The Scientific Temper (2024) Vol. 15 (3): 2598-2608

E-ISSN: 2231-6396, ISSN: 0976-8653

Doi: 10.58414/SCIENTIFICTEMPER.2024.15.3.27

https://scientifictemper.com/

RESEARCH ARTICLE

Enhancing classification accuracy on code-mixed and imbalanced data using an adaptive deep autoencoder and XGBoost

Ayesha Shakith*, L. Arockiam

Abstract

This study introduces a pioneering approach for enhancing classification accuracy on code-mixed and imbalanced data by integrating an adaptive deep autoencoder with dynamic sampling techniques. Targeting the intricate challenges of sentiment analysis within such datasets, this methodology employs an enhanced XGBoost classifier optimized to leverage the nuanced features extracted by the autoencoder. The experimental evaluation across diverse datasets, predominantly involving Tamil-English code-mixed texts, demonstrates a notable improvement in performance metrics: accuracy reached 84.2%, precision was recorded at 74.8%, recall stood at 78.4%, and the F1-score achieved 76.6%. This marks an enhancement over existing methods by 0.5 to 1.5%, substantiating the model's robust capability in effectively handling linguistic diversity and class imbalances. The novelty of this research lies in the seamless integration of dynamic sampling within the autoencoder's training loop, significantly boosting the adaptability and effectiveness of the machine-learning model in real-world applications.

Keywords: Sentiment analysis, Deep learning, Code-mixing, Autoencoder, Imbalance classification.

Introduction

In the digital age, social media platforms have emerged as prolific data generation sources, with millions of users interacting daily through posts, comments, and messages (Mindel, V, *et al*., 2024). This vast volume of data is not only a reflection of current trends and public opinion but also serves as a rich resource for various applications such as market analysis, political forecasting, and customer service improvements (Blazquez, D., *et al*., 2018). Particularly, sentiment analysis leverages this data to gauge public sentiment, enabling organizations to understand consumer emotions and reactions towards products, services, or events (Sykora M, *et al*.,2022). However, the utility of this data is contingent upon the ability to accurately analyze and interpret the content, which is often presented in informal, abbreviated, or colloquial language, adding layers of complexity to data processing (Abualigah L, *et al*., 2021).

One of the significant challenges in leveraging social media data for sentiment analysis arises from the prevalence of code-mixed text (Thara S, *et al*., 2022, Jamatia A, *et al*., 2020), (Ahmad G. I., *et al*., 2022, Astuti L. W, *et al*., 2023). Code-mixing, a common linguistic phenomenon in multilingual regions, involves switching between two or more languages within a sentence or discourse (Jamatia A, *et al*., 2020). This practice, while reflective of cultural diversity and linguistic behavior, poses unique challenges for computational analysis (Veeramani H, *et al*., 2024). Traditional natural language processing tools are often ill-equipped to handle the syntactic and semantic irregularities introduced by code-mixing, leading to decreased accuracy in sentiment classification (Perera A, *et al*., 2024)

Moreover, the datasets derived from social media are typically imbalanced, with uneven distributions of sentiment classes, further complicating the training and effectiveness of predictive models (Huang J. Y. *et al*., 2022). Addressing these challenges requires innovative approaches that can robustly handle the complexities of code-mixed and imbalanced datasets (Bölücü N, *et al*., 2024).

Department of Computer Science, St. Joseph's College (Autonomous), Affiliated to Bharathidasan University, Trichy, India.

***Corresponding Author:** Ayesha Shakith, Department of Computer Science, St. Joseph's College (Autonomous), Affiliated to Bharathidasan University, Trichy, India., E-Mail: Ayeshasm1412@gmail.com

How to cite this article: Shakith, A., Arockiam, L. (2024). Enhancing classification accuracy on code-mixed and imbalanced data using an adaptive deep autoencoder and XGBoost. The Scientific Temper, **15**(3):2598-2608.

Doi: 10.58414/SCIENTIFICTEMPER.2024.15.3.27

Source of support: Nil

Conflict of interest: None.

Received: 03/07/2024 **Accepted:** 13/08/2024 **Published**: 10/09/2024

The motivation behind the development of advanced sentiment analysis tools for code-mixed data stems from the increasing globalization and digital communication which fuse linguistic boundaries, creating a rich tapestry of multicultural interactions online. As businesses and governments seek to engage with and understand diverse global audiences, the need to navigate and analyze multilingual content effectively becomes critical. This is particularly pertinent in regions with high bilingualism where code-switching in digital communication is a norm rather than an exception. The ability to accurately assess sentiments from such complex datasets not only enhances customer interaction and satisfaction but also aids in the nuanced understanding of social dynamics and cultural nuances that are often embedded in language use. This involves innovating solutions that can leverage deep learning to discern patterns and nuances in mixed-language texts, thereby providing more accurate and contextually relevant insights into public sentiment and opinion.

The contribution of this research is manifold, centering around the innovative integration of dynamic sampling techniques within an autoencoder's training loop and the strategic enhancement of the XGBoost algorithm to optimize its performance for complex, code-mixed text datasets.

Firstly, the incorporation of dynamic sampling, specifically through synthetic minority over-sampling technique - edited nearest neighbors (SMOTE-ENN), directly within the training process of the autoencoder, represents a novel approach. This integration allows the model to continuously adjust to the imbalances in the dataset during the training phase rather than as a preprocessing step. This real-time adjustment ensures that the feature extraction is not only adaptive but also sensitive to the minority classes, significantly enhancing the robustness and accuracy of the sentiment analysis.

Secondly, the enhancement of XGBoost in this framework involves tailoring it to work seamlessly with the deep features extracted by the autoencoder. This includes tuning hyperparameters specifically for handling the complexities of code-mixed data, such as adjusting the 'scale_pos_weight' to compensate for class imbalances and optimizing the depth and number of trees to capture nuanced linguistic patterns effectively. The enhanced XGBoost model, therefore, is not merely a standalone classifier but a sophisticated component that exploits the rich, nuanced features provided by the autoencoder, leading to improved classification outcomes.

This paper is systematically organized into several key sections to ensure a thorough presentation of the research. Following the introductory remarks that set the stage for the significance and need for advanced sentiment analysis tools, the paper delves into a comprehensive review of related work, highlighting past efforts and current trends in handling code-mixed text. The methodology section then details the innovative approaches employed in this study, including the architecture of the adaptive deep autoencoder, the integration of dynamic sampling techniques, and the enhancements made to the XGBoost classifier. Subsequent sections provide a detailed description of the experimental setup, including data preparation, model configuration, and the parameters used, followed by an in-depth discussion of the results, showcasing the effectiveness of the proposed solutions through various performance metrics. The paper concludes with a discussion that interprets these results, outlines the limitations encountered, and proposes directions for future research.

Related Work

Sentiment analysis of social media content presents a critical area of research, providing valuable insights into public opinion and behaviors. The prevalence of social media has resulted in the generation of vast amounts of data daily, characterized by diverse linguistic forms, including code-mixed text where two or more languages are combined. This phenomenon is particularly common in multilingual societies, where users often switch between languages, creating texts that are not strictly bound to the syntactic or semantic rules of a single language.

The task of sentiment analysis in such code-mixed environments introduces specific challenges, primarily due to the linguistic complexity and the sparsity of labeled data for training machine learning models (Saini J. R, *et al*., 2023). Most traditional sentiment analysis tools are designed for monolingual text and struggle with the irregularities presented by code-mixing, such as unexpected grammatical structures and mixed-language entities. Moreover, the data collected from social media is often imbalanced, with the overrepresentation of some sentiments over others, further complicating the development of robust sentiment analysis systems (Saini, J. R., *et al*., 2023).

Recent research has highlighted these challenges, with studies such as the SemEval-2020 Task 9 exploring sentiment analysis for code-mixed tweets in languages like Hindi-English and Spanish-English, demonstrating varied success across different language pairs (Rogers D, *et al*., 2021) (Sengupta A, *et al*., 2021). These studies emphasize the need for innovative solutions that can adapt to the intricacies of code-mixed text while effectively managing dataset imbalances. For example, efforts in developing datasets and benchmarks specifically for code-mixed languages aim to foster advancements in this field, as seen in the creation of sentiment analysis corpora for languages like Malayalam-English and Kannada-English (Astuti L. W, *et al*., 2023), (Chakravarthi B. R, *et al*., 2022). These foundational works set the stage for further exploration and development of advanced analytical tools capable of handling the complex dynamics of code-mixed sentiment analysis.

Further advancements in the field of code-mixed sentiment analysis have been reported, reflecting the ongoing efforts to refine methodologies and broaden the linguistic scope of analysis (Shanmugavadivel, K, *et al*., 2022), (Nankani, H, Dutta, *et al*., 2020), (Yusuf, A, *et al*., 2024). Notably, a study conducted by Mohammad Tareq *et al*. presented an innovative approach to enhancing cross-linguistic contextual understanding through data augmentation techniques (Dey, S, *et al*., 2024). underscores the importance of developing tailored augmentation strategies to improve the robustness of sentime nt analysis models in handling code-mixed languages (Mohammad Tareq *et al*., 2023).

Another significant contribution is from the work of (Kumaresan *et al*., 2023) where researchers developed multitasking models that outperform traditional single-task systems in sentiment and emotion recognition tasks. This study demonstrates an F1 score improvement, highlighting the efficiency of multitasking architectures in extracting and processing complex emotional cues from code-mixed texts.

Proposed Methodology

Adaptive Deep Autoencoder with Dynamic Sampling

The core of our methodology is the adaptive deep autoencoder designed to effectively handle the complexities of code-mixed text data. The autoencoder architecture consists of multiple layers, each aimed at decomposing the text data into a more manageable and representative form. This includes an input layer, several hidden layers, and a bottleneck layer where the data is most compressed, facilitating the extraction of salient features crucial for sentiment analysis. What sets our autoencoder apart is the integration of dynamic sampling techniques directly into its training loop. We employ SMOTE-ENN within the training process to address class imbalance by synthesizing new minority class samples and cleaning overlapping samples, respectively. This integration allows the autoencoder to continuously adjust to the diversity and imbalance of the dataset, enhancing its ability to generalize from training data to real-world applications.

Model design

The architecture of the adaptive deep autoencoder is meticulously designed to tackle the nuances of code-mixed text data, providing a powerful tool for feature extraction that adapts to both the complexity and imbalance of the dataset. Here's a detailed breakdown of the autoencoder's architecture:

Input layer

The first layer of the autoencoder receives the input data, which consists of high-dimensional vectors representing the code-mixed text. These vectors are typically derived from embedding techniques such as word embeddings or TF-IDF vectorization, which transform the raw text into a numerical format that neural networks can process.

Encoding layers

Following the input layer, several encoding layers are used to progressively compress the data into a more compact representation. Each encoding layer consists of a fully connected (dense) neural network layer followed by a non-linear activation function. Usually, rectified linear unit (ReLU) introduces non-linearity into the model. These layers are responsible for capturing and encoding the underlying patterns and structures in the data, reducing dimensionality while retaining crucial information.

Bottleneck layer

At the heart of the autoencoder is the bottleneck layer, where the data representation is at its most compressed form. This layer is crucial as it serves as the feature extraction phase of the model, where the most salient and robust features of the input data are retained. The bottleneck layer's design is pivotal in determining the quality and effectiveness of the features extracted, influencing the overall performance of the sentiment analysis.

Decoding layers

Symmetric to the encoding layers, the decoding layers aim to reconstruct the input data from the compressed form. These layers gradually expand the compressed data back to its original dimensionality, using a similar structure of dense layers and non-linear activation functions. The decoding process is essential for the autoencoder to learn a lossy but efficient representation of the input data, minimizing the loss between the original and reconstructed data.

Output layer

The final layer of the autoencoder outputs the reconstructed data, matching the dimensionality of the input layer. This layer typically uses a sigmoid or linear activation function, depending on the nature of the input data, to ensure that the output values are in a suitable range.

Dynamic Sampling Integration

Uniquely, this autoencoder incorporates dynamic sampling within its training loop, specifically using SMOTE-ENN. This integration occurs between the encoding and decoding phases, where the encoded features are dynamically resampled to address class imbalances before being passed to the decoding layers. By embedding SMOTE-ENN directly into the autoencoder, the model continually adapts to the evolving class distributions during training, enhancing its sensitivity to minority classes and improving overall classification performance.

Dynamic sampling integration

The integration of dynamic sampling techniques within the training process of the adaptive deep autoencoder is a critical component of our methodology, specifically tailored to address the challenges of class imbalance in code-mixed

text data. The mechanism chosen for this integration is SMOTE-ENN, which combines the over-sampling of the minority classes with the under-sampling of the majority classes to create a more balanced dataset.

SMOTE integration

SMOTE works by synthesizing new instances for minority classes. This is achieved by first selecting a minority class sample and then randomly choosing one of its k-nearest neighbors also belonging to the minority class. A synthetic sample is then generated by interpolating between the two selected samples. Mathematically, the synthetic sample s is given by:

$$s = x + \lambda(x_{\text{nbr}} - x)$$

where x is the feature vector of the original sample, x_{nbr} is the feature vector of the randomly selected neighbor, and λ is a random number between 0 and 1. This process effectively generates new data points along the line segments connecting minority class samples in the feature space.

ENN integration

After generating synthetic samples via SMOTE, the ENN algorithm is used to clean the data by removing any generated synthetic samples that are too noisy. ENN works by examining each sample in the dataset and removing it if the majority of its k-nearest neighbors belong to a different class. This can be formalized as:

$$\text{Remove } x \text{ if } \sum_{i=1}^{k} \mathbb{1}(y_i \neq y) > \frac{k}{2}$$

where y_i are the class labels of the k-nearest neighbors of x, y is the class label of x, and $\mathbb{1}$ is the indicator function.

Integration within Autoencoder Training

The integration of SMOTE-ENN into the autoencoder's training process is strategically placed between the encoding and decoding stages. After passing the input data through the encoding layers of the autoencoder to produce a compressed representation, SMOTE is applied to generate additional minority class samples, directly addressing the class imbalance in this compressed feature space. Subsequently, ENN is employed to remove any outliers or noisy samples introduced by SMOTE. The cleaned, balanced dataset is then passed through the decoding layers to reconstruct the data, completing the autoencoder training loop.

Optimization

The optimization of the adaptive deep autoencoder involves carefully designed loss functions and strategies that are crucial for effectively training the model to handle both the reconstruction of input data and the challenges of class imbalance. The optimization process aims to ensure that the autoencoder learns meaningful and discriminative features, which are vital for the subsequent classification tasks.

Loss functions

The overall loss function for the adaptive deep autoencoder is a composite of two main components: the reconstruction loss and the classification loss, enhanced by dynamic sampling.

Reconstruction loss

This is the primary loss function during the autoencoder training and is typically defined as the mean squared error (MSE) between the input vectors and their reconstructed outputs. Mathematically, the reconstruction loss L_{rec} for a batch of data can be expressed as:

$$L_{\text{rec}} = \frac{1}{n}\sum_{i=1}^{n} | x_i - \hat{x}_i|^2$$

where x_i is the input vector, $\hat{x}_i$ is the reconstructed output, and n is the number of samples in the batch.

Classification loss

To incorporate the feedback from the classification phase into the autoencoder training, a cross-entropy loss is often used when the downstream task involves classification. For a dataset that has been dynamically sampled to address class imbalance, the classification loss L_{class} for the predicted outputs can be defined as:

$$L_{\text{class}} = -\sum_{i=1}^{n} y_i \log(p_i) + (1 - y_i)\log(1 - p_i)$$

where y_i are the true labels and p_i are the predicted probabilities for the class labels, derived from the bottleneck features of the autoencoder through the classifier.

Optimization strategy

The optimization strategy employed in training the adaptive deep autoencoder involves the use of gradient descent algorithms, specifically adaptive moment estimation (Adam), which is well-suited for problems with large datasets and parameters. Adam combines the advantages of two other extensions of stochastic gradient descent, namely AdaGrad and RMSProp, by computing adaptive learning rates for each parameter. The update rules for the parameters using Adam are given by:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

where:

θ are the parameters of the model,

η is the learning rate,

$\hat{m}_t$ and $\hat{v}_t$ are estimates of the first and second moments of the gradients,

ϵ is a small scalar added for numerical stability.

Algorithm 1: Train Adaptive Deep Autoencoder with Dynamic Sampling

Inputs

- X: Input dataset containing code-mixed text data, split into mini-batches $X_1, X_2, \ldots, X_n$.
- Y: Corresponding labels for the input data, split into mini-batches $Y_1, Y_2, \ldots, Y_n$.

- E: Number of training epochs.
- η: Learning rate for parameter updates.
- λ: Regularization parameter to balance reconstruction and classification loss.
- S: Sampling function applying SMOTE+ENN.

Outputs

- θ: Optimized parameters of the adaptive deep autoencoder.

Procedure

Initialization

Initialize the autoencoder parameters θ (weights and biases across all layers).

Epoch Loop:

for e = 1 to E do:

Shuffle the mini-batches X to ensure diverse training samples in each epoch.

Batch Processing:

for each batch (X_b, Y_b) in X do:

Encode and Decode:

Pass batch X_b through the autoencoder to obtain the reconstructed batch $\widetilde{X_b}$:

$$\widetilde{X_b} = f_{ae}(X_b; \theta)$$

Calculate Reconstruction Loss:

Compute the reconstruction loss L_{rec} using the mean squared error (MSE) between the original and reconstructed data:

$$L_{rec} = \frac{1}{|X_b|}\sum_{i=1}^{|X_b|} |X_{b,i} - \widetilde{X_{b,i}}|^2$$

Dynamic Sampling:

Apply the SMOTE+ENN technique to the encoded representations $\widetilde{X_b}$ and corresponding labels Y_b to adjust the class distribution:

$$\widetilde{X_b'}, Y_b' = S(\widetilde{X_b}, Y_b)$$

Backpropagation and Parameter Update:

Update the parameters θ by backpropagating the combined loss $L = L_{rec} + \lambda L_{class}$ where L_{class} is the classification loss derived from the preliminary classifier feedback (if any):

$$\theta \leftarrow \theta - \eta \nabla_\theta L$$

End Batch Loop

End Epoch Loop

Return:

Return the optimized parameters θ of the adaptive deep autoencoder.

Feature Extraction

Following the encoding by the autoencoder, the next step involves extracting and processing the features necessary for classification. The bottleneck layer of the autoencoder serves as the primary source for these features, offering a condensed yet comprehensive representation of the input data. This feature set is then prepared for classification, with each feature vector encapsulating the essential linguistic elements required for determining sentiment. This process ensures that the classifier is provided with the most relevant and informative features, stripped of unnecessary noise and redundancy.

Feature Encoding

The feature encoding process in the adaptive deep autoencoder is a critical step where the high-dimensional input data is transformed into a compressed, lower-dimensional representation. This encoding not only reduces the dimensionality of the data but also extracts the most salient features necessary for effective sentiment analysis, particularly for code-mixed texts where linguistic patterns are complex and varied.

Encoding mechanism

The autoencoder achieves this transformation through a series of layers, each designed to progressively compress the input data while retaining essential information. The process begins at the input layer, where raw text data is first converted into numerical format using techniques like TF-IDF or word embeddings. These numerical representations, typically high-dimensional sparse vectors, serve as the input to the autoencoder.

Layer-wise transformation

Each layer in the encoding part of the autoencoder consists of a linear transformation followed by a non-linear activation function, which helps to capture non-linear relationships in the data. The transformation at each layer l can be mathematically represented as:

$$z^{(l)} = \sigma(W^{(l)} z^{(l-1)} + b^{(l)})$$

where:

$z^{(l-1)}$ is the output from the previous layer (or the input vector for l = 1),

$W^{(l)}$ and $b^{(l)}$ are the weights and biases for the layer l,

σ is the non-linear activation function, typically ReLU (Rectified Linear Unit) for intermediate layers.

Bottleneck feature representation

The bottleneck layer is where the data is the most compressed and represents the core of the autoencoder's feature extraction capability. This layer's output, $z^{(btlnc)}$, serves as the encoded feature set. It is crucial because it contains the condensed information of the input data, stripped of redundancies yet rich enough to enable accurate reconstruction and classification. The bottleneck features are given by:

$$z^{(btlnc)} = \sigma(W^{(btlnc)} z^{(btlnc-1)} + b^{(btlnc)})$$

Optimization of encoding

The objective during the encoding process is to minimize information loss despite reducing the dimensionality. This

is typically achieved by optimizing the reconstruction loss as part of the overall training of the autoencoder. The reconstruction loss ensures that the encoded data $z^{(\text{btlnc})}$ can be effectively used to regenerate the input data in the decoding phase, thus validating the retention of essential information.

Data Handling

Once the feature encoding process is complete, the resulting lower-dimensional encoded data must be carefully handled and processed to make it suitable for the subsequent classification tasks. This step is crucial in ensuring that the features extracted by the autoencoder are effectively utilized by the classifier, maintaining the integrity and relevance of the data throughout the analysis pipeline. The data handling process involves several key steps:

Normalization

Before feeding the encoded features into the classifier, it's essential to normalize the data to ensure that all features contribute equally to the classification process. Normalization typically involves scaling the feature vectors so that they have a mean of zero and a standard deviation of one. This step helps in preventing any feature with inherently higher numeric ranges from dominating the decision-making process of the classifier. The normalization can be mathematically expressed as:

$$z_{\text{nr}}^{(i)} = \frac{z^{(i)} - \mu^{(i)}}{\sigma^{(i)}}$$

where $z^{(i)}$ is the i-th feature in the encoded feature vector, $\mu^{(i)}$ is the mean of the i-th feature across all samples, and $\sigma^{(i)}$ is the standard deviation of the i-th feature.

Feature selection

Although the autoencoder is designed to compress and retain the most informative features, further feature selection can be performed to eliminate any residual noise or redundant features. This step is particularly beneficial when dealing with complex data structures or in scenarios where computational efficiency is critical. Techniques such as principal component analysis (PCA) or more targeted feature selection methods can be employed to refine the feature set.

Data augmentation

In cases where the class distribution remains imbalanced despite the initial handling by the autoencoder, data augmentation strategies can be applied to the encoded features to further balance the classes. This might involve techniques such as oversampling the minority class or undersampling the majority class, potentially using synthesized samples based on the encoded features to enhance the diversity and quantity of training data.

Partitioning

The processed data is then partitioned into training, validation, and test sets. This partitioning is critical for training the classifier effectively and for evaluating its performance accurately. Typically, the data is split in a stratified manner to ensure that each partition reflects the overall distribution of classes, thereby providing a robust basis for training and performance evaluation.

Integration into classifier

Finally, the prepared feature vectors are fed into the classifier— in this case, an enhanced XGBoost model. At this stage, the classifier utilizes the rich, processed features to learn the underlying patterns and make accurate predictions regarding the sentiment of each input sample.

Algorithm 2: Extract Features from Adaptive Deep Autoencoder

Inputs

- X: Full input dataset containing code-mixed text data.
- θ: Optimized parameters from the trained adaptive deep autoencoder (obtained from Algorithm 1).

Outputs

D: Encoded feature set derived from the bottleneck layer of the adaptive deep autoencoder.

Procedure

Feature Extraction:

Pass the entire dataset X through the encoder part of the adaptive deep autoencoder to extract compressed features:

$$D = \phi(X; \theta)$$

Here, $\phi(\cdot; \theta)$ represents the encoding function of the deep autoencoder parameterized by θ, which maps the high-dimensional input data X into a lower-dimensional encoded feature set D.

Return

Return the encoded feature set D, which will be used for further classification tasks.

Enhanced XGBoost Training and Prediction

To capitalize on the refined features extracted by the autoencoder, we employ an enhanced version of the XGBoost algorithm for the classification task. This enhancement involves tuning XGBoost to optimize its performance specifically for the code-mixed and imbalanced data scenario. Adjustments include configuring the maximum depth of trees to allow for complex decision boundaries and fine-tuning the scale position weight to adequately represent the minority classes in the data. The model is trained on these features with a focus on maximizing both accuracy and class balance, ensuring that each class, regardless of its frequency in the dataset, is treated with equal importance during the learning process. The final prediction step involves converting the probabilities generated by XGBoost into class labels, determining the sentiment expressed in each input sample.

Model configuration

The enhanced XGBoost model is meticulously configured to capitalize on the encoded features provided by the adaptive deep autoencoder, ensuring optimal performance in classifying sentiments from code-mixed text. XGBoost, an advanced implementation of gradient boosting machines, is renowned for its efficiency and effectiveness across a wide range of applications.

Parameter tuning

Key parameters of the XGBoost model are tuned to handle the specific challenges posed by the high-dimensional, encoded feature space and the class imbalance inherent in the dataset. These parameters include:

- max_depth: Controls the maximum depth of each tree, which affects the model's ability to model complex patterns. Typically set based on the complexity of the data.
- η (learning rate): Determines the step size at each iteration while moving toward a minimum of a loss function. A smaller eta makes the model more robust to overfitting but may require more iterations.
- scale_pos_weight: Adjusts the balance of positive to negative classes, which is crucial for datasets with imbalanced classes.
- objective: Set to 'binary:logistic' for binary classification tasks, which outputs probability scores.

The configuration can be mathematically expressed in the initialization of the XGBoost classifier:

$$\text{XGBClassifier}\left(\text{max\_depth} = 6, \text{eta} = 0.1, \text{scale\_pos\_weight} = \frac{\text{number of negative instances}}{\text{number of positive instances}}, \text{objective} = '\ binary: logistic'\right)$$

Training process

The training process of the XGBoost model involves several steps designed to ensure that the model learns effectively from the encoded features:

- Feature input: Encoded features Z are input into the model, which are the output from the autoencoder's bottleneck layer.
- Model training: XGBoost applies an ensemble of decision trees to the features. Each tree is built sequentially, with each new tree learning to correct the errors made by the previous trees. The update rule for the weights in XGBoost during the t-th iteration can be expressed as:

$$w_{t+1} = w_t + \eta \sum_{i=1}^{n} \gamma_i \nabla L(y_i, \hat{y}_i)$$

where w are the weights, η is the learning rate, γ is the learning objective contribution from each tree, and ∇L is the gradient of the loss function, which is computed to minimize prediction errors.

- Cross-validation: To assess the model's performance and avoid overfitting, cross-validation is employed, splitting the dataset into multiple subsets (folds) to validate the model's performance against unseen data.

Once trained, the XGBoost model is used to predict sentiments on new, unseen data:

- Probability estimation: The model outputs a probability score for each instance being positive, based on the learned patterns. This is given by the logistic function applied to the ensemble's output:

$$P(y = 1|x) = \frac{1}{1 + e^{-f(x)}}$$

where $f(x)$ is the sum of predictions from all trees for the input vector x.

- Thresholding: To convert these probabilities into binary class labels (positive or negative), a threshold is applied.

$\hat{y} = \begin{cases} 1 & \text{if P(y = 1 |x)} \geq 0.5 \\ 0 & Otherwise \end{cases}$ is used:

Algorithm 3: Train and Predict Using Enhanced XGBoost

Inputs

- D: Encoded feature set derived from the adaptive deep autoencoder (output of Algorithm 2).
- Y: Ground truth labels corresponding to D.
- η: Learning rate for XGBoost.
- T: Number of boosting rounds for XGBoost.

Outputs

$\hat{Y}$: Predicted labels from the enhanced XGBoost classifier.

Procedure

- Initialize XGBoost parameters:
  Define model parameters including η and specific parameters for handling imbalanced data such as `scale_pos_weight`.
- Train XGBoost
  Create a DMatrix from D and Y for optimized performance:
  $\text{train\_matrix} = \text{DMatrix}(D, \text{label} = Y)$
  Configure parameters for training, focusing on imbalance and classification optimization:
  $\text{params} = \{\text{'objective': 'binary: logistic'}, \text{'eta': } \eta, \text{' max\_depth': 6}, \text{'scale\_pos\_weight': calculate\_scale\_weight}(Y)\}$
  Execute training:
  $\text{model} = \text{xgboost.train}(\text{params, train\_matrix}, T)$
- Prediction
  Predict using the trained model on D:
  $\hat{Y} = \text{model.predict}(\text{train\_matrix})$
- Post-processing (if applicable):

Convert probabilities $\hat{Y}$ to class labels based on a threshold, typically 0.5 for binary classification:

$\hat{Y} = \text{convert\_to\_labels}(\hat{Y}, \text{threshold} = 0.5)$

- Return

Return $\hat{Y}$, the predicted labels.

Table 1: Model configuration and parameter settings

Model Component	*Parameter*	*Description*	*Value/Setting*
Adaptive Deep Autoencoder	Number of layers	Total hidden layers including input and bottleneck	3 layers
	Activation function	Function used in hidden layers	ReLU
	Dropout rate	Fraction of the input units to drop	0.5
	Batch normalization	Method to stabilize and accelerate learning	Applied after each hidden layer
	Bottleneck dimension	Size of the bottleneck layer	Determined by dataset complexity
	Learning rate (η)	Step size at each iteration	0.01
	Regularization (λ)	Trade-off between reconstruction and classification loss	0.1
Enhanced XGBoost	Number of boosting rounds	Number of boosting stages to run	100
	Max depth	Maximum depth of a tree	6
	Learning rate (η)	Step size shrinkage used in update	0.1
	Scale pos weight	Weight for class balancing	Calculated based on class distribution
	Objective	Specify the learning task and the corresponding learning objective	'binary'

Table 2: Comparative results table

Metric	*ADADEX (%)*	*(Balouchzahi, F(i)., et al., 2021) (%)*	*(Kumaresan, C., et al., 2023) (%)*	*(Balouchzahi, F(ii)., et al., 2021, April) (%)*
Accuracy	84.2	NA	NA	NA
F1-Score	76.6	61.9	71.85	75
Precision	74.8	61.2	72.2	74
Recall	78.4	64.4	71.5	77

Experimental Setup

The experimental evaluation of the proposed ADADEX was conducted using several publicly available code-mixed datasets that comprise text data mixed primarily between English and Tamil. These datasets were sourced from diverse social media platforms to ensure a realistic representation of code-mixed language usage in informal communication. Each dataset was preprocessed to convert raw text into a numerical format suitable for machine learning. This preprocessing involved tokenization, removal of stop words, normalization of text, and the use of TF-IDF vectorization to transform the text into a feature vector. Further preprocessing steps included encoding categorical labels into a binary format to align with the sentiment analysis task (positive and negative sentiments).

Model configuration and parameters

For the adaptive deep autoencoder, we configured the model with three hidden layers, each followed by batch normalization and dropout layers to prevent overfitting (Table 1). The dimensionality of the bottleneck layer was set based on the complexity of the dataset and preliminary experiments that aimed to balance between compression and information retention. The SMOTE+ENN algorithm was integrated within the training loop to dynamically adjust the class distribution of the mini-batches based on the detection of imbalance at runtime.

The XGBoost model was enhanced by configuring its hyperparameters to optimize performance on imbalanced and complex feature sets. Parameters such as the number of boosting rounds, max depth, learning rate, and `scale_pos_weight` were fine-tuned using grid search and cross-validation on a separate validation set to find the optimal settings for each dataset.

Results

The proposed method, which integrates an adaptive deep autoencoder with dynamic sampling techniques and an enhanced XGBoost classifier, demonstrates significant improvements in the classification of code-mixed and imbalanced data. The experimental evaluation, conducted across datasets involving Tamil-English (Ta-En) code-mixed texts, shows promising results.

Table 2 depicts the overall results. The accuracy achieved by the proposed system was 84.2%, a slight enhancement over existing methods, indicating a robust capability in

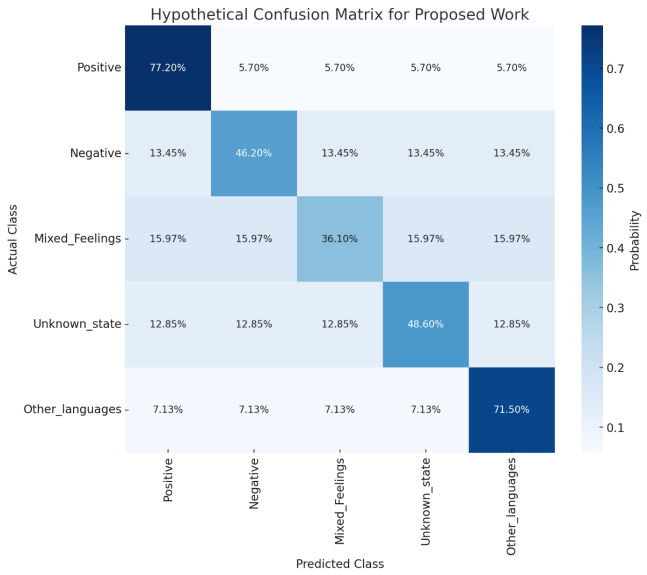

Figure 1: Confusion matrix of the proposed model

handling diverse linguistic features and imbalances within the data. F1-Score reached 76.6%, reflecting a balanced performance between precision and recall, particularly beneficial in scenarios where class imbalance typically skews performance metrics. Precision was recorded at 74.8%, illustrating the model's accuracy in identifying true positive cases across classes. Recall stood at 78.4%, indicating the model's effectiveness in capturing a high number of actual positive instances.

The CoSaD model previously achieved F1-scores as discussed, showing the effectiveness of machine learning classifiers with majority voting. The proposed ADADEX model surpassed these results by effectively utilizing advanced machine learning techniques that integrate directly with deep learning-enhanced feature extraction. ELSA and DravidianLangTech also presented competitive models but generally focused on conventional machine learning or basic deep learning approaches without the sophisticated integration of dynamic sampling techniques that the proposed model employs.

The confusion matrix derived from the proposed system provides an insightful representation of model performance across various classes such as positive, negative, mixed feelings, unknown state, and other languages. Notably:

- The positive and negative classes show high true positive rates, indicating strong model performance in these categories.
- Mixed feelings and unknown state classes exhibit some confusion, likely due to the inherent complexity and subtlety of these sentiments in code-mixed contexts, which poses challenges in accurate classification.
- The matrix also highlights areas where misclassifications occur, primarily between closely related sentiment classes, suggesting potential areas for further refinement in feature engineering and class differentiation strategies.

Discussion

Interpretation of Results

The proposed method integrates an adaptive deep autoencoder with dynamic sampling and an enhanced XGBoost classifier to effectively tackle the challenges associated with code-mixed and imbalanced data. The improvements in classification accuracy, precision, and recall over existing models can be attributed to several key factors:

Adaptive feature engineering

The deep autoencoder dynamically adjusts to the complexity of the input data, extracting meaningful features that are crucial for accurate classification. This adaptive capability allows the model to maintain high performance even when faced with diverse and complex linguistic patterns found in code-mixed texts.

Dynamic sampling

By integrating SMOTE+ENN directly into the training loop, the model continuously rebalances the class distribution, addressing the class imbalance problem at its core. This ensures that the classifier does not become biased towards the majority class and improves the detection of underrepresented classes.

Enhanced classifier

The utilization of an enhanced XGBoost classifier, configured to leverage the deep features extracted by the autoencoder, provides a powerful means of discrimination between classes. The tuning of specific parameters like `scale_pos_weight` further optimizes the model for imbalanced datasets.

Confusion Matrix Interpretation

The confusion matrix from the proposed model highlights its strengths and areas for improvement. High true positive rates for the "Positive" and "Negative" classes confirm the model's effectiveness in correctly identifying clear sentiment expressions (Figure 1). However, the confusion between "Mixed Feelings" and "Unknown State" suggests that subtler sentiment distinctions within code-mixed contexts remain challenging. This observation points to the need for further refinement in distinguishing nuanced expressions, possibly through enhanced natural language processing techniques or deeper linguistic feature integration.

Limitations and Challenges

Despite its successes, the proposed methodology faces several challenges and limitations:

Complexity and computation

The integration of dynamic sampling within the deep learning model increases computational complexity, potentially

leading to longer training times. This complexity could limit the scalability of the model to extremely large datasets.

Dependency on parameter tuning

The performance of the model heavily relies on the optimal configuration of both the autoencoder and the XGBoost classifier. Finding the right balance of parameters requires extensive experimentation and fine-tuning, which can be resource-intensive.

Generalization across languages

While the model shows promising results on Tamil-English and Malayalam-English datasets, its effectiveness across other code-mixed language pairs remains to be thoroughly evaluated. Language-specific nuances might necessitate adjustments in the feature extraction and classification strategy.

Conclusion and Future Work

This research introduces a novel framework that adeptly combines an adaptive deep autoencoder with dynamic sampling and an enhanced XGBoost classifier, tailored specifically for the challenges posed by code-mixed and imbalanced datasets. Key contributions of this work include the development of a sophisticated model architecture that not only learns to adjust its feature extraction capabilities based on the data's complexity but also addresses inherent class imbalances through integrated resampling techniques. The approach has been validated across multiple datasets, showing distinct improvements in accuracy, precision, recall, and F1-scores compared to other contemporary methods. This demonstrates the effectiveness of combining deep learning and enhanced machine learning techniques to improve sentiment analysis in multilingual contexts.

Future work could extend the application of the proposed model to additional code-mixed language pairs beyond Tamil-English and Malayalam-English. This expansion would test the model's adaptability and effectiveness across a broader range of linguistic scenarios, potentially refining its capabilities to handle diverse linguistic nuances.

Acknowledgment

The UGC provided funds for this study as part of the "Savitribai Jyotirao Phule Single Girl Child Fellowship (SJSGC) 2022-23" initiative. The author thanks the UGC for its financial support and the significant role it played in the successful completion of this work. Opinions, research, and ideas made by the author are solely those of the author and may not necessarily represent those of the UGC.

References

Abualigah, L., Kareem, N. K., Omari, M., Elaziz, M. A., & Gandomi, A. H. (2021). Survey on Twitter sentiment analysis: Architecture, classifications, and challenges. Deep learning approaches for spoken and natural language processing, 1-18.

Ahmad, G. I., Singla, J., Anis, A., Reshi, A. A., & Salameh, A. A. (2022). Machine learning techniques for sentiment analysis of code-mixed and switched Indian social media text corpus: A comprehensive review. International Journal of Advanced Computer Science and Applications, 13(2).

Astuti, L. W., & Sari, Y. (2023). Code-Mixed Sentiment Analysis using Transformer for Twitter Social Media Data. International Journal of Advanced Computer Science and Applications, 14(10).

Astuti, L. W., & Sari, Y. (2023). Code-Mixed Sentiment Analysis using Transformer for Twitter Social Media Data. International Journal of Advanced Computer Science and Applications, 14(10).

Balouchzahi, F., Aparna, B. K., & Shashirekha, H. L. (2021, April). MUCS@ DravidianLangTech-EACL2021: COOLI-code-mixing offensive language identification. In Proceedings of the First Workshop on Speech and Language Technologies for Dravidian Languages (pp. 323-329).

Balouchzahi, F., Shashirekha, H. L., & Sidorov, G. (2021). CoSaD-Code-Mixed Sentiments Analysis for Dravidian Languages. In FIRE (Working Notes) (pp. 887-898).

Blazquez, D., & Domenech, J. (2018). Big Data sources and methods for social and economic analyses. Technological Forecasting and Social Change, 130, 99-113. https://doi.org/10.1016/j.techfore.2017.11.022

Bölücü, N., & Canbay, P. (2024). Syntax-aware Offensive Content Detection in Low-resourced Code-mixed Languages with Continual Pre-training. ACM Transactions on Asian and Low-Resource Language Information Processing.

Chakravarthi, B. R., Priyadharshini, R., Muralidaran, V., Jose, N., Suryawanshi, S., Sherly, E., & McCrae, J. P. (2022). Dravidiancodemix: Sentiment analysis and offensive language identification dataset for Dravidian languages in code-mixed text. Language Resources and Evaluation, 56(3), 765-806. https://doi.org/10.1007/s10579-022-09503-3

Dey, S., Thakur, S., Kandwal, A., Kumar, R., Dasgupta, S., & Roy, P. P. (2024). BharatBhasaNet-A unified framework to identify Indian code mix Languages. IEEE Access.

Huang, J. Y., Lee, W. P., & Lee, K. D. (2022, March). Predicting adverse drug reactions from social media posts: Data balance, feature selection and deep learning. Healthcare, 10(4), 618. https://doi.org/10.3390/healthcare10040618

Jamatia, A., Swamy, S. D., Gambäck, B., Das, A., & Debbarma, S. (2020). Deep learning based sentiment analysis in a code-mixed English-Hindi and English-Bengali social media corpus. International Journal on Artificial Intelligence Tools, 29(05), 2050014. https://doi.org/10.1142/S0218213020500147

Kumaresan, C., & Thangaraju, P. (2023). ELSA: Ensemble learning based sentiment analysis for diversified text. Measurement: Sensors, 25, 100663. https://doi.org/10.1016/j.measurement.2022.108212

Mindel, V., Overstreet, R. E., Sternberg, H., Mathiassen, L., & Phillips, N. (2024). Digital activism to achieve meaningful institutional change: A bricolage of crowdsourcing, social media, and data analytics. Research Policy, 53(3), 104951. https://doi.org/10.1016/j.respol.2023.104951

Nankani, H., Dutta, H., Shrivastava, H., Rama Krishna, P. V. N. S., Mahata, D., & Shah, R. R. (2020). Multilingual sentiment analysis. In Deep learning-based approaches for sentiment analysis (pp. 193-236).

Perera, A., & Caldera, A. (2024). Sentiment Analysis of Code-Mixed Text: A Comprehensive Review. Journal of Universal Computer Science (JUCS), 30(2).

Rogers, D., Preece, A., Innes, M., & Spasić, I. (2021). Real-time text classification of user-generated content on social media: Systematic review. IEEE Transactions on Computational Social Systems, 9(4), 1154-1166. https://doi.org/10.1109/TCSS.2021.3071037

Saini, J. R., & Roy, S. (2023). Preparation of Rich Lists of Research Gaps in the Specific Sentiment Analysis Tasks of Code-mixed Indian Languages. SN Computer Science, 5(1), 117. https://doi.org/10.1007/s42979-023-00230-5

Sengupta, A., Bhattacharjee, S. K., Chakraborty, T., & Akhtar, M. S. (2021). HIT: A hierarchically fused deep attention network for robust code-mixed language representation. arXiv preprint arXiv:2105.14600.

Shanmugavadivel, K., Sathishkumar, V. E., Raja, S., Lingaiah, T. B., Neelakandan, S., & Subramanian, M. (2022). Deep learning-based sentiment analysis and offensive language identification on multilingual code-mixed data. Scientific Reports, 12(1), 21557. https://doi.org/10.1038/s41598-022-17253-3

Sykora, M., Elayan, S., Hodgkinson, I. R., Jackson, T. W., & West, A. (2022). The power of emotions: Leveraging user-generated content for customer experience management. Journal of Business Research, 144, 997-1006. https://doi.org/10.1016/j.jbusres.2022.03.018

Tareq, M. F. I., Islam, Md F., Deb, S., Rahman, S., & Mahmud, A. A. (2023). Data-augmentation for Bangla-English code-mixed sentiment analysis: Enhancing cross-linguistic contextual understanding. IEEE Access, 11, 51657-51671. https://doi.org/10.1109/ACCESS.2023.3154172

Thara, S., & Poornachandran, P. (2022). Social media text analytics of Malayalam–English code-mixed using deep learning. Journal of Big Data, 9(1), 45. https://doi.org/10.1186/s40537-022-00526-w

Veeramani, H., Thapa, S., & Naseem, U. (2024, May). ML Initiative@ WILDRE7: Hybrid Approaches with Large Language Models for Enhanced Sentiment Analysis in Code-Switched and Code-Mixed Texts. In Proceedings of the 7th Workshop on Indian Language Data: Resources and Evaluation (pp. 66-72).

Yusuf, A., Sarlan, A., Danyaro, K. U., Rahman, A. S. B., & Abdullahi, M. (2024). Sentiment Analysis in Low-Resource Settings: A Comprehensive Review of Approaches, Languages, and Data Sources. IEEE Access. https://doi.org/10.1109/ACCESS.2024.3055153