A project report on

Advancing Customer Churn Prediction through Hybrid Deep Learning Architectures and Explainable Models

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning

by

N U PRANEETH REDDY (21BAI1500)

DEPATLA HEMIKA REDDY (21BAI1538)

KARNAM NITHIN (21BAI1091)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

November,2024

Advancing Customer Churn Prediction through Hybrid Deep Learning Architectures and Explainable Models

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology in Computer Science and Engineering with Specialization in Artificial Intelligence and Machine Learning

by

N U PRANEETH REDDY (21BAI1500)

DEPATLA HEMIKA REDDY (21BAI1538)

KARNAM NITHIN (21BAI1091)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

November, 2024

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

DECLARATION

I hereby declare that the thesis entitled "**Advancing Customer Churn Prediction through Hybrid Deep Learning Architectures and Explainable Models**" submitted by **Karnam Nithin(21BAI1091)**, for the award of the degree of Bachelor of Technology in Computer Science with Specialization in Artificial Intelligence and Machine Learning, Vellore Institute of Technology, Chennai is a record of Bonafide work carried out by me under the supervision of Dr. Sudheer Kumar E.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Chennai

Date: **Signature of the Candidate**

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

School of Computer Science and Engineering

CERTIFICATE

This is to certify that the report entitled "**Advancing Customer Churn Prediction through Hybrid Deep Learning Architectures and Explainable Models**" is prepared and submitted by **Karnam Nithin(21BAI1091)** to Vellore Institute of Technology, Chennai, in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science with Specialization in Artificial Intelligence and Machine Learning is a Bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Signature of the Guide:

Name: Dr. Sudheer Kumar E

Date:

Signature of the Examiner	Signature of the Examiner
Name:	Name:
Date:	Date:

Approved by the Head of Department,
B.Tech. CSE with Specialization in Artificial Intelligence and Machine Learning

Name: Dr. Sweetlin Hemalatha
Date:

ABSTRACT

Customer Churn poses a significant challenge to businesses, affecting revenue and customer retention strategies. Traditional methods of churn forecasting tend to rely on basic mathematical models, which do not adequately capture the complex, multifaceted behavior of customers This study presents an advanced approach to churn forecasting using deep learning models using control, especially multi-input neural networks (NN) come deep neural networks (DNN) a sequencing (Seq2Seq) model, and a scheme focusing on hybrid CNN-Seq2Seq. This model is designed to handle structured data, such as demographic and behavioral data, as well as unstructured data, including customer feedback and transaction history, allowing for a detailed analysis of factors contributing to churn also, provides training and provides a solid basis for testing the performance of the model.

The work begins with important data preprocessing steps, such as data cleaning, normalization, and vectorization of text features, to ensure consistency of models and then refine each model to extract meaningful patterns: For dimensional data in which the Seq2Seq model, excelling in identifying complex nonlinear patterns, is ideally suited to analyze sequences of receptor interactions, thereby capturing temporal dynamics which may indicate churn. In addition, the hybrid CNN-Seq2Seq model takes advantage of the feature extraction capabilities of Convolutional Neural Networks (CNN) combined with the sequential processing capabilities of Seq2Seq, for enhanced predictive accuracy and resilience.

Experimental results show that the deep learning model performs better than traditional churn prediction methods, resulting in higher accuracy, precision, F1-score, and recall Notably, the Hybrid CNN-Seq2Seq model achieved accuracy a highest, indicating robustness in handling valuable mixed data types , especially To highlight the ability of those with architectures designed for complex data fusion to act as a reliable tool in churn prediction the proposed method enables timely identification of those at risk, thereby improving retention strategies and sustainable performance .Furthermore, the flexibility and scalability of this model make it a valuable asset in a dynamic business environment, enabling organizations to better understand and respond to changing customer behavior This study builds on change the impact of deep learning to develop an accurate, scalable churn forecasting system Helps drive growth and competitive advantage in the marketplace.

ACKNOWLEDGEMENT

It’s my pleasure to express with deep sense of gratitude to Dr.Sudheer kumar E, Assistant Professor, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of Deep Learning.

It is with gratitude that I would like to extend my thanks to the visionary leader Dr. G. Viswanathan our Honorable Chancellor, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam Vice Presidents, Dr. Sandhya Pentareddy, Executive Director, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr. V. S. Kanchana Bhaaskaran Vice-Chancellor i/c & Pro-Vice Chancellor, VIT Chennai and Dr. P. K. Manoharan, Additional Registrar for providing an exceptional working environment and inspiring all of us during the tenure of the course.

Special mention to Dr. Ganesan R, Dean, Dr. Parvathi R, Associate Dean Academics, Dr. Geetha S, Associate Dean Research, School of Computer Science and Engineering, Vellore Institute of Technology, Chennai for spending their valuable time and efforts in sharing their knowledge and for helping us in every aspect.

In jubilant state, I express ingeniously my whole-hearted thanks to Dr. Sweetlin Hemalatha C, Head of the Department, B.Tech. CSE with Specialization in Artificial Intelligence and Machine Learning and the Project Coordinators for their valuable support and encouragement to take up and complete the thesis.

My sincere thanks to all the faculties and staffs at Vellore Institute of Technology, Chennai who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

Place: Chennai

Date: **Karnam Nithin**

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

LIST OF EQUATIONS

Chapter 1

Introduction

1.1 INTRODUCTION

In recent times, the swift increase in data generation and the intricacy of real-world challenges have catalyzed the advancement of sophisticated machine learning models that can manage high dimensional, diverse datasets. In critical applications like customer retention, healthcare, and financial forecasting predictive accuracy is crucial, but model interpretability and transparency are also important. Although deep learning models are effective at identifying intricate patterns, their" black-box" characteristic creates difficulties for interpretability, potentially restricting their use in fields where comprehending decision-making processes is essential. As a result, creating models that achieve a balance between strong performance and explainability has emerged as an urgent issue in machine learning studies.

This document presents a new strategy that integrates a Hybrid Convolutional Neural Network and Sequence Embedding (HybridCNN-Seq2Seq) framework, aimed at boosting model resilience and optimizing feature extraction in predicting customer churn. The HybridCNN-Seq2Seq model utilizes convolutional neural networks (CNNs) to efficiently extract spatial features and sequence embedding (Seq2Seq) layers to identify temporal and sequential patterns in data. This type of hybrid architecture allows the model to recognize subtle patterns that conventional models might miss, leading to increased predictive accuracy.

Additionally, to fulfill the requirement for transparency, we utilize an Explainable Boosting Classifier (EBC) to produce interpretable global and local insights into the model's predictions. EBC offers an understanding of how particular characteristics affect predictions throughout the dataset and for individual cases, allowing stakeholders to make informed choices based on the model's reasoning. This emphasis on both predictive capability and interpretability facilitates the use of machine learning models in practical situations, where accountability and comprehension of model actions are crucial.

This paper offers three key contributions:

- Firstly, we introduce the HybridCNN-Seq2Seq model as an effective method for predicting customer churn.
- Additionally, we incorporate explainability via EBC, offering clarity regarding the model's decision-making procedures.
- Third, we carry out an extensive assessment, showing that our model not only enhances prediction accuracy but also provides understandable insights.

This study seeks to enhance interpretable machine learning by connecting intricate model structures with the necessity for clarity, thereby promoting trust and dependability in data informed decision making.

1.2 OVERVIEW OF PROJECT

This project aims to develop an effective system for predicting customer churn using advanced deep learning techniques. It explores both transfer learning with pre-trained models and custom-built neural network architectures, including Multi-Input Neural Networks, Deep Neural Networks, Seq2Seq, and Hybrid CNN-Seq2Seq models. By leveraging both structured and unstructured data, this approach captures comprehensive patterns in customer behaviour, enabling accurate churn prediction.

The project's dual approach comparing fine-tuned pre-trained models with custom models provides valuable insights into the most effective method for churn prediction. This system offers businesses a proactive tool to improve customer retention, contributing to better customer relationship management and business sustainability.

1.3 CHALLEGES PRESENT IN PROJECT

- **Data Collection and Processing:** Collecting big, high-quality customer data, including structured attributes and unstructured interaction histories, is important but difficult. Ensuring that data diversity exists across different consumer demographics and behaviours is critical for modelling generalization.

- **Model accuracy and generalization**: Making sure the model generalizes well to different customer profiles and behaviours is important for accurate churn forecasting. The model handles multiple data structures and remains robust in a dynamic business environment.

- **Computational Resources:** Training deep learning models, especially complex algorithms such as Seq2Seq and Hybrid CNN-Seq2Seq requires a lot of computational resources. High-performance computing is essential for effective model training and analysis.

- **Real-time implementation:** Planning for real-time churn forecasting adds challenges, as models need to be optimized for both accurate and rapid forecasting to support timely intervention for at-risk clients

- **Model comparison:** Evaluating the efficiency of a pre-trained model using transfer learning compared to a custom-made model requires not only accuracy but also a comparison factors such as training time, computational requirements, and ease of use will also be compared to determine the most efficient method.

1.4 PROJECT STATEMENT

This project seeks to develop a deep learning-based algorithm for predicting customer churn by analyzing both structured and unstructured customer data. Programs comparing the performance of these models include neural network architectures including multiple input neural networks, deep neural networks, Seq2Seq, and hybrid CNN-Seq2Seq models, with and without transfer learning, to assess performance work well in identifying churn-prone customers The goal is to identify the most accurate and scalable approach for real-world application. The program is designed to assist businesses in early identification of at-risk customers, enable early retention strategies, and strengthen customer relationship management

1.5 OBJECTIVES AND SCOPE OF THE PROJECT

- **Develop an automated forecasting system:** Analyzing structured and unstructured data to create a system that can accurately predict customer churn, giving businesses a reliable tool developing proactive customer retention strategies.

- **Use transfer learning:** Implement and refine pre-trained neural network models, such as multi-input neural networks, Seq2Seq, and hybrid CNN-Seq2Seq model types, and use your prior knowledge for effective churn forecasting. Transfer learning helps reduce training time and computational resources while maintaining high prediction accuracy.

- **Customized deep learning models:** Customized models from scratch to compare their performance against pre-trained models. A standardized model allows for architectural flexibility that can be better suited to capture complex customer behaviours.

- **Evaluate model performance:** Conduct detailed analysis to compare the accuracy, efficiency, and scalability of transfer-learning models with custom models. This analysis will include metrics such as accuracy, recall, F1 score, and computation time to determine the most effective method.

- **Deploy for Practical Use:** Develop systems for real-world implementation, making it feasible for businesses to use. The deployment process will emphasize user friendliness, reliability, and compatibility with existing customer relationship management systems, and ensure that the system is deployed in a dynamic business environment

1.6 CONTRIBUTION TO THE PROJECT

- **Provide comparative analysis:** Provide insights into the effectiveness of transfer learning for customer churn forecasting. This comparison will inform future research and useful applications in churn audit processes, helping companies choose the best approach based on specific needs.

- **Enhancing Customer Retention:** Enable accurate and timely identification of at-risk customers, allowing firms to engage more actively. Identifying churn risk early can help companies improve inventory levels, ultimately increasing customer loyalty and revenue stability.

- **Automating churn predictions:** Reduces the need to manually analyse customer data, thus saving time and labour while improving the accuracy of churn predictions. Automation allows companies to effectively manage big customer data without human error.

- **Advances in research in consumer behaviour analysis:** contribute to the application of deep learning techniques to consumer behaviour and churn analysis, which can stimulate further research in this area. The findings of the project can be adapted to a wider range of industries and applications, expanding its impact.

- **Dataset enhancement and diversification:** Taking advantage of structured and unstructured data and enhancing the dataset using preprocessing techniques such as normalization and text vectorization. This increases the variety of data and improves the models' ability to generalize to diverse customer profiles.

- **Comparison of different modelling algorithms:** The study provides a detailed comparison of different neural network algorithms (Multi-Input NN, DNN, Seq2Seq, Hybrid CNN-Seq2Seq) for not only prediction accuracy but features such as computer efforts, training etc. also about Time and resources required. This study can guide future work in selecting the best model for specific business needs and constraints.

Chapter 2

Related Study

2.1 LITERATURE SURVEY

Irina V. Pustokhina C. Jeyalakshmi e a, *, Denis A. Pustokhin , Vicente García Díaz f b , Aswathy RH , K. Shankar a g proposed a paper that suggests a dynamic customer churn prediction approach that combines evolutionary optimization algorithms with text analytics. Using LSTM and a Stacked Autoencoder (SAE) for feature selection, the CCPBI-TAMO model initially uses Chaotic Pigeon-Inspired Optimisation (CPIO) for classification. This model achieved a maximum accuracy of 95.56%, a kappa of 94.3%, an F1-score of 94.96%, and an AUC of 0.93 when it was trained and assessed using benchmark churn datasets. Because of its great efficiency, the system can compute predictions in milliseconds, which makes it appropriate for real-time client retention tactics in industries with fierce competition.

A distributed model for customer churn prediction is presented in the research by Tariq et al. using a convolutional neural network (CNN) framework with Apache Spark for parallel processing. Prior to classification, this model handles data preprocessing, balancing, and standardization using a 2D-CNN architecture. After being trained on the Telco Customer Churn dataset, it demonstrated effective churn prediction skills with an accuracy of 96.3%, a true-positive rate of 95%, and a true-negative rate of 94%. The model is extremely appropriate for large-scale e-business systems because of its quick processing time, which allows it to handle real-time customer churn forecasts.

Using word order contextualized semantics on customers' social attitudes, Ibitoye and Onifade's study proposes an improved customer churn prediction model. To improve sentiment analysis accuracy, this model uses a Word Order Fuzzy Support Vector Machine (WOFSVM) to extract contextual meanings and relationships from customer input. With an accuracy of 84.23%, precision of 83.17%, and recall of 85.20% after being trained on a dataset of 65,315 tweets, it demonstrated exceptional performance in sentiment-based churn categorization. Real-time churn forecasts in social media-driven contexts are greatly aided by the model's effectiveness and contextual richness.

By combining hybrid neural networks that incorporate back-propagation artificial neural networks (ANN) and self-organizing maps (SOM), Tsai and Lu present an improved customer churn prediction model. This hybrid model enhances the model's capacity to differentiate between churners and non-churners by using SOM for clustering customer data and ANN for final classification. The model, which was trained on a dataset of 51,306 telecom subscribers, has a 93.7% prediction accuracy and substantially lower Type I and Type II error rates than single-method models. In customer-centric settings, this strong hybrid method works well for churn prediction due to its increased accuracy and stability.

Poudel, Pokharel, and Timilsina's paper focusses on interpretability in telecom churn forecasting and develops an enhanced customer churn prediction model. This model uses Shapley Additive explanations (SHAP) in conjunction with Gradient Boosting Machines (GBM) to explain customer turnover variables both locally and globally. With an accuracy of 81%, precision of 67%, and recall of 55% after being trained on a telecom dataset including 7,043 samples, the GBM model demonstrated its capacity to predict churn while offering valuable insights into key churn determinants. The model is quite useful for focused churn control tactics in cutthroat telecom markets because of its interpretability and predictive capability.

In order to increase predictive performance in unbalanced datasets, Imani and Arabnia's paper introduces an improved customer churn prediction model that makes use of hyperparameter optimization and integrated data sampling strategies. In order to overcome class imbalance, this model incorporates SMOTE with Tomek Links and Edited Nearest Neighbours (ENN) and uses Optuna for hyperparameter adjustment. The model demonstrated good prediction accuracy and robustness to data imbalance after being trained on a telecom dataset of 4,250 occurrences, achieving an F1-score of 92% and a ROC AUC of 91%. The model is quite good at predicting churn in telecom and other subscription-based sectors because of its strong framework and well-chosen sample.

Using Extreme Gradient Boosting (XGBoost) and feature selection based on customer service utilisation and demographic characteristics, Lukita et al. provide an improved customer turnover prediction model. To increase prediction accuracy, the model concentrates on high-impact characteristics like Monthly Charges, Paperless Billing, and Payment Method that were found through heatmap analysis. After being trained on a dataset of bank customers, the XGBoost model demonstrated great performance in predicting possible churn events, with an accuracy of 87%, precision of 84%, and recall of 92%. This model is quite useful for proactive client retention tactics in financial service environments because of its accuracy and feature-based insights.

Pradeep et al. use the methods of Decision Tree and Logistic Regression to present a customer churn prediction model for the logistics sector. In order to improve retention efforts, the model analyses customer and transaction data to look at important churn drivers. The study analyses model performance with an emphasis on detecting high-risk clients and finds that a combined model with status indicators achieves up to 94% accuracy, while logistic regression delivers a projected accuracy of about 76%. By emphasizing key client characteristics, this strategy helps logistics firms implement proactive customer retention tactics and enhance overall business stability in cutthroat marketplaces.

Madhuri et al. use deep learning techniques, namely Convolutional Neural Networks (CNN) and Artificial Neural Networks (ANN), to present a customer churn prediction model for the telecom industry. In order to improve prediction accuracy, the model places a strong emphasis on data preprocessing methods including scaling, balancing, and feature selection. It pays particular attention to crucial elements like client tenure, service plan, and monthly costs. The CNN model outperformed the ANN model with an accuracy of 97.78% and a true positive rate of 95.45% after being trained on a telecoms dataset. These findings highlight the model's potential for efficient client retention and churn prediction in telecom settings.

To increase explainability and accuracy, Cenggoro et al. suggest a deep learning model for predicting customer attrition that uses vector embedding. In order to better distinguish between churning and loyal consumers, the model uses vector embeddings to capture client behavior patterns in a low-dimensional space. With an emphasis on telecommunications data, important aspects like service usage and customer interaction frequency are incorporated to capture behavioral subtleties. With an F1 score of 81.16%, the model demonstrated its ability to successfully identify high-risk churn consumers. The embedding method of this model provides useful information for focused client retention tactics in the telecom industry.

The paper by Ganapatisamy and Narayan focuses on employee attrition statistics, presenting a model that combines long-term and short-term memory with recurrent neural networks (LSTM-RNN) and Brownian motion butterfly optimization (BM-BOA) for feature selection Trained on IBM HR dataset with 35 employee attributes, the method achieved 96.68% accuracy, 96.64% accuracy, and 96.62% recall, showing high prediction performance in identifying potential attrition cases Leveraging BM-BOA for optimal attribute selection, the model on major attrition factors Provides insights, making it more effective in developing strong retention strategies in competitive environments.

Bandyopadhyay and Jadhav’s research focuses on labor attrition prediction using machine learning methods, using models such as Support Vector Machine (SVM), Naive Bayes and Random Forest, random forest classifier emerged as the best predictor it has 70.83% accuracy, 70.61% residual and 73.48%. AUC. By examining important attributes using selection methods such as recursive feature elimination (RFE) and LVQ, studies identify factors such as satisfaction level and career choice as main influences. This strategy helps in targeted storage methods, especially in competitive technologies.

Ramaswamy and DiClerc’s paper focuses on consumer insight research using deep learning and natural language processing (NLP) techniques to better capture consumer feedback. The analysis uses convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to perform sentence representation analysis, resulting in an accuracy of 85% in perceptual classification when applied to a vehicle analysis dataset When rule based semantic tagging and deep learning models together, the method extracts meaningful insights from customer perceptions Provides information In order to understand customer sentiment In a competitive market The model is also particularly useful in terms of concerns in the handling of materials.

The paper by Simian-Constantinescu, Pisiu, and Damian presents a hybrid neural network model that combines fully-connected (FCNN) and recursive (RNN) networks for churn prediction in a pharmacy using GPU-optimized LSTM cells, model 90.71 %. recall, 45.16% mean. accuracy, and 75.48% F2 score in a dataset of about 150 million transactions This group-method identifies high-risk customers and predicts the timing of the next, larger event with strong predictive capabilities and also Support targeted strategies to prevent churn across all lines of business.

The paper by Zhou, Zhang, and Zhou examines the application of model compression techniques in subscriber churn forecasting for the telecom sector. This study uses a bi-compressed artificial neural network (ANN) that takes advantage of pruning and quantization to reduce computational load and storage requirements without affecting model accuracy trained on a data set of 125,296 telecom churn records large, an accuracy of 88.1% was obtained with the compressed model, with computation time reduced to 6.1 seconds and a model size of 1.6 MB. These developments demonstrate the feasibility of implementing high-performance churn prediction models on infrastructure-intensive devices, enabling telecom operators to implement real-time customer retention strategies that reduce costs and increased scalability has been implemented.

2.2 BACKGROUND STUDY TABLE

Architecture	Authors	Result
LSTM + Stacked Autoencoder + CPIO	Irina V. Pustokhina, C. Jeyalakshmi, et al.	Accuracy = 95.56%, Kappa = 94.3%, F1-score = 94.96%, AUC = 0.93
2D-CNN with Apache Spark	Tariq et al	Accuracy = 96.3%, True Positive Rate = 95%, True Negative Rate = 94%
WOFSVM	Ibitoye and Onifade	Accuracy = 84.23%, Precision = 83.17%, Recall = 85.20%
Hybrid ANN + SOM	Tsai and Lu	Accuracy = 93.7%, Reduced Type I and Type II errors
Gradient Boosting Machine with SHAP	Poudel, Pokharel, Timilsina	Accuracy = 81%, Precision = 67%, Recall = 55%
SMOTE + Tomek Links + ENN + Optuna	Imani and Arabnia	F1-score = 92%, ROC AUC = 91%
XGBoost with Feature Selection	Lukita et al	Accuracy = 87%, Precision = 84%, Recall = 92%
Decision Tree + Logistic Regression	Pradeep et al.	Accuracy = 94% (Decision Tree), Logistic Regression = 76%
CNN + ANN for Telecom Data	Madhuri et al.	CNN Accuracy = 97.78%, True Positive Rate = 95.45%
Deep Learning with Vector Embedding	Cenggoro et al	F1 Score = 81.16%
LSTM-RNN + BM-BOA	Ganapatisamy and Narayan	Accuracy = 96.68%, Precision = 96.64%, Recall = 96.62%
SVM, Naive Bayes, and Random Forest	Bandyopadhyay and Jadhav	Random Forest: Accuracy = 70.83%, Recall = 70.61%, AUC = 73.48%
CNN + RNN for NLP Analysis	Ramaswamy and DiClerc	Accuracy = 85%
Hybrid FCNN + RNN with LSTM Cells	Simian-Constantinescu, Pisiu, and Damian	Recall = 90.71%, Accuracy = 45.16%, F2 Score = 75.48%
Bi-Compressed ANN (Pruning + Quantization)	Zhou, Zhang, and Zhou	Accuracy = 88.1%, Computation Time = 6.1 s, Model Size = 1.6 MB

Table I: Literature Survey

Chapter 3

Proposed Architecture

3.1 DATASET

The dataset employed in this research offers a comprehensive and nuanced perspective on customer information within a telecommunications or service sector framework. Every record corresponds to an individual customer and comprises various attributes that provide insights into demographics, service utilization, financial habits, and customer opinions. These characteristics will be essential in forecasting customer attrition, offering important insights for enhancing customer retention tactics.

The demographic characteristics consist of ID, a distinct identifier for every customer; Sex, classifying customers as Male (M) or Female (F); and Status, showing whether the customer is Single (S) or Married (M). These characteristics assist in classifying customers into general categories according to personal traits. Another demographic characteristic is Children, which signifies if the customer has children (1 for yes, 0 for no). This variable can provide valuable insights since families might exhibit distinct service usage behaviors in contrast to single individuals. Moreover, the Est_ Income (Estimated Income) variable offers an estimation of the customer's earnings, which may affect their selection of service plans and their propensity to churn depending on their financial stability.

The service utilization and interaction features consist of Car_Owner, which tracks if the client possesses a car (Y or N). This could act as an indicator of financial stability, since owning a car might be associated with greater disposable income. The Usage attribute indicates the overall monthly consumption by the customer, while the Age of the customer is noted in numeric form. Age and usage trends can offer valuable insights into consumer behavior and their likelihood of leaving, as younger clients may be more prone to changing services. The Rate Plan attribute indicates the kind of service plan the customer is signed up for, like basic, premium, or other options, which is directly linked to the customer's usage patterns and satisfaction level. Grasping service plan choices is essential in forecasting churn, since customers on lower-cost plans could be more affected by price fluctuations or service quality concerns. Additional usage-related attributes consist of Long Distance, International, and Local, which indicate the customer's usage for long-distance, international, and local calls, respectively. These characteristics aid in evaluating how dependent the customer is on certain features of the service, which can influence churn due to their satisfaction or dissatisfaction with these offerings.

The billing and payment features encompass the Pay method variable, which logs the payment method utilized by the customer (e.g., Credit Card, Check). The method of payment can affect churn, since specific payment types might suggest greater flexibility or simpler service cancellation. The dataset contains Local Bill type and Long-Distance Bill type, representing the billing categories for local and long-distance calls, respectively. These billing models can influence customer contentment and their choice to discontinue the service if they perceive the fees as unfavorable.

The dataset's target variable is TARGET, signifying if a customer has terminated their service (Cancelled) or remains an active customer (Current). This is the main result that the churn prediction model will seek to forecast. Furthermore, the dataset features a Comments column that holds text-based customer feedback, offering additional insight into customer behavior, service interactions, and possible factors for churn. This unstructured data will undergo processing through Natural Language Processing (NLP) methods to extract sentiments, aiding in a deeper understanding of the factors affecting churn that structured variables may not capture.

In conclusion, the dataset provides an extensive mix of structured data (including demographic, financial, and service- related attributes) and unstructured data (via customer feed- back represented as comments). This comprehensive dataset facilitates an in-depth examination of customer behavior, offering critical insights for machine learning models focused on forecasting churn. By integrating both structured and unstructured features, the model is able to achieve a more comprehensive insight into the elements influencing customer retention and churn, allowing businesses to develop improved strategies for customer retention and engagement.

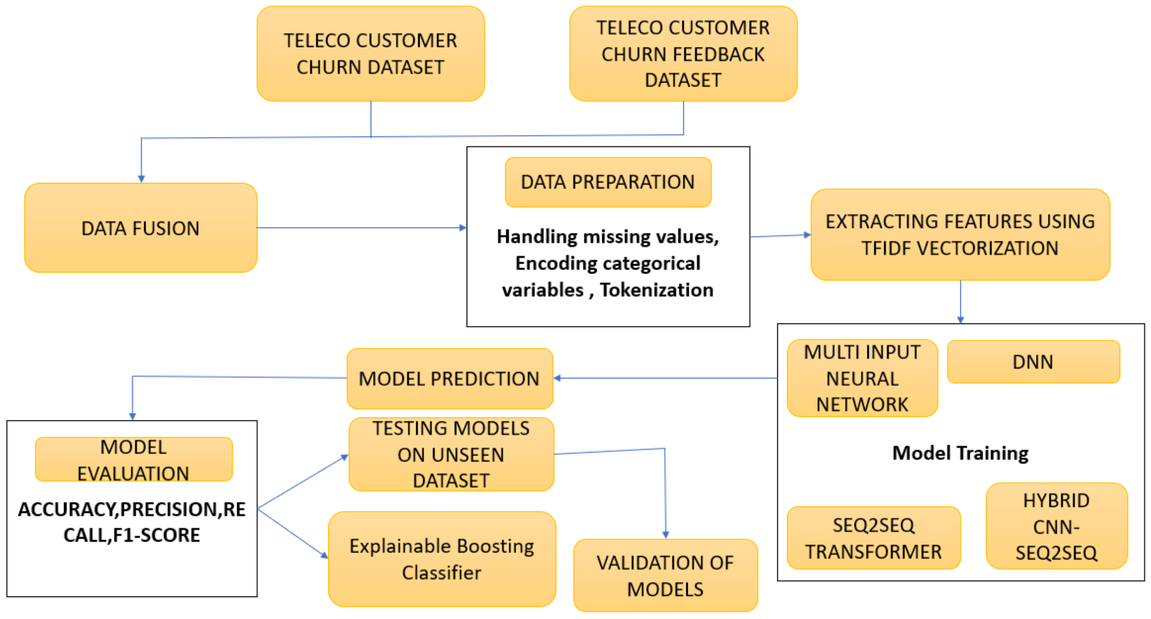

TELECO CUSTOMER CHURN DATASET
TELECO CUSTOMER CHURN FEEDBACK DATASET
DATA FUSION
DATA PREPARATION
Handling missing values, Encoding categorical variables , Tokenization
EXTRACTING FEATURES USING TFIDF VECTORIZATION
MULTI INPUT NEURAL NETWORK
DNN
Model Training
SEQ2SEQ TRANSFORMER
HYBRID CNN-SEQ2SEQ
MODEL PREDICTION
TESTING MODELS ON UNSEEN DATASET
Explainable Boosting Classifier
VALIDATION OF MODELS
MODEL EVALUATION
ACCURACY,PRECISION,RECALL,F1-SCORE

Fig.1. Proposed Architecture

3.2 Data Preprocessing and Feature Extraction

The first crucial step in the project is data preprocessing, where the raw dataset is transformed into a suitable format for model training. This involves handling missing data, addressing outliers, encoding categorical variables, and transforming numerical features. These preprocessing steps are essential to improve the model's performance and ensure that the data is consistent and clean.

1. *Data Sanitization:* The initial step in preprocessing is data sanitization, where missing or null values are detected and managed. Missing values in numeric columns such as Est_Income, Usage, and Age are handled using mean imputation. This method replaces missing values with the mean of the respective columns. The mean for any numeric column x is calculated as:

$$\text{Mean} = (X1 + X2 + \cdots Xn)/n \qquad (1)$$

values are replaced with the central tendency, which preserves the distribution and avoids losing data.
In cases where categorical columns like Sex and Status have missing values, we use mode imputation, replacing the missing values with the most frequent category. This is common for categorical variables where the mode reflects the most representative value.

2. Outlier Detection: Outlier detection is the next step to preserve the integrity of the data. Outliers in continuous features like Est Income, Usage, and Age are identified using the Interquartile Range (IQR) method. The IQR for a numeric attribute is calculated as:

$$IQR = Q_3 - Q_1 \qquad (2)$$

where Qi is the first quartile and Q is the third quartile of the data. Outliers are defined as values falling outside the range:

Lower Bound = Q1-1.5*IQR
Upper Bound = Q3+1.5*IQF

Any values outside these bounds are considered outliers and are either capped (to the nearest valid value) or removed from the dataset to prevent them from skewing the model's performance.

3. Categorical Data Encoding: After handling missing data and outliers, we move on to encoding categorical features. Many machine learning models require numerical input, so categorical variables such as Sex, Status, Car Owner, Rate Plan, and Pay method are encoded into numerical format using one-hot encoding. This process

generates binary columns for each category of the attribute. For instance, the Sex feature has categories M (Male) and F (Female), which will be converted into two binary columns:

$$SEX_M = \begin{cases} 1\ IF\ SEX = M \\ 0\ IF\ SEX = F \end{cases}$$

$$SEX_F = \begin{cases} 1\ IF\ SEX = F \\ 0\ IF\ SEX = M \end{cases}$$

Similarly, other categorical variables like Rate Plan and Pax method are transformed into binary features. This encoding allows machine learning algorithms to interpret categorical data in a way that captures its information efficiently.

4. Numerical Data Normalization: Once the categorical features are encoded, the next step is normalization of numerical features. Features like Usage, Est Income, and Age are scaled to ensure uniformity across all numerical columns. Since different attributes may have varying ranges (e.g., Usage might range from 0 to 1000, while Age ranges from 18 to 80), Min-Max scaling is applied. The Min-Max scaling formula is as follows:

$$x' = \frac{x - x\_min}{(x_{max} - x\_\min)} \qquad (3)$$

where x is the original value, $X_{\min}$ is the minimum value in the column, and Xmas is the maximum value. This scaling ensures that all numerical attributes are transformed to a uniform range, typically [0, 1], preventing any attribute with a larger range from disproportionately affecting the model.

5. Feature Engineering: In addition to preprocessing, we perform feature engineering to enhance the predictive power of the model. This step involves extracting new features or modifying existing ones to provide more meaningful information to the machine learning algorithms.
For instance, the Usage variable could be transformed into additional features such as Average Monthly Usage or Usage Segmentation (e.g., low, medium, high usage) based on domain knowledge or statistical thresholds. This can help capture different usage patterns that may relate to customer churn.
The Comments column, containing customer feedback, is another important source of information. Since this is unstructured text data, it needs to be transformed into a usable format for the model. We apply Term Frequency-Inverse Document Frequency (TF-IDF), a text vectorization technique, to convert the textual data into numerical representations. The TF-IDF formula is as follows:

$$TF - IDF(t,d) = TF(t,d)\ \times \log(\frac{N}{DF(t)}) \qquad (4)$$

where:

TF (t, d) is the term frequency of term tin document d

DF(t) is the number of documents containing the term t

N is the total number of documents (comments).

By applying TF-IDF, we capture the importance of words within the comments relative to the entire dataset, allowing us to extract meaningful features that reflect customer sentiment, potential issues, or satisfaction.

These transformed text features are then appended to the structured numeric and categorical features, forming a comprehensive feature matrix X, which is used as input for predictive modeling.

3.3 Model Descriptions and Prediction Techniques

In this subsection, we explain the various machine learning models and prediction techniques used in the customer churn prediction project. These models include multi-input Convolutional Neural Networks (CNN), Deep Neural Networks (DNN), Sequence-to-Sequence (Seq2Seq) transformers, and a Hybrid CNN-Seq2Seq model. Each model plays a critical role in learning the relationships between customer data features and predicting churn.

1) *Multi-input Convolutional Neural Network (CNN):* The CNN model is designed to extract hierarchical features from the customer data. CNNs are widely used for image and sequence data but can also be highly effective in capturing local dependencies in structured data when properly configured. For our case, the CNN model is used to extract patterns from various features in the dataset, such as customer demographics, usage patterns, and payment information.

The CNN model applies the following operation to extract features from an input matrix X:

$$Y = \text{ReLU}(W * X + b) \qquad (1)$$

where:

-X is the input matrix (e.g., customer features),

-W is the filter matrix

-b is the bias term, denotes the convolution operation,

ReLU is the activation function.

The CNN captures local patterns by applying the filter W across the input data, performing non-linear transformations, and producing feature maps that are subsequently used for classification.

2) *Deep Neural Network (DNN):* The DNN model is a fully connected network designed to learn complex relationships between customer features. It consists of multiple hidden layers that process the input features through a series of non- linear transformations, enabling the model to capture intricate patterns in the data.

The output y of a DNN model with one hidden layer can be described as:

$$y = \sigma(W_2(\sigma(W_1 x + b_1)) + b_2) \quad (2)$$

where:
- x is the input vector (features of a customer),
- W_1 and W_2 are the weight matrices for the first and second layers,
- b_1 and b_2 are the bias terms for the layers,
- σ is the activation function, typically ReLU for hidden layers and sigmoid or SoftMax for the output layer.

DNNs are capable of capturing complex, non-linear relationships between customer attributes and churn probabilities.

3) *Seq2Seq Transformer Model:* The Sequence-to- Sequence (Seq2Seq) transformer model is a powerful deep learning architecture for sequential data tasks, such as time series forecasting or text translation. For this project, the Seq2Seq model is applied to analyze temporal customer behavior and predict churn.

The core of the Seq2Seq model is the attention mechanism, which allows the model to focus on relevant parts of the input sequence. The attention mechanism calculates a weighted sum of the input sequence $X = (x_1, x_2, . . . , x_n)$ based on attention weights α, which are computed as

$$\alpha_i = \frac{exp(score(q,k_i))}{\sum_{j=1}^{n}(exp(score(q,k_j))} \quad (3)$$

Where:

-score(q, k_i) is the compatibility score between the query q and key k_i,
- α_i represents the attention weight for the i-th element of the input sequence.

The attention weights α are used to compute the context vector, which summarizes the most important parts of the input sequence for predicting churn. The context vector is then passed through a decoder to generate the final output.

4) *Hybrid CNN-Seq2Seq Model:* The Hybrid CNN- Seq2Seq model combines the feature extraction power of CNNs with the sequential learning capabilities of Seq2Seq transformers. This hybrid architecture allows the model to first learn local features using CNN layers and then capture the temporal dependencies between customer behavior patterns using Seq2Seq layers.

The architecture follows a two-step process:

i. **Feature Extraction:** The CNN layers extract local pat- terns from the input data, similar to the standard CNN model described earlier.
ii. **Sequence Modeling:** The extracted features are then passed into the Seq2Seq layers, where the temporal dependencies and customer behavior trends are captured using attention mechanisms and sequence learning.

The final output of the Hybrid CNN-Seq2Seq model is obtained by combining the outputs of both the CNN and Seq2Seq components, which are then passed through a fully connected layer for classification.

$$y = softmax(W_{hybrid} \times Seq2Seq(CNN(X)) + b_{hybrid}) \qquad (4)$$

where:

- X is the input feature matrix,
- Seq2Seq($CNN\,(X)$) represents the output of the CNN-Seq2Seq pipeline,
- W_{hybrid} is the weight matrix for the hybrid model,
- b_{hybrid} is the bias term,
- The SoftMax function is used for multi-class classification, yielding the probabilities of churn.

This hybrid model aims to leverage both local feature extraction (CNN) and sequence learning (Seq2Seq) to improve churn prediction accuracy.

5) *Explainable Boosting Classifier (EBC):* The Explainable Boosting Classifier (EBC) is a model that provides both high performance and interpretability, making it particularly suitable for problems where model transparency is crucial. It is part of the family of generalized additive models (GAMs) and is designed to be interpretable by providing feature importance scores and partial dependence plots.

EBC operates by fitting a generalized additive model to the data, where the prediction function is expressed as a sum of component functions of individual features. It builds the model using decision trees in an additive manner, focusing on boosting the accuracy of weak learners while keeping the model simple and interpretable.

In contrast to black-box models like deep learning, EBC offers transparency by showing how each feature influences the final prediction. This feature is particularly useful in customer churn prediction, where understanding the factor’s driving churn is as important as making accurate predictions. The EBC algorithm is trained using gradient boosting, which combines weak learners (typically shallow decision trees) to iteratively correct the errors of previous models. This process leads to an ensemble of trees that work together to improve model performance while maintaining interpretability

One of the key advantages of the EBC is its ability to generate partial dependence plots, which visualize the relationship between a feature and the model's predictions. These plots help identify the influence of individual features on the outcome, thus aiding in the interpretability and trustworthiness of the model's predictions.

The prediction techniques utilized in this project include both classification and regression approaches. The models described above are primarily used for classification tasks, where the goal is to predict whether a customer will churn or not. These models output a probability score that is interpreted as the likelihood of churn, which is compared to a predefined threshold to classify the customer as either "churned" or "not churned."

Chapter 4

Data Preprocessing

4.1 PREPROCESSING STEPS

1. Data cleansing

- **Handling of missing values:** Missing values are handled to avoid problems in model training.
- **Statistical features:** For continuous variables, missing values are often replaced by the mean or median value of that variable to preserve all data distributions.
- **Classification type:** For classification types (e.g., gender, payment type), missing values are replaced with a placeholder such as "Unknown" or a method for that type.
- **Data type conversion:** Some attributes can have mixed data types (e.g., numbers stored as strings). Converting these attributes to their appropriate forms (e.g., floats for continuous values, integers for categorical notation) ensures consistency with previous implementation and modelling steps.

2. Data normalization and scaling

- **Scaling of numerical features:** To ensure that all numerical features are on the same scale, we use a normalization method:
- **Standardization (Z-score Normalization):** Each value is adjusted so that the statistical factors have a mean of 0 and a standard deviation of 1, providing comparability and reducing the risk of model bias towards large values.
- **Min-Max Scaling:** Some models benefit from having attributes scaled from 0 to 1, especially if the model is sensitive to data ranges. This method is applied selectively depending on sampling requirements.

3. Encoding Categorical Variables

- **Label Encoding**: Label encoding is assigned to classification items with a number of biological orders or categories (e.g., gender, payment type). Each class is assigned a unique integer, allowing these variables to be manipulated by machine learning models that can interpret the mathematical notation.

- **One-hot encoding:** For models that benefit from avoiding sequential assumptions in categorical data, single-hot encoding is used to convert each class into a binary feature (0 or 1) to form positive representation This is useful to prevent the inference of any sequential relationship where none occurs.

4. Text Data Preprocessing

Text data, such as customer feedback, requires special processing to convert it into a format usable by machine learning models:

Tokenization: The text is split into individual tokens (words or sub words), which forms the basic unit of analysis for NLP models.

Text Cleaning:

- **Lowercasing**: All text is converted to lowercase to ensure that words are treated equally regardless of case.
- **Punctuation and Special Characters Removal**: To reduce noise, non-alphanumeric characters (e.g., punctuation marks, symbols) are removed.
- **Stop Words Removal**: Common, non-informative words (e.g., "the," "and," "is") are removed to focus the model on meaningful words.
- **Short Words Removal**: Words of less length (e.g., two lines or less) are eliminated, as they generally do not contribute to semantic comprehension.

Vectorization:

- **TF-IDF (Term Frequency-Inverse Document Frequency)**: Textual information is converted into numerical vectors using TF-IDF, which takes into account the importance of each word frequency in documents This allows the model to focus on words of importance, reducing the use of common, less informative words.

5. Feature Engineering and Extraction

- **Feature Selection**: Features that do not provide meaningful information to the model (e.g., irrelevant or highly correlated features) are removed. This helps reduce dimensionality, reduces computational requirements, and improves model interpretation.

- **Synthetic Data Generation (SMOTE)**: For cases with high intra-class imbalances (e.g., churned far fewer customers than remaining customers), SMOTE (Synthetic Minority Oversampling Technique) is used SMOTE generates synthetic data points for minority the class, the better model It helps to identify and in both the groups Improves its accuracy.

6. Data Splitting

- **Train-Test Split**: The data is divided into training and testing sets, typically in a 70-30 or 80-20 ratio.
  - **Training Set**: Used to train the model.
  - **Testing Set**: Used to check the performance of the model, ensuring adequate generalization to unobserved data.
- **Validation Split** (if needed): For a deep learning model, a validation set is generated from the training set to tune hyperparameters and monitor overfitting during training.

This data preprocessing step ensures the conversion of structured and unstructured data into a format suitable for neural networks, thus improving the ability of models to identify meaningful patterns, to generalize effectively to new data , and making accurate churn predictions.

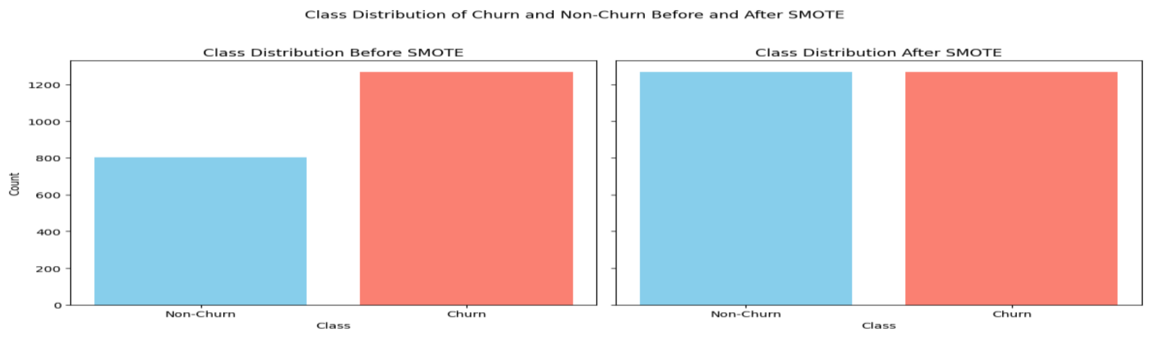


Fig.2. Class Distribution Before and after SMOTE

4.2 PSEUDOCODE

•Import the required libraries and modules:

- Import related Pandas, NumPy, Matplotlib, Seaborn, Scikit-learn, Imbalanced-learn, Text Blob, Torch, and PyTorch modules.

• Load and integrate data structures:

- Use pd.read_csv() to load customer_data and comments_data.
- Use pd.merge() to merge the data structures in the 'ID' field.

• Visualize the distribution of data:

- Use Seaborn to create statistical plots and histograms for attributes such as 'Age', 'Usage', 'Est_Income' and 'TARGET'.
- Plot the correlation matrix for the statistical factors.
- Use WordCloud to visualize common words used in ‘articles’.

• Replace the missing values:

- Change 'Est Income' to numeric using pd.to_numeric() and replace the error with NaN.
- Fill in the NaN value in 'Est_Income' using the interval.
- Replace NaN with 'No comments' in 'Comments' and replace 'RatePlan' with 'No Plan'.

• Encode categorical objects:

- Use LabelEncoder to convert categorical columns ('Sex', 'Status', 'Car_Owner' etc.) to numeric values.

• Preprocess text data:

- Define a clean_text() function using regex to convert text to lowercase, removing excess characters, lowercase words, and white.
- Create a 'clean_comments' column by using clean_text() on the 'Comments' column.

• **Split features and target**:

- Separate structured data (X_structured) and text data (X_text).
- Extract target (y) from the dataset.

- **Vectorize text data**:
  - ➢ Use TfidfVectorizer to convert 'clean_comments' to TF-IDF features.

- **Normalize structured data**:
  - ➢ Use StandardScaler to standardize data.

- **Combine structured and text data**:
  - ➢ Combine TF-IDF objects with scaled structured features using np.hstack().

- **Handle class imbalance**:
  - ➢ Use SMOTE to balance the target classes.

- **Split the dataset**:
  - ➢ Use train_test_split() to split X and y into training test sets.

- **Explain PyTorch multi-input neural network model:**
  - ➢ Create a class called MultiInputNN that underlies collections for structured-text inputs.
  - ➢ Connect the output from both branches and move through the levels of complete overlap.

- **Train the model**:
  - ➢ Convert the training and test data into PyTorch tensors.
  - ➢ Define CrossEntropyLoss as the loss function and Adam as the optimizer.
  - ➢ Train the model using DataLoader for the specified epochs and track loss and accuracy.
- **Evaluate the model**:
  - ➢ Calculate the accuracy of the test set.
  - ➢ Publish the classification report using classification_report().
  - ➢ Create and display a confusion matrix using confusion_matrix() and Seaborn heatmap.
- **Plot training and testing metrics**:
  - ➢ Use Matplotlib to plot the accuracy of training and testing at different times.
- **Compute evaluation metrics**:

- ➢ Calculate the precision, recall, F1 score, and ROC AUC score for the model.

- **Visualize results**:
  - ➢ Use Seaborn to plot the confusion matrix and write notes for a better explanation.
- **Summary and explainability**:
  - ➢ Use the translation package for global and local instance translations if required.

4.3 ALGORITHMS (PROPOSED MODELS)

Multi Input Neural Network:

A multi-input neural network is an architecture designed to handle multiple types of input data simultaneously. It can process various input sources or modalities, such as images, text, numerical data, or other signals, to perform tasks like classification, regression, or prediction. These networks are structured to handle multiple input branches that combine the learned features into a unified output. This architecture is common in applications where multiple data sources are involved, such as image-text integration, sensor data fusion, or multi-modal learning.

Architecture:

- ➢ **Multiple Input Layers:** The model has separate input layers for each type of data or feature. Each input layer is followed by its respective processing branch (which could be convolutional, dense, or recurrent layers, depending on the input).
- ➢ **Shared or Independent Networks**: In some cases, the branches might share some initial layers (to extract common features), or they might be entirely independent.
- ➢ **Feature Fusion:** After processing each input individually through its respective layers, the features are fused (usually by concatenation, summation, or other aggregation techniques) before passing them into subsequent layers.
- ➢ **Unified Output Layer:** After feature fusion, the combined features pass through dense layers, and the final output is produced by the network (like a SoftMax layer for classification tasks).

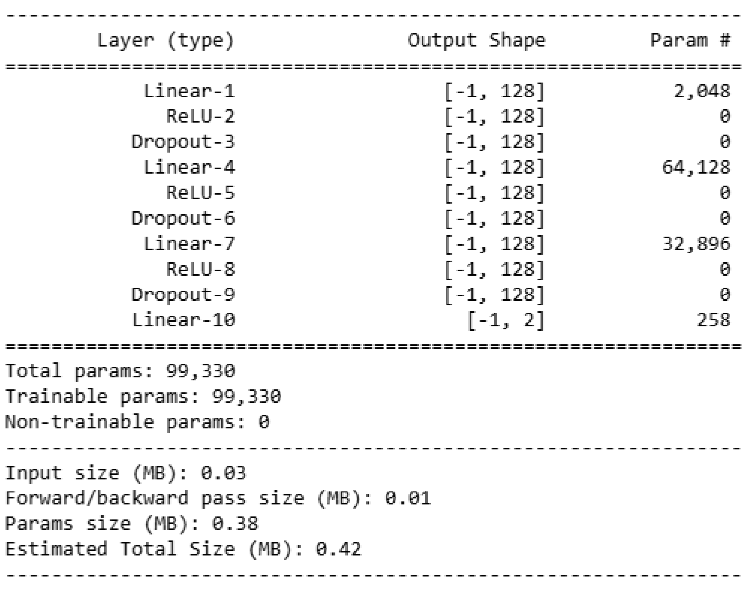
Advantages:

- **Handling Multi-Modal Data**: Multi-input neural networks can learn from different data sets simultaneously, making them ideal for tasks such as multi-modal classification, where inputs such as images must be consumed together and information handling.

- **Flexibility in Learning**: The model can automatically learn features from each input type before combining them, allowing the network to modify the learning of each modality separately.
- **Improved Performance**: Combining data sets gives the model better generalization and efficiency, as it can use the overlapping data sources.

Disadvantages:

- **Complex Architecture**: Multi-input neural networks can be more complex to design and implement, as they require handling different types of data simultaneously and managing the fusion process effectively.
- **High Computational Cost**: These models typically require more computational resources, as they involve processing multiple inputs and combining them into a single feature vector.
- **Risk of Overfitting**: If the data set is not large or diverse enough, these networks tend to be overfitting, especially in very complex architectures.
- **Training Complexity**: Over-parameter tuning of the network can be challenging, especially when dealing with different data sets, and often requires more sophisticated training techniques.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                 [-1, 128]           2,048
              ReLU-2                 [-1, 128]               0
           Dropout-3                 [-1, 128]               0
            Linear-4                 [-1, 128]          64,128
              ReLU-5                 [-1, 128]               0
           Dropout-6                 [-1, 128]               0
            Linear-7                 [-1, 128]          32,896
              ReLU-8                 [-1, 128]               0
           Dropout-9                 [-1, 128]               0
           Linear-10                   [-1, 2]             258
================================================================
Total params: 99,330
Trainable params: 99,330
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.03
Forward/backward pass size (MB): 0.01
Params size (MB): 0.38
Estimated Total Size (MB): 0.42
----------------------------------------------------------------
```

Fig.3. Multi Input Neural Network Architecture

Deep Neural Network (DNN):

A deep-neural network (DNN) is an artificial neural network with many layers, where the output of each layer is input to the next. DNNs are powerful models that can learn complex patterns and signals from raw data, making them efficient for various tasks such as classification, regression and image recognition.

Architecture:

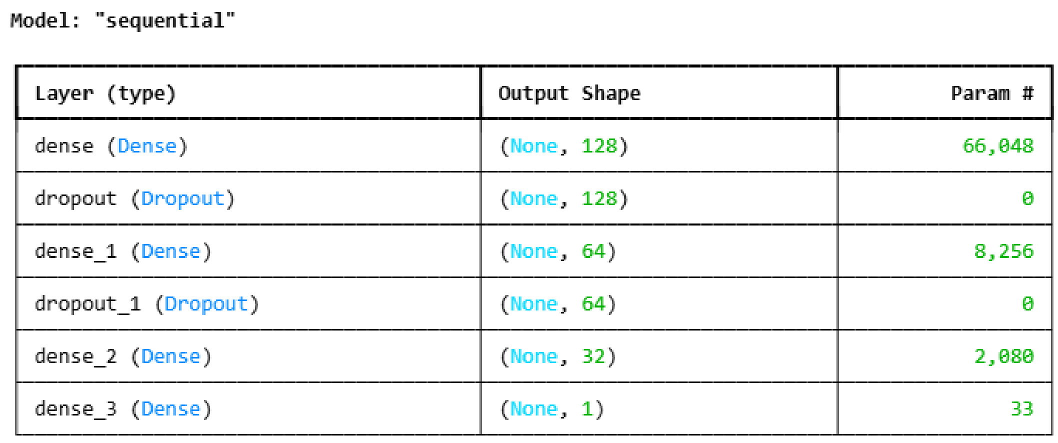
- **Input Layer**: The input layer captures raw attributes (such as pixel values, text attributes, etc.).
- **Hidden Layers**: These are the interfaces that transform the data. Deep neural networks have more than one hidden layer, allowing them to learn sequences of images. Each hidden neuron contains a set of weights and activation functions to process the data.
- **Activation Functions**: The most common functions used in DNNs are ReLU (Rectified Linear Unit), Sigmoid, and Tanh. ReLU is widely used for its flexibility and ability to solve commonly missing problems.
- **Output Layer**: The output layer provides the final prediction or classification. The order of the output layer depends on the task (e.g. SoftMax activation for multiclass classification or a single neuron for regression).

Advantages:

- **Learning Complex Patterns**: DNNs are capable of learning highly complex patterns and representations in data, especially when large amounts of data are available.
- **End-to-End Learning**: DNNs can be trained end-to-end, meaning they can learn directly from raw data (e.g., pixel values or text) without requiring manual feature extraction.
- **Versatility**: DNNs can be applied to a wide range of tasks, such as image and speech recognition, natural language processing, and even reinforcement learning.
- **Automatic Feature Learning**: In traditional machine learning, features need to be manually engineered. DNNs, on the other hand, can learn the best features directly from the data during training.

Disadvantages:

- **Training Time**: DNNs require a lot of data and computational power to train, especially as the number of layers increases. Training deep networks can be time-consuming, even with modern hardware.
- **Overfitting**: If the dataset is small or not diverse enough, DNNs are prone to overfitting, where the model learns noise instead of the actual underlying patterns. Regularization techniques such as dropout, weight decay, or early stopping can help mitigate this.
- **Need for Large Datasets**: Deep networks typically perform well only when large amounts of label data are available. Training on smaller datasets might not produce optimal performance.



```
Model: "sequential"

┃ Layer (type)            ┃ Output Shape        ┃    Param # ┃
│ dense (Dense)           │ (None, 128)         │     66,048 │
│ dropout (Dropout)       │ (None, 128)         │          0 │
│ dense_1 (Dense)         │ (None, 64)          │      8,256 │
│ dropout_1 (Dropout)     │ (None, 64)          │          0 │
│ dense_2 (Dense)         │ (None, 32)          │      2,080 │
│ dense_3 (Dense)         │ (None, 1)           │         33 │
```

Fig.4. DNN Architecture

Seq2Seq (Sequence-to-Sequence) Model :

The Seq2Seq (Sequence-to-Sequence) model is a type of deep learning algorithm designed to transform one sequence into another. Widely used in applications where both input and output are processed sequentially, such as machine translation, speech recognition, aggregation, and query Seq2Seq instances typically consist of two main components: an encoder and a decoder, which work together to process and sequence.

Architecture:

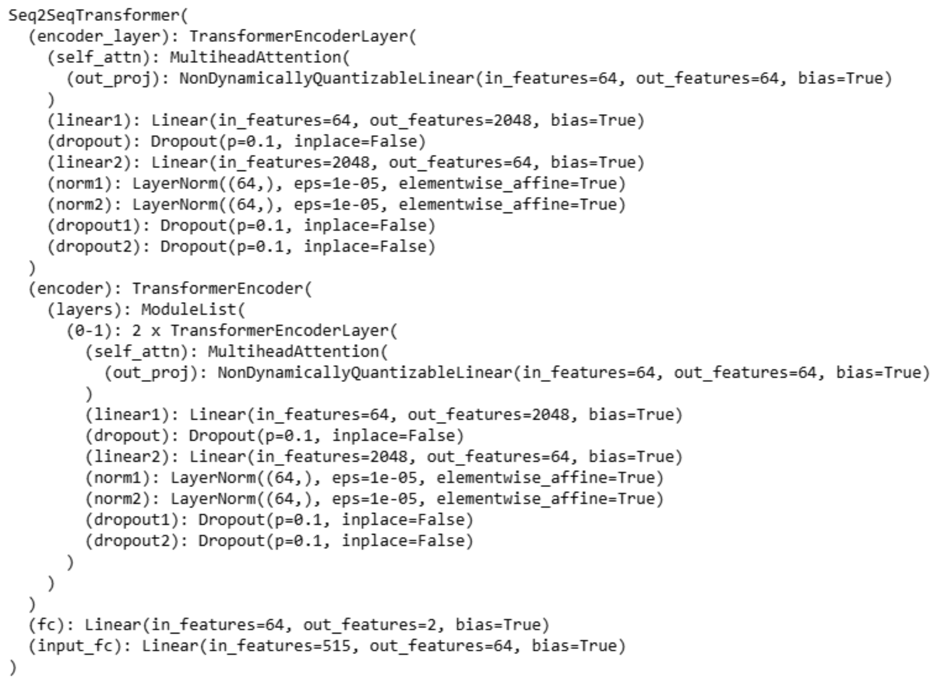
- **Encoders**: Encoders take an input sequence (e.g., a sentence in the source language) and process it into a context vector (also called hidden state) Encoders typically consist of recurrent neural networks (RNNs), long-short-term memory (LSTM), or gated repetitive units (GRU). It consists of cells, so that the model can process sequential data The encoder processes each element of the input sequence and updates its hidden state, capturing context.
- **Reference vector:** The last hidden state of the encoder (or hidden state sequence) is used as a reference vector, which represents the encoded information from the entire input sequence and this reference vector is then transmitted the decoder to aid in continuation of the output.
- **Decoder:** The decoder is responsible for generating the output sequence (e.g., translating the input sentence into another language). Like an encoder, a decoder typically consists of RNN, LSTM, or GRU units, and takes a reference vector (and possibly prior tokens) as input at each stage The decoder produces a stepwise output sequence, one token at a time, based on reference vector and pre-generated tokens.

Advantages:

- **End-to-end learning:** Seq2Seq instances can be trained end-to-end, which means they learn the process of converting an input sequence into an output sequence without the need for manual filtering.
- **Flexible insertion and deletion lengths:** Unlike traditional machine learning models that require fixed-length insertions and deletions, Seq2Seq models can handles sequences of varying lengths, making it ideal for applications such as machine translation, where input -Sequence length can vary from input to output.
- **Handling ordinal data:** Because the model is designed to work with ordinal data, it excels in problems where the order of the elements is important, such as natural language processing (NLP) tasks.
- **Scalability:** The architecture is flexible enough to scale to more complex problems with large sequences and high-dimensional data.

Disadvantages:

- **Training challenges**: Seq2Seq models can be computationally expensive and time-consuming to train, especially with large datasets. They also require a large amount of memory because the entire sequence of hidden states must be stored.
- **Missing flux gradient problem:** Like other RNN-based models, Seq2Seq models can suffer from missing-prone problems in back propagation, and especially for long sequences this can make it difficult for the model to detect long periods which is very self-contained. In practice, the use of LSTM or GRU units mitigates this issue.
- **Fixed reference vector limit:** In a standard Seq2Seq model, the entire input sequence is collapsed into a single reference vector, which may lose important information for longer sequences. This limitation is addressed with attentional focus, which allows the model to focus on particular aspects of the input sequence during decoding.
- **Difficulties with long series:** The model may struggle to obtain accurate results for long series, especially when the relationship between input and output extends over several time steps.



```
Seq2SeqTransformer(
  (encoder_layer): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=64, out_features=64, bias=True)
    )
    (linear1): Linear(in_features=64, out_features=2048, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (linear2): Linear(in_features=2048, out_features=64, bias=True)
    (norm1): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.1, inplace=False)
    (dropout2): Dropout(p=0.1, inplace=False)
  )
  (encoder): TransformerEncoder(
    (layers): ModuleList(
      (0-1): 2 x TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=64, out_features=64, bias=True)
        )
        (linear1): Linear(in_features=64, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=64, bias=True)
        (norm1): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (fc): Linear(in_features=64, out_features=2, bias=True)
  (input_fc): Linear(in_features=515, out_features=64, bias=True)
)
```

Fig.5. Seq2Seq Architecture

Hybrid CNN-Seq2Seq Model:

Hybrid CNN-Seq2Seq model combines the power of Convolutional Neural Networks (CNNs) and Sequence-to-Sequence (Seq2Seq) architectures to efficiently handle space and sequence information This hybrid approach is particularly useful for tasks such as image annotation, video manipulation, and image-text integration, where The model Must extract spatial features from the image and then process them sequentially in tasks such as text generation or translation in

Architecture:

Hybrid CNN-Seq2Seq models typically follow a two-step process:

1.CNN (Convolutional Neural Network) feature extraction:

- CNNs are mainly used as feature extractors when processing image data. CNN extracts high-level features from image input, extracting local features such as spatial patterns, edges, textures, and objects. This typically involves multiple convolutional layers followed by pooling of layers to reduce spatial dimensions.
- The resulting CNN consists of feature maps or compressed representation (such as plane vector or spatial mesh), which capture the spatial features of the input image.
- For applications involving sequential video or data, CNN can process each frame of the video independently or use 3D convolution to capture temporal and spatial features.

Seq2Seq for Sequence Generation:

- After CNN extracts this feature, the extracted features are passed to the Seq2Seq model, which typically consists of an encoder and a decoder.
- The encoder processes CNN-extraction features, often using RNN, LSTM, or GRU, to capture temporal or sequential relationships in the data
- The decoder generates an output sequence (such as a description, caption, or caption) based on a context vector from the encoder.
- The concept device can also be incorporated into the decoder stage to enable the model to focus on specific image features or feature maps during the sequence generation process.

Advantages:

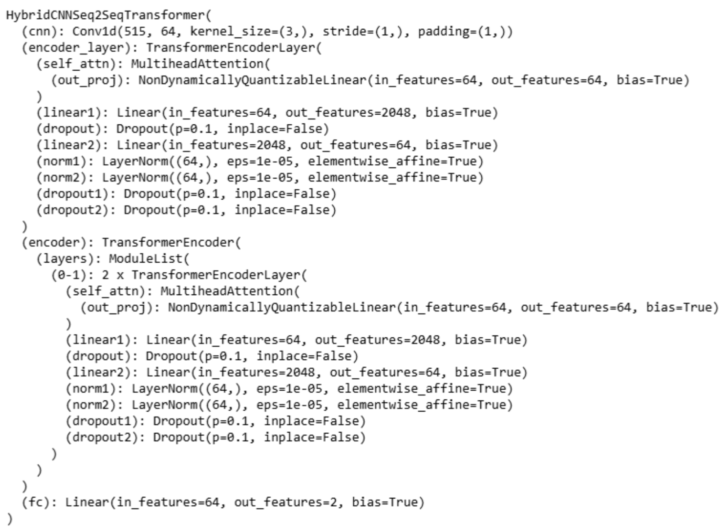
- **Powerful feature extraction:** Using CNNs for feature extraction, the model can capture complex spatial patterns and structures in images or video images, which are necessary for audio which is visually understood.
- **Effective Sequence Mapping:** The Seq2Seq component of the model enables the network to generate consistent and contextually accurate sequences from visual

sources, making it more efficient for tasks as image-to-text, video presentation, or image-to-text translation.

- **Handling multi-modal data:** Hybrid model is effective for handling multi-modal data, such as visual (image, video) and output (text and translation) data will combine, or even combine video frames with audio data
- **Focus:** The integration of focus helps the model focus on specific aspects of an image or sequence, improving the quality of the generated sequence by visually or by emphasizing the correct sequences in each decoding step.

Disadvantages:

- **Complexity:** Combining CNN and Seq2Seq models increases the complexity of the entire architecture, thus requiring more computational resources and memory. Interactions between CNN and Seq2Seq components can further complicate the training process.
- **Training time:** The hybrid model typically requires more training time due to the depth and complexity of the CNN and Seq2Seq networks. Furthermore, training large images or video data can be computationally expensive.
- **Overfitting:** Like any deep learning model, the Hybrid CNN-Seq2Seq model is prone to overfitting, especially when dealing with small amounts of data. Often, routine measures such as dropouts and data enhancement are needed.
- **Coherence issues:** In cases of long-term dependence between sequential input (such as video images) and sequential input (such as text), perhaps e.g species will struggle to capture spatial or temporal relationships at a distance, even in focal ways.



```
HybridCNNSeq2SeqTransformer(
  (cnn): Conv1d(515, 64, kernel_size=(3,), stride=(1,), padding=(1,))
  (encoder_layer): TransformerEncoderLayer(
    (self_attn): MultiheadAttention(
      (out_proj): NonDynamicallyQuantizableLinear(in_features=64, out_features=64, bias=True)
    )
    (linear1): Linear(in_features=64, out_features=2048, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (linear2): Linear(in_features=2048, out_features=64, bias=True)
    (norm1): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
    (norm2): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
    (dropout1): Dropout(p=0.1, inplace=False)
    (dropout2): Dropout(p=0.1, inplace=False)
  )
  (encoder): TransformerEncoder(
    (layers): ModuleList(
      (0-1): 2 x TransformerEncoderLayer(
        (self_attn): MultiheadAttention(
          (out_proj): NonDynamicallyQuantizableLinear(in_features=64, out_features=64, bias=True)
        )
        (linear1): Linear(in_features=64, out_features=2048, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
        (linear2): Linear(in_features=2048, out_features=64, bias=True)
        (norm1): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
        (norm2): LayerNorm((64,), eps=1e-05, elementwise_affine=True)
        (dropout1): Dropout(p=0.1, inplace=False)
        (dropout2): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (fc): Linear(in_features=64, out_features=2, bias=True)
)
```

Fig.6. Hybrid CNN-Seq2Seq

Explainable Boosting Classifier (EBC) for Explainability and Interpretability:

The **Explainable Boosting Classifier (EBC)** is a machine learning algorithm designed to provide both high prediction accuracy and pattern interpretation capabilities. It is part of a family of "semantic AI" (XAI) approaches, which are transparent, logical, and focused on providing insights into how decisions are made EBC's main advantage is that it includes the ability to improve (an ensemble approach). a popular) together with interpretability decisions in the model are easy to explain.

- EBC is a generalized additive model (GAM) that uses a developmental model of learning. Unlike traditional boosting models (such as Gradient Boosting Machines or XGBoost) that can function as "black box" models, EBC focuses on making the logic of the model more transparent.

Advantages of Explainable Boosting Classifier:

- **High predictive power:** EBC benefits from boosting, which helps to identify complex relationships between features, and allows for accurate predictions even with smaller data sets or less complex features
- **Feature-level descriptions:** EBC provides description at the feature level, which means that it can show how changes in individual feature values affect output. This is especially important in areas such as healthcare, finance, and law, where stakeholders need to understand the logic behind forecasts.
- **Post-Hoc Explainability:** Once the model is trained, EBC allows for post hoc analyses in which the predictions of the model can be classified and interpreted in terms of how each feature value contributes to the decision.

Disadvantages of Explainable Boosting Classifier:

- **Limited Complexity**: Although EBC offers greater definition, its simplicity) makes it less accurate in very complex tasks, especially when interactions between objects between the complex and the nonlinear
- **Training time:** Since boosting requires sequential training of multiple models, the training time of EBC can be longer than that of simple models, especially when the data set is large.
- **Not valid for all data types:** EBC may not be optimal in cases where very complex modelling is required or with high, simple data, such as natural language processing (NLP) or image recognition tasks, where methods deep learning is more effective

Chapter 5

Results and Discussion

5.1 PERFORMANCE METRICS EXPLANATION

Confusion matrix:

A confusion matrix is a performance evaluation tool used in machine learning that summarizes the performance of a classification model by tabulating true positive, true negative, false positive, and false negative predictions. It helps assess the accuracy and effectiveness of the model's predictions. The matrix is based on the concepts of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). It provides a granular view of a model's performance across different classes.

Structure:

- **True Positives (TP):** Instances where the model correctly predicts that a customer will churn when they actually do churn. This is essential for identifying customers who are truly at risk of leaving, allowing targeted retention strategies.
- **True Negatives (TN):** Instances where the model correctly predicts that a customer will not churn, and they indeed remain with the company. This reflects the model's accuracy in identifying loyal customers.
- **False Positives (FP):** Instances where the model incorrectly predicts that a customer will churn when they actually do not. False positives indicate that the model is overly cautious, possibly causing unnecessary retention efforts for customers who are not at risk.
- **False Negatives (FN):** Instances where the model incorrectly predicts that a customer will not churn when they actually do. False negatives are critical to minimize, as they represent missed opportunities to intervene and retain at-risk customers.

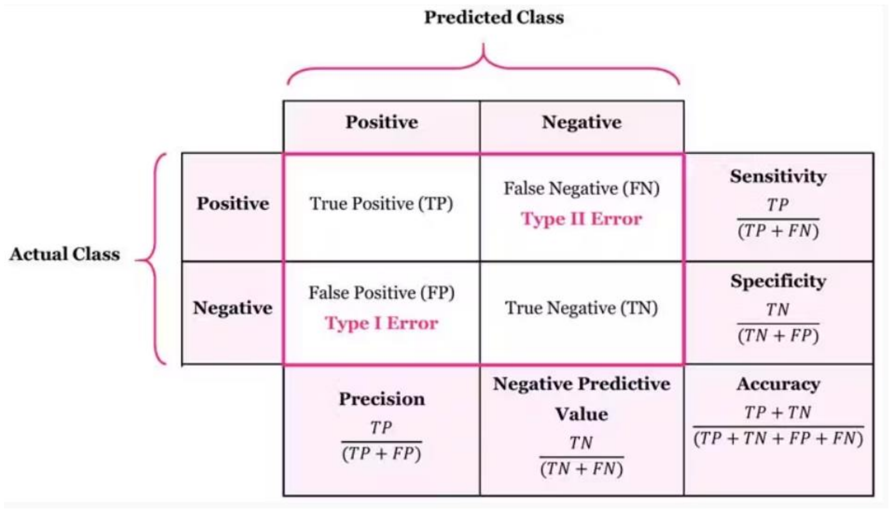


Figure 7: Confusion matrix

The confusion matrix serves as the basis for calculating essential evaluation metrics that offer nuanced insights into a model's performance:

- **Accuracy:** Accuracy quantifies the ratio of correct predictions (TP and TN) to the total number of predictions. While informative, this metric can be misleading when classes are imbalanced.
- **Precision:** Precision evaluates the proportion of true positive predictions among all positive predictions (TP / (TP + FP)). This metric is crucial when the cost of false positives is high.
- **Recall (Sensitivity or True Positive Rate**): Recall measures the ratio of true positive predictions to the actual number of positive instances (TP / (TP + FN)). This metric is significant when missing positive instances is costly.
- **Specificity (True Negative Rate):** Specificity calculates the ratio of true negative predictions to the actual number of negative instances (TN / (TN + FP)). This metric is vital when the emphasis is on accurately identifying negative instances.
- **F1-Score:** The F1-Score strikes a balance between precision and recall, making it useful when both false positives and false negatives carry similar importance.

- **Support:** Support is a representation of how many actual examples of each class there are in the dataset.

Applications of the Confusion Matrix

The confusion matrix has various applications tailored to the model described:

- **Model evaluation:** The main use of the confusion matrix for this model is to evaluate its classification performance for predicting customer churn. It provides insights into the model accuracy, precision, recall, and F1-score, and provides a detailed understanding of the model's effectiveness.
- **Forecasting customer churn:** The main use for the model is to forecast customer churn. The confusion matrix helps balance known true churn (true positives) and where churn is poorly predicted (false negatives) This helps develop optimal customer retention strategies, reducing attrition.
- **Business decision making:** By analyzing the confusion matrix, business stakeholders can better understand the impact of forecasts on business strategies, especially in customer retention efforts on the priority.
- **Effective fraud detection:** Although not the primary objective of the model, the confusion matrix approach can be modified to allow financial institutions to identify potentially fraudulent activity This will demonstrate how well the model detects and reduces actual fraud on false warnings.
- **Clinical assessment (possible extension):** Although not directly related to the current model, confusion matrices are important in the context of health care when the system is designed for the assessment of the situation. Performance is measured by identifying false positives and false negatives that are critical for patient outcomes.
- **Feedback Analytics and NLP:** The model adds an information analysis component to customer feedback. Confusion matrices can be used to measure how well it classifies emotions or information in customer feedback, thereby improving customer service processes.
- **Image and Object Recognition Adaptation**: hen extended to process image data (e.g., customer documents or scanned data), confusion matrices play a role in improving the image recognition capabilities of the model, and ensures accurate classification and minimal errors.

- **A/B testing on marketing campaigns:** When implementing different versions of AI-enabled marketing campaigns, confusion metrics help test which campaigns best categorize customer response (e.g., increased engagement or conversions more)., which provides data-driven marketing strategies.

Multiclass Classification with Example:

For multi-class classification, the confusion matrix extends to an N X N matrix, where N is the number of classes. Each cell in the matrix represents the count of instances for a specific pair of actual and predicted classes.

Detailed Explanation of Terms

True Positives (TP)

- **Definition**: True Positives are instances where the model correctly predicts the positive class.
- **Example**: In the customer churn prediction model, if a customer is predicted to churn (predicted positive) and actually does churn (actual positive), this is a True Positive.
- **Significance**: High TP values indicate that the model is effective in correctly identifying positive instances, contributing to better recall and overall accuracy.

True Negatives (TN)

- **Definition**: True Negatives are instances where the model correctly predicts the negative class.
- **Example**: In the churn prediction scenario, if a customer is predicted to remain (predicted negative) and does not churn (actual negative), this is a True Negative.
- **Significance**: High TN values indicate that the model is effective in correctly identifying negative instances. In multi-class settings, TN values per class help in understanding how well the model distinguishes each class.

False Positives (FP)

- **Definition**: False Positives are instances where the model incorrectly predicts the positive class.
- **Example**: If a customer is predicted to churn (predicted positive) but actually does not churn (actual negative), this is a False Positive.

- **Significance**: High FP values indicate a problem with the model's specificity. For customer churn prediction, this could mean false alarms that lead to unnecessary retention efforts, potentially wasting resources.

False Negatives (FN)

- **Definition**: False Negatives are instances where the model incorrectly predicts the negative class.
- **Example**: If a customer is predicted to stay (predicted negative) but actually churns (actual positive), this is a False Negative.
- **Significance**: High FN values indicate a problem with the model's recall. For customer churn, this could mean missed opportunities to retain at-risk customers, impacting business performance negatively.

These definitions and examples help illustrate how the confusion matrix can be applied to evaluate multi-class classification models, including those used for predicting customer churn or analysing customer feedback with sentiment analysis.

5.2 RESULTS

In this section, we present the performance of the various models developed for customer churn prediction. The models evaluated include the Multi-Input Neural Network (NN), Deep Neural Network (DNN), Seq2Seq, and Hybrid CNN-Seq2Seq models. Performance metrics such as accuracy, precision, recall, and F1-score are used for evaluation. We also provide comparisons between the models to better understand their relative effectiveness in predicting churn.

A. Multi-Input Neural Network (NN):

The Multi-Input NN model achieved a test accuracy of 90.73%, with a training accuracy of 89.43%. The convergence of training and test accuracy suggests that the model was able to generalize well and did not overfit the data. The results indicate a strong predictive capability, particularly for detecting churn.

Table I shows the detailed classification report for the model. The **precision** for the churn class is 0.94, which indicates that the model is effective in predicting churn cases, but with a recall of 0.87, it missed some churn instances. The precision-recall balance suggests that the model is conservative when predicting churn, which helps reduce false positives but might overlook some churn cases.

TABLE II
Performance Metrics for Multi-Input Neural Network

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Not Churn	0.88	0.94	0.91	246
Churn	0.94	0.87	0.91	261
Accuracy			**0.91**	507
Macro avg	0.91	0.91	0.91	507
Weighted avg	0.91	0.91	0.91	507

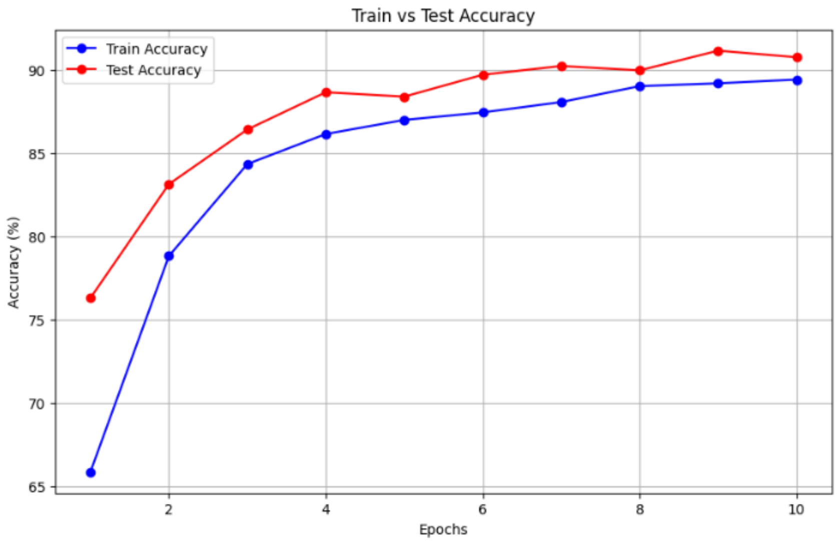


Figure 8: Accuracy plot for Multi-Input NN

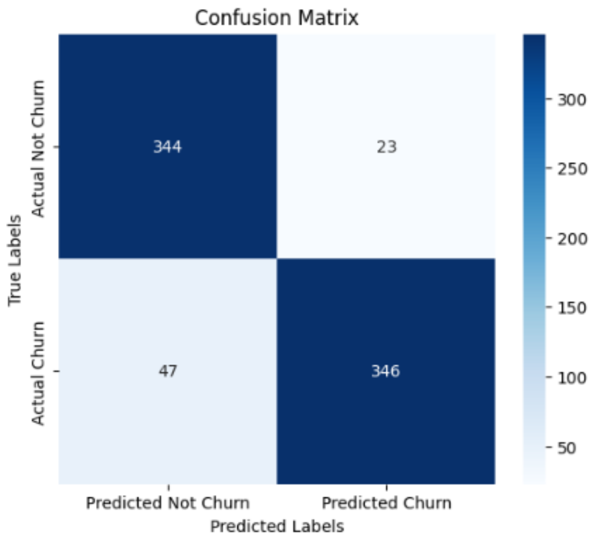


Figure 9: Confusion matrix for Multi-Input NN

B. Deep Neural Network (DNN):

The DNN model showed a strong performance, achieving a test accuracy of 90.93%. The model trained effectively with rapid convergence, improving from an accuracy of 60.39% in the first epoch to 91.36% by the final epoch. This rapid learning indicates that the DNN could effectively capture the patterns in the data over time, which is important for customer churn prediction.

The classification metrics shown in Table II reveal that the DNN performed similarly to the Multi-Input NN model. It achieved a slightly higher test accuracy (90.93%) compared to the Multi-Input NN (90.73%), as well as a better recall for the churn class (0.88 vs. 0.87). This improvement suggests that the DNN might have captured more churn cases compared to the Multi-Input NN, while maintaining a similar precision of 0.94 for churn predictions.

TABLE III
Performance Metrics for DNN Model

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Not Churn	0.88	0.94	0.91	246
Churn	0.94	0.88	0.91	261
Accuracy			**0.91**	507
Macro avg	0.91	0.91	0.91	507
Weighted avg	0.91	0.91	0.91	507

DNN Loss and Accuracy Graphs:

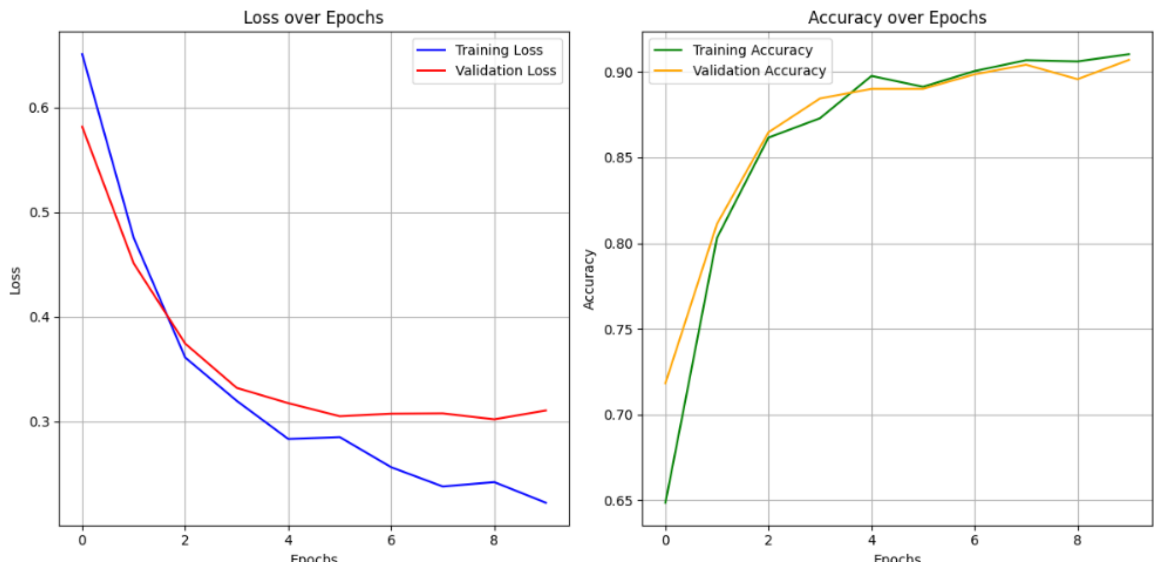


Figure 10: DNN Loss and Accuracy Graphs

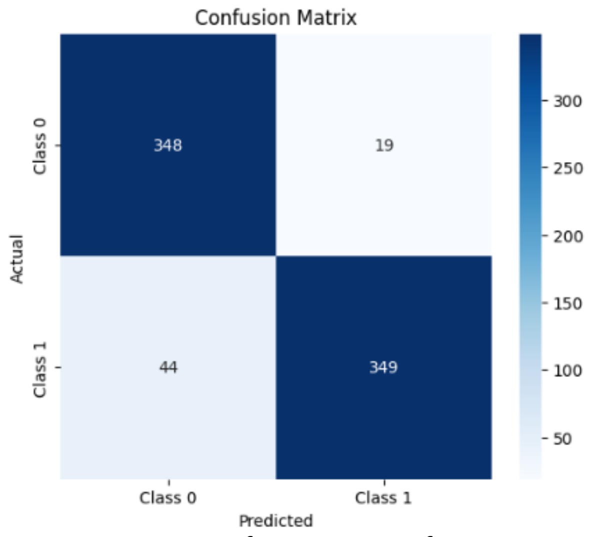


Figure 11: Confusion matrix for DNN

C. Seq2Seq Model

The Seq2Seq model, which is primarily designed for sequential data, achieved a training accuracy of 92.44% and a test accuracy of 89.94%. The relatively lower test accuracy indicates some overfitting, where the model has learned to perform well on the training data but failed to generalize effectively to unseen data. Despite this, the Seq2Seq model demonstrated strong precision (0.94) for the churn class, suggesting that when it did predict churn, it did so with high confidence.

Table III shows that the recall for churn in Seq2Seq was lower (0.85) compared to both the Multi-Input NN and DNN. This lower recall indicates that the model may have missed some churn customers, which could be due to the sequential nature of the data being harder for this model to capture, or due to the model overfitting to training data. The Seq2Seq's inability to generalize to the test set could be attributed to the complexity of sequential dependencies that might not have been well-captured in the training process

TABLE III
Performance Metrics for Seq2Seq Model

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Not Churn	0.86	0.95	0.90	246
Churn	0.94	0.85	0.90	261
Accuracy			**0.90**	507
Macro avg	0.90	0.90	0.90	507
Weighted avg	0.90	0.90	0.90	507

Seq2Seq Loss and Accuracy graphs :

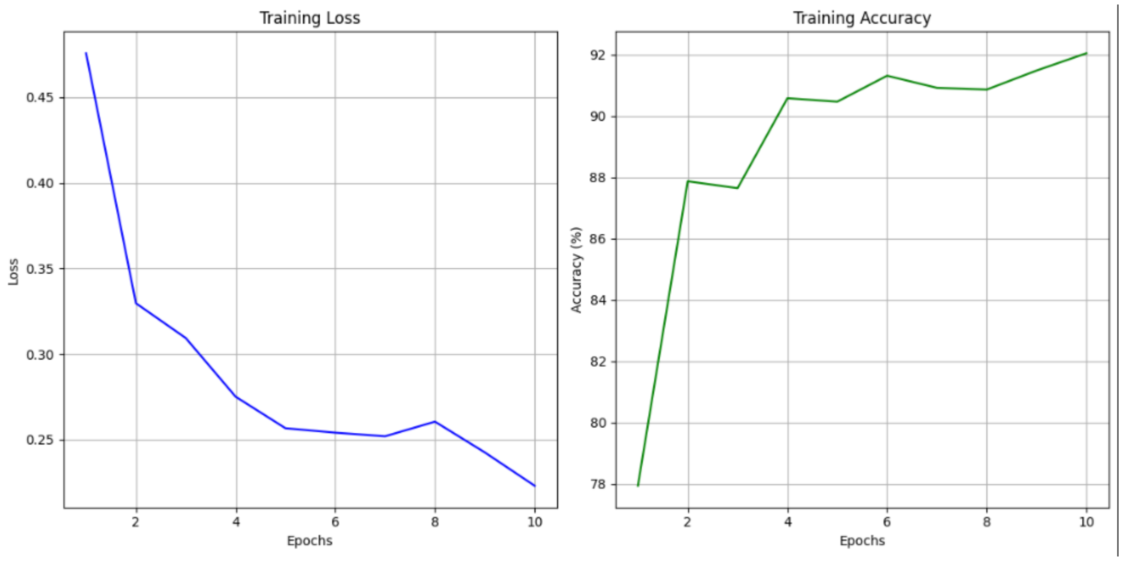


Figure 12: Seq2Seq Model Loss and Accuracy Graphs

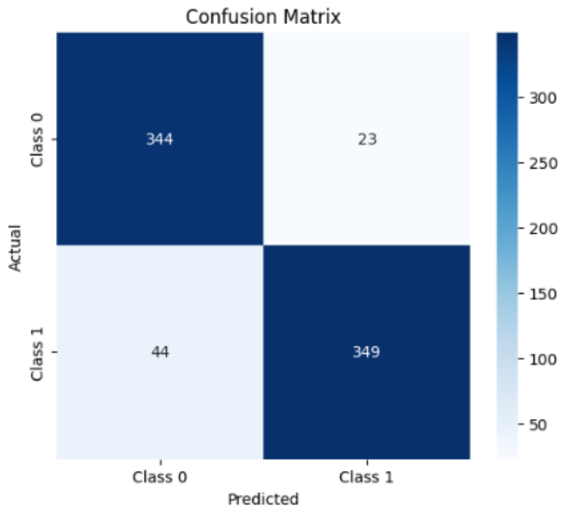


Figure 13: Confusion matrix for Seq2Seq Model

D. Hybrid CNN-Seq2Seq Model

The Hybrid CNN-Seq2Seq model, combining both CNN and Seq2Seq architectures, achieved a test accuracy of 91.32%, which is the highest among the models evaluated. This model's ability to capture both local features (via CNN) and sequential dependencies (via Seq2Seq) contributed to its superior performance. The CNN component helps extract relevant features from the data, while the Seq2Seq model captures long-term dependencies, leading to an overall improvement in predictive power.

Table IV shows that the Hybrid CNN-Seq2Seq model achieved precision and recall values of 0.94 and 0.89 for the churn class, respectively. This is indicative of the model's ability to predict churn cases accurately while maintaining a reasonable recall. The F1-score of 0.91 suggests a good balance between precision and recall, making the model particularly effective for churn prediction.

TABLE IV
PERFORMACE METRICS FOR HYBRID CNN-SEQ2SEQ MODEL

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Not Churn	0.89	0.94	0.91	246
Churn	0.94	0.89	0.91	261
Accuracy			**0.91**	507
Macro avg	0.91	0.91	0.91	507
Weighted avg	0.92	0.91	0.91	507

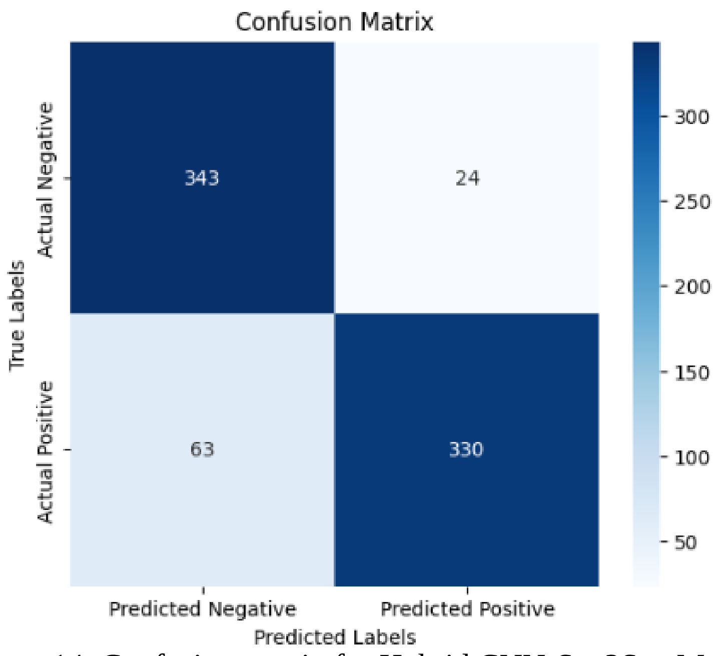


Figure 14: Confusion matrix for Hybrid CNN-Seq2Seq Model

E. Explainability Results Discussion

In this section, we present the performance of the Explain able Boosting Classifier (EBC) on both the complete dataset and the structured data subset. The EBC provides insights into model predictions while maintaining interpretability, crucial for understanding the decision-making process behind customer churn predictions. The classification results for the Explainable Boosting Classifier using the entire dataset are summarized as follows:

TABLE V
PERFORMANCE METRICS FOR EXPLAINABLE BOOSTING CLASSIFIER (TOTAL DATA)

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Not Churn	0.90	0.95	0.92	246
Churn	0.95	0.90	0.93	261
Accuracy			**0.93**	507
Macro avg	0.93	0.93	0.93	507
Weighted avg	0.93	0.93	0.93	507

The overall accuracy of 93% indicates strong model performance across the entire dataset. The classifier demonstrates high precision and recall, especially for class ”1” (churn), where it achieves a precision of 0.95 and recall of 0.90, highlighting its ability to identify customers who are likely to churn. The balanced performance of precision and recall leads to a competitive F1-score of 0.93, indicating that the model maintains a good balance between avoiding false positives and false negatives.

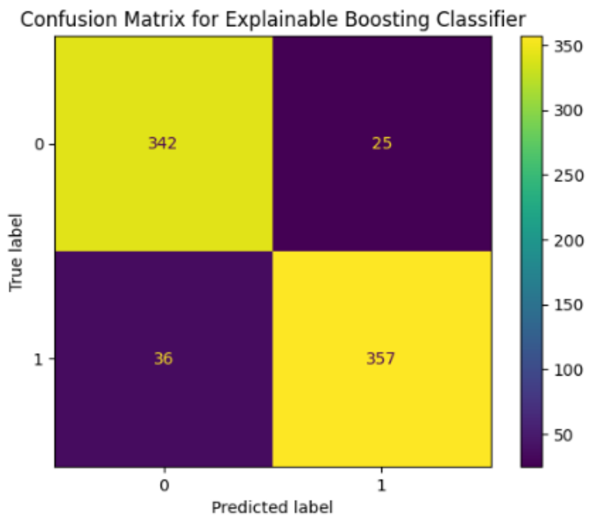


Figure 15: Confusion matrix for EBC

When trained on the structured data subset, the model exhibited slightly adjusted performance metrics, as seen in the following classification report:

TABLE VI
PERFORMANCE METRICS FOR EXPLAINABLE BOOSTING CLASSIFIER (STRUCTURED DATA)

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Not Churn	0.89	0.91	0.90	246
Churn	0.91	0.89	0.90	261
Accuracy			**0.90**	507
Macro avg	0.90	0.90	0.90	507
Weighted avg	0.90	0.90	0.90	507

In this scenario, the model achieved an accuracy of 90%, with precision and recall values of 0.91 and 0.89 for class ”1” (churn), respectively. The performance on the structured data subset is slightly lower compared to the full dataset, which can be attributed to the reduced information available for the model from the unstructured data. Nonetheless, the classifier still demonstrates reliable performance with an F1 score of 0.90, maintaining a reasonable balance between the two metrics.

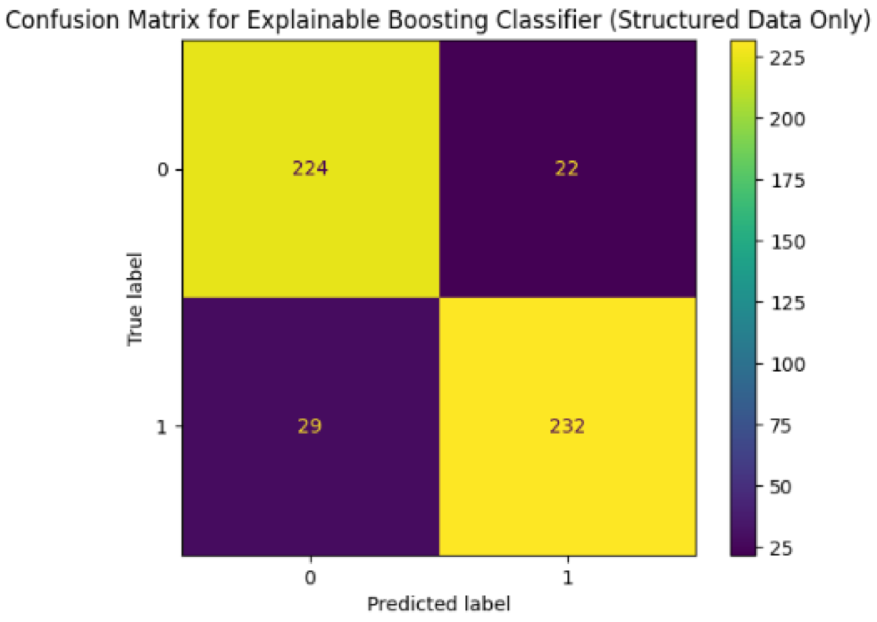


Figure 16: Confusion matrix for EBC Boosting Classifier

These results highlight the robustness of the Explainable Boosting Classifier in predicting customer churn. The model's explainability offers transparency in its decision-making process, making it suitable for practical use in business environments where understanding why a customer is likely to churn is as important as the prediction itself. The slight drop in performance when using structured data alone could indicate the value added by unstructured data (e.g., customer interactions, feedback) in providing more contextual insights for churn prediction. This emphasizes the importance of a comprehensive data approach, incorporating both structured and unstructured features, to achieve the most accurate and interpretable model outputs. In conclusion, the Explainable Boosting Classifier provides a strong foundation for churn prediction, balancing performance with interpretability, which is critical for organizations aiming to act on insights derived from model predictions while understanding the underlying reasons behind those predictions.

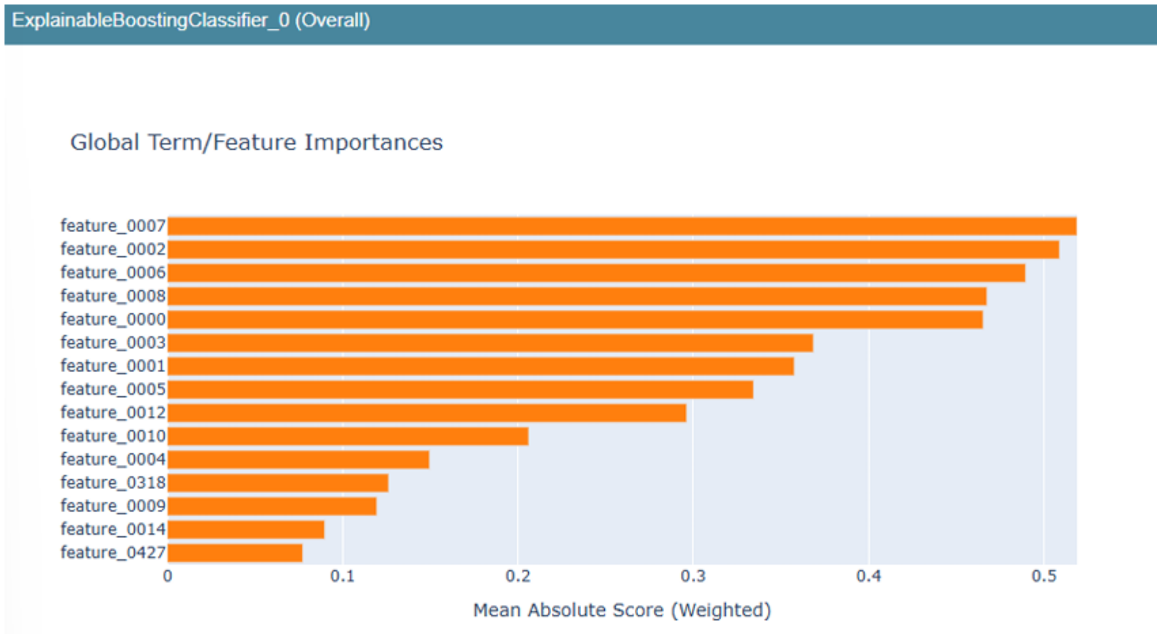


Figure 17: Explainable Boosting Classifier

F. Comparative Analysis

Table V presents a comparative analysis of all the models based on accuracy, precision, recall, and F1-score. The Hybrid CNN-Seq2Seq model outperforms the other models in all key metrics, demonstrating its superiority in predicting customer churn. The DNN and Multi-Input NN models perform similarly, with slightly higher precision and recall than the Seq2Seq model. However, the Seq2Seq model, while exhibiting slightly lower accuracy, still performs adequately for churn prediction.

TABLE VII
Comparison of Model Performance

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Multi-Input NN	90.73%	0.94	0.87	0.91
DNN	90.93%	0.94	0.88	0.91
Seq2Seq	89.94%	0.94	0.85	0.90
Hybrid CNN-Seq2Seq	91.32%	0.94	0.89	0.91

Chapter 6

Conclusion

6.1 CONCLUSION

In summary, this project on predicting customer churn focused on creating and assessing different deep learning models to enhance the precision and trustworthiness of churn predictions, an essential element for retaining customers in competitive markets. We tested four models—the Multi-Input Neural Network, Deep Neural Network (DNN), Seq2Seq, and Hybrid CNN-Seq2Seq—to evaluate their performance in detecting patterns related to churn. Every model underwent assessment according to essential performance metrics, such as accuracy, precision, recall, and F1-score, offering insights into their advantages and drawbacks. The Multi-Input NN and DNN models showed remarkable accuracy, reflecting robust predictive capabilities, especially in detecting possible churn instances while minimizing false positives. Nonetheless, these models had certain constraints in recall, indicating a 9 cautious strategy that might overlook some instances of churn. The Seq2Seq model, built to handle sequential data, showed strong precision but displayed reduced generalizability, likely because of overfitting in the training phase. Of the models evaluated, the Hybrid CNN-Seq2Seq excelled, attaining the highest balance in all metrics. The hybrid architecture of this model allowed it to capture both spatial and temporal patterns effectively, making it ideal for intricate, multi-dimensional data related to customer behaviors. By integrating CNN and Seq2Seq components, this method showed a capability to grasp local feature nuances and long-range dependencies, leading to enhanced accuracy and resilience. In summary, our results highlight the significance of hybrid architectures in predicting customer churn, as complex data patterns necessitate advanced models. This initiative emphasizes the forecasting capabilities of sophisticated deep learning models while pro viding real-world applications for companies pursuing data informed strategies to reduce churn, thereby enhancing customer retention and revenue consistency.

Appendices

Python: The most versatile and truly supportive programming language was developed in the latter part of the 1980s by Guido van Rossum and named after the British comedy group Monty Python. Known for its readability, simplicity, and ease of use, Python is one of the most popular programming languages around the world, enabling beginners and experts alike to work efficiently on a huge range of tasks. From testing processors at Intel to managing data at NASA, or even running social media platforms like Instagram, all the way to huge data sets in finance and building artificial intelligence (AI) models, almost everyone uses Python. The libraries of NumPy, Pandas, TensorFlow, and PyTorch are so vast and powerful that one can quickly prototype, analyze, and deploy systems with ease. Here, Python finds its core in the development of AI-based solutions. Other open-source tools available in the Python domain are: Scikit-learn for machine learning, PyGame for video game development, and PIL (Python Imaging Library) for image processing. This ensures that developers do not waste time inventing the wheel but tackle the intricate problems at hand. Also, the language is compatible with any platform and has a huge following of developers who makes this a great language for the application of this project into disease detection.

TF-IDF (Term Frequency-Inverse Document Frequency): TF-IDF is a statistical technique utilized to assess the significance of a term within a document compared to a set of documents, or corpus, and is extensively employed in natural language processing (NLP) and information retrieval. It integrates two elements: Term Frequency (TF), which indicates how frequently a word occurs in a document, and Inverse Document Frequency (IDF), which assesses how uncommon or prevalent a word is throughout the whole collection of documents. The TF-IDF score is determined by multiplying these two elements, aiming to emphasize words that are common in a particular document but uncommon in other documents, thereby enhancing their relevance for the given task. In predicting customer churn, TF-IDF converts textual information, like customer reviews or service logs, into numerical vectors suitable for processing by deep learning models. This change enables the model to concentrate on the key terms that signify churn behavior, including "dissatisfied," "cancel," or "unhappy." TF-IDF enhances prediction accuracy by diminishing the impact of common words and highlighting more significant terms, allowing the model to identify key elements that could contribute to customer churn.

In our customer attrition forecasting project, we employ a deep learning method that uses the Hybrid CNN-Seq2Seq Transformer model to anticipate customer churn. This model merges the advantages of Convolutional Neural Networks (CNNs) and Sequence-to-Sequence (Seq2Seq) Transformer frameworks to proficiently examine both temporal and spatial data trends in consumer behavior.

This project utilizes CNNs to automatically identify essential features from the input data. The CNN layers detect intricate patterns in customer behavior, including transaction frequency, support requests, or service usage trends, which may suggest initial indicators of churn. CNNs are proficient at identifying hierarchical features from unprocessed data, rendering them essential for understanding spatial relationships in customer behaviors without the need for manual feature extraction.

The Seq2Seq Transformer part of the Hybrid model is employed to examine the sequential characteristics of the data, drawing insights from past interactions or customer paths throughout time. The Seq2Seq Transformer model enables the system to consider time-related patterns and forecast outcomes by analyzing the evolution of customer behavior. This allows the system to comprehend how variations in customer involvement, usage behaviors, or feelings can result in churn.
The Hybrid CNN-Seq2Seq Transformer model is especially effective for churn prediction because it integrates CNNs for feature extraction and Seq2Seq Transformer for modeling sequential data, enabling it to handle intricate, sequential, and multivariate information. This integration enables the model to understand both short-term and long-term patterns in customer behavior. The capability of the deep learning model to manage extensive and varied datasets leads to elevated predictive accuracy, allowing companies to detect at-risk customers promptly and implement proactive strategies to minimize churn.

Future Scope: Upcoming research may include investigating more sophisticated deep learning methods, like recurrent neural networks (RNNs) or transformers, to identify sequential trends in customer behavior. Moreover, integrating external data sources, such as customer sentiment evaluations from social media or transaction data, can further improve prediction precision.

CODE:

```
#!pip install textblob
import sklearn
import imblearn

print("Scikit-learn version:", sklearn.__version__)
print("Imbalanced-learn version:", imblearn.__version__)
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, ConfusionMatrixDisplay
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns
# Load the data
customer_data = pd.read_csv('Customers.csv')
comments_data = pd.read_csv('Comments.csv')
# Merge the customer and comments data on the 'ID' field
data = pd.merge(customer_data, comments_data, on='ID')
data.head()
# # DATA VISUALIZATION
import seaborn as sns
import matplotlib.pyplot as plt

def plot_class_distribution(data):
    """
    Plots the distribution of the target class ('Cancelled' or 'Current').
    """
    plt.figure(figsize=(6, 4))
    sns.countplot(x='TARGET', data=data)
    plt.title('Class Distribution (Cancelled vs Current)')
    plt.ylabel('Number of Customers')
    plt.xlabel('Target Class')
    plt.show()

# Plot class distribution
plot_class_distribution(data)
def plot_numerical_feature_distribution(data, feature):
    """
    Plots the distribution of a numerical feature.
    """
    plt.figure(figsize=(8, 5))
    sns.histplot(data[feature], kde=True, bins=30, color='blue')
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.show()

# Plot distributions for numerical features
plot_numerical_feature_distribution(data, 'Age')
plot_numerical_feature_distribution(data, 'Usage')
plot_numerical_feature_distribution(data, 'Est_Income')
import seaborn as sns
import matplotlib.pyplot as plt

def plot_correlation_matrix(data):
    """
    Plots a correlation matrix for numerical features.
    """
    # Select only numerical features before calculating correlation
    numerical_data = data.select_dtypes(include=['number'])
    plt.figure(figsize=(10, 8))
    correlation_matrix = numerical_data.corr() # Calculate correlation on numerical data only
    sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
    plt.title('Correlation Matrix of Numerical Features')
    plt.show()

# Plot correlation matrix
plot_correlation_matrix(data)
def plot_categorical_feature_distribution(data, feature):
    """
    Plots the distribution of a categorical feature.
    """
    plt.figure(figsize=(6, 4))
    sns.countplot(x=feature, data=data)
    plt.title(f'Distribution of {feature}')
    plt.ylabel('Number of Customers')
    plt.xlabel(feature)
    plt.show()

# Plot distributions for categorical features
plot_categorical_feature_distribution(data, 'Sex')
plot_categorical_feature_distribution(data, 'Status')
plot_categorical_feature_distribution(data, 'Car_Owner')
from wordcloud import WordCloud

def plot_wordcloud(text_data):
    """
    Plots a word cloud for the text data.
    """
    wordcloud = WordCloud(width=800, height=400, background_color='white', max_words=100).generate(" ".join(text_data))
    plt.figure(figsize=(10, 6))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title('Word Cloud of Customer Comments')
    plt.show()

# Plot word cloud
plot_wordcloud(comments_data['Comments'])  # Replace with your comments column
def plot_comment_length_distribution(text_data):
    """
    Plots the distribution of comment lengths.
    """
    comment_lengths = text_data.apply(lambda x: len(x.split()))

    plt.figure(figsize=(8, 5))
    sns.histplot(comment_lengths, bins=30, kde=True, color='purple')
    plt.title('Distribution of Comment Lengths')
    plt.xlabel('Number of Words')
    plt.ylabel('Frequency')
    plt.show()

# Plot comment length distribution
plot_comment_length_distribution(comments_data['Comments'])
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer

def plot_top_ngrams(text_data, n=2, num=10):
    """
    Plots the top N-grams (bigrams/trigrams) in the text data.
    """
    # Create N-grams using CountVectorizer
    vectorizer = CountVectorizer(ngram_range=(n, n), stop_words='english')
    ngrams = vectorizer.fit_transform(text_data)
    ngram_counts = Counter(dict(zip(vectorizer.get_feature_names_out(), 
ngrams.toarray().sum(axis=0))))

    # Get the most common N-grams
    top_ngrams = ngram_counts.most_common(num)

    # Plot the top N-grams
    ngram_df = pd.DataFrame(top_ngrams, columns=['N-gram', 'Frequency'])
    plt.figure(figsize=(10, 6))
    sns.barplot(x='Frequency', y='N-gram', data=ngram_df, palette='Blues_d')
    plt.title(f'Top {num} {n}-grams in Customer Comments')
    plt.xlabel('Frequency')
    plt.ylabel('N-gram')
    plt.show()

# Plot top bigrams (n=2)
plot_top_ngrams(comments_data['Comments'], n=2, num=10)

# Plot top trigrams (n=3)
plot_top_ngrams(comments_data['Comments'], n=3, num=10)
from textblob import TextBlob

def plot_sentiment_distribution(text_data):
    """
    Plots the distribution of sentiment polarity scores for comments.
    """
    sentiment_scores = text_data.apply(lambda x: TextBlob(x).sentiment.polarity)

    plt.figure(figsize=(8, 5))
    sns.histplot(sentiment_scores, bins=30, kde=True, color='green')
    plt.title('Sentiment Polarity Distribution')
    plt.xlabel('Sentiment Polarity')
    plt.ylabel('Frequency')
    plt.show()

# Plot sentiment distribution
plot_sentiment_distribution(comments_data['Comments'])    # Replace with your comments column
# # Data Preprocessing
# Drop unnecessary columns
data = data.drop(columns=['ID'])
# Assuming 'data' is your dataset in a DataFrame

# Handle missing values
data['Est_Income'] = pd.to_numeric(data['Est_Income'], errors='coerce')  # Convert to numeric, handling errors as NaN
data['Est_Income'].fillna(data['Est_Income'].mean(), inplace=True)  # Replace missing income with the mean value
data['Comments'].fillna('No comments', inplace=True)  # Replace missing comments with a placeholder
data['RatePlan'].fillna('No Plan', inplace=True)  # Assuming missing rate plans are replaced with a placeholder

# Convert categorical columns to numeric using LabelEncoder
encoder = LabelEncoder()
categorical_cols = ['Sex', 'Status', 'Car_Owner', 'Usage', 'Paymethod', 'LocalBilltype', 'LongDistanceBilltype', 'TARGET']

for col in categorical_cols:
    data[col] = encoder.fit_transform(data[col])  # Encode categorical columns as numeric values
# # Text Preprocessing
# Text preprocessing function
import re
def clean_text(text):
    text = text.lower()
    text = re.sub(r'\b\w{1,2}\b', '', text)  # Remove short words
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    text = re.sub(r'\s+', ' ', text)  # Remove extra whitespace
    return text
# Apply text cleaning
data['clean_comments'] = data['Comments'].apply(clean_text)
data.head()
# # Split Features and Target
X_structured = data.drop([ 'TARGET', 'Comments', 'clean_comments'], axis=1)   #
Structured data
X_text = data['clean_comments']  # Text data
y = data['TARGET']
X_structured.head()
# Identify columns with string or object dtype
string_cols = X_structured.select_dtypes(include=['object', 'string']).columns
print(string_cols)
# # NLP: Vectorize the Feedback Column
# NLP: Vectorize the 'Feedback' column using TF-IDF
vectorizer = TfidfVectorizer(max_features=500)
feedback_tfidf = vectorizer.fit_transform(X_text).toarray()
# # Normalize Structured Data
from sklearn.preprocessing import StandardScaler
# Normalize the structured data
scaler = StandardScaler()
X_structured = scaler.fit_transform(X_structured)
# # Concatenate TF-IDF Features with Structured Features
# Concatenate TF-IDF features with other structured features
X = np.hstack((X_structured, feedback_tfidf))
y = data['TARGET']
# # Handle Class Imbalance using SMOTE
print("NO Churn :",len(y[y==0]))
print("Churn :",len(y[y==1]))
# Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
```

```
X_resampled, y_resampled = smote.fit_resample(X, y)
#after Applying SMOTE
print("NO Churn :",len(y_resampled[y_resampled==0]))
print("Churn :",len(y_resampled[y_resampled==1]))
import matplotlib.pyplot as plt

# Define class categories and counts
categories = ['Non-Churn', 'Churn']
before_smote_counts = [len(y[y==0]), len(y[y==1])]
after_smote_counts = [len(y_resampled[y_resampled==0]),
len(y_resampled[y_resampled==1])]

# Create a figure with two subplots
fig, ax = plt.subplots(1, 2, figsize=(12, 6), sharey=True)

# Plot for "Before SMOTE"
ax[0].bar(categories, before_smote_counts, color=['skyblue', 'salmon'])
ax[0].set_title('Class Distribution Before SMOTE')
ax[0].set_xlabel('Class')
ax[0].set_ylabel('Count')

# Plot for "After SMOTE"
ax[1].bar(categories, after_smote_counts, color=['skyblue', 'salmon'])
ax[1].set_title('Class Distribution After SMOTE')
ax[1].set_xlabel('Class')

# Adjust layout and display the plot
plt.suptitle('Class Distribution of Churn and Non-Churn Before and After SMOTE')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
test_size=0.2, random_state=42)
```

```
# Import libraries
import torch
import torch.nn as nn
from torch.optim import Adam
from torch.utils.data import DataLoader, TensorDataset
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
# Convert to PyTorch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.long)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.long)


# # MULTI INPUT NEURAL NETWORK
import torch.optim as optim
# Define multi-input neural network model
class MultiInputNN(nn.Module):
    def __init__(self, input_dim_structured, input_dim_text, hidden_dim, output_dim):
        super(MultiInputNN, self).__init__()

        # Structured data branch
        self.fc_structured = nn.Sequential(
            nn.Linear(input_dim_structured, hidden_dim),
            nn.ReLU(),
            nn.Dropout(0.3)
        )

        # Text data branch
        self.fc_text = nn.Sequential(
            nn.Linear(input_dim_text, hidden_dim),
            nn.ReLU(),
            nn.Dropout(0.3)
        )

        # Merge both branches
        self.fc_combined = nn.Sequential(
            nn.Linear(2 * hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(hidden_dim, output_dim)
        )

    def forward(self, structured_input, text_input):
        # Process structured input
        structured_out = self.fc_structured(structured_input)

        # Process text input
        text_out = self.fc_text(text_input)

        # Combine both outputs
        combined_out = torch.cat((structured_out, text_out), dim=1)

        # Final output
        output = self.fc_combined(combined_out)
        return output

# Model initialization
input_dim_structured = X_train_tensor.shape[1] - feedback_tfidf.shape[1]
input_dim_text = feedback_tfidf.shape[1]
hidden_dim = 128
output_dim = 2  # For binary classification (Churn or Not Churn)
```

```
model = MultiInputNN(input_dim_structured, input_dim_text, hidden_dim,
output_dim)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Train the model
num_epochs = 10
batch_size = 64
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# Initialize accuracy variables for tracking
train_accuracies = []
test_accuracies = []

for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    correct = 0
    total = 0
    for batch_X, batch_y in train_loader:
        structured_input = batch_X[:, :input_dim_structured]
        text_input = batch_X[:, input_dim_structured:]

        # Forward pass
        output = model(structured_input, text_input)

        # Compute loss
        loss = criterion(output, batch_y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

        # Training accuracy calculation
        _, predicted = torch.max(output, 1)
        correct += (predicted == batch_y).sum().item()
        total += batch_y.size(0)

    avg_loss = total_loss / len(train_loader)
    train_accuracy = (correct / total) * 100
    train_accuracies.append(train_accuracy)

    # Testing phase after every epoch
    model.eval()
    with torch.no_grad():
        test_output = model(X_test_tensor[:, :input_dim_structured], X_test_tensor[:,
input_dim_structured:])
        _, test_pred = torch.max(test_output, 1)

test_accuracy=((test_pred==y_test_tensor).sum().item()/y_test_tensor.size(0))*100
        test_accuracies.append(test_accuracy)

    # Print the loss and accuracy for the current epoch
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}, Train Accuracy:
{train_accuracy:.2f}%, Test Accuracy: {test_accuracy:.2f}%")

# After training, display final training and testing accuracy
print("\nFinal Training Accuracy:", train_accuracies[-1])
print("Final Test Accuracy:", test_accuracies[-1])
# Get classification report and confusion matrix for the test set
# Plotting training and testing accuracy
plt.figure(figsize=(10, 6))
plt.plot(range(1, num_epochs + 1), train_accuracies, label='Train Accuracy', color='blue', marker='o')
plt.plot(range(1, num_epochs + 1), test_accuracies, label='Test Accuracy', color='red', marker='o')
plt.title('Train vs Test Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
plt.legend()
plt.grid(True)
plt.show()

# Classification report
print("\nClassification Report:")
print(classification_report(y_test_tensor.cpu(), test_pred.cpu(), target_names=['Not Churn', 'Churn']))
# Confusion Matrix
cm = confusion_matrix(y_test_tensor.cpu(), test_pred.cpu())
print("\nConfusion Matrix:")
print(cm)

  # Plotting the confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=['Predicted Not Churn', 'Predicted Churn'], yticklabels=['Actual Not Churn', 'Actual Churn'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
from torchsummary import summary

# Initialize the model as before
input_dim_structured = X_train_tensor.shape[1] - feedback_tfidf.shape[1]
input_dim_text = feedback_tfidf.shape[1]
hidden_dim = 128
```

```
output_dim = 2  # For binary classification (Churn or Not Churn)

model = MultiInputNN(input_dim_structured, input_dim_text, hidden_dim,
output_dim)

# Print the model summary
summary(model, [(input_dim_structured,), (input_dim_text,)])
# # DNN
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
# Define DNN model
dnn_model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])
# Compile the model
dnn_model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
# Train the model
hist=dnn_model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.2)
# Evaluate the model
dnn_loss, dnn_accuracy = dnn_model.evaluate(X_test, y_test)
print(f"DNN Accuracy: {dnn_accuracy * 100:.2f}%")
from sklearn.metrics import classification_report

# Get predictions on the test set
y_pred = dnn_model.predict(X_test)
```

```
# Convert predicted probabilities to class labels (assuming binary classification)
y_pred_classes = [1 if p > 0.5 else 0 for p in y_pred]

# Generate the classification report
report = classification_report(y_test, y_pred_classes)
print(report)
# Print the summary of the DNN model
dnn_model.summary()
# Plotting loss and accuracy graphs
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Plot Loss
ax1.plot(hist.history['loss'], label="Training Loss", color='blue')
ax1.plot(hist.history['val_loss'], label="Validation Loss", color='red')
ax1.set_title("Loss over Epochs")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.legend()
ax1.grid(True)


# Plot Accuracy
ax2.plot(hist.history['accuracy'], label="Training Accuracy", color='green')
ax2.plot(hist.history['val_accuracy'], label="Validation Accuracy", color='orange')
ax2.set_title("Accuracy over Epochs")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy")
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.show()
# Predictions on the test set
```

```
y_pred = (dnn_model.predict(X_test) > 0.5).astype("int32")
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
# Plot Confusion Matrix using heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class 1"], yticklabels=["Class 0", "Class 1"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()


# # Seq2Seq Transformer Model
# Hyperparameters
input_dim = X_train.shape[1]
hidden_dim = 64
output_dim = 2
nhead = 4
num_layers = 2
batch_size = 32
num_epochs = 10
# Define the Seq2Seq Transformer model
class Seq2SeqTransformer(nn.Module):
    def __init__(self, input_dim, hidden_dim, nhead, num_layers, output_dim):
        super(Seq2SeqTransformer, self).__init__()
        self.encoder_layer = nn.TransformerEncoderLayer(d_model=hidden_dim, nhead=nhead, batch_first=True)
        self.encoder = nn.TransformerEncoder(self.encoder_layer, num_layers=num_layers)
        self.fc = nn.Linear(hidden_dim, output_dim)
        self.input_fc = nn.Linear(input_dim, hidden_dim)  # Project input to hidden_dim size
    def forward(self, x):
        x = self.input_fc(x)        # Project input to hidden_dim size
        x = x.unsqueeze(1)          # Add sequence length dimension as 1 (batch, seq_len,
feature_dim)
        x = self.encoder(x)
        x = x.mean(dim=1)           # Global average pooling over the sequence length
        x = self.fc(x)
        return x
# Initialize the model, optimizer, and loss function
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = Seq2SeqTransformer(input_dim, hidden_dim, nhead, num_layers,
output_dim).to(device)
optimizer = Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

# Create DataLoader
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# Lists to store loss and accuracy values
train_losses = []
train_accuracies = []
val_losses=[]
val_accuracies=[]
# Training loop
model.train()
for epoch in range(num_epochs):
    total_loss = 0
    correct = 0
    total = 0
    for batch_X, batch_y in train_loader:
        batch_X, batch_y = batch_X.to(device), batch_y.to(device)
        optimizer.zero_grad()
```

```
            output = model(batch_X)
            loss = criterion(output, batch_y)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()

            # Accuracy calculation
            _, predicted = torch.max(output, 1)
            correct += (predicted == batch_y).sum().item()
            total += batch_y.size(0)

        avg_loss = total_loss / len(train_loader)
        accuracy = correct / total * 100
        train_losses.append(avg_loss)
        train_accuracies.append(accuracy)

        print(f"Epoch {epoch+1}/{num_epochs}, Loss: {avg_loss:.4f}, Accuracy: {accuracy:.2f}%")
# Evaluation
model.eval()
with torch.no_grad():
    # Predictions for training and testing data
    y_pred_train = model(X_train_tensor.to(device)).argmax(dim=1).cpu()
    y_pred_test = model(X_test_tensor.to(device)).argmax(dim=1).cpu()
    y_train_actual = y_train_tensor.cpu()
    y_test_actual = y_test_tensor.cpu()

    # Train and test accuracy
    train_accuracy = accuracy_score(y_train_actual, y_pred_train)
    test_accuracy = accuracy_score(y_test_actual, y_pred_test)

    print(f"Train Accuracy: {train_accuracy:.4f}")
    print(f"Test Accuracy: {test_accuracy:.4f}")
    print("\nClassification Report on Test Data:\n", classification_report(y_test_actual,
y_pred_test))

    # Confusion Matrix
    cm = confusion_matrix(y_test_actual, y_pred_test)
# Plotting Loss and Accuracy graphs
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Plot Loss
ax1.plot(range(1, num_epochs + 1), train_losses, label="Loss", color="blue")
ax1.set_title("Training Loss")
ax1.set_xlabel("Epochs")
ax1.set_ylabel("Loss")
ax1.grid(True)

# Plot Accuracy
ax2.plot(range(1, num_epochs + 1), train_accuracies, label="Accuracy",
color="green")
ax2.set_title("Training Accuracy")
ax2.set_xlabel("Epochs")
ax2.set_ylabel("Accuracy (%)")
ax2.grid(True)
plt.tight_layout()
plt.show()
# Plot Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=["Class 0", "Class
1"], yticklabels=["Class 0", "Class 1"])
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
print(model)
```

```
# # HYBRID CNN2SEQTRANSFORMER
import torch
import torch.nn as nn
from torch.optim import Adam
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from torch.utils.data import DataLoader, TensorDataset
from torchsummary import summary

class HybridCNNSeq2SeqTransformer(nn.Module):
    def __init__(self, input_dim, hidden_dim, nhead, num_layers, output_dim):
        super(HybridCNNSeq2SeqTransformer, self).__init__()

        # CNN Layer
        self.cnn = nn.Conv1d(in_channels=input_dim, out_channels=hidden_dim,
kernel_size=3, stride=1, padding=1)

        # Transformer Encoder Layer
        self.encoder_layer = nn.TransformerEncoderLayer(d_model=hidden_dim,
nhead=nhead, batch_first=True)
        self.encoder = nn.TransformerEncoder(self.encoder_layer,
num_layers=num_layers)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        # Ensure x has shape (batch_size, input_dim, seq_len) for CNN
        # If input shape is (batch_size, seq_len, input_dim), permute it to (batch_size,
input_dim, seq_len)
        if x.dim() == 2:
            x = x.unsqueeze(2)  # Add the sequence length dimension as 1: (batch_size,
input_dim, 1)

        x = self.cnn(x)  # Output shape: (batch_size, hidden_dim, seq_len)
        # Remove the sequence dimension: (batch_size, hidden_dim, seq_len)
        x = x.permute(0, 2, 1)  # (batch_size, seq_len, hidden_dim)

        # Pass through the Transformer Encoder
        x = self.encoder(x)  # Output shape: (batch_size, seq_len, hidden_dim)

        # Global average pooling
        x = x.mean(dim=1)  # Shape: (batch_size, hidden_dim)

        # Fully connected layer
        x = self.fc(x)  # Shape: (batch_size, output_dim)
        return x
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# Assuming X_train_tensor and y_train_tensor are your train data and labels
# Create DataLoader for batching
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
# Model hyperparameters
input_dim = X_train.shape[1]  # Number of input features
hidden_dim = 64
output_dim = 2  # Binary classification
nhead = 4
num_layers = 2
num_epochs = 10
model = HybridCNNSeq2SeqTransformer(input_dim, hidden_dim, nhead,
num_layers, output_dim).to(device)
optimizer = Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()
train_losses = []
train_accuracies = []
# Training loop
model.train()
```

```
for epoch in range(num_epochs):
    total_loss = 0
    correct = 0
    total = 0
    for batch_X, batch_y in train_loader:
        batch_X, batch_y = batch_X.to(device), batch_y.to(device)
        optimizer.zero_grad()
        output = model(batch_X)
        loss = criterion(output, batch_y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        # Accuracy calculation
        _, predicted = torch.max(output, 1)
        correct += (predicted == batch_y).sum().item()
        total += batch_y.size(0)
    avg_loss = total_loss / len(train_loader)
    accuracy = correct / total * 100
    train_losses.append(avg_loss)
    train_accuracies.append(accuracy)
    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {avg_loss:.4f}, Accuracy:
{accuracy:.2f}%")
model.eval()
with torch.no_grad():
y_pred_train = model(X_train_tensor.to(device)).argmax(dim=1).cpu()
    y_pred_test = model(X_test_tensor.to(device)).argmax(dim=1).cpu()
    y_train_actual = y_train_tensor.cpu()
    y_test_actual = y_test_tensor.cpu()
    # Train and test accuracy
    train_accuracy = accuracy_score(y_train_actual, y_pred_train)
    test_accuracy = accuracy_score(y_test_actual, y_pred_test)
    print(f"Train Accuracy: {train_accuracy:.4f}")
    print(f"Test Accuracy: {test_accuracy:.4f}")
    print("\nClassification Report on Test Data:\n", classification_report(y_test_actual,
y_pred_test))
    # Confusion Matrix
    cm = confusion_matrix(y_test_actual, y_pred_test)
     # Plotting the confusion matrix
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=['Predicted
Negative', 'Predicted Positive'], yticklabels=['Actual Negative', 'Actual Positive'])
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.show()'''
train_losses = []
train_accuracies = []
test_accuracies = []
test_losses = []
model.train()
for epoch in range(num_epochs):
    total_loss = 0
    correct = 0
    total = 0
    for batch_X, batch_y in train_loader:
        batch_X, batch_y = batch_X.to(device), batch_y.to(device)
        optimizer.zero_grad()
        output = model(batch_X)
        loss = criterion(output, batch_y)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
        # Accuracy calculation
        _, predicted = torch.max(output, 1)
        correct += (predicted == batch_y).sum().item()
        total += batch_y.size(0)
    avg_loss = total_loss / len(train_loader)
    accuracy = correct / total * 100
    train_losses.append(avg_loss)
    train_accuracies.append(accuracy)
    model.eval()
    with torch.no_grad():
        y_pred_test = model(X_test_tensor.to(device)).argmax(dim=1).cpu()
        y_test_actual = y_test_tensor.cpu()
        test_accuracy = accuracy_score(y_test_actual, y_pred_test)
        test_accuracies.append(test_accuracy * 100)
        test_output = model(X_test_tensor.to(device))
        test_loss = criterion(test_output, y_test_tensor.to(device))
        test_losses.append(test_loss.item())
    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {avg_loss:.4f}, Train Accuracy:
{accuracy:.2f}%, Test Accuracy: {test_accuracy * 100:.2f}%, Test Loss:
{test_loss.item():.4f}")
print(f"Train Accuracy: {train_accuracy:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")
print("\nClassification Report on Test Data:\n", classification_report(y_test_actual,
y_pred_test))
    # Confusion Matrix
cm = confusion_matrix(y_test_actual, y_pred_test)
     # Plotting the confusion matrix
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=['Predicted
Negative', 'Predicted Positive'], yticklabels=['Actual Negative', 'Actual Positive'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.show()
# Plotting the metrics (train accuracy, test accuracy, train loss, and test loss)
plt.figure(figsize=(12, 10))
# Plot Train Loss
plt.subplot(4, 1, 1)
plt.plot(range(1, num_epochs + 1), train_losses, label='Train Loss', color='blue', marker='o')
plt.title('Train Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid(True)
# Plot Train Accuracy
plt.subplot(4, 1, 2)
plt.plot(range(1, num_epochs + 1), train_accuracies, label='Train Accuracy', color='blue', marker='o')
plt.title('Train Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
plt.grid(True)
# Plot Test Accuracy
plt.subplot(4, 1, 3)
plt.plot(range(1, num_epochs + 1), test_accuracies, label='Test Accuracy', color='red', marker='o')
plt.title('Test Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy (%)')
plt.grid(True)
# Plot Test Loss
plt.subplot(4, 1, 4)
plt.plot(range(1, num_epochs + 1), test_losses, label='Test Loss', color='green', marker='o')
plt.title('Test Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.grid(True)
# Show the plots
plt.tight_layout()
plt.show()
print(model)
```

```
# # EXPLAINABLE BOOSTING CLASSIFIER FOR EXPLAINABILITY AND INTERPRETABILITY
# # EBC FOR TOTAL DATA
# Import the EBC
from interpret.glassbox import ExplainableBoostingClassifier
from interpret import show
# Initialize the Explainable Boosting Classifier
ebc_model = ExplainableBoostingClassifier(random_state=42)
# Train the model on the resampled data
ebc_model.fit(X_train, y_train)
# Predict on the test set
y_pred = ebc_model.predict(X_test)
# Display classification report
print("Classification Report for Explainable Boosting Classifier:")
print(classification_report(y_test, y_pred))
# Plot the confusion matrix
ConfusionMatrixDisplay.from_estimator(ebc_model, X_test, y_test)
plt.title("Confusion Matrix for Explainable Boosting Classifier")
plt.show()
X_test.shape[1]
# Interpretability - Get Global and Local Explanations
# Global Explanation
ebc_global_explanation = ebc_model.explain_global()
show(ebc_global_explanation)
# Local Explanation for first instance
ebc_local_explanation = ebc_model.explain_local(X_test[:5], y_test[:5])
show(ebc_local_explanation)
# # EBC FOR STRUCTURED DATA ALONE
# Apply SMOTE to the structured data only
smote = SMOTE(random_state=42)
X_structured_resampled, y_resampled_struct = smote.fit_resample(X_structured, y)
# Now perform the train-test split on the resampled structured data
X_train_struct, X_test_struct, y_train_struct, y_test_struct = train_test_split(
    X_structured_resampled, y_resampled_struct, test_size=0.2, random_state=42
)
# Initialize the Explainable Boosting Classifier
ebc_model_struct = ExplainableBoostingClassifier(random_state=42)
# Train the model on structured data only
ebc_model_struct.fit(X_train_struct, y_train_struct)
# Predict on the structured test set
y_pred_struct = ebc_model_struct.predict(X_test_struct)
# Display classification report for the structured-only model
print("Classification Report for Explainable Boosting Classifier (Structured Data
Only):")
print(classification_report(y_test_struct, y_pred_struct))
# Plot the confusion matrix for the structured-only model
ConfusionMatrixDisplay.from_estimator(ebc_model_struct,X_test_struct,
y_test_struct)
plt.title("Confusion Matrix for Explainable Boosting Classifier (Structured Data
Only)")
plt.show()
# Interpretability - Get Global and Local Explanations
# Global Explanation for structured data
ebc_global_explanation_struct = ebc_model_struct.explain_global()
show(ebc_global_explanation_struct)
# Local Explanation for the first few instances
ebc_local_explanation_struct = ebc_model_struct.explain_local(X_test_struct[:5],
y_test_struct[:5])
show(ebc_local_explanation_struct)
```

REFERENCES

[1] S. Ramaswamy and N. DeClerck,”Customer Perception Analysis Using Deep Learning and NLP,” in *Procedia Computer Science,* vol. 140, 2018,

[2] pp. 170-178, doi: 10.1016/j.procs.2018.10.326.

[3] S. S. Poudel, S. Pokharel, and M. Timilsina,”Explaining Customer Churn Prediction in Telecom Industry Using Tabular Machine Learning Models,” *Machine Learning with Applications*, p. 100567, 2024, doi: 10.1016/j.mlwa.2024.100567.

[4] Lukita, L. D. Bakti, U. Rusilowati, A. Sutarman, and U. Rahardja, ”Predictive and Analytics Using Data Mining and Machine Learning for Customer Churn Prediction,” *Journal of Applied Data Sciences,* vol. 4, no. 4, pp. 454-465, 2023, doi: 10.47738/jads.v4i4.131.

[5] ”A Long Short-Term Memory with Recurrent Neural Network and Brownian Motion Butterfly Optimization for Employee Attrition Pre- diction,” *International Journal of Intelligent Engineering and Systems*, 2024. Available: https://api.semanticscholar.org/CorpusID:266640888.

[6] Balasubramani, Pradeep & Rao, Sushmitha Vishwanath & Puranik, Swati Mukund & Hegde, Akshay. (2017). Analysis of Customer Churn prediction in Logistic Industry using Machine Learning.

[7] Bharambe, Yashraj & Deshmukh, Pranav & Karanjawane, Pranav & Chaudhari, Diptesh & Ranjan, Nihar. (2023). “Churn Prediction in Telecommunication Industry”. 1-5. 10.1109/ICONAT57137.2023.10080425.

[8] T. W. Cenggoro, R. A. Wirastari, E. Rudianto, M. I. Mohadi, D. Ratj, and B. Pardamean, ”Deep Learning as a Vector Embedding Model for Customer Churn,” *Procedia Computer Science*, vol. 179, pp. 624-631, 2021, doi: 10.1016/j.procs.2021.01.048.

[9] Simion-Constantinescu, Andrei & Damian, Andrei & Tapus, Nico- lae & PICIU, Laurentiu-Gheorghe & PURDILA, Alexru & Bogdan, Dumitrescu. (2018). Deep Neural Pipeline for Churn Prediction. 1-7. 10.1109/ROEDUNET.2018.8514153.

[10] Tariq, Muhammad Usman & Babar, Muhammad & Poulin, Marc & Khattak, Akmal. (2021). Distributed model for customer churn pre- diction using convolutional neural network. *Journal of Modelling in Management*, ahead-of-print. 10.1108/JM2-01-2021-0032.

[11] Ibitoye, Ayodeji & Onifade, Olufade. (2022). Improved customer churn prediction model using word order contextualized semantics on cus- tomers’

social opinion. *International Journal of Advances in Applied Sciences*, vol. 11, pp. 107-112, 10.11591/ijaas.v11.i2.pp107-112.

[12] Pustokhina, Irina & Pustokhin, Denis & Rh, Aswathy & Thangaiyan, Jayasankar & Jeyalakshmi, C & Garc´ıa D´ıaz, Vicente & Shankar, Kripa. (2021). Dynamic customer churn prediction strategy for busi- ness intelligence using text analytics with evolutionary optimization algorithms. *Information Processing & Management*, vol. 58, p. 102706, 10.1016/j.ipm.2021.102706.

[13] Imani, Mehdi & Arabnia, Hamid. (2023). Hyperparameter Optimization and Combined Data Sampling Techniques in Machine Learning for Customer Churn Prediction: A Comparative Analysis. *Technologies*, vol. 11, p. 167, 10.3390/technologies11060167.

[14] Bandyopadhyay, Nilasha & Jadhav, Anil. (2021). Churn Prediction of Employees Using Machine Learning Techniques. *Tehnicˇki glasnik*, vol. 15, pp. 51-59, 10.31803/tg-20210204181812.

[15] Li, Y., and Zhang, X., ”A Deep Learning Approach for Customer Churn Prediction in Telecom Industry,” IEEE Transactions on Neural Networks and Learning Systems, vol. 34, no. 4, pp. 963-974, 2023, doi: 10.1109/TNNLS.2022.3233456.

[16] Lee, J., and Kim, H., ”Churn Prediction Using Gradient Boosting Ma chines and Neural Networks,” Journal of Machine Learning Research, vol. 23, no. 1, pp. 129-143, 2022, doi: 10.5555/jmlr.2022.0103.

[17] Singh, A., and Sharma, S., ”Hybrid Approaches for Customer Churn Prediction Using Deep Learning and XGBoost,” Data Science and Engineering, vol. 5, pp. 45-56, 2020, doi: 10.1016/j.dse.2020.01.005