Universidad Michoacana de San Nicolás de Hidalgo

División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica

ENSEMBLES OF PRE-TRAINED MODELS APPLIED TO TWO TEXT CLASSIFICATION COMPETITIONS

TESIS

presenta

Jaime Cerda Flores

Que para obtener el grado de

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

José Ortiz Bejar

Director de Tesis

Morelia, Michoacán, Febrero 2025

ENSEMBLES OF PRE-TRAINED MODELS APPLIED TO TWO TEXT CLASSIFICATION COMPETITIONS

Los Miembros del Jurado de Examen de Grado aprueban la **Tesis de Maestría en Ciencias en Ingeniería Eléctrica** de **Jaime Cerda Flores.**

Dr. Alejandro Zamora Méndez
Presidente del Jurado

Dr. José Ortíz Béjar
Director de Tesis

Dr. Félix Calderón Solorio
Vocal

Dr. José Antonio Camarena Ibarrola
Vocal

Dr. Mario Graff Guerrero
Revisor Externo (INFOTEC)

Dr. J. Aurelio Medina Rios
Jefe de la División de Estudios de Posgrado de la Facultad de Ingeniería Eléctrica. UMSNH
(Por reconocimiento de firmas)

UNIVERSIDAD MICHOACANA DE SAN NICOLÁS DE HIDALGO
Enero 2025

Acknowledgments

I would like to express my gratitude to my thesis advisor, José Ortiz Bejar, for his guidance, patience, and encouragement provided throughout this programme. His insightful feedback and support has been vital to the shaping of this research, and I am truly grateful for his mentorship.

I am also thankful to my thesis committee members for their time, expertise, and constructive observations. Their suggestions contributed to the improvement of this work, and I appreciate their dedication to my academic growth.

I'd also like to acknowledge the support provided by CONAHCYT through Project CF-2023-I-1174 within the framework of the "Ciencia de Frontera 2023".

I give my heartfelt appreciation to my parents, Jaime Cerda Jacobo and Sandra Cecilia Flores Valencia. This achievement would not have been possible without their encouragement, presence and reassurance. My siblings, Sandra Cecilia Cerda Flores and Pablo Cerda Flores, for the laughs along the way. And my cousin, Ángel Eduardo Munguía Cerda, my initial inspiration and motivator to pursue studies in this field, and whose memory lies fondly in my heart.

Finally, I want to thank my partner, Alejandra Damayanti Aguilar Espinosa, with whom I have shared all the ups and downs of this journey. Her unwavering love and endless patience have been a constant source of strength and motivation through every challenge, and I am deeply grateful for her companionship.

To my loving family and Damayanti, with hope for many great moments ahead.

List of Publications

From this research, two conference papers were published. The list of publications is presented below.

1. **Cerda-Flores, J.** , Hernández-Mazariegos, R., Ortiz-Bejar, J., Calderón-Solorio, F., & Ortiz-Bejar, J. (2024). *UMSNH at RestMex 2023: An XGBoost Stacking with Pre-Trained Word-Embeddings over Data Batches.* In IberLEF@ SEPLN.

2. **Cerda-Flores, J.**, Juárez-Jiménez, M.G, Pineda-Castro, D., Hernández-Mazariegos, R., Graff, M., Cerda, J. & Ortiz-Bejar, J. (2023). *Text Classification Based on Ensembles of Pre-Trained Models for Sarcasm Identification in Dravidian Languages.* In Forum for Information Retrieval Evaluation, December 12–15, 2024, India.

Resumen

En este trabajo, se propone un sistema de clasificación basado en ensambles para mejorar el rendimiento de clasificación de texto en dos casos de estudio competitivos: la tarea de análisis de sentimiento de Rest-Mex 2023, centrado en reseñas turísticas en español, y la de identificación de sarcasmo de DravidianCodeMix@FIRE-2024, enfocada en datos de redes sociales con código mixto en lenguas dravídicas. La clasificación de texto es una tarea fundamental en el procesamiento del lenguaje natural, y es de gran relevancia en áreas como el turismo, donde comprender las opiniones de turistas es vital para el desarrollo económico, así como en la detección de sarcasmo, donde significados literales difieren de los reales.

El sistema de ensambles propuesto integra una variedad de modelos, incluyendo modelos basados en BERT, fastText y μTC. Las predicciones finales son obtenidas apilando las salidas de estos métodos y entrenando un clasificador XGBoost con las representaciones vectoriales resultantes. Debido a limitaciones computacionales iniciales, los primeros experimentos con el conjunto de datos de Rest-Mex se llevaron a cabo utilizando subconjuntos del conjunto de entrenamiento, logrando resultados aceptables aunque mejorables. Tras la terminación de la competencia y con la disponibilidad de un hardware más potente, la metodología se aplicó al conjunto de datos completo, lo que confirmó que el sistema de ensamblaje supera de manera consistente a los modelos individuales.

En la tarea de detección de sarcasmo de DravidianCodeMix@FIRE-2024, el enfoque basado en ensambles logró resultados superiores comparados con los modelos individuales, demostrando su adaptabilidad y robustez a través de diferentes idiomas y dominios. En la subtarea en lengua Tamil, el enfoque de ensamble obtuvo el primer lugar entre todos los participantes, mientras que un enfoque de fastText optimizado, que conforma parte del marco general del ensamble, empató en el primer puesto en la subtarea en lengua Malayalam.

Estos hallazgos resaltan la adaptabilidad del enfoque basado en ensambles en diversos conjuntos de datos e idiomas. Las mejoras de rendimiento obtenidas por los conjuntos en comparación con los modelos individuales también atribuyen a esto. Así, este trabajo contribuye al amplio campo del procesamiento del lenguaje natural, ofreciendo un método escalable para mejorar tanto el análisis de sentimientos en el sector turístico como la detección de sarcasmo en contextos multilingües con código mixto.

Palabras clave: Categorización de Texto, Ensambles, BERT, μTC, fastText, XGBoost

Abstract

In this work, an ensemble-based classification system is designed to improve text classification performance in two competitive case studies: the sentiment analysis challenge from the Rest-Mex 2023 competition, which focuses on Spanish language tourist reviews, and the sarcasm identification task from the DravidianCodeMix@FIRE-2024 competition, centred on code-mixed social media data in Dravidian languages. Text classification has long been a fundamental task in natural language processing, and is of particular significance in areas such as tourism, where understanding tourist feedback is vital for economic development, and in sarcasm detection, where literal meanings differ from intended sentiments.

The proposed ensemble system integrates a diverse array of models, including BERT-based models, fastText, and μTC. It achieves final predictions by stacking the outputs of these methods and training an XGBoost classifier on the resulting vector representations. Due to initial computational limitations, early experiments on the Rest-Mex dataset were conducted with subsets of the training data, with acceptable yet improvable results. Following the competition phase and the availability of more powerful hardware, the approach was applied to the entire dataset, confirming that the ensemble system consistently outperforms individual models.

For the DravidianCodeMix@FIRE-2024 sarcasm detection task, the ensemble approach achieved superior results compared to individual models, demonstrating its adaptability and robustness across languages and domains. In the Tamil-language subtask, the ensemble approach obtained the top rank among all participants, while an optimized fastText approach, as part of the overall ensemble framework, tied for first place in the Malayalam-language subtask.

These findings highlight the ensemble's adaptability across diverse datasets and languages. The performance improvements obtained by the ensembles compared to the individual models also attribute to this. Thereby, this work contributes to the broader field of natural language processing, offering a scalable method for improving both sentiment analysis in the tourism sector and sarcasm detection in multilingual, code-mixed contexts.
Keywords: Text Categorization, Stacking, BERT, μTC, fastText, XGBoost

Contents

List of Figures

List of Tables

List of Symbols

A	fastText embedding input matrix.
At	Transformer attention matrix.
B	Hidden layer weight matrix in fastText's architecture.
C	Number of classes in a classification task.
C_P	Set of polarity labels for the Rest-Mex competition, $[1, 2, 3, 4, 5]$.
$\mathcal{C}_\mu$	Configuration space defined by μTC.
D	Corpus of text documents.
$D_{transformer}$	Amount of decoder layers in the transformer architecture.
$DS_{FIRE\_TestMal}$	Test partition in Malayalam from the FIRE competition.
$DS_{FIRE\_TestTam}$	Test partition in Tamil from the FIRE competition.
$DS_{FIRE\_TrainMal}$	Training partition in Malayalam from the FIRE competition.
$DS_{FIRE\_TrainTam}$	Training partition in Tamil from the FIRE competition.
DS_{FIRE_ValMal}	Validation partition in Malayalam from the FIRE competition.
DS_{FIRE_ValTam}	Validation partition in Tamil from the FIRE competition.
DS_{RM}	Complete dataset from the Rest-Mex 2023 competition.
DS_{RM_Test}	Test partition from the Rest-Mex competition.
DS_{RM_Train}	Training partition from the Rest-Mex competition.
DS_{RM_TrainT}	Training split from DS_{RM_Train}.
DS_{RM_TrainV}	Validation split from DS_{RM_Train}.
d	Document within corpus D.
d_k	Length of rows in W^K.
d_q	Length of rows in W^Q.
d_v	Length of rows in W^V.
$E_{transformer}$	Amount of encoder layers in the transformer architecture.
F1-score	Classification metric.
$F_A(k)$	Macro F1-score for the "Attractive" class obtained by the system k.
$F_{Col}(k)$	Macro F1-score for the "Colombia" class obtained by the system k.
$F_{Cub}(k)$	Macro F1-score for the "Cuba" class obtained by the system k.
$F_H(k)$	Macro F1-score for the "Hotel" class obtained by the system k.
$F_{Mex}(k)$	Macro F1-score for the "Mexico" class obtained by the system k.
$F_{P_i}(k)$	F1-score for the i-th class in C_P given by the system k.
$F_R(k)$	Macro F1-score for the "Restaurant" class obtained by the system k.
$f_{softmax}$	Softmax function.

h	Hidden layer in fastText's architecture.
$h_{attention}$	Number of attention heads.
k	Model or method being evaluated.
K	Key matrix in the transformer architecture.
LM	Set of language models to be used for experimentation, [μTC, fastText, BETO, Multilingual BERT, MuRIL Large, l3cube-pune/malayalam-bert, l3cube-pune/tamil-bert, XLM-RoBERTa Base].
N	Total number of documents in a corpus.
n	Embedding size for tokens or documents.
n_e	Set of amount of subsets to be used for experimentation, $[2, 4, 8, 16, 20, 32]$.
$\mathcal{N}$	Neighbourhood relationship between configurations in $\mathcal{C}_\mu$.
o	Output layer in fastText's architecture.
O	Final document embedding matrix from the attention mechanism.
Q	Query matrix in the transformer architecture.
$Res_A(k)$	Resulting type score of the model k, used in the Rest-Mex competition.
$Res_C(k)$	Resulting country score of the model k, used in the Rest-Mex competition.
$Res_P(k)$	Resulting polarity score of the model k, used in the Rest-Mex competition.
s	Example sentence or text sequence.
$Sentiment(k)$	Resulting sentiment score of the model k, used in the Rest-Mex competition.
t	Token in a vocabulary.
$tf\text{-}idf(t, d, D)$	TF-IDF weight for token t in document d and corpus D.
T	Amount of tokens in a sentence s.
$T_{C_{P_i}}$	Total number of instances of the i-th label in C_P found in T_{C_R}.
T_{C_R}	Total number of samples in any Rest-Mex dataset.
V	Vocabulary derived from tokenizing example sentences.
V_l	Length of a vocabulary.
V	Value matrix in the transformer architecture.
W^K	Weight matrix to build K.
W^O	Output weight matrix.
W^Q	Weight matrix to build Q.
W^V	Weight matrix to build V.
x	Vector representation of a sentence or text sequence.

List of Acronyms

BERT	*Bidirectional Encoder Representations from Transformers* - Representaciones de Codificador Bidireccional de Transformadores
BoW	*Bag-of-Words* - Bolsa de Palabras
CBoW	*Continuous Bag of Words* - Bolsa de Palabras Continua
CPU	*Central Processing Unit* - Unidad Central de Procesamiento
DT	*Decision Tree* - Árbol de Decisión
FIRE	*Forum for Information Retrieval Evaluation* - Foro de Evaluación de Recuperación de Información
FN	*False Negative* - Falso Negativo
FP	*False Positive* - Falso Positivo
GPT-2	*Generative Pre-trained Transformer 2* - Transformador Generativo Preentrenado 2
GPU	*Graphics Processing Unit* - Unidad de Procesamiento Gráfico
K-NN	*K-Nearest Neighbors* - K Vecinos Más Cercanos
LR	*Logistic Regression* - Regresión Logística
LSTM	*Long Short-Term Memory* - Memoria de Corto y Largo Plazo
MLM	*Masked Language Model* - Modelo de Lenguaje Enmascarado
MLP	*Multi-Layer Perceptron* - Perceptrón Multicapa
NBC	*Naïve Bayes Classifier* - Clasificador Naïve Bayes
NLP	*Natural Language Processing* - Procesamiento de Lenguaje Natural
NSP	*Next Sentence Prediction* - Predicción de la Siguiente Oración
OOV	*Out-of-Vocabulary* - Fuera del Vocabulario
PCIe	*Peripheral Component Interconnect Express* - Interconexión de Componentes Periféricos Express
RAM	*Random Access Memory* - Memoria de Acceso Aleatorio
RF	*Random Forest* - Bosque Aleatorio
SVM	*Support Vector Machine* - Máquina de Vectores de Soporte
TLM	*Translation Language Modelling* - Modelado de Lenguaje para Traducción
TF-IDF	*Term Frequency-Inverse Document Frequency* - Frecuencia de Término-Frecuencia Inversa de Documento
TN	*True Negative* - Verdadero Negativo
TP	*True Positive* - Verdadero Positivo
VSM	*Vector Space Model* - Modelo de Espacio Vectorial

Chapter 1

Introduction

This chapter covers the general ideas related to the work developed for this project, a brief description of the challenges that were undertaken, and the objectives proposed to take on these challenges. It also includes a background review of previous work that has led to the current state of the field of study.

1.1. Problem Statement

Automatic text classification (referred to as text classification throughout the rest of this work) is a task of the natural language processing (NLP) field, and has been a subject of study since the 1960s [Sebastiani02]. In particular, supervised text classification is the process of automatically assigning a label from a predefined set to a textual document based on its content. This is accomplished by using a training set where the classifier knows the target labels, and a test set with known target labels where the classifier must predict these labels. The text classification task can be further divided into more specific subtasks, such as the ones proposed by Gasparetto et al. in [Gasparetto22], which include sentiment analysis, topic labelling, named entity recognition, and question answering, among others. Out of these, the objectives outlined in this work will focus on the sentiment analysis and topic labelling subtasks. Sentiment analysis encompasses various goals, such as classifying a document as objective or subjective and determining its polarity as positive or negative (binary classification) or somewhere along this spectrum (multiclass classification) [Taboada16]. This

often includes the challenge of detecting implicit emotions in the text, such as irony or sarcasm [Fang20, Filatova12].

In essence, the process of text classification begins with a labelled dataset of text, commonly referred to as a corpus, where each document (or sample) in the corpus is annotated with its corresponding category or label. Subsequently, pre-processing occurs, encompassing actions such as case normalization, stop word removal, stemming, and the implementation of strategies to handle links, usernames, hashtags, and other elements. Tokenization follows, consisting of breaking down the text into smaller units called tokens, yielding a vocabulary from the normalized documents. These tokens can be words, sub-words, phrases, symbols, or any other meaningful units of text. Tokens are turned into numerical representations through a process called vectorization. These numerical representations can then be used to fit a machine learning algorithm. Converting documents into numerical vectors is also known as feature extraction [Kowsari19]. Feature extraction in itself is a significant challenge in the text classification pipeline, and there exists an extensive array of methodologies and dedicated research addressing this task comprehensively, such as the use of the Bag-Of-Words (BOW) model and a variety of weighting schemes to create a Vector Space Model (VSM) [Manning08], word2vec [Mikolov13], fastText [Bojanowski17], and the representations created by transformer based models [Vaswani23].

In light of the exponential growth of digital text data, particularly through social media, online reviews, and digital communications, the use of text classification models has become a point of interest for businesses and organizations that depend on user feedback to better understand the needs and wants of their clients, as well as the perception of the products and services they provide. Building on this trend, the Iberian Languages Evaluation Forum (IberLEF) has included the "Rest-Mex: Research on Sentiment Analysis Task for Mexican Tourist Texts" (Rest-Mex) task in the 2021, 2022 and 2023 editions. The Rest-Mex task refers to a set of annual challenges centred around text analysis in the Mexican tourism sector. Participants engage in tasks like sentiment analysis, recommendation system development, and data clustering to contribute to the field and improve tools that help businesses and organizations understand and respond to user feedback effectively [Álvarez-Carmona23]. In the 2023 edition of Rest-Mex, one of the goals is to perform sentiment analysis on reviews

shared by tourists who visited prominent places (e.g., monuments, museums, parks, etc.), restaurants, and hotels in Mexico, Cuba, and Colombia. The reviews are in Spanish and were taken from TripAdvisor. Apart from performing sentiment analysis, the task also involved topic labelling by classifying the reviews to the country in which the reviewed place is located, as well as the type of place the review is about.

Tourism is a vital component of the global economy, with its impact differing from one country to another. In some nations, the local tourism industry is a major economic pillar. As a modern socio-economic phenomenon that encompasses both material and human elements, tourism plays a significant role in the development of economies and societies worldwide. It promotes globalization and is a key factor in sustainable development, with its contribution to GDP reflecting the development level of each country [Bunghez16]. Specifically in Mexico, tourism holds great significance as a cornerstone of the nation's social and economic structure, as the year 2023 saw an influx of over 42 million tourists with a contribution of over 30 billion Mexican dollars, while providing over 4.7 million jobs related to the tourism industry [1].

As of 2022, the majority of scientific study related to the applications of NLP in the tourism industry has been focused on the English language [Álvarez-Carmona22a]. Given the critical role of tourism in economies like Mexico's, understanding and interpreting tourist feedback is essential for enhancing services and ensuring customer satisfaction. However, analysing such feedback presents challenges, especially when it involves nuanced language elements like sarcasm. Sarcasm is a form of irony that indirectly communicates an opinion with the intended meaning diverging from the literal one, such as a word or expression with prior positive polarity being used figuratively for expressing a negative opinion [Carvalho09].

To address these challenges, another sentiment analysis task introduced in the Forum for Information Retrieval Evaluation (FIRE) is taken into consideration. The 2024 edition of this task focuses on sarcasm detection in code-mixed comments or posts collected from social media platforms like YouTube, particularly in Dravidian languages like Tamil-English and Malayalam-English. Code-mixing is prevalent in multilingual communities, where users switch between languages within a single conversation or even a single sentence. This

[1] https://www.datatur.sectur.gob.mx/RAT/RAT-2023-12(ES).pdf

phenomenon is further complicated when texts are written in non-native scripts, such as using the Roman or Latin alphabet for Tamil or Malayalam [Chakravarthi23a].

The main focus of this work is to develop a classification system based on ensembles from the stacked probability outputs of deep learning pre-trained models and other traditional machine learning methods, with the intent of acquiring better performance using the ensemble system compared to using the individual methods. The proposed system will be applied to the sentiment analysis task from the Rest-Mex 2023 competition and the sarcasm identification task from the DravidianCodeMix@FIRE-2024 competition, which will serve as a reference to evaluate the effectiveness of the system.

The ensemble-based system proposed in this work is created with the intent to leverage the strengths of various classifiers, specifically BERT, fastText, and μTC. By stacking the decision function and probability outputs of these models, a new vector space is constructed. This vector space is then used to train an XGBoost classifier, which adjusts the contributions of each model to generate a final prediction.

Due to initial limitations in computing resources, the first implementation of this ensemble-based system was performed using subsets from the original dataset from the Rest-Mex 2023 competition, given its size. The results from this approach were submitted for evaluation to the competition organizers. After the competition concluded, the use of a more powerful cluster was made available, and it was possible to implement the system using larger subsets as well as the complete dataset. As such, the experiments for the Rest-Mex 2023 competition were performed using different sizes of subsets, giving an opportunity to compare the performance across various subset sizes.

1.2. Background

The roots of automatic text categorization can be traced back to the information retrieval domain. In 1960 Maron and Kuhns' [Maron60] published a work in which they introduced a technique called "Probabilistic Indexing" that quantised document relevance through statistical methods that allowed for organizing and searching literature in computerized library systems. Their approach allowed a computer to assign a "relevance number" to each

document based on a user's query. This number estimates the probability that the document will satisfy the user's information needs. Consequently, the system could generate a list of documents ranked by how likely they are to be relevant, making searches more effective. This technique is an application of a Naïve Bayes Classifier (NBC) [Lewis98].

Since this implementation, a variety of text classification approaches have been proposed and are generally divided in traditional methods and deep learning methods in more recent surveys [Li22, Kowsari19]. Traditional methods include rule based systems in the form of **if** $DNF_{formula}$ **then** *category* where DNF stands for disjunctive normal form, and it represents a disjunction of conjunctive clauses. These systems predominated in the field throughout the 1980s [Sebastiani02], but had as a drawback that they relied heavily on expert knowledge and manually predefined linguistic rules.

The successors of the rule based systems were applications of machine learning that saw greater use from the 1990s up until the early 2010s, still considered to be traditional methods. These are methods related to conventional machine learning, and will be briefly overviewed in the following paragraphs.

One category within these methods is the previously mentioned NBC. After it's initial implementation by Maron and Kuhns, it saw a revival in its use during the 1990s [Lewis98] and its variants were used as a baseline in text classification [Xu18]. Some of its popular variants are Gaussian NBC, Multinomial NBC and Bernoulli NBC, and have been used for sentiment analysis in multiple research papers [Wongkar19, Abbas19, Singh19].

Logistic Regression (LR), while originally a statistical method, is also widely used as a machine learning technique for binary and multi-class text classification. Its simplicity and efficiency make it a strong baseline method for many classification problems. Genkin et al. presented a large-scale Bayesian logistic regression for text categorization, showing that LR can effectively handle high-dimensional text data [Alexander Genkin07].

Another important category is the K-Nearest Neighbors (K-NN) algorithm. K-NN is a method that, in the context of text classification, classifies a document based on the majority class of its k closest neighbours in the document feature space. First introduced by Cover and Hart [Cover67], it has been the subject of study to improve its performance, such as the work done by Han et al. [Han01] who proposed a weight-adjusted K-NN algorithm for

text categorization. The K-NN algorithm has also been widely used for sentiment analysis tasks [Hota18, Isnain21].

Support Vector Machines (SVMs) were introduced by Cortes and Vapnik for the first time to perform binary classification [Cortes09] and applied to the field of text classification by Joachims, achieving better performance than competing methods at the time [Joachims98]. Given an n-dimensional space, the SVM model seeks to find an (n-1)-dimensional hyperplane that can separate the data points into two classes. This hyperplane acts as a decision boundary. There are many hyperplanes that could separate the two classes of data. However, not all hyperplanes provide the best separation. An SVM selects the hyperplane that maximizes the margin between the two classes. The margin is the distance from the hyperplane to the nearest data point of each class. In the case of multiple classes, the multiclass problem is transformed into multiple binary classification problems. SVMs are a popular method of text classification, and have been extensively used for sentiment analysis tasks [Ahmad17, Huq17, Zainuddin14]. Furthermore, the SVM model has been used as the classification algorithm employed by Tellez et al. in their text classification library μTC, which has been shown to perform competitively with state-of-the-art methods [Tellez18].

Decision Trees (DT) and Random Forests (RF) constitute another category of traditional methods. Decision Trees classify instances by traversing from the root to a leaf node, making decisions based on feature values at each node. Random Forests enhance this approach by constructing multiple decision trees and averaging their results, which improves classification accuracy and reduces the risk of overfitting. The RF technique is considered a bootstrap aggregation or bagging method [Breiman01] and is further categorized as an ensemble learning or integration-based algorithm, along with boosting [Gasparetto22, Kowsari19]. Boosting methods train weak classifiers sequentially, with each classifier taking into account the mistakes from the previous classifier until a termination condition is met. Although bagging or boosting is not necessarily performed using DTs, popular implementations of these methods use DTs, such as AdaBoost [Freund99] and XGBoost [Chen16].

In particular, XGBoost, an optimized gradient boosting framework, has gained significant popularity in machine learning due to its efficiency and high performance in predictive modelling [Chen16]. It has achieved state-of-the-art results in various structured

data tasks, such as the classification of fake news [Haumahu21].

Shallow neural network-based methods bridge the gap between traditional methods and deep learning methods. An example of these methods is the Multi-Layer Perceptron (MLP) model, a type of feedforward neural network with one or more hidden layers. Wiener et al. [Wiener95] explored using neural networks, including MLPs, for topic spotting in text, demonstrating their potential in capturing complex patterns. Another example is the fastText classifier developed by Joulin et al. that only uses three layers while achieving competitive results [Joulin16]. fastText offers a text classification approach that modifies the popular Word2Vec algorithm. Instead of training word embeddings to predict surrounding words using a Continuous Bag of Words model, fastText trains embeddings to predict predefined labels directly.

The use of deep learning methods has seen an increase in popularity, as these methods allow to better capture meaningful relationships between units of information in textual data, as well as reduce the dimension of their feature vectors compared to traditional methods [Gasparetto22]. There exists an ample amount of research dedicated to these methods, with the most important advances in the field leading to the development of this work being the development of the transformer architecture.

Transformers, introduced by Vaswani et al. in 2017 [Vaswani23], have contributed significantly to natural language processing by using self-attention mechanisms to capture long-range dependencies in text. Since their introduction, transformers have been applied successfully to various tasks. For instance, BERT has been used for machine translation [Devlin19], and GPT-2 has been utilized for text generation [Radford19], both achieving state-of-the-art performance in multiple domains.

Transformer-based models have also become prevalent in text classification. This was evident in the past edition of the sarcasm identification task from the FIRE 2023 [Chakravarthi23b]. In this competition, Bhaumik and Das [Anik Basu23] achieved the best performance for the Tamil-English language pair using MuRIL [Khanuja21b], and they also secured second place in the Malayalam-English category with the same model.

However, not all participants opted for transformer models. Krishnan et al. employed simpler classification methods such as Multi-Layer Perceptrons (MLPs), Random Forests,

K-Nearest Neighbors (K-NN), and Support Vector Machines (SVMs) [Krishnan23]. They used count and frequency-based feature extraction techniques instead of the word embeddings typically used in transformers. This approach proved effective, earning them first place in the Malayalam-English category and highlighting the continued value of traditional classification methods.

More examples of the dominance of the transformer model in text classification can be seen in other popular competitions, such as the detection of online sexism from SemEval-2023 [Kirk23], in which the majority of participants used transformer based models. This can also be observed in the sentiment analysis task of past editions of the Rest-Mex competition, where the top participants presented BERT-based models [Álvarez-Carmona21, Álvarez-Carmona22b, Álvarez-Carmona23]

Performing a variety of classification methods on a document and stacking their outputs to use as a feature vector for a final classifier can be considered an application of stacked generalization [Wolpert92]. This technique has been used by Graff et al. through their sentiment analysis system EvoMSA across datasets from numerous competitions [Graff20]. In their work, they describe the different models used to create a feature space to fit a final classifier, which is based on genetic algorithms. A similar idea is also explored by Tasnia et al. in 2023 for multilingual humour and irony detection, where instead of stacking outputs from different classification methods, they stack document embeddings obtained from different sources, and then passing the unified embedding to an LSTM classifier [Tasnia23].

This has culminated in the work presented in this document, with an initial attempt taken in the sentiment analysis task of the Rest-Mex 2023 competition. The research presented for this competition employed the use of stacked outputs from BERT-based models, fastText, and μTC as the input of an XGBoost classifier, and showed acceptable results compared to other participants in the competition [Cerda-Flores23]. These studies faced challenges due to computational limitations, which led to the use of subsets of the original dataset for model training. The current work addresses this issue by using more powerful computing resources.

1.3. Objectives of the Thesis

1.3.1. General Objectives

To develop a text classification system for sentiment analysis, topic labelling and sarcasm identification in two different competitions that achieves high-ranking results, using ensembles of stacked outputs from different pre-trained classification models and methods, while also performing better when compared to the individual models and methods.

1.3.2. Particular Objectives

- Utilize different text classification methods to perform the sentiment analysis task proposed in the 2023 edition of Rest-Mex as well as the sarcasm identification task from the DravidianCodeMix@FIRE-2024 competition, particularly μ-TC, fastText and BERT.
- Perform the Rest-Mex sentiment analysis task using subsets from the original training dataset to reduce the processing load, a limiting factor in less powerful systems.
- Stack the outputs obtained as a result of the previous objectives and use them in the creation of XGBoost ensemble classification models.
- The evaluation of this ensemble approach aims to demonstrate its potential to improve sentiment analysis performance, differing from current literature by using outputs from pre-trained deep-learning models instead of training from scratch, along with shallow-learning and traditional classification methods, ultimately contributing to the broader field of text classification and NLP.

1.4. Chapter Descriptions

Chapter 1 presents a brief introduction to text classification, the background, significance, and objectives of the study, outlining the problem statement and case studies to be addressed. Chapter 2 details the theory behind the techniques and tools used for this research, along with a description of the data provided to conduct experimentation.

Chapter 3 describes the implementation of the methods, elaborating on the experimental design and setup followed to conduct the study. Chapter 4 presents the findings derived from the experimentation, offering an analysis and interpretation of the data in relation to the research objectives. Finally, Chapter 5 summarizes the key insights gained, discusses the implications of the results, and provides proposals for future research in the field.

Chapter 2

Methods

This chapter provides a theoretical overview of the methodologies employed in this research. It begins with a data analysis on the datasets provided for the Rest-Mex 2023 and the DravidianCodeMix competitions. This is followed by detailing common model evaluation metrics used to assess classification performance, as well as specific metrics proposed by the Rest-Mex 2023 competition organizers. Next, the main ideas behind representing text as vectors are explained, such as tokenization and vectorization techniques that serve as the base for text processing methods. It is concluded by discussing the machine learning models and frameworks utilized in the study, namely μTC, fastText, BERT, and XGBoost, explaining their architectures and functionalities.

2.1. Analysis of the Competition Datasets

2.1.1. Rest-Mex 2023 Dataset

The dataset DS_{RM} provided by the organizers of the Rest-Mex 2023 consisted of 359565 reviews by tourists who shared their opinions about places, restaurants and hotels in Mexico, Cuba and Colombia between 2002 and 2022 taken from TripAdvisor [Álvarez-Carmona23]. It is split into two sets; a training set DS_{RM_Train} containing 251702 of the reviews, which amounts to around 70 % of the complete dataset, and a test set DS_{RM_Test} containing the remaining 107863 reviews, which amounts to the remaining 30 %. Every sample from the

training data set has a review title, the review itself, the polarity of the review (1, 2, 3, 4, 5), the country of the reviewed place (Colombia, Cuba, Mexico) and the type of place that was reviewed (Attractive, Hotel, Restaurant). The polarity values can be interpreted as the amount of stars given by a review, ranging from 1 star to 5 stars. The samples from the test dataset only contained a sample identifier, a review title and the review.

Table 2.1 shows the distribution of classes in the training dataset DS_{RM_Train} for each of the classification tasks that are to be performed, while the Figure 2.1 shows a graphical representation of the table.

Table 2.1: Distribution of Classes in DS_{RM_Train}

Polarity		**Country**		**Type**	
Class	**Count**	**Class**	**Count**	**Class**	**Count**
5	157095	Mexico	118776	Attractive	111188
4	60227	Colombia	66703	Restaurant	76042
3	21656	Cuba	66223	Hotel	64472
2	6952				
1	5772				

From Figure 2.1 it can be observed that the dataset DS_{RM_Train} is unbalanced, particularly in the polarity distribution. The majority class from polarity, label 5, represents 62.41 % of the total instances, while the minority class, label 1, represents 2.29 %. In the polarity spectrum, labels 4 and 5 are considered positive, label 3 is considered neutral, and labels 1 and 2 are considered negative. As such, the overwhelming majority of samples in the polarity distribution consist of positive reviews, which could cause issues in training due to the unbalanced nature of the dataset. In the case of the country and type distributions, although also unbalanced, they are not unbalanced enough to pose a problem. To handle the unbalanced nature of the polarity classes, the organizers propose the metric (2.6) discussed in detail in Section 2.2.2.

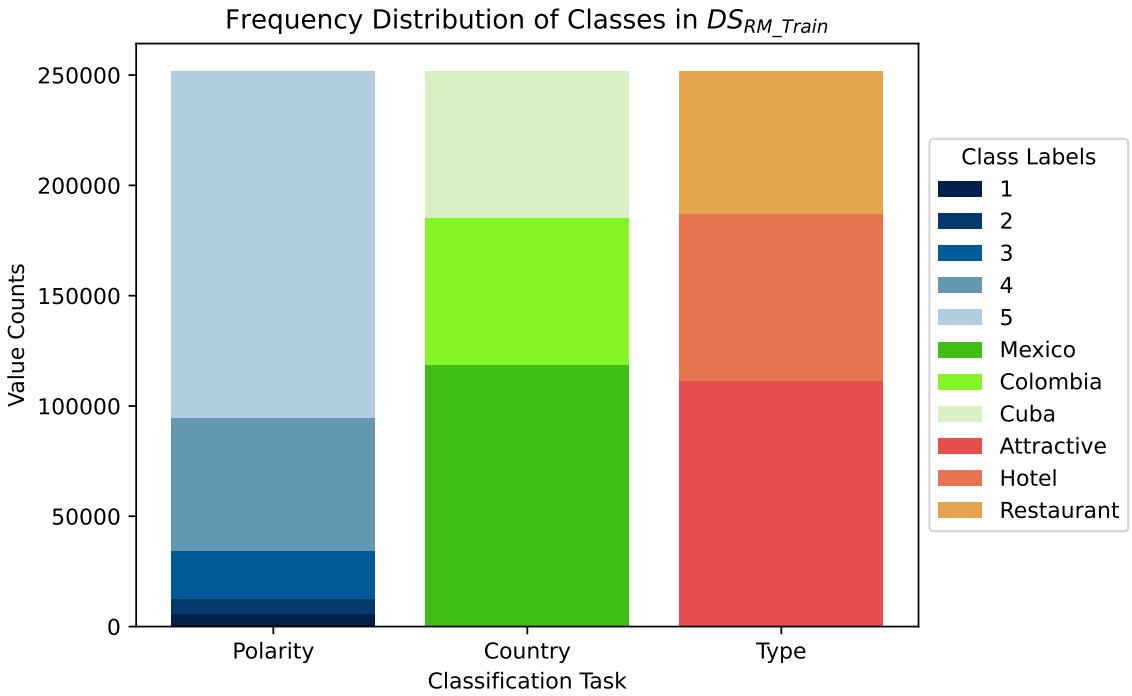

Figure 2.1: Distribution of Classes in DS_{RM_Train}

2.1.2. DravidianCodeMix@FIRE-2024 Dataset

The organizers of the sarcasm detection task from FIRE 2023 provided the datasets for training, validation and testing for Malayalam and Tamil languages code-mixed with English sourced from the comment section of movie trailer videos in YouTube[1], resulting in a total of 6 datasets. Henceforth, the Malayalam datasets for training, validation, and testing will be referred to as $DS_{FIRE\_TrainMal}$, DS_{FIRE_ValMal}, and $DS_{FIRE\_TestMal}$ respectively, and the datasets for Tamil will similarly be referred to as $DS_{FIRE\_TrainTam}$, DS_{FIRE_ValTam}, and $DS_{FIRE\_TestTam}$. The samples from the training and validation sets contain the comment and its label indicating if it is sarcastic or non-sarcastic, while the samples from the test sets contained the sample identifier and the comment.

Table 2.2 shows the distribution of classes in the training datasets $DS_{FIRE\_TrainMal}$ and $DS_{FIRE\_TrainTam}$, as well as the validation sets DS_{FIRE_ValMal} and DS_{FIRE_ValTam}. Figure 2.2 shows the percentage of classes for the sum of $DS_{FIRE\_TrainMal}$ with DS_{FIRE_ValMal}

[1]https://sites.google.com/view/dravidian-codemix-fire2024/home

and the sum of $DS_{FIRE\_TrainTam}$ with DS_{FIRE_ValTam}.

Table 2.2: Distribution of Classes in Training and Validation sets for Malayalam and Tamil

Dataset	Count	Non-sarcastic	Sarcastic
$DS_{FIRE\_TrainMal}$	13188	10689	2499
DS_{FIRE_ValMal}	2826	2305	521
Malayalam Total Count	16014	12994	3020
$DS_{FIRE\_TrainTam}$	29570	21740	7830
DS_{FIRE_ValTam}	6336	4630	1706
Tamil Total Count	35906	26370	9536

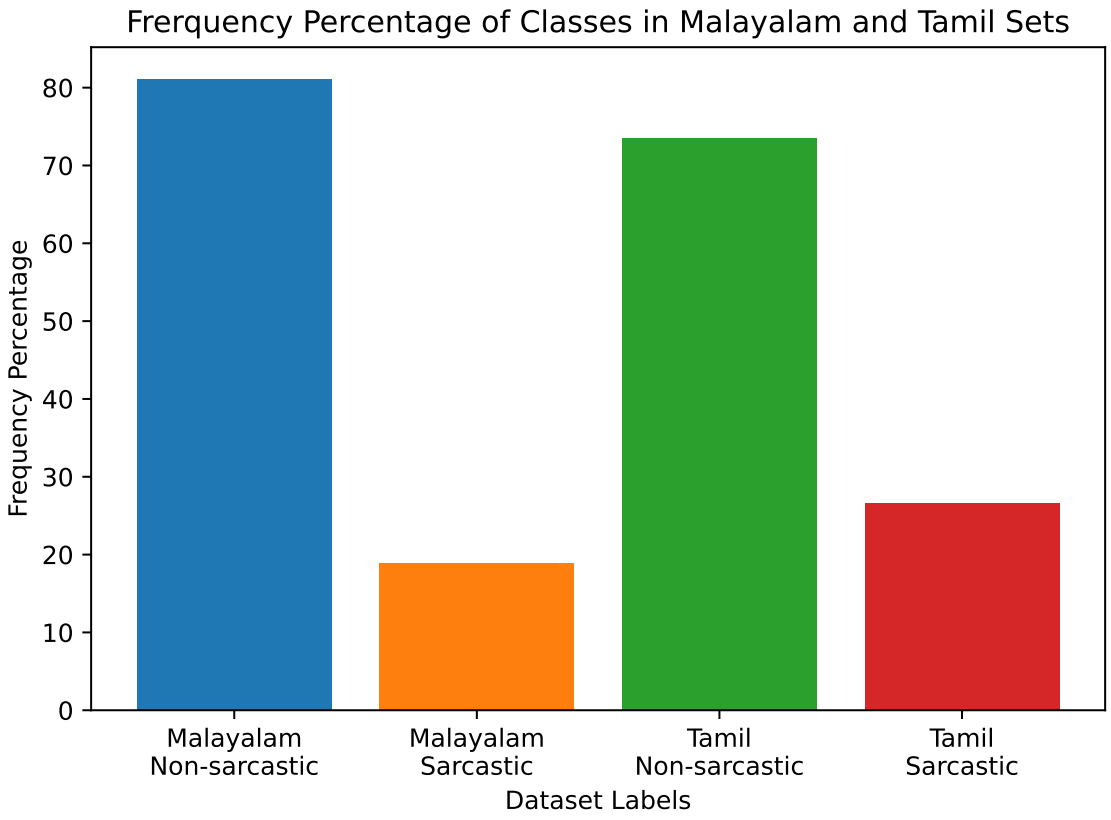

Figure 2.2: Frequency Percentage of Classes in Sum of $DS_{FIRE\_TrainMal}$ and DS_{FIRE_ValMal} and Sum of $DS_{FIRE\_TrainTam}$ and $DS_{FIRE\_ValTrain}$

As observed in Figure 2.2, approximately 80 % of the Malayalam dataset samples and 70 % of the Tamil dataset samples are Non-sarcastic, with the remaining samples being sarcastic. There is considerable class imbalance in the datasets for both languages. This is intentional, as the organizers sought to depict real-life situations in which only a fraction of comments tends to express sarcasm[2].

2.2. Model Evaluation Metrics

2.2.1. General Text Classification Metrics

There exists an extensive array of publications and surveys that cover evaluation metrics for classification problems, such as [Obi23, Hossin15, Ferri09, Vujovic21], to list a few. An important part of binary and multiclass text classification is the correct evaluation of the proposed model. During the training and evaluation process, this is done by comparing the predictions from the model with the true labels of the dataset, also known as the ground truth. To analyse the model's performance, the predictions are categorized into four different outcomes: **true positives** (TP), **true negatives** (TN), **false positives** (FP), and **false negatives** (FN).

In binary classification, the classes are encoded to a positive (1) and a negative (0) class, and a TP occurs when the model correctly classifies a sample in the positive class, while a TN occurs when the model correctly classifies a sample as negative. Conversely, an FP occurs when the model classifies a negative sample as positive, and a FN occurs when it classifies a positive sample as a negative. While simple to understand, it is important to review these terms, as they provide the basis for the classification metrics to be reviewed, which will in turn be limited to those used by the organizers of each competition. Table 2.3 illustrates a confusion matrix for the case of binary classification.

In the case of multiclass classification, the same principles from binary classification are applied, but the outcomes are applied on a per-class basis. This is to say, when applying a performance metric, one class is treated as a positive class, while the remaining classes are grouped together to form the negative class. For each class, TP, TN, FP, and FN are

[2]https://sites.google.com/view/dravidian-codemix-fire2024/home

Table 2.3: Confusion Matrix for Binary Classification

Class Designation		**Actual Class**	
		True	**False**
Predicted Class	**Positive**	TP	FP
	Negative	FN	TN

calculated individually, and the results are aggregated to compute a desired metric.

As previously mentioned, the metrics that will be reviewed in this section are limited to those that were used in the competitions in which the proposed classification system was applied, and are described similarly to the previously mentioned work that delves deeper into these methods. These are **accuracy**, **precision**, **recall**, **F1-score**, and **Macro F1-score**, as well as several others proposed by the organizers of the Rest-Mex competition.

Accuracy

Accuracy measures the ratio of correctly classified samples over the entire amount of samples. The resulting value lies in the range [0, 1], where a higher value indicates better performance. In terms of the previously defined concepts, it is represented by (2.1).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

While this metric provides information on the general performance of the model, it doesn't allow one to assess how well the model performed predictions for each class.

Precision

Precision measures the ratio of correctly predicted samples of the positive class over the total amount of predicted positive samples; the fraction of positive predictions that is truly positive. The resulting value lies in the range [0, 1], where a higher value indicates better performance. It is represented by (2.2).

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

In more general terms, it represents a model's ability to not predict a sample of the negative class as positive.

Recall

Recall, also known as Sensitivity or True Positive Rate, measures the ratio of correctly predicted samples of the positive class over the total amount of samples from the positive class. The resulting value lies in the range [0, 1], where a higher value indicates better performance. It is represented by (2.3).

$$Recall = \frac{TP}{TP + FN} \tag{2.3}$$

It represents a model's ability to identify samples of the positive class.

F1-score

The F1-score is a measure that calculates the harmonic mean of the precision and recall metrics for a single class. The resulting value lies in the range [0, 1], where a higher value indicates better performance. It is represented by (2.4).

$$\text{F1-score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.4}$$

It is used as an alternative to the **accuracy** metric and is useful for the cases where there is class imbalance in the dataset, as it focuses on how well a model performs on both positive and negative predictions. Being the harmonic mean of precision and recall, it provides a balanced score between them in the case of one significantly outperforming the other, only resulting in a high score if both metrics are reasonably high as well.

Macro F1-score

The Macro F1-score is a variation of the F1-score for multi-class classification problems that evaluates the performance of a model across all classes. This metric is obtained by calculating the F1-score for each class and averaging all the computed F1-scores. The resulting value lies in the range [0, 1], where a higher value indicates better performance. It is represented by (2.5).

$$\text{Macro F1-score} = \frac{1}{N} \sum_{i=1}^{N} \text{F1-score}_i \tag{2.5}$$

Where N is the total number of classes.

2.2.2. Rest-Mex Specific Evaluation Metrics

The Sentiment Analysis task from the Rest-Mex 2023 consisted of determining the polarity of a given review about a tourist place, as well as the type of place the review is about and the country of origin of the place being reviewed. The organizers of the Rest-Mex 2023 competition proposed (2.6), (2.7), (2.8), and (2.9) to evaluate the sentiment score of a proposed system.

$$Res_P(k) = \frac{\sum_{i=1}^{|C_P|} \left(\left(1 - \frac{T_{C_{P_i}}}{T_{C_R}} \right) \cdot F_{P_i}(k) \right)}{\sum_{i=1}^{|C_P|} \left(1 - \frac{T_{C_{P_i}}}{T_{C_R}} \right)} \tag{2.6}$$

$$Res_A(k) = \frac{F_A(k) + F_H(k) + F_R(k)}{3} \tag{2.7}$$

$$Res_C(k) = \frac{F_{Mex}(k) + F_{Cub}(k) + F_{Col}(k)}{3} \tag{2.8}$$

$$Sentiment(k) = \frac{2Res_p(k) + Res_A(k) + Res_C(k)}{4} \tag{2.9}$$

In all equations, k represents the participant system, which is to say, a model or method being evaluated.

(2.6) calculates a weighted arithmetic mean of the Macro F1-score for the polarity prediction task. In this equation, the term $\left(1 - \frac{T_{C_{P_i}}}{T_{C_R}} \right)$ serves as the weight for the i-th label from the set of polarity classes $C_P = [1, 2, 3, 4, 5]$, with T_{C_R} being the total number of samples in a Rest-Mex dataset and $T_{C_{P_i}}$ being the total number of instances of the i-th label in C_P. $F_{P_i}(k)$ is the F1-score for the i-th class in C_P given by the system k.Using this measure ensures that the Macro F1-score for minority classes in any given dataset will have more importance than the score from majority classes. This can be interpreted as a measure taken by the organizers to address the class imbalance present in the provided datasets.

(2.7) calculates the average Macro F1-score for the type prediction task, giving equal importance to each class. $F_A(k)$ is the Macro F1-score for the "Attractive" class, $F_H(k)$ is the Macro F1-score for the "Hotel" class and $F_R(k)$ is the Macro F1-score for the "Restaurant" class, all obtained by the system k.

(2.8) calculates the average Macro F1-score for the country prediction task, similar to (2.7). $F_{Mex}(k)$ is the Macro F1-score for the "Mexico" class, $F_{Cub}(k)$ is the Macro F1-score for the "Cuba" class and $F_{Col}(k)$ is the Macro F1-score for the "Colombia" class, all obtained by the system k.

(2.9) is the final sentiment measure for a system k. This equation combines the three measures into a final sentiment score, with more weight given to the polarity classification. The final score is the average of the results from the previous equations, $Res_P(k)$, $Res_A(k)$ and $Res_C(k)$, giving twice the weight to polarity classification to emphasize its importance in the overall sentiment score.

2.3. Representation of Text as Vectors

The classification methods used throughout this work make use of machine learning models that require numerical data to make predictions. The process of transforming text into numerical data, also known as vectorization, has been extensively studied, and ample amounts of methodologies exist to carry out this objective. The following section provides a general overview of the terminology and procedures utilized by the libraries which were worked with.

2.3.1. Tokenization

In the context of NLP, a token is a single unit of text extracted from a larger text sequence, such as a sentence. Tokenization is the process of dividing the text in a given document into smaller, meaningful units. Very often, it is a good idea to perform a pre-processing step on the raw textual data before proceeding with tokenization. Pre-processing consists of a variety of text transformations, ranging from very general lower casing, punctuation removal or emoji and emoticon handling, to more language specific transformations such as lemmatization, stemming, or removal of stop-words.

A tokenizer can perform the segmentation of the sentence at various levels, the most intuitive being splitting the sentence into the words it contains. Other levels include but are not limited to n amount of characters (known as character n-grams), and n amount of

words (known as word n-grams). Utilizing character n-grams captures the structures of words (such as prefixes, suffixes or roots) which is useful in the representation of morphologically rich languages. Another benefit of using character n-grams is the ability to handle Out-of-Vocabulary (OOV) words; if and OOV word occurs when predicting a new sentence, an embedding for it is still generated based on the character n-grams that compose it.

It is even possible to create tokens using a word skipping scheme known as skip-grams, where a minimum of two words are taken from the document, but skipping a minimum of one middle word. For example, taking the following sentence s_1:

"The sky is cloudy today."

After some simple pre-processing (lower casing and punctuation removal), tokenizing s_1 by words would yield the following tokens:

the | sky | is | cloudy | today

Tokenizing s_1 using a scheme of word 3-grams would yield the following tokens:

the_sky_is | sky_is_cloudy | is_cloudy_today

While tokenizing s_1 using a skip-grams scheme taking two words and skipping one in the middle ((2, 1) skip-grams) would yield the following tokens:

the_is | sky_cloudy | is_today

More than one tokenization scheme can be performed on a given document, i.e., tokenizing s_1 by words and using (2, 1) skip-grams yields:

the | sky | is | cloudy | today | the_is | sky_cloudy | is_today

It is important to note that the tokenization process can also append extra characters to a token or extra tokens to denote the beginning of a sentence or the conjunction of one or more words, such as the "_" character in the previous examples. The use of these characters and tokens depends on the implementation of the tokenizer and the considerations taken in the later vectorization process.

The set of tokens obtained are used to build a vocabulary, which is used to perform the transformation of text into vectors. The usage of the vocabulary depends on the method employed to create the vector representations.

2.3.2. Vectorization of Tokens

There are a variety of methods to represent text as vectors, and they can be divided in traditional count and frequency-based methods, and the more relatively modern prediction-based methods, the latter being commonly referred to as *word embeddings*. The following sections give a brief overview of the methods primarily used by the software libraries employed for this work.

Count and Frequency-Based Methods

An initial approach to obtain a numerical representation from text could be to employ a binary representation of each token in a given text sequence from a larger corpus. Making an example corpus from s_1 and a second sentence s_2:

"Is tomorrow Monday?"

By tokenizing both s_1 and s_2 by words, after lower casing and removing punctuation, the resulting vocabulary V_1 in alphabetical order would be:

[cloudy, is, monday, sky, the, today, tomorrow]

Each token can be represented as a binary vector, where 1 indicates the index position of the token in the established vocabulary. So the vector representations for the tokens "monday", "cloudy", and "today" would be, respectively:

$$x_{monday} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \; x_{cloudy} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \; x_{today} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

This representation is known as ***One-Hot Encoding***, where only one element in the vector is set to one, thus, *hot*. Using this method, each token's vector has a length equal to that of the vocabulary. Once the tokens from a text document are one-hot encoded, it can be represented as a matrix, where each row corresponds to a word in the document and each column represents a word from the vocabulary. While this method is simple to carry out, it comes with several disadvantages that can greatly impact the performance of a classifier, such as high dimensionality and sparsity when dealing with a larger vocabulary. It also does not capture the similarities or contextual meaning between words.

The next step in getting numerical representations of text is to employ the ***Bag-Of-Words*** (BoW) model. Unlike One-Hot Encoding, where vectors are created for each token in a given text document and a matrix is built to represent the entire document, BoW creates a single vector for the whole text document by counting the occurrence of each token from the vocabulary in the target document. Another difference between BoW and One-Hot Encoding is that while the vectors obtained from One-Hot Encoding are binary, the vectors obtained through BoW aren't necessarily so, as there could be multiple instances of the same token in a single document. Taking as an example once again s_1 and s_2 with the vocabulary V_1, the BoW representations would be, respectively:

$$x_{s1} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \; x_{s2} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Using this approach provides a slight improvement over One-Hot Encoding because by counting the frequency with which tokens appear provides insight into their importance. However, with large vocabularies, this method still suffers from sparsity and high dimensionality problems, as well as also lacking contextual and semantic information due to the fact that the order of the tokens in the document being ignored. Another drawback to the BoW model is that while the frequency of certain tokens could give more insight on their importance, there's a considerable amount of scenarios where certain tokens may be very common across different documents but do not contribute much information, such as "the", "a", "an", "and", etc. For the purposes of text classification, it is relevant to take into consideration the importance of certain tokens in the vocabulary, seeing as these could be useful features for the machine learning algorithms employed in classification.

To take into account the possible high occurrence of uninformative tokens and boost the importance of others, a weighting scheme known as *term frequency-inverse document frequency* (tf-idf) can be applied to the document vectors that takes into account the amount of times a token appears in a single document as well as the amount of documents it appears in. When applying this weighting scheme, the vectorization process turns the BoW model into what is known as a ***Vector Space Model*** (VSM).

The tf-idf is a numerical statistic of term importance. It indicates the relevance of a token to the document it appears in, taking in consideration the entire corpus of documents from which the vocabulary of tokens was built. It is composed of two statistics; *term frequency* and *inverse document frequency*, with the resulting tf-idf weight being the product of these statistics, as described in (2.10).

$$tf\text{-}idf(t, d, D) = tf_{t,d} \cdot idf_{t,D} \tag{2.10}$$

Term frequency ($tf_{t,d}$) is a measure that refers to how many times token t appears relative to the total number of tokens in the document d, while inverse document frequency ($idf_{t,D}$) measures how much information a token t provides to the document vector relative to all documents in the corpus D, or put more intuitively, how rare a token is in the collection of documents. The formulas for $tf_{t,d}$ and $idf_{t,D}$ are provided in (2.11) and (2.12), respectively.

$$tf_{t,d} = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{2.11}$$

$$idf_{t,D} = \log \frac{N + 1}{|d : d \in D \text{ and } t \in d| + 1} \tag{2.12}$$

In (2.11), the numerator $f_{t,d}$ is the amount of times the token t appears in the document d, and the denominator is the sum of all occurrences of all tokens t' in the document d. In (2.12), the numerator N is the total amount of documents in the corpus D, and the denominator is the cardinal of the set of documents d contained in D that also contain the token t. The addition of one to the document frequencies is done to prevent division by zero.

Applying the $tf\text{-}idf$ weighting scheme to the vectors v_{s1} and v_{s2} from the previous BoW model example would yield the vectors x'_{s1} and x'_{s2}:

$$x'_{s1} = \begin{bmatrix} 0.0352 \\ 0 \\ 0 \\ 0.0352 \\ 0.0352 \\ 0.0352 \\ 0 \end{bmatrix}, \; x'_{s2} = \begin{bmatrix} 0 \\ 0 \\ 0.0587 \\ 0 \\ 0 \\ 0 \\ 0.0587 \end{bmatrix}$$

Now, it is noticeable that the index position for the token “is” (second position) is represented by 0, indicating that maybe this token is not as important as others. However,

while the VSM representation is more context-sensitive than BoW by using a weighting scheme, it still ignores word order, semantics, and can result in high dimensional and sparse vectors with larger vocabularies. In the previous examples, the document vectors x'_{s1} and x'_{s2} are only 7-dimensional vectors because there are only 7 tokens in the example vocabulary V_1. However, in real-life scenarios where there are many more documents in a corpus, the dimensionality of the document vectors greatly increases.

Prediction-Based Methods

The vectorization methods under this section are mostly based on neural networks, and the resulting vector representations are known as **word embeddings**. Unlike the count and frequency-based representations, embeddings are dense representations of lower dimensional space. Depending on the method used to create the embeddings, they may also consider the semantic and contextual relationships between tokens by taking into account the surrounding tokens. Vectors generated from words with similar meanings exist close to each other in the vector space.

Currently, there are several methods of generating word embedding, each with its own approach to capturing the relationship between tokens. These range from relatively simple neural network architectures, such as the shallow neural networks used by Word2Vec and fastText, to more complicated architectures, such as the transformer-based Bidirectional Encoder Representations from Transformers (BERT) model. Due to the variety of currently available model architectures, only the methods used throughout this work will be discussed in their corresponding section, namely fastText and BERT.

2.4. Overview of the μTC Framework for Text Classification

μTC is described by Tellez et al. in [Tellez18]. The following section is largely based on their paper, as well as the source code from their implementation, which is open source and can be found in their GitHub repository[3]. They define it as a text classification framework designed to find a competitive text classifier among a potentially large set of possible classifiers

[3]https://github.com/INGEOTEC/microtc

that approaches the search process as a combinatorial optimization problem. It creates a configuration space composed of text transformation functions, text tokenization functions, token vectorization functions, and classification functions. The configuration space defined by μTC is denoted as $\mathcal{C}_\mu$. $\mathcal{C}_\mu$ consists of over 45 million possible configurations at the moment of writing, making the evaluation of $\mathcal{C}_\mu$ in search for the best configuration impractical in most computing setups. The authors of μTC tackle this problem by using local search-based meta-heuristics, namely Random Search and Hill Climbing.

To perform the mentioned meta-heuristics, a configuration graph is created, where the search for a well-performing configuration is performed. In this graph, vertices represent individual configurations from the space $\mathcal{C}_\mu$ and edges represent neighbouring relationships between the configurations in $\mathcal{C}_\mu$, denoted by $\mathcal{N}$. And so, a μTC graph is represented as $(\mathcal{C}_\mu, \mathcal{N})$. Searching through $(\mathcal{C}_\mu, \mathcal{N})$ and evaluating the configurations using a score function thus becomes a combinatorial optimization problem, specifically as the maximization of the score function.

The score function used by μTC evaluates the performance of a configuration for a given dataset D, which in turn is divided into D_{train} and D_{test}. Up to this step in the μTC pipeline, any given configuration from $\mathcal{C}_\mu$ has processed the labelled text in D_{train} using the text transformation functions, text tokenization functions and token vectorization functions. Using the vectorized representation of the text in D_{train} and it's labels, the classification functions in $\mathcal{C}_\mu$ are used to train a classifier. The training process can include the use of stratified k-folds validation. The classifier is used to compute the labels for D_{test}, and its performance is evaluated by comparing the computed labels from D_{test} against the real ones. The metric used to evaluate the configuration is set as default to the macro F1 score, but can be set to use a variety of different metrics, including variations of the F1 score, macro recall and accuracy.

The optimization process used by μTC involves navigating the graph $(\mathcal{C}_\mu, \mathcal{N})$ using the previously mentioned meta-heuristics; Random Search and Hill Climbing. Random Search randomly creates a subset $\mathcal{C}_\mu{}'$ of configurations from $\mathcal{C}$ and finds the best performing configuration in this subset. Then, Hill Climbing searches the neighbourhood of the best configuration in $\mathcal{C}_\mu{}'$ until there is no further improvement on the best configuration.

2.4.1. The μTC Configuration Space

As previously mentioned, the configuration space $\mathcal{C}_\mu$ consists of a series of text transformation functions, text tokenization functions, token vectorization functions, and classification functions. Table 2.4 shows the functions in $\mathcal{C}_\mu$ for text transformation, text tokenization and token vectorization.

As for the classification functions, an unoptimized Support Vector Machine (SVM) with a linear kernel is used as the only option to perform the classification steps in the μTC pipeline. At the moment of writing, the configuration space contains over 45 million possible configurations with the available options for tokenizers and token filters.

2.5. Overview of fastText for Text Classification

fastText is a text classification library that, in contrast to μTC, uses a neural network-based approach to the vectorization of the textual data input, where the resulting vector representations are called *word embeddings*. The tokenization process used by fastText follows both word n-grams and a character n-grams schemes to create the vocabulary of tokens used for vectorization. It was originally implemented as a word embedding generation tool [Bojanowski17], and later adapted for text classification tasks [Joulin16]. The latter cited work is the source for most of the following section.

The library provides different neural network architectures depending on the task that is to be performed. For purely learning word representations as vectors that capture semantic relationships between words, it has available the use of the Continuous Bag of Words (CBoW) and the Skip-Gram models. The webpage provides pre-trained embeddings for 157 languages trained on Common Crawl and Wikipedia using a variation of the CBoW model. In the case of a text classification task where the goal is to learn representations that are useful for predicting the target class labels, it uses a simple linear classifier architecture, where it directly trains the word embeddings and classifies the input text into predefined categories using these learned representations. As this work is centred on text classification, only the architecture used for classification will be explained in greater detail.

Table 2.4: μTC Configuration Space

Function Type	Function	Use
Text transformation	hashtag_option	Remove hashtags, group them together or leave as is
	num_option	Remove numbers, group them together or leave as is
	url_option	Remove URLs, group them together or leave as is
	usr_option	Remove usernames, group them together or leave as is
	emo_option	Remove emojis and emoticons, group them together or leave as is
	ent_option	Remove entities, group them together or leave as is
	del_diac	Remove diacritics or leave as is
	del_dup	Remove duplicated contiguous characters or leave as is
	del_punc	Remove punctuation characters or leave as is
	lc	Turn all text to lower case or leave as is
Text tokenization	token_list	List that dictates the word n-grams, character n-grams, and skip-grams that the model will use to build a vocabulary
Token vectorization	weighting	Select weighting scheme to create vector space (TF, TF-IDF, Entropy)
	token_min_filter	Keep tokens that appear more than set parameter
	token_max_filter	Keep tokens that appear less than set parameter

2.5.1. fastText Classification Model Architecture

The first step taken by fastText is the tokenization and vocabulary creation. The tokenization schemes used by fastText take all text samples in a given dataset D and split the sentences into the specified word n-grams and character n-grams. The resulting tokens are added to the vocabulary. For example, using word 1-gram and character 3-grams on s_1 and s_2 after normalization, the vocabulary generated by fastText would be something similar to the one represented by Table 2.5:

Table 2.5: Vocabulary for s_1 and s_2 Generated with Word 1-grams and Character 3-grams

the	sky	is	cloudy	today	monday
<th	the	he	e s	sk	sky
ky	y i	is	is	s c	cl
clo	lou	oud	udy	dy	y t
to	tod	oda	day	ay>	<is
s t	y m	mo	mon	ond	nda
day	ay>				

It is important to note that Table 2.5 is presented in this format for illustration purposes, in reality the vocabulary is stored as both a list of words (word n-grams) and a hash table of sub-words (character n-grams). Also, in this example the "<" and ">" characters are used to represent the beginning and end of a sentence, respectively. fastText does not necessarily use this representation.

During training, the text classification model initializes an embedding matrix A of dimensions $(V_l \times n)$, where V_l is the length of the vocabulary and each index position points to a specific token, and n is the specified length of the token embedding. The matrix A is initially filled with random values using a uniform distribution over a small range. In essence, the matrix A is a lookup table for the embedding of each token in the vocabulary. Figure 2.3 gives a visual representation of the classification network architecture.

Given a sentence, it is tokenized into words and sub-words. For each token generated from the sentence, the corresponding token embeddings are retrieved from the embedding

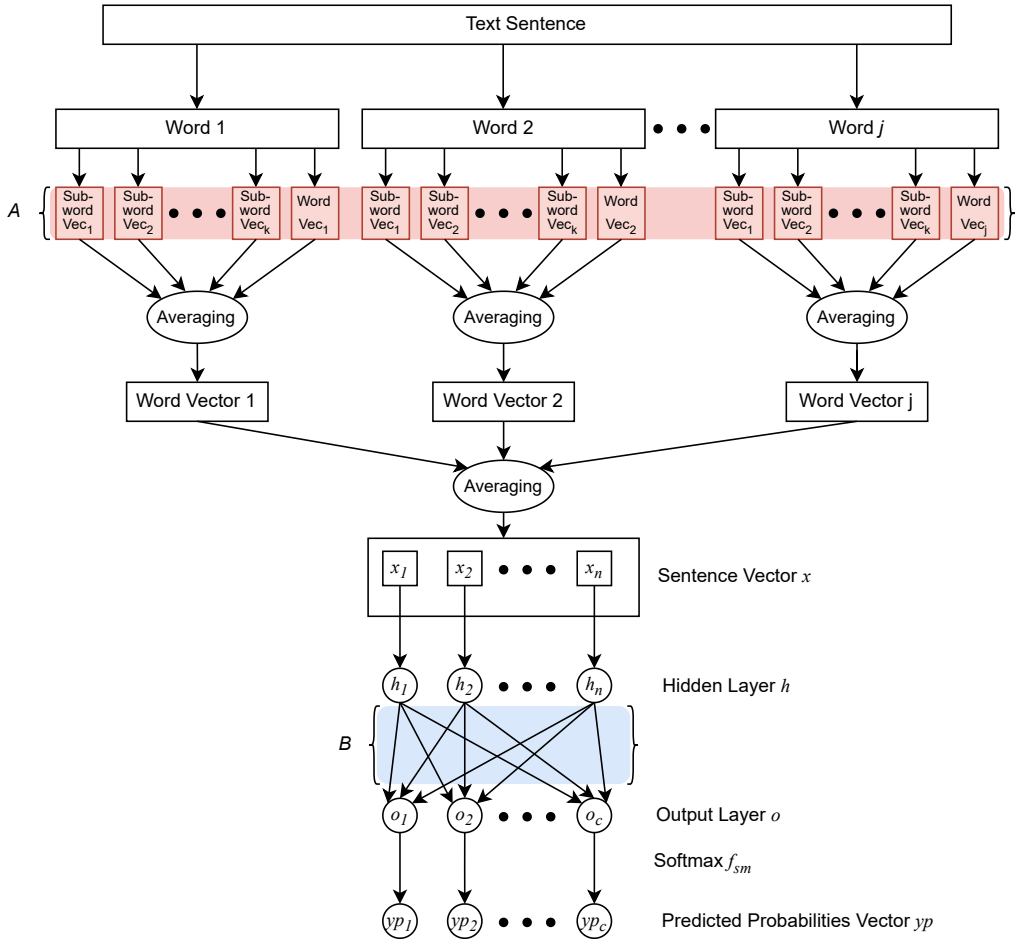

Figure 2.3: fastText Classification Model Architecture

matrix A and averaged to create a word embedding. The word embeddings are then averaged to create a single feature embedding x that represents the entire sentence and is of length n. This sentence embedding is passed directly as input of a hidden layer h. The hidden layer is then multiplied by a second matrix B of dimensions $(C \times n)$, where C is the number of detected classes. The result of this multiplication is an output layer o of length C, which is then passed to a softmax function $f_{softmax}$ to obtain a final probability distribution over the detected classes yp.

The backpropagation step is performed using the negative log likelihood of the predicted classes over the real classes as the loss function and stochastic gradient descent to update the values of A and B. For a collection of N documents, the entire model architecture can be summarized by (2.13), where y_i^t is the real class t for the i-th document and $f_{softmax}(BAx_i)$ is the predicted class for that same document.

$$-\frac{1}{N}\sum_{i=1}^{N}\log L(y_i^t \mid f_{softmax}(BAx_i)) \tag{2.13}$$

2.5.2. fastText Hyperparameter Optimization

fastText, like μTC, also provides the option to perform hyperparameter optimization. While there isn't much documentation explaining in detail the optimization process [4], looking through the source code provides some insights into how hyperparameter optimization was implemented[5].

The optimization process iterates over different configurations of hyperparameters. Firstly, initial values for tuneable hyperparameters are randomly generated. The model is trained with the current set of hyperparameters and its performance is evaluated based on the defined metric (e.g., accuracy, F1-score). Gaussian sampling is then used to explore the hyperparameter space, and in the next iteration of the process the new hyperparameter values are used to once again train and evaluate the model. If a better set of parameters is found, it is saved as the current best configuration. This process is repeated until a pre-defined time limit is reached or a sufficient number of configurations have been tested. This process also

[4] https://ai.meta.com/blog/fasttext-blog-post-open-source-in-brief/
[5] https://github.com/facebookresearch/fastText

includes the option to constrain the size of the model.

As of writing, the hyperparameters that are optimizable are listed in Table 2.6, based on the source code.

2.6. Fundamentals of BERT and Transformer Models

BERT (Bidirectional Encoder Representations from Transformers) is a transformer based model designed to carry out NLP tasks introduced by Devlin et al. in 2019 [Devlin19], from which this following section is based off. The transformer model in itself is a deep learning neural network model popularized in 2017 by Vaswani et al. [Vaswani23]. Unlike μTC and fastText, using BERT and subsequent BERT-based models usually requires downloading the desired pre-trained model instead of training one's own model. Using the pre-trained model allows one to ***fine-tune*** it to a specific task by modifying the input and output layers of the pre-trained model. This section will provide a brief overview on the transformer model, as well as the changes applied on the transformer model to create BERT.

2.6.1. The Transformer Model

The following subsection references the work in [Vaswani23] and [Ravichandiran21]. Transformers were originally implemented as a language translation model that generated a sentence in the target language from a source sentence sequentially, this is to say, one word at a time. The transformer model has at its core the **self-attention mechanism** that is used to compute vector representations of words and sentences, and will be the main point of discussion in this subsection, as it was the main talking point of this model when it was introduced. This mechanism captures the relationships between different words in a sentence, regardless of their distance from one another in the sentence. After discussing the self attention mechanism, the rest of the architecture components will be briefly explained as the BERT model is based on them.

Table 2.6: fastText Optimizable Hyperparameters

Hyperparameter	**Definition**
Epochs	Number of complete passes through the training set
Learning Rate	Factor that influences the step size when minimizing the loss function
Word Embedding Dimension	Length of the word embedding for each token in the vocabulary
Word n-grams	Amount of words treated as an individual token for the vocabulary
Minimum Character n-grams	Minimum length of character n-grams generated for the vocabulary
Maximum Character n-grams	Maximum length of character n-grams generated for the vocabulary
Bucket Dimension	Dimension of the hash stable for sub-word tokens
Loss Function	Loss function used for back propagation (currently only softmax available)
Sub-vector Dimension	Dimension of sub-vectors when specifying a model size

The Self-Attention Mechanism

To understand the working of this mechanism, consider a random sentence s. An initial embedding matrix X is created from s of dimensions $(T \times n)$, where T is the amount of tokens in s and n is the length of each embedding (the initial embeddings are pre-learned from another model). This implies that each row in the matrix X represents the embedding of each token in s. From X, three new matrices are created: Q (query matrix), K (key matrix), and V (value matrix). This is done by multiplying X by three corresponding weight matrices: W^Q, W^K, and W^V. These weight matrices are initialized randomly and are optimized during model training. The dimensions of W^Q and W^K is $(n \times d_k)$, where d_k is the length of the rows in the K matrix, and the dimensions of W^V is $(n \times d_v)$, where d_v is the length of the rows in the V matrix. The Q matrix also has its associated variable that indicates the length of the rows in it, d_q. However, the value of d_q has to be equal to d_k. Thus, the dimensions of Q and K is $(T \times d_k)$, while for V it is $(T \times d_v)$. These operations are represented by (2.14), (2.15), and (2.16).

$$Q = XW^Q \tag{2.14}$$

$$K = XW^K \tag{2.15}$$

$$V = XW^V \tag{2.16}$$

These matrices are used to compute similarity scores by firstly multiplying Q times K^T. This multiplication results in a matrix QK^T of dimensions $(T \times T)$ that indicates the similarity between all tokens in the sentence. In the QK^T matrix, the value of the first element from the first row represents the similarity of the first token with itself, the second element from the first row represents the similarity of the first token with the second token, the third element from the first row represents the similarity of the first token with the third token, and so on until the last token is reached. The same goes for the rest of the rows (similarity between the second token and first token, second token and second token, etc.). A higher value of any element indicates greater similarity between the tokens.

Next, the QK^T matrix is scaled by the scaling factor $\frac{1}{\sqrt{d_k}}$. This is done to improve performance in cases where there are larger values of d_k, should one chose to use them.

The scaled similarity scores are then normalized by applying the softmax function to the matrix. In doing so, all values in the matrix are brought to the range [0, 1] with the sum of all values in a row equalling to 1, and the similarity score of each token can be expressed as percentages that interpret these scores as a measure of relative importance.

The final step is multiplying the normalized similarity scores by the value matrix, V. This produces an attention matrix At for the sentence with dimensions $(T \times d_v)$. Intuitively, it can be interpreted as building a new representation for each token in s that captures the influence of all the other words in the sentence based on their relevance as measured by the similarity scores. The entire process up until now can be summarized by the (2.17).

$$At = \text{softmax}\left(\frac{QK^T}{d_k}\right)V \tag{2.17}$$

A single instance of this process is called an attention head, and while the representation given by At could be used as a document embedding matrix for s, the transformer model employs $h_{attention}$ amount of instances of the self-attention mechanism per sentence, each computing its own values for W^Q, W^K, W^V and consequently, Q, K, and V. This was denominated as the **multi-head attention mechanism**. All outputs from each head are concatenated and multiplied by another weight matrix W^O of dimensions $(h_{attention}d_v \times n)$, resulting in a final document embedding matrix O for s of dimensions $(T \times n)$. This final operation is represented in (2.18).

$$O = \text{concatenate}\left(At_1, At_2, ..., At_i, ...At_{h_{attention}}\right)W^O \tag{2.18}$$

The multi-head attention mechanism allows the model to learn different representations based on the different types of relevance that each attention head could observe. It is worth pointing out that the original transformer model used $h_{attention} = 8$ instances of the self-attention mechanism per-multi-head mechanism, as well as $n = 512$ and $d_k = d_v = n/h_{attention} = 64$.

The Transformer Model Architecture

As previously stated, the self attention mechanism is the core of the transformer model. It is, however, only one of the components in the overall architecture used by transformers. Figure 2.4 shows a simplified depiction of the transformer architecture and the rest of the components of which it is made up of. As can be observed in the figure, the transformer consists of an encoder-decoder structure in which the encoder has $E_{transformer}$ amount of encoder layers and the decoder has $D_{transformer}$ amount of decoder layers. The original transformer implementation used $E_{transformer} = D_{transformer} = 6$. Each layer of the encoder has a multi-head attention sublayer and a feedforward network sublayer. The decoder also has the multi-head attention and a feedforward network sublayers, as well as an additional masked multi-head attention sublayer. The encoder takes an input sentence and creates a vector representation of said sentence. The decoder generates the target sentence sequentially based on the output from the encoder and the output from the decoder after each time step in the text generation sequence.

The function of each of the components is briefly descried as follows:

(a) Positional Encoding: Creates a matrix representation of the input embedding based on the position of each token in the sentence and adds it to the initial embedding fed to the encoder or decoder. This way, the model is able to capture information regarding the order of the tokens in the sentence. The positional encoding occurs before the input embedding is fed to both the encoder and decoder.

(b) Multi-Head Attention: Generates a sentence representation from the input embedding using the multi-head attention mechanism described in Section 2.6.1. In the encoder, the first layer takes the position encoded embedding as input and subsequent layers take the output of the previous layer. In the decoder, the representation from the last layer of the encoder is used as input to all decoder layers as the K and V matrices, as well as the representation generated from the masked multi-head attention component in each layer as the Q matrix.

(b') Masked Multi-Head Attention: Functions similarly to the multi-head attention

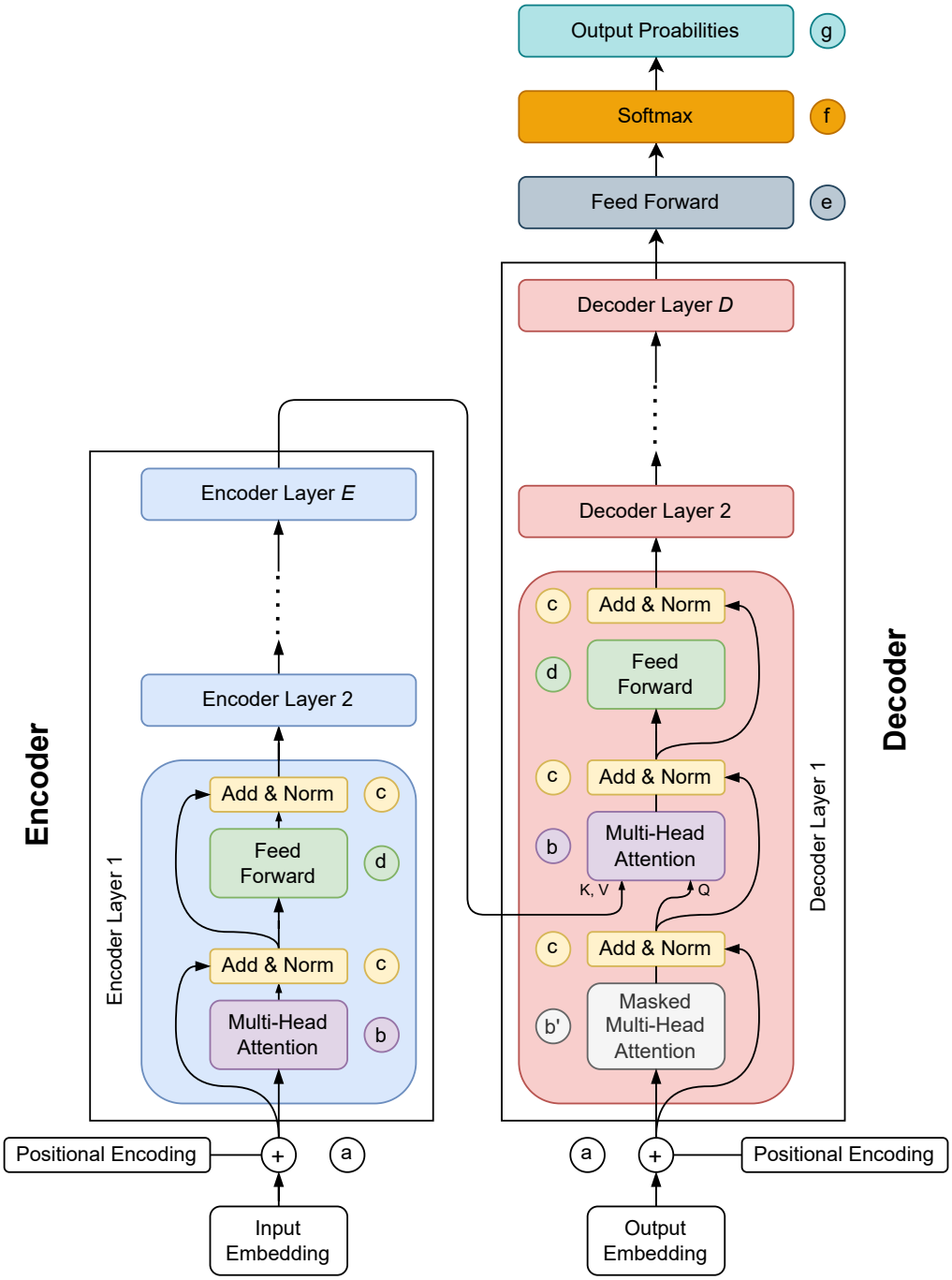

Figure 2.4: Transformer Model Architecture

layer from (b), but prevents the self attention mechanism from computing relationships between tokens that have not been generated yet by setting them to zero. This was implemented mainly for training the model, as the ground truth is already known and providing it to the model for training is considered cheating. The input embedding is shifted to the right by prepending a start of sentence token to the target sentence during training and initiating the target sentence with the start of sentence token during inference. The model predicts the next token for the first time step and feeds it back to the decoder along with the previous token during the second time step, and so on.

(c) Add & Normalization: These components are present after every sublayer and adds the input from the sublayer to its output. This operation is known as a residual connection. After the performing the residual connection, the output is normalized, preventing values from changing too much. These operations are performed mainly for numerical stability.

(d) Inner Feed Forward Networks: Simple fully connected feed forward network at the end of every encoder and decoder layer with an input, middle and output layers. The input and output layers are of size n while the middle layer is of size $4n$ and uses a ReLU activation function. The structure of each network is identical in every component instance, but their associated parameters are different.

(e) Final Feed Forward Network: Takes the output embedding from the final decoder layer and passes through a linear dense to convert it to a vector with values over each index position from the target language vocabulary. This means that the resulting output is a vector of the size of the pre-established target language vocabulary. This output vector is called a logits vector.

(f) Softmax: Turns the logits vector into a probability distribution.

(g) Output Probabilities: The probabilities for each token in the vocabulary to be the next token in the sequence. The one with the highest probability is chosen as output

for the current time step and is fed back to the decoder in (b') until an end of sentence token is chosen, at which point the algorithm stops.

As previously mentioned, the transformer was originally implemented as a language translation model. It was trained on the WMT 2014 English-German dataset and on the WMT English-French dataset, using a vocabulary size of approximately 25000 tokens for each language.

2.6.2. The BERT Implementation

The BERT model is essentially the encoder section from the transformer. Its architecture components are almost the same as that of the original encoder, with there being changes to the amount of encoder layers $E_{transformer}$, embedding size n, and self-attention heads used $h_{attention}$. Two different sized models were implemented, a **BASE** model ($E_{transformer} = 12$, $n = 768$, $h_{attention} = 12$) and a **LARGE** model ($E_{transformer} = 24$, $n = 1024$, $h_{attention} = 16$). Apart from the changes in size to the architecture components, the sequences of text fed to BERT during training were modified to be able to represent a single sentence or a pair of sentences in order to be able to handle the tasks for which it was trained. This consisted of prepending a classification token at the beginning of each sequence, inserting a separation token between pairs of sentences in a sequence, and adding an embedding to every token that indicated if said token belongs to the first sentence or the second sentence. BERT was trained on the BooksCorpus and English Wikipedia using a Masked Language Model (MLM) task and a Next Sentence Prediction (NSP) task, unlike the original transformer trained on a text translation task.

The MLM task consists of selecting a certain percentage of the input tokens randomly (15 % in the original paper), where 80 % of the time a selected token is replaced with a mask token, 10 % of the time it's replaced by a random token, and the remaining 10 % of the time it's left as is. This implementation of the MLM along with the self-attention mechanism from the transformer architecture is what makes BERT a ***bidirectional*** model.

The NSP task is essentially a classification task. It generates a sequence composed of two sentences, s_1 and s_2, where 50 % of the time the sentence s_2 is actually a sentence that

follows s_1 and 50 % of the time it is not, instead being another random sentence from the corpus. The final vector representation of the classification token prepended at the beginning of the sequence is used as the entire document representation for this task.

2.6.3. Variations of the BERT Model

As BERT was originally trained on an English corpus, it is only natural that it does not perform well on tasks that use different languages. After the first implementation of BERT, different versions have been developed that were trained on multilingual datasets as well as different monolingual datasets. Seeing as the tasks that were to be performed in this work used datasets in the Spanish, Tamil, and Malayalam languages, it was necessary to find versions of BERT that supported them. The versions used are listed as follows:

BETO - Trained on Spanish Wikipedia and documents from the OPUS Project that had text in Spanish. Its architecture is identical to the original BERT **BASE**, but its training process consisted of modifying the MLM task by using Dynamic Masking as well as Whole-Word Masking, and removing the NSP task entirely. Dynamic Masking modifies MLM by changing the tokens that are masked during training after every epoch, while Whole-Word Masking masks the entire word of a selected token if it is detected as part of said word. This version of BERT was selected to be used for experimentation in the Rest-Mex 2023 sentiment analysis task [Cañete23].

Multilingual BERT - Trained on the top 104 languages from Wikipedia in terms of text content, including Tamil and Malayalam. Its architecture and training process is identical to that of the original BERT **BASE**. This version of BERT was selected to be used for experimentation in the FIRE-2024 sarcasm detection task [6].

XLM-RoBERTa Base - Similar to Multilingual BERT, it is a multilingual model trained on 100 languages, including Tamil and Malayalam, but sourcing its data from Common Crawl instead of Wikipedia to have a larger dataset size per language. Its architecture is identical to the original BERT **BASE**, though it was trained only using the MLM task.

[6] https://huggingface.co/google-bert/bert-base-multilingual-cased

This version of BERT was selected to be used for experimentation in the FIRE-2024 sarcasm detection task [Conneau20].

MuRIL Large - Another multilingual model, differentiating itself by being trained for a variety of Indian languages, including Tamil and Malayalam. The model was also trained on the English language, as samples of Indian languages are often found code-mixed with English. Two training tasks were used for training: MLM using monolingual data, and Translation Language Modelling (TLM) using translated and transliterated data. The monolingual data was sourced from the Common Crawl OSCAR corpus and Wikipedia, while the translated data was sourced from the PMINDIA dataset as well as translations from the monolingual dataset. The transliterated data was sourced from the Dakshina Dataset as well as the transliterated Wikipedia dataset. The TLM task consisted of concatenating sentence pairs (source sentence and target sentence) as inputs, randomly masking tokens from both sentences and predicting the masked tokens. MuRIL Large's architecture is identical to that of the **LARGE** BERT model. This version of BERT was selected to be used for experimentation in the FIRE-2024 sarcasm detection task [Khanuja21a].

l3cube-pune/tamil-bert - A fine-tuned version of previous multilingual BERT models, such as MuRIL, XLM-RoBERTa, and IndicBERT. Training details are not provided by the creators. This version of BERT was selected to be used for experimentation in the FIRE-2024 sarcasm detection task [Joshi23].

l3cube-pune/malayalam-bert - Another fine-tuned version of multilingual BERT models by the same creators of "l3cube-pune/tamil-bert". Training details are not provided by the creators. This version of BERT was selected to be used for experimentation in the FIRE-2024 sarcasm detection task [Joshi23].

2.7. Conclusions of Methods Chapter

This chapter covers the theory and methodologies used as the base for this research, structured as follows:

- **Dataset Analysis**:
  - **Rest-Mex 2023**: A multilingual dataset of TripAdvisor reviews (2002–2022) labelled for **polarity** (1–5 stars), **country**, and **place type**. Severe class imbalance in polarity (62.4 % positive vs. 2.3 % negative) necessitated custom evaluation metrics (2.6).
  - **DravidianCodeMix@FIRE-2024**: Code-mixed Tamil and Malayalam YouTube comments for sarcasm detection, with intentional class imbalance (80 % non-sarcastic for Malayalam, 70 % for Tamil).
- **Evaluation Metrics**:
  - Standard metrics (accuracy, precision, recall, F1, Macro F1) were complemented by Rest-Mex's weighted F1 (2.6) and composite Sentiment score (2.9), prioritizing minority classes.
- **Text Representation**:
  - Tokenization via word/character n-grams and skip-grams.
  - Vectorization using BoW, TF-IDF, and embeddings (fastText, BERT).
- **Text Classification Libraries**:
  - μ**TC**: A combinatorial framework optimizing 45 million+ configurations via meta-heuristics (Random Search, Hill Climbing) and SVM classification.
  - **fastText**: Neural classifier leveraging sub-word embeddings with hyperparameter optimization.
  - **BERT**: Transformer-based models (BETO, MuRIL, XLM-RoBERTa, etc.) fine-tuned for Spanish and Dravidian code-mixed tasks.

Chapter 3

Experimentation

3.1. Experimental Setup

The main idea for the experiments is to develop a classification system based on ensembles of stacked outputs. The system is to be evaluated by participating in the Rest-Mex 2023 and FIRE 2024 sentiment analysis competitions. In broad terms, the system begins by preprocessing text data, after which multiple classification models are trained. For the complete dataset approach, predictions from these individual models, obtained from training, validation, and test sets, are combined to form a new feature space, which is then fed into an XGBoost classifier for final predictions. In the subset-based approach, the training data is split into multiple subsets, each used to train a separate instance of each model; their outputs are averaged for evaluation and similarly combined into a feature space for XGBoost. The dimensional structure of the XGBoost input remains consistent across both approaches, with flexibility in the number and type of models used.

3.1.1. Hardware and Software Specifications

Initial experiments were run on a computer with an Intel(R) Xeon(R) W-2125 CPU @ 4.00GHz, using 32 GB of RAM. For later experiments, the use of a GPU cluster was made available. This cluster operated with 2 Intel(R) Xeon(R) Gold 6330 CPUs @ 3.1GHz using 1007.33 GB of RAM and 3 NVIDIA GA100 [A100 PCIe 80GB] GPUs running driver version

525.116.04.

As for software used, all experiments were done using Python 3.11.2. The libraries used were standard Python libraries, such as NumPy 1.24.3 and Pandas 2.0.3, as well as scikit-learn 1.3.0, PyTorch 2.0.1, Transformers 4.35.0, microTC 2.4.9, and fastText 0.9.2.

3.1.2. Rest-Mex Datasets Pre-processing

Both DS_{RM_Train} and DS_{RM_Test} contained a "Title" and "Review" columns; for the experiments, these columns were merged to build a "text" column, as the title of a review often contains some sort of sentiment behind it. For example, a review with a positive sentiment could have as a title "Extremely satisfied with my experience!", while a review with negative sentiment could have a title like "Never coming back again!". The document in the "text" column was used to conduct the experiments involving the evaluation of the classification models. For the use of μTC, no further pre-processing was used, as common techniques are already applied and optimized by the framework. For the BERT-based model, only lowercasing was applied for compatibility with the model. In the case of fastText, all text was normalized by removing non-alphanumeric characters, replacing multiple spaces with a single space, stripping leading or trailing spaces, and converting the text to lowercase, per the recommendation from their documentation. This pre-processing was applied to all Rest-Mex datasets.

The original training dataset DS_{RM_Train} of 251702 documents needed to be split into a smaller training set and a validation set. The new training set would consist of 80 % of the samples from the original training set, leaving 20 % for the validation set. This would result in a training set with 201361 samples and a validation set with 50341 samples. However, the intent was to keep the same percentage of samples for each class across all the different types of classification tasks that were going to be performed. Thus, before splitting DS_{RM_Train}, a new "klass" label was created as the cross-product of the three sets of the provided categories (i.e., Country, Polarity, and Type). For example, given a random sample with a polarity label of "5", a country label of "Mexico" and a type label of "Hotel", the resulting corresponding label would be "5 | Mexico | Type". As there are five possible polarity labels, three possible country labels, and three possible type labels, the number of

possible labels is $5 \times 3 \times 3$, which equals to 45. The resulting amount of instances of each new class label as well as its corresponding percentage distribution in the original training dataset, before splitting, is presented in Table 3.1. The percentages have been rounded to four decimal places.

The creation of the ”klass“ label was solely intended to enable splitting the original dataset into training and validation sets that maintain the same class distribution for the three classification tasks in each new set, not for performing a classification task on the resulting 45 classes of the ”klass“ label.

To maintain a percentage distribution of classes between the training and validation sets roughly equal to that of DS_{RM_Train}, a stratified split was applied to it, the results of which can be appreciated in the Figure 3.1, where the new training set is defined as DS_{RM_TrainT} and the validation set as DS_{RM_TrainV}.

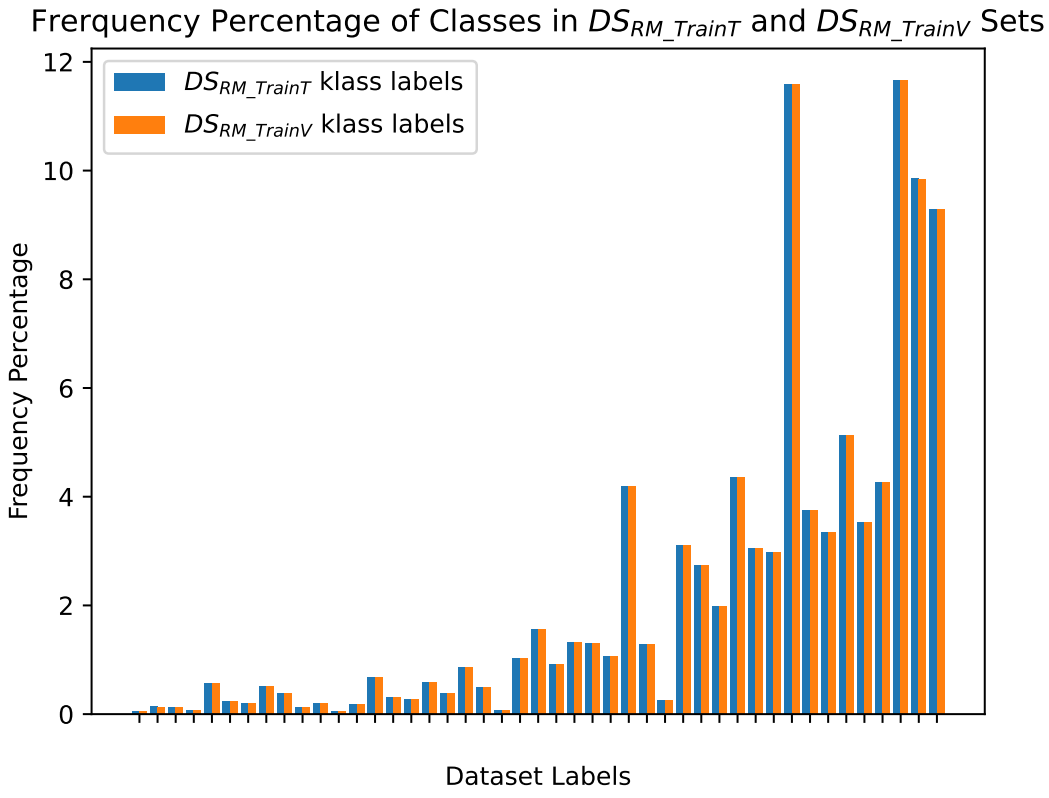

Figure 3.1: Frequency Percentage of Classes in DS_{RM_TrainT} and DS_{RM_TrainV}

Table 3.1: Frequency for the classes in `klass` in DS_{RM_Train}

klass	**Frequency**	**Frequency Percentage (%)**
5 | Mexico | Attractive	29353	11.6618
5 | Colombia | Attractive	29175	11.5911
5 | Mexico | Hotel	24787	9.8478
5 | Mexico | Restaurant	23378	9.2879
5 | Cuba | Attractive	12929	5.1366
4 | Mexico | Attractive	10979	4.3619
5 | Cuba | Restaurant	10750	4.2709
4 | Colombia | Attractive	10542	4.1883
5 | Colombia | Hotel	9428	3.7457
5 | Cuba | Hotel	8891	3.5324
5 | Colombia | Restaurant	8404	3.3389
4 | Cuba | Attractive	7832	3.1116
4 | Mexico | Hotel	7654	3.0409
4 | Mexico | Restaurant	7485	2.9738
4 | Cuba | Hotel	6908	2.7445
4 | Cuba | Restaurant	4979	1.9781
3 | Cuba | Hotel	3904	1.5510
3 | Mexico | Attractive	3333	1.3242
3 | Mexico | Hotel	3297	1.3099
4 | Colombia | Hotel	3218	1.2785
3 | Mexico | Restaurant	2654	1.0544
3 | Cuba | Attractive	2602	1.0338
3 | Cuba | Restaurant	2304	0.9154
3 | Colombia | Attractive	2148	0.8534
2 | Cuba | Hotel	1682	0.6683
2 | Mexico | Hotel	1455	0.5781
1 | Cuba | Hotel	1444	0.5737
1 | Mexico | Hotel	1292	0.5133
3 | Colombia | Hotel	1248	0.4958
1 | Mexico | Restaurant	960	0.3814
2 | Mexico | Restaurant	959	0.3810
2 | Cuba | Restaurant	775	0.3079
2 | Mexico | Attractive	706	0.2805
4 | Colombia | Restaurant	630	0.2503
1 | Cuba | Restaurant	598	0.2376
2 | Colombia | Hotel	497	0.1975
1 | Mexico | Attractive	484	0.1923
2 | Cuba | Attractive	430	0.1708
1 | Colombia | Hotel	337	0.1339
2 | Colombia | Attractive	333	0.1323
1 | Colombia | Restaurant	315	0.1251
1 | Cuba | Attractive	195	0.0775
3 | Colombia | Restaurant	166	0.0660
1 | Colombia | Attractive	147	0.0584
2 | Colombia | Restaurant	115	0.0457

While ideally this would have been the only initial pre-processing performed on DS_{RM_Train}, during the first run of experimentation, computing resources were limited to the use of a system with an Intel(R) Xeon(R) W-2125 CPU @ 4.00GHz, with 32 GB of RAM, as mentioned in Section 3.1.1. This initial system was unable to process DS_{RM_TrainT} all at once, so DS_{RM_TrainT} was divided into 20 smaller units of training sets, as suggested by my thesis advisor. The split into smaller subsets was done using the same logic as splitting DS_{RM_Train} into DS_{RM_TrainT} and DS_{RM_TrainV}, which is to say, using stratified split over the "klass label", as described in [Cerda-Flores23].

Embracing this subset-based approach, further experiments were conducted once a more powerful computing cluster became available. These experiments aimed to measure the performance of the proposed classification system based on different numbers of subsets into which DS_{RM_TrainT} was split. To do so, it was necessary to define the amount of subsets each experiment would have: $n_e = [2, 4, 8, 16, 20, 32]$. And so, besides evaluating the classification system using DS_{RM_Train} with the cluster system, the subset approach was tested with these defined values of n_e.

It was also necessary to encode all labels from each task for the classification models to work. The polarity labels, although already numerical, were also encoded for consistency. The resulting encoding is depicted in Table 3.2.

3.1.3. FIRE Datasets Pre-processing

A variety of pre-processing techniques were applied to the datasets from the FIRE competition, depending on the model that was to be used. For the use of certain BERT-based models, a series of transliteration functions were applied to convert the code-mixed text in the samples to either Dravidian script or to Latin script. These functions use the `indic_transliteration` library for script conversion and the `langdetect` library to detect the language of the input text. These functions iterate through each string in the input list of strings and checks if the text is already in the target script using `langdetect`. If it is in the target script, it adds the text to an output list. If it isn't in the target script, it is transliterated using `indic_transliteration` and added to the output list. Once it finishes iterating over the list of strings, it returns the output list of text in the target script.

Table 3.2: Label Encoding for Polarity, Country, and Type Categories

Original Label	Encoded Label	Category
1	0	Polarity
2	1	Polarity
3	2	Polarity
4	3	Polarity
5	4	Polarity
Colombia	0	Country
Cuba	1	Country
Mexico	2	Country
Attractive	0	Type
Hotel	1	Type
Restaurant	2	Type

For the use of other BERT-based models as well as fastText, the samples in the datasets were normalized via a function that converted all text to lowercase, standardizing whitespace, removing specific punctuation and symbols from the set [;:@\-"'/()[]¿¡{}~<>#$%^&*|], translating emojis and emoticons into text descriptions, eliminating URLs and substituting usernames with placeholders. This function shall be referred to as $Norm()$. When evaluating the performance of a given model using either the validation or test sets, the same procedures that were performed on the training set were also applied to the set that was being evaluated. Table 3.3 shows the pre-processing procedures applied to each model. It is important to note that while the models were trained using pre-processed datasets, some experiments were also carried out without prior manipulation of the original data.

As was the case with the Rest-Mex dataset, the labels from each task were also encoded. The resulting encoding is depicted in Table 3.4.

Table 3.3: Preprocessing Performed on Different Models

Model Name	Target Language	Preprocessing
fastText	Malayalam	$Norm()$
fastText	Tamil	$Norm()$
Multilingual BERT	Malayalam	$Norm()$
Multilingual BERT	Tamil	$Norm()$
MuRIL Large	Malayalam	$Norm()$
MuRIL Large	Tamil	$Norm()$
l3cube-pune/malayalam-bert	Malayalam	Text to Dravidian Script
l3cube-pune/tamil-bert	Tamil	Text to Dravidian Script
XLM-RoBERTa Base	Malayalam	Text to Latin Script
XLM-RoBERTa Base	Tamil	Text to Dravidian Script

3.2. System Pipeline

3.2.1. Complete Dataset Approach

The classification pipeline for both competitions generally involved three main stages: preprocessing the text data, training individual models, and combining the outputs of individual models for the final classification using XGBoost. The established text pre-processing procedures were applied to all samples in the training, validation, and test datasets. This was followed by training the selected classification models that are part of the set $LM =$

Table 3.4: Label Encoding for Malayalam and Tamil Tasks

Original Label	Encoded Label	Language
Non-sarcastic	0	Malayalam
Sarcastic	1	Malayalam
Non-sarcastic	0	Tamil
Sarcastic	1	Tamil

[μTC, fastText, BETO, Multilingual BERT, MuRIL Large, l3cube-pune/malayalam-bert, l3cube-pune/tamil-bert, XLM-RoBERTa Base].

Some of the BERT-based models were used specifically for extracting document embeddings instead of employing their built-in classification pipelines. This was done by averaging the token embeddings from the models' final layers to generate document embeddings. These embeddings were created for all training, validation, and test samples. The document embeddings from the training datasets were then used to train a Logistic Regression classifier.

Next, the individual models were used to obtain the predictions on the validation set. These predictions were evaluated using the desired scoring metrics to assess the performance of each model. The training and validation sets were then fed to each model to obtain decision functions or prediction probabilities that were stacked horizontally to create the new feature space. This combined feature space was fed into an XGBoost classifier for training and validation, as well as for generating predictions on the test set.

The dimensions of the input data supplied to the XGBoost model is determined by $(N \times (C \times n_s) \times n_{LM})$ transformed into a two-dimensional array $(N, (C \times n_s \times n_{LM}))$, where N is the size of the dataset, C is the number of classes in the task, n_s is the number of training subsets (set to 1 for the complete dataset approach), and n_{LM} is the number of classification models used from LM. Throughout these experiments, n_{LM} is limited to the set $[2, 3, 4, 5]$. It is important to note that in the case of the μTC models, when dealing with binary classification ($C = 2$) the value taken by $C = C - 1$, as the SVM classifier that it uses outputs a single value decision function, making the said value positive for the positive class and negative for the negative one. This behaviour is only present in the FIRE classification tasks.

Figure 3.2 illustrates a flowchart of the previously described steps. In this figure, DBX_{t_t} is any training dataset and DBX_{t_v} is any validation dataset. The training and evaluation phase of the individual models is shown by the flow in the black line, the creation of the training vector space is depicted by the flow in the purple line, and the creation of the validation vector space and its evaluation is depicted by the flow in the red line.

It is important to point out that this pipeline is run for every classification task that is to be performed per model and XGBoost ensemble; three for each sub-task in the

Rest-Mex competition (polarity, country and type) and two for each language in the FIRE competition (Malayalam and Tamil).

To better illustrate the pipeline procedures, consider the following example from the FIRE competition, in which the objective is to classify code-mixed comments in Malayalam as "Non-sarcastic" and "Sarcastic", a binary classification task ($C = 2$). The selected language models will be μTC, fastText, and MuRIL ($n_{LM} = 3$). The training set $DS_{FIRE\_TrainMal}$ consists of 13,188 comments ($N_t = 13188$), of which 10,689 are labelled as "Non-sarcastic" and 2499 are labelled as "Sarcastic". The validation set DS_{FIRE_ValMal} consists of 2826 comments ($N_v = 2826$), where 2305 are "Non-sarcastic" and 521 are "Sarcastic". The entire dataset will be processed in one go ($n_s = 1$).

Firstly, the flow of the black line from Figure 3.2 shall be explained. Table 3.5 contains the first 5 samples from $DS_{FIRE\_TrainMal}$. As can be observed, the text samples contain documents in a mix of different scripts.

Table 3.5: Sample of Malayalam Training Database

	Text	labels
0	Screenshot edukkan vannth njan മാത്രമാണോ	Sarcastic
1	നമ്മുടെ അനു സിത്താര ചേച്ചി ഇങ്ങനെ വരുന്നത് നോക...	Sarcastic
2	Mollyhood is getting bigger and bigger	Non-sarcastic
3	Ho aaa BGM. Mammookka ithu oru pwoli pwolikkum	Non-sarcastic
4	Enthaale, sambhavam puraanam aanelum backgroun...	Sarcastic

As stated in Table 3.3, the text used to train the fastText and MuRIL classifiers is preprocessed using the $Norm()$ function. Applying it on the first 5 samples from $DS_{FIRE\_TrainMal}$ shown in Table 3.5 results leads to Table 3.6. μTC does not require text preprocessing in this instance as it is part of its classification pipeline.

Table 3.6: Sample of Preprocessed Malayalam Training Database using $Norm()$ Function

	Text	labels
0	screenshot edukkan vannth njan മാത്രമാണോ	Sarcastic
1	നമ്മുടെ അനു സിത്താര ചേച്ചി ഇങ്ങനെ വരുന്നത് നോക...	Sarcastic
2	mollyhood is getting bigger and bigger	Non-sarcastic
3	ho aaa bgm. mammookka ithu oru pwoli pwolikkum	Non-sarcastic
4	enthaale, sambhavam puraanam aanelum backgroun...	Sarcastic

In this example, MuRIL will be used to obtain document embeddings that will be fed as feature vectors for a Logistic Regression Classifier. Extracting the embeddings of the samples from Table 3.6 obtained trhough MuRIL and normalizing them for later use in the Logistic Regression classifier results in the following 768 dimensional arrays:

$$x_{s1} = \begin{bmatrix} 0.49215677 & 0.31914943 & 0.7004523 & \dots & 0.47872964 & 0.66571367 & 0.36946923 \end{bmatrix}$$

$$x_{s2} = \begin{bmatrix} 0.5781543 & 0.5844143 & 0.5286694 & \dots & 0.5683198 & 0.5905484 & 0.41493526 \end{bmatrix}$$

$$x_{s3} = \begin{bmatrix} 0.1357696 & 0.64907116 & 0.57666117 & \dots & 0.6290546 & 0.32759428 & 0.6495048 \end{bmatrix}$$

$$x_{s4} = \begin{bmatrix} 0.6124053 & 0.36739635 & 0.5071091 & \dots & 0.47206974 & 0.58201194 & 0.50332314 \end{bmatrix}$$

$$x_{s5} = \begin{bmatrix} 0.62504905 & 0.18483686 & 0.48188287 & \dots & 0.54516774 & 0.7880733 & 0.49096254 \end{bmatrix}$$

Now, the individual models can be trained, using the normalized embeddings for the Logsitic Regression classifier, the preprocessed text for the fastText classifier, and the text as is for the μTC classifier. The resulting models are then used to get predictions on the validation set DS_{FIRE_ValMal}, which is preprocessed according to the already mentioned requirements for each model. Table shows the first 5 documents from DS_{FIRE_ValMal} with

their real label, as well as the predicted class obtained by each model. Using these predicted labels, it is possible to evaluate the performance of each individual model, ending the process of the black line from the pipeline.

Table 3.7: Example of Predicted Classes of DS_{FIRE_ValMal} from Individual Models

	Text	labels	microTC	fastText	MuRIL
0	Ee parnjel pakhuthi scene pollum cinemel illalo	Non-sarcastic	Non-sarcastic	Non-sarcastic	Sarcastic
1	ആമയെ വരെ വെള്ളത്തിൽ മുക്കി കൊല്ലുന്ന ഇനമാ	Non-sarcastic	Non-sarcastic	Sarcastic	Non-sarcastic
2	ഈ ഓണം ഇട്ടിമാണി എടുത്തു കഴിഞ്ഞു. ഇനി 6ആം തീയതി...	Non-sarcastic	Non-sarcastic	Non-sarcastic	Non-sarcastic
3	Rajappanod pokan para....njangalkku njangade s...	Sarcastic	Sarcastic	Non-sarcastic	Sarcastic
4	Sagarji de songs nu vendi wait cheyyunnavar aa...	Sarcastic	Sarcastic	Non-sarcastic	Non-sarcastic

The next step in the pipeline follows the purple line. The training set $DS_{FIRE\_TrainMal}$ is passed through each model. The output array of probabilites or decision functions of each model is extracted and stacked horizontally to create a new vector space which will be used to train the XGBoost classifier. Continuing with the example, the output array of the first 5 samples from $DS_{FIRE\_TrainMal}$ for each model is shown in Table 3.8. In this table a positive value of the μTC output represents the "Non-sarcastic" label and a negative value represents the "Sarcastic" label. The output arrays for fastText and MuRIL have in the first value the probability of the sample belonging to the "Non-sarcastic" class and the second value the probability of it belonging to the "Sarcastic" class.

Table 3.8: Example of $DS_{FIRE\_TrainMal}$ Output Arrays from Individual Models

Sample	μTC Output	fastText Output	MuRIL Output
1	0.44001857	[0.982103110, 0.017916884]	[0.12548175, 0.87451825]
2	0.61455542	[0.348203960, 0.65181607]	[0.2281774, 0.7718226]
3	-0.94338514	[0.998573840, 0.0014462206]	[0.66925125, 0.33074875]
4	-1.29322458	[0.999947790, 0.000072178773]	[0.93926122, 0.06073878]
5	0.41595255	[0.791002570, 0.20901746]	[0.37670634, 0.62329366]

Stacking these otuputs horizontally results in the new vector space used for training the XGBoost model, where each row in X_{train_xgb} represents a sample from $DS_{FIRE\_TrainMal}$. The flow of the purple arrow ends with the training of the XGBoost model.

$$X_{train\_xgb} = \begin{bmatrix} 0.44001857 & 0.982103110 & 0.017916884 & 0.12548175 & 0.87451825 \\ 0.61455542 & 0.348203960 & 0.65181607 & 0.2281774 & 0.7718226 \\ -0.94338514 & 0.998573840 & 0.0014462206 & 0.66925125 & 0.33074875 \\ -1.29322458 & 0.999947790 & 0.000072178773 & 0.93926122 & 0.06073878 \\ 0.41595255 & 0.791002570 & 0.20901746 & 0.37670634 & 0.62329366 \end{bmatrix}$$

This same process is followed with the flow of the red line on DS_{FIRE_ValMal} to create the validation vector space for XGBoost. The resulting example vector space X_{val_xgb} of the first 5 samples from DS_{FIRE_ValMal} would be:

$$X_{val\_xgb} = \begin{bmatrix} -0.20754098 & 0.36521533 & 0.63480467 & 0.84968164 & 0.15031836 \\ 0.31091145 & 0.37467635 & 0.62534368 & 0.97281516 & 0.02718484 \\ -1.08061816 & 0.99903703 & 0.00098301447 & 0.82064669 & 0.17935331 \\ -0.99110758 & 0.98865765 & 0.011362390 & 0.94441274 & 0.05558726 \\ -0.64707915 & 0.87161845 & 0.12840156 & 0.8415837 & 0.1584163 \end{bmatrix}$$

The last step in the validation of X_{val_xgb} would be to pass it through the trained XGBoost model to produce the final predictions on DS_{FIRE_ValMal}, which are evaluated using the metrics from Section 2.2. The predictions obtained from the test set $DS_{FIRE\_TestMal}$ follow the same process as the flow of the red line from Figure 3.2, and are sent for evaluation to the competition organizers.

These series of steps are followed for each ensemble configuration for each classification task in both competitions, with slight changes dependeing on the BERT model used.

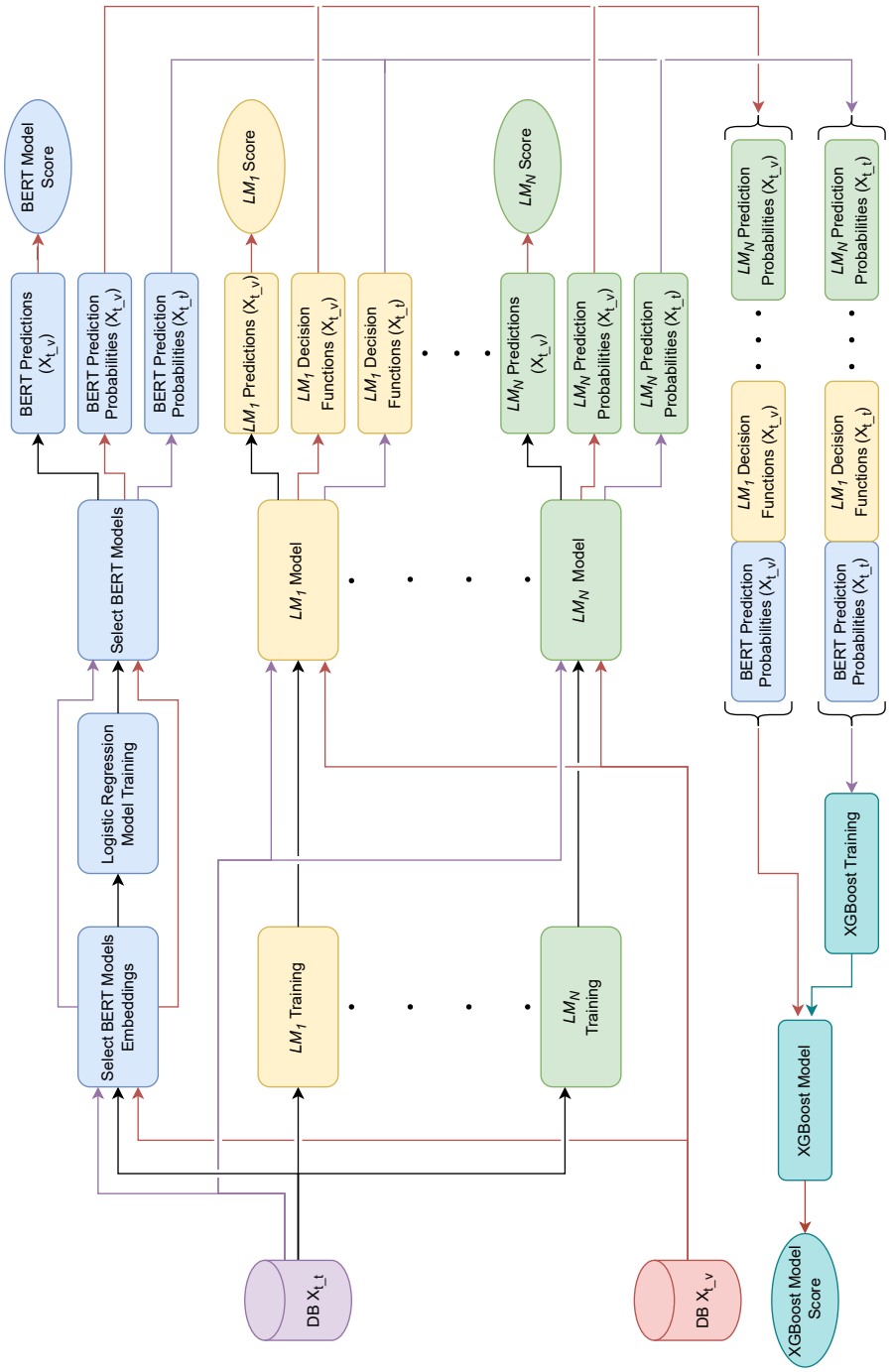

Figure 3.2: System Flowchart for Complete Dataset Approach

3.2.2. Subset-Based Approach

The subset-based approach closely follows the procedure as the one depicted in Figure 3.2. The difference lays in the splitting of the training set DBX_{t_t} into n_s subsets derived from n_e, and a separate LM_i model is trained on each of these subsets. The prediction probabilities or decision functions from all subset-trained LM_i models are then averaged to represent the performance of LM_i as a whole.

Afterwards, the complete DBX_{t_t} and DBX_{t_v} datasets were passed through each of these subset-trained models. The resulting outputs are stacked horizontally to form LM_i's contribution to the new vector space. Figure 3.3 shows a portion of this process, focusing on a single LM_i.

It can be observed that the structure of the final XGBoost classifier remains unchanged, with its input dimensions remaining as $(N, (C \times n_s \times n_{LM}))$. Another thing to note is that the selected BERT-based model used in this approach follows the same overall procedure, with the additional initial step of generating document embeddings for each dataset, as depicted in Figure 3.2.

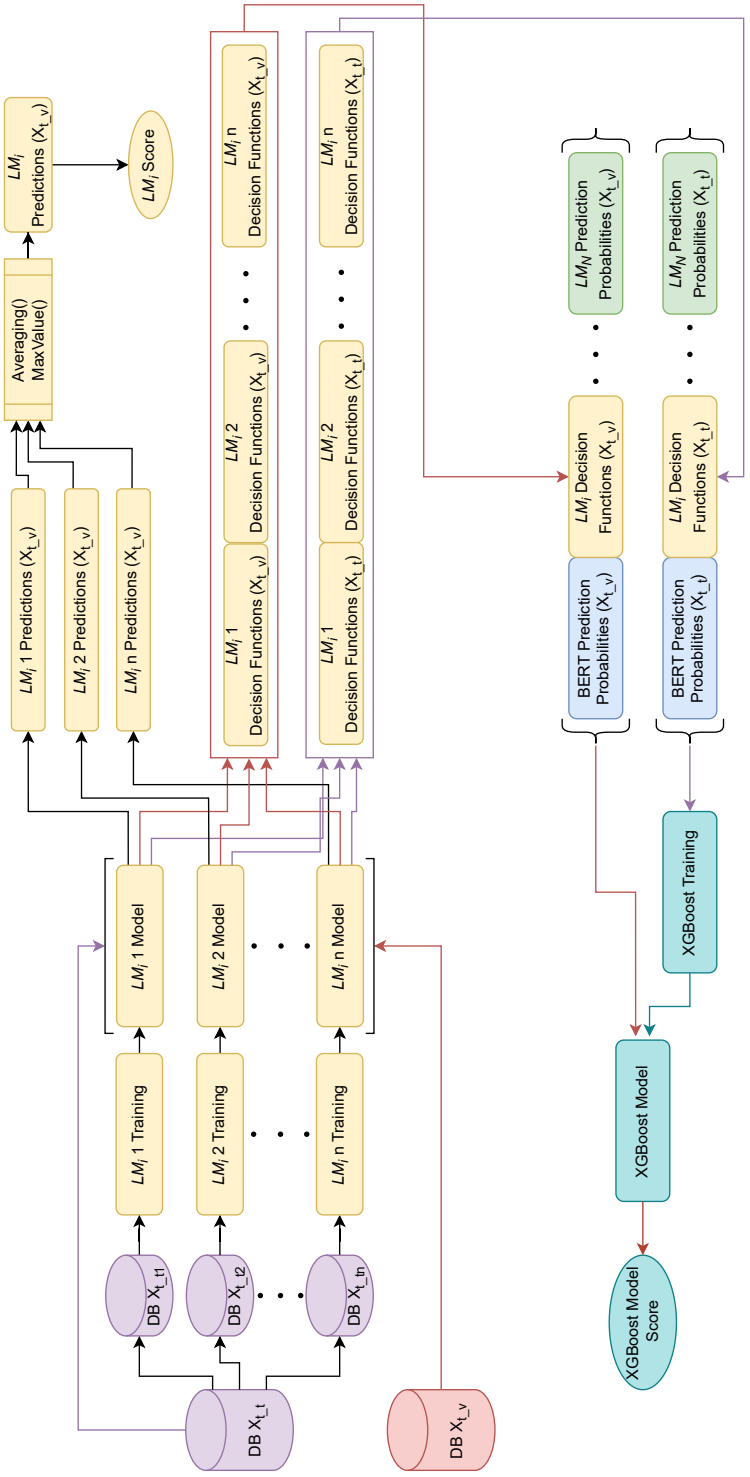

Figure 3.3: Partial System Flowchart for Subset Approach

3.3. Usage of μTC for Experimentation

All models were trained using μTC's command line interface. The configuration search phase for all models used the default 5 folds for cross-validation and Macro F1-score as the optimization metric. The models for the Rest-Mex competition were trained by sampling the default 32 points of the configuration space due to the size of the dataset, while the models for the FIRE competition were sampled in 80 points. The μTC configuration space is described in Table 2.4. μTC needed each sample to be formatted as "text" for the textual document and "klass" for its corresponding label. Other than this, no further changes to the datasets were needed to use the μTC framework.

An example of the commands used for training and testing a μTC model through the command line interface is found in listing 3.1, specifically for the Malayalam language model. The `microtc-params` command searches the configuration space with the given search parameters and places each found configuration in order from best to worst in terms of their Macro F1-score. After, the `microtc-train` takes the best configuration and trains the model. Finally, the `microtc-predict` command uses the model to obtain label prediction for a given dataset. The output of this command also includes the decision function obtained for each sample, which were used to build the XGBoost input feature vector. These commands were placed in a loop for each classification task of the Rest-Mex competition, which in turn was placed in another loop for each subset in the subset approach.

```
1 microtc-params -k5 -n80 -s80 -Smacrof1 mtc_mal_train.json -o
    mtc_mal_parameters.json
2 microtc-train -o mtc_mal_model.mtc -m mtc_mal_parameters.json mtc_mal_train.
    json
3 microtc-predict -m mtc_mal_model.mtc -o mtc_mal_val_prediciton.json
    mtc_mal_val.json
4 microtc-predict -m mtc_mal_model.mtc -o mtc_mal_train_prediciton.json
    mtc_mal_train.json
```

Listing 3.1: μTC Script Malayalam Model

In the case of the FIRE competition and the complete dataset approach of the Rest-Mex competition, the predicted labels were used as is to evaluate the generated model's

performance. The subset approach of the Rest-Mex competition followed the pipeline described in Figure 3.3, with the addition of weighing the decision functions from each model by the Macro F1-score obtained from `microtc-params` before averaging them, giving a higher emphasis to predictions from models that performed better in the training phase. The decision functions are then normalized and used to get the final prediction.

The models for FIRE were evaluated on accuracy and Macro F1-score, while the models for Rest-Mex were evaluated on the metrics described in Section 2.2.2.

3.4. fastText Model Training and Output Processing

Using fastText required more pre-processing of the datasets, as it needs a specific format for classification tasks, where each line in the training file represents a sample along with its label. The format is presented in listing 3.2:

```
__label__[label_1] [text_1]
__label__[label_2] [text_2]
__label__[label_3] [text_3]
...
__label__[label_i] [text_i]
...
__label__[label_n] [text_n]
```

Listing 3.2: fastText Training Format for Classification

In this format, [label_i] represents the label or class of the corresponding [text_i], and [text_i] represents the text example. The "__label__" prefix is required to indicate the beginning of the label. One training file was created per classification task (and for every subset in the case of the Rest-Mex datasets), as each task requires the model to focus on different aspects of the text.

Each classification task's models were trained by using fastText's automatic hyperparameter optimization, which searches for the best model parameters from the given options in 2.6. By default, this search process has a duration of 5 minutes per model, but it was changed to 10 minutes for each trained model.

The models were trained used to predict the labels of the pre-processed validation,

test and training sets (for the XGBoost feature vector), as was the case with μTC, but making sure to remove the string "__label__<label_i>" format from each sample in the training sets, leaving only normalized text.

fastText's prediction method by default returns a single label for a given sample, which is the one with the highest probability. This was changed to return all possible labels and their corresponding probabilities for a given sample in a tuple, where the first element is a list of labels of dimensions (number of samples $\times$ number of classes) and the second element is a list of probabilities of the same dimensions for each classification task. These labels and probabilities of each sample are arranged in order of most to least probable, so for example, classifying for polarity given a document d, the resulting predicted labels could be:

['__label__4', '__label__5', '__label__3', '__label__2', '__label__1']

and its corresponding probabilities could be:

[0.482686, 0.3030804, 0.1698321, 0.03413432, 0.01031715]

The lists were processed and placed in a new list arranged in ascending order, so the previous examples labels would look like:

['__label__1', '__label__2', '__label__3', '__label__4', '__label__5']

and its predictions would be:

[0.01031715, 0.03413432, 0.1698321, 0.482686, 0.3030804]

This was done to facilitate model evaluation and vector space creation. The procedure for evaluating fastText models for both competitions and both approaches was the same as with the μTC models, following the pipelines from Figures 3.2 and 3.3 and using the designated competition metrics.

3.5. BERT Models

The usage of the BERT-based models varied by classification task. As all tasks were based on languages different from the English, it was necessary to use BERT models that had been trained or fine-tuned for each language. Doing so offers several advantages due to the

language specific pre-training of the model, allowing it to capture the nuances, idiomatic expressions, and unique syntactic structures of the language more effectively than a general-purpose BERT model that is trained on multiple languages. This in turn results in improved tokenization that respects the morphology and syntax of each language, leading to better output embeddings. The selected BERT-based models are described in Section 2.6.3. All models were sourced from the Hugging Face hub using the transformers library.

3.5.1. Rest-Mex Competition

For all tasks in the Rest-Mex competition, it was decided to use a Spanish version of BERT, namely BETO, which was trained on a Spanish corpus, as the samples from the datasets were in Spanish. Furthermore, instead of using the BETO model to directly perform classification as it was available in their pipeline, it was used to create document embeddings for each sample, which were fed to a Multi-layer Perceptron (MLP) classifier for the complete dataset approach and Logistic Regression classifiers the subset approach.

For any given text document, the maximum number of tokens was set to 256. Consequently, each document produced up to 256 token embeddings, which were then transformed into a single sentence embedding. To produce the document embeddings from BERT models, the text is passed through the pre-trained model and the embeddings from all tokens in the final layer of the model (which are context aware embeddings) are averaged, resulting in a single 768-dimensional embedding for the entire document. These operations were performed on a GPU for efficiency, and were placed in a function to use on each dataset from both Rest-Mex and FIRE competitions.

Implementation of a Custom Three-Layered MLP Classifier Using PyTorch for the Complete Dataset Approach

The document embeddings were normalized and used as feature vectors for a Logistic Regression classifier. Or rather, this was the initial intention for both the complete dataset approach and the subset approach. However, training the model with the complete dataset often led to convergence errors. The algorithm repeatedly reached the maximum number of iterations without satisfying the default stopping criterion, which was a tolerance of 1^{-4}.

To avoid compromising the model's performance, the tolerance parameter was not altered. Instead, it was decided to increase the number of iterations.

The maximum iterations were first increased from 100 to 1,000. When this failed to resolve the issue, the iterations were further increased to 10,000. Using 10,000 iterations, the algorithm occasionally converged, albeit not as consistently as desired. Consequently, it seemed necessary to switch to a different classification method instead of continuing with Logistic Regression, keeping in mind that this issue only came up when training using the entire training set as a single batch.

After discussing this issue with my thesis advisor, he suggested building an MLP classifier using the neural network module from PyTorch, as it provides a collection of classes, functions, and utilities designed for creating and training neural networks without starting from scratch. It is also able to transfer the training process to the GPU for faster computation. To use this module, the processed embeddings from DS_{RM_TrainT}, DS_{RM_TrainV} and DS_{RM_Test}, as well as the encoded labels for DS_{RM_TrainT} and DS_{RM_TrainV} were converted into PyTorch tensors.

A custom three layered "MLP" class was created similar to the sklearn implementation of the MLP Classifier module, that is, using parameters that were equal to the sklearn module. An instance of this class was set to be initialized by three arguments; "input_size", which is the number of input features (i.e., the length of the input vectors, which is 768 for BERT embeddings), "hidden_size", which is the number of neurons in the hidden layer, and "output_size", which is the number of output classes for the classification task (C). It used a "classifier" model that uses a sequential container, which is a simple way to build a feed-forward neural network. The input layer is fully connected to the hidden layer, to which a ReLU activation function is applied (default for sklearn) and is then fully connected to the output layer.

A "forward" method was defined that indicates how the input data is processed through the network layers. In this method, the input "x" is passed through the "classifier" sequential model, which processes the input through the defined layers. The method is able to process an array of document embeddings as "x". This is also true for later methods that were created for the MLP class that take "x" as an input.

A "predict" method was defined that takes an input "x", designed to make class predictions for the given input data. In this method, the input data is passed through the network using the previously described "forward" method. The output of the network is a tensor of class scores, from which the index of the maximum score in each input instance is found, which corresponds to the predicted class. The resulting tensor of predicted classes is then detached from the computation graph and converted to an array, which is the output of this method.

The "predict" method is suitable for classification tasks where the goal is to predict the single most likely class of a given input, and is relevant to this work for evaluating the performance of the MLP classification model using BERT embeddings. However, another method that returned the probabilities of all classes for a given input was created, as they are necessary for building a new vector space when using XGBoost, which is the main objective of this work. The new method was defined as "predict_proba", and it operates similarly to the "predict" method up until the search for the index of maximum score. Instead of doing this, a softmax function is applied to these scores to convert them into probabilities. The tensor of class probabilities is converted to an array and returned as the result.

Finally, a "fit" method to train the model was created. It takes as input the training data, which would be the array of document embeddings, as well as their corresponding labels. It also takes as input the loss function and optimizer to be used during training, to provide flexibility in the training process. The last input that the method needs is the number of epochs for which the training should be performed.

This method first initiates an empty "error" list to store the loss values for all epochs, and can be used to monitor the training process and the model's performance. It then starts a for loop that runs for the input number of epochs. In each iteration of the loop, the gradients of the model parameters are firstly reset to zero to avoid mixing up gradients between epochs. Then, a forward pass is performed through the network using the "forward" method by passing it the training data. The output logits of this forward pass are used to compute the loss between the training data's predicted labels and its true labels using the input loss function. A backward pass is performed to compute the gradients of the loss for the current model parameters. The provided optimizer uses the gradients to update the

model parameters in the direction that minimizes the loss. The loss value for the current epoch is appended to the "error" list, which is the final output returned by the method. The loss function used for all tasks was the cross-entropy loss function, a standard choice for multi-class classification problems and the default for the sklearn implementation. Similarly, all tasks used the Adam optimizer for training and shared a learning rate of 0.001 and an L2 regularization term of 0.0001, also the default for sklearn.

Figure 3.4 depicts the architecture of the custom MLP network described in this section.

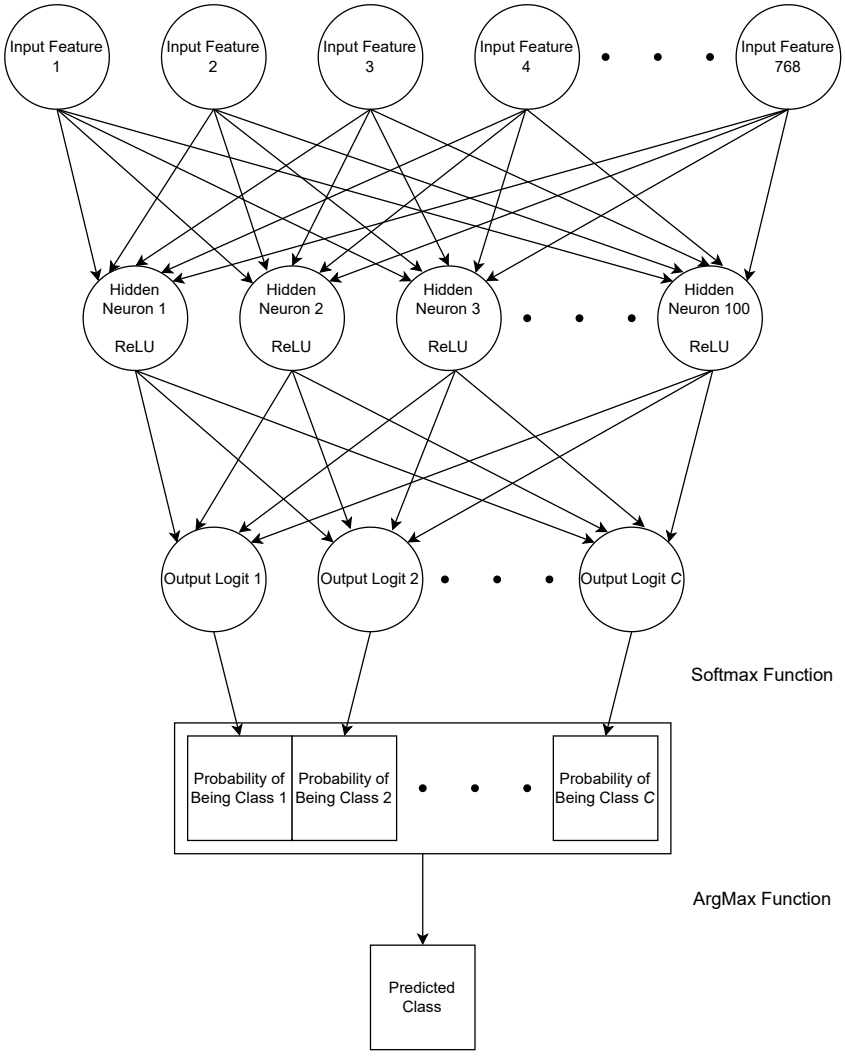

Figure 3.4: Custom PyTorch MLP Architecture

Having prepared the MLP class, it was time to use it for training, following the pipeline from Figure 3.2 to later obtain individual model performance and XGBoost vector space contributions. The subset approach followed the pipeline from 3.3, but it was possible to train Logistic Regression models per subset, as was originally intended.

3.5.2. FIRE Competition

For the tasks in the FIRE competition, the BERT models used were Multilingual BERT, MuRIL Large, l3cube-pune/malayalam-bert, 3cube-pune/tamil-bert, and XLM-RoBERTa Base. Multilingual BERT and MuRIL Large were used to extract document embeddings to fit Logistic Regression models, as was done in the Rest-Mex competition, with the addition of 5-fold stratified cross-validation and using the max amount of tokens, 512.

The rest of the BERT models were used to directly perform classification using the provided pipeline from the transformers library. This pipeline followed the standard fine-tuning procedure for BERT models, modifying the hyperparameters to 500 warm up steps, learning rate to $1e^{-5}$, and epochs to 7.

As was the case of the Rest-Mex competition, the validation and vector space creation followed the pipeline from Figure 3.2.

3.6. XGBoost

XGBoost was used to create an ensemble classifier for each task that combined the outputs from the previously used classification methods as training data and as validation data. The training and validation sets for each task were loaded, and the labels of each set were encoded according to Table 3.2 or 3.4.

To prepare the training data per set, the output decision functions (in the case of μTC) and prediction probabilities (in the case of fastText and BERT-based models) for each task were stacked into a single array. The same was done for the validation data. These outputs were used as input feature vectors for the XGBoost classifier. One classifier was trained per task and tested on the corresponding validation or test dataset.

The XGBoost classifiers trained for the Rest-Mex competition used default parame-

ters, save for the "objective" parameter, set to "multi:softmax" for multi-class classification, and "num_class" set to the appropriate amount of classes for each task. The XGBoost models trained for the FIRE competition were built using randomized parameter search cross-validation using stratified k-folds with five splits. The parameters with their respective values that were chosen for the random search are detailed in Table 3.9.

Table 3.9: Parameter Grid for Randomized Search in XGBoost

Parameter	**Values**
n_estimators	100, 600, 1000
min_child_weight	1, 5, 10
gamma	0.5, 1, 1.5, 2, 5
learning_rate	0.01, 0.05, 0.1, 0.15, 0.2
subsample	0.6, 0.8, 1.0
colsample_bytree	0.6, 0.8, 1.0
max_depth	3, 4, 5

The selected combinations of model outputs can be seen in Table 3.10. For each of these combinations, a model was trained per task.

Table 3.10: Combinations Used to Create New VSMs

Competition	**Combination**
Rest-Mex	μTC + BERT
Rest-Mex	μTC + FastText
Rest-Mex	BERT + FastText
Rest-Mex	μTC + BERT + FastText
FIRE	MuRIL + fastText
FIRE	MuRIL + fastText + Multilingual BERT
FIRE	MuRIL + fastText + μTC + bert/l3cube-pune
FIRE	MuRIL + fastText + μTC + bert/l3cube-pune + XLM-RoBERTa

The predictions for each task per combination were stacked to form a combined prediction array and processed along with the real labels from the validation datasets to make a final evaluation of each combination of classification methods using the same scoring metrics as in the case of the previous individual classification methods. Of these, the best performing models were used to get predictions on the test datasets and submitted for evaluation to the competition organizers, the results of which are discussed in Chapter 4.

3.7. Conclusions of Experimentation Chapter

This chapter covers the experimental pipeline for evaluating the proposed ensemble classification system across the Rest-Mex 2023 and DravidianCodeMix@FIRE-2024 competitions. Key components are outlined below:

- **Experimental Setup**:
  - **Hardware/Software**: Initial experiments used an Intel Xeon CPU (32GB RAM); later trials leveraged a GPU cluster (3 × NVIDIA A100 80GB VRAM GPUs, 1TB RAM). Python libraries (scikit-learn, PyTorch, Hugging Face Transformers) and frameworks (μTC, fastText, XGBoost) were standardized.
  - **Dataset Preprocessing**:
    - **Rest-Mex**: Merged "Title" and "Review" columns into a unified text field. Stratified splits preserved class balance (Table 3.1), with subsets n_e=[2,4,8,16,20,32] addressing computational constraints. Labels encoded per Table 3.2.
    - **FIRE**: Applied script transliteration (Latin/Dravidian) and normalization. Labels encoded as binary (Sarcastic/Non-sarcastic) per Table 3.4.
- **System Pipeline**:
  - **Complete Dataset Approach**: Stacked predictions from individual models (μTC, fastText, BERT variants) into feature vectors for XGBoost (Figure 3.2).
  - **Subset-Based Approach**: Split training data into n_s subsets, trained models on each, averaged outputs, and combined into XGBoost input (Figure 3.3).

- **Model Implementation**:
  - μ**TC**: Optimized via Random Search/Hill Climbing, with Macro F1-score as the objective. Decision functions weighted by validation performance.
  - **fastText**: Hyperparameter optimization (10-minute search) with label probabilities restructured for compatibility.
  - **BERT**:
    - **Rest-Mex**: BETO embeddings averaged and classified via custom MLP (PyTorch) for complete dataset and Logistic Regression for subsets.
    - **FIRE**: Multilingual BERT and MuRIL embeddings averaged and classified via Logistic Regression, XLM-RoBERTa and l3cube-pune fine-tuned (500 warm up steps, 7 epochs, learning rate $1e^{-5}$).
- **XGBoost Ensembles**:
  - Trained on stacked model outputs (Table 3.10).
  - **Rest-Mex**: Default parameters (multi:softmax objective).
  - **FIRE**: Randomized search over hyperparameters (Table 3.9) with stratified 5-fold cross-validation.

Chapter 4

Results

This chapter will have a discussion on the results obtained from experimentation on the validation and test sets for both competitions.

4.1. Rest-Mex 2023 Validation Results

Table 4.1 presents the results from evaluating the different text classification models and their ensembles on DS_{RM_TrainV} using the complete dataset for training. The columns represent the value obtained for the specified performance metric, which are defined in Section 2.2.2. The values in bold highlight the highest score for the particular metric in the column.

The results for validation using the subset approach are presented in Appendix A. A variety of performance patterns can be observed from these tables. Firstly, it can be denoted that the ensemble models generally outperform the individual models in all metrics. In particular, the **BERT + fastText** configuration emerges as a consistently strong ensemble, often achieving top or near-top results in the *Sentiment* metric and frequently also in the Res_P metric. This pattern recurs in most subset counts, with the only exception being with 2 subsets where the **μTC + fastText** configuration outperforms it in the Res_P metric.

Regarding the individual models, while μTC contributes positive results in some aspects, it doesn't seem as consistently strong on the primary *Sentiment* metric when compared to BERT or fastText. μTC particularly stands out amongst the individual models in the Res_P metric up to the use of 8 subsets, but its lower results on Res_C and Res_A greatly

impacted its overall performance.

Table 4.1: Results for 1 Set

Model / Ensemble	***Sentiment***	***Res$_P$***	***Res$_C$***	***Res$_A$***
BERT	0.64481	0.44947	0.73960	0.94070
μTC	0.66731	0.47663	0.81906	0.89691
fastText	0.70808	0.46905	0.91638	0.97783
BERT + μTC	0.69307	0.44642	0.91109	0.96837
BERT + fastText	**0.71959**	**0.49633**	0.91480	0.97090
μTC + fastText	0.70414	0.45729	**0.92339**	**0.97859**
BERT + μTC + fastText	0.70236	0.45388	0.92312	0.97855

The graphs depicted in Figure 4.1 illustrate the results of the *Sentiment*, Res_P, Res_C and Res_A metrics, respectively, for each model and ensemble as the number of subsets increases. From these, the previously mentioned patterns are more apparent; across all metrics, the **BERT + fastText** ensemble delivers the best or near-best performance in all subset scenarios. It can also be seen that it is the model with the least decrease in performance for *Sentiment* throughout subsets.

Another observation is that increasing the number of subsets past a certain amount often deteriorates performance across all models and metrics, with an occasional boost in performance. However, in general, there is a downward trend in the cases where there is an initial bump in performance.

The graphs depicted in Figure 4.2 show the performance of all metrics for each model with the increase of subsets used. These are provided mainly for clarity, as the same observations can be reached from them as from the previous graphs.

Appendix B contains graphs of the macro-average, micro-average and individual class ROC curves for all classification tasks from the RestMex competition, as this one presented the heaviest class imbalance, in particular for the polarity class. Only the curves for the fastText + BERT ensemble trained on the complete dataset were created, as this ensemble provided the best overall results for the validation set, as seen in Table 4.1.

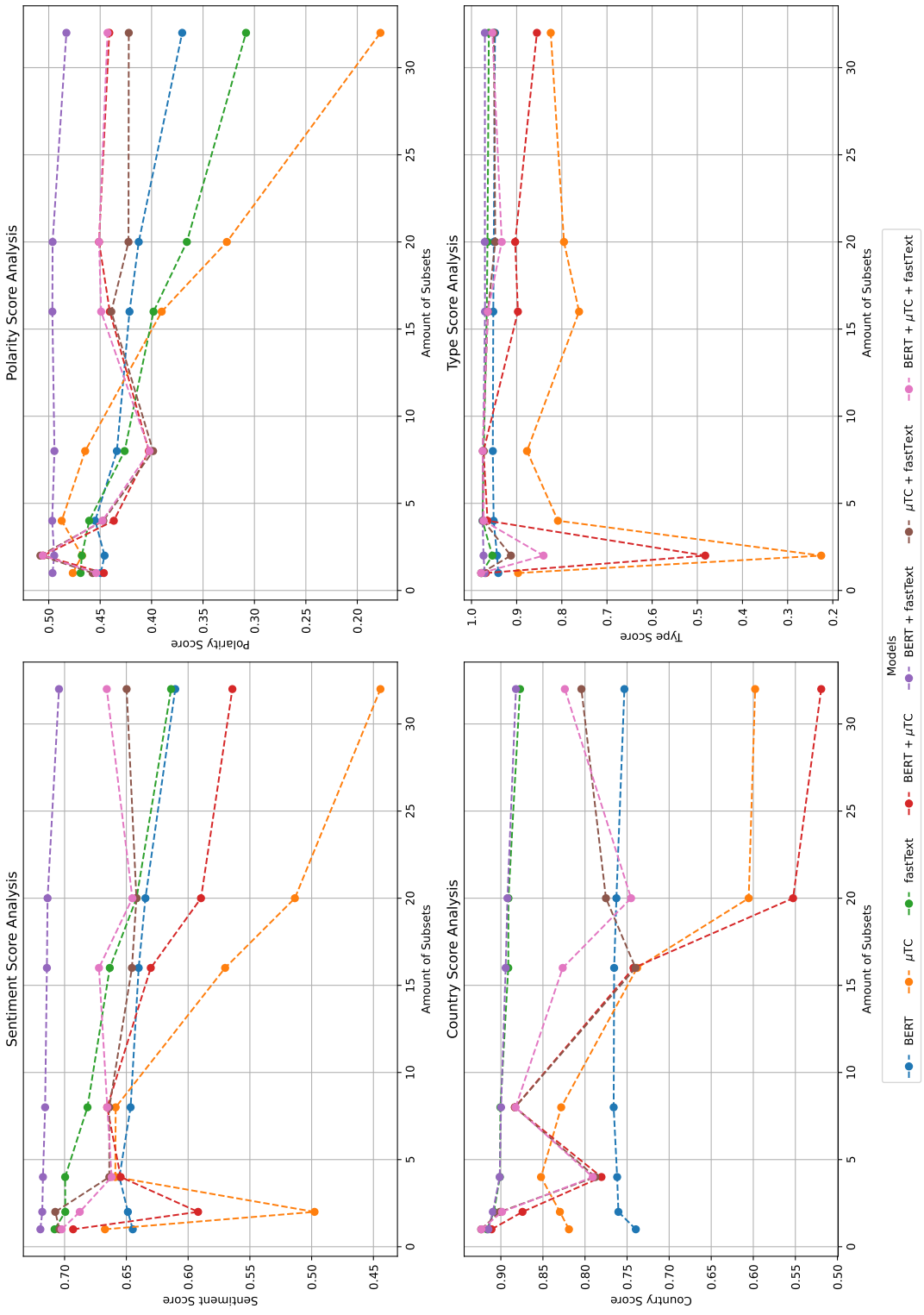

Figure 4.1: Polarity Score Analysis Across Subsets

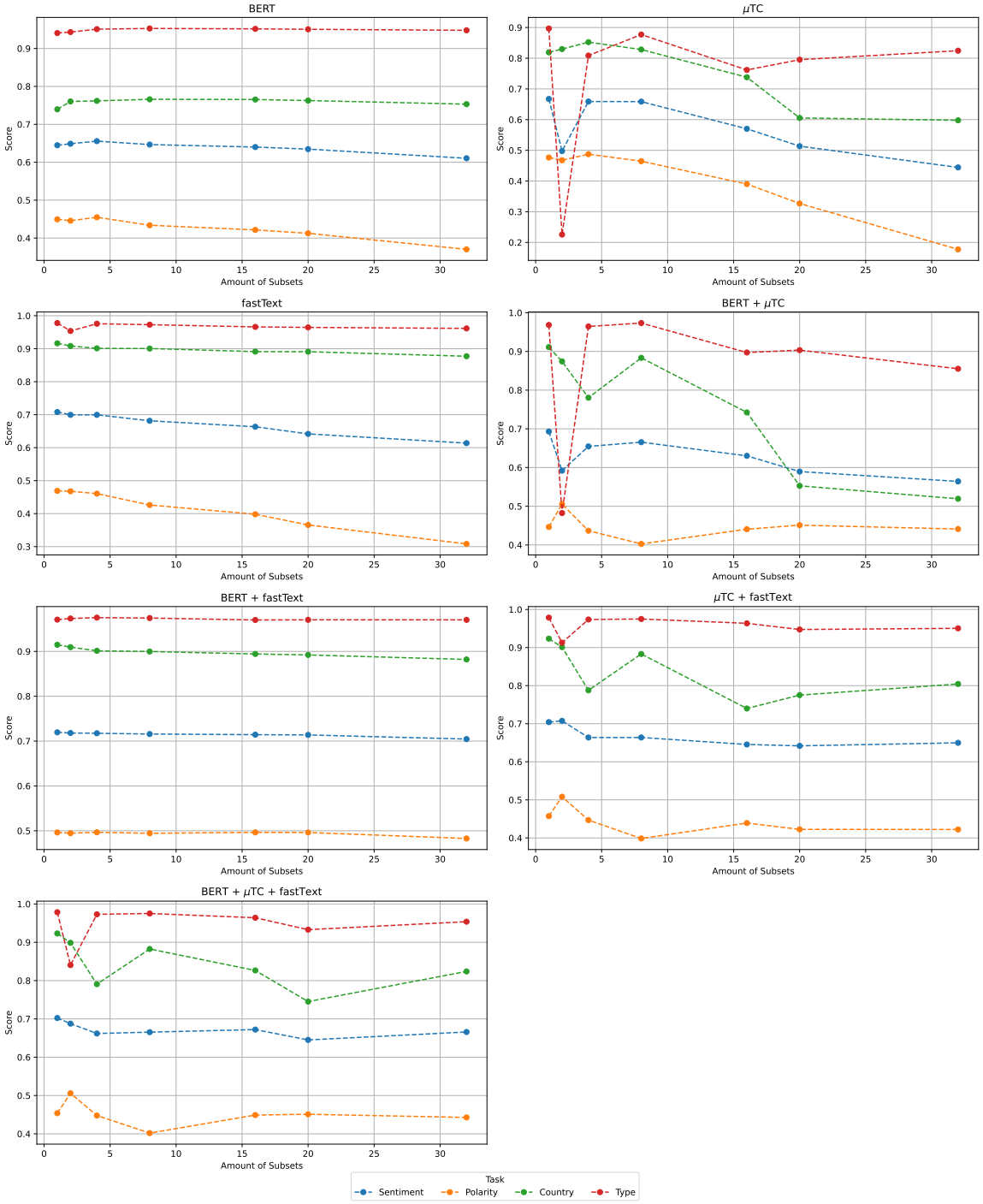

Figure 4.2: BERT + μTC + fastText Score Analysis for Each Metric Across Subsets

4.2. Rest-Mex 2023 Test Results

As the ensemble model that consistently came up on top from the results on the validation set was the **BERT + fastText** configuration trained on a single dataset, it was used to perform the evaluation on the test set DS_{RM_Test}. In addition to this, the fastText models from all subsets were also used to compare the results with the ensemble. The predicted classes were sent to the organizers of the competition. The results for the original competition are shown in Table 4.2, highlighted in bold. This table also includes the highest score obtained by the predictions sent to the organizers, which were those obtained by the ensemble model, highlighted in bold and italics.

Table 4.2: RestMex 2023 Sentiment Analysis Task Results

Run	Sentiment Track Score	Rank
LKE-IIMAS-Team_RUN_2	0.7790190145	1
javier_alonso-Team_sentiment_sub6	0.766199648	2
IIMAS-UNAM-Team_resultados	0.7500702313	3
INGEOTEC-Team	0.7375714733	4
UMSNH-Team_sentiment_results _xgb_ensemble (Updated Score)	***0.7195501814***	**-**
UCT_UA-Team_run_01	0.7190152899	5
BUAA-Team_M1_M1_M2	0.7189219428	6
Dataverse-Team_Results_RestMex23	0.7173586609	7
SENA-Team_i-1	0.700868302	8
UMSNH-Team_sentiment_results _xgb_ensemble (Original Score)	**0.6991728136**	**9**
Camed_CU-ES-Team_camed_camila _y_eduardo_results_output	0.6977828325	10
JL-Team-Added	0.6888903674	11
ITT-Team-Topics_k=1	0.6814168808	12
Algiedi-Team-5000	0.6691846165	13
Arandanito-Team_POS-ADJ+NOUN+VERB	0.6392190289	14
ABCD-Team_submission	0.4727653249	15
Olga-Team-dfTestfinal_LyS-SALSA	0.3144048884	16
The_Last-Team	0.2295272137	17

The results reinforce the belief that an ensemble model from the outputs of individual models generate a better classification system. The updated score obtained would have also

fared better in the original competition, landing the 5th rank instead of the 9th.

4.3. FIRE 2024 Validation Results

Table 4.3 presents the performance of all trained models and ensembles for both languages. The results indicate that ensemble models outperform individual models in at least one metric for Malayalam and achieve the highest F1 score for Tamil, albeit for a small margin. In these tests, the best performing ensemble was the **MuRIL + fastText + μTC + bert/l3cube-pune + XLM-RoBERTa** configuration, while the best performing individual model could arguably be optimized fastText, μTC or XLM-RoBERTa Base. None perform better than the other across all metrics, but the speed and ease of implementation of the fastText and μTC models arguably make up for the minimal differences.

Table 4.3: Results on DS_{FIRE_ValMal} and DS_{FIRE_ValTam} Across Trained Models and Ensembles

Model Name	**Malayalam**		**Tamil**	
	macro-F1	**Accuracy**	**macro-F1**	**Accuracy**
Default fastText+prepocessing	0.69	0.85	0.7	0.78
2-gram fastText+prepocessing	0.65	0.85	0.7	0.79
Optimized fastText (Malayalam/Tamil)	0.72	0.85	0.75	0.81
BERT (Malayalam/Tamil)	0.74	0.86	0.73	0.79
μTC	0.73	0.86	0.75	0.81
l3cube-pune/(Malayalam/Tamil)-bert	0.73	0.86	0.74	0.80
XLM-RoBERTa Base (Malayalam/Tamil)	0.74	0.85	0.74	0.80
MuRIL Large (Malayalam/Tamil)	0.72	0.86	0.69	0.76
Multilingual BERT (Malayalam/Tamil)	0.64	0.76	0.50	0.50
Optimized XGBoost (MuRIL + fastText)	0.73	0.86	0.75	0.80
Optimized XGBoost (MuRIL + fastText + Multilingual BERT)	0.73	0.86	0.75	0.81
Optimized XGBoost (MuRIL + fastText + μTC + bert/l3cube-pune)	0.74	0.86	0.76	0.81
Optimized XGBoost (MuRIL + fastText + μTC + bert/l3cube-pune + XLM-RoBERTa)	0.74	0.86	0.76	0.81

4.4. FIRE 2024 Test Results

Given that three submissions were allowed, the results from the models highlighted in bold in Table 4.3 were chosen for submission. For Tamil, the ensemble model secured the top rank among all contestants, while for Malayalam, the optimized fastText approach achieved a tied first-place ranking alongside five other teams, as shown in Table 4.4.

Table 4.4: Final Rankings for Malayalam and Tamil Languages

Malayalam			**Tamil**		
Team Name	**F1-Score**	**Rank**	**Team Name**	**F1-Score**	**Rank**
UMSNH_NLP	**0.76**	**1**	Awsathama	0.74	1
Awsathama	0.75	2	Team_catalysts	0.74	1
Codespark	0.74	3	Change_makers	0.74	1
IRLab@IITBHU	0.74	3	MUCS	0.74	1
Sarcasm_NLP	0.72	4	**UMSNH_NLP**	**0.74**	**1**
MUCS	0.72	4	IRLab@IITBHU	0.74	1
PixelPhrase	0.72	4	Sarcasm_NLP	0.73	2

4.5. General Discussion

In both competitions, combining multiple models into an ensemble tends to yield better results than relying on any single model. This improvement likely stems from the complementary strengths of different architectures, which mitigate the weaknesses inherent in a single approach. In Rest-Mex 2023, the **BERT + fastText** combination stood out, while in FIRE 2024, a more extensive **MuRIL + fastText + μTC + bert/l3cube-pune + XLM-RoBERTa** configuration achieved the top scores. These findings indicate that mixing diverse model architectures and feature representations generally improves classification quality.

The results from the Rest-Mex 2023 competition suggest that increasing the number of training subsets does not necessarily translate into continuous performance improvements.

While splitting the training data into subsets can initially improve generalization (perhaps by preventing overfitting or by creating more diverse training scenarios), there appears to be a threshold beyond which additional subsets contribute to performance deterioration. This suggests a delicate balance: too few subsets may limit exposure to varied training conditions, while too many can fragment the data and reduce model stability, ultimately harming performance.

Regarding model training times, the most noticeable observations are derived from the use of the BERT-based models and μTC. As one might imagine, the time was greatly affected by the size of the dataset being processed. For instance, the document embedding extraction process for the complete dataset of the Rest-Mex competition ($\tilde{2}$50000 samples) using BETO took 30 minutes, while the training of the MLP network took 4 minutes for the polarity model and 1.5 minutes for the country and type models. As the amount of subsets increased, these embedding extraction and model training times decreased.

The same is true for the μTC model training, where the parameter search phase takes the longest in the framework process, as there are a vast amount of possible configurations and the search is performed over 5 folds cross validation. For the complete Rest-Mex dataset, the parameter optimization process took 2 hours, while model training took 30 minutes. This time for μTC was considerably reduced when obtaining models for the FIRE competition, with the entire process lasting less than 10 minutes per model. The reduction in time is likely due to the smaller size of each dataset ($\tilde{1}$3188 training samples for Malayalam and 29570 for Tamil).

As for fastText, as the models were obtained using the library's automatic hyperparameter optimization set to last for 10 minutes, model training did not vary between different sized datasets.

Although ensembles generally dominate in terms of raw accuracy or F1 score, the differences compared to top individual models (e.g., fastText, μTC, or XLM-RoBERTa) can sometimes be minimal. When performance margins are slim, practical considerations such as model training speed, ease of deployment, and computational cost become significant.

Lastly, the superiority of ensembles is not confined to one particular metric, model architecture, or even language. In Rest-Mex 2023, ensembles excelled on multiple sentiment-

related metrics, while in FIRE 2024 they led to top ranks in different languages (Tamil and Malayalam). This shows that the general principle of ensembling as a performance enhancer is robust and broadly applicable.

4.6. Conclusion of Results Chapter

This chapter analyses the performance of individual models and ensembles across the Rest-Mex 2023 and DravidianCodeMix@FIRE-2024 competitions:

- **Rest-Mex 2023**:
  - **Validation Results**:
    - Ensembles (e.g., **BERT + fastText**) outperformed individual models across metrics (Table 4.1), achieving the highest Sentiment score (0.71959) and Res_{P} (0.49633).
    - Increasing training subsets (n_e) beyond 8 degraded performance (Figure 4.1), suggesting diminishing returns from data fragmentation.
  - **Test Results**:
    - The **BERT + fastText** ensemble achieved a revised Sentiment score of 0.71955 (Table 4.2), outperforming the original submission (0.69917) and ranking 5th post-evaluation.
- **DravidianCodeMix@FIRE-2024**:
  - **Validation Results**:
    - The **MuRIL + fastText + μTC + bert/l3cube-pune + XLM-RoBERTa** ensemble achieved the highest macro-F1 (0.74 for Malayalam, 0.76 for Tamil) (Table 4.3).
  - **Test Results**:
    - Ensembles secured **1st rank** for Tamil (0.76) and tied 1st for Malayalam (0.74) (Table 4.4).

- **Observations**:
  - Ensembles consistently outperformed individual models (e.g., fastText, μTC, BERT variants) by leveraging complementary strengths.
  - Optimal subset counts (n_e) balanced generalization and stability, with excessive subsets degrading performance (Figure 4.2).
  - Practical trade-offs (training speed, deployment ease) favoured fastText/μTC in FIRE despite marginal ensemble gains.

Chapter 5

Conclusions

5.1. General Conclusions

The work presented in this thesis shows that an ensemble-based approach to text classification, particularly when integrating deep learning models like BERT and simpler yet efficient methods such as fastText and μTC, can lead to perceivable improvements in performance across various tasks and datasets. By stacking the outputs of individual classifiers to form a new feature space and subsequently training an XGBoost classifier, the proposed system effectively leverages the strengths of diverse models while attempting to mitigate their individual shortcomings.

In the Rest-Mex 2023 competition, the ensemble configurations consistently outperformed individual models in most cases, especially the BERT + fastText configuration. Even under conditions involving smaller training subsets, the ensemble approach proved superior when compared to the individual models. Although certain models, like μTC, occasionally excelled in specific metrics, no single approach consistently rivalled the stable performance of the ensembles. This was further supported by the final test results, where the single set ensemble model would have achieved a higher ranking had it been submitted during the competition.

Similarly, the experiments conducted for the FIRE 2024 sentiment analysis task upheld the efficacy of ensemble strategies. Despite the relatively close performance margins

compared to individual models, the ensembles secured top positions in the validation phase, going so far as to achieve the highest score for the Tamil language in the testing phase. These results confirm that combining the predictions of multiple models is beneficial not only in sentiment analysis tasks but also in subtler classification challenges, such as identifying sarcasm in code-mixed data.

These findings highlight the potential of ensemble-based classification systems in handling both large and limited training sets, as well as in adapting to multiple languages and writing conventions. Ultimately, this thesis contributes to the growing body of evidence that ensembles are a powerful tool for improving classification outcomes in complex, real-world NLP tasks.

5.2. Future Work

1. Investigate methods that automatically select which models to include in an ensemble based on their performance on specific data segments or metrics, instead of using a fixed combination (e.g., always BERT + fastText).

2. Experiment with more sophisticated ensemble techniques, such as weighted voting schemes, where weights are learned from validation sets. This could better leverage each model's strengths and reduce the influence of weaker components.

3. Conduct further research into the optimal number of subsets and their composition. Methods could include data stratification based on similarity measures, aiming to form subsets that yield maximum diversity. In the same line of thought, integration of techniques like Bayesian optimization or genetic algorithms to find an optimal balance in the number of subsets, their size, and distribution without the need for extensive manual trial and error.

4. Test the proposed ensemble configurations and subset strategies on additional languages, domains, or modalities (e.g., speech, images) to assess their generality and robustness.

5. Evaluate the trade-offs between performance gains and computational overhead as ensemble sizes or subset counts grow.

Appendix A

Rest-Mex Validation Results Using Subset Approach

Table A.1: Results for 2 Subsets

Model / Ensemble	***Sentiment***	***Res$_P$***	***Res$_C$***	***Res$_A$***
BERT	0.64875	0.44569	0.76031	0.94333
μTC	0.49768	0.46763	0.82983	0.22561
fastText	0.69943	0.46783	0.90827	0.95379
BERT + μTC	0.59192	0.50552	0.87423	0.48242
BERT + fastText	**0.71806**	0.49467	**0.90950**	**0.97338**
μTC + fastText	0.70753	**0.50813**	0.90111	0.91274
BERT + μTC + fastText	0.68770	0.50585	0.89858	0.84053

Table A.2: Results for 4 Subsets

Model / Ensemble	*Sentiment*	Res_P	Res_C	Res_A
BERT	0.65562	0.45485	0.76180	0.95096
μTC	0.65894	0.48742	0.85230	0.80862
fastText	0.69956	0.46068	0.90116	**0.97572**
BERT + μTC	0.65460	0.43674	0.78041	0.96450
BERT + fastText	**0.71750**	**0.49664**	**0.90137**	0.97534
μTC + fastText	0.66378	0.44688	0.78767	0.97368
BERT + μTC + fastText	0.66190	0.44777	0.79074	0.97299

Table A.3: Results for 8 Subsets

Model / Ensemble	*Sentiment*	Res_P	Res_C	Res_A
BERT	0.64656	0.43377	0.76593	0.95279
μTC	0.65865	0.46458	0.82830	0.87714
fastText	0.68146	0.42622	**0.90054**	0.97287
BERT + μTC	0.66551	0.40264	0.88342	0.97333
BERT + fastText	**0.71576**	**0.49447**	0.89975	0.97435
μTC + fastText	0.66380	0.39854	0.88313	0.97501
BERT + μTC + fastText	0.66542	0.40200	0.88251	**0.97518**

Table A.4: Results for 16 Subsets

Model / Ensemble	***Sentiment***	***Res$_P$***	***Res$_C$***	***Res$_A$***
BERT	0.64002	0.42147	0.76537	0.95175
μTC	0.57005	0.39018	0.73805	0.76178
fastText	0.66340	0.39811	0.89119	0.96616
BERT + μTC	0.63022	0.44063	0.74241	0.89720
BERT + fastText	**0.71432**	**0.49647**	**0.89429**	**0.97007**
μTC + fastText	0.64548	0.43917	0.73996	0.96362
BERT + μTC + fastText	0.67216	0.44905	0.82649	0.96405

Table A.5: Results for 20 Subsets

Model / Ensemble	***Sentiment***	***Res$_P$***	***Res$_C$***	***Res$_A$***
BERT	0.63457	0.41250	0.76268	0.95061
μTC	0.51359	0.32683	0.60546	0.79523
fastText	0.64174	0.36569	0.89096	0.96463
BERT + μTC	0.58958	0.45113	0.55272	0.90335
BERT + fastText	**0.71377**	**0.49626**	**0.89216**	**0.97042**
μTC + fastText	0.64186	0.42247	0.77511	0.94738
BERT + μTC + fastText	0.64512	0.45110	0.74527	0.93300

Table A.6: Results for 32 Subsets

Model / Ensemble	***Sentiment***	***Res$_P$***	***Res$_C$***	***Res$_A$***
BERT	0.61041	0.37024	0.75319	0.94796
μTC	0.44432	0.17749	0.59781	0.82450
fastText	0.61378	0.30818	0.87721	0.96155
BERT + μTC	0.56417	0.44111	0.51927	0.85517
BERT + fastText	**0.70453**	**0.48283**	**0.88206**	**0.97038**
μTC + fastText	0.64978	0.42218	0.80428	0.95046
BERT + μTC + fastText	0.66585	0.44285	0.82395	0.95375

Appendix B

ROC Curves for fastText + BERT Ensemble on Rest-Mex Validation Set

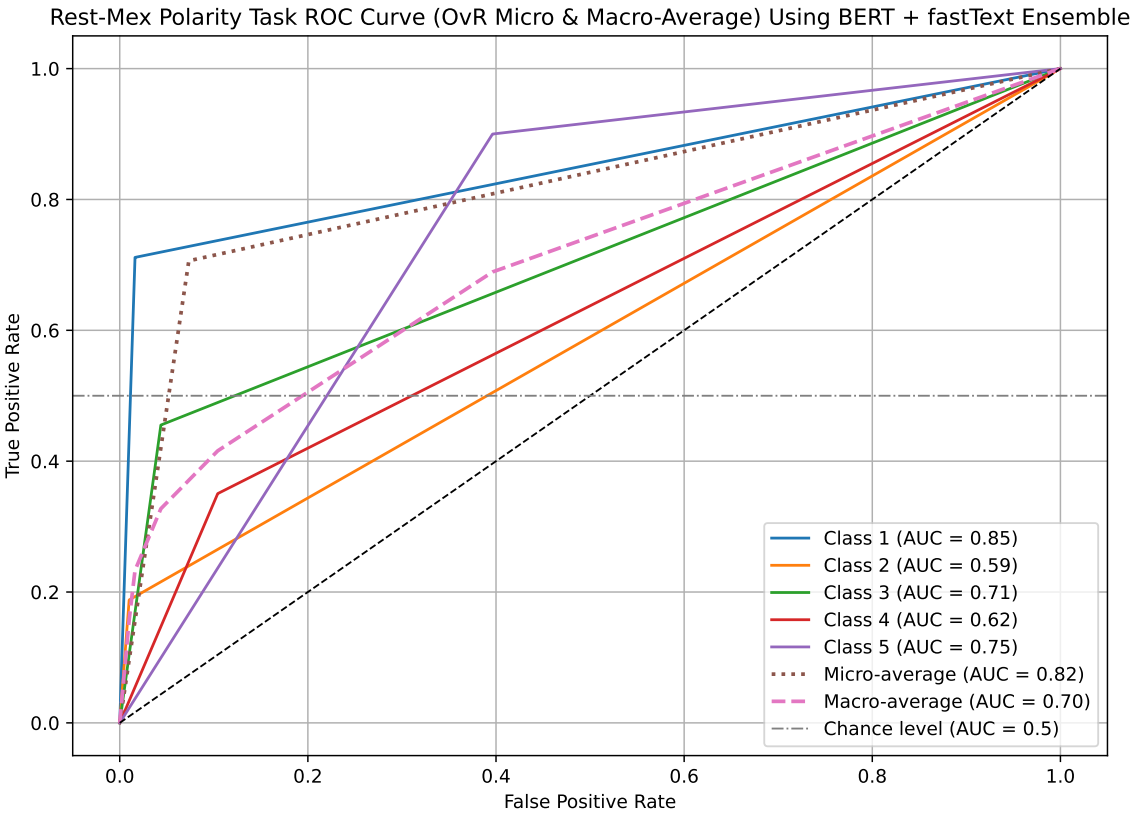

Figure B.1: BERT + fastText ROC Curves for Rest-Mex Polarity Task

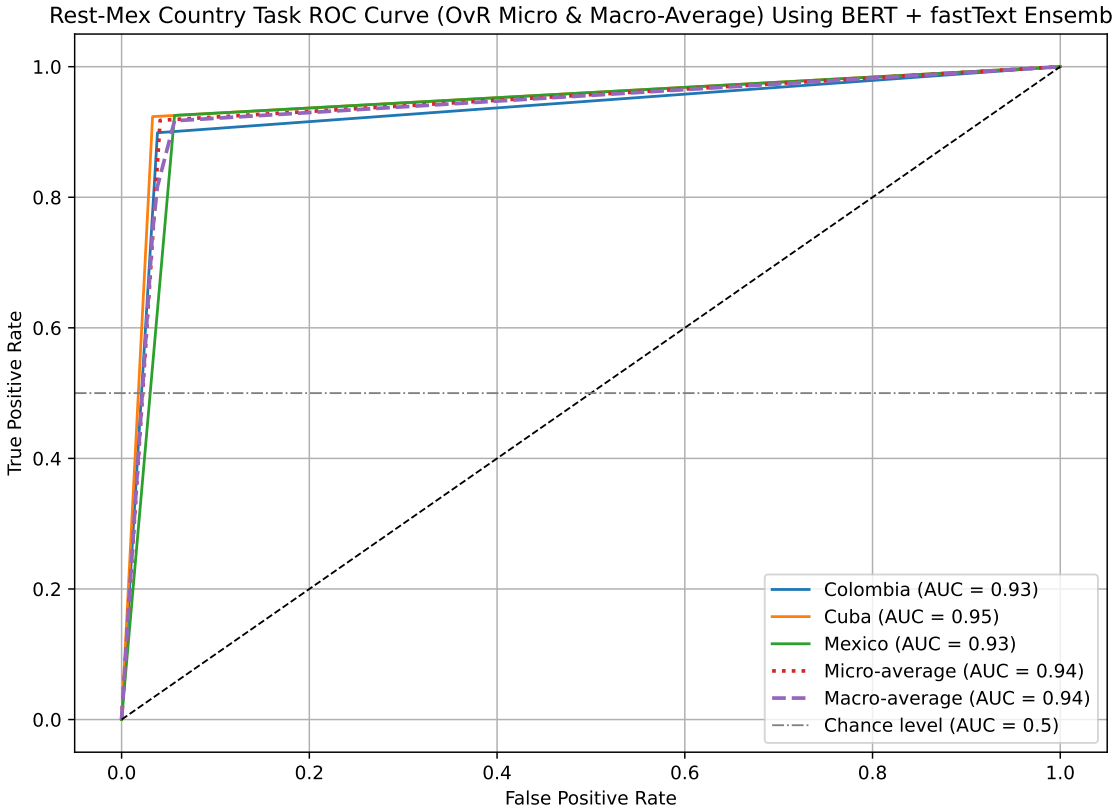

Figure B.2: BERT + fastText ROC Curves for Rest-Mex Country Task

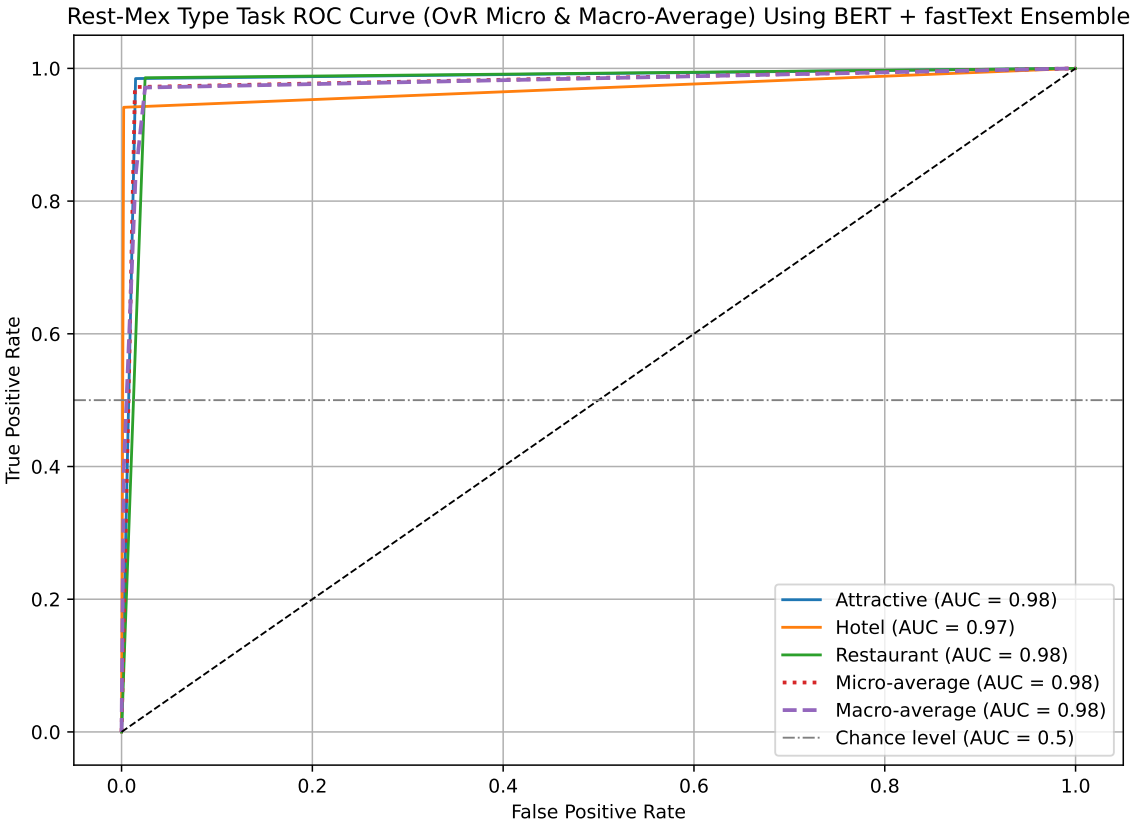

Figure B.3: BERT + fastText ROC Curves for Rest-Mex Type Task

References

[Abbas19] Abbas, M., Memon, K. A., Jamali, A. A., Memon, S., y Ahmed, A. Multinomial naive bayes classification model for sentiment analysis. *IJCSNS Int. J. Comput. Sci. Netw. Secur*, 19(3):62, 2019.

[Ahmad17] Ahmad, M., Aftab, S., y Ali, I. Sentiment analysis of tweets using svm. *Int. J. Comput. Appl*, 177(5):25–29, 2017.

[Alexander Genkin07] Alexander Genkin, D. D. L. y Madigan, D. Large-scale bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304, 2007. doi:10.1198/004017007000000245.
URL https://doi.org/10.1198/004017007000000245

[Álvarez-Carmona21] Álvarez-Carmona, M. A., Aranda, R., Arce-Cardenas, S., y Rafael Guerrero-Rodríguez, D. F.-D., López-Monroy, A. P., Martínez-Miranda, J., Pérez-Espinosa, H., y Rodríguez-González, A. Y. Overview of rest-mex at iberlef 2021: Recommendation system for text mexican tourism. *Procesamiento del Lenguaje Natural*, 67(0):163–172, 2021. ISSN 1989-7553.
URL http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/6386

[Álvarez-Carmona22a] Álvarez-Carmona, M. A., Aranda, R., Rodríguez-Gonzalez, A. Y., Fajardo-Delgado, D., Sánchez, M. G., Pérez-Espinosa, H., Martínez-Miranda, J., Guerrero-Rodríguez, R., Bustio-Martínez, L., y Díaz-Pacheco, A. Natural language processing applied to tourism

research: A systematic review and future research directions. *Journal of King Saud University - Computer and Information Sciences*, 34(10, Part B):10125–10144, 2022. ISSN 1319-1578. doi:https://doi.org/10.1016/j.jksuci.2022.10.010.
URL https://www.sciencedirect.com/science/article/pii/S1319157822003615

[Álvarez-Carmona22b] Álvarez-Carmona, M. A., Díaz-Pacheco, A., Aranda, R., Rodríguez-González, A. Y., Fajardo-Delgado, D., Guerrero-Rodríguez, R., y Bustio-Martínez, L. Overview of rest-mex at iberlef 2022: Recommendation system, sentiment analysis and covid semaphore prediction for mexican tourist texts. *Procesamiento del Lenguaje Natural*, 69(0):289–299, 2022. ISSN 1989-7553.
URL http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/6449

[Álvarez-Carmona23] Álvarez-Carmona, M. A., Díaz-Pacheco, A., Aranda, R., Rodríguez-González, A. Y., Muñiz-Sánchez, V., López-Monroy, A. P., Sánchez-Vega, F., y Bustio-Martínez, L. Overview of rest-mex at iberlef 2023: Research on sentiment analysis task for mexican tourist texts. *Procesamiento del Lenguaje Natural*, 71(0):425–436, 2023. ISSN 1989-7553.
URL http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/6572

[Anik Basu23] Anik Basu, B. y Mithun, D. Sarcasm detection in dravidian code-mixed text using transformer-based models. *En* Ghosh et al. [Ghosh23], págs. 249–258.
URL https://ceur-ws.org/Vol-3681/T5-3.pdf

[Bojanowski17] Bojanowski, P., Grave, E., Joulin, A., y Mikolov, T. Enriching word

vectors with subword information, 2017.
URL https://arxiv.org/abs/1607.04606

[Breiman01] Breiman, L. Random forests. *Mach. Learn.*, 45(1):5–32, oct. 2001. ISSN 0885-6125. doi:10.1023/A:1010933404324.
URL https://doi.org/10.1023/A:1010933404324

[Bunghez16] Bunghez, C. L. The importance of tourism to a destination's economy. *Journal of Eastern Europe Research in Business & Economics*, 2016:1–9, 2016.

[Carvalho09] Carvalho, P., Sarmento, L., Silva, M., y Oliveira, E. Clues for detecting irony in user-generated contents: Oh...!! it's "so easy";-). *International Conference on Information and Knowledge Management, Proceedings*, 11 2009. doi:10.1145/1651461.1651471.

[Cañete23] Cañete, J., Chaperon, G., Fuentes, R., Ho, J.-H., Kang, H., y Pérez, J. Spanish pre-trained bert model and evaluation data, 2023.
URL https://arxiv.org/abs/2308.02976

[Cerda-Flores23] Cerda-Flores, J., Hernández-Mazariegos, R., Ortiz-Bejar, J., Calderón-Solorio, F., y Ortiz-Bejar, J. Umsnh at restmex 2023: An xgboost stacking with pre-trained word-embeddings over data batches. *En* M. M. y Gómez, F. Rangel, S. M. Jiménez-Zafra, M. Casavantes, B. Altuna, M. A. Álvarez-Carmona, G. Bel-Enguix, L. Chiruzzo, I. de la Iglesia, H. J. Escalante, M. A. García-Cumbreras, J. A. García-Díaz, J. A. G. Barba, R. L. Tamayo, S. Lima, P. Moral, F. M. P. del Arco, y R. Valencia-García, eds., *Iberian Languages Evaluation Forum 2023*, nº 3496 en CEUR Workshop Proceedings, págs. 1–10. Aachen, 2023. ISSN 1613-0073.
URL https://ceur-ws.org/Vol-3496/restmex-paper19.pdf

[Chakravarthi23a] Chakravarthi, B. R., N, S., B, B., K, N., Navaneethakrishnan, S. C., Durairaj, T., Ponnusamy, R., Kumaresan, P. K., Ponnusamy, K. K.,

y Rajkumar, C. Overview of sarcasm identification of dravidian languages in dravidiancodemix@fire-2023. *En FIRE (Working Notes)*, págs. 231–239. 2023.
URL https://ceur-ws.org/Vol-3681/T5-1.pdf

[Chakravarthi23b] Chakravarthi, B. R., N, S., B, B., K, N., Navaneethakrishnan, S. C., Durairaj, T., Ponnusamy, R., Kumaresan, P. K., Ponnusamy, K. K., y Rajkumar, C. Overview of sarcasm identification of dravidian languages in dravidiancodemix@fire-2023. *En FIRE (Working Notes)* [Chakravarthi23a], págs. 231–239.
URL https://ceur-ws.org/Vol-3681/T5-1.pdf

[Chen16] Chen, T. y Guestrin, C. Xgboost: A scalable tree boosting system. *En Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pág. 785–794. Association for Computing Machinery, New York, NY, USA, 2016. ISBN 9781450342322. doi:10.1145/2939672.2939785.
URL https://doi.org/10.1145/2939672.2939785

[Conneau20] Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., y Stoyanov, V. Unsupervised cross-lingual representation learning at scale, 2020.
URL https://arxiv.org/abs/1911.02116

[Cortes09] Cortes, C. y Vapnik, V. Support-vector networks. *Chem. Biol. Drug Des.*, 297:273–297, 01 2009. doi:10.1007/%2FBF00994018.

[Cover67] Cover, T. y Hart, P. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. doi:10.1109/TIT.1967.1053964.

[Devlin19] Devlin, J., Chang, M.-W., Lee, K., y Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.
URL https://arxiv.org/abs/1810.04805

[Fang20] Fang, Z., Zhang, Q., Tang, X., Wang, A., y Baron, C. An implicit opinion analysis model based on feature-based implicit opinion patterns. *Artificial Intelligence Review*, 53:4547–4574, 2020.

[Ferri09] Ferri, C., Hernández-Orallo, J., y Modroiu, R. An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, 30(1):27–38, 2009. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2008.08.010.
URL https://www.sciencedirect.com/science/article/pii/S0167865508002687

[Filatova12] Filatova, E. Irony and sarcasm: Corpus generation and analysis using crowdsourcing. *En Lrec*, págs. 392–398. Citeseer, 2012.

[Freund99] Freund, Y., Schapire, R., y Abe, N. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[Gasparetto22] Gasparetto, A., Marcuzzo, M., Zangari, A., y Albarelli, A. A survey on text classification algorithms: From text to predictions. *Information*, 13(2), 2022. ISSN 2078-2489. doi:10.3390/info13020083.
URL https://www.mdpi.com/2078-2489/13/2/83

[Ghosh23] Ghosh, K., Mandl, T., Majumder, P., y Mitra, M., eds. *Working Notes of FIRE 2023 - Forum for Information Retrieval Evaluation (FIRE-WN 2023)*, nº 3681 en CEUR Workshop Proceedings. Aachen, 2023. ISSN 1613-0073.
URL https://ceur-ws.org/Vol-3681/

[Graff20] Graff, M., Miranda-Jimenez, S., Tellez, E. S., y Moctezuma, D. Evomsa: A multilingual evolutionary approach for sentiment analysis [application notes]. *IEEE Computational Intelligence Magazine*, 15(1):76–88, feb. 2020. ISSN 1556-6048. doi:10.1109/mci.2019.

2954668.
URL http://dx.doi.org/10.1109/MCI.2019.2954668

[Han01] Han, E.-H. S., Karypis, G., y Kumar, V. Text categorization using weight adjusted k-nearest neighbor classification. *En* D. Cheung, G. J. Williams, y Q. Li, eds., *Advances in Knowledge Discovery and Data Mining*, págs. 53–65. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. ISBN 978-3-540-45357-4.

[Haumahu21] Haumahu, J. P., Permana, S. D. H., y Yaddarabullah, Y. Fake news classification for indonesian news using extreme gradient boosting (xgboost). *IOP Conference Series: Materials Science and Engineering*, 1098(5):052081, mar 2021. doi:10.1088/1757-899X/1098/5/052081.
URL https://dx.doi.org/10.1088/1757-899X/1098/5/052081

[Hossin15] Hossin, M. y M.N, S. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5:01–11, 03 2015. doi:10.5121/ijdkp.2015.5201.

[Hota18] Hota, S. y Pathak, S. Knn classifier based approach for multi-class sentiment analysis of twitter data. *Int. J. Eng. Technol*, 7(3):1372–1375, 2018.

[Huq17] Huq, M. R., Ahmad, A., y Rahman, A. Sentiment analysis on twitter data using knn and svm. *International Journal of Advanced Computer Science and Applications*, 8(6), 2017.

[Isnain21] Isnain, A., Supriyanto, J., y Kharisma, M. Implementation of k-nearest neighbor (k-nn) algorithm for public sentiment analysis of online learning. *IJCCS (Indonesian Journal of Computing and Cybernetics Systems)*, 15(2):121–130, 2021. ISSN 2460-7258. doi:10.22146/ijccs.65176.
URL https://jurnal.ugm.ac.id/ijccs/article/view/65176

[Joachims98] Joachims, T. Text categorization with support vector machines: Learning with many relevant features. *En* C. Nédellec y C. Rouveirol, eds., *Machine Learning: ECML-98*, págs. 137–142. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-69781-7.

[Joshi23] Joshi, R. L3cube-hindbert and devbert: Pre-trained bert transformer models for devanagari based hindi and marathi languages, 2023.
URL https://arxiv.org/abs/2211.11418

[Joulin16] Joulin, A., Grave, E., Bojanowski, P., y Mikolov, T. Bag of tricks for efficient text classification, 2016.
URL https://arxiv.org/abs/1607.01759

[Khanuja21a] Khanuja, S., Bansal, D., Mehtani, S., Khosla, S., Dey, A., Gopalan, B., Margam, D. K., Aggarwal, P., Nagipogu, R. T., Dave, S., Gupta, S., Gali, S. C. B., Subramanian, V., y Talukdar, P. Muril: Multilingual representations for indian languages, 2021.
URL https://arxiv.org/abs/2103.10730

[Khanuja21b] Khanuja, S., Bansal, D., Mehtani, S., Khosla, S., Dey, A., Gopalan, B., Margam, D. K., Aggarwal, P., Nagipogu, R. T., Dave, S., et al. Muril: Multilingual representations for indian languages. *arXiv preprint arXiv:2103.10730*, 2021.

[Kirk23] Kirk, H., Yin, W., Vidgen, B., y Röttger, P. SemEval-2023 task 10: Explainable detection of online sexism. *En* A. K. Ojha, A. S. Doğruöz, G. Da San Martino, H. Tayyar Madabushi, R. Kumar, y E. Sartori, eds., *Proceedings of the 17th International Workshop on Semantic Evaluation (SemEval-2023)*, págs. 2193–2210. Association for Computational Linguistics, Toronto, Canada, jul. 2023. doi: 10.18653/v1/2023.semeval-1.305.
URL https://aclanthology.org/2023.semeval-1.305

[Kowsari19] Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., y Brown, D. Text classification algorithms: A survey. *Information*, 10(4), 2019. ISSN 2078-2489. doi:10.3390/info10040150.
URL https://www.mdpi.com/2078-2489/10/4/150

[Krishnan23] Krishnan, D., Dharanikota, K., y Bharathi, B. Cross-linguistic sarcasm detection in tamil and malayalam: A multilingual approach. *En* Ghosh et al. [Ghosh23], págs. 259–269.
URL https://ceur-ws.org/Vol-3681/T5-4.pdf

[Lewis98] Lewis, D. D. Naive (bayes) at forty: The independence assumption in information retrieval. *En* C. Nédellec y C. Rouveirol, eds., *Machine Learning: ECML-98*, págs. 4–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-69781-7.

[Li22] Li, Q., Peng, H., Li, J., Xia, C., Yang, R., Sun, L., Yu, P. S., y He, L. A survey on text classification: From traditional to deep learning. *ACM Trans. Intell. Syst. Technol.*, 13(2), abr. 2022. ISSN 2157-6904. doi:10.1145/3495162.
URL https://doi.org/10.1145/3495162

[Manning08] Manning, C., Raghavan, P., y Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN 9781139472104.
URL https://books.google.com.mx/books?id=t1PoSh4uwVcC

[Maron60] Maron, M. E. y Kuhns, J. L. On relevance, probabilistic indexing and information retrieval. *J. ACM*, 7(3):216–244, jul. 1960. ISSN 0004-5411. doi:10.1145/321033.321035.
URL https://doi.org/10.1145/321033.321035

[Mikolov13] Mikolov, T., Chen, K., Corrado, G., y Dean, J. Efficient estimation of word representations in vector space, 2013.
URL https://arxiv.org/abs/1301.3781

[Obi23] Obi, J. A comparative study of several classification metrics and their performances on data. *World Journal of Advanced Engineering Technology and Sciences*, 8:308–314, 02 2023. doi:10.30574/wjaets.2023.8.1.0054.

[Radford19] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[Ravichandiran21] Ravichandiran, S. *Getting Started with Google BERT: Build and train state-of-the-art natural language processing models using BERT*. Packt Publishing, 2021. ISBN 9781838826239.
URL https://books.google.com.mx/books?id=CvsWEAAAQBAJ

[Sebastiani02] Sebastiani, F. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, mar. 2002. ISSN 0360-0300. doi:10.1145/505282.505283.
URL https://doi.org/10.1145/505282.505283

[Singh19] Singh, G., Kumar, B., Gaur, L., y Tyagi, A. Comparison between multinomial and bernoulli naïve bayes for text classification. *En 2019 International Conference on Automation, Computational and Technology Management (ICACTM)*, págs. 593–596. 2019. doi:10.1109/ICACTM.2019.8776800.

[Taboada16] Taboada, M. Sentiment analysis: An overview from linguistics. *Annual Review of Linguistics*, 2(Volume 2, 2016):325–347, 2016. ISSN 2333-9691. doi:https://doi.org/10.1146/annurev-linguistics-011415-040518.
URL https://www.annualreviews.org/content/journals/10.1146/annurev-linguistics-011415-040518

[Tasnia23] Tasnia, R., Ayman, N., Sultana, A., Chy, A. N., y Aono, M. Exploiting stacked embeddings with lstm for multilingual humor and irony

detection. *Social Network Analysis and Mining*, 13:1–24, 2023. URL https://api.semanticscholar.org/CorpusID:257286492

[Tellez18] Tellez, E. S., Moctezuma, D., Miranda-Jiménez, S., y Graff, M. An automated text categorization framework based on hyperparameter optimization. *Knowledge-Based Systems*, 149:110–123, 2018. ISSN 0950-7051. doi:https://doi.org/10.1016/j.knosys.2018.03.003. URL https://www.sciencedirect.com/science/article/pii/S0950705118301217

[Vaswani23] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., y Polosukhin, I. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762

[Vujovic21] Vujovic, Z. Classification model evaluation metrics. *International Journal of Advanced Computer Science and Applications*, Volume 12:599–606, 07 2021. doi:10.14569/IJACSA.2021.0120670.

[Wiener95] Wiener, E., Pedersen, J. O., Weigend, A. S., et al. A neural network approach to topic spotting. *En Proceedings of SDAIR-95, 4th annual symposium on document analysis and information retrieval*, tomo 317, pág. 332. Citeseer, 1995.

[Wolpert92] Wolpert, D. H. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992. ISSN 0893-6080. doi:https://doi.org/10.1016/S0893-6080(05)80023-1. URL https://www.sciencedirect.com/science/article/pii/S0893608005800231

[Wongkar19] Wongkar, M. y Angdresey, A. Sentiment analysis using naive bayes algorithm of the data crawler: Twitter. *En 2019 Fourth International Conference on Informatics and Computing (ICIC)*, págs. 1–5. IEEE, 2019.

[Xu18] Xu, S. Bayesian naïve bayes classifiers to text classification. *Journal of Information Science*, 44(1):48–59, 2018. doi:10.1177/0165551516677946.
URL https://doi.org/10.1177/0165551516677946

[Zainuddin14] Zainuddin, N. y Selamat, A. Sentiment analysis using support vector machine. *En 2014 international conference on computer, communications, and control technology (I4CT)*, págs. 333–337. IEEE, 2014.

Jaime Cerda Flores

ENSEMBLES OF PRE-TRAINED MODELS APPLIED TO TWO TEXT CLASSIFICATION COMPETITIONS.pdf

Universidad Michoacana de San Nicolás de Hidalgo

Detalles del documento

Identificador de la entrega
trn:oid:::3117:432250599

Fecha de entrega
20 feb 2025, 8:15 a.m. GMT-6

Fecha de descarga
20 feb 2025, 8:19 a.m. GMT-6

Nombre de archivo
ENSEMBLES OF PRE-TRAINED MODELS APPLIED TO TWO TEXT CLASSIFICATION COMPETITIONS.pdf

Tamaño de archivo
2.5 MB

121 Páginas

27,984 Palabras

149,596 Caracteres

Formato de Declaración de Originalidad y Uso de Inteligencia Artificial

Coordinación General de Estudios de Posgrado
Universidad Michoacana de San Nicolás de Hidalgo

A quien corresponda,

Por este medio, quien abajo firma, bajo protesta de decir verdad, declara lo siguiente:

- Que presenta para revisión de originalidad el manuscrito cuyos detalles se especifican abajo.
- Que todas las fuentes consultadas para la elaboración del manuscrito están debidamente identificadas dentro del cuerpo del texto, e incluidas en la lista de referencias.
- Que, en caso de haber usado un sistema de inteligencia artificial, en cualquier etapa del desarrollo de su trabajo, lo ha especificado en la tabla que se encuentra en este documento.
- Que conoce la normativa de la Universidad Michoacana de San Nicolás de Hidalgo, en particular los Incisos IX y XII del artículo 85, y los artículos 88 y 101 del Estatuto Universitario de la UMSNH, además del transitorio tercero del Reglamento General para los Estudios de Posgrado de la UMSNH.

Datos del manuscrito que se presenta a revisión		
Programa educativo	Maestría en Ciencias en Ingeniería Eléctrica	
Título del trabajo	Text Categorization Using Ensembles Based on Stacked Outputs of Pre-trained Models	
	Nombre	**Correo electrónico**
Autor/es	Jaime Cerda Flores	1130834h@umich.mx
Director	José Ortiz Bejar	jose.ortiz@umich.mx
Codirector	Félix Calderón Solorio	felix.calderon@umich.mx
Coordinador del programa	Norberto García Barriga	norberto.garcia@umich.mx

Uso de Inteligencia Artificial		
Rubro	**Uso (sí/no)**	**Descripción**
Asistencia en la redacción	no	
Traducción al español	no	
Traducción a otra lengua	no	
Revisión y corrección de estilo	si	reducción de repetitividad y aclaración de ideas

Formato de Declaración de Originalidad y Uso de Inteligencia Artificial

Coordinación General de Estudios de Posgrado
Universidad Michoacana de San Nicolás de Hidalgo

Uso de Inteligencia Artificial		
Rubro	**Uso (sí/no)**	**Descripción**
Análisis de datos	no	
Búsqueda y organización de información	no	
Formateo de las referencias bibliográficas	no	
Generación de contenido multimedia	si	edición de tablas en látex
Otro	no	

Datos del solicitante	
Nombre y firma	Jaime Cerda Flores
Lugar y fecha	Morelia, Michoacán, 4 de diciembre de 2024