). This section will explore various forms of deep learning techniques in sentiment analysis.

2.3.3.1 Deep Neural Networks (DNN)

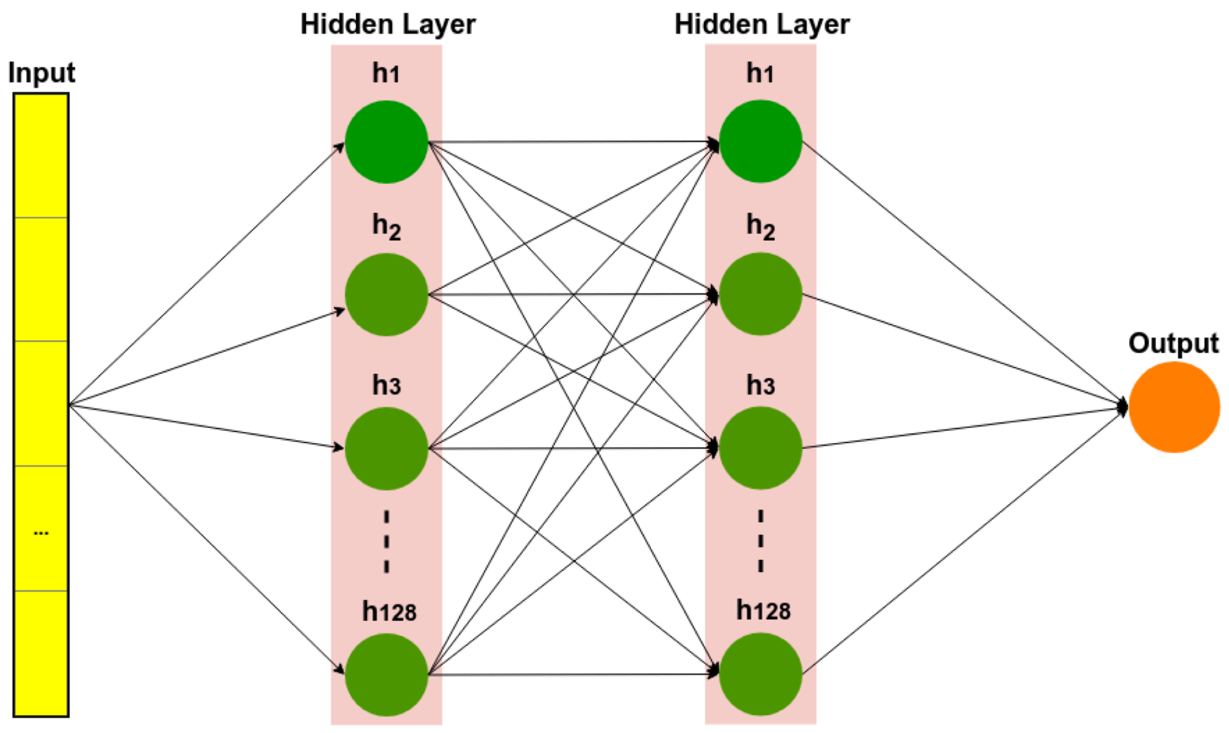


Figure 4. Deep Neural Networks (DNNs). From "Sentiment Analysis Based on Deep Learning: A Comparative Study" by N. C. Dang, M. N. Moreno-García, and F. De la Prieta, 2020, *Electronics, 9*(3), Background section, Figure 2 (https://doi.org/10.3390/electronics9030483). CC BY.

Deep Neural Networks (DNNs), also known as convolutional networks, are sophisticated deep learning models commonly used in sentiment analysis (Schmidhuber, 2014; Dang et al., 2020). These networks are structured with multiple layers of artificial neurons designed to extract increasingly abstract features from input data (Bengio et al., 2007; Krizhevsky et al., 2012). The hierarchical nature of DNNs allows them to learn data representations in layers, capturing both low-level and high-level features relevant to sentiment analysis (Dean et al., 2012). Hinton et al. (2006) demonstrated that deeper architectures, pre-trained with unsupervised learning algorithms at each layer and fine-tuned via supervised learning using back-propagation, achieve notably superior results. Research by Hinton et al. (2006) showcased that deeply structured networks, pre-trained via unsupervised learning and refined with back-propagation, yield superior results, especially surpassing Gaussian Mixture Models (GMM) and Hidden Markov Models (HMM) in speech processing tasks.

A paper from Sze et al. (2017) emphasised the exceptional accuracy of DNNs in image and speech recognition, owing to their capacity to extract high-level features from vast datasets, deviating from traditional approaches reliant on manually crafted features. Similarly, Hassani Niaki et al. (2022) highlighted the DNN's remarkable accuracy in predicting PC material tensile strength compared to other strengths, underscoring the effectiveness of deep learning even with limited datasets. Moreover, Huang et al. (2016) affirmed the DNN's exceptional feature expression ability, highlighting its capability to fit complex mappings and express features better than shallow neural networks.

However, Sze et al. (2017) noted that despite their accuracy, DNNs demand efficient processing techniques due to their computational complexity. This complexity necessitates strategies to enhance energy efficiency and throughput without compromising accuracy, a critical factor in expanding DNN deployment across domains. Similarly, Toohey et al. (2021) highlighted another challenge, which is the black-box nature of DNNs. These models often operate as complex black boxes, making it challenging to interpret their decision-making processes.

In short, while DNNs demonstrate remarkable accuracy and feature expression capabilities, addressing computational complexity, energy efficiency, and interpretability remains essential for their broader application across various domains.

2.3.3.2 Convolutional neural networks (CNNs)

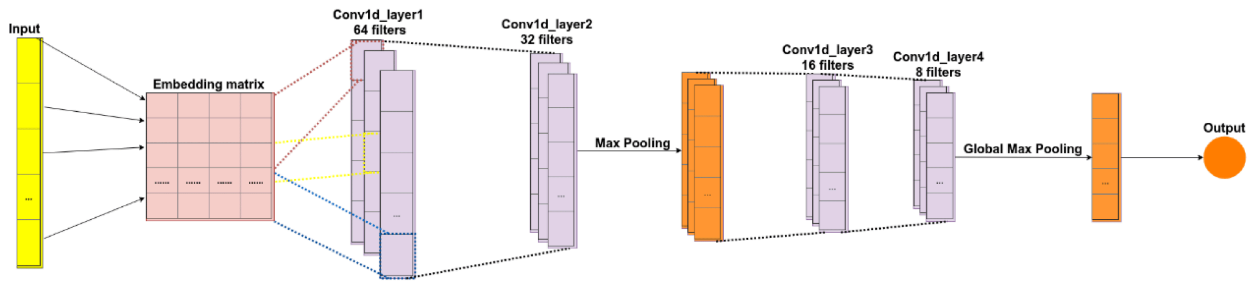


*Figure 5.*Convolutional Neural Network (CNNs). From "Sentiment Analysis Based on Deep Learning: A Comparative Study" by N. C. Dang, M. N. Moreno-García, and F. De la Prieta, 2020, *Electronics, 9*(3), Background section, Figure 3 (https://doi.org/10.3390/electronics9030483). CC BY.

Convolutional Neural Networks (CNNs) are a type of feedforward neural network designed for processing image-like inputs, as they excel in capturing translation invariance, allowing it to recognise features in an image regardless of their specific location (Teuwen, 2020). The architecture of CNN is more efficient than regular neural networks, reducing the number of parameters and making it easier to optimise and less dependent on data size. The framework involves various layers, with the most common building blocks being the convolution layer, pooling layer, and fully connected layers, which act as feature extractors, dimensionality reducers, and classifiers, respectively (Dang et al., 2020; Yadav & Vishwakarma, 2019). The convolutional layer produces outputs using its filter or kernel values. These outputs undergo a convolution operation, and the Pooling Layer reduces the representation's size to speed up computation.

CNNs are typically used for image processing tasks and offer better accuracy than common algorithms, as proven in Baischer et al.'s paper (2021). However, they can also be adapted and used for text classification tasks, although they are not the most typical choice for text-related tasks. In text classification, CNNs treat text as a kind of signal and apply convolution operations to capture the local dependencies within the text (Amin & Nadeem, 2018). For instance, they can process word embeddings (vector representations of words) to identify essential patterns or features (like phrases or word combinations) for classification. Pre-trained word embeddings like Word2Vec, GloVe, or FastText can be used to convert words into dense vectors, used as CNN input, and treated as image channels (Amin & Nadeem, 2018; Choubey, 2020). By using different sizes of filters and pooling layers, CNNs can effectively capture various levels of textual information, making them a powerful tool for tasks like sentiment analysis, topic categorisation, and question answering in natural language processing (Amin & Nadeem, 2018; Bhatt et al., 2021) However, there are limitations in explainability, such as gradient shattering and difficulty in distinguishing attribution signs, affect the ability to explain why the network made a particular decision (Mamalakis & Barnes, 2022).

2.3.3.3 Recurrent Neural Networks (RNN)

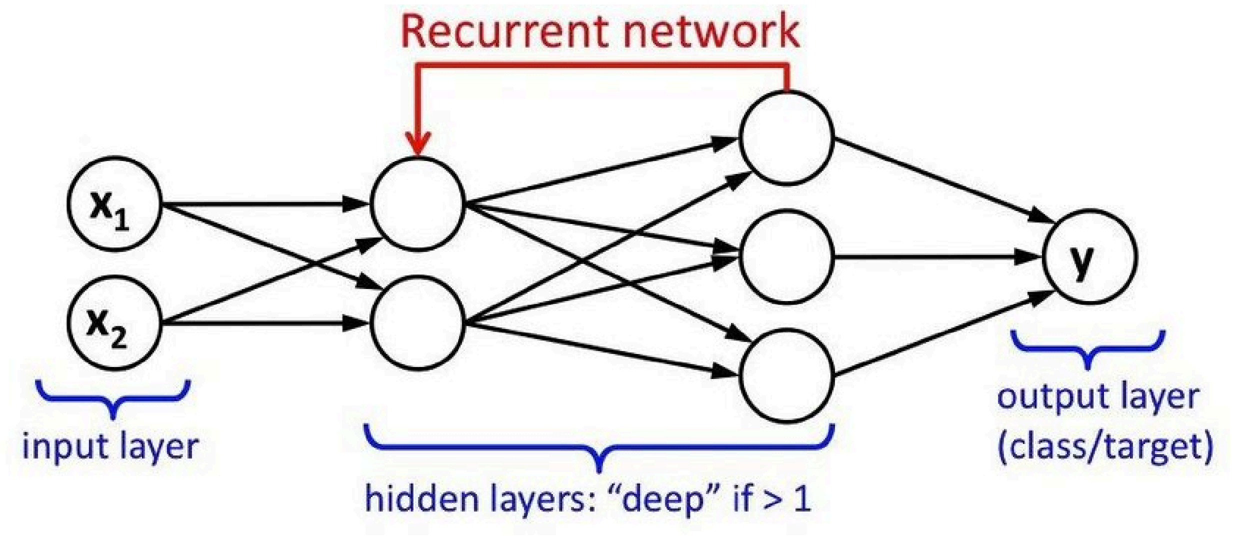


*Figure 6.*Recurrent Neural Networks (RNNs). From "Comprehensive and comparative analysis of neural network" by V. Mishra, S. M. Agarwal, and N. Puri 2018, *International Journal of Computer Application, 2*(8), Artificial Neural Network (ANN) Section, Figure 12 (https://dx.doi.org/10.26808/rs.ca.i8v2.15). CC BY.

Recurrent Neural Networks (RNNs) are an extension of a conventional feedforward neural network that is designed to work with sequential data, such as time series data, natural language text, speech and so on (DiPietro & Hager, 2019; Cheng & Zhao, 2018). Unlike normal feed-forward neural networks, where information flows in one direction from input to output, RNNs have connections that loop back on themselves, allowing them to maintain a hidden state that captures information about past observations in the sequence.

Vanilla Recurrent Neural Networks (RNNs)

Standard RNN updates its hidden state using the current input and the previous hidden state (Tsantekidis et al., 2021). This involves two weight matrices: one for the hidden-to-hidden transformation and one for the input-to-hidden transformation. The input sequence is processed step-by-step, applying a non-linear activation function (like tanh or sigmoid) to the sum of the transformed current input and the previous hidden state. RNNs can be used for various sequential data processing tasks, such as parsing a sequence for classification or generating new sequences. However, the standard RNN will face the problem of vanishing and exploding gradients, which prevents it from being accurate (Kostadinov, 2017; Tsantekidis et al., 2021). The problem happens when the continuous multiplication during the training process causes the gradient to either shrink excessively (vanish) or grow too large (explode). When the gradient vanishes, it becomes too small to make significant weight updates, particularly for those associated with earlier steps in the sequence. This makes the network struggle to learn dependencies from earlier time steps and rely more on recent inputs for its predictions. That is why more powerful models like LSTM come in handy.

Long Short-Term Memory Networks (LSTM)

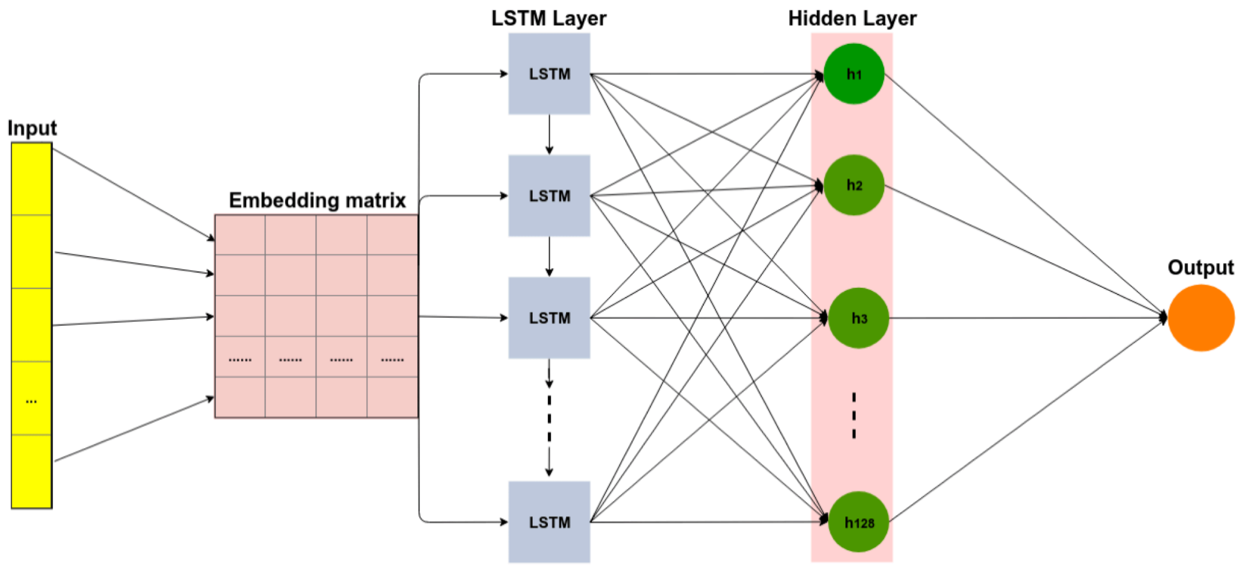


Figure 7. Long Short-Term Memory Networks(LSTM). From "Sentiment Analysis Based on Deep Learning: A Comparative Study" by N. C. Dang, M. N. Moreno-García, and F. De la Prieta, 2020, *Electronics, 9*(3), Background section, Figure 4 (https://doi.org/10.3390/electronics9030483). CC BY.

Long Short-Term Memory (LSTM) is an advanced type of RNN. The main introduction of LSTM was to overcome the vanishing gradient problem in traditional RNNs, providing a way to remember information for long periods. In this case, LSTMs achieve this through a unique internal structure comprising three gates: input, forget, and output (Lindemann et al., 2020; DiPietro & Hager, 2019).

a) **Input Gate:** Controls how much a new value flows into the cell.
b) **Forget Gate:** Decides what information should be discarded or kept.
c) **Output Gate:** Determines the next hidden state – a filtered version of the cell state.

These gates regulate the flow of information into and out of the cell, as well as the cell state across different time steps. This gating mechanism enables LSTMs to effectively handle long-term dependencies in sequential data, making them well-suited for tasks like language modelling, machine translation, and time series prediction. Moreover, variants of LSTMs, like bidirectional LSTMs, allow information to flow in both directions through a sequence, further enhancing their ability to capture context (Lindemann et al., 2020). However, the study also highlights some potential problems in LSTM, for example, according to a paper from Li et al. in

2023, the model does not recognise and extract features well and has a representational bottleneck. Besides, handling such multi-source and multi-frequency information can be challenging for neural network models, including LSTMs, as they may require sophisticated data preprocessing and feature engineering to incorporate all relevant information effectively. Also, a study from Zhaang outlined the problem of overfitting in neural networks like LSTM and the role of regularisation techniques in preventing overfitting.

Gated Recurrent Unit (GRU)

The Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) designed to enhance the computational efficiency of LSTM networks, especially when dealing with massive datasets. The fundamental idea behind GRU is to simplify the internal structure of the LSTM block, reducing the computational complexity and speeding up processing (DiPietro & Hager, 2019). GRU achieves this simplification by employing just two gates: the update gate and the reset gate. The update gate controls the flow of information between time steps, while the reset gate determines the extent to which previous information is retained or discarded.

While there is no definitive winner between LSTM and GRU, the choice between them depends on the specific requirements of the task at hand (Oğuz & Ertuğrul, 2022). GRU is preferred when speed and efficiency are paramount concerns, as it offers a streamlined architecture. On the other hand, LSTM is preferred when the accuracy and precise modelling of sequential data is of utmost importance, even though it may be computationally more intensive. Ultimately, the choice between LSTM and GRU should align with the trade-offs between processing speed and accuracy in addressing specific tasks.

2.3.3.4 Transfer Learning

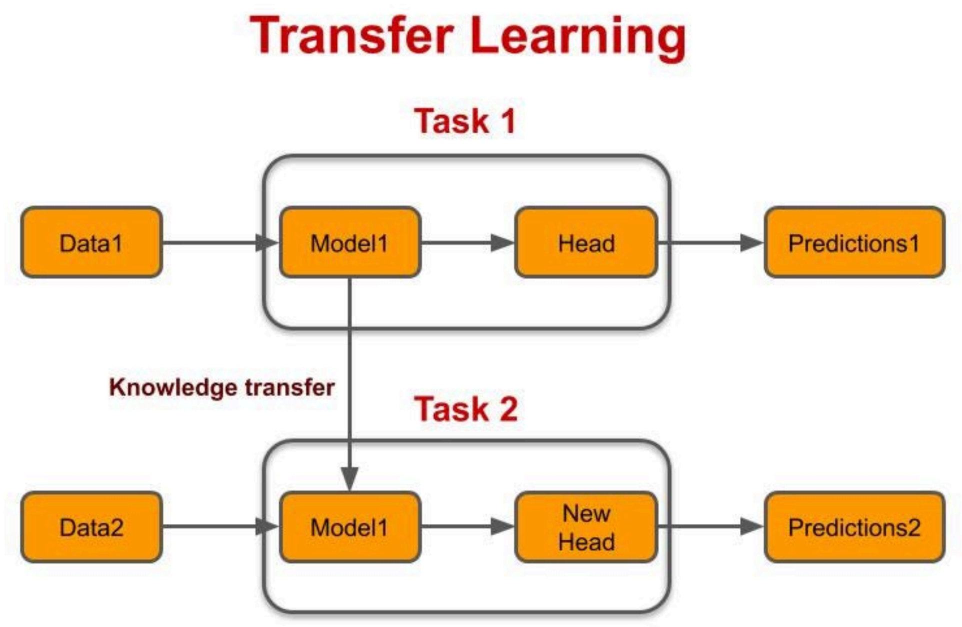


Figure 8. Transfer Learning. From "Homogeneous transfer learning for supporting pervasive edge applications" by T. Moustakas and K. Kolomvatsos, 2023, *Evolving Systems,* (https://doi.org/10.1007/s12530-023-09548-3). CC BY.

Transfer learning refers to a technique where a pre-trained model, which has already been trained on a large dataset, sometimes with a different number of classes and tasks, is used as a starting point for a new task or dataset (Koval et al., 2022). The primary goal of transfer learning is to leverage the knowledge and features learned by the pre-trained model to improve the model's performance on the new, possibly smaller or different, dataset. However, there is some consideration when applying the transfer learning method since the pre-trained model might have been trained on a dataset from a different domain or with different characteristics. Koval et al.'s paper used the pre-trained CNN model as a fixed feature extractor. The classification layer of the CNN is removed, and the features extracted from the CNN's earlier layers are used as input to a new classifier specific to the new dataset. Also, fine-tuning is performed to allow the model to adapt its learned features to the nuances of the new dataset. Similarly, Farhood et al.'s study in plant disease detection (2021) highlights the advantages of transfer learning when working with a dataset with limited training samples. However, there are some considerations that should be taken, including freezing weights of initial layers, graduation reduction of learning rate, and batch normalisation. In Natural language processing (NLP), as Yadav (2023) discussed, transfer learning, especially with pre-trained language models like BERT, holds significant promise in improving the accuracy of sentiment analysis tasks. However, at the same time, the study highlights the importance of being cautious about overfitting and the ability of the model to apply knowledge from training to unseen data, ensuring that models adapt well to diverse and new data, which is essential for real-world applications.

2.3.3.5 Hybrid Approaches

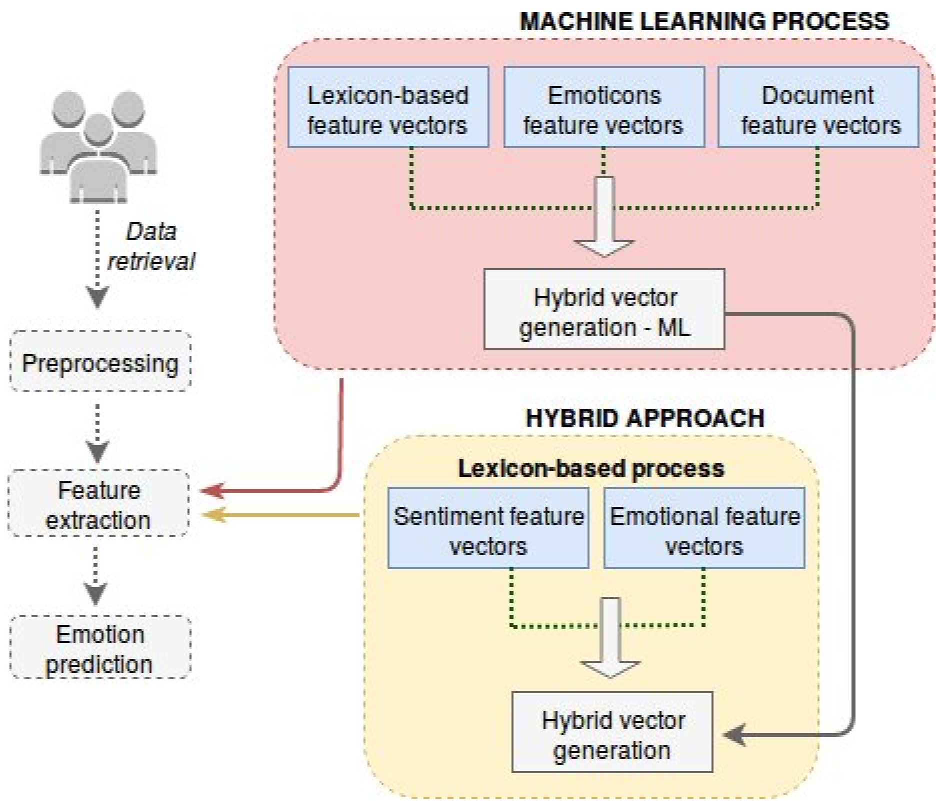


Figure 9. Machine Learning Hybrid Approach. From "Detecting variation of emotions in online activities" by D. Chatzakou, A. Vakali, and K. Kafetsios, 2017, *Expert Systems With Applications,* Figure 1 (https://doi.org/10.1016/j.eswa.2017.07.044). CC BY.

According to Birjali et al. (2021), the hybrid approach refers to combining techniques from two existing techniques to address their respective shortcomings and capitalise on their advantages. For instance, a hybrid approach combines machine learning with lexicon-based methods to achieve optimal results, often resulting in improved outcomes (Munir et al., 2017). The primary aim of this fusion is to achieve optimal results by leveraging the advantageous feature sets of both lexicon and machine learning techniques. This integration aims to address and surpass the limitations and drawbacks inherent in each approach. The study by Al Amrani et al. (2018) evaluated a hybrid model that combined an SVM and a random forest model in sentiment analysis of Amazon's product reviews. The findings indicated that RFSVM achieved an impressive accuracy rate of 83.4%, surpassing the performance of SVM (82.4%) and random forest (81%) when applied individually on a dataset consisting of 1000 reviews. Alqaryouti et al. (2020) make a similar point in their study of the integration of rule-based techniques and domain-specific lexicons for aspect-level sentiment detection, with a focus on assessing public opinions about government smart applications, demonstrated that their proposed technique outperformed other lexicon-based baseline models by a margin of 5%. In a separate investigation conducted by Ray and Chakrabarti (2020), a hybrid model was constructed by combining a rule-based approach for aspect extraction with a deep-learning CNN model consisting of seven layers for aspect tagging. The hybrid model achieved an impressive accuracy rate of 87%, outperforming the individual models, which attained 75% accuracy with the rule-based method and 80% accuracy using the CNN model.

2.4 Multilingual Sentiment Analysis

2.4.1 Techniques Handling Multilingual Content

Cross-Language Transfer

According to Xu et al. (2022), Cross-Langueage Sentiment Analysis (CLSA) is a technique that uses one or more source languages, typically those with abundant resources like English, to aid low-resource languages, which referred to as target languages in this case, in performing sentiment analysis tasks. Most existing sentiment analysis methods rely on supervised learning, which requires large annotated datasets to predict and analyse sentiment in unlabeled text. However, obtaining sentiment-annotated corpora, especially for languages other than English, is complex; in contrast, it has a wealth of sentiment-related resources, including annotated datasets and sentiment lexicons. To address the shortage of annotated data in non-English languages, Yan et al. introduced CLSA in 2004. This approach aims to solve cross-lingual sentiment analysis challenges using Machine Translation techniques to bridge the language barrier and apply sentiment analysis techniques from resource-rich to low-resource (Agüero-Torales et al., 2021). A recent study by Hajmohammadi et al. (2014) demonstrated significant advancements using a multi-view semi-supervised learning model. This approach, which incorporated labelled data from multiple source languages, consistently outperformed baseline methods in terms of accuracy, as evidenced by results like 79.85% in the EnGe-Fr dataset and 81.55% in the EnFr-Ge dataset. Another study by Rasooli et al. (2017) evaluates the feasibility of cross-lingual text classification, particularly sentiment analysis, in low-resource settings. It introduces novel deep-learning methods with partial lexicalisation and annotation projection strategies. The results indicate that these methods are effective in cross-lingual sentiment analysis, demonstrating robust performance across various languages. These findings highlight the practicality and effectiveness of cross-lingual approaches in sentiment classification tasks across different languages, especially in scenarios with limited linguistic resources. Nevertheless, the cross-lingual approach has several challenges and limitations (Wehrmann et al., 2017). These include significant variations in data distribution, the possibility of cultural differences even when translations are accurate, and the difficulty and expense of translating large corpora, which involves issues related to cost, accessibility, and performance.

Language-independent Features

Feature refers to a specific, quantifiable attribute or characteristic of a phenomenon in machine learning and pattern recognition (Anzai, 1991). The selection of informative, distinct, and uncorrelated features plays a pivotal role in the success of algorithms employed in tasks like pattern recognition, classification, and regression. While features typically consist of numerical values, structural attributes like strings and graphs are also utilised in cases of syntactic pattern recognition. The notion of a "feature" shares similarities with the concept of an explanatory variable, commonly employed in statistical methods like linear regression.

Language-independent features in text classification refer to using effective features across multiple languages without being specific to any single language. They play a crucial role in text classification as they represent the essential characteristics or attributes of the data that machine learning algorithms utilise to categorise or classify the text (Shah & Patel, 2016). Some common features used in text classification include:

a) **Term Frequency (TF)**: Measures how frequently a term occurs in a document.
b) **Inverse Document Frequency (IDF)**: Gauges the importance of a term across the entire document corpus.
c) **N-grams**: Captures sequences of 'n' items (words or characters) from the text.
d) **Part-of-Speech Tag**s: Uses grammatical tags (nouns, verbs, adjectives, etc.) as features.
e) **Semantic Features**: Derived from the meaning or context of words.
f) **Word Embeddings**: Vector representations of words capturing semantic meanings.

However, the effectiveness of these features is highly dependent on the nature of the text data and the specific requirements of the classification task. A previous study by Argueta and Chen in 2014 highlights the use of emotion-bearing patterns as a universal, language-agnostic approach to sentiment analysis, particularly in the diverse linguistic environment of social media. Besides, another study by De Leon et al. in 2020 presents a novel approach to sentiment analysis of code-switched social media posts, using word embeddings trained on code-switched data, specifically Spanglish (a mix of Spanish and English). In terms of results, the model achieved an F-1 score of 0.722, indicating a successful implementation of this approach in classifying sentiments in code-mixed text.

Advanced Deep Learning Approaches

In the context of text classification, deep learning models, particularly neural networks, are adept at automatically generating and extracting features from raw text data. This differs from manually selecting features like TF, IDF, or N-grams (Liang et al., 2017). For example, in the case of word embeddings (like Word2Vec and GloVe), deep learning models learn to represent words as vectors in a high-dimensional space. These embeddings capture semantic relationships between words, meaning that words with similar meanings are located close to each other in this space. This allows the model to understand similarities and relationships between words in a way that goes beyond simple one-hot encoding. The statement is supported by Shanmugavadivel et al. in 2022, who explored the use of deep learning models for analysing sentiment and identifying offensive language in multilingual code-mixed data, specifically Tamil and English. The study employs machine learning and deep learning models, including BERT, RoBERTa, and

Adapter-BERT, and compares their performance. The Adapter-BERT model shows promising results, with an accuracy of 65% for sentiment analysis and 79% for offensive language identification, suggesting the effectiveness of deep learning approaches in handling multilingual code-mixed content. Also, Agüero-Torales et al. (2021) again corroborated the effectiveness of deep learning in handling multilingual and code-mixed languages. These models can autonomously extract intricate features from extensive datasets, making them well-suited for the subtleties and context-specific meanings often found in such languages. Further evidence from Medrouk and Pappa's research involved the utilisation of Convolutional Neural Networks (CNNs) for sentiment polarity classification in various languages, including French, English, and Greek. This demonstrated the capability of deep learning in effectively classifying text in multiple languages, achieving an F1 score of 0.92 for bilingual data and 0.88 for trilingual data. However, it is essential to note that these models do exhibit limitations, such as making similar classification errors across different language settings. Agüero-Torales et al. (2021) also emphasised that the feasibility of these methods largely depends on the availability of extensive and diverse datasets for training. Deep learning models can achieve high accuracy only when adequate data is accessible.

2.4.2 Challenges for Multilingual Text

Normalization and Language Identification

Çetinoğlu et al. (2016) highlight the central challenge of multilingual text or code-mixed content—normalisation. Normalisation aims to standardise text, yet in code-switched data encompassing multiple languages, each with distinct linguistic norms, this task becomes intricate. The complexity lies in accommodating variations in grammar, vocabulary, and linguistic conventions across different languages within the text. The reconciliation of diverse linguistic elements is crucial to achieving a consistent and standardised form of code-mixed content. Srivastava and Singh (2020) further elaborate on this challenge. They emphasise that ambiguity in language identification arises when certain words could belong to either language, relying heavily on contextual cues for accurate identification. Additionally, words in code-mixed languages often exhibit multiple spelling variations, complicating the standardisation process. Hence, a deep understanding of the languages involved becomes exceedingly crucial in managing such intricacies. At the same time, when using the cross-language method, translating code-switched text poses unique challenges, as standard translation models are generally trained on monolingual data and may not effectively handle the nuances of mixed languages (Çetinoğlu et al., 2016).

Complexity in Language Processing

Multilingual text processing poses a significant challenge due to the inherent complexity of dealing with multiple languages in a single text. According to Jamatia et al. (2016), detecting utterance boundaries in social media text, especially in code-mixed contexts, is challenging.
Also, POS tagging is particularly problematic in social media contexts due to the informal and mixed nature of the language used. For effective automatic tagging, annotated code-mixed data is necessary, which can be challenging to obtain and process. This is also supported by Çetinoğlu et al. (2016), who also highlighted the problem in POS tagging and stated that it is difficult to correctly tag words from different languages since different languages will have different grammatical rules. In the same study, Çetinoğlu et al. also pointed out that in dependency parsing, when processing code-mixed text, the parser needs to account for and understand the diverse grammar and linguistic elements present in the different languages used within the text.

Data Scarcity

Data scarcity refers to the inadequacy of suitable data, hindering advancements in accuracy and prediction capabilities within systems (Vaidyanathan et al., 2021). It occurs when there is either a limited amount or a complete absence of labeled training data, or when available data for specific labels are insufficient compared to others (Joshi, 2021).

The challenge of data scarcity becomes pronounced in understanding the semantics and morphology of code-mixed languages, as pointed out by Sengupta et al. This issue becomes more complex due to the absence of reliable, language-independent techniques for acquiring representations. This scarcity significantly affects the development of end-to-end neural network-based models for multilingual text classification (Gupta et al., 2020). Furthermore, Finkel et al. (2022) highlighted the unique challenges in morphosyntactic analysis presented by code-mixing in endangered languages like Koda.

Additionally, traditional natural language processing (NLP) tools trained on monolingual corpora struggle with detecting code-mixed text due to non-standard variations in spelling, grammar, and foreign script usage (Banerjee et al., 2020). The difficulty in training specialized models for code-switched texts is compounded by the absence of extensive datasets (Gautam et al., 2021).

3.0 Methodology

The methodology section of this study outlines the systematic approach and steps undertaken to conduct sentiment analysis on social media content written in Malaysian code-switching and code-mixing language, commonly known as “Rojak Language” or “Bahasa Rojak”. The primary objective of this research is to gain insight into sentiment expressed within this unique linguistic context, where individuals frequently blend multiple languages and dialects.

The methodology employed in this project can be categorised into several key phases, each contributing to the overall analysis process. These phases encompass data collection, pre-processing, feature extraction, modelling, evaluation, fine-tuning, and developing a user-friendly interface. The following sections will provide a comprehensive overview of each step, detailing the methods and techniques employed to address the complexities of Rojak Language sentiment analysis on social media.

Through this methodology, we aimed to create a robust framework for understanding and interpreting sentiment patterns within code-switched and code-mixed text, contributing to a deeper understanding of linguistic diversity and sentiment expression in online conversations.

3.1 Data Collection

Data collection is the process of gathering information or data from various sources or subjects for the purpose of analysis, research, or study. It allows us to capture a record of past events to use data analysis to find recurring patterns.

3.1.1 Data Source

This study applied a two-source data collection approach to compiling a comprehensive dataset for sentiment analysis on social media text written in the Malaysian code-switching and code-mixing languages. This dual-source strategy was chosen to enhance the richness and diversity of the dataset, thus providing a more holistic understanding of sentiment expression within this linguistic context.

The utilisation of two distinct sources, the Reddit API and Hugging Face datasets, offered several advantages. Firstly, the Reddit API allowed us to collect a substantial amount of user-generated content from the subreddit "Malaysia," comprising titles, bodies, and comments from 2000 posts, resulting in a total of 14,059 records. This source provided a valuable representation of Malaysian language use in an online community setting, reflecting real-world interactions and discussions. Secondly, since using the Twitter API for academic purposes was not permitted, our team incorporated a dataset from Hugging Face, which consisted of 1,048,576 records resembling tweet text. By incorporating this source, the team expanded the dataset's size and diversity, further enriching the language samples. However, it is important to note that these data encompass a mixture of languages, including English, Malay, Chinese, Tamil, Manglish, and Rojak language. Thus, data sampling will be necessary to ensure consistency.

3.1.2 Data Sampling

By utilising the language detection module provided by the malaya library, the team ensured that only content in the Rojak language was included in the dataset. This approach effectively addressed the challenge posed by the coexistence of multiple languages within the collected social media content, such as English, Malay, Manglish, and Rojak. After successfully obtaining Rojak language data from both sources, we merged the data from the Reddit API and the Hugging Face Datasets. To create a balanced dataset, approximately 3,000 records were selected from the Rojak language data collected from the Reddit API. Additionally, we randomly chose 2,000 records from the filtered Twitter data, resulting in a total of 5,000 records. When selecting these records, we intentionally excluded those consisting only of URLs with no other textual content to eliminate irrelevant information. Lastly, the final cleaned and merged dataset was saved under the filename "combined_data_random.csv" for our subsequent sentiment analysis.

3.1.3 Data Annotation

In sentiment analysis, data annotation or manual labelling plays a crucial role in creating a labelled dataset for training supervised machine learning models. However, in our study, which focuses on sentiment analysis of Malaysian code-switching and code-mixing language on social media, our data collection from social media sources presented a unique challenge: the absence of pre-existing sentiment labels for the extensive amount of text data we acquired. While manual data annotation has advantages, such as ensuring precise and tailored sentiment labels for each text record, there were other feasible options due to the dataset's size. With thousands of records collected from various social media conversations, manually labelling each data point would have been time-consuming and resource-intensive. This approach would have caused significant delays in our research and could have introduced potential inconsistencies or biases in the labelling process.

Instead, our team adopted a transfer learning approach by leveraging sentiment analysis pre-trained transformer models from the malaya library on the Rojak languages dataset we had gathered. This approach saved a substantial amount of time and resources and benefited from the pre-trained models' ability to capture complex linguistic patterns and context-specific sentiment expressions. Additionally, this method allowed for scalability, enabling us to handle large volumes of data efficiently. However, it is essential to acknowledge the limitations of not manually annotating the data. Automated sentiment analysis models, while powerful, may only sometimes accurately capture the nuances of Rojak Language sentiment. They rely on patterns learned from the training data, which may only partially encompass the unique linguistic characteristics of our dataset. Therefore, some errors or misclassifications may occur in the automated labelling process. Nonetheless, this approach remained our primary choice for labelling the data due to its efficiency and scalability.

3.2 Data Preprocessing

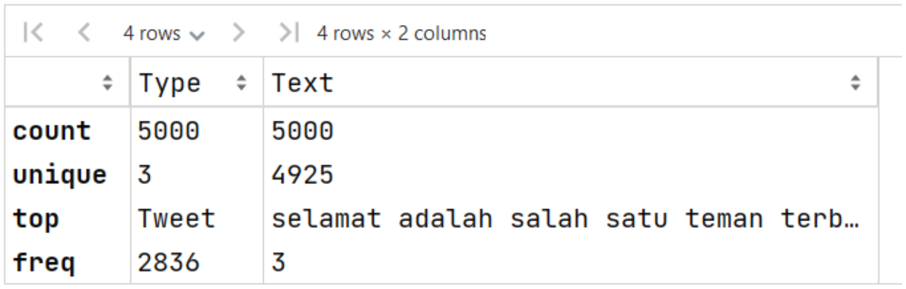
Data preprocessing involves transforming raw data into a refined dataset by addressing issues such as missing values, noisy data, and other inconsistencies to prepare it for use in algorithms (Singh, 2020).

3.2.1 Data Cleaning

Data cleaning is a critical initial step in the data pre-processing pipeline, focusing on enhancing the quality and usability of the raw text data collected from social media sources. This phase involves various actions aimed at removing noise, irrelevant content, and inconsistencies, ultimately resulting in a more standardised and manageable dataset.

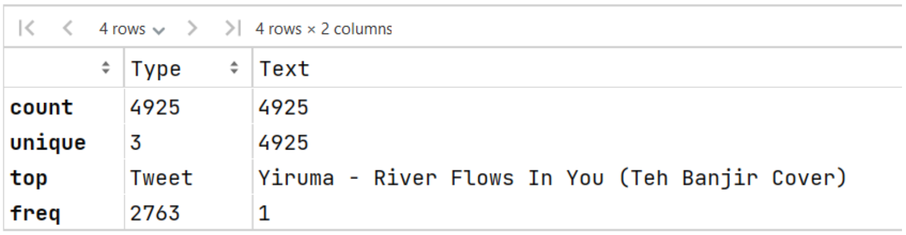
3.2.1.1 Duplicate Record Removal

Before any process of data pre-processing, duplicate records were identified and removed from the dataset. Duplicate records refer to instances where the exact same text entries appeared more than once within the dataset. The removal of duplicate records was performed to maintain dataset integrity and prevent the skewing of analysis results. Duplicate records can lead to overrepresentation of certain sentiments or language patterns, potentially biasing the sentiment analysis. By eliminating these duplicates, the dataset was made more robust, ensuring that each unique text entry contributed equally to the analysis. To do this, the team used the drop_duplicates function provided for the data frame, which can remove duplicated records effectively.



```
        Type    Text
count   5000    5000
unique  3       4925
top     Tweet   selamat adalah salah satu teman terb…
freq    2836    3
```

Figure 10. Dataframe Info before duplicated Records Removal



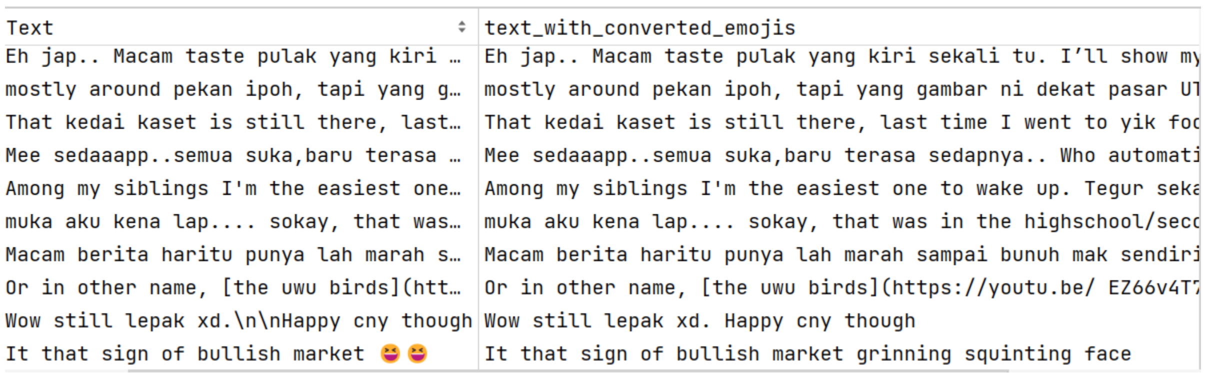
```
        Type    Text
count   4925    4925
unique  3       4925
top     Tweet   Yiruma - River Flows In You (Teh Banjir Cover)
freq    2763    1
```

Figure 11. Dataframe Info before duplicated Records Removal

3.2.1.2 Emoji Conversion

In the field of Natural Language Processing (NLP), emojis are often considered noise and are typically removed during text pre-processing. However, in the context of sentiment analysis, emojis can carry valuable emotional content and provide nuance to the sentiment expressed within a text. Therefore, in this study, the initial pre-processing step involved retaining emojis and converting them into textual representations before proceeding with further pre-processing steps. This approach aimed to capture potential sentiment information emojis convey in the text. The conversion from emojis to text representations was achieved using the emoji library in Python, which effectively replaced the Unicode or punctuation-based emoji format with descriptive text.

Additionally, an important consideration during this process was handling consecutive usage of the same emoji within a text, which could potentially introduce redundancy and noise in the data. For example, sequences like 🤣🤣🤣🤣🤣🤣, if directly converted, could result in an excessive and misleading representation of the same emoji. To address this issue, a strategy was implemented to reduce consecutive emoji descriptions to a single occurrence, thereby preventing the overrepresentation of emotional content.



Text	text_with_converted_emojis
Eh jap.. Macam taste pulak yang kiri …	Eh jap.. Macam taste pulak yang kiri sekali tu. I’ll show my
mostly around pekan ipoh, tapi yang g…	mostly around pekan ipoh, tapi yang gambar ni dekat pasar UT
That kedai kaset is still there, last…	That kedai kaset is still there, last time I went to yik foo
Mee sedaaapp..semua suka,baru terasa …	Mee sedaaapp..semua suka,baru terasa sedapnya.. Who automati
Among my siblings I'm the easiest one…	Among my siblings I'm the easiest one to wake up. Tegur seka
muka aku kena lap.... sokay, that was…	muka aku kena lap.... sokay, that was in the highschool/seco
Macam berita haritu punya lah marah s…	Macam berita haritu punya lah marah sampai bunuh mak sendiri
Or in other name, [the uwu birds](htt…	Or in other name, [the uwu birds](https://youtu.be/ EZ66v4T7
Wow still lepak xd.\n\nHappy cny though	Wow still lepak xd. Happy cny though
It that sign of bullish market 😆😆	It that sign of bullish market grinning squinting face

Figure 12. Application of Emoji Conversion

3.2.1.3 Reduplicated Word Conversion

Another crucial aspect of data pre-processing in the context of Malaysian text analysis involves addressing reduplicated words, known as "kata ganda." These reduplicated words are typically written in a shortened form, followed immediately by the numeral '2'. To handle this linguistic characteristic, the pre-processing step focused on identifying words ending in '2' and then duplicating the preceding word part. For example, "tiba2" was transformed into "tiba," and "jalan2" became "jalan." In Malay, reduplicated words with the '2' suffix are a common linguistic phenomenon. These words often convey a sense of repetition or emphasis. To ensure accurate sentiment analysis and linguistic understanding, it is imperative to handle these reduplicated words appropriately. By removing the '2' suffix and duplicating the preceding word part, the text data became more aligned with standard Malay language usage, thus improving the readability and interpretability of the text for analysis. To do so, the defined regular expression pattern (`\b(\w+)2\b`) captures word stems followed by the digit '2' at the end of a word boundary, then `re.sub` is used to replace these matched patterns with just the captured word stem, effectively removing the trailing '2'.

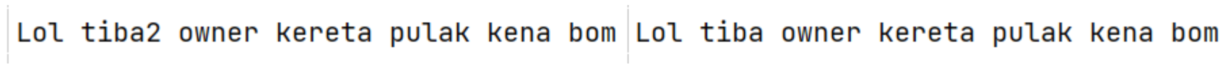
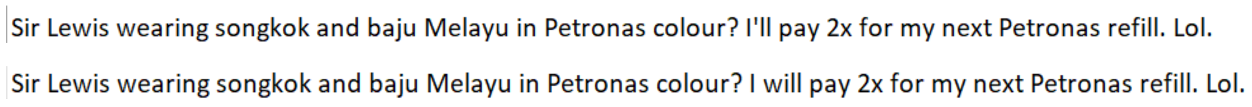

Lol tiba2 owner kereta pulak kena bom Lol tiba owner kereta pulak kena bom

Figure 13. Example of Conversion in Reduplicated Word

3.2.1.4 Contractions Expanding

Expanding contractions in text data was another critical data pre-processing step. Contractions, such as "I'll" or "won't," are commonly used in natural language. To enhance readability and accuracy during data cleaning and analysis, contractions were expanded to their complete form using the contractions library in Python. In this step, the project uses the contractions library in Python to expand contractions. By splitting the text into individual words, the program will iterate through each word and use `contractions.fix` to identify and replace contracted forms with their expanded equivalents. Expanding contractions was essential to ensure the text data accurately reflected the full words and phrases. In sentiment analysis, understanding the complete form of contractions is vital, as they can carry distinct meanings and sentiments. This pre-processing step improved the accuracy of sentiment analysis and the text's overall readability, aligning the dataset with standard language conventions for more precise linguistic analysis.


Sir Lewis wearing songkok and baju Melayu in Petronas colour? I'll pay 2x for my next Petronas refill. Lol.

Sir Lewis wearing songkok and baju Melayu in Petronas colour? I will pay 2x for my next Petronas refill. Lol.

Figure 14. Text before and after Expanding Contractions

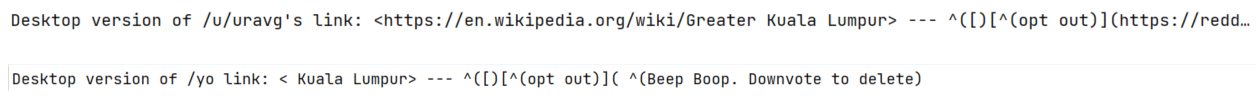
3.2.1.5 Noise Removal

The noise removal process in data pre-processing is a crucial step to enhance the quality and relevance of the text data for subsequent analysis, particularly for sentiment analysis on social media content. This process involves several key steps, each aimed at eliminating extraneous characters and information that may interfere with the accuracy of the analysis. The goal is to produce cleaner and more meaningful text, optimising it for various natural language processing tasks while retaining the punctuation and original letter casing for accurate segmentation.

The first step involves replacing newline characters ('\n') with spaces. Newline characters, often used for formatting, do not carry meaningful information for sentiment analysis. The text becomes more coherent and suitable for further analysis by converting newline characters to spaces. Then, URLs are removed from the text using regular expressions (regex). Though prevalent in social media content, URLs typically do not contribute to sentiment analysis. Eliminating them helps exclude irrelevant web links and their associated characters, ensuring that the text focuses on user-generated content.

Moreover, user identifiers such as usernames, commonly found on platforms like Twitter and Reddit, are removed using regex. These usernames, starting with '@' (for Twitter) or 'u/' and 'r/' (for Reddit), are generally irrelevant to sentiment analysis and can be distracting. Removing them ensures that the analysis concentrates solely on the content expressed by users. Finally, the noise removal eliminates duplicate spaces, tabs, and other whitespace characters while maintaining punctuation and original letter casing. This step maintains uniformity and readability in the text. Multiple consecutive whitespace characters are replaced with a single space, and trailing or leading spaces are trimmed, ensuring consistent spacing throughout the text.

The rationale behind this noise removal process is to improve the signal-to-noise ratio in the text data, thereby enhancing the accuracy and reliability of subsequent analysis tasks, especially sentiment analysis. The resulting text remains contextually accurate by eliminating extraneous characters and retaining essential punctuation and letter casing. It preserves linguistic nuances, making it well-suited for analysing sentiment in Malaysian Code-switching and code-mixing language (Rojak Language) social media content. It is worth noting that in this phrase, punctuation and original letter case are kept to allow for more accurate segmentation while maintaining the integrity of the text and preserving the potential nuances associated with uppercase letters.



```
Desktop version of /u/uravg's link: <https://en.wikipedia.org/wiki/Greater Kuala Lumpur> --- ^([)[^(opt out)](https://redd…

Desktop version of /yo link: < Kuala Lumpur> --- ^([)[^(opt out)]( ^(Beep Boop. Downvote to delete)
```

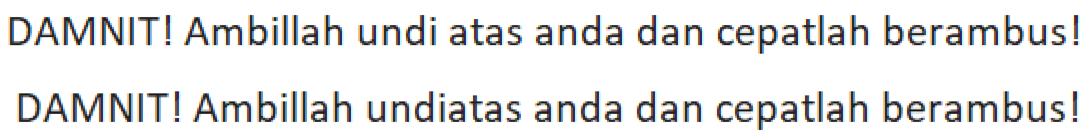
Figure 15. Text before and after Nosie Removal

3.2.1.6 Segmentation

Analyzing social media texts often presents the challenge of encountering missing spaces within the text, which makes it difficult to separate individual sentences accurately. Text segmentation becomes crucial to tackle this issue. In our study, we employed a pre-trained model from the HuggingFace module of the Malaya library for sentence segmentation. This model is equipped to handle code-switched text in Malay and English, making it well-suited for the linguistic diversity found in our dataset.

When dealing with large datasets, especially in situations where memory resources are limited, attempting to process the entire dataset all at once can exceed available memory, resulting in memory errors or crashes. To address these challenges, we adopted a batch-processing approach.

In summary, sentence segmentation, enabled by the pre-trained Malaya HuggingFace model, plays a vital role in ensuring accurate sentence-level analysis of social media texts. Addressing missing spaces and effectively managing code-switched content ensures that subsequent sentiment analysis is conducted with a detailed understanding of the text, leading to more precise and context-aware results.



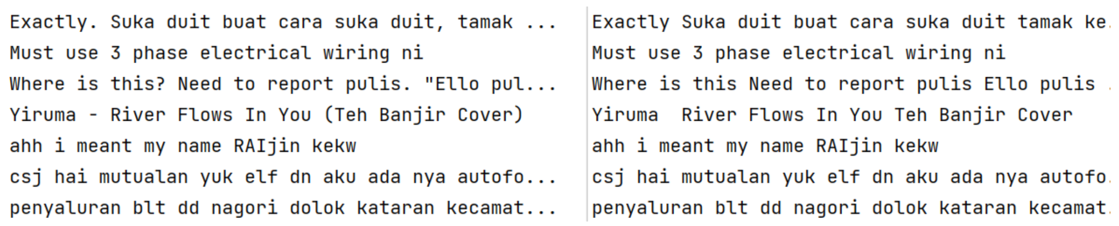
DAMNIT! Ambillah undi atas anda dan cepatlah berambus!

DAMNIT! Ambillah undiatas anda dan cepatlah berambus!

Figure 16. Text before and after Segmentation

3.2.1.7 Punctuation Removal

After performing sentence segmentation on the social media text, the subsequent step involved the removal of punctuation using a regular expression pattern (`[^A-Za-z0-9\s]+`). The pattern will match the character or symbol other than the alphanumeric character, and then the team uses `re.sub` to remove them. Since the segmentation process effectively separated the text into individual sentences, punctuation marks were no longer necessary for sentence delineation or syntactic structure. Removing punctuation simplified the text, making it more amenable to subsequent sentiment analysis.



```
Exactly. Suka duit buat cara suka duit, tamak ...   Exactly Suka duit buat cara suka duit tamak ke
Must use 3 phase electrical wiring ni               Must use 3 phase electrical wiring ni
Where is this? Need to report pulis. "Ello pul...   Where is this Need to report pulis Ello pulis
Yiruma - River Flows In You (Teh Banjir Cover)      Yiruma  River Flows In You Teh Banjir Cover
ahh i meant my name RAIjin kekw                     ahh i meant my name RAIjin kekw
csj hai mutualan yuk elf dn aku ada nya autofo...   csj hai mutualan yuk elf dn aku ada nya autofo
penyaluran blt dd nagori dolok kataran kecamat...   penyaluran blt dd nagori dolok kataran kecamat
```

Figure 17. Punctuation Removal

3.2.2 Rule-based Normalisation

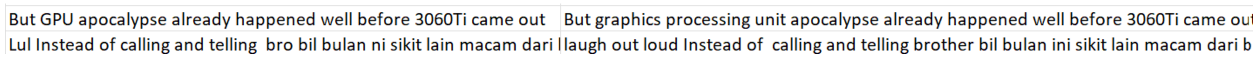
3.2.2.1 Abbreviation Handing

In the subsequent phase of data pre-processing, our methodology addresses handling common English and Malay abbreviations within the segmented and punctuation-removed text. This process comprises two primary components: replacing common English abbreviations and replacing common Malay abbreviations.

In this part, we utilise two sets of abbreviation dictionaries, one for Malay and one for English, to replace abbreviations in a text column within a data frame. Initially, our script loads a JSON file containing Malay abbreviations into a dictionary named "`abbreviations`" and then flattens it to a single level for easier access, storing it as "`flat_abbreviations`." Subsequently, we define a function called "`replace_abbreviations_malay`" to replace Malay abbreviations within a text using the provided "`abbrev_dict`." This function splits the text into words, checks if each word exists in the abbreviation dictionary, and replaces the abbreviation with its expanded form if found.

Similarly, we load an English abbreviation expansion dictionary from another JSON file and convert it into a single-level dictionary, naming it "`abbreviation_expansion_dict`." Using the English abbreviation dictionary, we create another function, "`replace_abbreviations`," to replace English abbreviations within the previously processed text column. Both replacement functions are applied sequentially to the `'segmented_text'` column of the DataFrame using the "`.apply`" method, ensuring that Malay and English abbreviations are replaced with their respective expansions in the text.

Replacing abbreviations with their expanded forms enhances text clarity and ensures that sentiment analysis accurately interprets the text's intended meaning. This step is essential for comprehensive language understanding.



But GPU apocalypse already happened well before 3060Ti came out	But graphics processing unit apocalypse already happened well before 3060Ti came out
Lul Instead of calling and telling bro bil bulan ni sikit lain macam dari l	laugh out loud Instead of calling and telling brother bil bulan ini sikit lain macam dari b

Figure 18. Text before and after Expanding Abbreviations

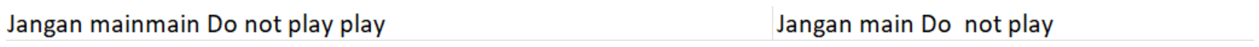
3.2.2.2 Regex-Based Word Removal

In the pursuit of refining the text data for more precise sentiment analysis, a set of carefully crafted regex-based rules were employed, each with distinct functions and outcomes. The following table provides a concise summary of these rules, elucidating their original form and the modifications they impart to the text:

Rule Type	Function related	Original	Modified
Elimination of Words with 2 or More Hyphens	`deduplicate_hyphenated_words`	PQR2 pokok2, PQR ayam2	PQR pokok, PQR ayam
		PQR-PQR itik-itik	PQR itik
		PQR PQR api api	PQR api
Elimination of Duplicate Characters	`eliminate_duplicate_characters`	PQRRRRR takkkkk	PQR tak
Splitting Words with Similar Terms into Two Groups	`split_repeated_terms`	PQPQPQPQ hahahaha	PQPQ haha
Remove duplicate word	`remove_duplicate_words`	PQ PQ PQ ha ha ha	PQ ha

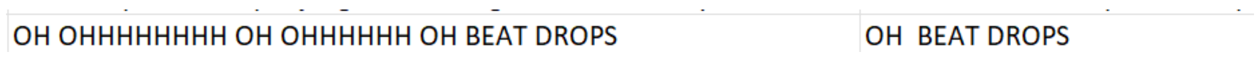
Table 1. Rule Defined to Normalise Text

This comprehensive set of rules reflects the methodical approach undertaken to normalize the text data. Each rule, grounded in a specific function, contributes to text refinement by addressing distinct aspects, such as the elimination of repeated characters, the separation of similar terms, and the removal of redundancy. These concerted efforts culminate in a more coherent and standardized text dataset, poised for rigorous sentiment analysis.



Jangan mainmain Do not play play	Jangan main Do not play

Figure 19. Example 1: Text before and after Rule Normalisation



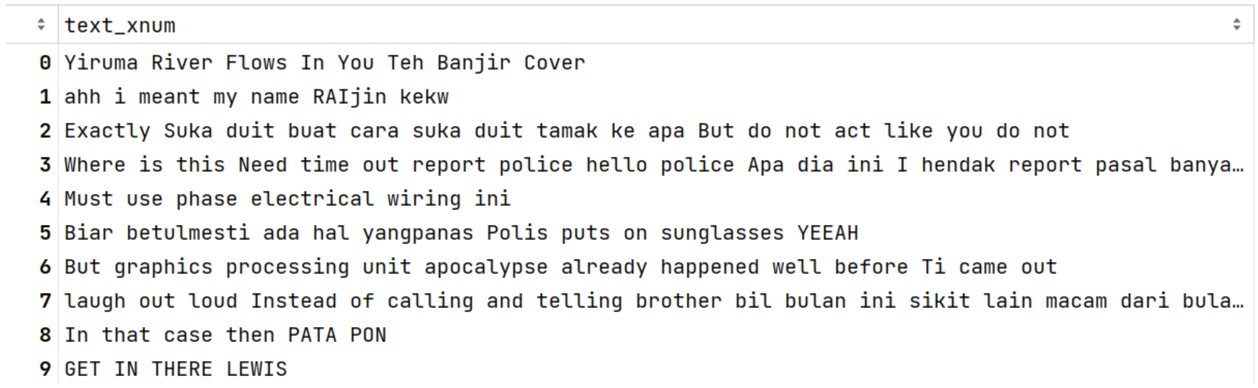
OH OHHHHHHHH OH OHHHHHH OH BEAT DROPS	OH BEAT DROPS

Figure 20. Example 2: Text before and after Rule Normalisation

3.2.2.3 Number Removal

In this section of the methodology, the focus shifts towards the exclusion of numeric characters from the text as a primary preprocessing step. The team employs a regular expression pattern (`\d+`) to identify and remove numeric characters from the text. The resulting text, stored in the 'text_xnum' column of the DataFrame, is devoid of numerical elements.

The decision to remove numbers from the text data is grounded in the consideration that, after expanding abbreviations, numbers may hold limited relevance in the context of sentiment analysis. Thus, their removal contributes to a more streamlined and focused text dataset, ready for subsequent sentiment analysis tasks.



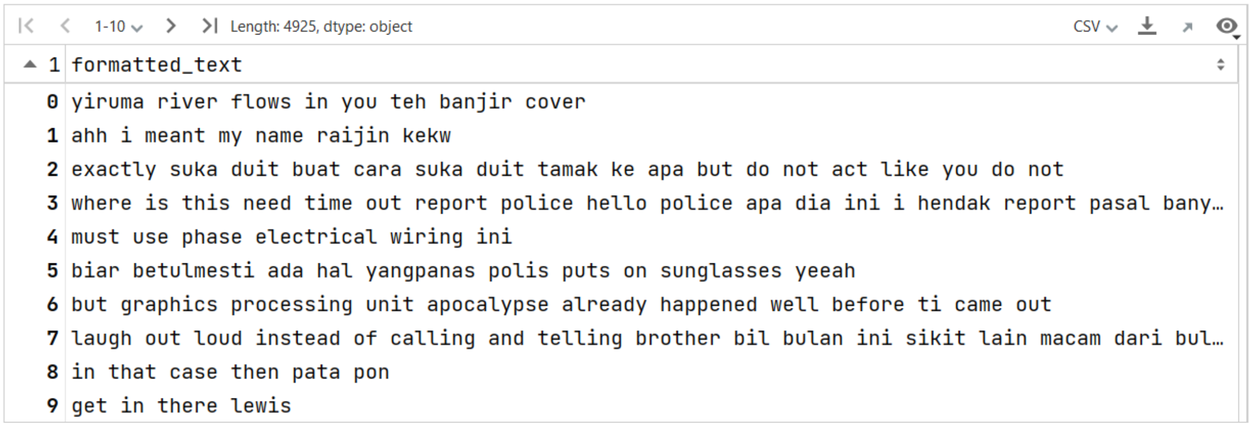
```
  text_xnum
0 Yiruma River Flows In You Teh Banjir Cover
1 ahh i meant my name RAIjin kekw
2 Exactly Suka duit buat cara suka duit tamak ke apa But do not act like you do not
3 Where is this Need time out report police hello police Apa dia ini I hendak report pasal banya…
4 Must use phase electrical wiring ini
5 Biar betulmesti ada hal yangpanas Polis puts on sunglasses YEEAH
6 But graphics processing unit apocalypse already happened well before Ti came out
7 laugh out loud Instead of calling and telling brother bil bulan ini sikit lain macam dari bula…
8 In that case then PATA PON
9 GET IN THERE LEWIS
```

Figure 21. Text after Number Removal

3.2.2.4 Lowercasing

Following the removal of numbers from the text preprocessing pipeline, the next step involves the application of lowercase transformation to the text data. This process entails converting all alphabetic characters in the text to their lowercase equivalents, ensuring uniformity and consistency in the dataset.

The application of lowercase transformation standardizes the text and eliminates potential discrepancies introduced by varying letter cases. It ensures that the subsequent sentiment analysis treats words with different letter cases as identical, promoting a more accurate and comprehensive analysis of the text. This step is instrumental in mitigating the impact of letter case variations on sentiment analysis outcomes, facilitating a more robust and reliable analysis of the text data.



```
|< < 1-10 v > >| Length: 4925, dtype: object                                   CSV v
  1 formatted_text
  0 yiruma river flows in you teh banjir cover
  1 ahh i meant my name raijin kekw
  2 exactly suka duit buat cara suka duit tamak ke apa but do not act like you do not
  3 where is this need time out report police hello police apa dia ini i hendak report pasal bany…
  4 must use phase electrical wiring ini
  5 biar betulmesti ada hal yangpanas polis puts on sunglasses yeeah
  6 but graphics processing unit apocalypse already happened well before ti came out
  7 laugh out loud instead of calling and telling brother bil bulan ini sikit lain macam dari bul…
  8 in that case then pata pon
  9 get in there lewis
```

Figure 22. Text after Lowercasing

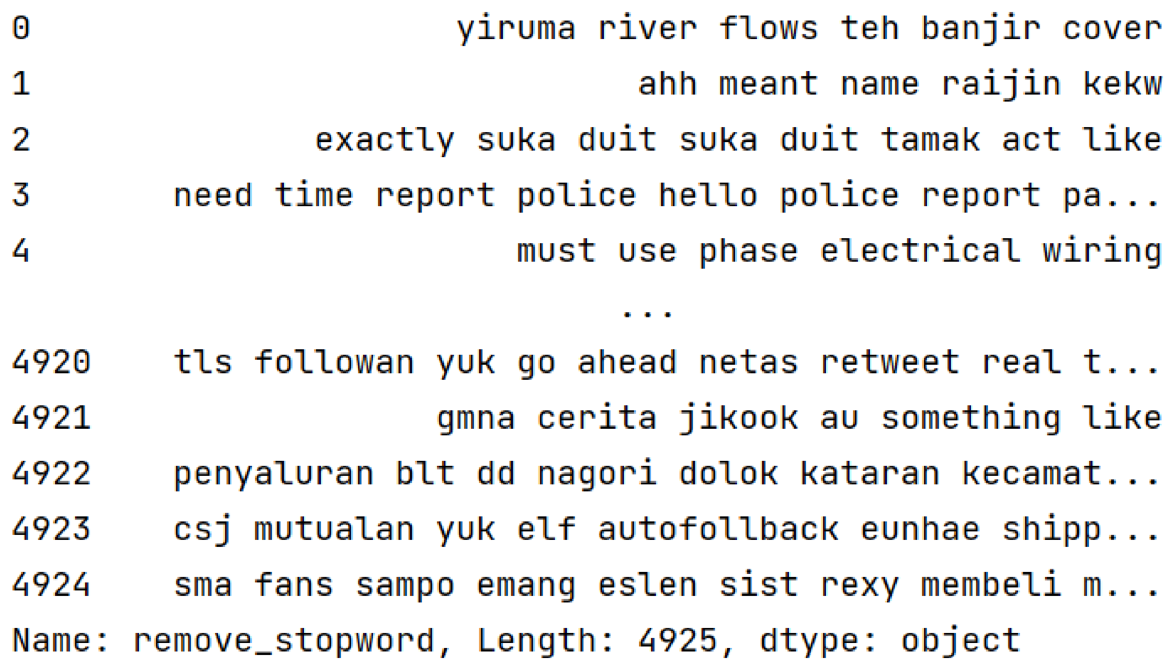
3.2.3 Stop words removal

The critical phase of stopwords Removal is designed to enhance the text data's quality and relevance for subsequent analysis. Stopwords, which encompass common, frequently occurring words with minimal contextual meaning, are systematically eliminated from the text. This process consists of two distinct functions applied to the 'formatted_text' column within the DataFrame 'df,' each tailored to handle specific language stopwords.

Removal of Malay Stopwords: The initial function, remove_malay_stopwords, harnesses the Malaya library to access a comprehensive set of Malay stopwords. The text is tokenized by splitting it into words, and words present in the Malay stopwords set are subsequently filtered out. This meticulous procedure results in a version of the text that is devoid of these frequently encountered Malay stopwords. The objective here is to refine the 'formatted_text' column by removing common words that add little to the overall contextual meaning, thus directing the focus toward more informative content for subsequent analysis.

Removal of English Stopwords: In a parallel vein, the second function, remove_english_stopwords, leverages the English stopwords list from NLTK (Natural Language Toolkit). Employing NLTK's word tokenizer, the text is tokenized, and English stopwords are meticulously excised from the text. This process mirrors the Malay stopword removal, aiming to enhance the text's suitability for analysis by eliminating common English words that may dilute its meaning.

By systematically removing stopwords in both Malay and English, the text undergoes a transformation that accentuates contextually meaningful content. This selective refinement prepares the text for more profound and insightful analysis or modelling tasks, focusing attention on the most pertinent aspects of the data.



```
0                        yiruma river flows teh banjir cover
1                                ahh meant name raijin kekw
2                    exactly suka duit suka duit tamak act like
3       need time report police hello police report pa...
4                             must use phase electrical wiring
                                    ...
4920    tls followan yuk go ahead netas retweet real t...
4921                      gmna cerita jikook au something like
4922    penyaluran blt dd nagori dolok kataran kecamat...
4923    csj mutualan yuk elf autofollback eunhae shipp...
4924    sma fans sampo emang eslen sist rexy membeli m...
Name: remove_stopword, Length: 4925, dtype: object
```

Figure 23. Text after Removing Stop Word

3.2.3 Tokenisation

Tokenization, a fundamental task in natural language processing (NLP), involves the segmentation of text into discrete units, commonly words or subwords. This step holds pivotal importance as a preprocessing stage preceding various NLP tasks. Tokenization breaks down the input text into tokens, which are individual linguistic units, facilitating easier analysis and manipulation.

The tokenization process initiates by segmenting the text into distinct words or phrases. For this purpose, the NLTK library's word_tokenize function is employed. The team utilizes the word_tokenize function to tokenize the text stored in the 'remove_stopword' column of the DataFrame. The result is a new column labelled 'tokenized_text' containing the text data segmented into individual tokens.

Tokenization is a pivotal step that lays the foundation for subsequent NLP tasks by breaking down text into manageable linguistic units, enabling more effective analysis and processing.

3.2.4 Word-Level Language Detection

This part utilises Malaya's language detection model, specifically the FastText-based model, to conduct word-level language detection within the 'tokenized_text' column of the DataFrame 'df'. First, the code loads the FastText language detection model from Malaya's library. Then, it defines a function named `apply_language_detection` that applies the language detection model to each word in a list. The function takes a row (which represents a list of tokenized words) as input and uses `model.predict` to predict the language of each word or token. This process generates pairs of words and their predicted languages, creating a list of tuples for each entry in the 'tokenized_text' column. Finally, the `apply_language_detection` function is applied to the 'tokenized_text' column using `.apply()`, resulting in a new column named 'token_lang_refer' in the DataFrame 'df'. This column contains lists of word-language pairs, allowing for subsequent processing tailored to the language of individual words or phrases within the text.

3.2.5 Stemming and Lemmatisation

This part aims to process tokens based on their detected language. It utilizes different stemming or lemmatization approaches depending on the language of each token within the 'token_lang_refer' column in the DataFrame 'df'. First, it loads the Malaya stemmers for the Naive and Sastrawi algorithms. Then, a function `apply_stemming_methods` is defined to apply both stemmers to the input text, generating stemmed results for comparison. Following this, the script continues by defining a function process_token responsible for processing tokens based on their detected language. Within this function, `process_token_inner` processes individual tokens by selecting the appropriate language-specific processing technique. For English tokens ('EN'), the script utilizes Spacy's English model to perform lemmatization. For Malay tokens ('MS'), the Sastrawi stemmer from Malaya is employed for stemming. Tokens detected as other languages ('OTHERS') or undefined ('NOT_LANG') are retained as is. Finally, the process_token function is applied to each row in the 'token_lang_refer' column using `.apply()`, generating a new column 'stemmed_token' in the DataFrame 'df' that contains processed tokens based on their respective languages, allowing language-specific text processing to enhance subsequent analysis or modeling tasks.

3.2.6 Special NLP Processes

Processes employ specialized algorithms and models to extract insights from text data, enabling targeted analysis and understanding of language patterns beyond basic NLP tasks.

NER (named entity recognition)
NER is a natural language processing (NLP) technique used to recognize named entities (e.g., persons, locations, organizations) in text. It involves identifying specific words or phrases in the text and assigning relevant labels to them based on their entity type.

Step 1: Initialisation

- `malaya.zero_shot.entity.available_huggingface():` This retrieves available zero-shot NER models from Hugging Face.
- `ner_model = malaya.zero_shot.entity.huggingface()`: Initializes the NER model using the default zero-shot model provided by Malaya through Hugging Face.
- `tags`: Defines a list of entity types you want the NER model to recognize. These could include names, locations, foods, political party names, politicians, songs, and countries.

Step 2: Defining a Function

- `apply_and_format_ner`: This function takes a row from a DataFrame and applies NER to the text in that row using the initialized model and the specified tags.

Step 3: Applying NER to DataFrame

- `df['ner_results'] = df.apply(lambda row: apply_and_format_ner(row, tags), axis=1):` This applies the apply_and_format_ner function to each row in the DataFrame (`df`) and stores the NER results in a new column called '`ner_results`'.
- The function `apply_and_format_ner` takes each row's text from the column '`text_xnum`', applies the NER model to recognize entities based on the defined `tags`, and formats the results.

Step 4: Second NER Application

- The code following `#%%` appears to repeat a similar process of applying NER to another column in the DataFrame ('`formatted_text`') and stores the results in another column (`'ner_results2'`).

Step 5: Library Import (spacy) and Model Loading

- import spacy: Imports the `SpaCy` library.
- `nlp = spacy.load("en_core_web_sm")`: Loads the English language model (`en_core_web_sm`) provided by SpaCy.

Step 6: NER Function Definition

- The function `extract_entities(text)` is designed to conduct Named Entity Recognition (NER) using SpaCy. It accepts text input, processes it using the SpaCy model (nlp).
- Identifies entities alongside their respective labels (such as PERSON, ORG, GPE, etc.). The output is a list of tuples containing the identified entity text paired with its label.

Step 7: Application of NER on DataFrame's Text Column

- `df['entities'] = df['text_xnum'].apply(extract_entities)`: Utilizes the extract_entities function to apply NER to each row in the DataFrame's '`text_xnum`' column.
- Creates a new column named 'entities' containing the extracted entities and their labels for each corresponding row in the DataFrame.

Step 8: Additional Code Block

- The second code block seems to repeat a similar process with some modifications.
- It loads the SpaCy model again (`nlp = spacy.load("en_core_web_sm")`) and defines a set of desired entity types (`desired_entity_types`) such as 'NORP' (Nationalities or religious or political groups), 'GPE' (Geopolitical entity), 'WORK_OF_ART' (Work of art), 'LANGUAGE'.
- Defines a new function `extract_and_filter_entities(text)` that extracts entities similar to the previous function but filters the results based on the desired entity types specified in `desired_entity_types`.
- Applies this filtering function to the DataFrame's 'text_xnum' column, storing the filtered entities and their labels in a new column named 'filtered_entities'.

This part is designed to use a Named Entity Recognition (NER) model on a DataFrame's text column. It identifies specific entities like names, locations, or organisations based on predefined tags. The extracted entities and their corresponding tags are then organised and stored in separate columns within the DataFrame (ner_results and ner_results2). This process structures the identified information, allowing easy access and analysis within the DataFrame. Furthermore, it applies SpaCy's Named Entity Recognition (NER) to the 'text_xnum' column within the DataFrame. The identified entities and their labels are stored in distinct columns ('entities' and 'filtered_entities'). Additionally, the second block refines this process by selectively storing entities of specific types in the DataFrame after performing the recognition.

Poor Performance in Named Entity Recognition(NER)

One significant factor that contributes to this could be code-mixing. The mixture of dual language in rojak content poses challenges for NER models, especially if they are trained on monolingual data. Most NER models, including those in SpaCy, are typically trained on monolingual corpora. The rojak content introduces ambiguity that makes it hard for the model to understand the context, which is crucial for accurate entity recognition. Also, different syntactic and semantic structures in English and Malay can confuse a model not trained to handle such diversity.

Even Malaya is trained on Malay and English content, there might also be some unfamiliar entities that make the model hard to recognize entities that are specific to a language or culture it wasn't trained on.

Reason no consider POS tagging and Parsing

- ***Lack of Standardisation:***
  Code mixing involves the seamless integration of multiple languages or dialects within a single text. This means that there may be a lack of standardized rules for how different languages or dialects are combined. POS taggers and parsers typically rely on consistent grammatical structures and patterns within a single language. When code mixing occurs, these structures can become irregular or non-standard, making it difficult for NLP tools to accurately analyze the text.
- ***Contextual Ambiguity:***
  Code-mixed text can be contextually ambiguous, as the intended meaning of a word or phrase may depend on the language or dialect in which it is used. POS taggers and parsers may have difficulty disambiguating between different possible interpretations, leading to errors in analysis.

- ***Non-Standard Vocabulary:***
  Code mixing often involves the use of non-standard vocabulary, slang, abbreviations, or transliterations of words from one language to another. These variations can be challenging for POS taggers to categorize accurately, as they may not fit neatly into predefined linguistic categories.

- ***Limited Code-Mixing Tools:***

There is a lack of specialized tools and resources for handling code-mixed text compared to standard text in a single language. Many NLP models and tools are primarily designed for monolingual or well-structured bilingual text, which may not effectively handle the complexity of code-mixing.

3.2.7 Data Splitting

Data splitting is a crucial step in machine learning and data analysis, especially when you're working with supervised learning tasks. It involves dividing your dataset into distinct subsets for different purposes, typically into training, validation, and test sets.

Purpose of Each Dataset:

- ***Training Set***: Used for teaching the model.
- ***Validation Set***: Helps tune model parameters and monitor performance.
- ***Test Set***: Assesses the model's final performance.

The ideal split percentages depend on your dataset size, but common choices include 70-80% for training and the rest for validation and testing. For larger datasets, you might use 80-90% for training. Balancing these proportions ensures effective learning and robust evaluation.

We snippet employs the `train_test_split` function from Scikit-learn (`sklearn.model_selection`) to partition the DataFrame's data into training, validation, and test sets. It assumes the 'stemmed_token' column contains the text data while 'sentiment' is the target variable to predict. Initially, the independent variable `'X'` is assigned the 'stemmed_token' column, and the dependent variable `'y'` holds the 'sentiment' column. The data is divided into two stages: first, it's split into a training set (`X_train, y_train`) and a temporary set (`X_temp, y_temp`) using a 70-30 ratio. Then, the temporary set is further divided into validation (`X_val, y_val`) and test (`X_test, y_test`) sets using a **50-50** split. The random_state parameter ensures reproducibility by fixing the random seed to **42** during the splitting process. This division allows for training the model on one portion, tuning it on the validation set, and finally evaluating its performance on the unseen test data.

For sentiment analysis, especially when using models like SVM and Multinomial Naive Bayes, we typically use preprocessed text that retains the most relevant information while reducing complexity. Therefore, only ‘stemmed_token’ will be used as the feature since it has already undergone preprocessing like tokenization and stemming, which are standard practices in text analysis. However, the NER result is not going to be applied because the result provided is worse, it will only add noise to the features. The target variable will be the sentiment column, which is the result of the sentiment analysis using a pre-trained model provided by Malaya library. Regarding the portion, a 70-15-15 split will be used.

3.2.8 Feature Extraction (Dimension Reduction)

In this section, we'll conduct feature extraction, aiming to streamline raw data into a more condensed form, retaining vital information while minimizing dimensionality. This process, known as feature extraction, simplifies the dataset by emphasizing crucial traits and minimizing irrelevant details.

Specifically, we'll implement:

- ***TF-IDF (Term Frequency-Inverse Document Frequency)***
- ***PCA (Principal Component Analysis)***

Two methods instrumental in transforming and condensing data for improved efficiency in machine learning models.

3.2.8.1 TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF is a natural language processing technique that determines a word's relative importance inside a text about a wider collection of documents (Anirudha, 2021). This is computed by taking into account a term's rarity over the entire corpus (IDF) as well as its frequency in the document (TF). The intention is to draw attention to terms that are common in one document but unusual in the collection as a whole, presuming that these words are more representative of the content of that particular document (Fatih, n.d.). TF-IDF lessens the weight of common terms that are present across the corpus while giving particular words in a document higher scores. This procedure aids in the identification of important terms for analysis or retrieval.

`‘Safely_convert_and_join’` defined intended to handle converting string representations of lists back into lists and joining their elements. The try-except block attempts to join elements in each row; if successful, it returns the joined string, otherwise, it raises an error message. The code then prepares the text data for TF-IDF (Term Frequency-Inverse Document Frequency) by joining the tokenized and stemmed text back into strings for each dataset split (training, validation, and test). Subsequently, it initializes a TF-IDF Vectorizer from Scikit-learn (`‘TfidfVectorizer()’`), which will convert the text data into numerical TF-IDF representations. The training data is both fitted and transformed using `‘fit_transform’`, while the validation and test sets are transformed using `‘transform’` to ensure they are represented in the same feature space as the training data. This process prepares the text data for machine learning models that require numerical inputs by converting them into TF-IDF representations.

These vectors are now ready to be used for training and testing the SVM and Multinomial Naive Bayes models. The shapes of these TF-IDF vectors are as follows:

- ***Training set:*** 3447 samples and 10859 features.
- ***Validation set:*** 739 samples and 10859 features.
- ***Test set:*** 739 samples and 10859 features.

Sparsity of the TF-IDF

We imports csr_matrix from scipy.sparse and converts the TF-IDF vectors derived from the training data (X_train_tfidf) into a sparse matrix format using csr_matrix. The code then calculates the sparsity of this sparse TF-IDF matrix. Sparsity represents the proportion of zero or non-filled elements in a matrix compared to its total size. The calculation involves subtracting the ratio of non-zero elements (nnz) to the total number of elements in the matrix from 1, and then converting the result to a percentage format.

3.2.8.2 PCA (Principal Component Analysis)

PCA is a well-discussed machine learning technique online, but numerous resources delve deeply into its complexities. Most of us seek a simplified understanding of how it operates without getting lost in intricate details (Jaadi, 2023). The transformed features resulting from PCA are termed Principal Components, widely used in exploratory data analysis and predictive modelling. This technique aims to extract strong patterns from datasets by reducing their variance, effectively searching for lower-dimensional surfaces to represent high-dimensional data. PCA operates by prioritizing attributes with high variance, as they tend to delineate classes effectively, thus facilitating dimensionality reduction. Its real-world applications span image processing, movie recommendation systems, and optimizing power allocation across communication channels. Acting as a feature extraction method, PCA retains crucial variables while discarding less significant ones (JavatPoint, n.d.).

We imports PCA (Principal Component Analysis) from Scikit-learn's decomposition module and initializes it to retain 95% of the variance in the data using `'n_components=0.95'`. The code attempts to fit and transform the TF-IDF vectors obtained from the text data; however, the `'toarray()'` method tries to convert sparse TF-IDF matrices to dense arrays, causing a memory error due to potentially large data sizes. Assuming the data fits into memory, the `'fit_transform'` method applies PCA on the training TF-IDF vectors, finding the principal components that capture 95% of the variance. Subsequently, the validation and test TF-IDF vectors are transformed using the already-fitted PCA model to project them onto the reduced-dimensional space defined by the principal components obtained from the training data.

3.2.8.3 Truncated SVD

Truncated Singular Value Decomposition (SVD) involves decomposing a matrix M into three constituent matrices: U, Σ, and V. While similar to PCA, SVD differs in that it factors the data matrix directly, unlike PCA which operates on the covariance matrix. SVD is commonly employed to determine the principal components within a matrix, often serving as an underlying technique for this purpose (Garreta, n.d.).

We explores different component counts for dimensionality reduction using TruncatedSVD from Scikit-learn. To begin, a list named `'n_components_list'` with various component counts is created. After that, the method iterates through this list, reducing the dimensionality of the TF-IDF vectors in the training and validation sets according to each component count using TruncatedSVD. Using the reduced data (`'X_train_svd'`), it trains an example Support Vector Machine (SVM) model in each iteration that can be replaced by other models. The accuracy scores obtained from evaluating the model's performance on the validation set (`'X_val_svd'`) are gathered in the `'performance'` list.

3.3 Model Training

Model training stands as a pivotal phase in the realm of machine learning and deep learning, where a model undergoes the process of learning from data to subsequently make predictions or accomplish specific tasks. This phase involves the model acquiring the capability to discern patterns, relationships, and valuable insights within the provided training dataset.

In this section of the methodology, we embark on model training endeavours, encompassing both traditional models, such as Multinomial Naive Bayes (MultinomialNB) and Support Vector Machine (SVM), as well as a deep learning model utilizing Long Short-Term Memory (LSTM).

Reasons for Model Selection:

- Multinomial Naive Bayes (MultinomialNB): MultinomialNB is chosen as one of the traditional models due to its suitability for text classification tasks, especially when dealing with count-based features like word frequencies. Its inherent simplicity, efficiency, and effectiveness in handling text data make it a robust choice for sentiment analysis.

- Support Vector Machine (SVM): SVM, a powerful classification algorithm, is another traditional model selected for its versatility in handling text data. SVM can efficiently capture complex relationships between features and offers excellent generalization capabilities. Its ability to handle high-dimensional data and adapt to different kernel functions adds to its appeal for sentiment analysis tasks.

- Long Short-Term Memory (LSTM): LSTM, a deep learning model, is chosen for its proficiency in capturing sequential dependencies and long-range dependencies in text data. Sentiment analysis often involves understanding the contextual flow of language, making LSTM a strong candidate. Its recurrent architecture allows it to consider the entire input sequence, which can be instrumental in discerning nuanced sentiment patterns.

TextClassifier Class for Traditional Models:

To accommodate multiple traditional machine learning models, a versatile class named "TextClassifier" is introduced. This class is designed to handle various machine learning models, such as Multinomial Naive Bayes (MultinomialNB) and Support Vector Machine (SVM). It encompasses methods for data preparation, training, prediction, evaluation, and cross-validation. The class is adaptable for hyperparameter tuning and extensible to accommodate additional models or techniques.

Multinomial Naive Bayes Training:

Multinomial Naive Bayes is a classic choice for text classification tasks. It works effectively with count-based or TF-IDF features. In this phase, the MultinomialNB model is initialized and trained using TF-IDF features to capture text patterns.

SVM Using TF-IDF Features:

Support Vector Machine (SVM) is another traditional model that excels in text classification. Here, the SVM model is initialized and trained on TF-IDF vectors, seeking to find the optimal hyperplane that separates classes in the high-dimensional TF-IDF feature space.

SVM Using TruncatedSVD Features:

An alternative approach involves reducing the dimensionality of the TF-IDF feature space using Truncated Singular Value Decomposition (TruncatedSVD). This dimensionality reduction technique enhances SVM's speed and robustness in high-dimensional data scenarios.

LSTM-based Neural Network (Deep Learning):

Deep learning is harnessed with an LSTM-based neural network, designed to capture long-term dependencies within text data. Key considerations include word embeddings, architecture design, normalization, regularization, and multi-class classification adjustments. The LSTMTextClassifier class is introduced to facilitate the building and training of LSTM models.

These diverse model choices are driven by their respective strengths and suitability for sentiment analysis tasks. MultinomialNB and SVM excel in traditional text classification, while LSTM offers the ability to capture intricate sequential dependencies within text data, making it well-suited for this project's goals.

3.4 Model Evaluation

Model evaluation is a pivotal stage in the machine learning workflow, crucial for assessing how well a trained model performs on unseen data. It provides insights into the model's accuracy, reliability, and suitability for its intended task, aiding in informed decisions regarding deployment or further enhancements. This section delves into the methodologies employed to evaluate the sentiment analysis models, encompassing both traditional machine learning models and deep learning models.

Traditional Machine Learning Models Evaluation:

1. **Feature Engineering**: To prepare the text data for model evaluation, the 'stemmed_token' column is transformed into a consolidated text representation. TF-IDF features are then extracted using the TfidfVectorizer.

2. **Label Encoding**: Sentiment labels are encoded using the LabelEncoder, converting them into numerical format for model compatibility.

3. **Data Splitting**: The dataset is divided into three subsets: training, validation, and test sets, using the train_test_split function. Approximately 70% of the data is assigned to the training set, with 15% each for validation and testing.

4. **Truncated Singular Value Decomposition (TruncatedSVD)**: TruncatedSVD is applied to reduce the dimensionality of the TF-IDF features. This dimensionality reduction enhances the speed and robustness of the SVM model, reducing noise in high-dimensional data.

5. **Cross-Validation**: The models (Multinomial Naive Bayes, SVM, and Truncated SVM) are evaluated using cross-validation. Scoring metrics such as accuracy, precision (macro), recall (macro), and F1 score (macro) are utilized. Results for each model are displayed to assess their performance.

Deep Learning Model (LSTM) Evaluation:

1. **Tokenizer Fitting**: The Keras Tokenizer is fitted on the entire dataset, allowing it to build a vocabulary and assign numerical values to words.

2. **LSTMTextClassifier Initialization**: The LSTMTextClassifier class is initialized with various hyperparameters, including maximum sequence length, LSTM units, embedding dimensions, and others.

3. **Dataset Splitting:** The dataset is divided into training, validation, and test sets, with 80%, 20%, and 20% of the data allocated to each, respectively.

4. **Model Training**: The LSTM model is trained using the training and validation sets. This involves tokenizing and padding the input texts, designing the neural network architecture, and using categorical cross-entropy loss for optimization.

5. **Model Evaluation**: The trained LSTM model is evaluated using the test set. Evaluation metrics such as accuracy, precision, recall, and F1 score are computed and reported for assessing the model's performance.

In summary, this model evaluation section provides a comprehensive approach to assessing the sentiment analysis models' performance, ranging from traditional machine learning methods to deep learning models. It ensures a thorough understanding of how well these models generalize to new, unseen data, ultimately guiding decision-making processes.

3.5 Model Tuning

Traditional Machine Learning Models (Multinomial Naive Bayes, SVM):

For tuning the traditional machine learning models, Grid Search Cross-Validation (GridSearchCV) can be employed. GridSearchCV systematically explores a predefined hyperparameter grid, searching for the combination that yields the best model performance. In the context of sentiment analysis, hyperparameters like alpha (for Multinomial Naive Bayes), kernel type, and regularization parameters (for SVM) can be tuned.

Deep Learning Model (LSTM):

For enhancing the LSTM-based deep learning model, several adjustments and hyperparameter tuning strategies can be considered:

1. **Dropout:** Implement dropout layers within the LSTMTextClassifier model to prevent overfitting. Dropout randomly deactivates a fraction of neurons during training, promoting model generalization.

2. **L1 and L2 Regularization:** Introduce L1 and L2 regularization to the LSTM layers. Regularization techniques help control overfitting and enhance model robustness.

3. **Learning Rate Adjustment**: Experiment with different learning rates (e.g., higher or lower than the initial learning rate) to observe their impact on training and convergence speed.

4. **Epoch Testing**: Vary the number of training epochs to determine the optimal number of iterations for model convergence. This helps prevent overfitting and underfitting.

By systematically adjusting these hyperparameters and employing techniques like dropout and regularization, the LSTM model can be fine-tuned to achieve better generalization and sentiment analysis performance. Experimentation and validation on a validation dataset are crucial to identify the best hyperparameter settings.

3.6 User Interface

UI allows users to input text, obtain sentiment predictions, and visualize the sentiment using images. Here is an overview of the methodology behind this UI:

The first step is to import the necessary libraries and dependencies. This includes libraries for UI development (Tkinter), image processing (PIL), and deep learning model loading (Keras). Additionally, pickle is used for loading a tokenizer and label encoder. The UI loads a pre-trained deep learning model for sentiment analysis. This model has been trained to predict sentiment labels (e.g., positive, negative) based on text input.

A key function, 'predict,' is defined to perform sentiment analysis. It tokenizes the user's input text, processes it, and then uses the loaded deep learning model to predict the sentiment. The predicted sentiment label is displayed to the user. The UI components are created using Tkinter. This includes defining the main window, customizing fonts, and loading and resizing images representing different sentiments (positive, negative, neutral).

The UI includes a text input area where users can enter the text they want to analyze. This input area is designed with a specific height, width, and font style for readability. A "Predict Sentiment" button is provided, prompting users to initiate the sentiment analysis process. The button is styled for a visually appealing design.

Upon clicking the "Predict Sentiment" button, the user's input text is analyzed. If the input contains at least five words, the sentiment analysis process proceeds. Otherwise, the user is prompted to input more text. The sentiment analysis results, including the predicted sentiment label, are displayed within the UI. Depending on the predicted sentiment, the text color and a corresponding sentiment image are updated to provide visual feedback to the user.

An auxiliary function is used to resize images while maintaining their aspect ratio. This function ensures that the sentiment images fit the designated space in the UI. The UI operates within the main Tkinter loop, continually managing user interactions and responding to button clicks. This loop keeps the UI interactive and functional.

In practical usage, users input text into the provided text input area. By clicking the "Predict Sentiment" button, the UI initiates the sentiment analysis process and provides users with feedback on the predicted sentiment, enhancing the overall user experience.

3.7 Tools or software utilised for analysis.

Tools comparison	**Remark**	**Jupyter notebook**	**PyCharm**	**DataSpell**
Type of license and open source license	State all types of license	3-Clause BSD License (also known as New or Revised or Modified BSD License, or the BSD-3-Clause license	Community (free and open-source) and a Professional (commercial) license.	Community (free and open-source) and a Professional (commercial) license.
Year founded	When is this tool being introduced?	2014	2010	2021
Founding company	Owner	Fernando Pérez	JetBrains	JetBrains
License Pricing	Compare the prices if the license is used for development and business/commercialization	Free	**Community Edition**: Free and open-source. **Professional Edition**: -free for students and teachers -individuals: $99 first year -organisations: $249 first year	**Community Edition**: Free and open-source. **Professional Edition**: -free for students, teachers and classroom assistance -individuals: $99 first year -organisations: $229 first year
Supported features	What features that it offer?	Text and HTML, code and output, visualisations, multimedia, data, allows users to share live code in the form of notebooks, supports different programming languages	Code completion, intelligent code analysis, debugging, unit testing, version control integration, database tools, web development support, scientific tools, and more.	Intelligent Jupyter notebooks with smart coding assistance, interactive outputs, local and remote connections, and debugging tools Interactive Python scripts with code cells, scientific console, data and visualisation outputs, and debugging tools Built-in tools and integrations for version control, terminal, database,

				R language, plugins, and more
Common applications	In what areas this tool is usually used?	Data science, mathematics, machine learning, and other research projects that involve visualisations of data or formulas.	Software development, data science, scientific computing, web development (Django, Flask, etc.), artificial intelligence, and machine learning.	- Data cleaning and preprocessing - Data visualisation and exploration -Statistical analysis and hypothesis testing -Machine learning model development and evaluation -Model deployment and monitoring
Customer support	How the customer support is given, e.g. proprietary, online community, etc.	Online Community	Proprietary Support, Online Community	Online Community
Limitations	The drawbacks of the software	Notebooks aren't self-contained, Session state cannot be saved easily, No interactive debugging or other IDE features	- necessitates a paid licence to unlock its complete feature set - exhibits heavier resource usage - The learning curve for beginners due to its feature-rich nature.	- necessitates a paid licence to unlock its complete feature set - It may not support all the features or libraries that are available in other tools or platforms and might have some bugs as it is a new product

3.8 Justify why the selected tool is suitable

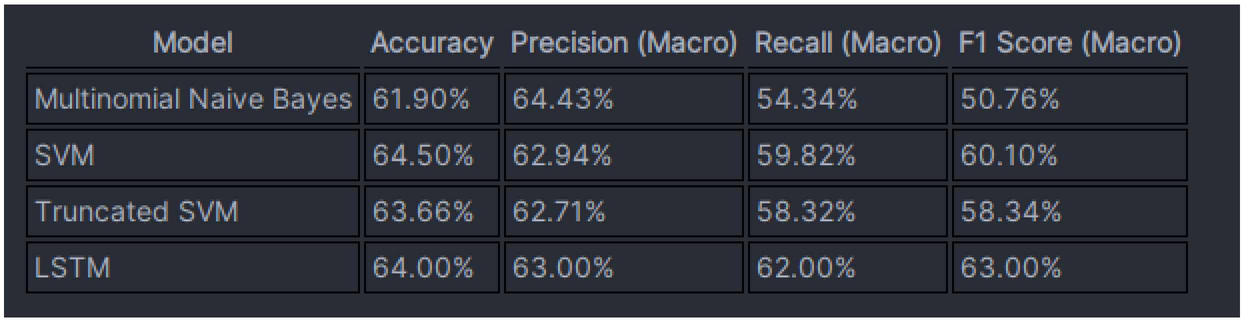
PyCharm stands out as an excellent choice for sentiment analysis in the Malaysian Rojak language due to several compelling reasons. Its robust features and tools cater to language-specific analysis, facilitating Natural Language Processing (NLP) operations effectively. Seamless integration with popular Python libraries like Malaya, NLTK, and spaCy simplifies code-mixed Malaysian Rojak language handling, aiding tasks such as text preprocessing, tokenization, stemming, and lemmatization. The accessibility of these libraries within PyCharm streamlines essential language preparation tasks crucial for sentiment analysis.

The IDE's interactive programming environment and robust debugging functionalities expedite the iterative development of sentiment analysis models for the Malaysian Rojak language. PyCharm's debugger allows users to navigate code, inspect variables, and identify issues, ensuring the accuracy and reliability of sentiment analysis models. Its interactive Python console and live coding feature enable real-time experimentation with diverse NLP techniques, facilitating the creation of sentiment analysis algorithms tailored to the linguistic nuances present in the Malaysian Rojak language.

Furthermore, PyCharm promotes collaboration and teamwork by integrating with collaborative platforms and supporting version control systems like Git. This facilitates code sharing, and version tracking, and enhances team productivity, particularly beneficial for collaborative projects focusing on sentiment analysis in the Malaysian Rojak Language. Overall, PyCharm emerges as an effective and apt platform for delving into the subtleties of sentiment analysis in the Malaysian Rojak language, owing to its feature-rich environment, debugging prowess, and collaborative tools..

4.0 Analysis and Findings

4.1 Overall Comparison



Model	Accuracy	Precision (Macro)	Recall (Macro)	F1 Score (Macro)
Multinomial Naive Bayes	61.90%	64.43%	54.34%	50.76%
SVM	64.50%	62.94%	59.82%	60.10%
Truncated SVM	63.66%	62.71%	58.32%	58.34%
LSTM	64.00%	63.00%	62.00%	63.00%

The LSTM model demonstrates accuracy levels comparable to those of the SVM and Truncated SVM models, with a marginally higher accuracy than the Multinomial Naive Bayes model. Precision and recall metrics across different sentiment classes reveal that the LSTM model performs particularly well in classifying 'neutral' sentiments, while its performance for the 'positive' class is less effective. Compared to the earlier models, the LSTM model achieves a similar or slightly improved balance between precision and recall, especially for the 'neutral' class. The F1-scores indicate a reasonable equilibrium between precision and recall for each sentiment class, though the model struggles with the 'positive' class. Macro average scores suggest that the LSTM model's overall performance is comparable to that of the SVM models, reflecting a similar effectiveness across all sentiment classes.

Interpretation

The LSTM model excels in classifying neutral sentiments but exhibits challenges with the positive sentiment class, as evidenced by lower precision, recall, and F1-scores for this category. The observed performance pattern implies that the model may be more adept at distinguishing neutral sentiments from others but might benefit from additional improvements or training data to enhance the classification of positive sentiments. Despite these challenges, the LSTM model's performance remains competitive with traditional machine learning models like SVM, highlighting its potential for handling complex language patterns in sentiment analysis.

4.2 Training vs. Validation Accuracy

The LSTM model quickly achieves near-perfect accuracy on the training set (99.97% by Epoch 6). However, validation accuracy remains relatively stagnant at approximately 66%, indicating overfitting. This overfitting suggests that the model has learned the training data, including its noise and fluctuations, too well, leading to diminished generalization capability on new, unseen data.

Increasing Validation Loss

The observed increase in validation loss with each epoch further confirms overfitting. As the model becomes more attuned to the training data, its performance on validation data deteriorates, underscoring the need for strategies to mitigate overfitting.

Test Set Evaluation

Performance metrics on the test set—precision, recall, and F1-score—provide a more realistic assessment of the model's performance on unseen data. An accuracy of 64% indicates that the model performs better than random guessing but still has significant room for improvement.

To address the problem, the following steps are recommended:

1. **Dropout:** Incorporate dropout layers in the model to randomly set a fraction of input units to zero during training. This technique helps to prevent overfitting by reducing the model's reliance on specific neurons.

2. **Regularization:** Apply L1 or L2 regularization to the LSTM and Dense layers to penalize overly complex models and encourage simpler, more generalizable solutions.

3. **Hyperparameter Tuning:** Experiment with various values for LSTM units, embedding dimensions, batch size, and learning rates. Fine-tuning these hyperparameters can enhance the model's performance and reduce overfitting.

4.3 Model Tuning Results

Multinomial Naive Bayes (Alpha = 0.5) Cross-Validation Results

- **Accuracy:** The model achieves an average accuracy of approximately 63.76%. This indicates that the Multinomial Naive Bayes model correctly predicts the target class around 63.76% of the time across different data subsets.
- **Precision (Macro):** The average precision is about 63.10%. This metric reflects the model's effectiveness in identifying positive instances among those it labels as positive.
- **Recall (Macro):** The average recall is approximately 57.53%, demonstrating the model's ability to identify all actual positive instances.
- **F1 Score (Macro):** With an average F1 score of 55.83%, the Multinomial Naive Bayes model shows a moderate balance between precision and recall, suitable for scenarios requiring equilibrium between these metrics.

SVM (C = 1, Gamma = 1, Kernel = Linear) Cross-Validation Results

- **Accuracy:** The average accuracy is approximately 64.50%, representing a slight improvement over the Multinomial Naive Bayes model.
- **Precision (Macro):** The average precision is around 62.94%.
- **Recall (Macro):** The average recall is about 59.82%.
- **F1 Score (Macro):** The SVM model achieves an average F1 score of 60.10%, surpassing the Multinomial Naive Bayes model and indicating a better balance between precision and recall.

Truncated SVM (C = 1, Gamma = 1) Cross-Validation Results

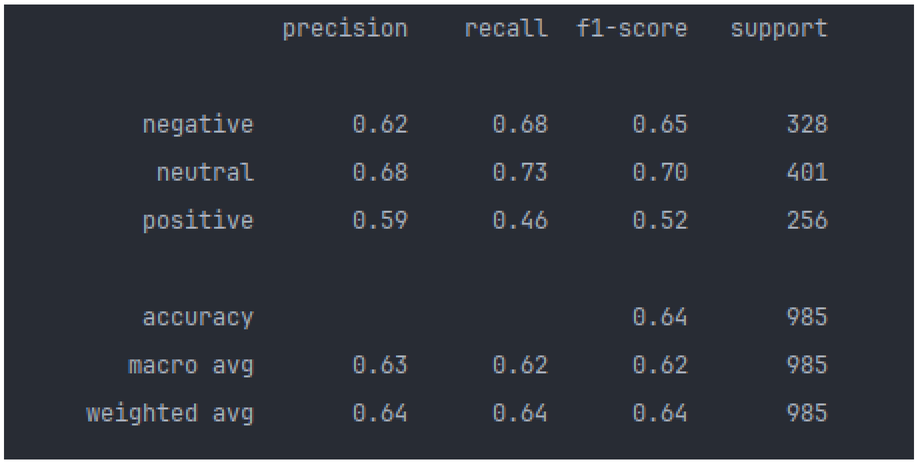
- **Accuracy:** The average accuracy is approximately 63.71%, which is comparable to the Multinomial Naive Bayes model.
- **Precision (Macro):** The average precision stands at about 62.78%.
- **Recall (Macro):** The average recall is approximately 58.20%.
- **F1 Score (Macro):** The average F1 score is 58.13%, showing a similar performance to the regular SVM model but with a slight decrease in recall and F1 score.

Overall Interpretation

The SVM model with a linear kernel demonstrates a marginal edge over the other models regarding overall accuracy and F1 score. The performance metrics across all models are relatively moderate, with accuracies ranging from 63% to 65%. Precision and recall values indicate that while the models effectively predict positive instances to a reasonable extent, there is still potential for improvement, particularly in recall.

The F1 scores across all models reflect a moderate balance between precision and recall, suggesting that further improvements could enhance their overall performance while the models achieve an acceptable equilibrium. Addressing the recall deficits and refining the models to improve precision and recall could lead to more robust and effective predictions.

4.4 LSTM Model Tuning



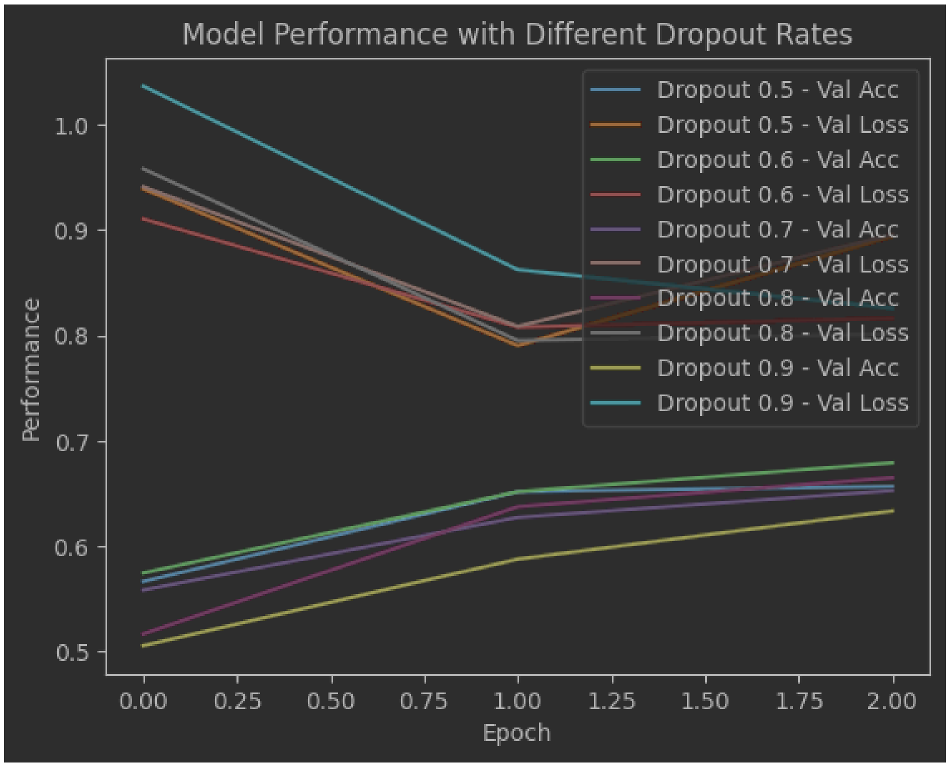
```
              precision    recall  f1-score   support

    negative       0.62      0.68      0.65       328
     neutral       0.68      0.73      0.70       401
    positive       0.59      0.46      0.52       256

    accuracy                           0.64       985
   macro avg       0.63      0.62      0.62       985
weighted avg       0.64      0.64      0.64       985
```

4.4.1 Impact of Dropout Layers

The inclusion of dropout layers in the LSTM model did not significantly improve performance compared to the model without dropout layers. Both models improved loss and accuracy as training progressed, reflecting the model's capability to adjust parameters to minimize loss and enhance predictions.

Epoch Analysis

The highest validation accuracy was observed at the 7th epoch (val_accuracy: 0.6660), indicating optimal performance on the validation dataset at this point. However, the 3rd epoch emerges as the optimal choice when considering both validation accuracy and loss, providing the lowest validation loss (0.8732) and high accuracy (0.9228). The elevated accuracy in later epochs suggests that the model may have memorized the training data, potentially leading to poor generalization of new data. Starting from the 4th epoch, a divergence is noted where training accuracy increases while validation accuracy decreases, signalling possible overfitting. This is further supported by the observation that validation loss initially decreases but starts to increase from the 3rd epoch onwards, while training loss consistently decreases.



Test Set Performance

Performance metrics on the test set reveal that the model performs best in classifying ‘neutral’ sentiments, followed by ‘negative’ and ‘positive’ sentiments. The dataset's predominance of neutral labels likely influences this result.

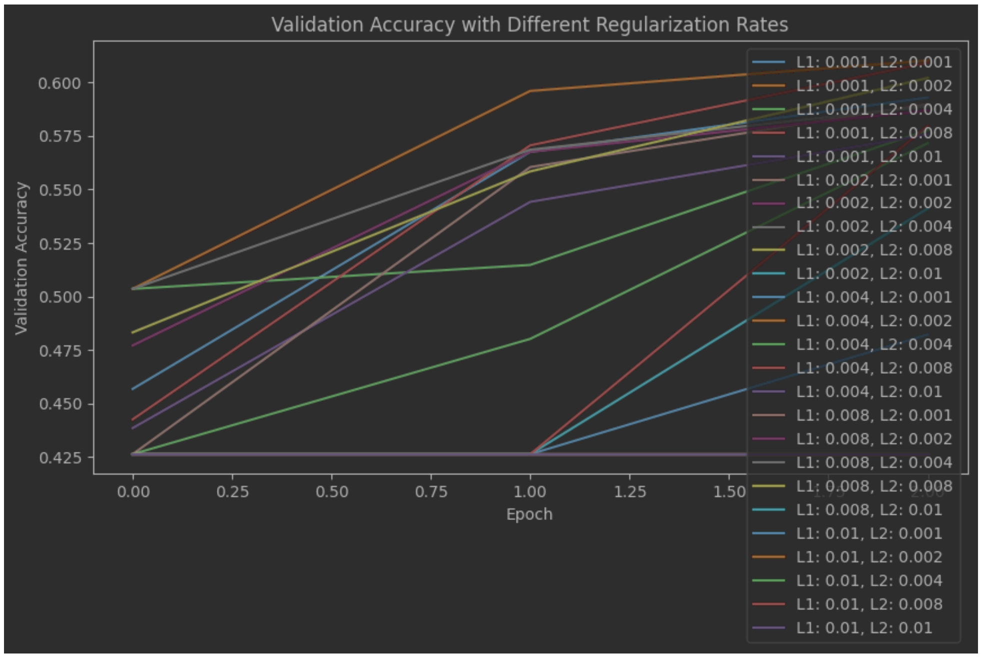
Comparison of Dropout Rates

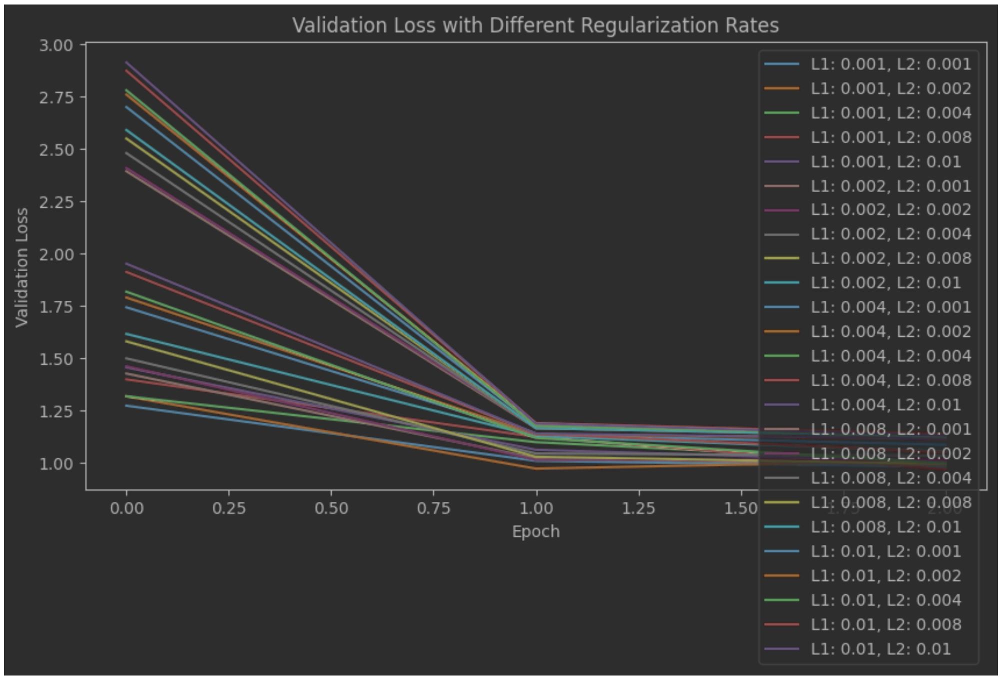
The performance of the model with various dropout rates after 3 epochs is as follows:

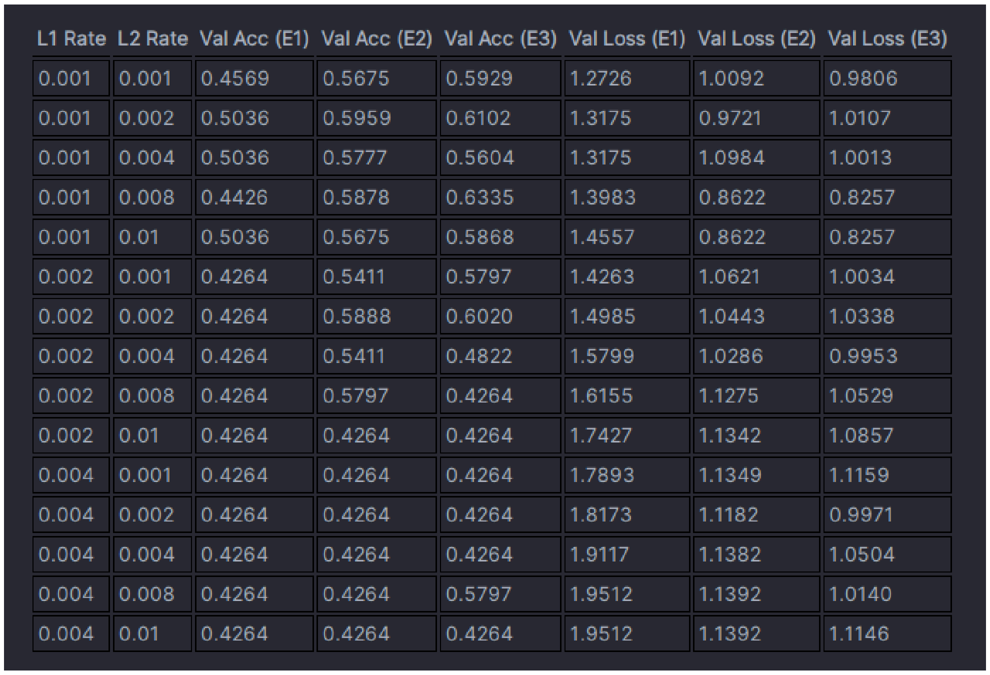
- **Dropout Rate 0.5:** Val Accuracy = 65.69%, Val Loss = 0.8942
- **Dropout Rate 0.6:** Val Accuracy = 67.92%, Val Loss = 0.8162
- **Dropout Rate 0.7:** Val Accuracy = 65.28%, Val Loss = 0.8955
- **Dropout Rate 0.8:** Val Accuracy = 66.50%, Val Loss = 0.8010
- **Dropout Rate 0.9:** Val Accuracy = 63.35%, Val Loss = 0.8257

Among these, a dropout rate of 0.6 exhibits the most favourable balance between validation accuracy and loss. It achieves the highest validation accuracy (67.92%) and a relatively low validation loss (0.8162), suggesting that this configuration is the most effective at generalizing to unseen data among those tested.

4.4.2 Impact of Regularisation







L1 Rate	L2 Rate	Val Acc (E1)	Val Acc (E2)	Val Acc (E3)	Val Loss (E1)	Val Loss (E2)	Val Loss (E3)
0.001	0.001	0.4569	0.5675	0.5929	1.2726	1.0092	0.9806
0.001	0.002	0.5036	0.5959	0.6102	1.3175	0.9721	1.0107
0.001	0.004	0.5036	0.5777	0.5604	1.3175	1.0984	1.0013
0.001	0.008	0.4426	0.5878	0.6335	1.3983	0.8622	0.8257
0.001	0.01	0.5036	0.5675	0.5868	1.4557	0.8622	0.8257
0.002	0.001	0.4264	0.5411	0.5797	1.4263	1.0621	1.0034
0.002	0.002	0.4264	0.5888	0.6020	1.4985	1.0443	1.0338
0.002	0.004	0.4264	0.5411	0.4822	1.5799	1.0286	0.9953
0.002	0.008	0.4264	0.5797	0.4264	1.6155	1.1275	1.0529
0.002	0.01	0.4264	0.4264	0.4264	1.7427	1.1342	1.0857
0.004	0.001	0.4264	0.4264	0.4264	1.7893	1.1349	1.1159
0.004	0.002	0.4264	0.4264	0.4264	1.8173	1.1182	0.9971
0.004	0.004	0.4264	0.4264	0.4264	1.9117	1.1382	1.0504
0.004	0.008	0.4264	0.4264	0.5797	1.9512	1.1392	1.0140
0.004	0.01	0.4264	0.4264	0.4264	1.9512	1.1392	1.1146

Impact of Regularisation Rates

The results across different regularization rates indicate a pattern of underfitting, particularly at higher regularization rates. While regularization is crucial for preventing overfitting by penalizing complex models, excessively high regularization can lead to underfitting, as observed in our results. The performance of models with the highest regularization rates was notably similar, suggesting that these configurations may be too restrictive, thereby failing to capture the complexity of the data.

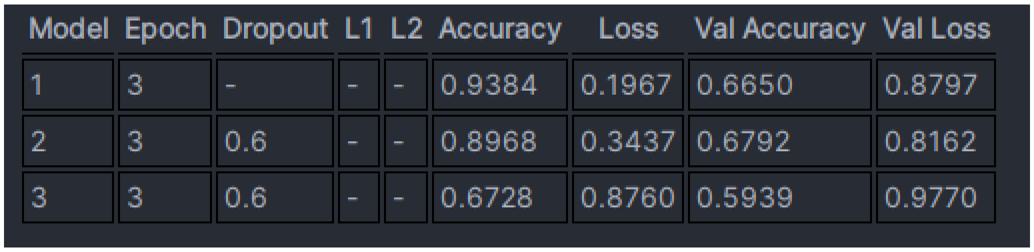
Conversely, lower regularization rates permit more model complexity, which correlates with improved validation accuracy. Specifically, configurations with lower regularization rates consistently outperformed those with higher rates regarding validation accuracy. This observation implies that moderate regularization allows the model to balance complexity and generalization.

Best Configuration

The optimal model configuration is characterized by the highest validation accuracy and a reasonable gap between training and validation accuracy, indicating effective generalization. Based on the results, configurations with moderate regularization rates, such as L1: 0.002 and L2: 0.002, show promise. These configurations achieve higher validation accuracies while maintaining a more balanced performance between training and validation data. This suggests that moderate regularization effectively prevents overfitting without introducing significant underfitting.

4.5 Analysis Before and After Regularization in the LSTM Model

Here the model we select before overfitting(val loss < val loss+1):



Model	Epoch	Dropout	L1	L2	Accuracy	Loss	Val Accuracy	Val Loss
1	3	-	-	-	0.9384	0.1967	0.6650	0.8797
2	3	0.6	-	-	0.8968	0.3437	0.6792	0.8162
3	3	0.6	-	-	0.6728	0.8760	0.5939	0.9770

Model 1: Before Regularization

- **Validation Accuracy:** The model exhibits a validation accuracy of 66.50%, which is higher than that of the model incorporating dropout and regularization but slightly lower than the model with only dropout.
- **Validation Loss:** The validation loss for this model remains relatively stable and does not indicate pronounced overfitting.

Model 2: Applying Dropout Layer

- **Validation Accuracy:** This model reaches a validation accuracy of 67.92% by the third epoch, demonstrating improved performance over the other models.
- **Validation Loss:** While validation loss shows a slight increase, indicating potential overfitting, it is less severe compared to the model with both dropout and regularization.

Model 3: Applying Dropout Layer & L1 & L2 Regularization

- **Validation Accuracy:** This model starts with lower accuracy compared to the other models and achieves a validation accuracy of 59%, which is notably lower than the model with only dropout.
- **Validation Loss:** The inclusion of both dropout and regularization appears to contribute to higher validation loss, suggesting potential underfitting due to the combined regularization effects.

Interpretation

The analysis indicates that the addition of both dropout and L1 & L2 regularization seems to inhibit the model's learning effectiveness, as evidenced by the lower validation accuracy and higher validation loss. The combined impact of dropout and regularization may be excessively restrictive, leading to underfitting where the model fails to capture the complexity of the data.

In contrast, the model with only dropout (0.6) demonstrates the best validation accuracy, suggesting that dropout alone is sufficient to mitigate overfitting for this particular dataset. The model without dropout performs reasonably well but does not achieve the same level of performance as the dropout-only model.

Given these observations, the model with a dropout rate of 0.6 at epoch 3 is identified as the optimal configuration for further hyperparameter tuning. This model provides the best balance between validation accuracy and loss, indicating effective management of overfitting without compromising the model's ability to learn from the data. Future tuning efforts should focus on this configuration to refine model performance further.

4.6 Hyperparameters Tuning For LSTM Model

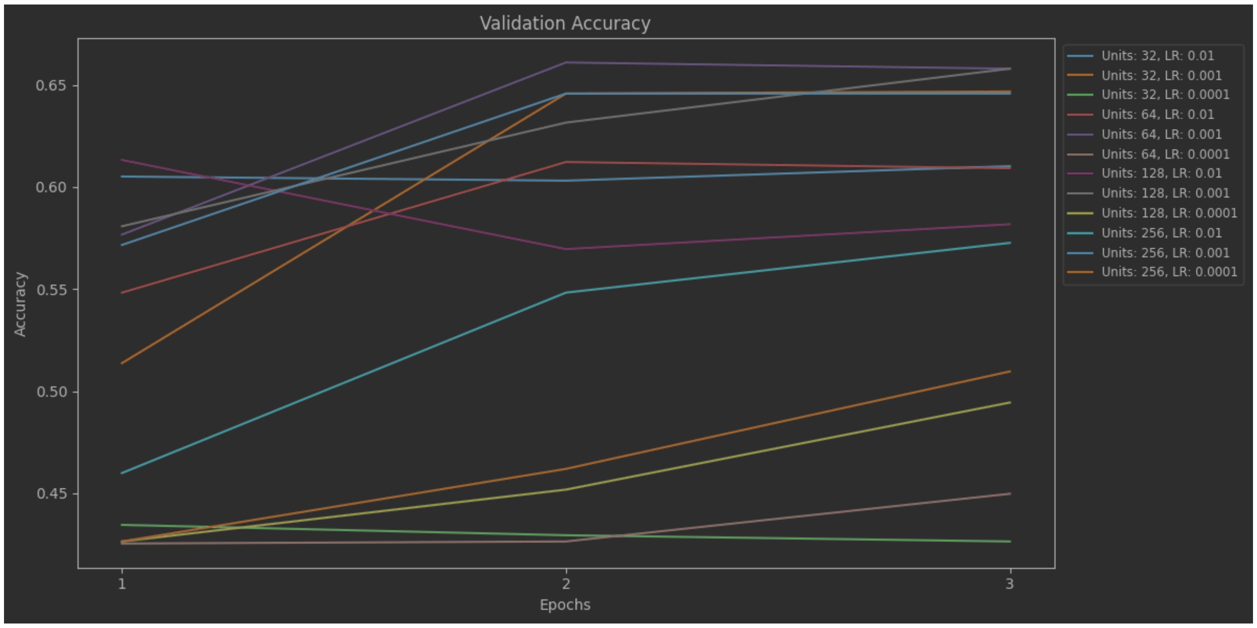
To further optimize the LSTM model's performance, we adjusted key hyperparameters, specifically the number of LSTM units and the learning rate. These hyperparameters are critical in the model's ability to learn and generalize from the data.

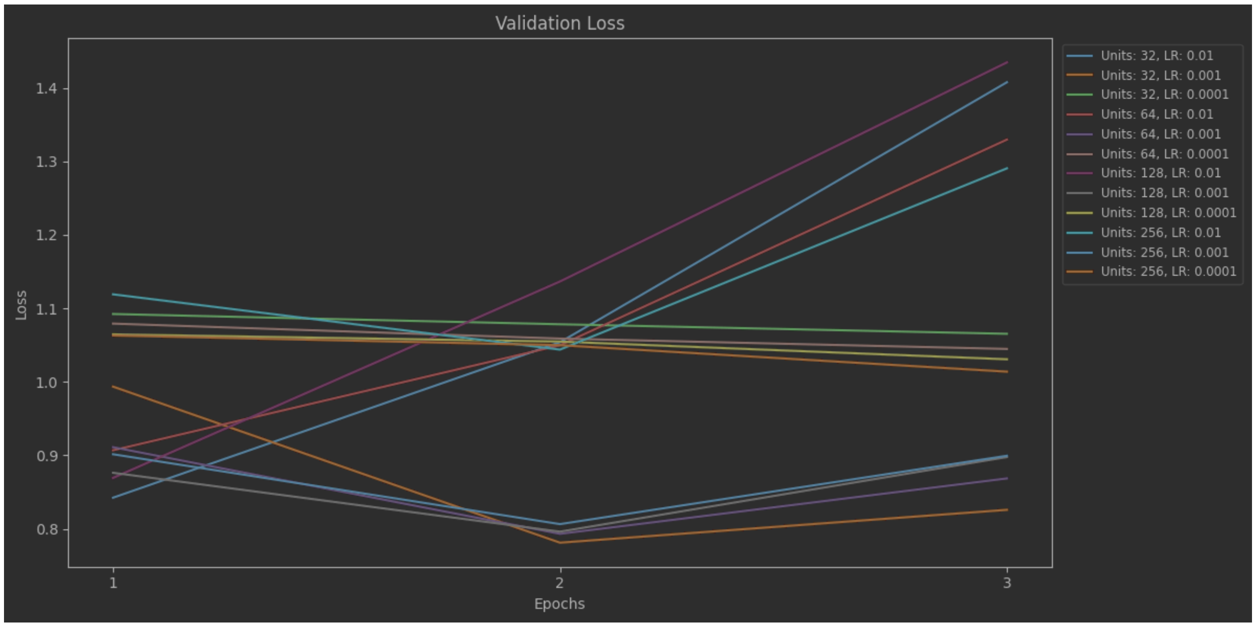
Adjusting the Number of LSTM Units

The number of LSTM units in a layer directly influences the model's capacity to capture information. A higher number of units allows the model to learn more complex patterns, but it also increases the risk of overfitting. We experimented with configurations using 32, 128, and 256 LSTM units to observe their effects on model performance.

Adjusting the Learning Rate

The learning rate controls the magnitude of weight updates during training. A lower learning rate might yield more precise convergence but can prolong the training process. We tested common learning rates of 0.01, 0.001, and 0.0001 to identify the most effective rate for balancing model accuracy and training time.





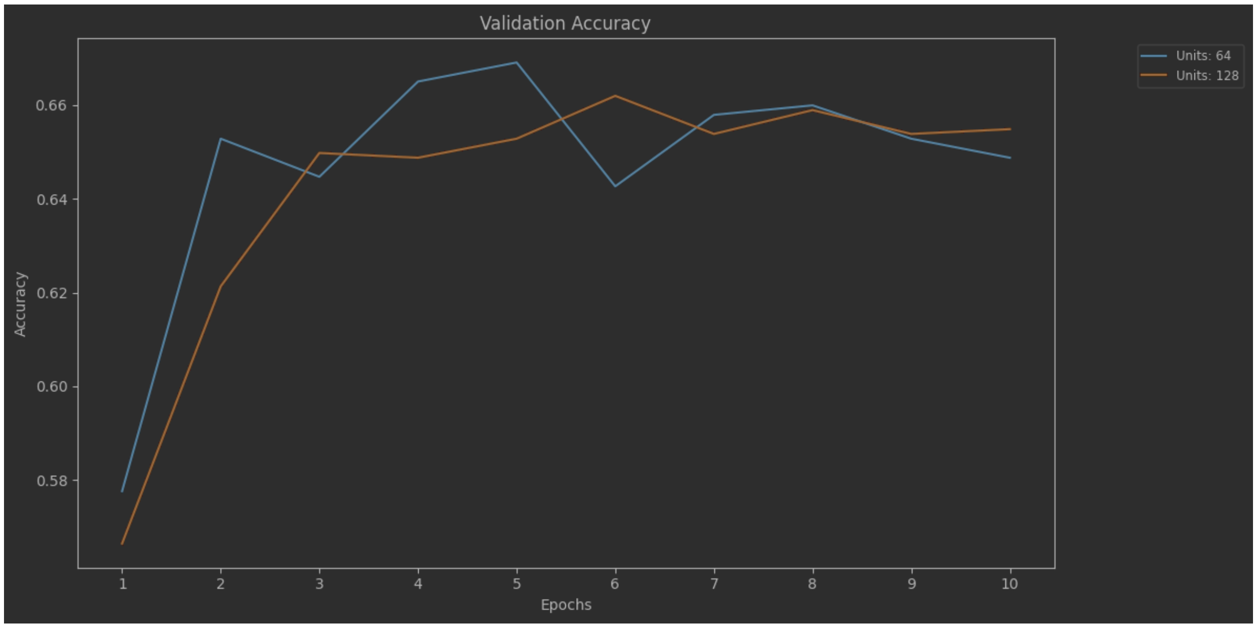
4.6.1 Result Interpretation

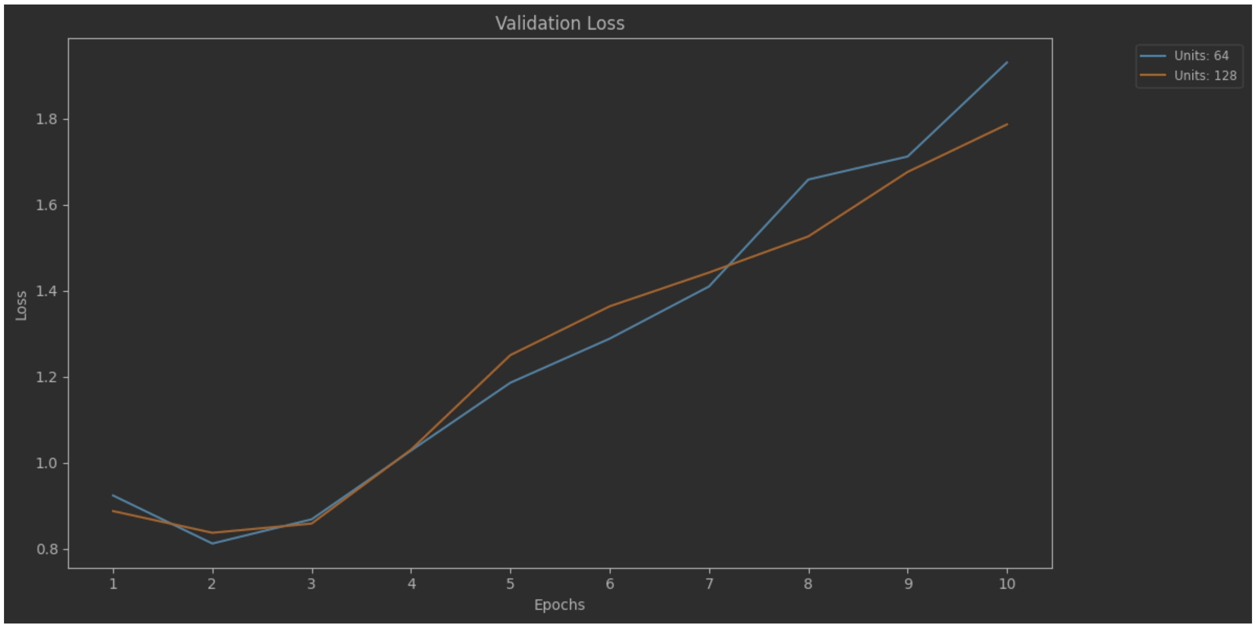
The impact of varying the number of LSTM units and learning rates on model performance is summarized below:

- **32 LSTM Units:** With a learning rate of 0.001, the model achieved a balance between training and validation accuracy without significant overfitting.
- **64 LSTM Units:** A learning rate of 0.001 again led to a good balance, with slightly higher validation accuracy than the model with 32 units.
- **128 LSTM Units:** The 0.001 learning rate proved effective, providing a good balance and relatively high validation accuracy.
- **256 LSTM Units:** While the model performed well with a 0.001 learning rate, increasing the LSTM units did not significantly enhance performance compared to configurations with fewer units.

Choosing the Best Configuration:
Based on these results, a configuration with 64 or 128 LSTM units and a learning rate of 0.001 emerged as the most effective. This configuration provided a good balance between learning capacity and generalization without significant overfitting.

4.7 Detailed Comparison of Model Configurations





We compared the performance of models with 64 and 128 LSTM units, both using a 0.001 learning rate and a dropout layer of 0.6. The results are as follows:

- **Model with 64 LSTM Units:**
  - Validation Accuracy: Peaked at 66.50% in the fourth epoch before gradually decreasing.
  - Validation Loss: Increased from 0.9241 in the first epoch to 1.9299 in the tenth epoch.

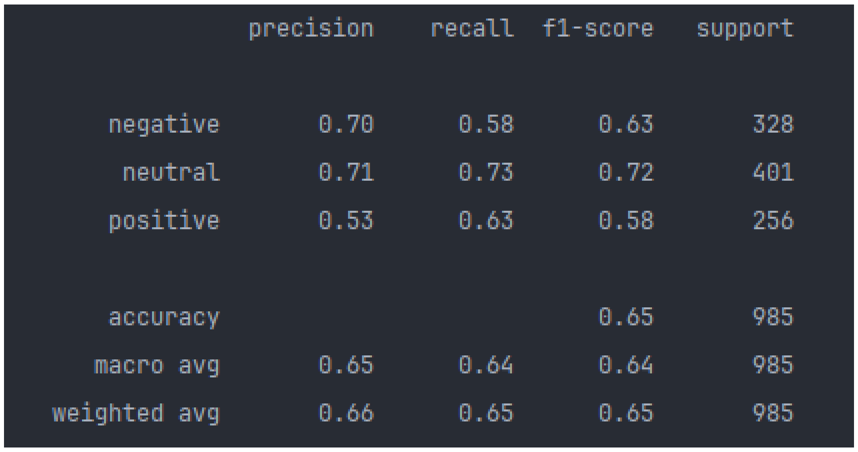
    - Observation: This model exhibited signs of overfitting as the epochs progressed, indicated by the increasing validation loss despite high training accuracy.

- **Model with 128 LSTM Units:**
    - Validation Accuracy: Peaked at 64.97% in the third epoch, with slight fluctuations before ending at 65.48%.
    - Validation Loss: Rose steadily from 0.8879 in the first epoch to 1.7860 in the tenth epoch.
    - Observation: Similar to the 64-unit model, this one also showed signs of overfitting, as evidenced by the increasing validation loss.

Conclusion

The model with 128 LSTM units appears slightly better, maintaining a higher validation accuracy in the later epochs compared to the 64-unit model. The optimal epoch for the 128-unit model seems to be around the third epoch, where it reaches its peak validation accuracy of 64.97% without a significant increase in validation loss.

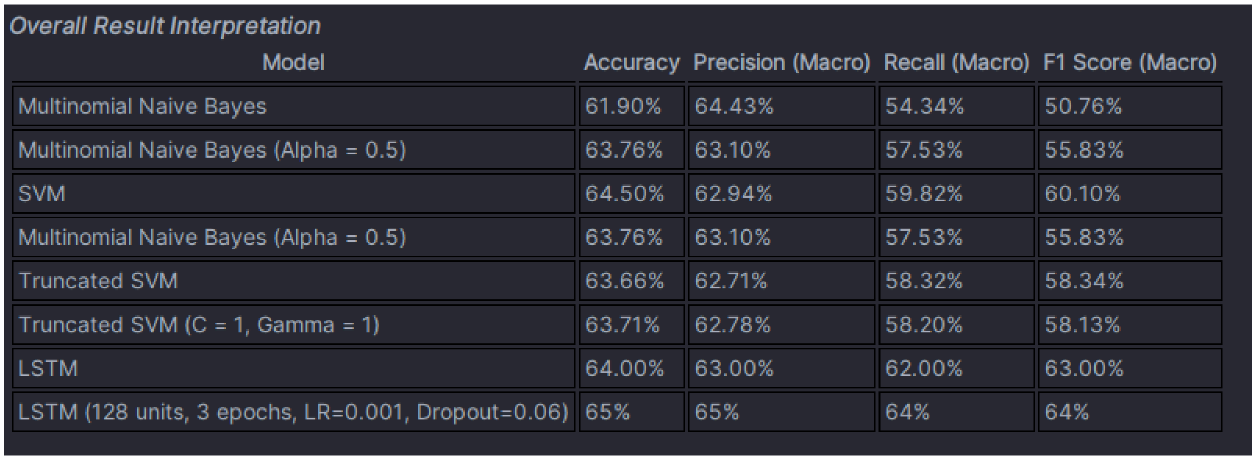
The final evaluation of the tuned models is summarised below:



```
              precision    recall  f1-score   support

    negative       0.70      0.58      0.63       328
     neutral       0.71      0.73      0.72       401
    positive       0.53      0.63      0.58       256

    accuracy                           0.65       985
   macro avg       0.65      0.64      0.64       985
weighted avg       0.66      0.65      0.65       985
```

4.8 Final Model Evaluation

The final evaluation of the tuned models is summarized below:



Overall Result Interpretation

Model	Accuracy	Precision (Macro)	Recall (Macro)	F1 Score (Macro)
Multinomial Naive Bayes	61.90%	64.43%	54.34%	50.76%
Multinomial Naive Bayes (Alpha = 0.5)	63.76%	63.10%	57.53%	55.83%
SVM	64.50%	62.94%	59.82%	60.10%
Multinomial Naive Bayes (Alpha = 0.5)	63.76%	63.10%	57.53%	55.83%
Truncated SVM	63.66%	62.71%	58.32%	58.34%
Truncated SVM (C = 1, Gamma = 1)	63.71%	62.78%	58.20%	58.13%
LSTM	64.00%	63.00%	62.00%	63.00%
LSTM (128 units, 3 epochs, LR=0.001, Dropout=0.06)	65%	65%	64%	64%

The final evaluation of the tuned models is summarized below:

- **Multinomial Naive Bayes:**
  Post-tuning, there was an improvement in accuracy, recall, and F1 score, indicating that tuning the alpha parameter helped the model make more balanced and accurate classifications. However, the slight drop in precision suggests a slight increase in false positives.
- **SVM:**
  The performance metrics remained unchanged post-tuning, implying that the default parameters were likely optimal for the dataset, or that the parameter space explored did not include better values. SVM's performance seems less sensitive to parameter changes within the tested range.
- **Truncated SVM:**
  Marginal changes were observed in performance metrics post-tuning, likely due to inherent randomness in the training process or minor parameter adjustments. The almost unchanged performance suggests that the truncated SVM might have reached its capacity on this dataset with the given features.
- **LSTM:**
  The LSTM model showed noticeable improvements in all metrics after tuning, highlighting its strength in capturing temporal dependencies in the data, which is particularly beneficial for text data. Tuning parameters like the number of LSTM units, learning rate, and dropout significantly impacted the model’s ability to learn without overfitting.

Best Model Configuration

Determining the best model depends on various factors, including task requirements, trade-offs between performance metrics, and practical deployment considerations. Based on the results:

- **Accuracy:** If overall accuracy is the primary concern, the LSTM and SVM models are top performers, with the LSTM slightly ahead at 65%.
- **Balance Between Precision and Recall:** The LSTM model provides a good balance, making it preferable when both false positives and false negatives are equally costly.
- **Complexity and Interpretability:** Traditional models like SVM and MNB are easier to interpret and quicker to train compared to LSTM. For scenarios requiring interpretability or limited computational resources, these models are viable choices.

Specific Use Cases

- For tasks where false negatives are more critical (e.g., detecting negative sentiments in customer feedback), the LSTM, with a recall of 64%, might be better suited.
- For tasks where precision is more critical (e.g., minimizing irrelevant recommendations in content recommendation systems), a model with higher precision may be preferred.

Conclusion

The LSTM model appears to be the best overall, considering its balanced performance in accuracy, precision, and recall. However, if interpretability and computational efficiency are more important, the SVM model is a strong contender, offering competitive performance with slightly lower complexity.

5.0 Discussion

5.1 Insights on Model Performance

Evaluating the machine learning models—LSTM, SVM, Truncated SVM, and Multinomial Naive Bayes—has provided valuable insights into their performance in sentiment analysis. The LSTM model, leveraging its ability to capture temporal patterns, demonstrated a strong capacity for distinguishing 'neutral' sentiments from others. However, the model's difficulty in accurately classifying 'positive' sentiments underscores a limitation that may stem from the inherent complexity of such sentiments or the need for more diverse training data.

Although SVM was slightly behind the LSTM in accuracy, traditional models, particularly SVM, offered competitive performance. SVM's balance between precision and recall highlights its reliability, especially in cases where computational efficiency and model interpretability are critical. The consistency of SVM's performance across different parameter settings suggests its robustness in diverse datasets.

5.2 The Challenge of Overfitting

The LSTM model's tendency to overfit the training data was a notable challenge. Despite achieving near-perfect accuracy on the training set, the model struggled to generalize, as indicated by the stagnant validation accuracy and rising validation loss. The application of dropout layers helped mitigate this issue to some extent. Still, the results indicate that more sophisticated approaches might be needed to prevent overfitting without leading to underfitting, as seen with excessive regularization.

5.3 Effectiveness of Hyperparameter Tuning

Hyperparameter tuning was critical in enhancing model performance, particularly for the LSTM model. Adjusting the number of LSTM units and the learning rate was key to balancing model complexity and generalization. The observation that increasing the number of LSTM units beyond a certain point did not yield substantial performance gains suggests that there is a threshold for model complexity, beyond which returns diminish.

The choice of learning rate also proved crucial, with a rate of 0.001 offering the best balance between convergence speed and model accuracy. These findings underscore the importance of careful hyperparameter tuning in optimizing deep learning models.

5.4 Comparative Strengths of Traditional Models

The study also revealed that traditional models like SVM and Multinomial Naive Bayes, despite their simpler architectures, remain highly relevant in specific contexts. SVM, in particular, demonstrated robustness and ease of tuning, making it an attractive option for scenarios where computational resources are limited or model interpretability is paramount. The stability of SVM's performance across different parameter settings highlights its reliability for practical applications.

5.5 Implications for Future Applications

These findings have significant implications for deploying sentiment analysis models in real-world applications. The LSTM model's strength in handling complex, sequential data makes it particularly suited for tasks requiring detailed sentiment analysis, such as social media monitoring or customer feedback evaluation. However, the efficiency and interpretability of traditional models like SVM suggest that they should not be overlooked, especially in resource-constrained environments or applications where explainability is crucial.

5.6 Future Research Directions

Future research should explore alternative regularization techniques and model architectures to address the overfitting observed in the LSTM model. Additionally, investigating ensemble methods could combine the strengths of both deep learning and traditional models, potentially leading to better overall performance. Expanding the scope of evaluation metrics beyond accuracy, precision, and recall to include measures such as AUC-ROC and MCC could also provide a more nuanced understanding of model performance across different tasks.

6.0 Conclusion

This study has comprehensively evaluated various machine learning models for sentiment analysis, with a particular focus on the LSTM model. The key takeaway is that while the LSTM model demonstrates strong potential in handling complex, sequential data, it requires careful tuning and regularization to prevent overfitting and ensure robust generalization.

6.1 Summary of Findings

- **LSTM Model:** The LSTM model outperformed traditional models in accuracy and capturing temporal dependencies. However, its struggle with positive sentiment classification and tendency to overfit indicates areas for improvement.
- **Traditional Models:** Despite being less complex, SVM and Multinomial Naive Bayes provide competitive performance and are valuable in scenarios requiring efficiency and interpretability. SVM, in particular, showed robustness across various parameter settings, making it a reliable choice for practical applications.

6.2 Practical Recommendations

For practitioners, the choice between LSTM and traditional models should be guided by the specific requirements of the task at hand. If handling complex, sequential data is paramount and computational resources are sufficient, the LSTM model is a strong candidate. However, traditional models like SVM should be considered for tasks where interpretability and efficiency are critical.

6.3 Future Work

Further research should refine the LSTM model by exploring advanced regularization techniques and alternative architectures that can better handle the nuances of positive sentiment classification. Additionally, combining ensemble methods could leverage the strengths of both deep learning and traditional approaches, potentially leading to superior performance.

6.4 Final Thoughts

In conclusion, this study underscores the importance of aligning model selection and tuning strategies with the specific demands of the application. While deep learning models like LSTM hold great promise, traditional models continue to offer valuable benefits, particularly in terms of efficiency and ease of use. By carefully considering these factors, practitioners can make informed decisions that optimize the performance of sentiment analysis systems in real-world settings.

7.0 References

Agüero-Torales, M. M., Abreu Salas, J. I., & López-Herrera, A. G. (2021). Deep learning and multilingual sentiment analysis on social media data: An overview. *Applied Soft Computing*, *107*, 107373. https://doi.org/10.1016/j.asoc.2021.107373

Al Amrani, Y., Lazaar, M., & El Kadiri, K. E. (2017). Random Forest and Support Vector Machine based Hybrid Approach to Sentiment Analysis. *Procedia Computer Science*, *127*, 511-520. https://doi.org/10.1016/j.procs.2018.01.150

Alqaryouti, O., Siyam, N., Monem, A. A., & Shaalan, K. (2020). Aspect-based sentiment analysis using smart government review data. *Applied Computing and Informatics*. https://doi.org/10.1016/j.aci.2019.11.003

Amin, M. Z., & Nadeem, N. (2018). Convolutional Neural Network: Text Classification Model for Open Domain Question Answering System. *ArXiv*. /abs/1809.02479

Anzai, Y. (1991). Pattern Feature Extraction. *Pattern Recognition & Machine Learning*, 89-140. https://doi.org/10.1016/B978-0-08-051363-8.50008-9

Argueta, C., & Chen, Y. S. (2014). Multi-Lingual Sentiment Analysis of Social Data Based on Emotion-Bearing Patterns. In S. Lin, L.-W. Ku, E. Cambria, & T.-T. Kuo (Eds.), *Proceedings of the Second Workshop on Natural Language Processing for Social Media (SocialNLP)* (pp. 38-43). Dublin, Ireland: Association for Computational Linguistics and Dublin City University. https://aclanthology.org/W14-5906

Baischer, L., Wess, M., & TaheriNejad, N. (2021). Learning on Hardware: A Tutorial on Neural Network Accelerators and Co-Processors. *ArXiv*. /abs/2104.09252

Balaji, P., Nagaraju, O., & Haritha, D. (2017). Levels of sentiment analysis and its challenges: A literature review. In *2017 International Conference on Big Data Analytics and Computational Intelligence (ICBDAC)* (pp. 436-439). Chirala, Andhra Pradesh, India. https://doi.org/10.1109/ICBDACI.2017.8070879.

Banerjee, S., Raja Chakravarthi, B., & McCrae, J. P. (2020). Comparison of Pretrained Embeddings to Identify Hate Speech in Indian Code-Mixed Text. In *2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)* (pp. 21-25). Greater Noida, India. https://doi.org/10.1109/ICACCCN51052.2020.9362731

Bangert, P. (2021). Chapter 3 - Machine Learning. In P. Bangert (Ed.), *Machine Learning and Data Science in the Oil and Gas Industry* (pp. 37-67). Gulf Professional Publishing. ISBN 9780128207147. https://doi.org/10.1016/B978-0-12-820714-7.00003-0.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2006). Greedy layer-wise training of deep networks. Advances in neural information processing systems, 19. https://doi.org/10.7551/mitpress/7503.003.0024

Bhatt, D., Patel, C., Talsania, H., Patel, J., Vaghela, R., Pandya, S., Modi, K., & Ghayvat, H. (2020). CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope. *Electronics*, *10*(20), 2470. https://doi.org/10.3390/electronics10202470.

Birjali, M., Kasri, M., & Beni-Hssane, A. (2021). A comprehensive survey on sentiment analysis: Approaches, challenges and trends. *Knowledge-Based Systems, 226*, 107134. ISSN 0950-7051. https://doi.org/10.1016/j.knosys.2021.107134.

Çetinoğlu, Ö., Schulz, S., & Vu, N. T. (2016). Challenges of Computational Processing of Code-Switching. In M. Diab, P. Fung, M. Ghoneim, J. Hirschberg, & T. Solorio (Eds.), *Proceedings of the Second Workshop on Computational Approaches to Code Switching* (pp. 1-11). Austin, Texas: Association for Computational Linguistics. https://doi.org/10.18653/v1/W16-5801

Chatzakou, D., Vakali, A., & Kafetsios, K. (2017). Detecting variation of emotions in online activities. *Expert Systems With Applications*, *89*, 318-332. https://doi.org/10.1016/j.eswa.2017.07.044

Cheng, F., & Zhao, J. (2018). A novel process monitoring approach based on Feature Points Distance Dynamic Autoencoder. *Computer Aided Chemical Engineering*, *46*, 757-762. https://doi.org/10.1016/B978-0-12-818634-3.50127-2

Choubey, V. (2020, July 8). Text classification using CNN. *Medium*. https://medium.com/voice-tech-podcast/text-classification-using-cnn-9ade8155dfb9

Dai, Y., & Radford, B. (2018). Multilingual word embedding for zero-shot text classification. In *Annual Meeting of the American Political Science Association, Boston*.https://aclanthology.org/N16-1083.pdf

Dang, C N., García, M N M., & Prieta, F D L. (2020, March 14). Sentiment Analysis Based on Deep Learning: A Comparative Study. Electronics, 9(3), 483. https://doi.org/10.3390/electronics9030483

De Leon, F. A., Guéniat, F., & Madabushi, H. T. (2020). CS-Embed at SemEval-2020 Task 9: The effectiveness of code-switched word embeddings for sentiment analysis. *ArXiv*. /abs/2006.04597

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., ... & Ng, A. (2012). Large scale distributed deep networks. Advances in neural information processing systems, 25. https://doi.org/10.5555/2999134.2999271

DiPietro, R., & Hager, G. D. (2019). Deep learning: RNNs and LSTM. *Handbook of Medical Image Computing and Computer Assisted Intervention*, 503-519. https://doi.org/10.1016/B978-0-12-816176-0.00026-0

DiPietro, R., & Hager, G. D. (2019). Deep learning: RNNs and LSTM. *Handbook of Medical Image Computing and Computer Assisted Intervention*, 503-519. https://doi.org/10.1016/B978-0-12-816176-0.00026-0

Farhood, H., Bakhshayeshi, I., Pooshideh, M., Rezvani, N., & Beheshti, A. (2021). Recent advances of image processing techniques in agriculture. *Artificial Intelligence and Data Science in Environmental Sensing*, 129-153. https://doi.org/10.1016/B978-0-323-90508-4.00007-1

Finkel, R., Kaufman, D., & Shamim, A. (2022). Analyzing Code-Mixing in Linguistic Corpora Using Kratylos. *Journal of Computer Culture & Heritage, 15*(1), Article 3, 1-15. https://doi.org/10.1145/3480238

Gautam, D., Gupta, K., & Shrivastava, M. (2021). Translate and Classify: Improving Sequence Level Classification for English-Hindi Code-Mixed Data. In T. Solorio, S. Chen, A. W. Black, M. Diab, S. Sitaram, V. Soto, E. Yilmaz, & A. Srinivasan (Eds.), *Proceedings of the Fifth Workshop on Computational Approaches to Linguistic Code-Switching* (pp. 15-25). Online: Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.calcs-1.3

Geoffrey E. Hinton, Simon Osindero, Yee-Whye Teh; A Fast Learning Algorithm for Deep Belief Nets. Neural Comput 2006; 18 (7): 1527–1554. https://doi.org/10.1162/neco.2006.18.7.1527

Goutham, R. (2023, January 23). Multilingual (non-English) NLP — 7 things to know before getting started. *Toward Data Science*. https://ramsrigoutham.medium.com/7-things-to-know-before-getting-started-in-multilingual-non-english-nlp-eb42d097e2b0

Gupta, D., Ekbal, A., & Bhattacharyya, P. (2020). A Semi-supervised Approach to Generate the Code-Mixed Text using Pre-trained Encoder and Transfer Learning. In T. Cohn, Y. He, & Y. Liu (Eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020* (pp. 2267-2280). Online: Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.findings-emnlp.206

Hajmohammadi, M. S., Ibrahim, R., & Selamat, A. (2014). Cross-lingual sentiment classification using multiple source languages in multi-view semi-supervised learning. *Engineering*

Applications of Artificial Intelligence, *36*, 195-203. https://doi.org/10.1016/j.engappai.2014.07.020

Hamed, A. R., Qiu, R., & Li, D. (2016). The importance of neutral class in sentiment analysis of Arabic tweets. *Int. J. Comput. Sci. Inform. Technol*, 8, 17-31. https://www.researchgate.net/profile/Hamed-Al-Rubaiee/publication/302979046_The_Importance_of_Neutral_Class_in_Sentiment_Analysis_of_Arabic_Tweets/links/57d9b22608ae601b39b15fdb/The-Importance-of-Neutral-Class-in-Sentiment-Analysis-of-Arabic-Tweets.pdf

Hassani Niaki, M., Ghorbanzadeh Ahangari, M., & Pashaian, M. (2022). A material-independent deep learning model to predict the tensile strength of polymer concrete. *Composites Communications*, *36*, 101400. https://doi.org/10.1016/j.coco.2022.101400

Hauthal, E., Burghardt, D., Fish, C., & Griffin, A. L. (2020). Sentiment Analysis. In A. Kobayashi (Ed.), *International Encyclopedia of Human Geography (Second Edition)* (pp. 169-177). Elsevier. ISBN 9780081022962. https://doi.org/10.1016/B978-0-08-102295-5.10593-1.

Henrickson, K., Rodrigues, F., Pereira, F. C. (2019). *Chapter 5 - Data Preparation.* In C. Antoniou, L. Dimitriou, & F. Pereira (Eds.), Mobility Patterns, Big Data and Transport Analytics. Elsevier. pp. 73-106. ISBN 9780128129708. https://doi.org/10.1016/B978-0-12-812970-8.00005-1

Ho, V. A., Nguyen, D. H. C., Nguyen, D. H., Pham, L. T. V., Nguyen, D. V., Nguyen, K. V., & Nguyen, N. L. T. (2020). *Emotion Recognition for Vietnamese Social Media Text*. arXiv preprint arXiv:1911.09339.

https://www.academia.edu/download/53953860/paper5.pdf

Huang, Y., Sun, S., Duan, X., & Chen, Z. (2016). A study on Deep Neural Networks framework. In *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)* (pp. 1519-1522). Xi'an, China. https://doi.org/10.1109/IMCEC.2016.7867471

Jamatia, A., Gambäck, B., & Das, A. (2016, April). Collecting and annotating indian social media code-mixed corpora. In *International Conference on Intelligent Text Processing and Computational Linguistics* (pp. 406-417). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-75487-1_32

Joshi, P. (2021, February 19). Handling Data Scarcity while building Machine Learning applications. *Towards Data Science*. https://towardsdatascience.com/handling-data-scarcity-while-building-machine-learning-applications-e6c243b284b0

Kostadinov, S. (2017, December 2). How Recurrent Neural Networks work. *Medium*. https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7

Koval, A., Sharif Mansouri, S., & Kanellakis, C. (2022). Machine learning for ARWs. *Aerial Robotic Workers*, 159-174. https://doi.org/10.1016/B978-0-12-814909-6.00016-0

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25. https://doi.org/10.1145/3065386

LeCun, Y., Bengio, Y., & Hinton, G. (2015, May 1). Deep learning. Nature, 521(7553), 436-444. https://doi.org/10.1038/nature14539

Li, S., Tian, Z., & Li, Y. (2023). Residual long short-term memory network with multi-source and multi-frequency information fusion: An application to China's stock market. *Information Sciences*, *622*, 133-147. https://doi.org/10.1016/j.ins.2022.11.136

Liang, H., Sun, X., Sun, Y., & Gao, Y. (2017). Text feature extraction based on deep learning: A review. *EURASIP Journal on Wireless Communications and Networking*, *2017*(1), 1-12. https://doi.org/10.1186/s13638-017-0993-1

Lindemann, B., Müller, T., Vietz, H., Jazdi, N., & Weyrich, M. (2020). A survey on long short-term memory networks for time series prediction. *Procedia CIRP*, *99*, 650-655. https://doi.org/10.1016/j.procir.2021.03.088

Liu, B. (2012). Sentiment Analysis: A Fascinating Problem. In: Sentiment Analysis and Opinion Mining. Synthesis Lectures on Human Language Technologies. Springer, Cham. https://doi.org/10.1007/978-3-031-02145-9_1

Mamalakis, A., & Barnes, E. A. (2022). Investigating the fidelity of explainable artificial intelligence methods for applications of convolutional neural networks in geoscience. *ArXiv*. https://doi.org/10.1175/AIES-D-22-0012.1

Medrouk, L., & Pappa, A. (2017). Deep Learning Model for Sentiment Analysis in Multi-lingual Corpus. *Lecture Notes in Computer Science*, 205–212. https://doi.org/10.1007/978-3-319-70087-8_22

Mishra, V., Agarwal, S. M., & Puri, N. (2018). Comprehensive and comparative analysis of neural network. *International Journal of Computer Application, 2*(8), 126-137. https://dx.doi.org/10.26808/rs.ca.i8v2.15

Moustakas, T., & Kolomvatsos, K. (2023). Homogeneous transfer learning for supporting pervasive edge applications. *Evolving Systems*. https://doi.org/10.1007/s12530-023-09548-3

Munir, A., Shabib, A., Iftikhar, A., & Noureen, H. (2017) Hybrid Tools and Techniques for Sentiment Analysis: A Review.

Nandwani, P., & Verma, R. (2021). A review on sentiment analysis and emotion detection from text. *Social network analysis and mining, 11*(1), 81. https://doi.org/10.1007/s13278-021-00776-6

Oğuz, A., & Ertuğrul, Ö. F. (2022). Introduction to deep learning and diagnosis in medicine. *Diagnostic Biomedical Signal and Image Processing Applications With Deep Learning Methods*, 1-40. https://doi.org/10.1016/B978-0-323-96129-5.00003-2

Onyenwe, I., Nwagbo, S., Mbeledogu, N. et al. The impact of political party/candidate on the election results from a sentiment analysis perspective using #AnambraDecides2017 tweets. *Soc. Netw. Anal. Min.* 10, 55 (2020). https://doi.org/10.1007/s13278-020-00667-2

Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up? Sentiment Classification using Machine Learning Techniques. *In Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)* (pp. 79–86). Association for Computational Linguistics. https://aclanthology.org/W02-1011.pdf

Prabowo, R., & Thelwall, M. (2009). Sentiment analysis: A combined approach. *Journal of Informetrics, 3*(2), 143-157. https://doi.org/10.1016/j.joi.2009.01.003

Rani, S., & Kumar, P. (2018). Deep learning based sentiment analysis using Convolution Neural Network. *Arabian Journal for Science and Engineering*, *44*(4), 3305–3314. https://doi.org/10.1007/s13369-018-3500-z

Rasooli, M. S., Farra, N., Radeva, A., Yu, T., & McKeown, K. (2017). *Cross-lingual sentiment transfer with limited resources. Machine Translation, 32(1-2), 143–165.* https://doi.org/10.1007/s10590-017-9202-6

Ray, P., & Chakrabarti, A. (2022). A mixed approach of deep learning method and rule-based method to improve aspect level sentiment analysis. *Applied Computing and Informatics, 18*(1/2), 163-178. https://doi.org/10.1016/j.aci.2019.02.002

Salinaamran. (2022, December 3). The depiction of 'Bahasa Rojak'. *Medium*. https://medium.com/@salinaamran/the-depiction-of-bahasa-rojak-3ae30816608e

Saraceni, M. (2010). *The relocation of English: shifting paradigms in a global era*. (Language and Globalization). Palgrave Macmillan. http://www.palgrave.com/products/title.aspx?PID=294280

Schmidhuber, J. (2014). Deep learning in neural networks: An overview. Neural Networks, 61, 85-117. https://doi.org/10.1016/j.neunet.2014.09.003\

Sengupta, A., Suresh, T., Akhtar, M. S., & Chakraborty, T. (2022). A Comprehensive Understanding of Code-mixed Language Semantics using Hierarchical Transformer. *ArXiv*. /abs/2204.12753

Shah, F. P., & Patel, V. (2016). A Review of Feature Selection and Feature Extraction for Text Classification. In *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)* (pp. 2264-2268). Chennai, India. https://doi.org/10.1109/WiSPNET.2016.7566545

Shanahan, J. G., Grefenstette, G., Qu, Y., & Evans, D. A. (2004). Mining multilingual opinions through classification and translation. In *Proceedings of AAAI Spring Symposium on Exploring Attitude and Affect in Text*. https://cdn.aaai.org/Symposia/Spring/2004/SS-04-07/SS04-07-027.pdf

Shanmugavadivel, K., Sathishkumar, V. E., Raja, S., Lingaiah, T. B., Neelakandan, S., & Subramanian, M. (2022). Deep learning based sentiment analysis and offensive language identification on multilingual code-mixed data. *Scientific Reports*, *12*(1), 1-12. https://doi.org/10.1038/s41598-022-26092-3

Shobha, G., & Rangaswamy, S. (2018). Chapter 8 - Machine Learning. In V. N. Gudivada & C. R. Rao (Eds.), *Handbook of Statistics, Volume 38* (pp. 197-228). Elsevier. ISSN 0169-7161, ISBN 9780444640420. https://doi.org/10.1016/bs.host.2018.07.004.

Singh, G. (2021). *Sentiment analysis of code-mixed social media text (hinglish)*. Ithaca: Cornell University Library, arXiv.org. Retrieved from https://www.proquest.com/working-papers/sentiment-analysis-code-mixed-social-media-text/docview/2493268856/se-2

Srivastava, V., & Singh, M. (2020). IIT Gandhinagar at SemEval-2020 Task 9: Code-Mixed Sentiment Classification Using Candidate Sentence Generation and Selection. *ArXiv*. /abs/2006.14465

Srivastava, V., & Singh, M. (2021). Challenges and considerations with code-mixed NLP for multilingual societies. *ArXiv,abs/2106.07823.*

Su, J., Chen, Q., Wang, Y., Zhang, L., Pan, Wei., & Li, Z. (2023). Sentence-level sentiment analysis based on supervised gradual machine learning. *Scientific Reports*, 13, 14500. https://doi.org/10.1038/s41598-023-41485-8

Sze, V., Chen, Y.-H., Yang, T.-J., & Emer, J. S. (2017). Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proceedings of the IEEE, 105*(12), 2295-2329. https://doi.org/10.1109/JPROC.2017.2761740

Teuwen, J., & Moriakov, N. (2020). Convolutional neural networks. *Handbook of Medical Image Computing and Computer Assisted Intervention*, 481–501. https://doi.org/10.1016/b978-0-12-816176-0.00025-9

Tonkin, E. L. (2016). *Chapter 2 - A Day at Work (with Text): A Brief Introduction*. In E. L. Tonkin & G. J. L. Tourte (Eds.), Working with Text (Chandos Information Professional Series). Chandos Publishing. pp. 23-60. ISBN 9781843347491. https://doi.org/10.1016/B978-1-84334-749-1.00002-0

Toohey, J. R., Raunak, M. S., & Binkley, D. (2021). From Neuron Coverage to Steering Angle: Testing Autonomous Vehicles Effectively. *Computer, 54*(8), 77-85. https://doi.org/10.1109/MC.2021.3079921

Tsantekidis, A., Passalis, N., & Tefas, A. (2021). Recurrent neural networks. *Deep Learning for Robot Perception and Cognition*, 101-115. https://doi.org/10.1016/B978-0-32-385787-1.00010-5

Unat, D., & Aktaş, M S. (2018, August 24). Special issue on High performance computing conference (BASARIM‐2017). https://scite.ai/reports/10.1002/cpe.4926

Vaidyanathan, S., Sivakumar, M., & Kaliamourthy, B. (2021). Challenges of Developing AI Applications in the Evolving Digital World and Recommendations to Mitigate Such Challenges: A Conceptual View. In *Confluence of AI, Machine, and Deep Learning in Cyber Forensics* (pp. 177-198). IGI Global.https://doi.org/10.4018/978-1-7998-4900-1.ch011

Waris, A. M. (2012). Code-switching and mixing (Communication in Learning Language). *Jurnal Dakwah Tabligh*, *13*(1), pp. 123-135. https://media.neliti.com/media/publications/77106-EN-code-switching-and-mixing-communication.pdf

Wehrmann, J., Becker, W., Cagnini, H.E.L., & Barros, R.C. (2017). A Character-based Convolutional Neural Network for Language-agnostic Twitter Sentiment Analysis. In *2017 International Joint Conference on Neural Networks (IJCNN)* (pp. 2384–2391). https://doi.org/10.1109/IJCNN.2017.7966145

Wu, W., Liu, T., & Haick, H. (2022). Electronic Nose Sensors for Healthcare. *Encyclopedia of Sensors and Biosensors*, 728-741. https://doi.org/10.1016/B978-0-12-822548-6.00097-2

Xu, Y., Cao, H., Du, W., et al. (2022). A Survey of Cross-lingual Sentiment Analysis: Methodologies, Models, and Evaluations. *Data Science and Engineering, 7*, 279–299. https://doi.org/10.1007/s41019-022-00187-3

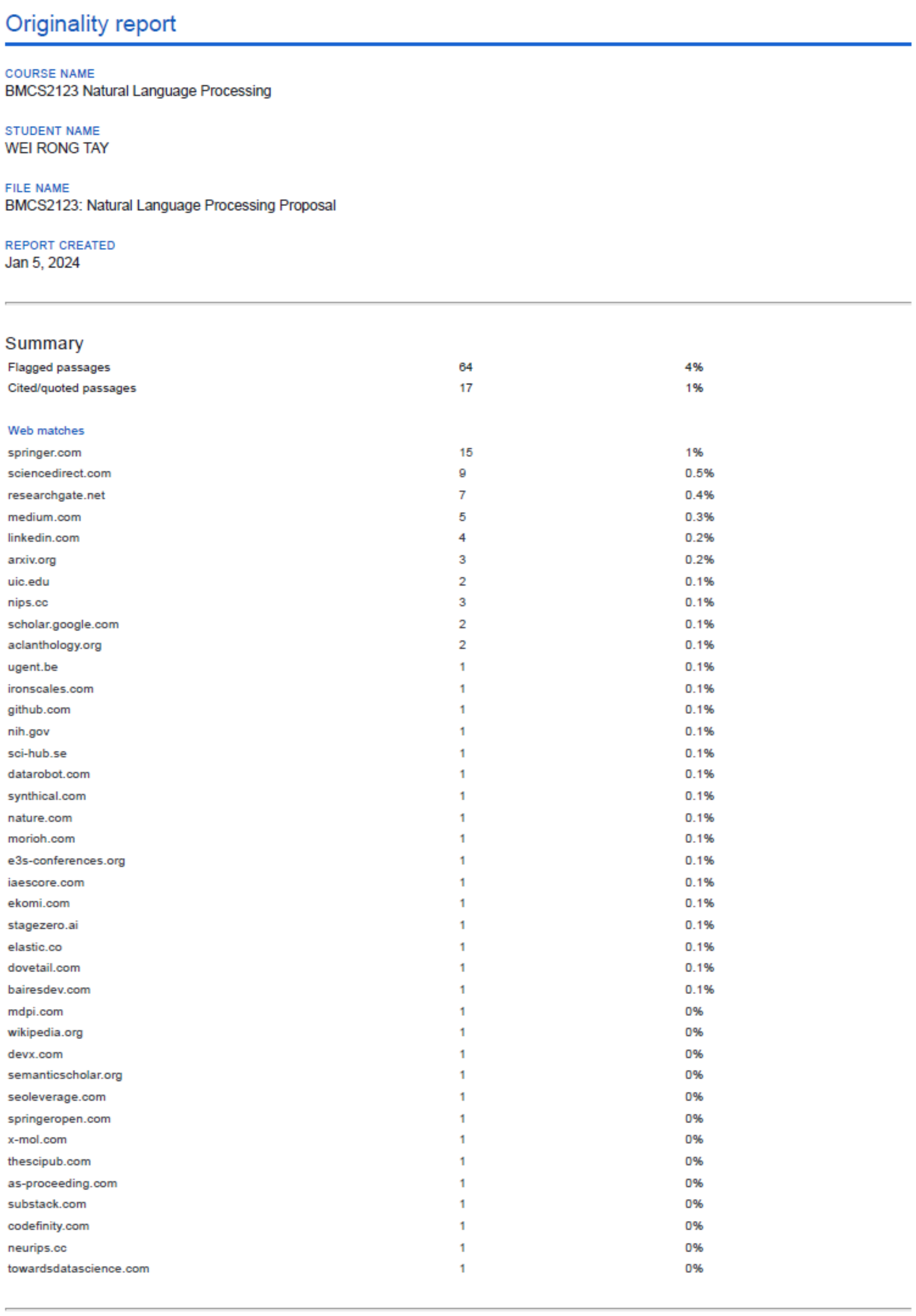
Yadav, A. (2023, November 28). Natural Language Processing through Transfer Learning: A Case study on Sentiment analysis. *arXiv.org.* https://arxiv.org/abs/2311.16965

Yadav, A., & Vishwakarma, D. K. (2019b). Sentiment analysis using deep learning architectures: a review. *Artificial Intelligence Review*, *53*(6), 4335–4385. https://doi.org/10.1007/s10462-019-09794-5

Zhang, J., Zhu, Y., Zhang, X., Ye, M., & Yang, J. (2018). Developing a Long Short-Term Memory (LSTM) based model for predicting water table depth in agricultural areas. *Journal of Hydrology*, *561*, 918-929. https://doi.org/10.1016/j.jhydrol.2018.04.065

8.0 Appendix

8.1 Originality Report



Originality report

COURSE NAME
BMCS2123 Natural Language Processing

STUDENT NAME
WEI RONG TAY

FILE NAME
BMCS2123: Natural Language Processing Proposal

REPORT CREATED
Jan 5, 2024

Summary

Flagged passages	64	4%
Cited/quoted passages	17	1%

Web matches		
springer.com	15	1%
sciencedirect.com	9	0.5%
researchgate.net	7	0.4%
medium.com	5	0.3%
linkedin.com	4	0.2%
arxiv.org	3	0.2%
uic.edu	2	0.1%
nips.cc	3	0.1%
scholar.google.com	2	0.1%
aclanthology.org	2	0.1%
ugent.be	1	0.1%
ironscales.com	1	0.1%
github.com	1	0.1%
nih.gov	1	0.1%
sci-hub.se	1	0.1%
datarobot.com	1	0.1%
synthical.com	1	0.1%
nature.com	1	0.1%
morioh.com	1	0.1%
e3s-conferences.org	1	0.1%
iaescore.com	1	0.1%
ekomi.com	1	0.1%
stagezero.ai	1	0.1%
elastic.co	1	0.1%
dovetail.com	1	0.1%
bairesdev.com	1	0.1%
mdpi.com	1	0%
wikipedia.org	1	0%
devx.com	1	0%
semanticscholar.org	1	0%
seoleverage.com	1	0%
springeropen.com	1	0%
x-mol.com	1	0%
thescipub.com	1	0%
as-proceeding.com	1	0%
substack.com	1	0%
codefinity.com	1	0%
neurips.cc	1	0%
towardsdatascience.com	1	0%